Gestor de exposiciones caninas en Ruby on Rails

Proyecto de Fin de Carrera

Autor: Raúl José Alcañiz de la Fuente
Tutor: D. Carmelo Cuenca Hernández
Titulación: Ingeniería en Informática
Abril de 2015

Proyecto de fin de carrera de la Escuela de Ingeniería en Informática de la Universidad de Las Palmas de Gran Canaria realizado por el alumno:

Raúl José Alcañiz de la Fuente

Título del proyecto:

Gestor de exposiciones caninas en Ruby on Rails

Tutores:

D. Carmelo Cuenca Hernández

A mis padres, hermano, novia y amigos

Agradecimientos

Me gustaría empezar los agradecimientos nombrando a mi tutor D. Carmelo Cuenca Hernández, porque sin él este proyecto nunca hubiera salido. Desde el primer momento ha confiado en mí y me ha ayudado en infinidad de aspectos relacionados con la elaboración del proyecto. Ha estado disponible en todo momento para cualquier consulta y siempre me ha aconsejado para seguir las pautas adecuadas que me llevasen a realizar el proyecto de la forma más correcta posible. No creo que hubiera podido conseguir un tutor con más dedicación que la que Carmelo me ha mostrado. Así que, muchísimas gracias Carmelo, ¡va por ti!

Gracias a mis padres, por hacerme como soy. Loli y José Ángel, las dos personas que durante toda mi vida han estado a mi lado y que siempre me han tendido una mano cuando lo he necesitado. Gracias a ellos tengo lo valores que tengo, y la ayuda que siempre me han dado ha sido de matrícula de honor. El amor que han mostrado por mí y el apoyo que me han proporcionado en todos los aspectos a lo largo de toda mi vida nunca podré agradecérselo del todo. Empezaré por el principio: Muchas gracias papá y mamá.

Agradezco el haber empezado esta carrera, porque gracias a ella tengo a una de las personas actualmente más importantes en mi vida, mi novia Saray. Muchísimas gracias Saray por apoyarme todos estos años en mis estudios, y por todas las vivencias que hemos tenido a lo largo de nuestro tiempo juntos. Te conocí en esta misma carrera, y la verdad es que te has convertido en un pilar de apoyo fundamental en mi vida. Por ello, no puedo tener otras palabras que no sean de agradecimiento hacia ti. Espero, a partir de ahora, lograr muchas más metas y proyectos contigo. Gracias de corazón, a ti, y al apoyo de toda tu familia.

Gracias a mi hermano David y a mi cuñada Bea, dos personas coraje que demuestran día a día que siempre puede haber algo por lo que sonreír. A pesar de tener su propia vida y luchar por ella, ánimos y apoyos nunca me han faltado de su parte. Os agradezco que siempre tuvierais buenas palabras para mis estudios, y que los valoraseis de una forma tan especial. Muchas gracias a los dos y mucha suerte en vuestra nueva etapa de “bipadres”.

A mis amigos más cercanos Pedro, Borja y Rubén, os agradezco mucho vuestra amistad, y que, a pesar de no tener toda la relación personal que me gustaría tener, os doy las gracias por seguir ahí, para lo bueno, y para lo malo. Dan, Vero, Jared, Cristo, Jorge, Mario, Alberto… Muchas gracias también a vosotros por todos los momentos buenos que hemos vivido.

Durante toda la carrera he conocido a mucha gente que merecería una mención especial de agradecimiento: Porti, David, José Luis, Marcos, Ana, Eliel… Tantas personas que sería imposible nombrarlas a todas, pero gracias para todos esos compañeros que saben que he pasado muy buenos momentos con ellos durante la carrera. Gracias también a esos buenos profesores que he tenido.

Muchas gracias a todas estas personas, y también a aquella familia y amigos que no nombré particularmente, pero que sé (y ellos saben) que me han ayudado todos estos años.

Raúl

Índice de contenidos

1- GESTOR DE EXPOSICIONES CANINAS EN ROR 1

2- OBJETIVOS 3

3- ESTRUCTURA DEL DOCUMENTO 7

4- ESTADO DEL ARTE 9

4.1- PLATAFORMAS DE INFORMACIÓN ESPECIALIZADA 9
4.1.1- Federación Cinológica Internacional (FCI) 9
4.1.2- Ingrus 11
4.2- EXHIBICIONES MUNDIALMENTE RECONOCIDAS 12
4.2.1- The Westminster Kennel Club 13
4.2.2- Crufts 14
4.3- PLATAFORMAS DE GESTIÓN DE EXHIBICIONES CANINAS 16
4.3.1- LANCA 16
4.3.2- Doglle 17

5- ESTUDIO DE HERRAMIENTAS 21

5.1- HERRAMIENTAS DE DESARROLLO 21
5.1.0- Drupal – Joomla! – WordPress 22
5.1.1- Joomla! 24
5.1.2- Ruby on Rails 25
5.1.3- Django 26
5.1.4- CakePHP 27
5.2- PLATAFORMAS DE CONTROL DE LA NUBE 28
5.2.1- Heroku 29
5.2.2- Windos Azure 30
5.2.3- OpenShift 31
5.3- HERRAMIENTAS DE AUTOMATIZACIÓN DE PRUEBA DEL SOFTWARE 33
5.3.1- Cucumber 34
5.3.2- RSpec 35
5.3.3- MiniTest 36
5.3.4- Concordion 36
5.3.5- Jasmine 37
5.4- ELECCIÓN DE HERRAMIENTAS 38
5.4.1- Elección de herramienta de desarrollo 38
5.4.2- Elección de plataforma de control de la nube 39
5.4.3- Elección de herramientas de pruebas 39

6- METODOLOGÍA DE DESARROLLO 41

6.1- PROGRAMACIÓN EXTREMA (XP – EXTREME PROGRAMMING) 41
6.2- METODOLOGÍA TDD 43
6.3- METODOLOGÍA BDD 46
6.4- DESARROLLO ATDD 48

7- RECURSOS UTILIZADOS 51

7.1- RECURSOS HARDWARE 51
7.2- RECURSOS SOFTWARE 51

7.2.1- Sistemas operativos 51
7.2.1.1- CentOS 51
7.2.1.2- Windows Vista Home Premium 52
7.2.2- Navegador Web 52
7.2.3- Entorno de Desarrollo Integrado 52
7.2.4- Sistema de Control de Versiones 52
7.2.5- Editores de texto 53
7.2.5.1- Microsoft Office Word 2007 53
7.2.5.2- Notepad++ 53
7.2.6- Editor de bocetos - Mockups 53
7.2.7- StarUML 53
7.2.8- Microsoft office Visio 2007 54
7.2.9- MySQL Workbench 54
7.3- TECNOLOGÍAS UTILIZADAS 54
7.3.1- Ruby / Ruby on Rails 54
7.3.2- Ruby Version Mannager 55
7.3.3- Active Record 55
7.3.4- HTML5 55
7.3.5- JavaScript 56
7.3.6- JQuery 56
7.3.7- AJAX 56
7.3.8- CSS3 57
7.3.9- Bootstrap 57
7.3.10- Cucumber 58
7.3.11- RSpec 58
7.3.12- Heroku 58
7.3.13- SQlite3 / PostgreSQL 58
7.3.14- JSON 59
7.3.15- Gemas 59

8- ANÁLISIS 61

8.1- SITIO WEB. 61
8.1.1- Estructura general 61
8.1.2- Contenidos 62
8.2- EL PORTAL 64
8.2.1- Especificación de requisitos del usuario 64
8.2.1.1- Descripción 64
8.2.1.2- Glosario de conceptos 65
8.2.1.3- Modelo del dominio 67
8.2.2- Especificación de requisitos de software 67
8.2.2.1- Descripción 68
8.2.2.2- Glosario de conceptos 69
8.2.2.3- Modelo del diseño 70
8.2.2.4- Actores del sistema 71
8.2.2.5- Listado de actores y roles 71
8.2.2.6- Listado de actores y objetivos 72
8.2.2.7- Diagramas UML de casos de uso 73
8.2.2.8- Listado de casos de uso 76
8.2.2.9- Diagramas de secuencia 77
8.3- PRUEBAS DEL SOFTWARE 88
8.3.1- Primera iteración – Implementación de los grupos 90

8.3.2- Segunda iteración – Implementación de las secciones 97
8.3.3- Iteraciones sucesivas 101
8.3.3.1- Sub-Secciones 101
8.3.3.2- Razas 102
8.3.3.3- Variedades de razas 102
8.3.3.4- Sub-Variedades de razas 102
8.3.3.5- Fichero semilla 102
8.3.3.6- Usuarios 102
8.3.3.7- Personas 102
8.3.3.8- Perros 103
8.3.3.9- Exhibiciones 103
8.3.3.10- Enrolments 104
8.3.3.11- Pagos 104
8.3.3.12- Arreglos y seguridad 104
8.3.3.13- Producción 104
8.3.3.14- Pruebas finales tipo beta 105
8.3.4- Ejemplos de refactorizaciones 105
8.3.4.1- Creating Groups 105
8.3.4.2- Viewing Groups 107
8.3.4.3- Deleting Groups 109

9- DISEÑO 111

9.1- ARQUITECTURA 111
9.1.1- Patrón Modelo Vista Controlador 111
9.1.2- ActiveRecord 114
9.1.3- Transferencia de Estado Representacional (REST) 115
9.2- PORTAL WEB 116
9.2.1- Estructura 116
9.2.2- Modelo de despliegue 121
9.3- DIAGRAMA DE SECUENCIA GENERAL 125
9.4- BASE DE DATOS 127

10- IMPLEMENTACIÓN 133

10.1- ESTRUCTURA DE FICHEROS 133
10.2- DESARROLLO DE LAS PRUEBAS 135
10.2.1- Cucumber 135
10.2.2- Pruebas con Rspec 140
10.3- *HASH* DE PRECIOS 143

11- RESULTADOS Y CONCLUSIONES 149

11.1- RESULTADOS 149
11.1.1- Portal web 149
11.1.2- Pruebas 150
11.2- CONCLUSIONES 151

12- TRABAJO FUTURO 153

12.1- MEJORAS 153
12.1.1- Usuarios 153
12.1.2- Sistema 154
12.2- AMPLIACIONES 156

BIBLIOGRAFÍA 159

ANEXOS 161

ANEXO I – DETALLE DE CASOS DE USO 161
Visitante 161
Usuario registrado 164
Administrador 174
ANEXO II – FICHERO DE PRUEBA DE ITERACIÓN 12 - VIEWING ENROLMENTS 199
ANEXO III – FICHERO DE IMPLEMENTACIÓN DE LAS PRUEBAS DE LOS ENROLMENTS 201
ANEXO IV – ESTRUCTURA HASH PARA EXPOSICIÓN CANINA DE CIEZA 203
ANEXO V – REFERENCIAS DE INTERNET 205

Índice de imágenes

Imagen 1.1.1 – Muestra de varios Bulldog Inglés 1
Imagen 1.1.2 – Sesión diurna en dos muelles de la exhibición: *The Westminster Kennel Club*.... 2
Imagen 4.1.1 – Página principal FCI 10
Imagen 4.1.2 – Mensaje Ingrus 11
Imagen 4.1.3 – Mensaje Ingrus (Bulldog Francés) 11
Imagen 4.1.4 – Página principal Ingrus 12
Imagen 4.2.1 – Página principal The Westminster Kennel Club 14
Imagen 4.2.2 – Página principal de Cufts 15
Imagen 4.3.1 – Página principal LANCA 17
Imagen 4.3.2 – Página principal doglle 19
Imagen 6.1.1 – Diagrama eXtreme Programming..... 42
Imagen 6.2.1 – Esquema ciclo TDD 44
Imagen 6.3.1 – Esquema BDD + TDD..... 47
Imagen 6.4.1 – Esquema ciclo ATDD 48
Imagen 8.2.1 – Modelo de Domino..... 67
Imagen 8.2.2 – Modelo de diseño..... 70
Imagen 8.2.3 – Diagrama de actores del sistema 71
Imagen 8.2.4 – Casos de uso – Visitante – *Darse de alta – Consultar exposición –*..... 74
Imagen 8.2.5 – Casos de uso – Usuario Registrado – *Administrar perfil – Administrar canes* - . 74
Imagen 8.2.6 – Casos de uso – Usuario – *Administrar canes (Detalle)* 74
Imagen 8.2.7 – Casos de uso – Administrador –Gestionar usuarios – Gestionar personas –..... 75
Imagen 8.2.8 – Casos de uso – Administrador – *Gestionar estructura de razas (Detalle)*..... 76
Imagen 8.2.9 – Diagrama de secuencia – Visitante – Darse de alta 78
Imagen 8.2.10 – Diagrama de secuencia – Visitante – Mostrar Detalle de Exposición 78
Imagen 8.2.11 – Diagrama de secuencia – Usuario Registrado – Añadir Can sin JavaScript 79
Imagen 8.2.12 – Diagrama de secuencia – Usuario Registrado – Añadir Can con JavaScript..... 80
Imagen 8.2.13 – Diagrama de secuencia – Usuario Registrado – Eliminar Can 81
Imagen 8.2.14 – Diagrama de secuencia – Usuario Registrado – Inscribir Can en Exposición ... 81
Imagen 8.2.15 – Diagrama de secuencia – Usuario Registrado – Pagar Tasas 82
Imagen 8.2.16 – Diagrama de secuencia – Usuario Registrado – Modificar Pagos 82
Imagen 8.2.17 – Diagrama de secuencia – Administrador – Cancelar Cuenta 83
Imagen 8.2.18 – Diagrama de secuencia – Administrador – Crear Persona..... 84
Imagen 8.2.19 – Diagrama de secuencia – Administrador – Eliminar Persona 84
Imagen 8.2.20 – Diagrama de secuencia – Administrador – Añadir Exposición..... 85
Imagen 8.2.21 – Diagrama de secuencia – Administrador – Modificar Exposición..... 86
Imagen 8.2.22 – Diagrama de secuencia – Administrador – Crear Raza 87
Imagen 8.2.23 – Diagrama de secuencia – Administrador – Modificar Raza..... 87
Imagen 8.2.24 – Diagrama de secuencia – Administrador –Eliminar Raza..... 88
Imagen 8.3.1 – Mockup Groups Page 91
Imagen 8.3.2 – Mockup Creating a group..... 92
Imagen 8.3.3 – Mockup página Group I..... 92
Imagen 8.3.4 – Mockup Creating a group without a name 93

Imagen 8.3.5 – Mockup Creating a group without a name 93
Imagen 8.3.6 – Mockup Updating a group..... 96
Imagen 8.3.7 – Mockup Show updating a group 96
Imagen 8.3.8 – Mockup Updating a group without a name 96
Imagen 9.1.1 – Diagrama de la arquitectura MVC..... 112
Imagen 9.1.2 – Diagrama detallado de MVC para Rails..... 113
Imagen 9.1.3 – Mapeo Objeto Relacional (ORM) 114
Imagen 9.1.4 – Recursos en la web y sus URLs 115
Imagen 9.2.1 – Mockup página principal 117
Imagen 9.2.2 – Mockup página principal (Visión Usuario Registrado) 118
Imagen 9.2.3 – Mockup página principal (Panel Administrador)..... 118
Imagen 9.2.4 – Mockup Detalles de exhibición 119
Imagen 9.2.5 – Mockup Detalles de exhibición (Visión Administrador)..... 120
Imagen 9.2.6 – Mockup Enrolments para un usuario..... 121
Imagen 9.2.7 – Esquema general de arquitectura Cloud Computing 122
Imagen 9.2.8 – Capas en Cloud Computing 125
Imagen 9.3.1 – Iteración Modelo-Vista-Controlador estándar..... 126
Imagen 9.3.2 – Iteración Modelo-Vista-Controlador Model2..... 127
Imagen 9.4.1 – Diagrama Entidad – Relación de la base de datos 129
Imagen 9.4.2 – Relación Section-Subsection-Breed..... 131
Imagen 10.1.1 – Estructura de directorios..... 133
Imagen 10.3.1 – Tabla de precios Exposición Canina de Cieza 2014 144
Imagen 10.3.2 – Clase Parejas / Grupo de cría Exp. Canina Cieza 146
Imagen 11.1.1 – Estadísticas Cucumber..... 150
Imagen 11.1.2 – Code Coverage 151
Imagen 11.1.3 – Estadísticas Rspec..... 151

Gestor de exposiciones caninas en Ruby on Rails

1- Gestor de exposiciones caninas en RoR

Este proyecto está enfocado a desarrollar una aplicación online para gestionar exposiciones caninas, mediante el framework **Ruby on Rails**. Para entender el concepto en el que está embutido el proyecto, y para tener claros contenidos que éste abarca, vamos a empezar definiendo y comentando información acerca de lo que son y cómo se realizan las exposiciones caninas.

Imagen 4.1.1 – Muestra de varios Bulldog Inglés

Una competición canina se puede definir como un evento en el que una serie de jueces, entendidos en diversas razas de perros, califican y evalúan cuán de bien conformado se encuentra un perro en comparación con el estándar racial. Estas competiciones están típicamente amparadas bajo los auspicios de una asociación o club nacional de registro caninos. En los niveles más altos de la competición están los campeonatos de todas las razas en el que se evalúan clases separadas para la mayoría de razas. A nivel más bajo nos encontramos con competiciones para una raza o grupo específico, organizadas generalmente por el club o asociación respectivo y se conocen como *exposiciones de especialidad* o *monográficas*. Las exposiciones tienen una historia de más de 150 años y la primera de ellas fue llevada a cabo en Newcastle upon Tyne, Inglaterra, en el año 1859.

La evaluación en estas competiciones consiste en, mediante el juicio de los jueces caninos y bajo los estándares publicados para cada raza, identificar aquellos perros que se encuentran mejor dentro de su categoría. Es una tarea de alto reto, ya que muchas evaluaciones serán necesariamente subjetivas: ¿Qué significa exactamente “patas delgadas”, “torso elevado” o “actitud alegre”?, descripciones que se pueden encontrar en los estándares de raza.

De forma más estricta, hay que diferenciar que una exposición no consiste exactamente en una comparación entre un perro y otro, sino en una comparación de cada competidor con un paradigma de ejemplar ideal que tiene el juez basándose en el estándar la raza, en el que están los atributos de una determinada raza y su lista de puntos de conformidad. Basándose en esto, los jueces dictaminan qué perro debe estar mejor colocado en el escalafón.

Los perros compiten en las exposiciones para acumular puntos con miras a obtener un título de campeón. Cada vez que un perro gana en alguna instancia de la exposición, acumula ciertos puntos para el campeonato. El número de puntos acumulados varía en cuanto al nivel o instancia en la que se encuentra la exposición, del número de perros que compiten y de si la exposición es larga o corta. Además, el número de puntos necesarios el título de campeón

también varía dependiendo del país o de la asociación. En este enlace podemos encontrar el estándar descriptivo del "perfecto" Podenco Canario.

La competición es de forma jerárquica en cada clase, en las que los ganadores de instancias inferiores son gradualmente combinados con otros ganadores, estrechándose cada vez más el círculo, hasta el punto en el que se escoge al *Mejor Ejemplar de Exposición*. En las instancias inferiores se dividen a los perros por raza, y en cada raza se dividen por género y edad. Los grupos en los que se encuentra cada raza se pueden encontrar en la página oficial de la FCI (Federación Cinológica Internacional) o, mirando un poco hacia casa, en nuestra propia aplicación desarrollada (DoogyHouse).

Las exposiciones caninas tienen lugar durante todo el año en varias localizaciones. Nos podemos encontrar con competiciones de varios tipos: pequeñas y locales, grandes, nacionales e internacionales. Algunas son tan grandes que limitan su entrada a perros que hayan ganado algún campeonato. Es por ello que ganar un título de "*Mejor de la raza*" o "*Mejor de la exposición*" suele elevar la categoría y reputación del perro, del criador, y del club o asociación de origen, pudiendo verse reflejado en un incremento en los precios de los cachorros de su respectiva propiedad. Esto también incrementa su popularidad, ya que mucha gente decidirá que desean un perro de la misma raza y "*justo como el que ganó tal competición*" (arma de doble filo ya que puede llevar a una reproducción irresponsable y en masa de la raza por parte de criadores sin ética, que encuentran en la popularidad de la raza una oportunidad de lucro).

Actualmente, las dos mejores, más grandes y más prestigiosas competiciones, probablemente serán la de Westminster Kennel Club Dog Show, en Estados Unidos y Crufts, en el Reino Unido.

Una vez hemos visto los conceptos básicos referentes a las exposiciones caninas, vamos a pasar a describir los objetivos que buscamos conseguir con la realización de nuestro proyecto.

Imagen 4.1.2 – Sesión diurna en dos muelles de la exhibición: *The Westminster Kennel Club*

2- Objetivos

En este punto del documento, vamos a analizar los objetivos que se quieren cumplir con la realización del proyecto, tanto a niveles didácticos como profesionales y de oportunidad que puede tener esta aplicación en el ámbito comercial.

Dentro del proyecto hay varios objetivos claves que se pretende y se quiere conseguir mediante el desarrollo del mismo. El primer punto que vamos a analizar es referente a los objetivos desde un enfoque de aprendizaje, es decir, desde un punto de vista didáctico para el alumno, en los que se pretende, entre otros, que se familiarice con tecnologías y metodologías actuales y que pueden resultar muy útiles a la hora de la salida al mundo laboral. El otro aspecto en el que nos vamos a centrar es en referencia al ámbito profesional y de oportunidades que la aplicación puede abarcar una vez acabada.

Centrémonos en el primer punto que hemos citado, los objetivos desde un ámbito didáctico:

La aplicación, como hemos mencionado anteriormente, se ha desarrollado utilizando un framework de Ruby, **Ruby on Rails** y va a estar dirigida por una **metodología de desarrollo basada en pruebas**, concretamente por una metodología ATDD (Acceptance Test Driven Development – Desarrollo guiado por pruebas de aceptación). Dicho framework está actualmente en auge, y cada vez está siendo más utilizado y requerido en el mundo de la informática. Cada mes que pasa son más los puestos de trabajo en los que solicita a personal con conocimientos en Ruby, y más concretamente en *Ruby on Rails*. Es por ello que uno de los objetivos principales es que, mediante la realización del proyecto, el alumno obtenga unos conocimientos de *Ruby on Rails* medios (o avanzados) que le abra puertas respecto a puestos de trabajo en un futuro inmediato.

Por otro lado, las metodologías de desarrollo basadas en pruebas son un tipo de metodologías que el mercado está demandando cada vez más. Actualmente hay varios bandos en el debate en cuanto a integrar este tipo de tecnologías en desarrollos. Por una parte están las mentes más abiertas al cambio y modernas, que aceptan y apoyan al cien por cien este tipo de desarrollo; por otra, los que están dubitativos y tienen algo más de reparo en incluir este tipo de tecnologías; y otro sector más es el del punto extremo en el lado opuesto, el que considera, simplemente, una pérdida de tiempo hacer pruebas de funcionamiento del software a nivel de código antes de realizar el desarrollo en sí. Hay que tener en cuenta que este tipo de desarrollo es un tipo de metodología de desarrollo ágil y todavía hay mucha reticencia al uso de este tipo de técnicas, aunque echando un poco la mirada hacia adelante y con una visión de futuro, se considera que es importante el conocer y manejar este tipo de metodologías, ya que se espera que, poco a poco, el sector laboral se unifique y que este tipo de técnicas se

conviertan en una metodología totalmente dominante. Es por ello que también se ha decidido incluir este tipo de metodologías en el desarrollo de la aplicación, de modo que el aprendizaje y manejo de metodologías de desarrollo guiadas por pruebas entre dentro de principales objetivos del proyecto.

El otro objetivo principal que hemos comentado es un objetivo a nivel funcional y comercial de la aplicación. Es de lo que pasaremos a hablar a continuación:

El sector de los perros, es un sector en el que se invierten muchos recursos, tanto tangibles, como intangibles. Si nos centramos en los recursos tangibles, y más concretamente en lo referido al recurso monetario, el campo de las exposiciones caninas es un campo en el que se invierte mucho dinero. El simple hecho de organizar una exposición canina (publicidad del evento, alquiler de medios, contratación de personal, adecuación del sitio…) conlleva unos gastos más que considerables para cualquier negocio; si, además, le sumamos los gastos de las personas que participan en ellas (inscripción de perros, desplazamientos, estancias…) y otros muchos gastos (e ingresos) derivados de esto, se puede apreciar que hay un buenísimo mercado en el que se puede fijar uno para plantearse un negocio.

Las competiciones caninas no llevan mucho tiempo tratándose por medios informáticos, es decir, con sistemas informáticos que apoyen y sostengan estos tipos de eventos. El objetivo comercial del prototipo a desarrollar va por este camino. Ya que no hay mucha variedad dónde elegir, y que este sector carece de buenos medios informatizados para una gestión completa de todo esto, se quiere conseguir una aplicación que vaya aportando al consumidor la mayor funcionalidad posible en lo referente a todo este mundo. El objetivo inicial del proyecto de final de carrera, era este, conseguir un sistema con estas características y montado en un servidor de forma estable.

Sin embargo, una aplicación que consiga esto, bien desarrollada, probada, totalmente operativa y con tecnologías – tanto de desarrollo, como de diseño – nuevas para el alumno, va mucho más allá de un proyecto individual de final de carrera. Por esto, el objetivo inicial de conseguir la aplicación completa se acotó a algo más adecuado para un proyecto de este tipo.

El objetivo comercial de la aplicación final sigue siendo el mismo, conseguir un sistema estable e integral que aporte la mayor funcionalidad posible para el mundo de las exposiciones caninas; sin embargo, para esta fase inicial de esta aplicación, y cómo objetivo del proyecto se ha fijado la meta de conseguir un buen prototipo base de este sistema. Es decir, acotar la finalidad tanto funcional como operativamente.

De modo funcional, el objetivo se ha acotado a conseguir un sistema capaz de inscribir a perros a exposiciones caninas. Para ello se deberá tener un sistema capaz de soportar usuarios (con varios roles), perros (con sus subsistemas derivados – cómo las razas y las variedades) y exposiciones en las que se puedan inscribir a los perros. De este modo, un usuario registrado podrá registrar a su perro en la aplicación, e inscribirlo en las competiciones habilitadas en el sistema, de acuerdo con las restricciones y precios propios de cada una de ellas.

En cuanto a la operatividad del sistema, lo que se busca es que el sistema sea estable y libre de errores, con un contenido totalmente testeado y abierto a modificaciones y futuras mejoras. También se pretende que el prototipo esté montado sobre una plataforma que permita su utilización, accesible por internet y desde un ordenador personal, teniendo así la posibilidad de mostrar un ejemplo de funcionamiento y captar un posible cliente, que quiera "comprar" nuestro producto.

Conseguir un prototipo de aplicación que reúna las características presentadas anteriormente, y ante tecnologías novedosas para el alumno, es una buena síntesis del objetivo final que el proyecto busca.

3- Estructura del documento

Con la intención de facilitar la lectura del presente documento, vamos a indicar, en este apartado, la estructura general y aspectos más relevantes que hay en el mismo.

Una vez realizada la introducción en el mundo de las exhibiciones caninas y haber presentado los objetivos fundamentales del proyecto, vamos a estudiar cada una de las fases en que ha consistido el proyecto.

En primer lugar veremos un estudio del estado del arte actual y las herramientas que existen que pueden competir con respecto a nuestra aplicación. Se analizarán aplicaciones actuales que están en funcionamiento y que realmente son utilizadas por el sector cinofílico.

A continuación nos centraremos en un estudio de herramientas en el que analizaremos las principales herramientas que existen para realizar el proyecto que queremos hacer. Aunque tengamos claro la metodología de desarrollo y las herramientas que vamos a utilizar, vamos a presentar diversas alternativas que se podrían haber tenido en cuenta para un trabajo similar.

Lo siguiente que veremos será el desarrollo de una explicación de la metodología que hemos utilizado para la realización del trabajo, así como los recursos que se emplearán.

El siguiente apartado que nos vamos a encontrar es el referente al análisis del sistema. Hablaremos de nuestro sitio web en prototipo, comparándolo con otros portales analizados en apartados anteriores; hablaremos de las pruebas realizadas en la fase previa al desarrollo de la plataforma, y especificaremos los requisitos del sistema, tanto de usuarios, cómo del software.

En lo que respecta al diseño, estudiaremos la arquitectura del sistema, que veremos que será una arquitectura basada en Modelo-Vista-Controlador. Por otro lado, veremos opciones de diseño del portal web, presentando algunos mockups de cómo se desearía que se presentase de cara al cliente dicho portal. También estudiaremos el diagrama de secuencia general del sistema y el diseño y la estructura de la base de datos del sistema.

Lo referente al desarrollo e implementación del sistema lo encontraremos en el siguiente bloque. No entraremos en detalles de código innecesario, el cual se puede consultar en los ficheros fuentes de la aplicación, sino que haremos un estudio general de la estructura de ficheros, detallaremos algunas pruebas realizadas con distintas aplicaciones de prueba, y mostraremos la manera en que se ha implementado la tabla de precios de las exhibiciones.

Para finalizar, veremos apartados de resultados y conclusiones, y también trabajo futuro que puede ser desarrollado en base a nuestro trabajo realizado.

En los anexos del documento empezaremos mostrando los detalles de los casos de uso citados en el apartado de análisis. A continuación veremos el detalle de un fichero correspondiente a una de las pruebas de aceptación y en el Anexo III se mostrará el código que implementan los *steps* de las pruebas referentes a las inscripciones de los perros en exposiciones (los *enrolments*). El último anexo mostrará las referencias utilizadas de internet.

4- Estado del arte

Este apartado del documento está destinado al estudio de la situación de portales web que, en la actualidad, se pueden encontrar operativos y utilizados en el sector de las exhibiciones caninas. Para ello agruparemos el estudio en tres categorías: Portales que proveen información especializada del sector de la cinofilia, sitios web de las más consagradas exposiciones caninas mundiales y plataformas para la administración de exhibiciones. Estudiaremos dos ejemplos de cada uno de los tres grupos especificados ya que entendemos que, a modo de referencia y de estudio, no hay necesidad de analizar particularmente más sitios que los que vamos a detallar, aunque dejaremos referencias de algunos portales más.

Por un lado estudiaremos los sitios de este sector que son útiles y muy utilizados por personas a las que le interesa la materia (tanto expertos, como novatos): La **FCI** (Federación Cinológica Internacional) e **Ingrus**. A continuación mostraremos y comentaremos los portales de dos de las exhibiciones más importantes del mundo: La **WKC** (Westminster Kennel Club) y **Crufts**. Y para finalizar, nos centraremos en los sitios de dos de las herramientas más utilizadas actualmente para la gestión de exposiciones caninas: **Doglle** y **Lanca**.

4.1- Plataformas de información especializada

En esta sección vamos a analizar distintos sitios que, aunque no están destinados directamente al control y a la gestión de las exhibiciones, tienen que ver con el sector canino de las mismas. Hay que considerar este estudio importante y relevante varios motivos principales: Entender el mercado por el que nos movemos; recopilar datos en cuanto a futuras maneras de obtener publicidad y posibles expansiones; analizar la estructura y el contenido de sitios importantes relacionados con el mundo canino, etc. Es decir, capacidades que tenemos que considerar a la hora de desarrollar la aplicación. Con todo ello, vamos a ver dos de los sitios más importantes en este campo: la **FCI** e **Ingrus.**

4.1.1- Federación Cinológica Internacional (FCI)

La **Federación Cinológica Internacional** – o en su forma original Federation Cynologique Internationale – es una referencia mundial para el sector cinológico y, más concretamente, para el de las exhibiciones caninas (click aquí para visitar). En él se puede encontrar información abundante acerca de muchos temas relacionados con las exposiciones:

- Razas (sus estándares, variantes, grupos homologados…)
- Calendarios de exposiciones (exposiciones, carreras, campeonatos, fechas…)
- Reglamentos (sobre estatutos, dopajes, exposiciones, jueces…)
- Resultados de exposiciones
- Estadísticas…

La **FCI** es la organización canina mundial. Consta de de 5 secciones: Europa, Américas y el Caribe, Asia y pacífico, Oriente Medio y África y reconoce un total de 343 razas mundiales. Posee un total de 90 miembros, y cada miembro de la federación lleva a cabo exposiciones

internacionales de Belleza así como concursos internacionales de trabajo, pruebas de caza, competiciones de Agility y Obedience, carreras, coursing y pruebas para perros de rebaño.

Esta organización se encarga también de procesar los resultados de las exposiciones de belleza, así cómo homologar los títulos conseguidos por los perros en las exposiciones.

Es decir, podemos encontrar una multitud de información, que cualquier persona que quiera conocer, entrar o, inclusive, expertos en este mundo, debe tener en cuenta y consultar. Además, toda esta información la encontramos de una forma muy bien organizada y clara. Actualmente se ha modificado el sitio (tano visualmente cómo su navegabilidad) y ha pasado a tener un aspecto mucho más amigable y de mayor facilidad para la utilización del usuario. A continuación presentamos una pequeña captura de pantalla de su página principal, a modo de tener una referencia gráfica del portal que nos hemos referido.

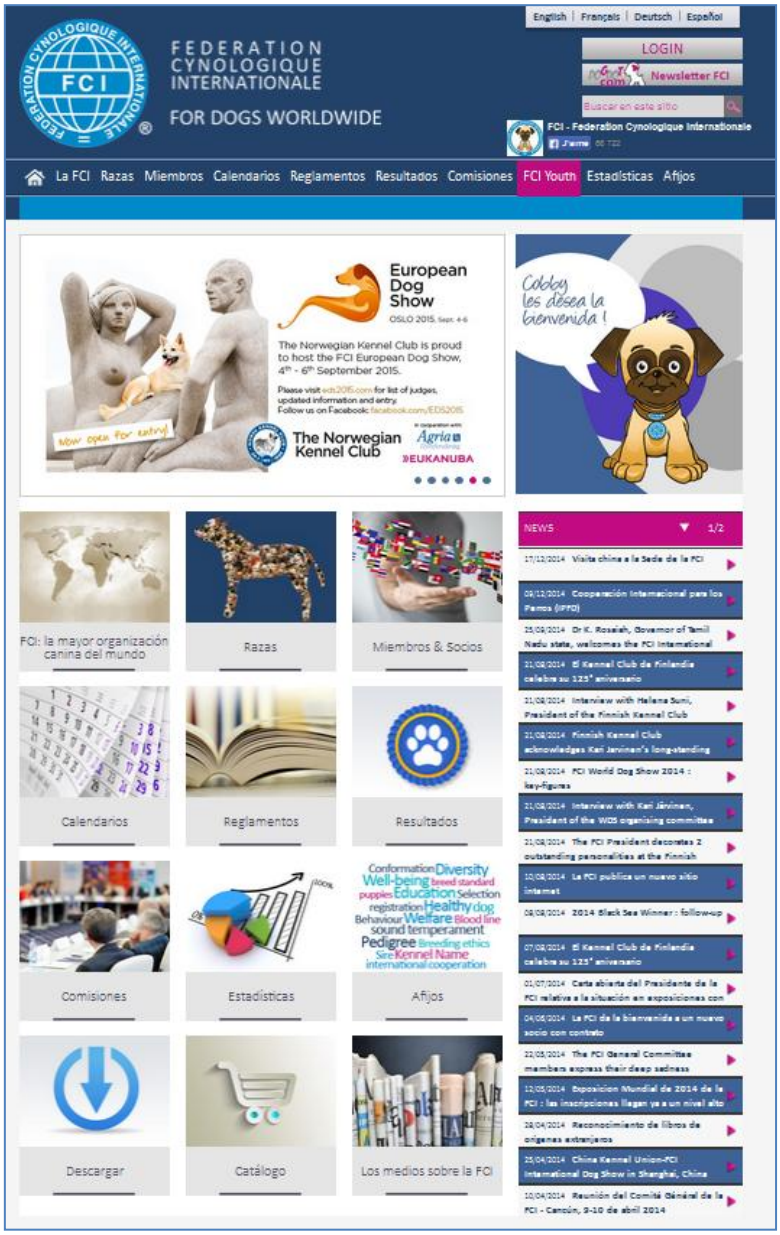


Imagen 4.1.1 – Página principal FCI

4.1.2- Ingrus

Ingrus es otra referencia de consulta mundial en el sector canino. Su simple misión es guardar información de perros, es decir, no es más que un portal dedicado a guardar información de perros, en el que los usuarios dejan registrados sus perros con los datos que se les requiere (click aquí para visitar).

Visto así, podríamos considerar a Ingrus como un portal muy simplista y un sitio poco atípico; nada más lejos de la realidad. La realidad es que Ingrus realmente es una base de datos genealógica mundial de pedigríes caninos. No solamente podemos encontrar información detallada y muy bien estructurada de la mayoría de los perros existentes en este mundo (o más bien de aquellos que a los dueños les interesa tener de forma pública), sino también su pedigrí y la información de su pedigrí, es decir, también acerca de su descendencia/ascendencia genealógica.

Nada más entrar en su portal (lamentablemente, una de las pegas que tiene la página principal es que está en ruso), podemos ver el siguiente mensaje:

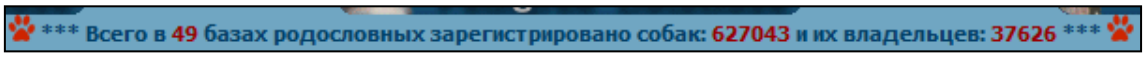


Imagen 4.1.2 – Mensaje Ingrus

Que viene a decir que existen 49 pedigríes, 627.043 perros y 37.626 dueños registrados en su sistema. Que sí, si lo comparamos con Amazon o Ebay, puede que no resulte tan sorprendente, pero para este sector, esos datos sí que son muy sorprendentes. Para hacernos una idea, si navegamos por su página y vamos a uno de los pedigrís (por ejemplo el del bulldog francés), podemos ver que existen ni más ni menos que 77.589 perros registrados con este pedigrí, algo que ya nos puede ir resultando más impresionante.

Imagen 4.1.3 – Mensaje Ingrus (Bulldog Francés)

Por su forma de gestionar todos los datos que en ella se recoge, y por la tradición que tiene en este mundo (activa desde el año 2004), es otra de las páginas consideradas de referencia mundial para la cinofilia.

También posee algunas otras funciones como la de un foro propio en el que se debaten diversos temas relacionados con el sector, galería de fotos, una sección de venta de cachorros… Hay que puntualizar que, aunque su página principal esté en ruso, una vez nos navegamos por la página y nos metemos en información más específica de pedigrís y demás, el contenido es personalizable al inglés.

A continuación vamos a ver una imagen recortada de la pantalla de bienvenida de Ingrus:

Imagen 4.1.4 – Página principal Ingrus

Accediendo a cualquiera de las secciones de pedigrís del sitio (tres de ellas mostradas en la parte superior del panel principal) ya podemos adquirir más información acerca del pedigrí en cuestión, así como un video explicativo de la manera de utilizar la página y alguna de sus funciones principales. Como hemos mencionado, una vez accedemos a información específica de algún pedigrí, podremos definir el idioma del contenido al inglés (o dejarlo en ruso).

Vistos los dos sitios que hemos considerado más relevantes en esta sección, vamos a nombrar algunos otros sitios interesantes que podríamos referenciar y analizar en el presente documento: American Kennel Club, Dog's Show, Our Dogs Internacional, RSCE… Todos ellos buenos referentes con información del sector. Sin embargo, como hemos mencionado al inicio del apartado, con los portales que hemos analizado ya tenemos un buen material para meternos en materia.

4.2- Exhibiciones mundialmente reconocidas

A continuación vamos a presentar portales web de exposiciones caninas mundiales. De hecho, los dos portales que vamos a ver, son los referentes a las dos exhibiciones caninas consideradas más importantes en el sector: **The Westminster Kennel Club** y **Crufts**, la primera organizada en Nueva York, y la segunda en el Reino Unido.

4.2.1- The Westminster Kennel Club

El **Westminster Kennel Club** es una organización destinada a la realización de uno de los mayores eventos dentro del mundo de las exhibiciones caninas, y que lleva este mismo nombre: "*The Westminster Kennel Club*". Su prestigio viene alabado por más de 100 años de exposiciones, concretamente, este año nos enfrentamos a su edición número 139 (su andadura comienza en 1877). Es una competición de dos días de duración, organizada en Nueva York, en el que los mejores perros compiten para obtener el título que les reconozca como mejores de su raza, grupo o, inclusive, el mejor del Show.

Estudiando el sitio web, podemos observar a primera vista que está organizado de manera muy bien estructurada y que presenta casi exclusivamente información respecto al evento que organiza (click aquí para ver). A parte de esto, nos muestra una pequeña información varia referente al mundo canino.

La información está organizada y ordenada por menús desplegable en la parte superior del cuadro principal. Son menús desplegables que, al clicar sobre ellos, se despliegan y nos enseñan accesos a información acerca de varios temas referentes al WKC Dog Show:

- Información muy variada para la exhibición de este año: **2015 Dog Show**.
- Los resultados de la exposición del año pasado
- Videos y fotos varias así cómo una sección de tienda
- Informaciones diversas acerca de la propia WCK, perros e historia y registros.

Lo que más salta a la vista y nos llama la atención de este sitio web es la forma tan compacta que tiene de organizar la información. Tiene claro sus objetivos base, que son los de promover e informar acerca del evento que se organiza, y, con ello, nos facilita mucha información de una forma muy bien organizada. De modo que, simplemente con unos cuantos clicks, podemos encontrar lo que buscamos. Este es uno de sus puntos más fuertes y con el que sin duda nos tenemos que quedar, el organizar la información de tal manera que sea de fácil acceso para el usuario, sin descuidar la funcionalidad del sitio y ofreciendo todo lo que el consumidor desea.

A parte de los accesos desplegables nombrados, también tiene accesos directos a varia información. Podemos acceder directamente al programa para el evento de este año, al calendario para el mismo, información de tickets para entrar, accesos a "*Follow us*", etc. Actualmente el sitio se encuentra en expansión, y está promoviendo la nueva aplicación para móvil, notificándonos que próximamente estará disponible.

La imagen que mostramos seguidamente corresponde a una captura de pantalla de la página principal de "*The Westminster Kennel Club*". Podemos ver la buena organización condensada en forma de paneles en la parte superior del cuadro principal, como nombramos anteriormente:

Imagen 4.2.1 – Página principal The Westminster Kennel Club

4.2.2- Crufts

Crufts es otra de las exhibiciones mundiales con mayor renombre. Si WKC, vista anteriormente, es la considerada más prestigiosa del mundo, Crufts Dog Show de Inglaterra es el subcampeón más cercano. Posee el título de la más grande exposición canina del mundo. La Crufts está dirigida por el Kennel Club y se ha convertido en más que un espectáculo de razas, como ya pasa con la WKC. En ella también se cuentan con eventos de agilidad, pruebas de rescate y competencias de DogDancing (o bailando con los perros).

Crufts se celebra cada año, en marzo, en Inglaterra y sus orígenes se remontan a 1891. Su duración es de 4 días, y los perros se dividen en 7 grupos. Para entender la magnitud de este evento atendamos al siguiente dato: la competición más importante de España, celebrada en Madrid, este año tuvo una tasa de participación de 4.000 perros; en Crufts se llegó en torno a los 28.000.

Accediendo al sitio web de este evento (click aquí para ver), podemos ver que la estructura general de la página es muy similar a la de su competidora directa la WKC. El modo de acceder a la información es mediante un panel de desplegables situado en la parte superior del portal. También predomina un color intenso de fondo y hay algunas secciones accesibles

directamente mediante un simple click. A modo de desplegables podemos acceder a información referente a lo siguiente:

- Gestión de entradas
- Programas de los 4 días de duración del evento
- Información acerca de la exhibición
- Otra información varia respecto a jueces, calificaciones, prensa…

Al igual que el portal estudiado previamente, la información la encontramos muy bien dividida en secciones y con muy fácil acceso a la misma. Con unos cuantos clicks accedemos directamente a cualquier contenido que estemos buscando dentro de sitio web.

Para acabad esta sección vamos a ver una captura de pantalla de la página de bienvenida de Crufts:

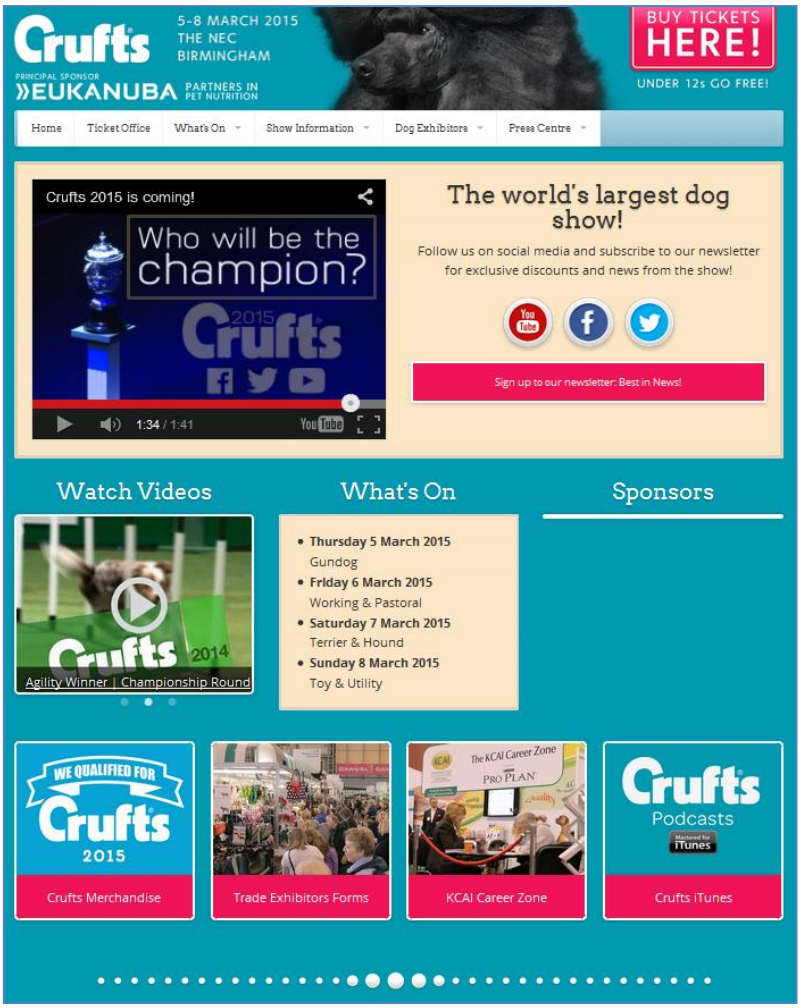


Imagen 4.2.2 – Página principal de Cufts

Normalmente las exposiciones son organizadas por clubs o asociaciones del país en el que se llevan a cabo, por lo que no suelen tener sitios web propios. Por lo tanto es difícil encontrar otros portales específicos a exposiciones, aunque, a modo de ejemplo, otro portal dedicado a una exposición canina en particular puede ser el World Dog Show, organizado por la FCI; o el Dog Lovers Show, emplazado en Melbourne y Sidney.

4.3- Plataformas de gestión de exhibiciones caninas

Entramos en esta última sección dentro del apartado destinada al estudio de las dos de las plataformas actuales más utilizadas para la gestión de exhibiciones caninas, es decir, nuestros competidores directos. Vamos a centrarnos en un portal a nivel nacional (**Lanca**), y otro a nivel internacional (**Doglle**). Empecemos con el primero.

4.3.1- LANCA

Lanca es un portal web que proporciona el acceso a multitud de información, servicios, programas y datos en internet relacionados con la cinofilia. Se define a sí mismo como un organizador de actividades caninas, y su sitio está íntegramente en español, sin posibilidad de cambiar de idioma.

La información en el sitio se presenta a modo de una página principal de bienvenida, una barra de navegación en la parte superior, y una serie de enlaces a distintas secciones de la web en la parte lateral derecha. La barra superior nos permite acceder al apartado de inscripciones, de exposiciones en preparación y contacto e información legal. Realmente, el potencial de información del portal lo encontramos en el panel lateral a modo de enlaces. Estos enlaces están agrupados en 5 bloques:

- Exposiciones en preparación
- Inscripciones online
- Exposiciones pasadas
- Exposiciones futuras
- Menú

El contenido de los 4 primeros es muy intuitivo de adivinar, y en el bloque de menú se nos aporta enlaces a sitios como el calendario de exposiciones, los estándares y las razas, reglamentación varia, contacto…

Estudiando y navegando un poco por el sitio, no se tardas demasiado en darse cuenta de que está un poco abandonado. Si bien es cierto que da la impresión de que está actualizado (ya que provee posibilidades de inscripción a exhibiciones actuales del 2015), al navegar por varias secciones del portal observamos de que muchos de sus enlaces son enlaces a sitios no operativos, o simplemente son enlaces a documentos PDF u otros portales que posee la información que se está buscando. De modo que algo hace pensar que no se invierte mucho tiempo en mantener el sitio correctamente actualizado. Además, otro punto a tener en cuenta es que no dispone de opción de registro de una cuenta.

Por otro lado, sí que es verdad que posee un contenido aceptable en cuanto a información de exposiciones pasadas, ya que nos informa de resultados de varios torneos nacionales acontecidos desde el año 2011 (mayoritariamente exposiciones valencianas).

En cuanto a la inscripción en exhibiciones caninas, se hace de manera inmediata, sin requerir registro alguno en la página: Se selecciona la exposición a la que deseas inscribirte y rellenas un formulario que se presenta a continuación de la información de la exposición. Los métodos de pago que permite son giro postal y recibo bancario, advirtiéndonos previamente

que ningún registro será concluido hasta que no se envíe el comprobante escaneado a una determinada dirección de correo electrónico.

En una visión general, el portal cumple con su objetivo de permitir la inscripción en exposiciones nacionales, proporcionando también información varia del sector de la cinofilia. Si bien, haciendo una valoración crítica del sitio, realmente nos deja con un sentimiento de falta: La parte de enlaces rotos, el sistema de inscripción (no requiriendo un registro previo en el portal), el contenido general que abarca… Son aspectos muy mejorables y que hacen de poca confianza al portal.

A continuación vamos a presentar su página principal en la que podemos observar lo comentado anteriormente:

Imagen 4.3.1 – Página principal LANCA

4.3.2- Doglle

A un nivel mucho más elevado respecto a gestión de exposiciones caninas nos encontramos con **Doglle**. Doglle es un portal web de origen francés, que únicamente ofrece un servicio y la información relacionada con ese servicio: La inscripción a exposiciones, eso así a

un nivel muy aceptable. Es uno de los portales más recurridos para la inscripción a exposiciones caninas y esta transcrito a 14 idiomas.

La información es bastante limitada, pero completa para la labor que se requiere en este sitio. Es posible la navegación por el portal sin tener una cuenta registrada siempre y cuando no se pretenda enrolar en una exhibición. Para inscribir a un perro en una exposición hay que tener una cuenta registrada y perros adheridos a esa cuenta.

Para la navegación por el sitio se nos presenta una barra superior con las siguientes posibilidades:

- Exposiciones
- Favoritos
- Mi cuenta
- Mis perros
- Mis compromisos
- Asistencia

No tiene ningún menú desplegable, ni ningún enlace directo a simple vista. Sin embargo, a medida que vamos estudiando el portal nos damos cuenta que tiene su labor muy bien establecida. Las exposiciones (en el apartado exposiciones) las permite ordenar por países y por años y además advierte de las que están a punto de cerrar el plazo de inscripción. Para cada una de las exposiciones nos da la posibilidad de descargarnos el documento PDF de inscripción y el estadístico, programación y resultado en caso de existir. Además, nos ofrece una sección de favoritos en la que podemos incluir aquellas exhibiciones a las que queremos seguir.

Para la gestión de las exhibiciones hay que estar registrado en el portal, y una vez registrado, podemos añadir nuestros perros a nuestra cuenta. De modo que quedan registrados bajo nuestro poder y podemos inscribirlos en las exhibiciones en las que sea posible. El intento de inscripción está supervisado bajo condiciones internas, de modo que no es posible hacer inscripciones aleatorias o erróneas de perros “fantasma”, es decir, inscripciones falsas o que no cumplen con los requisitos para ello.

Además, el portal contiene un servicio de mailing que utiliza, bajo el consentimiento del propietario de la cuenta, para avisar de eventos que están a punto de cerrar el plazo de inscripción.

En la parte inferior de la página principal ofrece una serie de enlaces a información varia respecto al sitio, y para dar la posibilidad de apoyarlo mediante una serie de banners. Como hemos dicho, no ofrece ninguna otra información relacionada con el mundo de la cinofilia, pero consideremos que realmente no es necesario, ya que cumple tan bien su función y lo hace de una forma tan clara y fácil, que no necesita de otros recursos para estar en el top de plataformas usadas.

Como hemos hecho con los demás casos, vamos a mostrar la página principal de Doglle:

Imagen 4.3.2 – Página principal doglle

Si comparamos los dos portales contra los que debemos competir, tal vez una mezcla de ambos sería una combinación perfecta. Por un lado la claridad, limpieza y funcionalidad de Doglle; por otro, información complementaria en el sitio, para que no sólo sea una referencia a la hora de inscripciones en exhibiciones caninas, sino que también sea posible consultar alguna información extra como puede ser reglamentaciones, estándares, historial de exhibiciones…

5- Estudio de herramientas

Nuestro proyecto va a estar desarrollado bajo el framework Ruby on Rails pero con la finalidad de entender y estudiar diversos caminos por los que podríamos haber optado para la evolución del proyecto "**Gestor de Exposiciones Caninas en Ruby on Rails**", pasaremos a continuación a especificar y analizar distintas herramientas que nos pueden complacer para un eficiente desarrollo de dicho proyecto, así como las necesidades que van a cubrir las mismas y las aportaciones que nos podrán ofrecer.

Estudiaremos cada una de las herramientas individualmente, excepto tres en particular (en la siguiente sección explicamos el porqué de agrupar estas tres en un mismo conjunto), y las vamos a agrupar en 3 grandes bloques. De modo que por un lado estudiaremos y comparemos las herramientas disponibles de desarrollo; por otro lado las plataformas que podemos utilizar para el control de la nube; y para terminar agruparemos en otro bloque herramientas de testeo que consideremos pueden ser competitivas para incluir en el proyecto. Al final de este apartado, y para concluir el bloque, haremos una valoración global y una explicación del porqué de las elecciones tomadas para realizar el proyecto.

5.1- Herramientas de desarrollo

Este proyecto en particular está orientado para trabajar con Ruby on Rails, pero no por ello vamos a descuidar el estudio de otras herramientas de desarrollo que nos proporcionan características similares a las que nos ofrece Ruby on Rails, y es por este motivo que vamos a hacer un estudio de las herramientas de desarrollo que hemos considerado más relevantes.

Para la elección de las distintas herramientas de desarrollo, y en general de gestión de la web, se han tenido en cuenta diferentes aspectos importantes que a continuación nombraremos:

- Ser de fácil utilización y parcialmente intuitivas para administradores y usuarios.
- Que sean adaptables a los distintos requerimientos que puedan surgir durante el desarrollo y que no estén preestablecidos desde un primer momento.
- Ser de uso extendido, compatibles, y con una suficiente documentación disponible.
- Dichas herramientas deben de ser fiables y seguras.
- Que estén basadas en lenguajes de programación web de uso extendido.
- Que sea una herramienta que aporte conocimiento al alumno a fin de proyecto de fin de carrera.

Cómo se ha nombrado anteriormente, se va a proceder un estudio individual de cada herramienta considerada con potencial para la implementación del proyecto. Sin embargo, previo a hacer esto, y debido a la similitud que hay entre tres herramientas en particular, vamos a hacer una pequeña comparativa entre los 3 sistemas de gestión de contenidos más comunes en la actualidad (**Joomla!**, **Drupal** y **Wordpress**), y vamos solamente a estudiar y detallar el que consideremos más acorde para nuestros objetivos.

Una vez realizada la comparativa de estos tres sistemas mostraremos las herramientas estudiadas para su posible elección, con una descripción de cada una de ellas y sus principales características.

5.1.0- Drupal – Joomla! – WordPress

Se han seleccionado estos tres únicos sistemas de gestión de contenidos para su equiparación ya que, en la actualidad, son las tres herramientas más extendidas y utilizadas por el sector informático debido a la simpleza y las amplias posibilidades que aportan.

Joomla, Drupal y WordPress son tres plataformas de software libre que permite a cualquier usuario desarrollar su propio sitio web. Realmente estamos hablando de tres sistemas de gestión de contenidos (en inglés **CMS** – *Content Management System*) con similares características pero con pequeñas diferencias. Una de los puntos que más se destacan para la elección de uno de estos sistemas, es tener en cuenta el nivel de profesionalidad del desarrollador. De mayor a menor nivel de experiencia en el desarrollo web se recomienda utilizar Drupal > Joomla! > WordPress, siendo este último recomendado para usuarios con conocimientos básicos.

De este modo, si eres un desarrollador web profesional y tienes un nivel de codificación avanzado, se recomienda la utilización de Drupal. Páginas como la de The Economist, MTV o The White House están construidas sobre esta plataforma.

Con Joomla! Estaríamos sobre la misma línea que Drupal, pero un nivel un poco inferior. Está ideada para conocedores o aficionados en el desarrollo web. Tanto Drupal como Joomla! fueron creadas como herramientas **CMS** avanzadas para webmasters, y por ello exigen al usuario un nivel medio avanzado de desarrollo web. La página de Linux, UNICEF o la propia Porsche trabajan sobre Joomla!.

Por último, para programadores con conocimientos nulos o básicos se recomienda la utilización de WordPress. Sin embargo, aunque la plataforma fue creada con ideal de generar blogs sencillos, el crecimiento de WordPress y más de 20.000 plugins permiten la integración de redes sociales, anuncios, tiendas online… y siempre manteniendo el espíritu original de simplificar la interfaz. Páginas como la de The Rolling Stones o University of Florida, han sido creadas con esta herramienta.

Vamos a exponer grosso modo algunas de las ventajas e inconvenientes de cada uno de los sistemas que estamos analizando:

Drupal

- **Ventajas**
  - *Extremadamente flexible* – Permite desarrollar desde portales sencillos hasta un poderoso back-end que pueda soportar millones de usuarios.
  - *Amigable con el desarrollador* – La instalación básica es esencial, animando a desarrolladores a crear sus propias soluciones.

  - *Grandes capacidades **SEO*** – Drupal fue diseñado para ser amigable con los motores de búsqueda.
  - *Amigable con las Empresas* – Fuerte control de versiones y capacidades **ACL** (Access Control List).
  - *Estabilidad* – Es escalable sin esfuerzo y estable incluso con miles de usuarios a la vez.
- **Inconvenientes**
  - *Gran curva de aprendizaje* – Cambiar de un **CMS** simple a Drupal puede ser muy complicado.
  - *Falta de plugins gratuitos* – La mayoría de los buenos plugins (llamados "*módulos"* en Drupal) no son gratuitos.
  - *Falta de temas* – Una instalación básica de Drupal es muy básica, y la falta de temas no mejora las cosas.

Joomla!

- **Ventajas**
  - *Amigable con el usuario* – Es relativamente sencillo de utilizar.
  - *Fuerte comunidad de desarrolladores* - Los plugins (llamadas '*extensiones'* en Joomla!) son incontables y gratuitos, de código abierto.
  - *Variedad de extensiones* – Las extensiones se dividen en 5 categorías que poseen diferentes funciones.
  - *Grandes capacidades de gestión de contenidos* – Fue diseñado inicialmente cómo un **CMS** empresarial, haciéndolo más capaz de manejar un gran volumen de artículos que WordPress.
- **Inconvenientes**
  - *Necesita de aprendizaje*
  - *Falta de capacidades **SEO*** – Se necesitan capacidades a nivel de experto para trabajar con motores de búsqueda.
  - *Soporte ACL limitado* – El soporte de ACL para la versión de Joomla! v2.5.1, sigue siendo limitado, no apto para clientes empresariales.

WordPress

- **Ventajas**
  - *Múltiples autores*
  - *Gran librería de Plugins* – Hay cientos de miles de plugins con infinidad de posibilidades.
  - *Amigable con el usuario* – La interfaz de uso es muy fácil de usar e intuitiva.
  - *Grandes capacidades **SEO*** – Con ciertos plugins se puede empezar a bloguear directamente sin preocuparse del **SEO** de la página.
  - *Fácil personalización* – El sistema de temas de WordPress está diseñado para una fácil personalización.
  - *Flexibilidad* – Se puede hacer que WordPress haga casi cualquier cosa.
- **Inconvenientes**
  - *Seguridad* – WordPress es a menudo diana de hackers y no es muy seguro.

- *Incompatibilidad con plugins más antiguos* – Se sacan continuas actualizaciones sin arreglar problemas o agujeros de seguridad de las anteriores y a menudo son incompatibles con plugins más antiguos.
- *Capacidades de gestión de contenido limitadas* – WordPress fue originalmente diseñado como plataforma de blogueo, por lo que esto ha afectado a su capacidad para manejar grandes cantidades de contenido.

Como vemos, a pesar de que WordPress, Joomla y Drupal están construidos con la misma pila de tecnologías, varían considerablemente en características y capacidades; hay características de unos que otros no poseen y viceversa. Podríamos seguir con el estudio, debate y comparación entre estos tres **CMS** cómo para llenar todo el presente documento, sin embargo no nos vamos a extender más en este tema. Hay varias páginas que nos proporcionan muchos datos relevantes acerca de estos sistemas. Por ejemplo, dos que queremos mencionar y que proporcionan una gran cantidad de información comparativa son: **Built With** y **CMS Matrix.** La primera nos ofrece información estadística acerca del uso de distintos CMS (tanto actual cómo pasada). La segunda página nos muestra información más técnica y específica referente a las capacidades que posee cada uno de los sistemas de gestión de contenidos.

En nuestro caso particular, para el presente proyecto, vamos a centrarnos en el estudio de Joomla!, ya que es un sistema de gestión de contenidos que consideremos en apogeo, con una buena documentación, gran flexibilidad y suficientes posibilidades para el correcto desarrollo del proyecto. Cómo hemos mencionado, se puede situar en un punto intermedio entre Drupal y WordPress y su uso es muy extendido, por lo que cumple con las metas que se han propuesto y que este proyecto requiere.

5.1.1- Joomla!

Joomla! es un sistema de gestión de contenidos (en inglés **CMS** – *Content Management System*) que permite desarrollar sitios web dinámicos e interactivos. Permite crear, modificar o eliminar contenido de un sitio web, de manera sencilla, a través de un panel de administración. Joomla! es un software de código abierto, está desarrollado en **PHP** (*PHP Hypertext Pre-procesor*) y liberado bajo licencia **GPL** (*General Public License*). Para su funcionamiento requiere una base de datos creada con un gestor de bases de datos (típicamente MySQL), así como de un servidor HTTP Apache. La primera versión de Joomla! fue publicada en el año 2005 y fue una evolución mejorada de *Mambo*, combinada con modificaciones de seguridad y anti-bugs. Joomla! está desarrollado en una arquitectura **MVC** (Modelo Vista Controlador), que nos dará características cómo la de interactuar directamente con la parte visual de Joomla!, un desarrollo de componentes módulos y plugins basados en la arquitectura base del **CMS**, actualizaciones rápidas en caliente…

Podemos ver las características principales de este sistema de gestión de contenidos a continuación:

- Hay muy buena documentación y soporte.

- Sigue el modelo **MVC**, lo que permite interactuar directamente con la parte vista de Joomla!, un desarrollo de componentes módulos y plugins basados en la arquitectura base del CMS, actualización de los elementos para cambios de versión con la plataforma funcionando.
- Excelente variedad de herramientas de personalización de contenidos, además de una gran disposición de módulos modificables. Las posibilidades de ampliación de Joomla! no tiene límites.
- Limpieza y simplicidad en el desarrollo y administración: Se pueden crear páginas ilimitadas y editarlas desde un sencillo editor, quedando los contenidos de dichas páginas organizados eficientemente en secciones y categorías. Además permite una cómoda administración, incluyendo la propia administración de usuarios, imágenes, módulos…
- Gran posibilidad de internacionalización del sitio web. Posee un **UTF-8** completo, soporte **RTL**, traducción de extensiones utilizando archivos **INI**.

Como vemos, las características que nos aporta Joomla!, también cumplen con la base establecida para la elección de la herramientas de desarrollo. Y además, también utiliza la arquitectura **MVC** (la cual consideramos cómo arquitectura principal en el desarrollo web), por lo que, sin duda, es un sistema de desarrollo a tener muy en cuenta. Como guinda al estudio de Joomla!, comentar que toda la documentación consultada referente a este **CMS** destaca la simplicidad en el desarrollo y la administración de los portales Joomla!, así como la limpieza a la hora de visualizar el sitio para el usuario. Dos puntos muy atractivos para la elección de la herramienta de desarrollo.

5.1.2- Ruby on Rails

Ruby on Rails es un framework de aplicaciones web de código abierto optimizado para la satisfacción de los programadores y para la productividad sostenible. Está escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (**MVC**). Rails fue creado en el año 2003 – Aunque la versión pública no salió hasta el 2004 – por David Heinemeier Hansson y busca combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real utilizando un mínimo de configuración y escribiendo menos código que con otros frameworks existentes. Rails hace uso de la metaprogramación que el lenguaje de programación Ruby permite, lo que resulta en una sintaxis que muchos de los usuarios del mismo encuentran muy legible. Se utiliza básicamente en la construcción de aplicaciones modernas de internet que utilizan bases de datos: Twitter, Github, Scribd, Hulu…

Las principales características que presenta este framework son las siguientes:

- Incluye soporte para:
  - AJAX (Ajax on Rails)
  - Servicios Web (Action Web Service)
  - Patrón de diseño Modelo Vista Controlador
  - Mapeo automático de objetos a modelo relacional

  - **RJS** (*Ruby to JavaScript compiler*)
- Sigue la filosofía de desarrollo **DRY** – **COC**:
  - "*Don't Repeat Yourself*" (**DRY**)
    - Cuyo significado incluye que las definiciones deberían hacerse una sola vez. Sigue un patrón de diseño Active Record, en el que, entre otras cosas, las definiciones de las clases no necesitan especificar los nombres de las columnas.
  - "*Convention Over Configuration*" (**COC**)
    - **COC** nos indica que el programador sólo necesita definir aquella configuración que no es convencional. Por ejemplo, en Ruby, una clase *User* en el modelo, corresponde con la tabla *users* de la base de datos.
- Rails incluye el concepto de migraciones, que nos permite definir o modificar la estructura de nuestra base de datos desde Ruby, y además de una manera independiente del motor de base de datos elegido.
- Sigue el paradigma de la arquitectura Modelo Vista Controlador (**MVC**), con el que se separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar las comunicaciones.

Como podemos ver en las características principales, este framework parece cumplir con los requisitos establecidos que se mencionaron en el comienzo de esta misma sección. Por otra parte tenemos el valor añadido de que trabaja mediante el modelo **MVC**, facilitándonos el mantenimiento de la aplicación. Además se considera un framework web muy productivo debido a su elegancia y acompañado con la simpleza de Ruby.

5.1.3- Django

Django es un framework de código abierto, escrito en Python, para el desarrollo web de alto nivel. Este framework, al igual que Ruby On Rails, respeta el paradigma **MVC** (Modelo Vista Controlador), pero con algunas salvedades, ya que la vista describe "qué" datos serán representados y no "cómo" se verán éstos; para ello se utilizarán los templates. Podríamos estar hablando de un framework "**MTV**" (Modelo Template Vista).

Django fue desarrollado por Adrian Holovaty, Simon Willison, Jacob Kaplan-Moss y Wilson Miner mientras trabajaban en World Online, y originalmente se utilizó para administrar tres sitios web de noticias: The Lawrence Journal-World, lawrence.com y KUsports.com. Fue expuesto al público bajo licencia **BSD** (*Berkeley Software Distribution*) en el 2005. Su nombre Django viene haciendo alusión al guitarrista de jazz Django Reinhardt.

Django es muy utilizado en la administración de páginas de noticias, ya que proporciona una serie de características que facilitan el desarrollo rápido de páginas orientadas a contenidos. Por poner un ejemplo, en lugar de obligar a que los desarrolladores escriban controladores y vistas para las áreas de administración de la página, Django proporciona una aplicación incorporada para la administración de los contenidos, que puede incluirse como parte de cualquier página hecha con Django y permite administrar varias

páginas desarrolladas con Django a partir de una misma instalación. Esta aplicación permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las accionas realizadas sobra cada uno, y proporciona una interfaz de administración de usuarios y grupos de usuarios.

Las características principales de Django las listamos a continuación:

- La principal función de Django es facilitar la creación de sitios webs complejos. Pone especial atención en la reutilización, la conectividad y la extensibilidad de componentes, el desarrollo rápido y el principio de **DRY** (*Don't Repeat Yourself*).
- Sigue el paradigma **MVC**, del que hemos hablado en las anteriores secciones. Sin embargo no es un modelo-vista-controlador "puro", ya que el controlador es llamado vista y la vista es el *template*.
- La distribución principal de Django también aglutina aplicaciones que proporcionan un sistema de comentarios, herramientas para sindicar contenido vía **RSS** (*Really Simple Syndication*) y/o Atom, "páginas planas" que nos permiten gestionar páginas de contenidos sin necesidad de escribir controladores o vistas para esas páginas y un sistema de redirección de URLs.
- Un mapeador objeto-relacional.
- Una API de base de datos robusta.
- Un sistema extensible de plantillas basado en etiquetas, con herencia de plantillas y soporte de internacionalización, incluyendo traducciones incorporadas de la interfaz de administración.

Vista la descripción general de en qué consiste Django, y qué características principales tiene, llegamos a la conclusión de que es bastante parecido a los otros dos frameworks analizados, con la peculiaridad de que este framework, por lo que parece, está más orientado a la creación de páginas de noticias que a otra cosa. Esta particularidad no nos incomoda en mayor grado, ya que nuestra aplicación se puede enfocar como un portal de noticias añadiendo mayor funcionalidades que simplemente una web de noticias. Es por ello que Django tiene que ser otro framework a ser considerado a la hora de desarrollar una aplicación web.

5.1.4- CakePHP

CakePHP es otro framework de desarrollo rápido de aplicaciones web en PHP, y está creado sobre los conceptos de Ruby on Rails.

Al igual que el resto de herramientas de desarrollo vistas en este documento, es libre, de código abierto, y busca permitir al desarrollador trabajar de forma estructurada y rápida sin pérdida de flexibilidad.

CakePHP empezó en el 2005, cuándo Ruby On Rails estaba ganando popularidad y utiliza muchos de sus conceptos. En este año Michal Tatarynowicz escribió una mínima versión del framework Rapid Application Framework en PHP. Desde ese momento supo que era el comienzo de un framework muy bueno, así que, bajo licencia **MIT**, publicó el framework

llamándolo Cake y lo abrió a la comunidad de desarrolladores, quienes ahora lo mantienen bajo el nombre de CakePHP.

Como el resto de herramientas vistas anteriormente, CakePHP utiliza una arquitectura Modelo, Vista, Controlador (**MVC**), pero con la particularidad de que incluye otras clases y objetos que hacen que el desarrollo en **MVC** sea un poco más rápido y agradable: Componentes (*Components*), comportamientos (*Behaviors*), y ayudantes (*Helpers*), clases que proporcionan extensibilidad y reusabilidad, agregando rápidamente funcionalidad a las clases base **MVC** de las aplicaciones.

Entre las particularidades de CakePHP podemos destacar las siguientes características:

- Cómo hemos mencionado, CakePHP utiliza el esquema Modelo-Vista-Controlador (**MVC**) para estructurar sus proyectos.
- Facilita el uso de la base de datos mediante el uso de ActiveRecord, al igual que Rails y tiene el **CRUD** (*Create, Read, Update and Delete*) de la base de datos integrado.
- Es compatible con **PHP4** y **PHP5**, y posee un sistema de plantillas rápido y flexible.
- Ayudas para AJAX, Javascript, HTML, forms y más herramientas, además de poseer componentes de seguridad y para control de sesión.
- Posee soporte de aplicación (*scaffolding*) y funciona en cualquier subdirectorio del sitio web, con poca o ninguna configuración de Apache

Por lo que se ha estudiado, CakePHP es un framework bastante similar a Ruby on Rails pero utilizando PHP en lugar de Ruby. En general parece que tiene una comunidad activa y amistosa y una buena aceptación del framework por parte de la misma. Se recomienda usar si nos gusta Ruby On Rails. En nuestro caso en particular, tenemos interés en Ruby On Rails, y aunque no sabemos Rails y sabemos PHP, nos encontramos en un entorno de aprendizaje que lo que se busca es estudiar y conocer nuevas metas y herramientas. Por ello, tal vez, antes que CakePHP, seleccionaríamos Ruby On Rails para el desarrollo del proyecto, por el principal motivo de la aportación que nos puede proporcionar en el sentido de aprendizaje.

5.2- Plataformas de control de la nube

Para evitar futuros problemas de servidores con recursos estáticos, como el sobredimensionamiento, el crecimiento inesperado de usuarios, la necesidad de inversiones extras en servidores o problemas derivados de la utilización de dichos servidores, se decidió utilizar la nube (en el apartado 9 sección 2.2 de este mismo documento se detallará el porqué). Para la correcta utilización de la nube, necesitaremos una plataforma que nos permita controlas las posibilidades que nos ofrece la misma.

Las características claves que buscamos para la selección de una plataforma de control de la nube son, en parte, similares a las que buscábamos en las herramientas de desarrollo:

- Ser de fácil utilización y en parte intuitiva para el uso del administrador.
- Que sean de uso extendido, compatibles, y con una suficiente documentación.
- Que las plataformas sean seguras y fiables.

- Que nos proporcionen servicios de aprovisionamiento, almacenamiento de datos, y de soporte.
- Ser gratuitas, competitivas y transparentes, permitiendo buen control sobre las herramientas que nos ofrecen.
- Soportar los lenguajes que se vayan a escoger para programación.
- Que presenten una modalidad de plataforma como servicio (**Paas**).

Veremos a continuación las distintas plataformas que hemos tenido en cuenta para ser estudiadas, con sus factores claves y las características que definen a cada una de ellas.

5.2.1- Heroku

Heroku es un servicio de Hosting en la nube (los clientes no tienen contar con la infraestructura; el tiempo de procesamiento y el almacenamiento se le renta a un tercero), que soporta varios lenguajes. Es propiedad de Salesforce.com y es una de las primeras plataformas en la nube. Fue desarrollada en el 2.007, con el objetivo de soportar solamente el lenguaje de programación Ruby. Posteriormente se ha extendido a Java, Node.js, Python, Clojure, Scala y otros.

Las aplicaciones se corren desde un servidor usando Heroku DNS Server para apuntar al dominio de la aplicación (generalmente *nombreaplicacion.herokuapp.com*). Cada aplicación corre sobre un motor a través de una red de bancos de pruebas que consta de varios servidores. El servidor Git de Heroku maneja los repositorios de las aplicaciones subidas por los usuarios.

Heroku se basa en los llamados *Dynos*, piezas fundamentales del modelo de arquitectura de éste. Proveen capacidad de cómputo dentro de la plataforma y están basados en contenedores de Linux.

Cada uno de estos *Dynos* está aislado del resto, por los que los comandos que se ejecutan y los archivos que se almacenan en un *Dyno* no afectan al resto. Además cada *Dyno* provee el ambiente requerido por las aplicaciones para ser ejecutadas.

Las características que sobresalen de esta plataforma, y que son llamativas a la hora de la elección, son las siguientes:

- Heroku es una plataforma como servicio.
- Elasticidad y crecimiento, ya que la cantidad de *Dynos* asignados a una aplicación puede cambiar en cualquier momento.
- Es gratuita (hasta 5 MB de espacio en disco para base de datos, y 50 MB para todos los archivos, incluyendo repositorios **GIT**).
- Su instalación es simple a través de una gema.
- El flujo de Heroku está basado en **GIT**. Se utiliza para hacer la instalación de la aplicación a través de repositorios.
- Su uso es simple e intuitivo, ya que lo que único que hay que hacer es un "*push*" de nuestro código a nuestro repositorio en Heroku. Al hacerlo se compila el *slug* de nuestra aplicación.

Algunos de los puntos negativos de Heroku, respecto a su versión gratuita, vienen referidos al tiempo de respuesta de la primera petición de página. Además no tiene acceso **SSH** y su sistema de archivos, debido a consecuencias de la escalabilidad, es readonly. Para las aplicaciones grandes su costo se eleva.

Hemos visto que entre las características principales destacan su sencillez y facilidad de uso. Además es compatible con Ruby on Rails y con un desarrollo basado en él, por lo que es muy buena opción para nosotros. Los puntos negativos no son algo que nos preocupen en exceso, y punto a parte, es gratuito para aplicaciones pequeñas.

5.2.2- Windos Azure

Windows Azure, al igual que Heroku, es una plataforma ofrecida como servicio y, en este caso, Azure está alojada en los Data Centers de Microsoft. Sus inicios se remontan a cuándo se anunció en el Professional Developers Conference de Microsoft (PDC) del 2008 en su versión beta, fase que pasó a comerciable el 1 de enero del 2009.

Windos Azure es una plataforma general que tiene diferentes servicios para aplicaciones, desde servicios que alojan aplicaciones en algunos de los centros de procesamiento de datos de Microsoft para que se ejecute sobre su estructura (Cloud Computing), hasta servicios de comunicación segura y federación entre aplicaciones.

Esta plataforma utiliza un sistema operativo especializado, llamado de la misma forma, para correr sus capas. Azure se describe como una “capa en la nube” funcionando sobre un número de sistemas que utilizan Windos Server, que a su vez funcionan bajo la versión 2008 del Windows Server y una versión customizada de Hyper-V. La capa controladora de Windows Azure se encarga de escalar y de manejar la confiabilidad del sistema, evitando así que los servicios se detengan si alguno de los servidores de datos tiene problemas.

Además, dado que la tecnología puede fallar, Windows ofrece una manera de proteger la información importante con una copia de seguridad automática dentro de un servicio de almacenamiento. Estas copias quedan cifradas antes de la transmisión y se almacenan cifradas en Windows Azure.

Dentro de la plataforma, el servicios de Windows Azure es el encargado de proporcionar el alojamiento de las aplicaciones y el almacenamiento no relacional. Estas aplicaciones deben funcionar sobre Windows Server 2008 y pueden estar desarrolladas en .**NET**, **PHP**, **C++**, **Ruby** y **Java**.

Las principales características que nos proporciona Windows Azure son las siguientes:

- El servicio de proceso de Windos Azure ejecuta aplicaciones basadas en Windows Server. Las aplicaciones se pueden crear utilizando **.NET** en **C#** o Visual Basic, o implementar sin **.NET** en **C++,** Java y otros lenguajes.

- Para el almacenamiento se utilizan objetos binarios grandes (blobs) que ofrecen un tipo de tablas con un lenguaje de consulta simple.
- La infraestructura nos permite desplegar de forma sencilla máquinas virtuales con Windows server o con distribuciones de Linux.
- Utiliza Active Directory para la administración de identidad y acceso, permitiendo gestionar de forma centralizada y sencilla el control de acceso e identidad.
- Windows Azure se ejecuta en una gran cantidad de máquinas combinadas en un solo centro de datos de Windows Azure, formando un conjunto armónico.
- La conectividad permite a las organizaciones interactuar con aplicaciones en la nube cómo si estuvieran dentro del propio firewall de la organización.

Como vemos, nos ofrecen unas muy tentativas características para su uso. La infraestructura, la facilidad para el control de acceso, el que no esté centralizado en una única máquina… Además ofrece una gran versatilidad a la hora de hospedaje para sitios web. Nos permite hospedar la aplicación en los sitios web de Azure, en la nube, o máquinas virtuales. Otro de los puntos fuertes es su documentación, ya que en su propio sitio web podemos encontrar mucho material y bien explicado.

Sin embargo, un inconveniente que tiene esta plataforma es que su servicio es de pago. Ofrece una versión trial gratuita de hasta 150€ en utilización de servicios de Azure, pero los recursos que nos ofrece para esta versión son bastante limitados, de modo que, por ejemplo, sólo tenemos un mes de uso gratis con sus máquinas. Punto altamente negativo para los intereses de nuestro proyecto, ya que buscamos algo no tan restrictivo en cuanto al tiempo de uso.

5.2.3- OpenShift

Openshift es un producto de Red Hat para computación en la nube de plataforma como servicio que automatiza el aprovisionamiento, la gestión y el escalado de aplicaciones para que el desarrollador se pueda centrar en el código para su negocio, la puesta en marcha, o la próxima idea de negocio. Este software funciona como un servicio de código abierto bajo el nombre de “OpenShift Origin”, y está disponible en *GitHub*.

Los desarrolladores pueden utilizar **Git** para desplegar sus aplicaciones Web en los diferentes lenguajes de la plataforma. También soporta programas binarios que sean aplicaciones web, con tal de que se puedan ejecutar en RHEL Linux, permitiendo con ello el uso de lenguajes arbitrarios y frameworks.

Una selección de lenguajes de programación como Java, Ruby, **PHP**, Node.js, Python y Perl y un completo set de herramientas para desarrolladores están disponibles dentro de OpenShift Online, para aumentar la productividad del desarrollador y acelerar la entrega de las aplicaciones. Un punto que realmente, aunque sea interesante, no es algo que nos vaya a beneficiar en nuestro proyecto.

OpenShift se encarga de mantener los servicios subyacentes a la aplicación y la escalabilidad de la aplicación tanto como se requiera.

Algunas de las características de las que presume OpenShift son:

- **Velocidad**: Reduce el tiempo necesario para construir y desplegar nuestras aplicaciones, permitiéndonos así centrarnos en nuestro código, en lugar de la administración de la infraestructura.
- **Versatilidad**: Ofrece la más amplia gama de lenguajes de programación, frameworks y tiempos de ejecución, incluyendo Java EE6 con JBoss EAP.
- **Código abierto**: Mediante su plataforma de código abierto nos ofrece asegurar la portabilidad de las aplicaciones y evitar el *lock-in*.
- **Fácil manejo**: Posee herramientas de desarrollo integradas y una interfaz intuitiva, permitiéndonos trabajar rápidamente.

Además, con su versión de pago nos ofrece características extras como capacidad de mayor almacenamiento, soporte de clases mundial, seguridad **SSL**, autoescalado…

El modelo de funcionamiento que OpenShift promociona se basa en lo siguiente:

- **Código:** Desarrollo del código utilizando el lenguaje y las herramientas que necesitemos y, a medida que avanzamos, subir los cambios incorporados a un repositorio **Git.**
- **Construir:** Cualquier paso necesario para transformar el código en una aplicación que se pueda ejecutar, puede ejecutarse en OpenShift; desde un simple script hasta un sistema de construcción externa, se puede preparar al código para su ejecución en la nube.
- **Despliegue:** Cada aplicación está compuesta por "*cartridges"* (trocitos independientes de la aplicación se apilan como servidores web y bases de datos que simplifican el mantenimiento y la configuración del servidor). Nosotros escogemos las tecnologías mediante la adición de los *cartridges*, y OpenShift nos aprovisiona los servicios automáticamente
- **Mantenimiento:** Cuando la aplicación está corriendo en la nube, podemos monitorizarla, depurarla y ajustarla sobre la marcha. Si el tráfico se vuelve mayor, se puede escalar de forma automática.

OpenShift parece una buena solución para subir nuestra aplicación a la nube: Sigue una visión de aplicación de código abierto, provee un muy bien servicio de forma gratuita y nos ofrece facilidad de uso y buena documentación para su utilización.

Estas son las tres plataformas de control de la nube que vamos a estudiar, ya que, aunque existen muchas más plataformas **PaaS**, nuestra elección va a estar guiada y decidida entre alguno de los tres casos estudiados.

Otras plataformas **PaaS** que podríamos tener en cuenta a la hora de la elección podría ser la propia de Google (Google APP Engine), la de Amazon (AWS), Force (SalesForce)… Sin embargo estas plataformas, aunque nos brindan un buen servicio, no son gratuitas o no proveen soporte para aplicaciones Rails.

5.3- Herramientas de automatización de prueba del software

Este proyecto está destinado a ser una aplicación que se desarrollará guiada por pruebas de aceptación (**ATDD**), y por ello necesitaremos una herramienta que nos permita la automatización de las mismas, obteniendo, de esta manera, un software probado y fiable en su fin. En las pruebas del software, la automatización de las pruebas consiste en comparar los resultados obtenidos con los resultados esperados, controlando la ejecución de las pruebas mediante un software específico. Nuestro sistema estará guiado por pruebas de aceptación, las cuales se pueden definir cómo un escenario de utilización del sistema y el comportamiento que de él se espera, visto desde la perspectiva del usuario. De una forma más concisa, una prueba de aceptación tiene como propósito demostrar al cliente el cumplimiento de un requisito del software. Dichas pruebas, tiene las siguientes características particulares:

- Cada una de ellas describe un escenario de ejecución o uso del sistema desde la perspectiva del cliente.
- Pueden estar asociadas a requisitos funcionales o no funcionales.
- Un requisito tiene una o más pruebas de aceptación asociadas.
- Una prueba de aceptación puede tener infinitas instanciaciones (ejecuciones con valores concretos). El diseño de las instanciaciones y su aplicación es trabajo del tester.

Además tendremos otros tipos de pruebas unitarias o *Unit Test*, que se encargarán de comprobar el correcto funcionamiento de distintos módulos de la aplicación.

El software de control de pruebas que querremos utilizar debe de tener una serie propiedades que podemos ver a continuación:

- Debe proporcionar soporte de testeo para la herramienta de desarrollo que se utilice.
- De manejo no complejo.
- Que se permita unas definiciones de pruebas que sean de una manera que el cliente pueda comprender, entender y evaluar.
- Disponible en un entorno UNIX y más concretamente para programación en Ruby.
- Que nos proporcione pruebas fiables.

La ventaja de utilizar este tipo de herramientas permite implicar en el mismo proceso continuo de aceptación y pruebas al propietario del producto o al analista que ha creado estas especificaciones con los desarrolladores. De este modo, es visible cada una de las especificaciones que se han definido sin tener que entrar en el código. Encaja muy bien en las metodologías ágiles, fomentando que el equipo trabaje sobre pruebas usando **BDD** y **TDD** a la par de aportar una documentación viva con la funcionalidad requerida por el equipo no especializado.

A continuación vamos a ver una serie de herramientas que nos permitirán cubrir los requisitos anteriormente nombrados y que podrán ser susceptibles a utilizar cómo herramientas de control para las pruebas del software.

5.3.1- Cucumber

Cucumber es una herramienta que permite ejecutar, de forma automatizada y por escrito, pruebas de aceptación en un desarrollo guiado por comportamiento. Cucumber está escrito en el lenguaje de programación Ruby, aunque hay adaptaciones para otros lenguajes. Permite escribir los escenarios de prueba en distintos idiomas y permite definir los mismos de una forma fácil de entender, en un lenguaje natural (Gherkin).

Sin embargo, Cucumber no se ha de considerar como una simple herramienta de pruebas, sino como un apoyo al **BDD** (Behaviour-Driven Development). Esto significa que las pruebas (descripciones de característica en texto plano con escenarios) deben ser típicamente escritas antes de realizar cualquier otra cosa, y verificadas por los analistas, los expertos del dominio, etc. Una vez descritos los escenarios de prueba, el código de producción es escrito de forma independiente para satisfacer estos escenarios.

Como hemos nombrado, Cucumber está escrito en Ruby, pero podemos utilizarlo para validar código escrito en Ruby, Java, C#, Python… Solamente requiere un pequeño conocimiento en Ruby, por lo que aunque lo que se pretenda desarrollar no vaya a estar en Ruby, no se debe dejar de tener en cuenta esta herramienta de testeo.

Las características principales de Cucumber son las siguientes:

- Cucumber utiliza la sintaxis de Gherkin para definir las especificaciones.
- Es posible escribir los escenarios en distintos idiomas.
- Cucumber incorpora las etiquetas necesarias para relacionar los pasos con el código en muchos idiomas.
- Se evitan confusiones al transformar especificaciones en funcionalidades o tests unitarios.
- La estructura de definición de los tests es muy intuitiva y razonable.

Por otro lado, algunas de las desventajas que podemos señalar en esta herramienta son aspectos cómo que el que escribe las pruebas tiene que adaptarse a un formato predeterminado o que no posee soporte para Junit o TestNG.

Como podemos ver, Cucumber es una herramienta bastante completa y fiable para nuestros objetivos. Sopesando las ventajas contra las desventajas, se puede determinar que, para nuestro proyecto en particular, las primeras tienen mucho más peso que las segundas, ya que las desventajas no nos deberían suponer prácticamente ningún problema a la hora de realizar este proyecto. Además, el estar orientado a desarrollo **BDD**, nos permite cumplir con los requisitos que hemos definido de nuestro proyecto **ATDD**. Otro punto a favor es la facilidad que parece tener a la hora de que el cliente considere las pruebas que estamos realizando. Al utilizar Gherkin, la sintaxis a la hora de especificar las pruebas va a ser intuitiva y totalmente legible para el cliente (o usuario final), opción muy interesante a la hora de desarrollar.

5.3.2- RSpec

RSpec es una framework para realizar tests en un desarrollo basado en el comportamiento (**BDD**), y diseñada para programadores de Ruby. **BDD** es un acercamiento al desarrollo del software que combina un desarrollo basado en tests (**TDD** – Test-Driven Development) con un diseño guiado por el dominio (**DDD** – Domain Driven Design) y una planificación basada en test de aceptación (**ATD** Planning – Acceptance Test-Driven Planning). RSpec está pensado para ayudar al desarrollador a cumplir la parte del **TDD**, enfocándose en aspectos de documentación y diseño del **TDD**.

Esta plataforma fue creada por Steven Baker en el 2005. Steven se había enterado acerca de la metodología **BDD** gracias a Aslak Hellesoy, debido a que había estado trabajando en un proyecto con Dan North acerca de esto. Steven ya estaba interesado en la idea cuando Dave Astels sugirió que estas ideas de **BDD** podrían ser fácilmente implementadas en lenguajes cómo Smalltalk o Ruby. Steven realizó una implementación inicial y de este modo nació RSpec.

En la actualidad RSpec está considerado como un Lenguaje Específico de Dominio (Domain Specific Language - **DSL**) utilizado para describir el comportamiento esperado de un sistema mediante ejemplos ejecutables y, al igual que Cucumber, es bastante intuitivo y cómodo de utilizar ya que se parece a una especificación de lenguaje natural. Actualmente se encuentra en la versión 3.0.0 y dicha versión es totalmente estable.

Las características más sobresalientes de este framework son las siguientes:

- Es un programa con un amplio conjunto de comandos.
- Provee la capacidad para describir ejemplos ejecutables de cómo debería comportarse el código y herramientas para determinar qué ejemplos deben funcionar y cuáles deben ser sus resultados.
- RSpec proporciona informes flexibles, con opciones de personalizarlos.
- Permite expresar las salidas que se esperan de un objeto en forma de ejemplo.
- Posee generadores especiales para modelos y controladores que generan specs en lugar de tests.
- Disponible en entorno Linux para programadores de Ruby.
- Permite especificaciones de forma legible sin ser engorrosas de escribir y nos permite realizar la misma tarea de varias formas posibles.

Vistas las características más relevantes que nos ofrece RSpec, también debemos mencionar algunos de los aspectos negativos que se han podido encontrar en el estudio de este framework de testeo. Por lo que se ha podido estudiar, no es un framework recomendable para empezar desde cero con la metodología de testeo, ya que se menciona de que es algo complejo de aprender y dominar, aunque una vez se ha aprendido es bastante claro y limpio. También hay que tener en cuenta que está ideado para tests unitarios o *Unit Test* (aunque es posible su uso para tests de aceptación), al contrario que Cucumber que está orientado a tests de integración (o aceptación).

Por lo analizado, RSpec es otra herramienta de testeo para un desarrollo guiado por pruebas muy a tener en cuenta, ya que los pros de este framework superan con creces a los

contras. De hecho, lo peor que se ha podido estudiar acerca de RSpec es su complejidad a la hora de aprenderlo, algo con lo que podríamos coexistir si realmente la herramienta nos proporciona buenas funcionalidades.

5.3.3- MiniTest

En el mundo de de Ruby, Test::Unit y RSpec son las principales librerías de testeo utilizadas. Sin embargo, con lanzamiento de Ruby 1.9 y 2.0, Test::Unit fue reemplazado por **MiniTest**. El reemplazo ha sido totalmente transparente para quien usase Test::Unit, ya que MiniTest ofrece una capa de compatibilidad con el primero, por lo que si todavía pretendemos utilizar '*test/unit*', lo que realmente estamos usando por debajo es MiniTest::Unit.

MiniTest es una librería que provee una amplia gama de recursos para proporcionarnos facilidades en el testeo de **TDD**, **BDD**, *"mocking"* y "*benchmarking*". Además, las capacidades que nos aporta Minitest nos facilita realizar tests de forma clara e inteligible. En la gama que aporta nos podemos encontrar con Minitest/{unit, spec, mock, benchmark}.

MiniTest/Unit es un pequeño e increíblemente rápido framework de testeo unitario, el cual posee un rico conjunto de declaraciones para hacer las pruebas limpias y legibles.

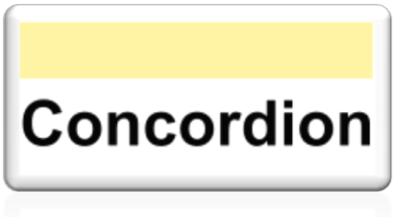
MiniTest/Spec es el motor de especificación funcional completo. Está enganchado a MiniTest/Unit.

MiniTest/Benchmark es una manera impresionante para comprobar el rendimiento del algoritmo mediante repeticiones. Con esto podemos comprobar que otra gente no remplace nuestro algoritmo lineal por uno exponencial.

Comparándolo con RSpec, MiniTest se jacta de que es, simplemente, Ruby, mientras que RSpec es un es un tester **DSL** (Domain Specific Language). Además, la velocidad de testeo de MiniTest generalmente supera a la de RSpec para un mismo test (aunque normalmente se ocupan más líneas de código).

Por otro lado, una de las cualidades que posee MiniTest es que viene integrado con las nuevas versiones de Ruby, por lo que no hay que instalar ninguna otra cosa. Por lo tanto, como puntos favorables podemos considerar la rapidez con que se puede testear, la no necesidad de aprender un lenguaje distinto a Ruby, y que no necesita de instalaciones extras. Sin embargo, por puntos en contra debemos tener en cuenta que tiene bastantes más limitaciones que RSpec a la hora de testear las aplicaciones y que la sintaxis de los test se realiza a un muy bajo nivel; pese a esto, lo suelen recomendar para nuevos proyecto en Rails y, sobre todo, para principiantes en desarrollos guiados por tests.

5.3.4- Concordion



Otro framework bastante utilizado y con buena aceptación en el mundo del desarrollo guiado por pruebas es el **Concordion**. Este framework nos permite construir y ejecutar test de aceptación. Aporta visibilidad a los requisitos definidos por el propietario del producto que el desarrollador ha plasmado en las pruebas de aceptación.

Es una herramienta de código abierto, y tiene versiones para Java, .NET, Pythin, Fantom, Scala y Ruby.

La filosofía de este producto es bastante simple: Se crean unas especificaciones en un documento HTML empleando lenguaje natural, posteriormente con el marcaje proporcionado por Concordion definimos los "*assert*" para instrumentar las pruebas y lo enganchamos con las pruebas JUnit. De esta forma, podemos separar el *qué* del *cómo*.

Cuando ejecutamos los tests genera un documento en el que se muestra en verde las especificaciones que se cumplen y en rojo las que han fallado.

Algunas de las características que proporciona este framework son:

- Ayuda a separar el *qué* del *cómo.*
- Conjunto de comandos bastante pequeño y sencillo de aprender
- Lenguaje normal, sin necesidad de estructura: *Dado/ Cuando/ Entonces.*
- Salida de datos vistosa.
- Los comportamientos complejos pueden descomponerse en secciones más simples para su tratamiento.

Concordion no es una herramienta que acabe de lanzarse, sino que lleva unos años gestándose. Quizás ahora sea el momento en el que las empresas que comienzan a tener una filosofía de desarrollo ágil utilizando BDD y TDD de forma cotidiana que dan visibilidad a las pruebas de código en formato amigable para los dueños del producto.

Otra de las utilidades en las que puede usarse Concordion es en la creación de una documentación viva unida a los cambios de código. Si nos planteamos la pregunta ¿Son los tests la auténtica documentación actualizada de nuestro código?, muchos desarrolladores responderían que sí. Por tanto… ¿Por qué no usarlos por medio de una fachada HTML cómo documentación?

5.3.5- Jasmine

Jasmine es un framework de BDD para comprobar código JavaScript. No depende de ningún otro framework de JavaScript, no requiere un **DOM**, y tiene una sintaxis muy limpia, por lo que, obviamente, permite escribir y leer pruebas fácilmente.

Como vemos, esta herramienta está limitada al testeo de código JavaScript, pero para conseguir un código totalmente testeado y libre de errores, debemos centrarnos también en revisar la parte del código que está implementada utilizan JavaScript.

Los inicios de este framework datan del 2001. Por estas fechas se utilizaba el "*Pivotal Labs' JsUnit Framework*" para testear aplicaciones con JavaScript y con el tiempo, este JsUnit, pasó a llamarse Jasmine.

Para la utilización de Jasmine lo único que es necesario es un navegador con JavaScript habilitado y una guía de utilización del framework.

A la hora de elegir un framework de testeo para JavaScript debemos tener en cuenta varias consideraciones. Jasmine es simplemente uno de los muchos participantes en el mundo del testeo de aplicaciones con JavaScript. Algunos utilizan *qUnit*, framework no oficial de testeo para *JQuery*, o YUI Test, el cual es parte de la librería YUI de Yahoo. De hecho, la mayoría de las librerías JavaScript tienen su propio framework de testeo. Estos deberían ser considerados si vamos a ser parciales a una biblioteca específica JavaScript. Por otro lado, podríamos querer tener clases de prueba que no estén vinculadas a otro framework. Esto es algo importante en Jasmine. Otra consideración a tener en cuenta es que Jasmine se basa en el navegador o requiere la utilización de un entorno de pruebas de software basadas en la web, como *Selenium*. Hay otros como *RhinoUnit* que no requieren un navegador para funcionar, haciendo las pruebas mucho más automatizadas.

En nuestro caso en particular, hemos querido referenciar Jasmine como aplicación para pruebas de JavaScript porque es un framework bastante extendido y con pocas limitaciones a la hora de comprobar código. Además, su sintaxis a la hora de desarrollar las pruebas es bastante limpia, punto que añade motivación a la hora de aprender su manejo.

5.4- Elección de herramientas

Una vez estudiada la diversidad de recursos que existen para la realización del proyecto, vamos a indicar el motivo de la elección de cada herramienta y lo que nos va a aportar en el desarrollo de la aplicación.

5.4.1- Elección de herramienta de desarrollo

Como hemos mencionado anteriormente (y el título del proyecto deja claro), hemos elegido una programación basada en Ruby on Rails. Hay diversos motivos para esta decisión, y podríamos empezar diciendo que uno de ellos es porque es un estilo de programación muy demandado en el mercado actual. Nótese que hablamos de estilo de programación porque realmente hay un mundo filosófico **RoR**: El **DRY**, la claridad del código, la simpleza, el estilo… Son un conjunto de cosas que hacen de este framework, algo más que un simple framework. Promueve las buenas prácticas, dándonos facilidades para evitar escribir código engorroso o ilegible, y facilitándonos el acceso al mundo del **DRY**. Las gemas también ayudan a no tener que escribir código de más y centrarnos en lo que queremos desarrollar.

Además hay un montón de características que hacen de Ruby on Rails una buena elección: es de uso muy extendido, y cada vez tiene un mercado más amplio; aparte es fácil de aprender, utilizar y es libre. Otro punto a favor es que utiliza **ActiveRecord**, lo que nos simplifica el manejo de las bases de datos. Para otros frameworks es necesario aprender hasta 3 lenguajes de programación (por ejemplo Django, que requiere de Python, de un lenguaje de plantillas y otro de manejo de bases de datos). Con Rails basta prácticamente con saber Ruby y ActiveRecord, el cual nos permite manipular las bases de datos de una forma simple.

Por otro lado no debemos olvidarnos que queremos un proyecto basado en pruebas, por lo que también hay que mirar la compatibilidad con herramientas que permitan realizar

tests. Rails sigue una metodología de desarrollo guiado por tests, ya que el mundo RoR se guía bajo la metodología del testeo.

Otro punto a tener en cuenta es que la comunidad Rails es muy grande y hay muchos libros de aprendizaje y experimentación al respecto. Existen multitud de foros y sitios web dónde aclaran dudas y enseñan el cómo y el porqué de los errores que puedan surgir a la hora de desarrollar.

Si a todo esto le sumamos las características técnicas y demás funcionalidades que vimos anteriormente en la descripción de la herramienta, no debemos pasar por alto la oportunidad de aprender en un proyecto de final de carrera esta herramienta, ya que sin duda alguna nos abrirá muchas puertas en el mercado laboral y nos permitirá obtener una buena experiencia en el mundo de las buenas prácticas de la programación.

5.4.2- Elección de plataforma de control de la nube

En nuestro proyecto en particular, la plataforma que más se adecúa a nuestro plan es Heroku. Heroku, como hemos mencionado anteriormente, nos provee un servicio de Hosting en la nube, sin tener que preocuparnos de la infraestructura, el tiempo de procesamiento y el almacenamiento. Además lo hace de forma gratuita (5 MB de espacio para BD y 500MB para todos los archivos, y es un servicio basado en la nube de **Amazon Web Services**.

Soporta y está muy bien integrado con Ruby on Rails, por lo que a nosotros nos resulta perfecto en ese sentido, ya que implementaremos nuestra aplicación con él. La implementación se hace con **Git** (recurso que también utilizaremos en nuestro desarrollo), y se instala a través de una "*gem*", todo de una manera muy intuitiva.

Otro punto a favor es el sencillo manejo que ofrece frente a cualquier aplicación y la simpleza con la que se puede empezar a utilizar. Simplemente registrándote, teniendo Rails y Git instalado junto con su gema en la aplicación y creando las llaves SSH podemos empezar a crear nuestra aplicación en Heroku. Todo se controla desde un simple terminal y se puede desplegar el código en Heroku con una única sentencia.

De las tres plataformas que hemos mostrado, la que más podría pesar en nuestra elección frente a Heroku es **OpenShift**, ya que Windows **Azure** realmente no nos proporciona los servicios que requerimos para este proyecto (sobre todo la importancia de que es de pago transcurridos 30 días). La elección de Heroku viene determinada principalmente por recomendación de la bibliografía estudiada y del consejo de expertos en el tema, añadiendo a esto las razones nombradas y las especificaciones detalladas en su descripción.

5.4.3- Elección de herramientas de pruebas

Este punto no va a estar determinado por una única herramienta, sino que vamos a tener que utilizar varias herramientas de testeo, cada una para su función específica.

Entre **Concordion** y **Cucumber** utilizaremos el segundo para los tests de aceptación que el cliente tendrá que identificar y validar. La sintaxis de **Cucumber** es perfecta para hacer tests de aceptación para el cliente, ya que son totalmente inteligibles y nos servirá, además, cómo herramienta de validación para los casos de uso. Realmente Cucumber proporciona una

forma especial, mediante código, de describir los casos de uso presentados en este mismo documento. De este modo, en un caso ideal, se implementaría en los archivos con extensión "*.feature*" lo que la aplicación debería realizar y se le presentaría al cliente para que este valide que realmente es lo que se desea obtener (en nuestro caso en particular el cliente es el propio tutor).

Además, **Cucumber** también ayudará a chequear el correcto funcionamiento del código, ya que mediante los "*steps"* testearemos que todo nuestra implementación va a realizar la función que realmente queremos que realice, preocupándonos en que se obtengan los resultados que esperamos conseguir al realizar ciertas acciones.

Por otro lado vamos a utilizar **RSpec** para la validación del fichero semilla de nuestra aplicación (fichero en el que se guardarán los datos iniciales que el sistema debe poseer al iniciarse por primera vez). El conocer cómo se utiliza **RSpec** se considera importante en la actualizad para desarrollos guiados por pruebas, por lo que es interesante que el alumno conozca esta otra herramienta para pruebas de aplicaciones.

En este caso, lo que vamos a comprobar es que el fichero semilla está correctamente elaborado y definido, y en el caso de que haya alguna falta, las pruebas RSpec nos lo indiquen.

Hemos elegido **RSpec** frente a **MiniTest**, porque se considera que el conocimiento de realizar tests con **RSpec** tiene más salida en la actualidad, y aunque es interesante aprender a realizar tests con **MiniTest**, la sintaxis entre ambos es muy parecida y hay que acotar en cierta medida el estudio e inclusión de nuevas tecnologías en el proyecto.

Caso muy similar ocurre con **Jasmine**. **Jasmine** es una herramienta de testeo que en un principio estaba planeada utilizarse en el proyecto para testear el funcionamiento de la parte de JavaScript de la aplicación. Sin embargo se ha invertido mucho tiempo en el estudio de todas las demás herramientas utilizadas en el proyecto, ya que se desconocía gran parte de la tecnología (así como la metodología) utilizada el proyecto. Por lo tanto, vamos a proponer la inclusión de tests con Jasmine como una mejora del proyecto y el estudio de esta herramienta como una formación extra fuera de los marcos del proyecto.

6- Metodología de desarrollo

Cómo hemos nombrado varias veces a lo largo del documento, nuestro proyecto va a estar guiado por una metodología **ATDD**, acrónimo de **A**cceptance **T**est **D**riven **D**evelpment – Desarrollo guiado por pruebas de aceptación. En este apartado vamos a empezar estudiando esta metodología, describiendo sus origines y estudiando de forma exhaustiva su forma de emplearla. El motivo de elección de este tipo de metodología es por una de las bases de la filosofía Rails, en la que se establece que todo debe estar correctamente testeado.

Este proyecto va a estar desarrollado bajo una metodología de desarrollo ágil, concretamente nos vamos a centrar un desarrollo guiado por pruebas de aceptación. Hemos mencionado numerosas veces durante el documento que vamos a seguir unas pautas de desarrollo guiado por pruebas y las siglas del "**ATDD**", pero… ¿Qué significa realmente un desarrollo guiado por pruebas?

En términos generales, las pruebas de aceptación del software o desarrollo son un conjunto de pruebas funcionales basadas en metodologías ágiles que se realizan sobre el sistema completo; todos los programas tienen errores y la **ATDD** los descubre. Estas pruebas permiten al cliente u usuario final validar los requisitos, y hay dos tipos de pruebas las pruebas formales, y las informales. Antes de estudiar en más detalle la metodología ATDD vamos a mirar unos escalones por arriba y vamos a estudiar sus orígenes.

6.1- Programación extrema (XP – eXtreme Programming)

Para hablar de los orígenes del desarrollo guiado por pruebas de aceptación debemos empezar hablando de la **metodología XP –** e**X**treme **P**rogramming. Esta metodología de desarrollo de la ingeniería del software, formulada por Ken Benk, es la más destacada de los procesos ágiles de desarrollo del software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradiciones principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de esta programación consideran que los cambios de requisitos sobre la marcha son un aspecto natura, inevitable e incluso deseable dentro de los proyectos. Es necesario ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto, siendo una forma más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

A continuación vamos a ver un diagrama representativo del ciclo de vida del software según **XP,** en el que se muestra que hay una fase de pruebas, planificación, diseño y codificación, y por las cuales se va iterando hasta conseguir el producto final:

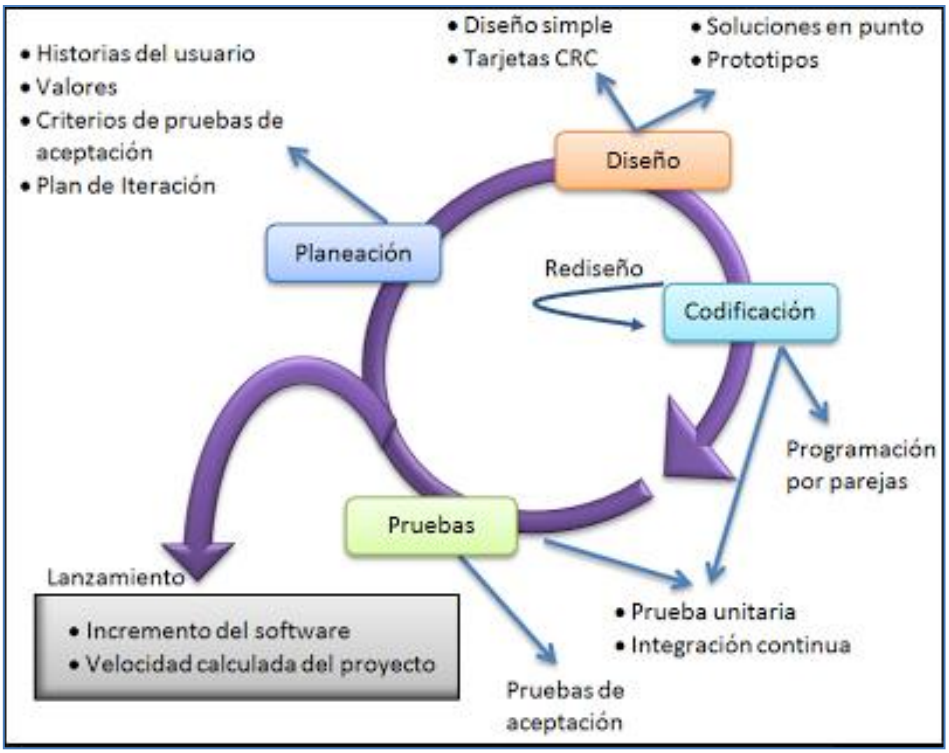


Imagen 6.1.1 – Diagrama **eXtreme Programming**

En el diagrama podemos observar los principios de la filosofía de desarrollo ágil. No es un proceso de desarrollo lineal en el que existe un principio y un fin, como en las metodologías tradicionales, sino que es algo incremental con realimentación, en el que se van realizando iteraciones de desarrollo de acuerdo con una serie de pautas establecidas. Al terminar cada iteración se vuelve con una nueva iteración de otra funcionalidad del software, y así continuamente de forma incremental hasta que el desarrollo se termina y se realiza el lanzamiento del producto.

Los principios básicos que persigue este tipo de programación son los siguientes:

- **Comunicación** – Mediante el código bien escrito, entre programadores, con el cliente…
- **Sencillez** – En el diseño, en la documentación, en el código…
- **Retroalimentación** – Gracias a la estrecha cooperación con el cliente e iteraciones cortas.
- **Valentía** – Para adoptar simplicidad y recodificación cuándo haga falta, programación por parejas e iteraciones cortas.
- **Respeto** – Entre todos los miembros del equipo.

Y la práctica bien definida de esta "nueva" metodología viene determinada por un modelo de **planificación incremental**, **entregas pequeñas**, **metáforas**, **diseño sencillo**, **pruebas continuas**, **refactorización**, **integración continua**… Una serie de condiciones que hacen que las metodologías tradicionales se conviertan en un nuevo tipo de metodología de desarrollo, la denominada metodología ágil (y concretamente, en este caso, la e**X**treme **P**rogramming).

El desarrollo con **XP** puede ir dirigido por varios caminos y en uno de ellos nos encontramos con la disciplina **TDD** – **T**est **D**riven **D**evelopment. La que nos interesa y los orígenes de nuestra metodología **ATDD**.

6.2- Metodología TDD

TDD es una práctica de la ingeniería del software que utiliza los principios de la programación ágil (particularmente los de **XP**) para involucrar dos prácticas en proyectos desarrollados bajo su filosofía: *Escribir las pruebas primero*, y la *refactorización*. Para escribir estas pruebas, generalmente se utilizan los tests unitarios, y el modelo que se persigue es: En primer lugar escribir las pruebas y verificar que éstas fallan. A continuación implementar el código que hace que la prueba pase satisfactoriamente y seguidamente refactorizar el código escrito. El propósito del desarrollo guiado por pruebas es lograr un código limpio que funcione, y su idea es que los requisitos sean traducidos a pruebas, de modo que cuando las pruebas pasen, garantizar que el software cumple con los requisitos que se han establecido.

Hay que resaltar la importancia de utilizar esta práctica en un desarrollo ágil, ya que, como hemos mencionado, un desarrollo ágil se caracteriza por pequeñas iteraciones y refactorizaciones y habrá momentos en los que haga falta cambiar alguna funcionalidad que ya está implementada y aceptada. Esto puede resultar muy desastroso si no tenemos alguna alerta que nos advierta de que algo que hemos tocado ha dejado de funcionar o ha empezado a funcionar mal; aquí está la importancia de los tests. Sin código de testeo que nos indique que lo que estamos haciendo (o modificando) nos produce los resultados que realmente queremos obtener, estaríamos muy perdidos y con mucho tiempo que invertir para averiguar en dónde se está produciendo el error. Sin embargo, supongamos que estamos en la iteración 12 de un proyecto que estamos realizando, y el cliente nos comunica que quiere hacer un cambio en una funcionalidad que hemos implementado respecto a los usuarios, que realizamos en la iteración 2. Con un código correctamente testeado, al modificar cualquier cosa de esta iteración tendremos la certeza de que el sistema se sigue comportando cómo queremos que se comporte, sin preocuparnos de que los nuevos cambios puedan afectar negativamente al mismo, y estando en la seguridad de que no vamos a tener que chequear las 10 iteraciones restantes.

Dicho esto, y una vez aclarada la importancia de las pruebas, vamos a continuar con nuestro estudio de la metodología de desarrollo.

Como mencionamos, **TDD** es una disciplina de testeo dentro de **XP** y se centra en tres pilares fundamentales:

- La implementación de las funciones justas que el cliente necesita, y no más.
- La minimización del número de defectos que llegan al software en fase de producción.
- La producción de software modular, altamente reutilizable y preparado para el cambio.

Con **TDD** se pasa de pensar en implementar tareas, a pensar en ejemplos certeros que eliminen la ambigüedad creada por el lenguaje natural. Hasta ahora estábamos

acostumbrados a que las tareas, o los casos de uso, eran las unidades de trabajo más pequeñas sobre las que ponerse a desarrollar código. Intentamos traducir el caso de uso o tarea en X ejemplos, hasta que el número de ejemplos sea suficiente como para describir la tarea sin lugar a malinterpretaciones de ningún tipo.

En otras metodologías del software, primero nos preocupamos de definir cómo va a ser nuestra arquitectura. Pensamos en las clases de infraestructura que van a homogenizar la forma de trabajar en todos y cada uno de los casos, pensamos si vamos a usar un patrón u otro, una comunicación mediante eventos, o **DTOs**, y una clase central que va a hacer esto y aquello. ¿Y si resulta que luego no necesitamos todo esto? ¿Cuánto vamos a tardar en darnos cuenta de que no es necesario? ¿Cuánto dinero se va a malgastar? En **TDD** dejamos que la propia implementación de pequeños ejemplos, en constantes iteraciones, haga emerger la arquitectura que va a utilizar. No es que se olvide completamente las características técnicas de la aplicación a priori, es decir, lógicamente tendremos que tener en cuenta si el desarrollo es para un teléfono móvil, para una web o para un PC; más que nada porque tenemos que escoger las herramientas para el desarrollo conformes las exigencias. Sin embargo nos limitamos a escoger el framework correspondiente y a usar su arquitectura como base y lo que eliminamos son las arquitecturas encima de esas arquitecturas, las que intentan que todo se haga siempre igual y tal cómo se le ocurrió al "genio" de la empresa. A ser posible, esas que nos obligan a modificar siete ficheros para cambiar una cadena de texto. **TDD** produce una arquitectura que emerge de la no-ambigüedad de los tests automatizados, lo cual no exime de las revisiones de código entre compañeros ni de hacer preguntas a los desarrolladores veteranos.

El esquema básico del **TDD** lo podemos ver a continuación:

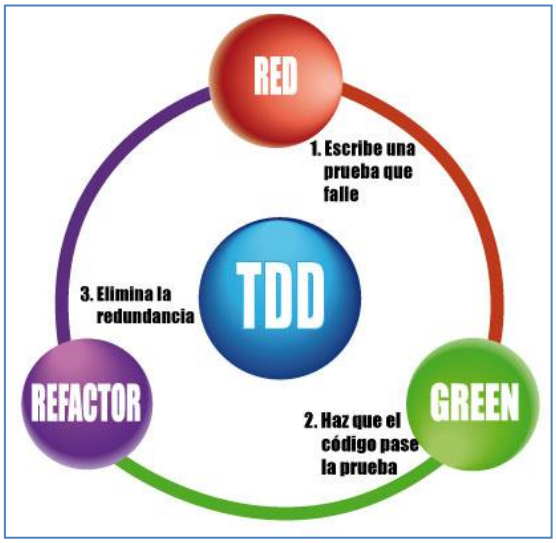


Imagen 6.2.1 – Esquema ciclo **TDD**

Cómo vemos, el algoritmo solo tiene 3 pasos: **Ciclo Rojo**, **Ciclo Verde** y **Refactorización**. Es una descripción metafórica, ya que los frameworks de tests suelen colorear en rojo aquellas especificaciones que no cumplen y en verde las que lo hacen. De modo que, cuando escribimos el test, el primer color es el rojo por que no existe código que implemente el requisito, y cuando ya está implementado se pasa a verde.

El **Ciclo Rojo** consta de escribir la especificación del requisito, el ejemplo, el test. Una vez tenemos claro cuál es el requisito, lo expresamos en forma de código. Y la gran pregunta puede ser ¿Cómo es posible escribir un test para un código que todavía no existe? Lo que hay que pensar es que realmente un test no es inicialmente un test, sino un ejemplo o especificación, pero modificable. Para escribirlo debemos de pensar en cómo queremos que sea la API del objeto que nos ocupa, es decir, trazar antes de implementar. Pero solamente una parte pequeña del objeto, una comportamiento bien definido, y sólo uno; hay que hacer el esfuerzo de imaginar cómo sería el código del SUT si ya estuviera implementado y cómo comprobaríamos que, efectivamente, hace lo que pedimos que haga. En sus inicios el test fallará, ya que todavía no tenemos el código que hace que el test pase y de un resultado correcto (de ahí que se denomine ciclo rojo).

En **Ciclo Verde** se busca implementar el código que hace funcionar el ejemplo. Teniendo el ejemplo escrito, codificamos lo mínimo necesario para que se cumpla, para que el test pase. Típicamente, el mínimo código es el de menor número de caracteres, porque mínimo quiere decir el que menos tiempo llevó escribirlo. No importa que el código parezca feo o chapucero, eso lo vamos a enmendar en el siguiente paso y en las siguientes iteraciones. En este paso, la máxima es no implementar nada más que lo estrictamente obligado a cumplir para la especificación actual. Estaremos tentados a escribir el código que gestiona comportamientos derivados de lo que estamos implementando, que harán falta en un futuro próximo; es aquí cuándo hay que contener el impulso y anotar todo ello en un margen, aparcándolo para una iteración próxima.

La **Refactorización** es el último paso en la iteración. No significa reescribir código, sino modificar el diseño sin modificar el comportamiento. Lo que debemos hacer es rastrear el código (también el del test) en busca de líneas duplicadas y las eliminamos refactorizando. Además revisamos que el código cumpla ciertos principios de diseño (SOLID, DRY…) y refactorizamos para que así sea. Existe mucha documentación en la web y buenas referencias bibliográficas para la refactorización, de hecho, se podría decir que un estudio bastante grande el conseguir una buena refactorización de tu código. Una referencia web en este sentido la podemos encontrar en esta página: Catalog of Refactorings, y un buen libro es el de *Fowler* **[REF99]**.

Cuando hayamos realizado los tres pasos de la especificación que nos ocupa, tomamos la siguiente y volvemos a repetirlos.

De una forma un poco más detallada, podríamos expandir estos tres ciclos en siete pasos más específicos para determinar el ciclo de desarrollo guiado por pruebas, en el que, mediante los colores, asemejamos estos pasos a la imagen 6.2.1:

- **Elegir un requisito –** Se elige de una lista el requerimiento que se cree que nos dará mayor conocimiento del problema y de fácil implementación.
- **Escribir una prueba –** Se comienza escribiendo una prueba para el requisito.
- **Verificar que la prueba falla –** Si la prueba no falla es porque el requerimiento ya está implementado o porque la prueba es errónea.
- **Escribir la implementación –** Escribir el código más sencillo que haga que la prueba funcione.

- **Ejecutar las pruebas automatizadas –** Verificar si todo el conjunto de pruebas funciona correctamente.
- **Eliminación de la duplicación** – Se realiza la refactorización.
- **Actualización de la lista de requisitos** – Se actualiza la lista de requisitos eliminando el requisito implementado.

Si buscamos motivos para no usar este tipo de práctica, seguramente encontremos muchos y muy variados (cómo cualquier cosa que buscamos no hacer, siempre hay motivos para no hacerlo). Sin embargo, realmente los beneficios que aporta la metodología guiada por tests son numerosos:

- La calidad del software aumenta
- Incrementa la productividad
- Conseguimos código altamente reutilizable
- El trabajo en equipo se hace más fácil
- Nos permite confiar en nuestros compañeros aunque tengan menos experiencia
- Multiplica la comunicación entre miembros del grupo
- Las personas encargadas de la garantía de la calidad adquieren un rol más inteligente e interesante
- Escribir el test antes que el código nos obliga a programar un mínimo de funcionalidad necesaria en la aplicación, evitando sobre-diseñar la misma
- Cuando revisamos un proyecto desarrollado mediante TDD, nos damos cuenta de que los tests son la mejor documentación técnica que podemos consultar a la hora de entender qué misión cumple cada pieza.

Ahora bien, como cualquier técnica, no es magia, y no dará el mismo resultado en un experto arquitecto de software que en un programador junior que está empezando. Sin embargo, es útil para ambos. Para el arquitecto porque se convierte en su mano derecha, pudiendo confiar la programación a su equipo sabiendo que la funcionalidad no va a resultar modificada. Para el programador junior porque se convierte en su director de orquesta, indicándole qué paso tiene que dar en cada momento y, paso tras paso, le guía en la implementación que ha sido asignada.

6.3- Metodología BDD

Un paso por encima del **TDD** nos encontramos con el **BDD – B**ehavior **D**riven **D**evelopment – o Desarrollo guiado por el comportamiento. **BDD** amplía las ideas de **TDD** y las combina con otras ideas de diseño de software y análisis de negocio. De este modo, se busca proporcionar un proceso (y una serie de herramientas) a los desarrolladores, con la intención de mejorar el desarrollo del software. **BDD** se basa en **TDD,** formalizando las mejores prácticas de **TDD**, clarificando cuales son y haciendo énfasis en ellas.

Algunos de los inconvenientes de utilizar **TDD** son el utilizarla cuando vamos a trabajar sobre tiempos de desarrollo reducidos, el que se requiera de conocer más técnicas o herramientas, los criterios sobre qué se constituye una buena prueba unitaria, las

herramientas… Y sobre todo los malentendidos, ya que TDD no es puramente una técnica de pruebas, sino que más bien está en el marco de diseño y codificación.

En **BDD** no se prueban unidades independientes o clases, probamos escenarios y el comportamiento de las clases a la hora de cumplir dichos escenarios, los cuales pueden estar compuestos de varias clases.

Vamos a ver de forma gráfica la integración de ambas técnicas (**TDD** y **BDD**):

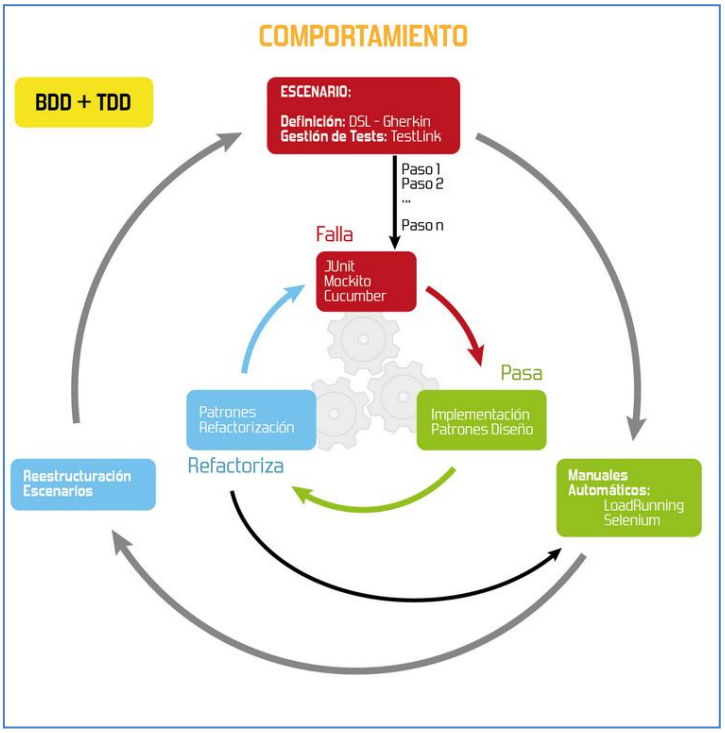


Imagen 6.3.1 – Esquema **BDD + TDD**

Como vemos, realmente son dos ciclos de desarrollo unidos. Por la parte exterior está el desarrollo **BDD**, a más alto nivel, y por la parte interior está el desarrollo **TDD**, implementado a más bajo nivel y que busca el testeo mediante pruebas unitarias. Lo que el diagrama refleja es que una vez que definimos el escenario a testear, el cual falla (**BDD**), iteramos en el círculo interno hasta que todo está correcto (**TDD**). Cuándo han pasado, y se ha refactorizado el código, volvemos al círculo externo y probamos que el escenario pase. Acto seguido refactorizamos el código de los escenarios.

Lo que busca y fomenta **BDD** no es más que lo que hemos visto con **TDD**, pero llevado a una programación a más alto nivel. Con **TDD** nos centrábamos en la implementación, **BDD** se centra en el comportamiento.

A la hora de llevar a la práctica BDD es muy recomendable usar un **DSL** (**D**omain **S**pecific **L**anguaje), el cual nos ofrece un lenguaje común sobre el que poder hacer los tests y así disminuir la fricción a la hora de compartir los tests. Los **DSL**s más usados suelen constar de pasos o Steps, y se suelen dividir en:

1- **Dado**…: (**Given**) Los pasos necesarios para poner el sistema en el estado que se desea probar.

2- **Cuando**…: (**When**) La interacción del usuario que acciona la funcionalidad que deseamos testear.
3- **Entonces**…: (**Then**) En este paso vamos a observar los cambios en el sistema y ver si son los deseados.

Utilizar un **DSL** en lugar de un lenguaje natural, nos puede ayudar a mantener el mismo nivel de abstracción en el momento de definir los escenarios; un buen **DSL** es flexible y potente una vez todos los miembros del grupo lo conocen y usan. Un ejemplo podría ser el siguiente:

Dado que he introducido "2" en la calculadora
Y que he introducido "3" en la calculadora
Cuando pulso el botón "suma"
Entonces el resultado que aparece en pantalla debería ser "5"

Todo esto nos sirve para comprobar el comportamiento interno de la aplicación. Una vez esté la funcionalidad hecha, queremos saber si es lo que buscaba realmente el usuario. Para ello existen los tests de aceptación y llegamos al punto al que queríamos llegar, el desarrollo **ATDD.**

6.4- Desarrollo ATDD

El Desarrollo Dirigido por Tests de Aceptación (**ATDD – A**cceptance **T**est **D**riven **D**evelopment) es igualmente **TDD** pero a un nivel diferente. Los test de aceptación, o del cliente, son el criterio escrito de que el software cumple los requisitos de negocio que el cliente demanda. Son ejemplos escritos de los dueños del producto.

El algoritmo es el mismo de tres pasos visto anteriormente, pero a mayor zancada. En **ATDD** la lista de ejemplos (tests) de cada historia, se escribe en una reunión que incluye a dueños del producto, desarrolladores y responsables de calidad. Todo el equipo debe entender qué es lo que hay que hacer y por qué.

A modo gráfico, vamos el diagrama que mostramos a continuación sería una representación de lo que abarca el ciclo de **ATDD**:

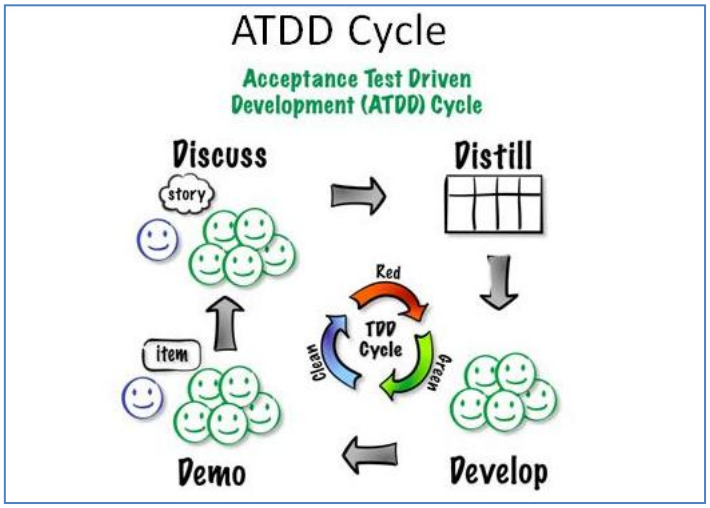


Imagen 6.4.1 – Esquema ciclo **ATDD**

Como observamos en el diagrama, lo primero que hacemos es una reunión con el propietario de la aplicación y debatimos los términos que vamos a tratar en la iteración. Refinamos los puntos tratados y nos ponemos a desarrollar mediante metodologías **BDD/TDD**, que a su vez, como hemos visto, consta del algoritmo de 3 pasos. Una vez terminado el desarrollo, se le hace una demostración al cliente de los *ítems* tratados y se pasa a la siguiente iteración en el caso de que se acepte. De esta manera añadimos el conjunto de *ítems* al bloque de historias del cliente. Si no es así, se vuelve a iterar sobre la misma característica.

En la imagen 6.4.1 aparece el concepto de "Historia" (*Story*) y el de "Item". Una historia de usuario posee similitudes con un caso de uso, salvando ciertas distancias. Por hacer una correspondencia entre historias y casos de uso, podríamos decir que el título de la historia se corresponde con el caso de uso tradicional. Sin embargo, la historia no pretende definir el requisito. Escribir una definición formal incurre en el peligro de la imprecisión y la malinterpretación, mientras que contarlo con ejemplos ilustrativos, transmite la idea sin complicaciones. En **ATDD** cada historia de usuario contiene una lista de ejemplos que cuentan lo que el cliente quiere, con total claridad y ninguna ambigüedad (el "Item"). El enunciado de una historia es tan sólo una frase en lenguaje humano, de alrededor de cinco palabras, que resume lo que hay que hacer. Algunos ejemplos podrían ser:

- *Login en el sistema*
- *Añadir un perro a mi cuenta*
- *Crear una exhibición*
- *Inscribir un perro en una exposición…*

Breves, concretas y que tengan valor. Son el resultado de escuchar al cliente y ayudarle a resumir el requisito en una sola frase. MUY IMPORTANTE: Están escritas en un vocabulario del negocio del cliente, no en vocabulario técnico. A base de iterar, estimar en cada iteración y hacer retrospectiva al final de la misma, vamos refinando la habilidad de escribir historias y estimarlas.

Cada historia nos provocará una serie de preguntas acerca de los múltiples contextos que se pueden dar. Son las que naturalmente hacen los desarrolladores a los analistas de negocio o al cliente:

- ¿Qué hace el sistema si el perro está en la cuenta de otro usuario?
- ¿Qué sucede si el perro es borrado de la cuenta de un usuario?
- ¿Se le indica al usuario que el perro no ha podido ser inscrito en alguna exposición y el motivo?, etc.

Las respuestas a estas preguntas son afirmaciones, ejemplos, que transformamos en tests de aceptación. Por lo tanto, cada historia de usuario tiene asociado uno o varios tests de aceptación (los *Items* que se reflejaban en la imagen 6.4.1). Por ejemplo, estos tests pueden ser:

- Cuando el usuario hace login con el nombre de usuario "usuario1" correctamente, el sistema devuelve un mensaje que muestra: "Welcome usuario1"

- Cuando el usuario añade un perro a su lista de perros, el sistema devuelve un mensaje que dice: “Perro añadido correctamente”
- Cuando queremos añadir un perro a una exhibición debemos clicar en el botón que pone “Añadir perro a exhibición”…

Con el tiempo y la experiencia en el manejo de este tipo de metodología y sintaxis, podremos aprender a simplificar la extensión de la frase, obteniendo simplificaciones de las frases anteriores:

- Hacer login con “usuario1” produce: “Welcome usuario1”
- Añadir un perro muestra: “Perro añadido correctamente”
- Clicar en “Añadir perro a exhibición” para enrolar un perro

Por lo tanto, podemos definir los tests de aceptación como afirmaciones en lenguaje humano que tanto el cliente, como los desarrolladores y la máquina entienden, siendo labor del equipo de desarrollo conseguir que también la máquina lo entienda. Para ello existen varios frameworks, algunos de los cuales hemos estudiado en la sección 5.3 de este mismo documento.

Para cada uno de los tests de aceptación de una historia de usuario, habrá un conjunto de tests unitarios y de integración de grano más fino, que se encargará, primero, de ayudar a diseñar el software y, segundo, de afirmar que funciona como su desarrolladores querían que funcionase. Por esto, **ATDD** es el comienzo del ciclo iterativo a nivel desarrollo, porque partiendo de un test de aceptación vamos profundizando en la implementación con sucesivos tests unitarios hasta darle forma al código que finalmente cumple con el criterio de aceptación definido.

7- Recursos utilizados

Una vez redactada la metodología de desarrollo, nos vamos a centrar en exponer los recursos que hemos utilizado para la realización del proyecto. Empezaremos definiendo los **recursos hardware**, seguidamente veremos los **recursos software**, y finalmente definiremos brevemente las **tecnologías utilizadas** en el desarrollo. De esta última parte veremos una descripción breve de cada una de las tecnologías empleadas, ya que consideremos que no es necesario extenderse en un estudio de todas estas tecnologías.

7.1- Recursos hardware

En el apartado de recursos hardware, y puesto que nuestra aplicación va a estar emplazada en la nube, no hemos necesitado de mayores complementos que un ordenador personal que se ha utilizado, así como los periféricos derivados del uso de este (Ratón y pantalla independiente). En nuestro caso se ha usado un PC portátil y esta es una breve descripción del sistema:

<table>
<tr>
<td>  </td>
<td><u>Acer Aspire Series</u>
- Procesador: Intel Dual Core T3200 2.0 Ghz.
- Memoria RAM: 3 GB.
- Disco duro: Hitachi HTS543225L 250 GB.
- Tarjeta gráfica: Mobile Intel(R) 4 Series Express – integrada.
- Sistemas operativos: Windows Vista Home Premium / Centos v.6.5
- Monitor Externo: VideoSeven TC. 17’’</td>
</tr>
</table>

7.2- Recursos software

En este apartado vamos a ver los distintos recursos software que se han utilizado para el desarrollo del proyecto.

7.2.1- Sistemas operativos

Los sistemas operativos son la base para el funcionamiento de los programas que se han utilizado en el proyecto. En nuestro caso, hemos hecho uso de la versión 6.5 de **CentOS** y **Windows Vista Home Premium**.

7.2.1.1- CentOS

CentOS (**C**ommunity **ENT**erprise **O**perating **S**ystem) es una bifurcación de la distribución de Linux Red Hat Enterprise RHEL, y es un sistema operativo de código abierto y libre. Su objetivo es ofrecer a los usuarios un sistema profesional y gratuito a la vez que robusto, estable y

fácil de instalar y utilizar.

Es un sistema que se ha utilizado durante la carrera y se ha considerado una buena elección a la hora de seleccionar un sistema operativo basado en Linux para trabajar. Será el sistema sobre el que trabajaremos para el desarrollo del proyecto.

7.2.1.2- Windows Vista Home Premium

Para la realización de este documento se han utilizado aplicaciones basadas en Windows. **Windows Vista Home** es una versión de Microsoft Windows enfocada para ser utilizada en equipos de escritorio de hogares y empresas, portátiles, tablets y equipos media center. Posee buenas características de rendimiento, fiabilidad y seguridad. Existen versiones de 32 y 64 bits; en nuestro caso en particular hemos utilizado la versión de 32 bits, que es la que se ajusta a nuestro sistema. No existe versión gratuita de este sistema operativo.

7.2.2- Navegador Web

El navegador web que hemos utilizado durante toda la realización del proyecto, tanto en CentOS como en Windows, ha sido **Mozilla Firefox. Firefox** es un navegador libre y gratuito que ofrece numerosas funcionalidades y opciones de personalización, un funcionamiento excelente y que es conocido por la protección de la vida privada. Existe en versión para Windows, Mac OS y GNU/Linux y por defecto incluye numerosas funcionalidades.

7.2.3- Entorno de Desarrollo Integrado

El sistema que vamos a desarrollar va a ser implementado en Ruby on Rails. Para un buen desarrollo en este framework vamos a utilizar **RubyMine. RubyMine** es un **IDE** comercial, del cual la universidad posee licencia de uso. Es considerado para muchos expertos en Rails unos de los mejores IDEs del mercado. Está construido sobre JetBrains' intelliJ IDEA.

RuyMine proporciona IntelliSense para código Ruby y **RoR**. Analiza el código a la misma vez que vamos escribiéndolo y apoya al desarrollo refactorizando el código, tanto para ficheros únicos de Ruby, cómo para aplicaciones webs construidas con Ruby On Rails.

7.2.4- Sistema de Control de Versiones

Para el control de versiones y sincronizar el desarrollo del proyecto con el tutor, así como para tener registrado los cambios y las distintas iteraciones en el proyecto se ha utilizado la herramienta **Git**. **Git** es un software de control de versiones diseñado por Linus Torvalds y diseñado pensando en la eficiencia y confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Es un sistema de funcionalidad plena y que proyectos de mucha relevancia utilizan (cómo por ejemplo el grupo de programación del núcleo Linux).

En particular, hemos utilizado **GitHub** como repositorio del proyecto. Github es una repositorio para alojar proyectos utilizando el sistema de control de versiones Git. Utiliza el framework **RoR** y el código que almacenamos es de forma pública (posee una versión de pago para hacerlo privado).

7.2.5- Editores de texto

Para la realización de la memoria, se han utilizado dos tipos distintos de editores de texto: **Microsoft Office Word** para el groso del documento, y **Notepad++** para presentar ciertas partes de este documento de una forma más vistosa.

7.2.5.1- Microsoft Office Word 2007

Hemos utilizado **Microsoft Office Word 2007** para redactar el marco teórico del proyecto. Es un software incluido en la suite de ofimática de Microsoft Office 2007 y está destinado al procesamiento de textos. Está disponible para Windows y para Mac OS, pero no tiene versión para GNU/Linux. Es un software de pago y es el procesador de textos más popular del mundo.

7.2.5.2- Notepad++

Notepad++ es un editor de texto y de código fuente libre con soporte para varios lenguajes de programación. Es de soporte nativo a Windows.

Es un editor muy similar al típico bloc de notas, pero incluye opciones avanzadas útiles para desarrolladores y programadores. En nuestro caso ha sido utilizado para el formateo de varios fragmentos citados en el presente documento y que han sido escritos durante la fase de desarrollo, permitiendo citar secciones de código de una manera amigable y vistosa.

7.2.6- Editor de bocetos - Mockups

Los bocetos gráficos presentados en este documento se han realizado mediante la aplicación **Balsamiq Mockups**. Esta aplicación permite crear los mockups de sistemas de una forma rápida y vistosa. Viene integrado con una gran librería de controles pre-construidos para poder realizar los bocetos sin necesidad de estar dibujándolos. Además permite la opción de incluir controles no incluidos importándolos como imagen, descargándolos o haciendo una petición en su portal. Es un software de pago aunque tiene una versión de prueba de 30 días.

7.2.7- StarUML

Esta herramienta ha sido utilizada para la creación de diversos diagramas representados en este documento. **StarUML** es una herramienta UML desarrollada por MKLab que genera todo tipo de diagramas compatibles con la plataforma de programas Microsoft Office. Sirve para crear diagramas de casos de uso, diagramas de clases, diagramas de secuencia, de estado, de entidad –interrelación…

StarUML se maneja con facilidad y permite dibujar los gráficos manualmente o mediante el uso de varias plantillas que contiene el archivo de instalación. Es un programa de uso gratuito y actualmente se encuentra en su versión 2.0.

7.2.8- Microsoft office Visio 2007

Microsoft Visio es un software de dibujo vectorial para Microsoft Windows. Las herramientas que lo componen permiten realizar diagramas de oficina, diagramas de bases de datos, diagramas de flujo, UML y demás cosas que permiten iniciar al usuario en los lenguajes de programación.

Aunque originalmente apuntaba a ser una aplicación para dibujo técnico para el campo de la ingeniería y arquitectura, cuándo Microsoft lo adquirió implicó cambios drásticos de directrices y expandió mucho más su alcance.

7.2.9- MySQL Workbench

MySQL Workbench es una herramienta visual de diseño de bases de datos que integra desarrollo de software, administración de bases de datos, diseño de bases de datos, creación y mantenimiento para el sistema de base de datos MySQL. Es el sucesor oficial de DBDesigner 4 de fabForce.net y reemplaza al anterior conjunto de software, MySQL GUI Tools Bundle.

MySQL Workbench es uno de los primeros productos de la familia MySQL que ofrece dos ediciones diferentes – una open source y una edición comercial. En nuestro caso en particular haremos uso de la edición open source, con ciertas limitaciones para con el manejo.

7.3- Tecnologías utilizadas

Esta sección la vamos a destinar a referenciar todas las tecnologías que se han usado para desarrollar nuestro proyecto.

7.3.1- Ruby / Ruby on Rails

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos. Fue creado por Yukihiro Matsumoto y lo presentó en 1995. Combina sintaxis de otros lenguajes de programación (Python y Perl) con características de programación orientada a objetos. Comporta también funcionalidad con otros lenguajes como Lisp, Lua, Dylan y CLU. Está implementado bajo licencia de software libre.

Como mencionamos en el estudio de las herramientas, **Ruby on Rails** es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación **Ruby**, siguiendo el paradigma Modelo Vista Controlador. En el apartado 5.1.2 de este documento se ha descrito en detalle el framework **Ruby on Rails**, por lo que ahora no vamos a entrar en más detalle. Se puede consultar el citado apartado para más información acerca de la tecnología.

En nuestro proyecto, hemos utilizado **Ruby on Rails** para todo el desarrollo de la aplicación en sí. Es la base sobre la que hemos asentado todas las demás tecnologías y sobre la cual hemos construido nuestro sistema.

7.3.2- Ruby Version Mannager

Ruby **V**ersion **M**annager (a partir de ahora **RVM**) es una herramienta de línea de comandos que permite instalar fácilmente, gestionar y trabajar con múltiples entornos de Ruby, desde interpretes hasta conjuntos de gemas.

Para evitar sorpresas siempre es recomendable utilizar el mismo intérprete que se usa en producción. El problema es cuando se trabaja en varios proyectos escritos en Ruby al mismo tiempo y cada uno de ellos utiliza un intérprete diferente, o unas versiones de gemas específicas, o una versión determinada del propio Ruby. **RVM** se utiliza para tener paquetes independientes de intérpretes, o versiones distintas de un mismo intérprete y poder utilizarlas sin necesidad de estar instalando o desinstalando nada.

La utilización de esta herramienta ha sido puramente académica, ya que nosotros únicamente vamos a estar trabajando en un proyecto concreto, sin embargo es importante el conocerla cuando se está trabajando con Ruby.

7.3.3- Active Record

El patrón **Active Record**, conocido sobre todo por su uso en **RoR**, es un patrón de diseño que define una forma de acceder a los datos de una base de datos relacional y convertir las filas de una tabla en objetos.

Su objetivo es simplificar enormemente la interacción con la base de datos y eliminar la tarea de escribir código **SQL** a mano para operaciones comunes. A diferencia de otros **ORM** (**O**bject-**R**elational **M**apping – Mapeo Objeto-Relacional), no es necesario utilizar generados de código sin mantener archivos de configuración para las tablas. Está basado en convenciones en lugar de configuraciones. La principal ventaja que objeta utilizar Active Record es la reutilización, permitiendo llamar a los métodos de un objeto de datos desde varias partes de la aplicación e incluso desde diferentes aplicaciones.

Active Record lo hemos usado como herramienta **ORM** para acceder a nuestra base de datos. Durante el desarrollo, cada vez que buscábamos hacer una consulta en la base de datos, apartábamos la sintaxis de SQL a un lado y modelábamos la consulta mediante sentencias Active Records, lo que nos ha resultado muy producente, ya que, una vez coges algo de experiencia, las consultas a las bases de datos se vuelven muy metódicas.

7.3.4- HTML5

HTML5 es un lenguaje markup (**H**yper **T**ext **M**arkup **L**anguaje) usado para estructurar y presentar el contenido para la web. Es uno de los aspectos fundamentales para el funcionamiento de los sitios. Es la quinta revisión importante para este lenguaje básico de la **WWW,** HTML. **HTML5** especifica dos variantes de sintaxis: un clásico HTML (text/html), conocida como **HTML5**

y una variante XHTML conocida como sintaxis XHTML5, que deberá ser servida como **XML**. Esta es la primera versión en la que HTML y XHTML se han desarrollado en paralelo. Esta versión de HTML se publicó en octubre del 2014.

Nuestro uso de **HTML5** ha sido definido para la construcción del esqueleto de las vistas de la aplicación. Es una aplicación web y la base sobre la que se va a montar el aspecto visual del proyecto ha sido desarrollada en **HTML5**.

7.3.5- JavaScript

JavaScript, comúnmente abreviado como "**JS**", es un lenguaje de programación interpretado. Se define a sí mismo como un lenguaje orientado a objetos, basado en prototipos, débilmente tipado y dinámico. En términos generales, permite a los desarrolladores crear acciones en sus aplicaciones web, haciendo las páginas web más dinámicas, es decir, incorporar efectos de texto, animaciones, acciones al pulsar botones, ventanas emergentes…

Es utilizado por profesionales e iniciados en el desarrollo y diseño de sitios web. No requiere de compilación ya que el lenguaje funciona en el lado del cliente; son los navegadores los encargados de interpretar estos códigos.

En nuestro caso en particular hemos utilizado **JavaScript** en ciertos elementos de nuestra aplicación para darle algo de dinamicidad al producto. Hemos tenido en cuenta que no siempre está habilitada esta opción en los navegadores, por lo que siempre se ha implementado ambos casos: el responder cuándo la petición viene con **JavaScript** o cuándo la petición es **HTML** puro.

7.3.6- JQuery

Una de las bibliotecas más utilizadas de JavaSript es **JQuery**. Fue creada por Jhon Resig y su funcionalidad radica en que nos permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos y agregar interacción con la técnica AJAX a páginas web.

JQuery es software libre y código abierto, pudiéndose utilizar en proyectos tanto públicos como privados y ofrece una serie de funcionalidades basadas en JavaScript que de no ser utilizado requeriríamos de mucho más código para realizar la misma función, es decir, con las funciones propias de **JQuery** se logran grandes resultados en menos tiempo y espacio. Precisamente este es el uso que le hemos dado en nuestro desarrollo. Nos hemos valido de **JQuery** para simplificar las tareas que hemos hecho con AJAX y JavaScript.

7.3.7- AJAX

AJAX, acrónimo de **A**synchronous **J**avaScript **A**nd **X**ml (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones, como JavaScript, se ejecutan en el cliente, en el navegador de los usuarios, mientras se mantiene una comunicación asíncrona con el servidor en segundo plano. De esta manera es

posible realizar cambios sobre las páginas sin necesidad de recargaras, mejorando la interactividad, velocidad y usabilidad de las aplicaciones.

JavaScript es el lenguaje interpretado en el que efectuaremos las funciones de llamada de **AJAX** mientras que el acceso a los datos los realizamos mediante XMLHttpRequest. Como nombramos anteriormente, y puesto que son herramientas que se utilizan conjuntamente, hemos utilizado **AJAX** mediante **JavaScript** y su biblioteca **JQuery** para darle algo de dinamismo a la aplicación web.

7.3.8- CSS3

Atendiendo a la parte estética, la base de la apariencia visual de la aplicación la hemos conseguido utilizando **CSS3. CSS**, en general es el acrónimo de **C**ascading **S**tyle **S**heets (Hoja de estilo en cascada) y es un lenguaje usado para definir la presentación de un documento estructurado escrito eh **HTML** o **XML**. El World Wide Web Consortium (**W3C**) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores. **CSS3** no es más que la tercera versión estable de CSS, con sus particulares características y mejoras.

La idea que está detrás de **CSS** es separar la estructura de un documento de su presentación. Las hojas de estilo nos permitirán definir de manera eficiente la representación de nuestras páginas y es uno de los conocimientos fundamentales que tenemos que tener como diseñador web. Al tener una aplicación orientada a la web, necesitaremos conocerlo y lo utilizaremos para vistosidad a nuestra aplicación junto con la siguiente tecnología presentada, **Bootstrap**.

7.3.9- Bootstrap

En pocas palabras, podemos definir **Bootstrap** como el framework de Twitter que permite crear interfaces web con **CSS** y **JavaScript** que adaptan la interfaz dependiendo del tamaño del dispositivo en el que se visualice de forma nativa, es decir, automáticamente se adapta al tamaño de un ordenador, de un dispositivo móvil, de una tablet… En otras palabras, nos ofrece posibilidades de un buen acabado gráfico para la aplicación en diferentes plataformas.

Aún ofreciendo todas las posibilidades que ofrece **Bootstrap** a la hora de crear interfaces web, los diseños creado con **Bootstrap** son simples, limpios e intuitivo, esto les da agilidad a la hora de cargar y adaptarse a los dispositivos.

El framework trae varios elementos con estilos predefinidos fáciles de configurar: botones, plantillas de diseño con tipografías, cuadros, menús de navegación, formularios… Es el más popular en **GitHub**.

Se ha decido utilizar **Bootstrap** atendiendo al principal motivo de que no somos diseñadores gráficos, y diseñar una aplicación que luzca un acabado visual excelente no es nuestro trabajo. Centrándonos en que es un proyecto individual, **Bootstrap** nos ha ofrecido la

posibilidad de tapar esas carencias de diseño gráfico que sin duda deberá estar presente en cualquier aplicación comerciable y nos ha permitido crear una presentación atractiva del producto final.

7.3.10- Cucumber

En la sección 5.3.1 estudiamos esta herramienta de testeo, **Cucumber**. Aquí simplemente lo vamos a referenciar de manera que quede recogido su uso como tecnología utilizada a la hora de realizar el proyecto.

Como nombramos, **Cucumber** es una de las herramientas que vamos a utilizar para la parte de testeo de la aplicación. La descripción y todos sus detalles los podemos encontrar en la referencia citada en el párrafo anterior. Para el uso de **Cucumber** tuvimos que aprender **Gherkin** (DSL de **Cucumber)**, e hicimos uso de la gema **factory-girl**, que detallaremos un poco más abajo de este mismo apartado. Además también hay que nombrar a **Capybara**, otra gema cuya misión es ayudarnos a probar las aplicaciones simulando que es un usuario real interactuando con la aplicación.

7.3.11- RSpec

Al igual que Cucumber, este framework de testeo ha sido estudiado anteriormente en este documento y la podemos encontrar en la sección 5.3.2. Se ha utilizado particularmente para el teste de que el contenido del fichero semilla de la aplicación sea el correcto. Para ello ha sido necesario el estudio de su lenguaje en particular y de la forma que hay que estructuras las sentencias escritas.

7.3.12- Heroku

Heroku es otra herramienta estudiada anteriormente y es la plataforma en la nube en que se va a hospedar nuestra aplicación. Su mención en este apartado es meramente referencial a que es otra de las tecnologías que hemos integrado en nuestra aplicación. Su descripción se encuentra en la sección 5.2.1 de este escrito. Para la migración al servidor en la nube, una vez desarrollada la misma, hemos tenido que hacer algunos cambios y corregir ciertos erros que el despliegue en **Heroku** requería (añadir herramientas para *mailing*, cambio de gestión de base de datos de SQlite3 a PostgreSQL, corrección de ciertas incompatibilidades en **Heroku**…)

7.3.13- SQlite3 / PostgreSQL

Tanto **SQlite3** como **PostgreSQL** son sistemas de bases de datos relacionales orientados a objetos y libre uso; la primera creada por Richard Hipp y la segunda por Michael Stonebraker.

A diferencias de los sistemas de gestión de base de datos cliente-servidor, el motor de estos sistemas no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, las bibliotecas se enlazan con el programa pasando a formar parte del mismo. El programa utilizará la funcionalidad del sistema de base de datos a través de llamadas simples a subrutinas y funciones.

El motivo de la utilización de dos sistemas distintos, es principalmente académico. A inicios, debido a que no se tenía experiencia ninguna en desarrollo en **RoR** y la metodología que esto conlleva, se decidió empezar con una gestión en **SQlite3** y no complicar más aún el desarrollo, ya que **SQlite** es el que viene por defecto y además hay algunas incompatibilidades entre ciertas versiones de Rails y **Postgre**. Por lo tanto, se inició con un desarrollo en **SQlite3** y luego, a la hora de llevar la aplicación a producción y cambiar a **Postgre**, se solucionaron las pequeñas complicaciones de compatibilidad que se produjeron.

7.3.14- JSON

JSON es el acrónimo de **J**ava**S**cript **O**bject **N**otation (Notación de Objetos JavaSript) y es, básicamente, un formato ligero para el intercambio de datos. Es una forma sencilla y eficaz de gestionar información. **JSON** describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos. Nació como alternativa a **XML**, y el fácil uso en javascript ha generado un gran número de seguidores de esta alternativa. Una de las mayores ventajas del uso de **JSON** es que puede ser leído por cualquier lenguaje de programación, por lo que puede ser utilizado para el intercambio de información entre distintas tecnologías.

7.3.15- Gemas

Además de tecnologías que hemos utilizado, hemos hecho uso de la cualidad que posee **RoR** para la utilización de las **gemas.** Una **gema** es un plugin y/o código añadido a nuestro proyecto que nos permite nuevas funcionalidades, nuevas funciones o nuevas herramientas para el desarrollo. Existen infinidad de gemas para Rails, cuyo listado se puede encontrar en la página de RubyForge. Es responsabilidad del desarrollador no sobrecargar su trabajo con gemas y utilizarlas en buena medida. Algunas de las gemas más recomendadas las podemos encontrar en la parte final del libro de *THE RAILS 4 WAY* **[THE14]**.

En nuestro caso en particular hemos hecho uso de las siguientes gemas, incluidas en el fichero "*Gemfile*" de nuestro proyecto:

Gema	Descripción
devise	Utilizada para la gestión de los usuarios. Es una solución de autentificación flexible para Rails con *Warden*. **Devise** gestiona autentificación a todos los niveles.
carrierwave	Gema que ofrece una manera simple y extremadamente flexible para subir archivos de aplicaciones Ruby.
coutry_select	Proporciona un simple *helper* para obtener un *select* HTML de un listado de los países según la norma ISO 3166.
spork	La utilizamos para acelerar la carga de los tests realizados.
capybara	Gema utilizada por Cucumber y cuya función es ayudar al testeo mediante la simulación cómo interactuaría nuestro sistema con un usuario real.
factory_girl	Nos permite la creación de elementos fijos para tests con una cómoda sintaxis y de manera sencilla.
simplecov	Presenta de una forma amigable datos sobre la cobertura que nuestros tests realizan sobre el código de la aplicación.

8- Análisis

A continuación, y posterior al estudio que hemos realizado, junto con la explicación de la metodología de desarrollo que se va a seguir y los recursos utilizados para la elaboración del proyecto, pasaremos a realizar un análisis conjunto de los sitios web y su estructura interna, determinando qué cualidades nos interesan que posea nuestro sistema. Debemos tener en cuenta, que estamos desarrollando un prototipo de un sistema, y que el sistema final que se desea obtener abarca capacidades mayores que lo que nuestro prototipo ofrece. Es por ello nos vamos a referir en ocasiones a nuestro trabajo en particular, y en otras ocasiones al sistema final que se desearía obtener.

8.1- Sitio web

Como hemos nombrado, a continuación vamos a ver qué características de los sitios web estudiados vamos a desear que contenga nuestro portal. Empezaremos detallando la estructura general y luego nos centraremos en el contenido que esperamos que contenga.

8.1.1- Estructura general

Analizando los diferentes sitios web referentes a la cinofilia, estudiados en el apartado 4, podemos sacar varios aspectos en común que deseáremos que tenga nuestro portal.

Por una parte, la mayoría de ellos tiene la información organizada y clasificada en grupos según los diferentes temas que tratan y, también en su mayoría, estos grupos están mostrados en forma de desplegables en una barra situada en la parte superior del portal web. Es una idea visual que se repite mucho a lo largo de los sitios web que hemos estudiado, y que querremos que nuestro portal también posea.

Todos ellos, independientemente de la información que contengan o el fin para el que se utilice, tienen la idea muy clara en cuanto a la organización y estructura de contenidos de su web. Como hemos mencionado, agrupan los contenidos en estos pequeños grupos de información, que luego despliegan y nos dan la posibilidad de navegar entre ellos para mostrar los distintos contenidos que el portal ofrece. Hay varias secciones generalmente comunes a estos sitios:

- Información general del sector cinofílico
- Información particular del tema al que se refiere el sitio
- Posibilidad de registro
- Ampliación de capacidades al registrarse
- Zonas públicas y zonas privadas para usuarios registrados

Debemos tener, al igual que existe en la mayoría de los sitios que hemos estudiado, un área en la que ofrezcamos información general, y un área en la que ofrezcamos información específica referente a las exhibiciones caninas y sus derivados. En nuestro trabajo en particular, la información general no va a ser abundante, ya que nos interesan cosas más allá de la información expuesta; sin embargo en el sistema final e ideal, lo que se debería buscar es

juntar información general que varios portales estudiados ofrecen (**FCI**, **Ingrus**, o incluso de la nombrada **RSCE**) y mostrarla de forma libre, de modo que no sólo se convierta en una aplicación de gestión de exhibiciones, sino de referencia general a este sector.

En cuanto al área particular de un usuario, se ha estudiado que existen las dos variantes de portales: Los que permiten registro y los que no. Se considera que proporciona más confianza para el usuario y que va a dar mayor estabilidad y profesionalidad al sistema el tener área de registro. Por este motivo, y al igual que hace **Doglle**, exigiremos estar registrados en el sistema para acceder a ciertos servicios que este proporciona.

Debemos tener en cuenta que buscamos tener un portal accesible por cualquier usuario, ya sea registrado o no. Es por ello que es muy conveniente tener áreas diferenciadas para cada tipo de usuario, dando la posibilidad de acceder a un área personal para los usuarios registrados.

Además, deberemos tener alguna sección y acciones específicas para que un "*superusuario*", o administrador, dirija el sitio web. Debe ser un área independiente y distinta a la de otros usuarios, y además dándole la posibilidad de que acceda a cualquier servicio que un usuario convencional puede obtener.

8.1.2- Contenidos

A la hora de analizar los contenidos que queremos que nuestro sistema tenga, y relacionándolos con los portales estudiados, debemos fijar unos límites y metas, ya que, por ejemplo, simplemente el pequeño número de sitios que hemos analizado ofrecen una amplia diversidad de contenidos que no queremos cubrir en su totalidad.

En nuestro prototipo en particular, la información general que puede interesarnos tener en nuestro sitio, y que no está directamente relacionada con la gestión de las exposiciones, son datos referentes a las razas y la organización de las mismas. De este modo un usuario podrá también consultar a qué grupo de razas pertenece su perro sin tener que acudir a otros sitios como el de la **FCI**. Por tanto, respecto a esto, deberíamos buscar tener los siguientes contenidos:

- Organización de los grupos de las razas
    - Secciones y sub-secciones existentes en cada grupo
    - Razas, variedades de raza y sub-variedades de raza que contiene cada uno

En términos más generales, y atendiendo al sistema totalmente construido, podríamos ampliar la capacidad que nuestro prototipo tiene referente a la consulta de las razas, con una adición de un documento con los estándares para cada raza, como el portal de la **FCI** ofrece. De esta manera, al llegar a la base de cada raza, podríamos ver de forma documentada el documento de estándar oficial de la raza.

El sistema final también sería muy interesante que tuviera un sistema de pedigrís como el que posee **Ingrus**, o al menos, que cómo usuario registrado pudiéramos referenciar a nuestros perros con un enlace directo a la página de **Ingrus** de dicho perro, de modo que muestre información más detallada sobre ellos.

Como contenidos específicos del portal de gestión de exhibiciones caninas, querremos que contenga aquella información que un aficionado o experto en la cinofilia querría buscar. En nuestro caso en particular, y atendiendo a la idea de prototipo, hay que tener información relacionada a exposiciones, para que los usuarios la puedan consultar. Por tanto, querremos tener un área de exhibiciones y dentro de esta área que cada una de ellas tenga su descripción propia, informando de precios, horarios, fechas, etc. Por otra parte, también se desea que el administrador tenga la posibilidad de insertar y definir nuevas exhibiciones, así como también modificar las ya existentes, por lo que se deberá también tener un área de contenido para estas acciones.

- Área de exhibiciones
    - Administración de exhibiciones
    - Información general de las exhibiciones disponibles
    - Información particular de cada una de ellas: Horario, precios, fechas…

Atendiendo al sistema totalmente construido e integrado, podríamos tener en cuenta más cualidades de cada exhibición para mostrar: geo-localización, acceso directo a su portal web en caso de que lo posea, posibilidad de descarga de trípticos…

También sería deseable que el sistema tuviera integradas otras funciones de otros subsistemas, como por ejemplo la posibilidad de subir resultados de exhibiciones de forma online y tener una estructura de exposiciones totalmente completo, no solamente ofrecer la opción de inscribirte en una de ellas, sino el poder ver una tabla de resultados o clasificaciones de las exhibiciones.

Además, puesto que nuestro sistema va a tener la posibilidad de tener usuarios registrados, debemos definir contenido específico para estos. Por lo tanto, habrá una zona privada personal a cada usuario registrado, en la que pueda administrar su cuenta y sus acciones:

- Área de usuarios registrados
    - Gestión de cuenta
    - Gestión de perros
    - Gestión de exhibiciones

Prestando atención a la hora de de permitir a los usuarios registrados inscribirse en exposiciones, y teniendo en cuenta que esto conlleva un coste (hay que pagar precio de inscripción), se deben tener métodos de pagos para que se pueda constatar el correcto abono del mismo. En nuestro sistema en particular, el método de pago que vamos a permitir es mediante un ingreso en cuenta y posterior envío del justificante. Por lo tanto deberá de haber cierto contenido que refleje esta opción.

En el sistema ideal se buscaría permitir el pago mediante otros métodos más cómodos y más actuales, utilizando alguna pasarela de pago o mediante tarjetas de crédito o débito.

Y este es el contenido básico que el sistema deberá tener. Como hemos podido observar, también se han definido ciertos aspectos que se desea que el sistema final (y no nuestro prototipo) tenga.

8.2- El portal

En esta sección analizaremos nuestro propio portal a desarrollar en el proyecto. En el primer punto de este apartado especificaremos los requisitos del usuario, en los que, a modo de descripción, explicaremos qué beneficios obtiene el usuario de nuestro sistema y las restricciones bajo las que tiene que operar.

El segundo punto del apartado en el que nos encontramos tratará sobre los requerimientos del sistema, en el que, de una forma más extendida y técnica, ampliaremos los requisitos del usuario vistos anteriormente.

8.2.1- Especificación de requisitos del usuario

Cómo comentamos anteriormente, pasaremos a describir los requerimientos funcionales de forma comprensible para los no expertos en la ingeniería de desarrollo. Ofreceremos en primer lugar una descripción concreta del portal web y a continuación trataremos un glosario de conceptos en el que se definirán palabras claves que se utilizaran tanto en el portal web como en este mismo documento del proyecto. En el tercer punto de esta sección mostraremos el modelo de dominio, el cual se utilizará como fuente de referencia para el diseño de los objetos del software, mostrando clases conceptuales significativas en un dominio del problema.

8.2.1.1- Descripción

El proyecto va a consistir diseñar un prototipo de un sistema final que deberá tener un repertorio completo de funcionalidad en cuanto al sector de las exhibiciones caninas.

Atendiendo a nuestro prototipo en particular, queremos diseñar una aplicación web que permita la administración de dichas exhibiciones caninas. Al ser un prototipo y no un sistema totalmente completo, limitaremos la funcionalidad del mismo, de modo que habrá restricciones parciales en cuanto a la administración de las exposiciones.

La funcionalidad principal que va a tener el sistema que vamos a diseñar es permitir a un usuario la inscripción de perros de su propiedad en exhibiciones caninas. Para ello, deberemos proveer de capacidad a dicho usuario para registrarse en el portal, añadir perros a su cuenta y poder inscribir a estos perros a ciertas exhibiciones según las reglas establecidas para ello.

Para que esto sea posible, también debemos proveer al sistema la capacidad para administrar exhibiciones. Debe existir un tipo de administrador en el sistema, que sea capaz de llevar la administración de dichas exhibiciones. Tiene que tener potestad para poder crear nuevas exhibiciones, modificar las exhibiciones ya creada, y borrar ciertas exhibiciones siempre y cuando sea posible (esto es cuándo el plazo de pago de las exhibiciones no ha comenzado).

Las exhibiciones del sistema deben de tener una serie de características de consulta útiles para el usuario: fecha de inicio, fecha de fin, descripción, tasas de inscripción, fechas de inscripción… A todas estas características debe poder acceder el usuario, de modo que pueda consultarlas y en base a su decisión, inscribir a sus perros en las exhibiciones (o no).

Una vez hecha la inscripción de los perros en la exhibición, el usuario podrá reflejar, mediante la agregación de un fichero, que el pago correspondiente al registro ha sido realizado.

Cuando el fichero esté adjunto a la inscripción, el administrador del sistema podrá verificar el correcto pago, y marcarlo como aceptado, momento en el cual el usuario no podrá volver a agregar el fichero. Mientras el pago no haya sido aceptado por el administrador se debe permitir la subida de ficheros tantas veces como el usuario desee.

De este modo, y en términos generales, tendremos un prototipo de sistema para la inscripción de perros en exhibiciones caninas, por la que habrá que pagar unas tasas mediante transferencia bancaria y acreditar dicho pago con la carga de una copia digital del recibo bancario. Este sistema estará accesible para el usuaria vía web.

A su vez, el portal web contendrá cierta información referente al sector de la cinofilia. Este proyecto en particular contiene la clasificación de las razas en grupos según la Federación Cinológica Internacional y es visible y accesible por cualquier usuario.

Entre las limitaciones que tiene nuestro prototipo podemos enumerar algunas más relevantes:

- En cuanto a la creación de exposiciones en el sistema, se ha acotado la capacidad de definir, vía web, los precios para ellas. No es posible la inserción de los precios mediante interfaz. Deberá hacerlo el administrador mediante comandos.
- A la hora de inscribir a perros en exhibiciones hay algunas opciones de clase que no se han implementado (*Couple*, *Working* y *Group Breeding*). Se le informa al usuario de ello.
- No tiene capacidades de búsquedas en el portal.

Estas limitaciones están reflejadas y definidas en el apartado de Trabajo Futuro.

8.2.1.2- Glosario de conceptos

- **Usuario:** Persona que tiene limitado el acceso a información pública de la página.
- **Usuario registrado:** Aquel usuario que está dado de alta en el portal y que ha accedido a su cuenta.
- **Usuario administrador:** Es un usuario registrado que tiene privilegios para acceder a mayores características del sistema que un usuario registrado común.
- **Exhibición/exposición canina:** Concurso donde jueces familiarizados con distintas razas evalúan cuán bien conformado se encuentra un perro en comparación con su estándar racial.
- **Cinología:** Término para referirse al estudio de lo relativo a los cánidos y perros domésticos.
- **Cinofilia o canofilia:** Labor que realizan los amantes de los perros por mejorar la calidad de sus ejemplares, exhibiéndolos en exposiciones de belleza y de obediencia, y dando a conocer las características de cada raza.
- **Estándar de raza:** Descripción de las características que conforman el espécimen ideal de una raza.

- **Can:** Perro.
- **Grupo:** Cada uno de los bloque principales en los que se encuentran los estándares de cada raza de perro utilizados para organizar exhibiciones de perros. Los grupos se dividen en secciones, y éstas a su vez en sub-secciones. Las secciones y las sub-secciones.
- **Sección:** Distinciones específicas dentro de cada grupo de razas.
- **Sub-Sección:** Un nivel más específico de distinción dentro de cada sección.
- **Variedad:** Característica particular de una raza de perro, lo suficientemente importante como para ser mencionada y catalogada en un caso particular.
- **Sub-Variedad:** Una característica determinada de la variedad de una raza.
- **Enrolar / Inscribir:** Matricular a un perro en una exhibición.
- **Clases de perros:** Agrupaciones de perros según unas determinadas características. Concepto que se utiliza para diferenciar precios a la hora de enrolar perros en las exhibiciones. Las distintas clases que hay son:
    - Clase Campeón – Champion (CCH): Ejemplares que tienen el título de Campeón de España o de un país extranjero reconocido internacionalmente.
    - Clase muy cachorro – Baby (CMC, CB): Perros entre 3 y 5 meses
    - Clase Cachorro – Puppy (CC, CP): Para perros entre 5 y 9 meses
    - Clase Jóvenes – Junior (CJ): Ejemplares entre 9 y 18 meses
    - Clase Abierta – Open (CA, CO): Perros mayores de 15 meses
    - Clase Intermedia – Intermediate (CI): Perros entre 15 y 24 meses
    - Clase Veteranos – Veteran (CV): Ejemplares con más de 8 años.
    - Clase Parejas – Couple (CO,CC): Macho y hembra de la misma raza, variedad y propietario.
    - Clase Trabajo – Working (CT, CW): Perros que han superado una prueba de utilidad y más de 15 meses.
    - Clase Grupo de Crías – Group Breeding (CCR, CGB): Mínimo de 3 ejemplares, y máximos de 5, de la misma raza, variedad, de un mismo criador.
- **Pago:** Tributos que hay que abonar por enrolar a perros, o grupo de perros en las exhibiciones. El pago por las inscripciones se debe de certificar adjuntando una copia digitalizada del extracto bancario correspondiente a la inscripción.
- **Estado del pago:** Condición de aceptado (*Accepted*), rechazado (*Rejected*) o no verificado (*Unverified*) del fichero subido por el usuario.

8.2.1.3- Modelo del dominio

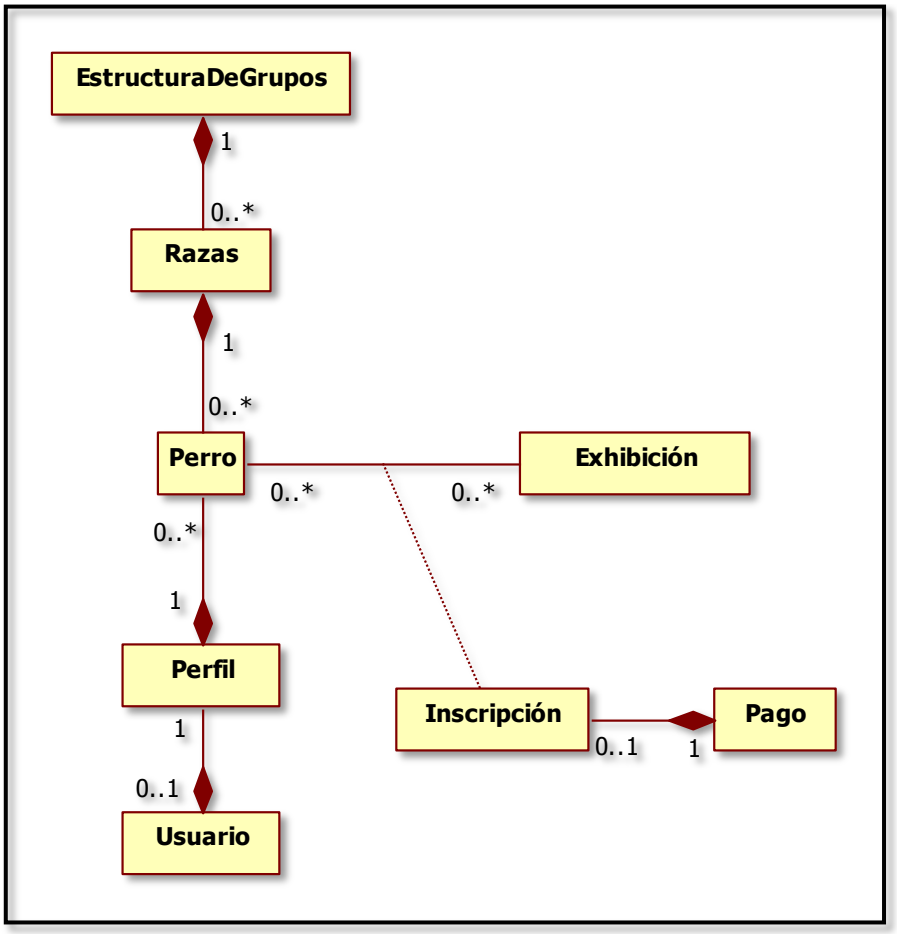


Imagen 8.2.1 – Modelo de Domino

En este modelo podemos identificar las clases que describimos a continuación:

- **EstructuraDeGrupos**: Representa toda la estructura de grupos del sistema (grupos, secciones y sub-secciones).
- **Razas**: Esta clase representa la estructura de razas que se implementará en el sistema (razas, variedades y sub-variedades).
- **Perro**: Elemento para representar los perros de los usuarios.
- **Exhibición**: Evento en el que se pueden inscribir los perros.
- **Inscripción**: Elemento que representa la inscripción de un perro en una exhibición.
- **Pago**: Cada uno de los tributos que se pagan por las inscripciones en las exhibiciones.
- **Perfil**: La clase perfil representa la información almacenada en el sistema respecto a las personas que pueden existir en el mismo.
- **Usuario**: Clase para representar las personas que interactuarán con nuestro sistema.

8.2.2- Especificación de requisitos de software

En la especificación de requisitos del software vamos a ver más técnica y detalladamente los requerimientos de nuestro sistema. En un primer apartado se hará una

descripción de los mismos. Seguidamente veremos un diccionario de términos un poco más específico para la tarea que nos comprende que el que vimos en la sección anterior. A continuación dispondremos de un diagrama de modelo del diseño, en el que veremos de una forma más desglosada y especializada el modelo de dominio presentado en la sección anterior. Los siguientes puntos que veremos serán apartados explicativos referentes a los actores en el sistema (junto con sus roles y acciones/objetivos), el modelado de los casos de uso y un listado de los mismos. Este listado se puede usar de referencia para localizar el detalle de los casos de uso en el ANEXO I del presente documento.

8.2.2.1- Descripción

Se va a utilizar como servidor, para alojar la aplicación, la nube; concretamente la plataforma Heroku para almacenar y gestionar los datos. La nube informática es un nuevo modelo de prestación de servicios de negocio y tecnología, que permite al usuario acceder a un catálogo de servicios estandarizados y responder a las necesidades de su negocio, de forma flexible y adaptativa, en caso de demandas no previsibles o de picos de trabajo, pagando únicamente por el consumo efectuado.

Las ventajas que nos proporciona la nube para alojar nuestro sistema son múltiples, y la ventaja más directa que podemos mencionar es la referente al coste. Cualquier sistema alojado en la nube abarata costes: No hay que comprar servidores, no hay que tener personal para mantenerlos, no hacen falta licencias, se paga sólo por lo que consumes… En el apartado 9 de este documento (Diseño) se estudiará más acerca de este modelo de almacenamiento.

Las exhibiciones son el pilar del sistema por el que va a circular todo lo referente al mismo. Son creadas mediante un formulario web que debe rellenar el administrador. Una vez creadas, la inserción de los costes de enrolar a los perros en éstas se hace de forma manual. Para ello, se ha diseñado un hash, cuya estructura se detalla en la sección 10.3, que deberá insertar el administrador del sistema vía consola de Rails. Una vez creada la exhibición, con sus precios, *deadlines*, grupos de clases y demás características definidas, se le habilitará la opción al usuario para poder enrolar a sus perros en la misma.

El sistema ha de ser capaz de tener usuarios registrados con perfiles personales y áreas privadas para cada uno. Debe de haber formularios de registro para los usuarios, y una vez registrados, se validará que el email es correcto, mediante una validación de registro por correo electrónico. Además estos usuarios deben poseer capacidades para insertar sus perros en el sistema. Debe de haber formularios que permitan crear perros y que se relacionen con el usuario en sí.

Estos usuarios, una vez han insertado sus perros, tienen que tener la posibilidad de inscribir a los perros a las exhibiciones cuyo *deadline* esté abierto. De este modo, una vez inscrito al perro, se crearán registros de *enrolments* que relacionen los perros de los usuarios con la exhibición que se ha seleccionado.

Una vez inscritos los perros, se debe permitir la subida de ficheros como medio de verificación del pago realizado. El administrador del sistema será el encargado de validar que el documento subido realmente corresponde al pago de lo que el usuario debe abonar.

El usuario registrado tiene que tener un perfil privado en el que tenga accesos a sus datos y pueda modificarlos o darse de baja en el sistema. Cuando un usuario registrado se dé de baja en el sistema, su información será eliminada del mismo.

8.2.2.2- Glosario de conceptos

- **Usuario:** Persona que accederá a nuestro sistema y tendrá permitido el acceso a aquellas partes del sistema dónde no se requiera registro.
- **Usuario registrado:** Aquel usuario que ha sido logueado en el sistema. Teniendo acceso a áreas personales y a lugares del sitio accesibles sólo para usuarios registrados.
- **Usuario administrador:** Es un usuario registrado que tiene privilegios para acceder a mayores características del sistema que un usuario registrado común y cuya función es administrar correctamente el sistema.
- **Enrolar / enrolment:** Entrada en la base de datos que crea una relación entre un perro, una exhibición y un pago en el caso de que este se haya realizado. Además tendrá el importe que el usuario debe abonar por dicho enrolment.
- **Pago / payment:** Entrada en la base de datos que refleja una cantidad de dinero a abonar y contiene una referencia a un fichero subido que debe ser el resguardo digitalizado de un extracto bancario correspondiente al ingreso de lo que el usuario tiene que pagar por el enrolment.
- **Estado del pago / status:** Es un campo de cada registro de un enrolment, que el administrador es el que tiene la potestad de cambiarlo cuando se ha verificado el fichero subido.
- **Acciones CRUD:** Capacidades para crear (**C**reate), leer (**R**ead), actualizar (**U**pdate) y borrar (**D**elete) elementos en el sistema.

8.2.2.3- Modelo del diseño

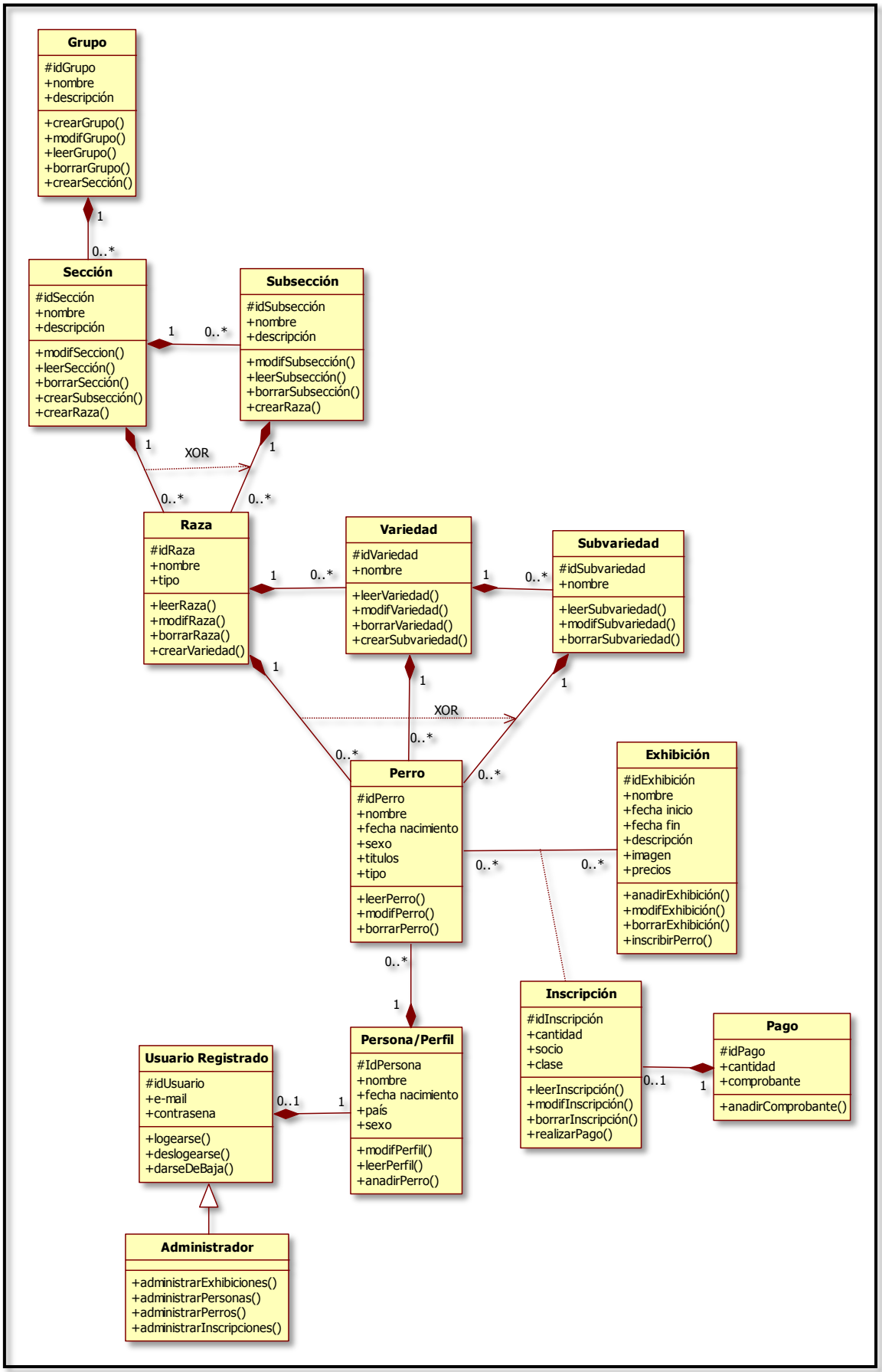


Imagen 8.2.2 – Modelo de diseño

8.2.2.4- Actores del sistema

En nuestro sistema nos vamos a encontrar con cuatro posibles tipos de actores interactuando con él. En el siguiente diagrama mostramos la relación existente entre dichos actores. Podemos observar cómo existe un **Visitante**, que es una generalización de un **Usuario no Registrado**. Cuando un usuario se da de alta en el sistema pasa a ser un **Usuario Registrado**. A su vez, en este último tipo de usuario se encuentra el **Administrador** del sistema, que será un **Usuario Registrado**, con mayores capacidades a las de éste en cuanto a interactuar con el sistema. Por último, existe otro tipo de actor en el sistema, que se asemeja con el perfil de un **Usuario Registrado**, sin embargo cumple más propósitos. Una **Persona** (o **Perfil**), es por un lado un perfil de cada usuario registrado, con información referente a ellos, y por otro lado también pueden ser **Personas** que no son usuarios del sistema, sino que el administrador ha creado un perfil de ellos para que en un futuro se pueda extender la funcionalidad del sistema.

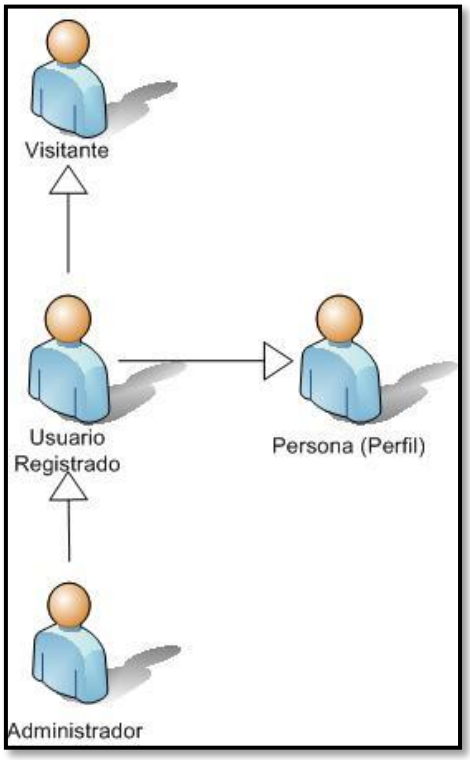


Imagen 8.2.3 – Diagrama de actores del sistema

8.2.2.5- Listado de actores y roles

A continuación veremos una lista con los actores del sistema citados anteriormente y la definición de cada uno de ellos. Cómo dijimos anteriormente hay cuatro actores en nuestro sistema y ofreceremos dos actores principales y dos secundarios.

Cabe destacar que el actor Persona actualmente no tiene funciones propias de un actor en el sistema, por lo que solamente se va a citar en este apartado para que quede constancia que la base para esta futura mejora de funcionalidad en el proyecto está contemplada y diseñada.

Actor	Tipo	Definición
Visitante	Secundario	Todo aquel usuario que tiene acceso a los servicios e informaciones esenciales que proporciona el sitio web; podrá ver exposiciones e información general en el sistema. No se registran de él datos relevantes, aunque tiene la capacidad de registrarse, loguearse y convertirse en un **usuario registrado**.
Usuario Registrado	Principal	Es un **visitante** que se ha registrado y logueado en el sistema y posee funciones ampliadas a las de un actor tipo **visitante**. El **usuario registrado** tiene la capacidad de insertar, en el sistema, canes con sus correspondientes datos, apuntar éstos a listas de exposiciones, tiene capacidades para ver listas de exposiciones, subir archivos de pagos de tasas…
Persona (Perfil)	Secundario	Una **Persona** es actualmente un simple **perfil** de un usuario, con información personal de ellos. Sin embargo, existe el actor **Persona** pensando en posibles mejoras del proyecto. En un futuro se deseará que en el sistema haya **Perfiles** de **Personas** que realmente no serán usuarios del sistema (ya que no interactuarán con nuestro sistema), sino que estarán en él para hacer referencia, por ejemplo, al organizador de una exposición, al responsable de un evento, a una persona de contacto…
Administrador del sistema	Principal	**Usuario registrado** con capacidades de ver exposiciones vigentes, las que no están en vigor, control **CRUD** sobre ellas, ver los canes inscritos en cada exposición, los pagos que hay en el sistema… Además posee las funciones propias de un administrador para gestionar a todos los usuarios del sitio y la información que hay en él.

8.2.2.6- Listado de actores y objetivos

Actor	Objetivos	Descripción de la acción
Visitante	Registrarse	Rellenando el formulario de registro del sitio permite al **Visitante** quedar registrado en el sistema y convertirse en un **Usuario Registrado**.
	Ver detalles de exposiciones caninas	Se le muestra al usuario un listado que exhibe las exposiciones que hay actualmente en el sistema. El usuario podrá seleccionar entre ver exposiciones activas, exposiciones ya finalizadas, conocer detalles alguna exposición…
	Ver información específica	El portal contendrá información referente al sector de la cinofilia que podrá consultar el **Visitante**.
Usuario registrado	Confirmación de pago de tasas	Por norma general, para hacer oficial la inscripción en una exposición hay que abonar un cierto importe impuesto por el organismo de la competición. El **Usuario Registrado** tendrá la posibilidad de cargar un fichero en el sistema que confirme que ha abonado dichas tasas.
	Gestionar su perfil	El **Usuario Registrado** poseerá un perfil privado, con potestad para su modificación, en el que se podrá encontrar informaciones personales: email, nombre de

		usuario, fecha de nacimiento…
	Gestionar sus inscripciones	Tendrá la posibilidad de borrar sus inscripciones en exposiciones siempre y cuando no haya cargado en el sistema el pago por las tasas.
	Gestionar sus perros	Un **Usuario Registrado** tiene la posibilidad de registrar ejemplares caninos en su cuenta y contar con una lista propia de canes vinculados a su cuenta. Tendrá autoridad para la administración de dichos ejemplares: inserción de nuevos ejemplares, borrado de canes existentes, modificado de los mismos…
	Inscripción en exposiciones	Cuando un **Usuario Registrado** tenga canes asociados a su cuenta (al menos uno), será posible la inscripción de los mismos a exposiciones existentes en el sistema, de modo que se permita la participación de éste en la competición.
Administrador	Gestionar exposiciones	Un **Administrador** podrá realizar cualquier operación referente al control y administración de eventos del tipo exposición: Creación, modificación, borrado… Siempre bajo la responsabilidad del propio **Administrador** y bajo ciertas normas.
	Gestionar estructura de razas	En el sistema actual hay una estructura general de esquema de razas, según la FCI (Grupos, secciones, sub-secciones, razas, variedades…) El administrador tiene las capacidades para realizar operaciones CRUD en cualquiera de estos modelos.
	Gestionar usuarios	Existen opciones para la correcta administración de usuarios. Se podrán modificar informaciones de usuarios tales como nombre de usuario, datos personales, perros… Todo ello bajo responsabilidad del **Administrador**.
	Confirmación de pagos	Deberá comprobar que los ficheros subidos por los usuarios, referentes al pago de las inscripciones, son correctos. De este modo podrá dar la aprobación para que la inscripción quede acaptada.
	Gestionar personas	Capacidades **CRUD** para gestionar **Personas/Perfiles** (No confundir con **Usuarios**)

8.2.2.7- Diagramas UML de casos de uso

Ya vistos y descritos los actores, junto con las funciones propias de cada uno de ellos, vamos a ver los diagramas UML de los casos de uso de los mismos. En estos diagramas se dispondrán de forma gráfica los casos de usos de dichos actores y a continuación se mostrará una tabla en la que se listarán estos casos de uso. El detalle de los casos de uso, especificado bajo un formato de tablas, se puede ver en el ANEXO I de este documento.

Visitante

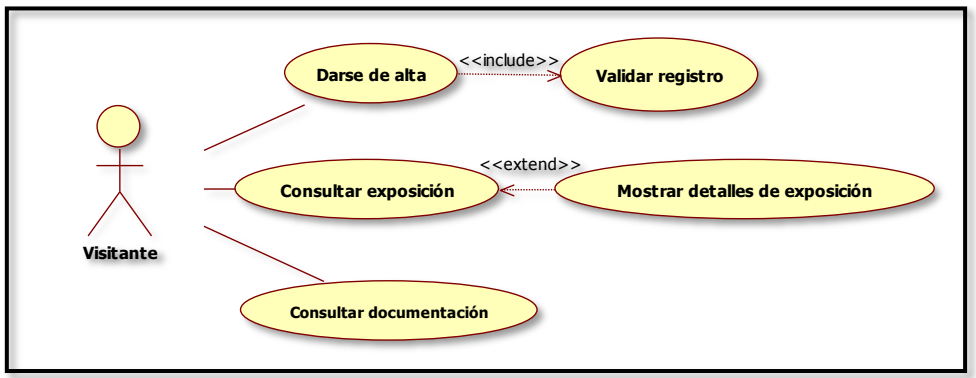


Imagen 8.2.4 – Casos de uso – Visitante – *Darse de alta – Consultar exposición – Consultar documentación*

Usuario Registrado

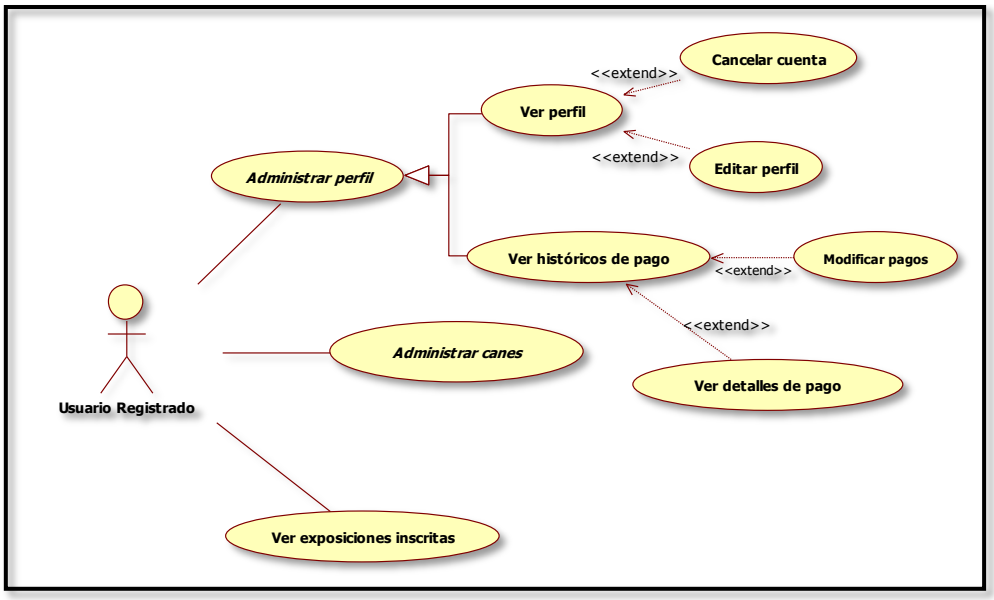


Imagen 8.2.5 – Casos de uso – Usuario Registrado – *Administrar perfil – Administrar canes - Ver exposiciones inscritas*

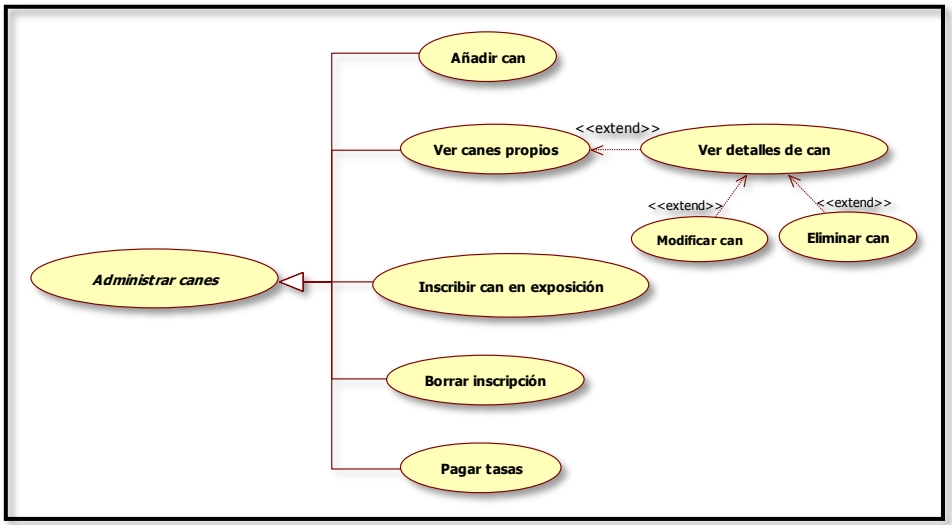


Imagen 8.2.6 – Casos de uso – Usuario – *Administrar canes (Detalle)*

Administrador

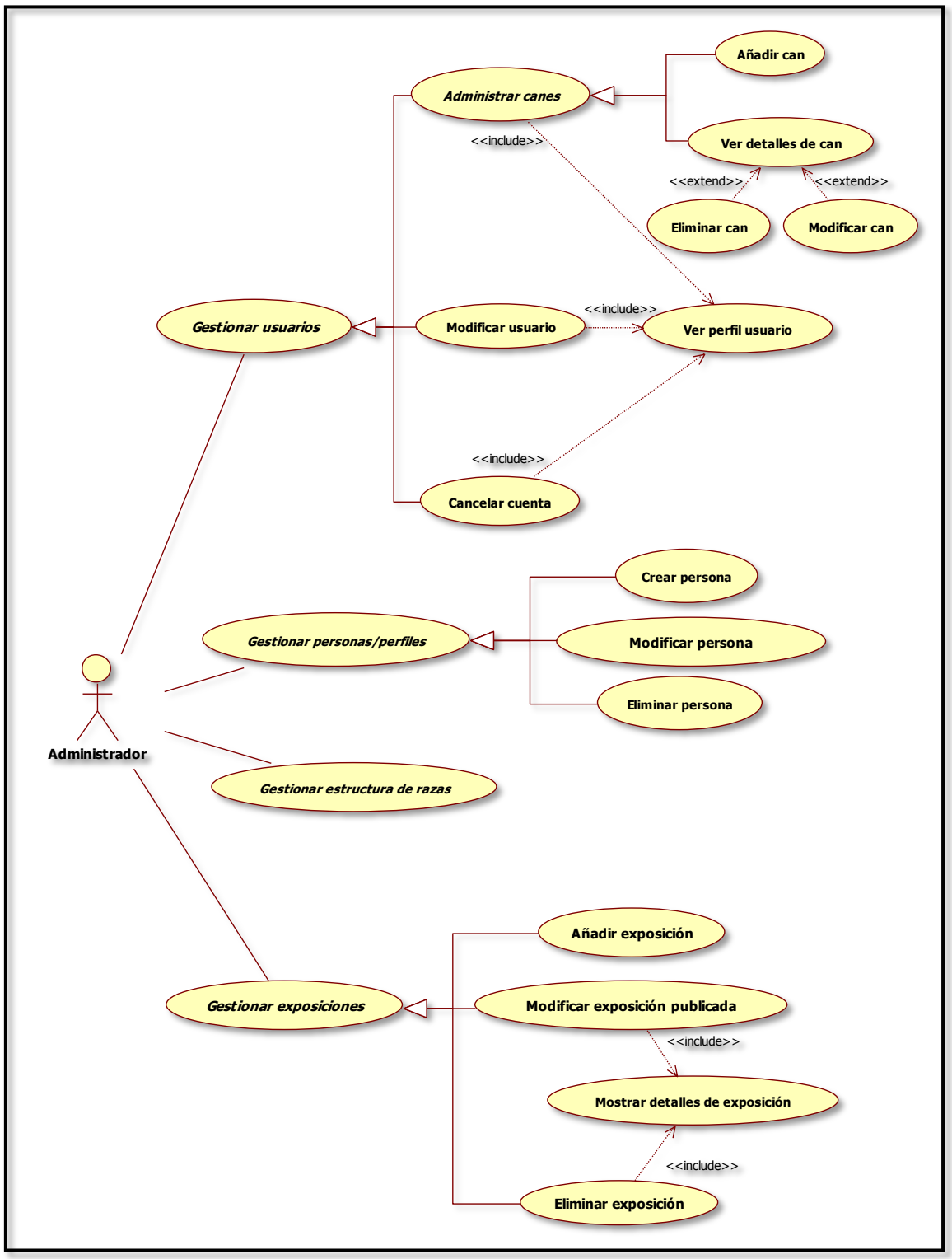


Imagen 8.2.7 – Casos de uso – Administrador –Gestionar usuarios – Gestionar personas – Gestionar estructura de razas – Gestionar exposiciones

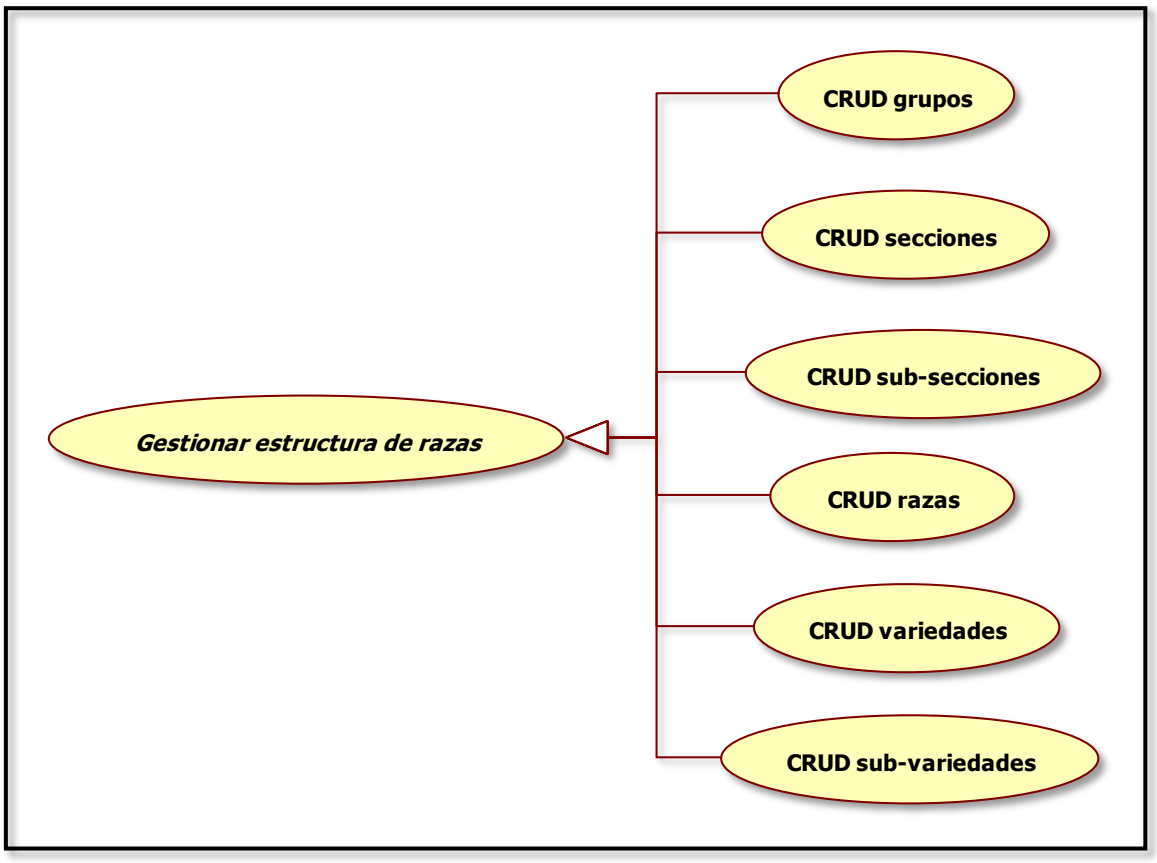


Imagen 8.2.8 – Casos de uso – Administrador – *Gestionar estructura de razas (Detalle)*

8.2.2.8- Listado de casos de uso

Actor	Caso de uso	Nº de CU
Visitante	Consultar documentación	01
	Consultar exposición	02
	Darse de alta	03
	Mostrar detalle de exposición	04
	Validar registro	05
Usuario Registrado	Añadir can	06
	Borrar inscripción	07
	Cancelar cuenta	08
	Editar perfil	09
	Eliminar can	10
	Inscribir can en exposición	11
	Modificar can	12
	Modificar pagos	13
	Pagar tasas	14
	Ver canes propios	15
	Ver detalles de can	16
	Ver detalles de pago	17
	Ver exposiciones inscritas	18
	Ver históricos de pagos	19
	Ver perfil	20

Administrador	**Módulo usuarios**	
	Añadir can	21
	Cancelar cuenta	22
	Eliminar can	23
	Modificar can	24
	Modificar usuario	25
	Ver detalles de can	26
	Ver perfil usuario	27
	Módulo personas/perfiles	
	Crear persona	28
	Eliminar persona	29
	Modificar persona	30
	Módulo exposiciones	
	Añadir exposición	31
	Eliminar exposición	32
	Modificar exposición publicada	33
	Módulo razas	
	CRUD Grupos	34/35/36/37
	CRUD Sección	38/39/40/41
	CRUD Sub-Sección	42/43/44/45
	CRUD Sub-Variedad	46/47/48/49
	CRUD Raza	50/51/52/53
	CRUD Variedad	54/55/56/57

8.2.2.9- Diagramas de secuencia

Los diagramas de secuencia están elaborados en lenguaje UML y representan de forma esquemática las acciones que debe ir realizando un rol en el sistema a lo largo de un periodo de tiempo para completar una tarea. Para la elaboración de estos diagramas nos hemos apoyado en el desarrollo que hemos hecho previamente de los diagramas de clase y los casos de uso, elementos comunes en la ingeniería del software.

En este apartado veremos los diagramas de secuencia más representativos del sistema, entendiendo que no es necesario el representar la totalidad de ellos, ya que muchos de los casos de uso del sistema siguen un patrón común y no hay que entrar en la redundancia.

Visitante

En este caso vamos a tres diagramas de secuencia del actor **Visitante**. El primero de ellos, será el referente al registro del usuario en el sistema (**Darse de alta**), que se considera de los más relevantes, junto con el diagrama de **Validar registro**, acciones conjuntas que debe realizar el usuario para quedar validado su registro en el sistema. El tercer diagrama que mostramos corresponde al caso de uso de **Mostrar detalles de exposición**.

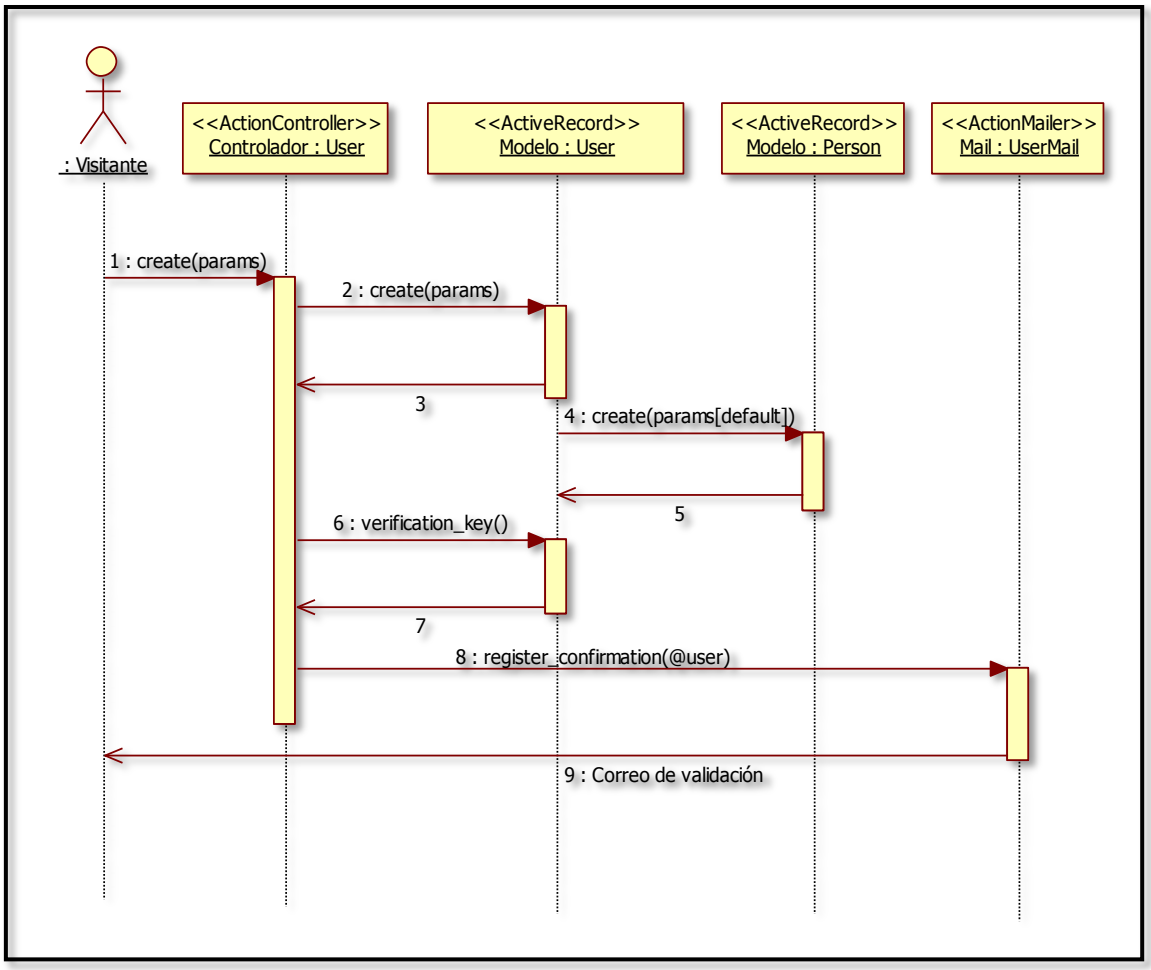


Imagen 8.2.9 – Diagrama de secuencia – Visitante – Darse de alta

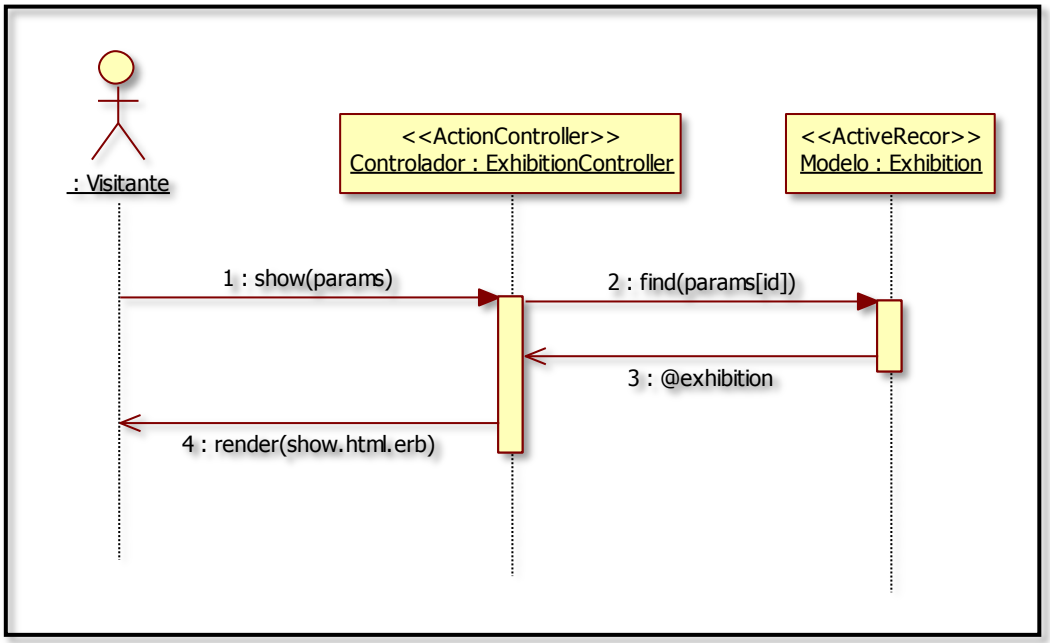


Imagen 8.2.10 – Diagrama de secuencia – Visitante – Mostrar Detalle de Exposición

Usuario Registrado

Los casos de uso más relevantes que podemos mostrar de un Usuario Registrado son los referentes a las acciones que puede hacer con respecto a sus perros y a las exhibiciones que hay en el sistema. De este modo podríamos estudiar el **Añadir Can**, **Eliminar Can**, **Inscribir Can en exhibición**, **Pagar Tasas** y **Modificar Pagos**. Empecemos viendo el primero:

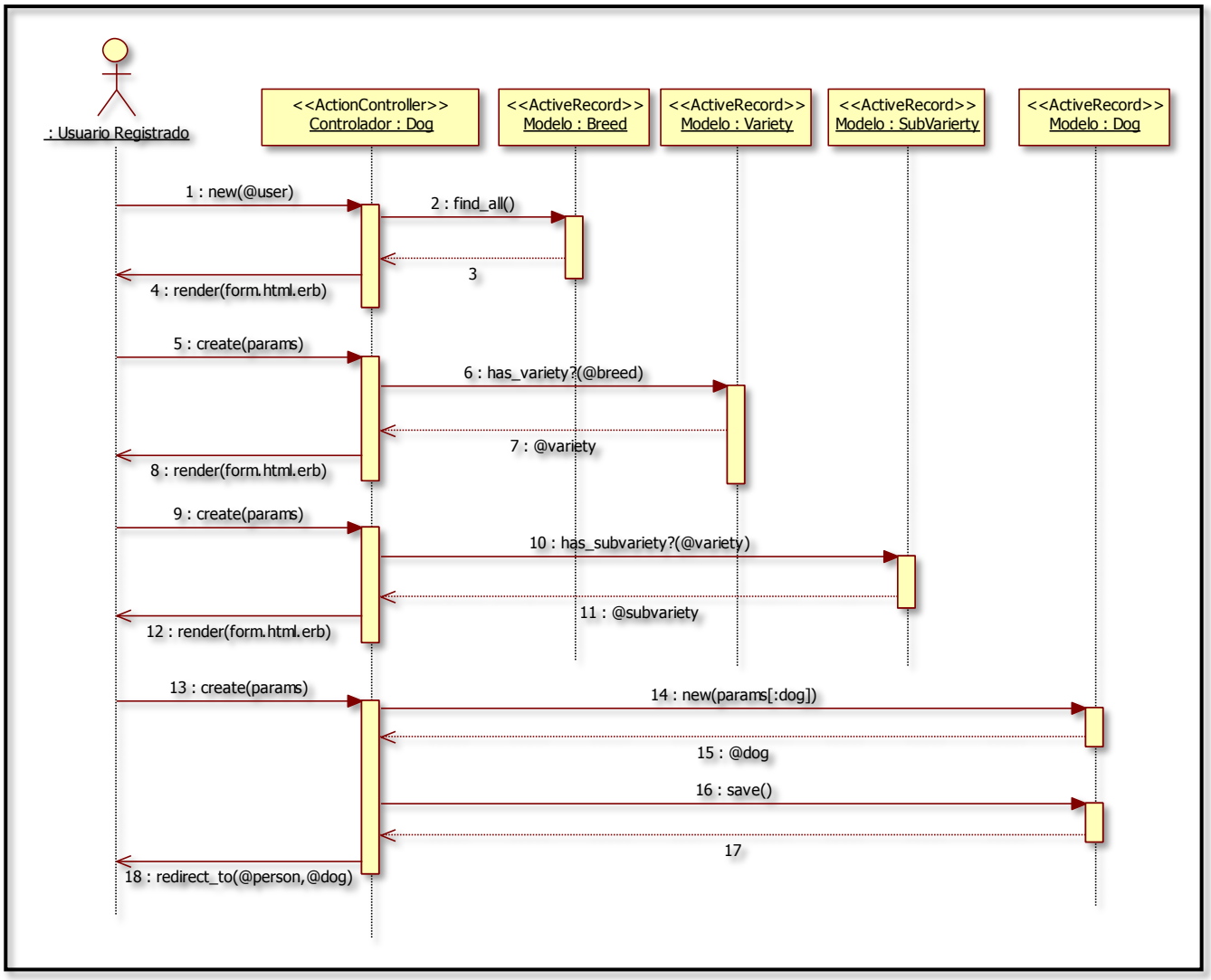


Imagen 8.2.11 – Diagrama de secuencia – Usuario Registrado – Añadir Can sin JavaScript

En este primer diagrama estamos mostrando el caso de crear un perro (en formato html, sin el uso de JavaScript), cuya raza tiene variedad y sub-variedad. Como vemos, al no utilizar JavaScript en el cliente, y para no perder la funcionalidad del sistema en navegadores que no tengan JavaScript, lo que se ha hecho es ir renderizando la misma página hasta que se llega al caso base (en este caso que la raza tiene sub-variedad). Así pues, cuando el usuario selecciona la raza, y rellena el formulario de creación de perro, al darle al botón de "crear perro", lo que se hará será renderizar la misma página, mostrando la variedad de la raza. A su vez, cuando seleccione la variedad y se le vuelva a dar a crear perro, se le mostrará otro nuevo cuadro para que seleccione la sub-variedad de la raza. Una vez se ha seleccionado la base, se hace un *save* del elemento.

En navegadores que utilizan JavaScript este caso es mucho más dinámico y cómo para los usuarios; veamos cómo se comportaría el sistema en el caso de que éste posea dicha herramienta:

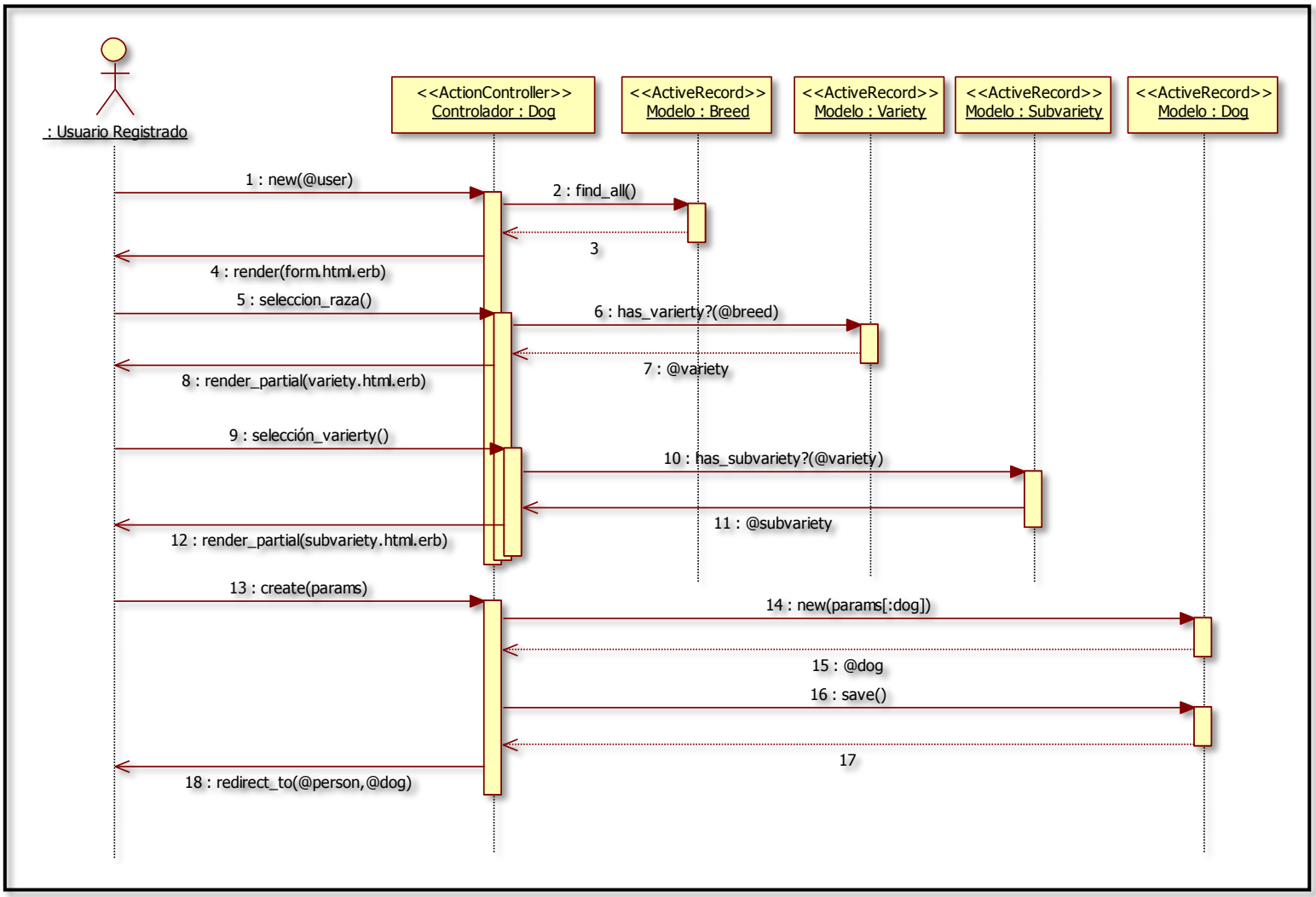


Imagen 8.2.12 – Diagrama de secuencia – Usuario Registrado – Añadir Can con JavaScript

Podemos observar la comodidad que el sistema ofrece a los usuarios con JavaScript habilitado en el navegador. En el caso anterior había que "crear" tres veces el perro; en este caso, en la misma página del formulario nos irán apareciendo los nuevos desplegables según nuestra selección anterior sin necesidad de ir pulsando ningún otro botón.

A continuación vamos a ver el diagrama que se genera al borrar a un perro de nuestra cuenta, es decir el caso de uso **Eliminar can**:

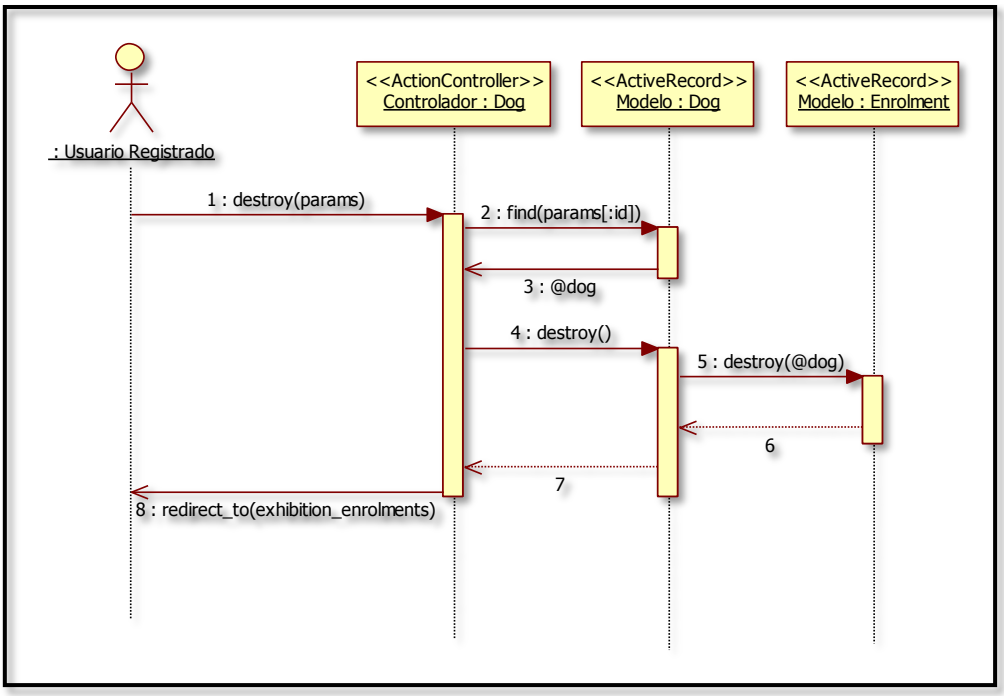


Imagen 8.2.13 – Diagrama de secuencia – Usuario Registrado – Eliminar Can

Observamos la comunicación existente entre el modelo *Dog* y el modelo de *Enrolment* (inscripciones en exposiciones). Cuando un perro se elimina del sistema, acto seguido también quedan eliminadas todas sus participaciones en exhibiciones.

El siguiente diagrama de secuencia que veremos será el de **Inscribir Can en Exposición** (*enrolar* un perro). Lo más relevante que podemos observar es la comunicación que se mantiene entre el controlador y el modelo del *enrolment* a la hora de comprobar que el perro es aceptado para la clase que el usuario ha seleccionado. Una vez aceptado, nos comunicamos con el modelo *exhibition* para recoger el precio que constará está inscripción (el *enrolment*).

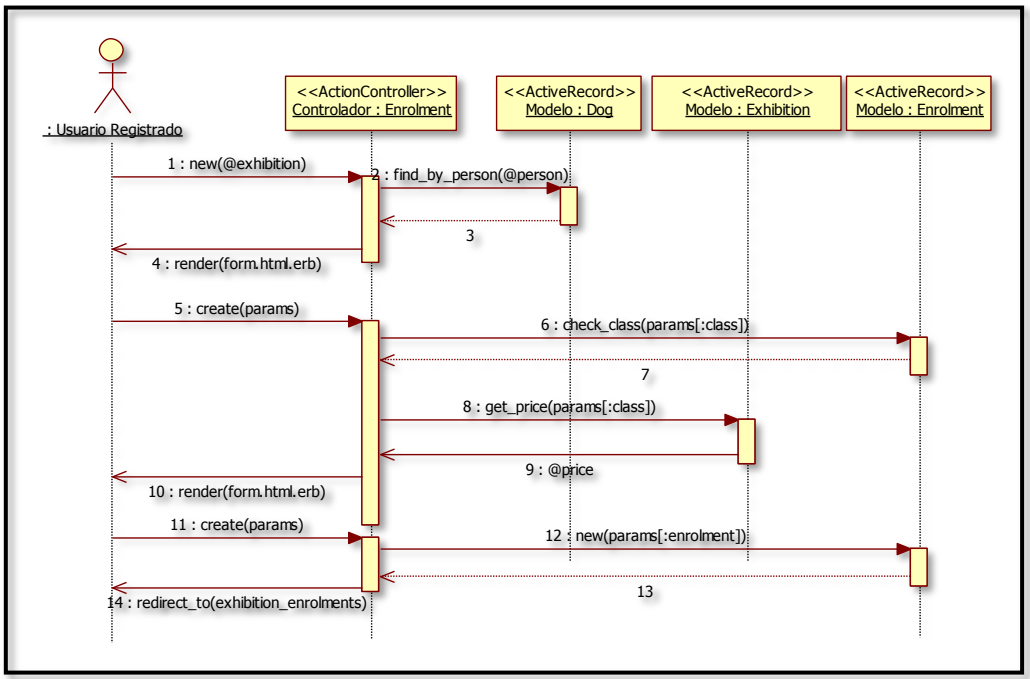


Imagen 8.2.14 – Diagrama de secuencia – Usuario Registrado – Inscribir Can en Exposición

Otro de los diagramas de secuencia que nos parecen interesantes y relevantes para presentar, es el diagrama que corresponde con el caso de uso **Pagar Tasas**, cuyo actor principal también es el Usuario Registrado. Veámoslo a continuación:

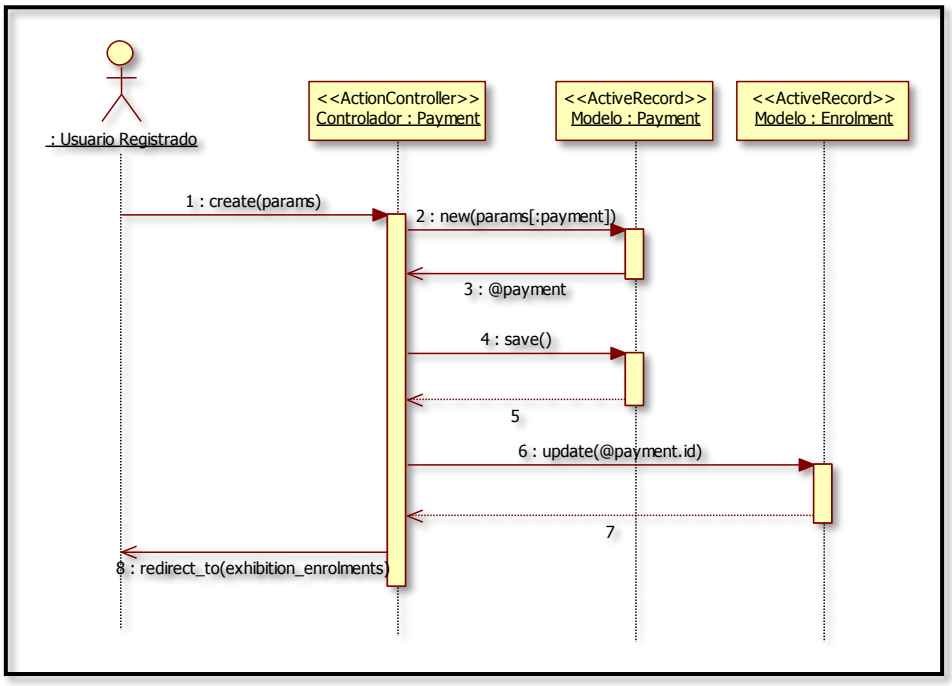


Imagen 8.2.15 – Diagrama de secuencia – Usuario Registrado – Pagar Tasas

El punto más relevante en este diagrama lo encontramos a cuando se salva el registro del *payment*. Una vez salvado, se actualizan los *enrolments* asociados a este *payment*, de modo que pasa a tener referencia cada uno de las inscripciones con su correspondiente pago.

Cómo último diagrama para los usuarios registrados vamos a estudiar el que corresponde al caso de uso **Modificar Pagos**. En este caso, lo que queremos hacer es modificar un pago que todavía no ha sido aceptado por el administrador (volver a subir el fichero, añadir un nuevo comentario…):

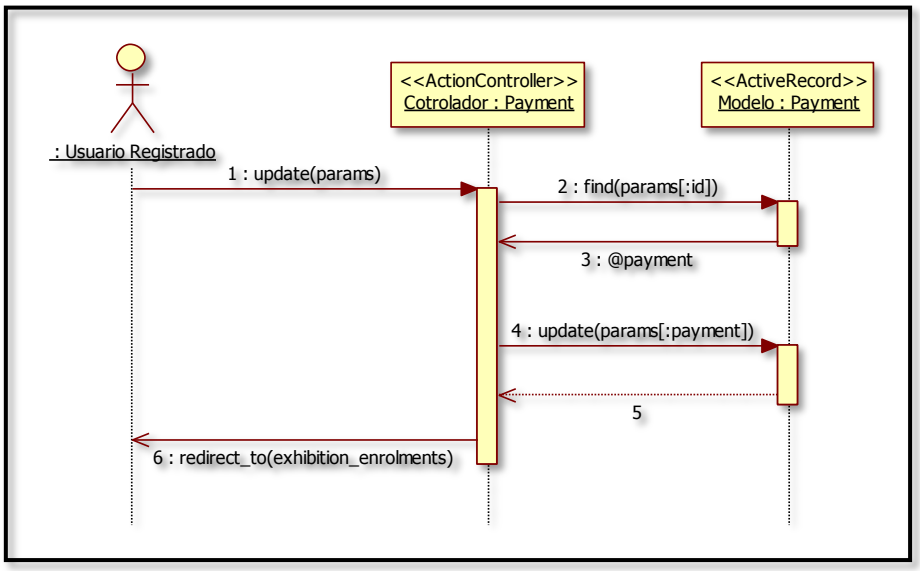


Imagen 8.2.16 – Diagrama de secuencia – Usuario Registrado – Modificar Pagos

Vemos como, al igual que muchos casos de uso, no es más que un diálogo típico entre el modelo y el controlador del caso que estamos tratando. Es por ello que se va a obviar el resto de casos de uso para un **Usuario Registrado**, y solamente se van a exponer los que hemos visto hasta ahora, evitando de esta manera sobrecargar la sección en exceso y con información redundante.

Administrador

Al igual que con los otros dos actores estudiados anteriormente, en esta sección vamos a exponer los diagramas de secuencia más importantes y con más relevancia del sistema para un usuario **Administrador**. Con ello, hemos considerado que deberíamos estudiar los diagramas de clase que representan a los casos de uso de **Cancelar Cuenta**, **Crear Persona**, **Eliminar Persona**, **Añadir Exposición**, **Modificar Exposición** y, a en cuanto a la gestión de las razas, hemos propuesto ilustrar el caso de uso **Crear Raza**, **Modificar Raza** y **Eliminar Raza**.

El primer caso que estudiamos es el de cancelar una cuenta de un usuario, correspondiente al caso de uso **Cancelar Cuenta**:

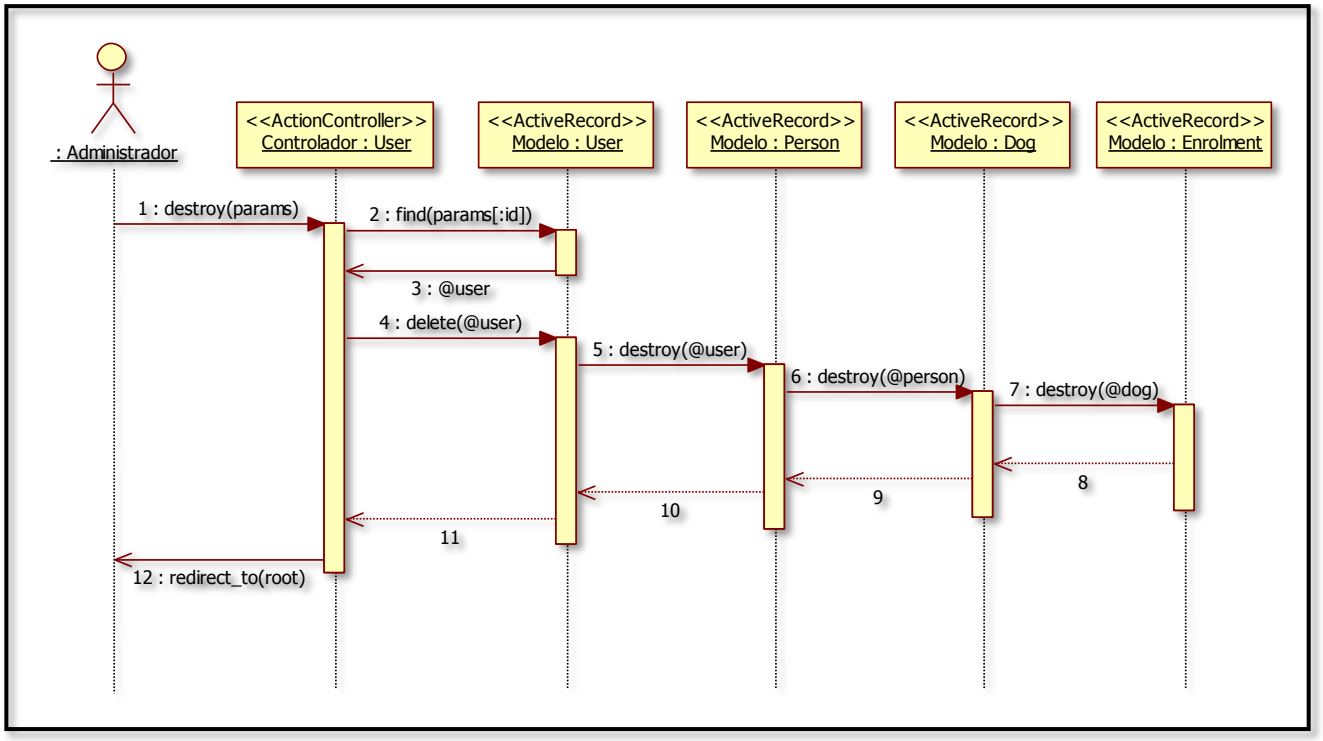


Imagen 8.2.17 – Diagrama de secuencia – Administrador – Cancelar Cuenta

Lo más relevante que podemos observar en el diagrama es que nos encontramos con un *callback* a todos los recursos que el usuario, cuya cuenta se ha borrado, tenía en el sistema. Se borra el modelo de persona, se borran los perros que el usuario poseía, y se borran las inscripciones que los perros tenían en las exhibiciones. Esto es así en nuestro prototipo de sistema. En un sistema totalmente construido y operativo, tal vez interesaría conservar ciertos datos de los usuarios cuándo estos se borren del sistema. Pero para nuestro proyecto en particular, el sistema se comportará de la forma estudiada.

Pasemos a ver el caso de **Crear Persona**:

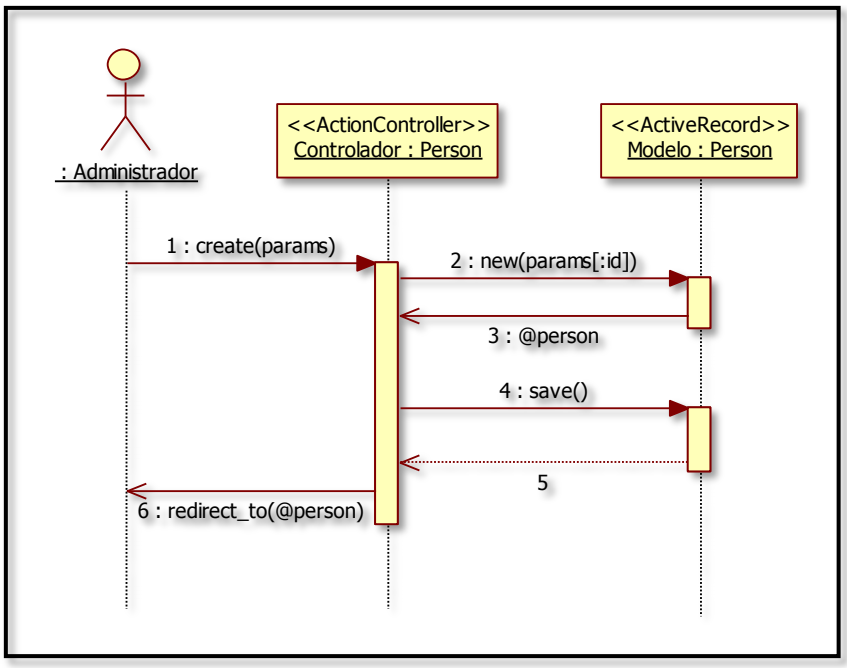


Imagen 8.2.18 – Diagrama de secuencia – Administrador – Crear Persona

Como vemos, no hay nada más interesante que un diálogo típico entre el controlador y el modelo del nuevo registro que queremos crear. Veremos que en el caso de borrar una persona, también pasa lo mismo; el principal motivo es que las funcionalidades que pueden tener estos elementos del sistema no están definidas en este prototipo, sino que estos elementos existen, como hemos nombrado en anteriores partes de este análisis, para futuras ampliaciones y mejoras del sistema. Veamos el diagrama de **Eliminar Persona**:

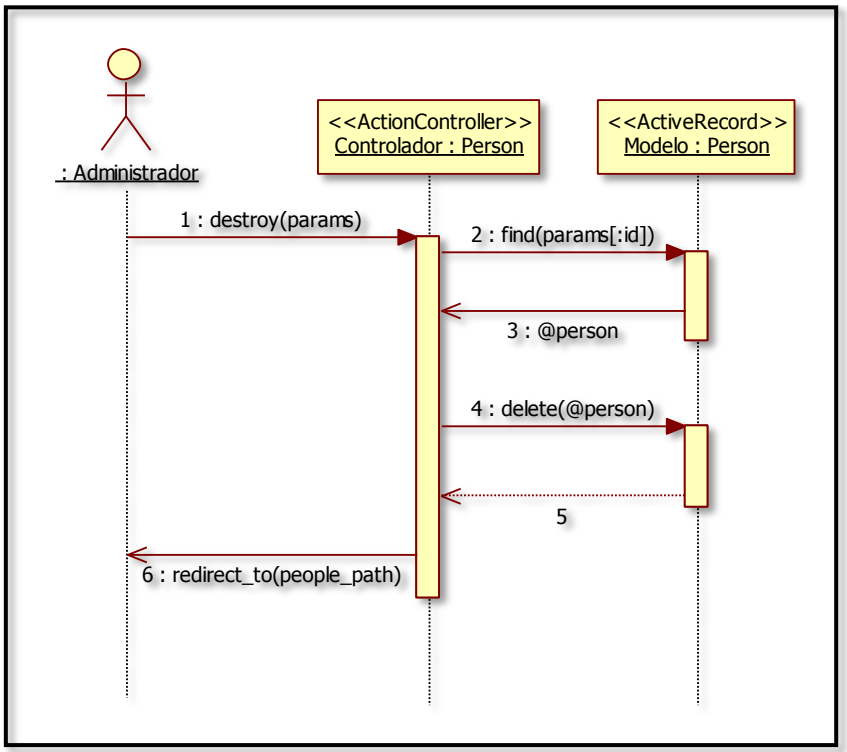


Imagen 8.2.19 – Diagrama de secuencia – Administrador – Eliminar Persona

El siguiente diagrama que vamos a tratar es otro de lo que más relevancia tienen en el sistema. Su estructura no es muy compleja, pero es uno de los modelos base del sistema, el de las exhibiciones. En este caso se representa el diagrama de secuencia de **Añadir Exposición**:

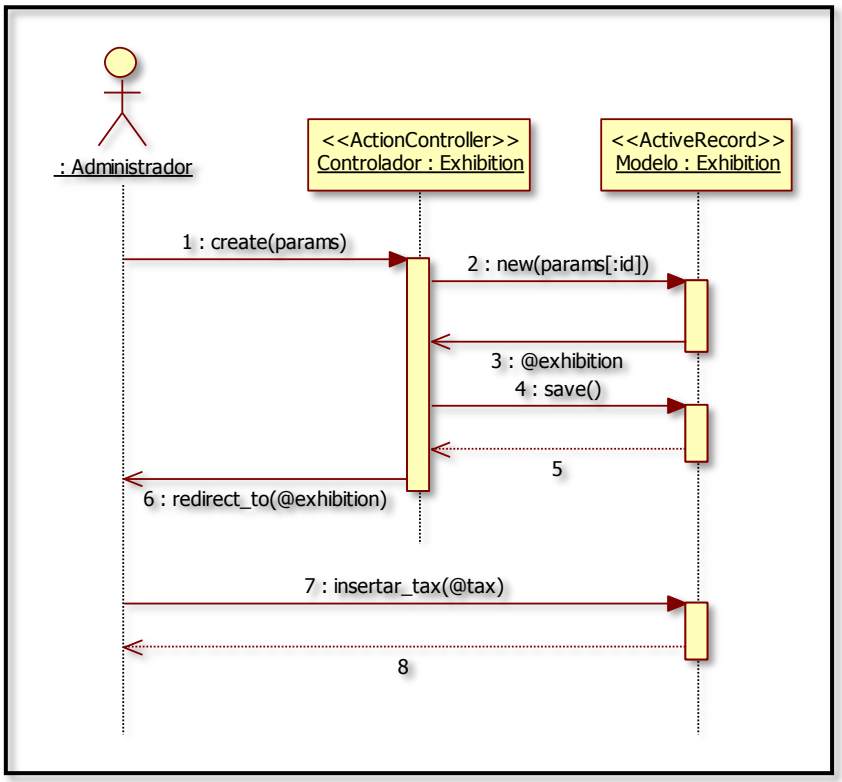


Imagen 8.2.20 – Diagrama de secuencia – Administrador – Añadir Exposición

En el diagrama, lo que más puede llamar la atención es el diálogo directo ente el **Administrador** y el modelo *exhibition.* Esto es debido a lo que comentamos anteriormente respecto a la manera que nuestro sistema tiene de recoger los datos referentes a las tasas de las exposiciones. Actualmente hay que insertar de forma manual la tabla de precios que contiene el coste de inscribir a los perros en la exhibición. Esto se hace siguiendo una estructura de hash en formato JSON determinada, que detallamos en la sección 10.3.

Vamos a mostrar el diagrama representativo del caso **Modificar Exposición**:

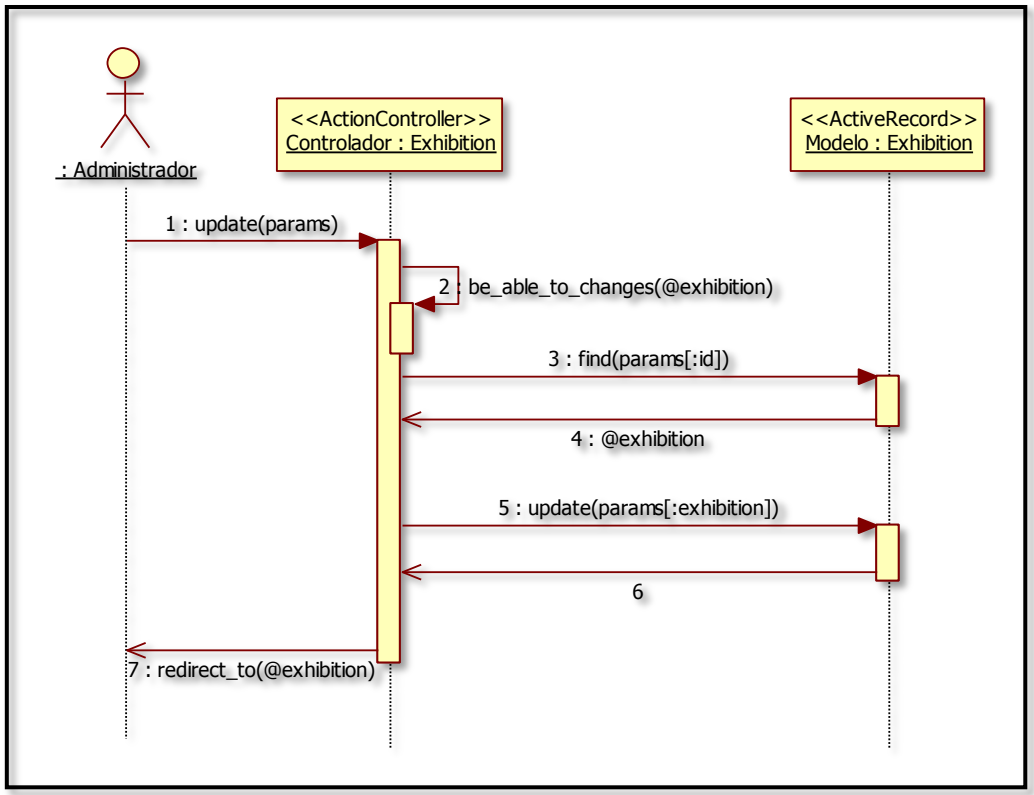


Imagen 8.2.21 – Diagrama de secuencia – Administrador – Modificar Exposición

Observamos que hay un diálogo recursivo entre el propio controlador de la exhibición (al igual que pasará con el caso de uso **Eliminar Exposición**). Su función es impedir que se modifiquen (o borren) exposiciones cuyo plazo de pago ha comenzado. De esta manera se evitan errores en el sistema relacionados con las inscripciones de perros en las exhibiciones y demás problemas que puedan derivarse de exhibiciones cuyo plazo de pago está abierto.

Para finalizar el estudio de los diagramas de secuencia vamos a ver algunos diagramas de secuencia del sistema de razas. Los diagramas que representan cada uno de los casos de uso de todo el sistema de razas son muy similares, por lo que solamente vamos a detallar 3 de ellos en particular: **Crear Raza, Borrar Raza y Actualizar Raza.** Como veremos, no tiene más ciencia que una comunicación entre el modelo y su controlador, por lo que no nos detendremos en detalles, únicamente expondremos los diagramas:

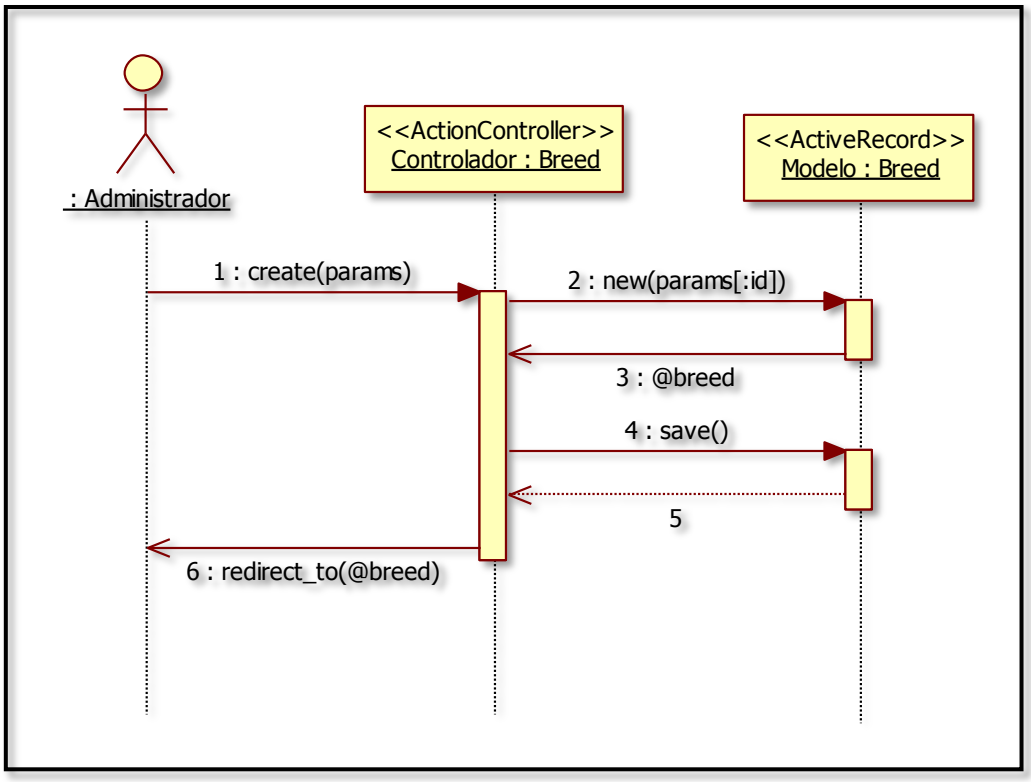


Imagen 8.2.22 – Diagrama de secuencia – Administrador – Crear Raza

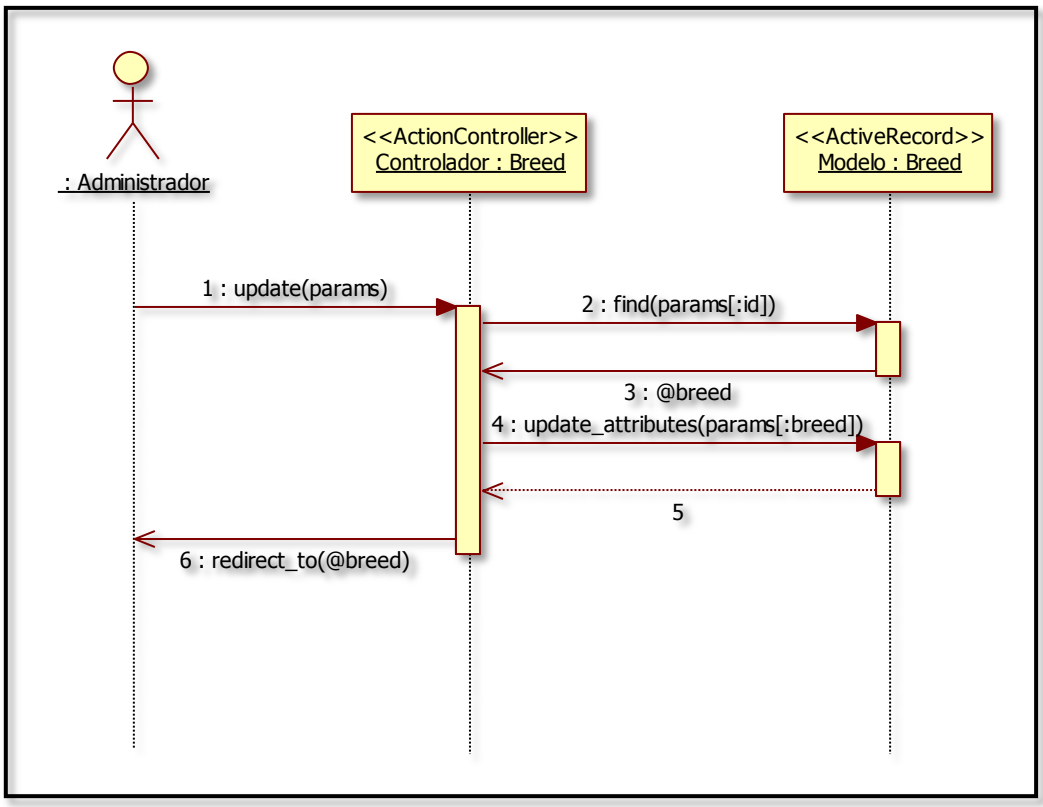


Imagen 8.2.23 – Diagrama de secuencia – Administrador – Modificar Raza

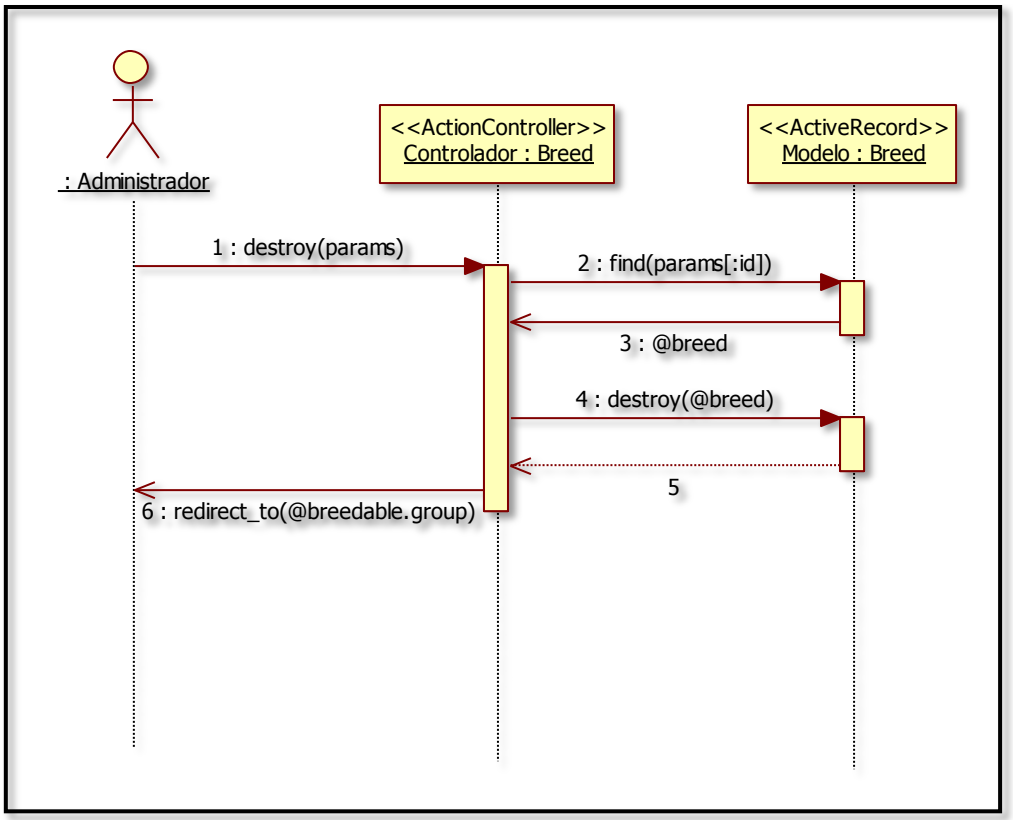


Imagen 8.2.24 – Diagrama de secuencia – Administrador –Eliminar Raza

8.3- Pruebas del software

Como comentamos anteriormente, el desarrollo de nuestro proyecto estará guiado por pruebas (**ATDD**), y, cómo el propio nombre de la metodología indica, lo primero que debemos de hacer son las pruebas que el sistema debe pasar para su correcto funcionamiento. Nuestro proyecto va a estar guiado por iteraciones incrementales, en las que en cada una se irán añadiendo nuevas funcionales al sistema.

Hay cierta controversia entre los expertos del tema respecto a definir en qué parte del documento oficial debe emplazarse el apartado de pruebas del software cuando estamos ante un desarrollo guiado por ellas. Algunos opinan que debe estar en la parte del diseño, otros en la implementación, otros en el análisis, hay opiniones respecto a que deberían estar un apartado distinto a todos ellos… En nuestro caso hemos decidido incluir el apartado de pruebas en la sección de análisis del sistema, ya que consideramos que unas pruebas de aceptación forman parte del análisis del sistema, y que incluso podrían sustituir al apartado de definición de casos de uso. Es cierto que el modelo que estamos tratando es iterativo y que cada vez que se acepta un conjunto de pruebas se realiza la implementación del código que cumple las mismas, pero no por ello deja de tener cierta correspondencia con la parte de análisis del sistema. Por este motivo, pese a encontrarnos en la sección de análisis, durante el estudio de las pruebas realizadas también se nombrarán algunos aspectos en cuanto a la implementación de la aplicación.

Ya se ha especificado que vamos a utilizar **Cucumber**, con el lenguaje **Gherkin** para escribir las pruebas de aceptación. La especificación de los escenarios de pruebas la

encontramos dentro del directorio "*features*" dentro del árbol de directorios, el cuál detallamos en la sección 10.1. En este directorio, podemos ver ficheros con extensión "*.feature*", en los que se escriben la descripción de los **tests de aceptación**. Las definiciones de los pasos de cada uno de estos tests se encuentran en la carpeta "*step_definitions*". En la carpeta "*support*" están los archivos de soporte para las pruebas realizadas (como por ejemplo la definición de los elementos de factory-girl).

Para aclarar un poco los términos que vamos a utilizar durante esta sección del documento, vamos a exponer un pequeño ejemplo de cómo está organizado el sistema de razas, el cual nos servirá de guía a partir de ahora:

> En el mundo de las exhibiciones caninas, cada raza de perro está asignada a un grupo. Hay 10 grupos de razas y cada uno de estos grupos contiene secciones. Éstas a su vez pueden contener sub-secciones, es decir, divisiones de las secciones. Las razas son la raíz de este árbol, y pueden estar asignadas a las secciones o a las sub-secciones. Por ejemplo:
>
> El **Dobermann** es una raza que pertenece a la sub-sección 1 de la sección 1 del grupo 2. El grupo 2 son "**Perros tipo pinscher y schnauzer - Molosoides - Perros tipo montaña y boyeros suizos**". Está divido en 3 secciones:
>
> - **Sección 1:** Tipo Pinscher y Schnauzer
> - **Sección 2:** Molosoides
> - **Sección 3:** Perros tipo montaña y boyeros suizos
>
> A su vez, cada una de estas secciones puede contener las sub-secciones; por ejemplo, para la sección 1 tenemos:
>
> - **Sub-sección 1.1:** Pinscher
> - **Sub-sección 1.2:** Schnauzer
> - **Sub-sección 1.3:** Smoushond
> - **Sub-sección 1.4:** Tchiorny Terrier
>
> En nuestro ejemplo, la raza **Dobermann** sería un Pinscher. Además, las razas de los perros pueden estar divididas en variedades y sub-variedades; por ejemplo el **Dobermann** puede ser:
>
> - **Variedad 1:** Negro con marcas de color rojo-óxido
> - **Variedad 2:** Café oscuro con marcas de color rojo-óxido
>
> Y estas variedades pueden contener sub-variedades: por ejemplo, variedad estándar con sub-variedades: pelo corto, pelo largo y pelo duro.
>
> De esta manera estructuramos el sistema de clasificación de razas de perros. Aunque en apartados anteriores de este documento ha quedado reflejado cómo está organizado el sistema, hemos querido

exponer este ejemplo para que el lector, no experto en la materia, no se encuentre perdido en esta sección del documento, y pueda entender bien las pruebas de aceptación que en ella se detallan.

En los siguientes puntos vamos a describir cada una de las iteraciones en las que se ha dividido la creación de las pruebas, y del sistema en sí. Empezaremos comentando en detalle cada una de las pruebas de aceptación realizadas, y luego simplificaremos la explicación de las pruebas y de las iteraciones para no sobrecargar este apartado de información, en cierta parte, redundante.

8.3.1- Primera iteración – Implementación de los grupos

Esta fase del proyecto se ha dedicado a la primera tarea que debemos plantearnos y que consiste en administrar los grupos que contendrán las razas caninas. Se ha decido implementar esta funcionalidad para ir cogiendo soltura con la metodología de desarrollo que estamos tratando.

Existen 4 aspectos claves que debemos poder hacer con los grupos: Crearlos, mostrarlos, modificarlos y poder borrarlos (**CRUD** – **C**reate, **R**ead, **U**pdate and **D**elete). Lo primero que se ha implementado son las pruebas de correcto funcionamiento que las acciones realizadas a los grupos deberán superar para asegurar que nuestro desarrollo va a ser fiable y sostenible.

A continuación vamos a ver los ficheros de prueba para las acciones que se pueden realizar con los grupos, así como algunos mockups de dichas acciones.

En primer lugar tenemos la función de crear grupos (*creating_groups.feature*):

```
Feature: Creating groups
  In order to have groups to assign sections
  As a user
  I want to create them easily

  Background:
    Given I am on the groups page
    And I follow "New Group"

  Scenario: Creating a group
    When I fill in "Name" with "Group I"
    And I fill in "Description" with "Sheepdogs and Cattle Dogs"
    And I press "Create Group"
    Then I should see "Group has been created."
    And I should be on the group page for "Group I"

  Scenario: Creating a group without a name
    When I press "Create Group"
    Then I should see "Group has not been created."
    And I should see "Name can't be blank"

  Scenario: Creating a group without a description is bad
    When I fill in "Description" with ""
    And I press "Create Group"
    Then I should see "Group has not been created."
    And I should see "Description can't be blank"
```

Al ser el primer fichero de pruebas que vemos, comentaremos los pasos que estamos mostrando. En los sucesivos ficheros "*.feature"* nombraremos solamente los aspectos relevantes en cada uno de ellos.

```
Feature: Creating groups
  In order to have groups to assign sections
  As a user
  I want to create them easily
```

Vemos que inicialmente tenemos una pequeña descripción de lo que tiene que hacer esta prueba: Queremos poder crear grupos fácilmente para poder asignarles secciones. Para ellos vamos a disponer de un **background,** es decir, unas situaciones iniciales que se deben dar para poder realizar la acción. En nuestro caso tenemos definido que nos encontraremos en las página de los grupos y seguiremos la marca "*New Group*".

```
  Background:
    Given I am on the groups page
    And I follow "New Group"
```

En la siguiente imagen podemos ver un mockup de cómo debería ser, a modo de idea, la interfaz relacionada con esta prueba:

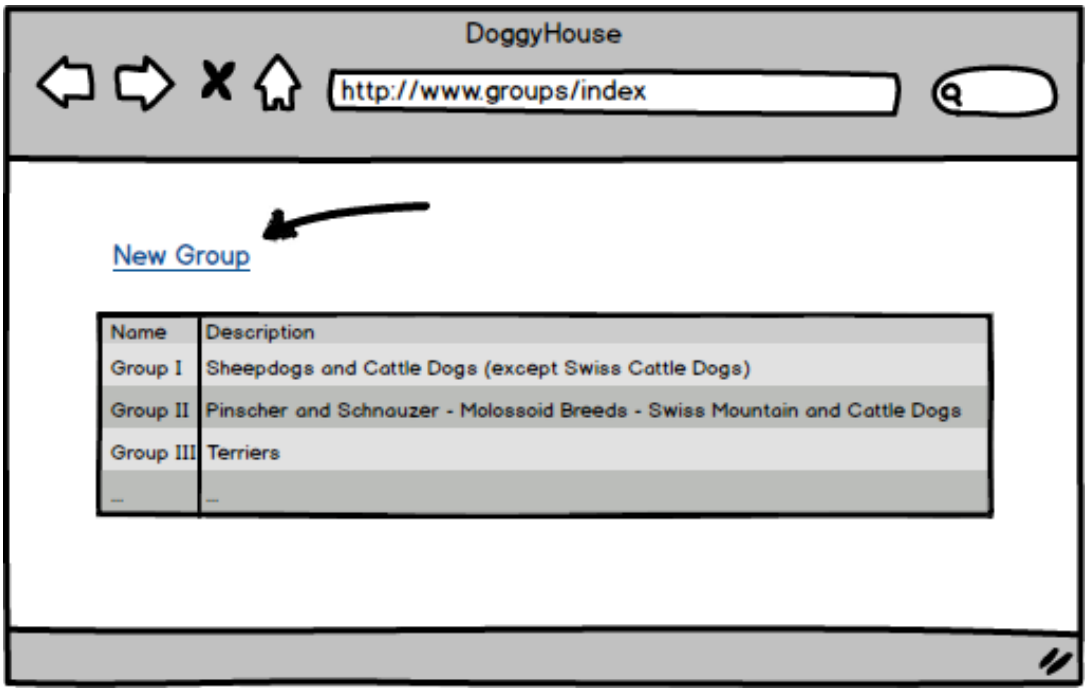


Imagen 8.3.1 – Mockup Groups Page

Una vez realizado el background, entraremos en un caso particular de este test: el escenario de crear el grupo propiamente dicho. En esta prueba en particular tenemos definidos dos escenarios de creación de grupos: Un escenario de crear un grupo "correctamente" y otro escenario de fallo, en el que no se define un nombre para el grupo en cuestión.

El primer escenario que la prueba debe pasar nos indica que cuando rellenamos el nombre de grupo, la descripción, y pulsamos sobre "*Create Group*", deberíamos ver un mensaje que nos indique que el grupo ha sido creado, y deberíamos ser redireccionados hacia la página del grupo creado. Veamos, cómo antes, la porción de código que describe esta prueba, así como los mockups relacionados con la misma.

```
Scenario: Creating a group
    When I fill in "Name" with "Group I"
    And I fill in "Description" with "Sheepdogs and Cattle Dogs"
    And I press "Create Group"
    Then I should see "Group has been created."
    And I should be on the group page for "Group I"
```

Este sería un mockup de relleno de los campos de un grupo:

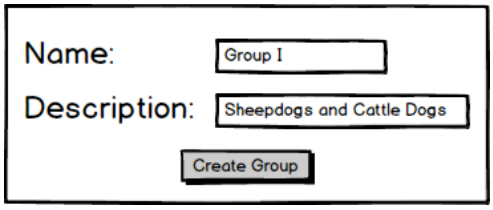


Imagen 8.3.2 – Mockup Creating a group

Y este sería el mockup de la página principal del grupo etiquetado cómo "Group I".

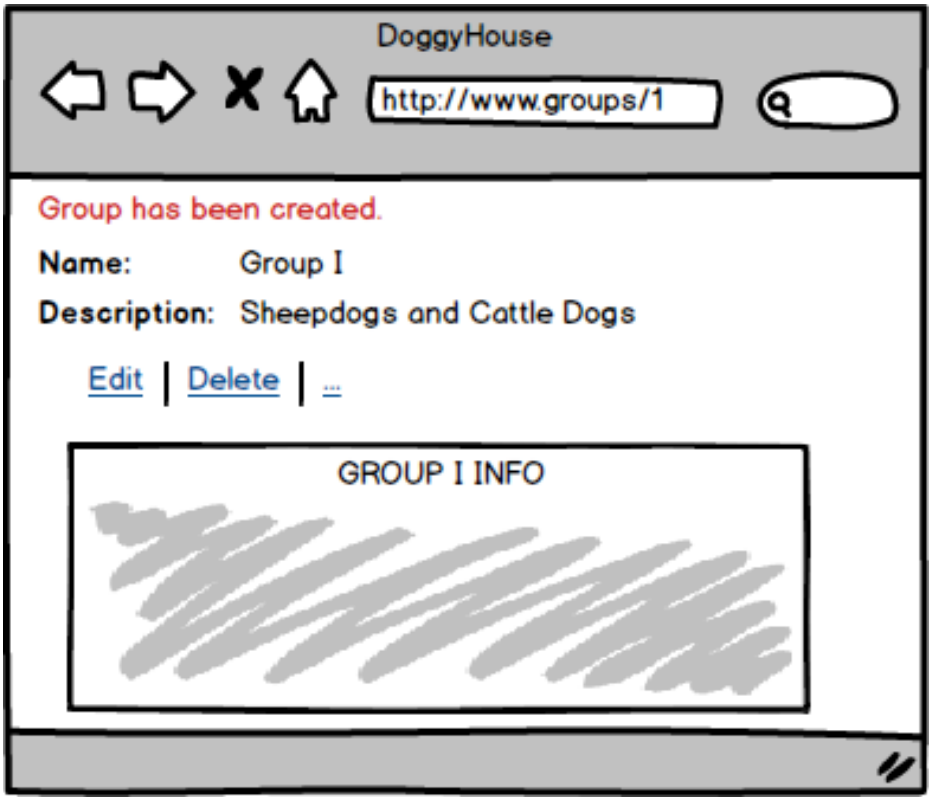


Imagen 8.3.3 – Mockup página Group I

El segundo escenario que nos encontramos está orientado a no permitir crear un grupo sin haberle asignado un nombre. Cuándo presionamos el botón de "Create group", y el

nombre está en blanco, debe salir una advertencia de que el grupo no ha sido creado porque el nombre no pueden estar en blanco.

```
Scenario: Creating a group without a name
  When I press "Create Group"
  Then I should see "Group has not been created."
  And I should see "Name can't be blank"
```

Veamos también ambos mockups:

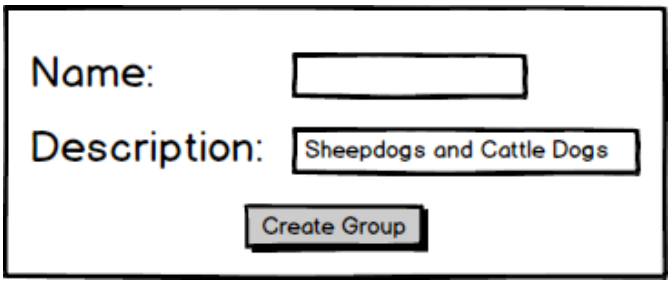


Imagen 8.3.4 – Mockup Creating a group without a name

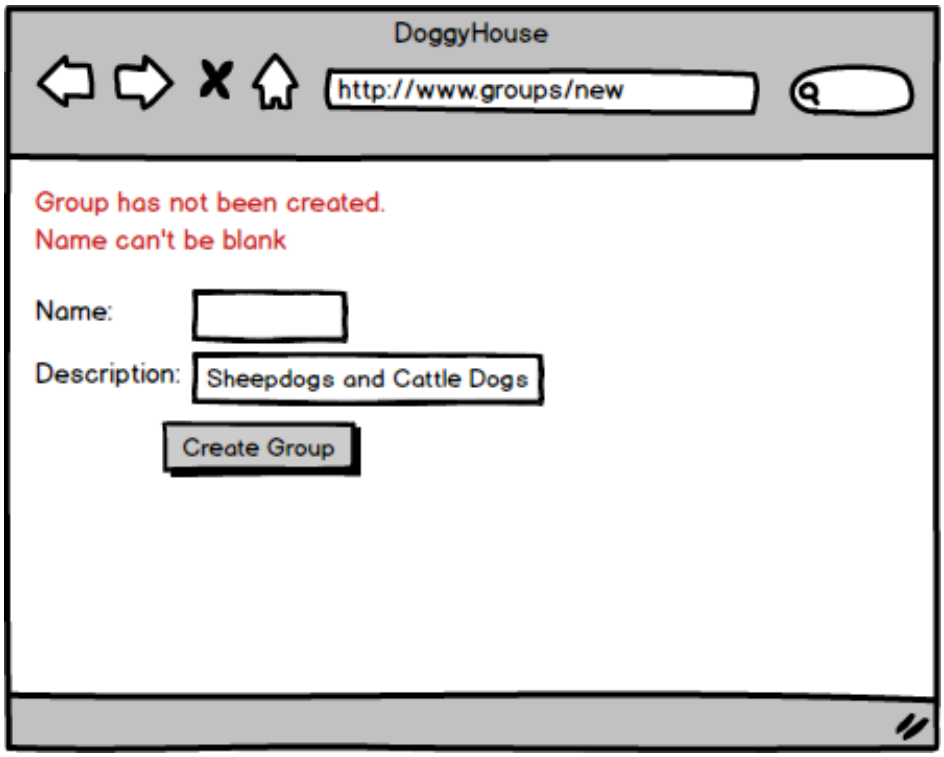


Imagen 8.3.5 – Mockup Creating a group without a name

Un caso similar ocurre con la parte de la descripción:

```
Scenario: Creating a group without a description is bad
  When I fill in "Description" with ""
  And I press "Create Group"
  Then I should see "Group has not been created."
  And I should see "Description can't be blank"
```

Y de esta manera se darían por concluidas las acciones de testeo a la hora de crear nuevos grupos para que sean incluidos en el sistema.

El siguiente fichero que vamos a ver es el referente al visionado de los grupos en cuestión (*seeing_groups.feature*).

```
Feature: See groups
  In order to view groups
  As a user
  I want to see them on the groups page

  Background:
    There are groups with a name and a description
    Given groups with these entries:
      |GROUP ID|Name       |Description                                        |
      |1       |Group I    |Sheepdogs and Cattle Dogs (except Swiss Cattle Dog)|
      |2       |Group III  |Terriers                                           |
      |3       |Group IX   |Companion and Toy Dogs                             |

  Scenario: A visit wants see what groups are defined
    When the standards and nomenclature page is visited
    Then the standards and nomenclature page should content:
      |GROUP ID|Name       |Description                                        |
      |1       |Group I    |Sheepdogs and Cattle Dogs (except Swiss Cattle Dog)|
      |2       |Group III  |Terriers                                           |
      |3       |Group IX   |Companion and Toy Dogs                             |

  Scenario: A visit wants see a specific group properties
    When the standards and nomenclature page is visited
    And I follow "Group I"
    Then I should be on the group page for "Group I"
```

En este caso, lo más relevante que podemos detallar es la descripción de la tabla que aparece en el **Background** del cuadro que hay sobre este párrafo. Simplemente estamos indicando que, tiendo un conjunto de grupos con sus correspondientes descripciones (lo cual está representado a modo de tabla dentro de la sentencia "*Given*"), si visitamos la página "*standars and nomenclature*" (es decir, la página principal de grupos, cuyo mockup está en Imagen 8.3.1), debemos ver una tabla exactamente igual que la que está dentro del "*Given*".

El otro escenario que se ve no tiene nada en especial; lo que viene a decir es: Si alguien desea ver las propiedades específicas dentro de un grupo, si clicamos en el link "*Group I*", esperamos ir directamente a la página particular de dicho grupo.

Ahora vamos a ver las pruebas realizadas cuando tratamos de modificar la información de alguno de los grupos existentes en el sistema (*editing_groups.feature*):

```
Feature: Editing Groups
  In order to update a group information
  As a user
  I want to be able to do that through an interface

  Background:
    Given there is a group called "Group I"
    And I am on the groups page
    When I follow "Group I"
    And I follow "Edit group"

  Scenario: Updating a group
    When I fill in "Name" with "Group I beta"
    And I fill in "Description" with "Description beta"
    And I press "Update Group"
    Then I should see "Group has been updated."
    And I should be on the group page for "Group I beta"

  Scenario: Updating a group with invalid name is bad
    When I fill in "Name" with ""
    And I press "Update Group"
    Then I should see "Group has not been updated."

  Scenario: Updating a group with invalid description is bad
    When I fill in "Description" with ""
    And I press "Update Group"
    Then I should see "Group has not been updated."
```

Si hemos entendido los ejemplos vistos anteriormente, podemos ver que estos escenarios de prueba no son nada complejos, y no hará falta entrar en mucho detalle.

En la sección "**Background**" se define que existe un grupo (Group I) y además estamos en la página de visionado de los grupos (Imagen 8.3.1). Cuándo cliqueamos en este grupo y en "Edit Group" (cuya página representativa está en el mockup de Imagen 8.3.3), se plantean una serie de escenarios de prueba descritos a continuación:

El primero de ellos ("*Updating a group*") nos indica que, cuándo rellenamos los campos "*Name*" y "*Description*" y presionamos el botón "*Update Group*", deberíamos ver un mensaje de aceptación ("*Group has been updated*") y deberíamos haber sido redireccionados a la página del grupo que hemos modificado.

```
Scenario: Updating a group
    When I fill in "Name" with "Group I beta"
    And I fill in "Description" with "Description beta"
    And I press "Update Group"
    Then I should see "Group has been updated."
    And I should be on the group page for "Group I beta"
```

Veamos una maqueta de lo que realmente queremos visualizar con este escenario:

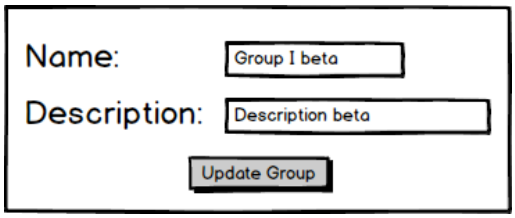


Imagen 8.3.6 – Mockup Updating a group

Imagen 8.3.7 – Mockup Show updating a group

Los otros dos escenarios mostrados arriba son escenarios similares; ambos son escenarios de no aceptación de los cambios cuándo el campo de nombre, o el campo de descripción, es incorrecto. Cuando esto ocurra, deberíamos poder ver una estructura de página similar a esta:

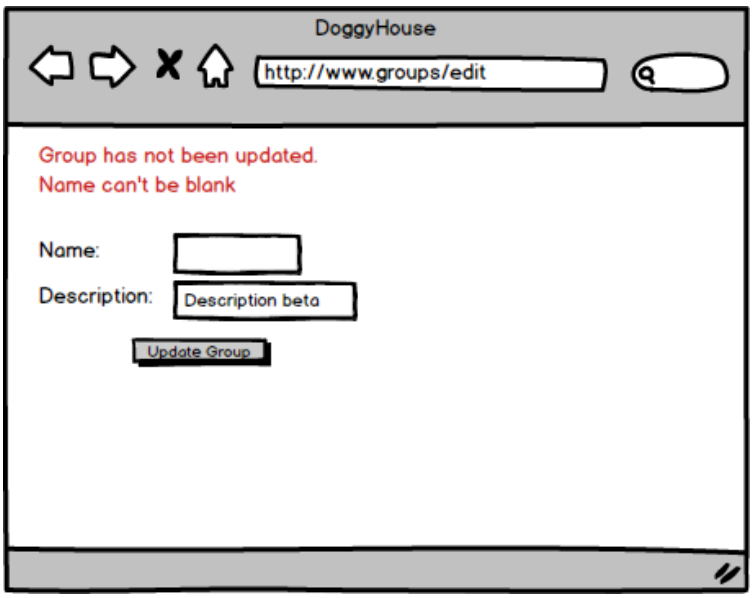


Imagen 8.3.8 – Mockup Updating a group without a name

Y para finalizar esta primera iteración se implementaron pruebas que confirmarán el correcto borrado de los grupos. Estas pruebas se pueden encontrar en el fichero "*deleting_groups.feature*", cuyo contenido es el siguiente:

```
Feature: Deleting groups
  In order to remove any group
  As user
  I want to make it disappear

  Background:
    Given there is a group called "Group I"
    And I am on the groups page
    When I follow "Group I"

  Scenario: Deleting groups
    When I follow "Delete group"
    Then I should see "Group has been deleted."
    And I should not see "Group I"
```

En este caso nos enfrentamos a unas comprobaciones poco complejas en las que el sector del "**Background**" nos resulta muy familiar, ya que se consideran prácticamente las mismas premisas que utilizamos en "*editing_groups.feature*".

El único escenario que tenemos en este caso es la comprobación que, al pulsar sobre la opción de borrar el grupo, dicho grupo deberá de dejar de aparecer en nuestro sistema. Va a ser muy conveniente, en este caso, pedir al usuario la confirmación de borrado, cuyo test debería realizarse con Jasmine, trabajo que se propondrá como posible ampliación del proyecto.

Una vez definidas las pruebas que deberá pasar el sistema a la hora de hacer operaciones relacionadas con los grupos, se procedió a la implementación de los grupos en la etapa de desarrollo. Se creó todo el sistema en base a estas pruebas, y se comprobó, que efectivamente, las pruebas pasaban y el sistema actuaba conforme los requerimientos del "cliente".

8.3.2- Segunda iteración – Implementación de las secciones

En la segunda iteración de la creación de nuestro sistema hemos tratado las pruebas de las secciones que va a contener cada grupo. Dichas secciones son, por así decirlo, las divisiones específicas en las que se fraccionará cada grupo.

Las secciones tendrán las mismas operaciones básicas que los grupos (**CRUD**), con la particularidad de que cada sección deberá estar relacionada con un único grupo. Por tanto, podremos crear, eliminar, modificar y mostrar las secciones de cada grupo. Como hicimos anteriormente, vamos a mostrar los ficheros de prueba que se han realizado para controlar el correcto funcionamiento de la administración de las secciones. Las pruebas realizadas con las secciones son similares a las pruebas realizadas con los grupos, así que no entraremos en mucho detalle.

El primer fichero de pruebas que vamos a ver es el de la creación de las secciones (*creating_sections.feature*):

```
Feature: Creating sections
  In order to create sections for groups
  As a user
  I want to be able to select a group and do it

  Background:
    Given there is a group called "Group I"
    And I am on the groups page
    When I follow "Group I"
    And I follow "New section"

  @done
  Scenario: Creating a section
    When I fill in "Name" with "1"
    And I fill in "Description" with "Sheepdogs"
    And I press "Create Section"
    Then I should see "Section has been created."
    And I should be on the section page for "1"

  @done
  Scenario: Creating a section without a name
    When I fill in "Name" with ""
    And I press "Create Section"
    Then I should see "Section has not been created."
    And I should see "Name can't be blank"

  @done
  Scenario: Creating a section without a description
    When I fill in "Description" with ""
    And I press "Create Section"
    Then I should see "Section has not been created."
    And I should see "Description can't be blank"
```

En comparación con las pruebas de los grupos, lo más relevante que podemos destacar de esta prueba, es la sentencia "*@done*" que vemos repetida 3 veces en el fichero. Hemos evolucionado en la creación de las pruebas, y ahora utilizamos estos tags. El tag, precedido del símbolo "@" nos sirve para ir ejecutando el fichero de pruebas por partes. De modo que si escribimos el comando para ejecutar la prueba "*cucumber –tag @done rutadelfichero*", solamente se ejecutarán los escenarios precedidos del tag "@done" (muy útil para realizar pruebas parciales de cada *feature* por separado). En este caso, el *done*, nos indica que el test está terminado y listo para realizarse.

Las interfaces de las operaciones que podemos realizar con las secciones son idénticas a las de los grupos, por lo tanto, en este caso, no vamos a hacer uso de mockups para guiar nuestra explicación (para más detalle se pueden consultar los mockups de los grupos vistos en el apartado anterior).

La siguiente implementación de pruebas que vamos a ver son las referentes al mostrado de las secciones (*viewing_sections.feature*):

```
Feature: Viewing sections
  In order to view the sections for a group
  As a user
  I want to see them on that group's page

  Background:
    Given there is a group called "Group I"
    And there are sections for this group:
      |SECTION ID|GROUP ID|Section number|Description                    |
      |1         |1       |1             |Sheepdogs                      |
    Given there is a group called "Group II"
    And there are sections for this group:
      |SECTION ID|GROUP ID|Section number|Description                    |
      |3         |2       |1             |Large and medium-sized Terriers|

    Given I am on the groups page

  Scenario: Viewing sections for a given group
    When I follow "Group I"
    Then I should see "Sheepdogs"
    And I should not see "Large and medium-sized Terriers"
    When I follow "Sheepdogs"
    Then I should see "1" within "#sections h2"
    And I should see "Sheepdogs"

    When I follow "Standards and nomenclature page"
    And I follow "Group II"
    Then I should not see "Sheepdogs"
    And I should see "Large and medium-sized Terriers"
    When I follow "Large and medium-sized Terriers"
    Then I should see "1" within "#sections h2"
    And I should see "Large and medium-sized Terriers"
```

En esta prueba vemos un único escenario en el que se recoge el caso a probar propiamente dicho. Como siempre, lo primero que se define es un background, en el que especificamos dos grupos distintos, con sus respectivas secciones (Group I y Group II). A continuación entramos en el escenario. No tiene demasiada particularidad, simplemente estamos concretando que si entramos en la página del "*Group I*", debemos de ver las secciones particulares de este grupo y no las del "*Group II*", y si entramos en la página del "*Group II*", deberemos ver las secciones de dicho grupo, y no las del "*Group I*". Cuando cliqueemos en las respectivas secciones se deberá ver la información oportuna (Número de sección y descripción), cuyo número de sección deberá estar dentro del tag "*sections*" y con una cabecera "*h2*".

A continuación vamos a ver el fichero dónde se han implementado las pruebas referentes a la modificación de una sección ya creada (*editing_sections.feature*):

```
Feature: Editing Section
  In order to alter section information
  As a user
  I want a form to edit the sections

  Background:
    Given there is a group called "Group I"
    And there are sections for this group:
      |SECTION ID|GROUP ID|Section number|Description|
      |1         |1       |1             |Sheepdogs  |
    And I am on the groups page
    When I follow "Group I"
    And I follow "Sheepdogs"
    And I follow "Edit Section"

  @done
  Scenario: Updating a section
    When I fill in "Name" with "beta"
    And I fill in "Description" with "Description beta"
    And I press "Update Section"
    Then I should see "Section has been updated."
    And I should be on the section page for "beta"
    And I should see "beta" within "#sections h2"
    But I should not see "1"

  @done
  Scenario: Updating a section without a name
    When I fill in "Name" with ""
    And I press "Update Section"
    Then I should see "Section has not been updated."
    And I should see "Name can't be blank"

  @done
  Scenario: Updating a section without a description
    When I fill in "Description" with ""
    And I press "Update Section"
    Then I should see "Section has not been updated."
    And I should see "Description can't be blank"
```

Como estamos habituados a ver en este apartado, las pruebas relativas al editado de las secciones no son muy diferentes a las mostradas en la parte de los grupos. Observamos un pequeño background en el que se define un grupo con una sección específica y a continuación 3 escenarios: Uno de actualización en el que se define que todo ocurre correctamente – "*Updating a section*"; otro en el que intentamos actualizar una sección sin especificar un nombre – "*Updating a section without a name*"; y otro en el que se impide modificar una sección sin asignarle una descripción – "*Updating a section without a description*". Ninguno de los 3 escenarios tiene alguna peculiaridad que merezca explicación; tal vez profundizar un poco más en el primero de todos, que es el más largo: Básicamente lo que nos refleja es que cuando modificamos un campo de las secciones (nombre o descripción), al cliquear sobre el botón de actualizar, deberíamos estar en la página principal de dicha sección y se deben ver los cambios que hemos realizado.

La última prueba implementada respecto a las secciones (y con ella cerramos esta segunda iteración) es la referente al borrado de las mismas (*deleting_sections.feature*).

Veremos un fichero con una prueba bastante sencilla, con un solo escenario, del que no hará falta mayor explicación:

```
Feature: Deleting Sections
  In order to remove sections
  As a user
  I want to press a button and make it disappear

  Background:
    Given there is a group called "Group I"
    And there are sections for this group:
      |SECTION ID|GROUP ID|Section number|Description|
      |1         |1       |1             |Sheepdogs  |
    And I am on the groups page
    When I follow "Group I"
    And I follow "Sheepdogs"

  @done
  Scenario: Deleting a section
    When I follow "Delete Section"
    Then I should see "Section has been deleted."
    And I should be on the group page for "Group I"
    And I should not see "Sheepdogs"
```

Al igual que ocurrió en el caso anterior, una vez realizadas las pruebas que debe superar el sistema, y que se han corroborado con el cliente, procedimos al desarrollo del código de esta iteración. Se crearon los controladores, modelos, vistas y demás elementos necesarios para la implementación, y se comprobó que, correctamente, los tests pasaban y se ejecutaban sin fallo; por lo que estábamos en la certeza de que lo que estábamos implementando funcionaba como tenía que funcionar según los requerimientos del cliente.

8.3.3- Iteraciones sucesivas

Siguiendo la misma rutina que hemos visto hasta ahora, se fueron implementando las pruebas de aceptación que debía cumplir nuestro código para que el sistema funcionase como el cliente quería en cada iteración: Se realizaban reuniones semanales o quincenales con el cliente y se iba concretando cómo debía comportarse el sistema en cada uno de los casos. Lo que se hacía era definir las pruebas que el sistema debía pasar y se le mostraban para conocer su opinión. Una vez que las pruebas estaban aceptadas, se procedía a la implementación del código que debía satisfacer estos requerimientos.

Como ya hemos estudiado unos casos comunes de pruebas de aceptación, a continuación no vamos a entrar en tanto detalle como hemos visto hasta ahora; lo que haremos será describir brevemente las siguientes iteraciones realizadas durante el desarrollo del proyecto, remarcando lo más significativo en cada una de ellas.

8.3.3.1- Sub-Secciones

Después de definir las secciones, definimos las sub-secciones, cuyas pruebas serán algo muy similar a las secciones. También tienen un modelo **CRUD** y la definición de las pruebas son bastante parecidas a las otras que hemos visto.

8.3.3.2- Razas

Las razas de los perros se trataron en la cuarta iteración. La peculiaridad de las razas es que podrían pertenecer a una sección o a una sub-sección, por lo que el *background* de las pruebas debía ser un poco más detallado, para comprobar que el sistema cumplirá con lo que se espera en todos los casos posibles.

8.3.3.3- Variedades de razas

En la siguiente iteración se trataron las variedades de cada raza. Pruebas de muy parecido a las realizadas con los grupos, secciones y sub-secciones.

8.3.3.4- Sub-Variedades de razas

Al igual que el apartado anterior, en esta iteración se trataron unos tests similares a otros que ya habíamos realizado.

Hay que aclarar que, aunque las pruebas de varias iteraciones eran similares a las ya realizadas, cada prueba del sistema es independiente a la de las otras iteraciones, y debe implementarse de acuerdo a ciertos criterios particulares para cada una, no convirtiéndose en algo estándar las pruebas.

8.3.3.5- Fichero semilla

En este punto, teníamos definida la parte de clasificación de las razas. Esta parte está estandarizada en Europa según la **FCI** y como el modelo de estándar no suele cambiarse (y si se cambia, son pequeños cambios), se decidió introducir los datos actuales de la **FCI** en un fichero semilla. Para comprobar que todo estaba correcto, se hicieron pruebas unitarias con RSpec, las cuales no serían de relevancia para el cliente, pero sí para el sistema y la comprobación de que los datos estaban introducidos correctamente. En la sección 10.2.2 veremos algunos de estas pruebas.

8.3.3.6- Usuarios

La iteración de los usuarios consistió realmente en dos iteraciones. En esta primera iteración reflejamos las acciones que un usuario del sistema podía hacer (el loguearse correctamente, registrarse en el sistema, cerrar la sesión…). La siguiente iteración se describe a continuación y tiene que haber con la administración de los mismos.

8.3.3.7- Personas

Se quería tener en el sistema el perfil de *"persona"* para trabajos futuros. El cliente quería poder en un futuro el crear perfiles en el sistema que no fueran usuarios (por ejemplo un juez de una exhibición, un organizador, un responsable…) Estos perfiles realmente no eran usuarios del sistema, por lo que no los debíamos meter en el mismo grupo de la iteración anterior. En esta iteración se comprobaron las capacidades que debía cumplir el sistema para permitir esta característica.

A partir de aquí la definición de las comprobaciones empezaba a cambiar. Los elementos que había que comprobar también cumplían con un modelo **CRUD**, pero las comprobaciones que había que hacer cambiaban y se complicaban más. El sistema iba conteniendo más elementos, y si bien de cara al cliente esto no suponía (y no debía suponer) cambio alguno, la implementación unitaria interna de cada prueba iba complicándose.

En este punto hubo que hacer una refactorización importante de las pruebas, ya que se añadía el perfil de administrador, y acciones como las de creado, borrado y actualizado de los grupos, secciones, razas… Sólo debían ser llevadas a cabo por un administrador. En la sección 8.3.4 vamos a ver un pequeño ejemplo de esta refactorización en cuanto a los grupos.

8.3.3.8- Perros

Esta iteración constaba de comprobar que el sistema daba los resultados esperados conforme integrábamos el concepto de perros en él. Las pruebas de aceptación de este caso fueron un poco más complejas, e incluso su tuvieron que utilizar algunas utilidades de Cucumber que hasta ahora no se habían utilizado.

Por un lado, los escenarios de Background se expandían un poco más. Cada vez se debía abarcar más contenido previo al test de comprobación propiamente dicho. Por otro lado, en esta parte del proyecto se deseaban introducir elementos AJAX y JavaScript, y al no incluir en nuestra aplicación una herramienta de testeo específica para testear estas técnicas, el testeo de la correcta funcionalidad de esta parte se iba a complicar. Sin embargo, estudiando Cucumber y sus capacidades, se descubrió que puede, en cierta parte, cubrir algunas capacidades de testeo para JavaScript: Con la utilización del tag "*@javascript"* antes de los tests que poseen ciertas características JS, Cucumber utilizará un sistema preparado para procesas peticiones web JS, en lugar de su sistema predeterminado no capacitado para JavaScript. Por tanto pudimos cubrir ciertas características de programación en JavaScript mediante pruebas de aceptación.

8.3.3.9- Exhibiciones

La iteración que se trato a continuación fue la referente al modelado del concepto de las exhibiciones. Se realizaron todas las pruebas que el sistema debía cumplir con respecto a este concepto, y, una vez aceptadas por el cliente, se pasó a su implementación. Si bien las pruebas de aceptación no tienen mayor interés que las que se han realizado anteriormente, la parte de desarrollo y las pruebas unitarias se iban complicando cada vez más.

Se estudió una nueva característica de Cucumber: la posibilidad de definir fechas concretas para las pruebas. Mediante la utilización ciertas sintaxis y funciones de Cucumber, pudimos definir un "*viaje en el tiempo*", es decir, que podríamos definir, en los tests, sentencias del tipo:

```
Given today is "10-12-2013"
```

Esta sentencia es muy interesante para comprobar casos, como por ejemplo, de exhibiciones que ya han concluido o para comprobar que no se puedan modificar exposiciones bajos ciertos criterios de fechas.

En esta iteración se decidió, para no complicar todavía más el sistema, que la definición de los precios de inscribir perros en exposiciones no se realizase por vía web, sino que fuera de forma manual mediante consola Rails. Además existe la posibilidad de que los precios ni si quiera estén definidos cuándo la exhibición se inserta en el sistema, por lo que será labor del administrador definir los precios de manera manual. En la sección 10.3 se hará una descripción de la sintaxis utilizada para la inserción de éstos. De cara al sistema totalmente

acabado y operativo, esta característica debería ser cubierta, y permitir la inserción de los mismos por medio de un formulario web.

8.3.3.10- Enrolments

Una vez definidos los usuarios, los perros y las exhibiciones, pudimos entrar en la labor de definir la manera en que un usuario va a inscribir a sus perros en las exhibiciones. En el argot de este mundo, a esta operación se le considera enrolar un perro en la exhibición.

Un caso particular de esta iteración lo vamos a detallar en el ANEXO II de este documento. Es una de las pruebas de aceptación más extensas que se han tenido que realizar, y hemos querido detallar uno de los ficheros que componen las pruebas de aceptación de los *enrolments* porque, observando este fichero, se puede ver la gran evolución que hay con respecto a las primeras pruebas realizadas en el proyecto. Así mismo, en el ANEXO III, podremos ver el fichero de implementación de las pruebas realizadas sobre los *enrolments* del sistema. Del mismo modo que lo anterior, se desea poder comparar estas implementaciones con las mostradas en la sección 10.2.1, observando la diferencia de complejidad entre la implementación de una las primeras pruebas y la de una de las últimas.

8.3.3.11- Pagos

La última iteración con respecto a añadir funcionalidades al sistema consistió en definir las capacidades que debía cubrir la opción del pago de inscripciones de perros en las exhibiciones. Como hemos nombrado en la definición de contenidos de este mismo apartado, vamos a permitir pagos únicamente mediante ingreso en cuenta bancaria. De este modo, la manera que tendremos de asegurarnos que el pago se ha realizado correctamente, es mediante la carga del justificante de pago en nuestro sistema. En esta iteración se comprobó que el sistema actuará de manera correcta a las operaciones referentes a los pagos y la carga de los ficheros: operaciones de clientes, operaciones de administración, visionado…

8.3.3.12- Arreglos y seguridad

Una vez definidas todas las pruebas y desarrollado un sistema que pase estas pruebas y se comporte de acuerdo a lo que el cliente espera y quiere, se hicieron una serie de iteraciones finales para comprobar el buen funcionamiento del sistema en cuanto a otras características. En esta iteración tratamos ciertos arreglos en el sistema y comprobaciones de seguridad: Cierta refactorización de código, asegurarnos que todo encaja perfectamente, renombrar ciertas características o ficheros de la aplicación, mejora en aspectos de seguridad…

Además, también se decidió comprobar mediante pruebas que no se producían ciertas violaciones de seguridad básicas que el sistema debía tener: evitar cargas indebidas de ficheros, imposibilitar el acceso de usuarios no registrados a áreas de usuarios registrados, controlar el acceso a áreas de administración, comprobar que no se posibilita el acceso de usuarios o administrador a áreas no permitidas para los mismos…

8.3.3.13- Producción

Cuando estuvo todo dispuesto y el esqueleto de la aplicación desarrollado y testeado, se introdujeron las vistas de la aplicación mediante Bootstrap y se volvió a hacer una refactorización del código para que todo quedase correctamente definido. Hubo algunas

incompatibilidades que se tuvieron que solventar, pero no hubo nada grave que impidió al sistema actuar conforme a lo que estaba esperado y lo que las pruebas dictaban.

En esta iteración también se preparó el sistema para subirlo a la plataforma Heroku. También nos encontramos con ciertas peculiaridades que se decidieron obviar en un inicio (como que Heroku no acepta una gestión basada en SQlite3, o la integración de un sistema de mailing) y que tuvimos que arreglar en esta iteración.

8.3.3.14- Pruebas finales tipo beta

Una vez que el sistema estuvo en producción, se realizaron pequeñas iteraciones con pruebas tipo beta, que fueron realizadas por los usuarios en su entorno de trabajo y sin observaciones añadidas.

8.3.4- Ejemplos de refactorizaciones

A medida que el proyecto avanzaba, se iban retomando iteraciones ya acabadas para su refactorización. En este apartado veremos algunas de estas refactorizaciones, como por ejemplo, la refactorización que se hizo en la parte de los **grupos** cuando se añadieron los roles de usuarios (registrado, invitado y administrador). Obviamente, se han hecho refactorizaciones a lo largo de todo el desarrollo en muchas de las iteraciones que ya se habían terminado, pero para no sobrecargar este documento, vamos a ver solamente, a modo de ejemplo, los cambios que se hicieron en las pruebas respecto a la primera iteración, la de los grupos.

En la sección 8.3.1 vimos los ficheros de pruebas de aceptación originales que se crearon en la primera iteración del proyecto con respecto a los **grupos**. Conforme el proyecto avanzaba, se tuvieron que hacer refactorizaciones de estas pruebas; vamos a estudiarlas:

8.3.4.1- Creating Groups

Vamos a mostrar el primer fichero visto en la sección anterior, el de “creating groups”:

```
Feature: Creating groups
  In order to have groups to assign sections to
  As a user
  I want to create them easily when I'm an admin

  Background:
    Given there are the following users:
      | email              | password | admin  |
      | admin@testing.com  | password | true   |
      | user@testing.com   | password | false  |

    Given I am signed in as "admin@testing.com"
    Given I am on the groups page
    And I follow "New Group"

  @done
  Scenario: Creating a group as admin
    When I fill in "Name" with "Group I"
    And I fill in "Description" with "Sheepdogs and Cattle Dogs"
    And I press "Create Group"
    Then I should see "Group has been created."
    And I should be on the group page for "Group I"
Continúa en la página siguiente...
```

```
  @done
  Scenario: Creating an existing group is bad
    Given there is a group called "Group I"
    When I fill in "Name" with "Group I"
    And I fill in "Description" with "Sheepdogs and Cattle Dogs"
    And I press "Create Group"
    Then I should see "Group has not been created."
    And I should see "Name has already been taken"

  @done
  Scenario: Creating a group with a description already taken is bad
    Given there is a group which description is "Terriers"
    When I fill in "Name" with "Group I"
    And I fill in "Description" with "Terriers"
    And I press "Create Group"
    Then I should see "Group has not been created."
    And I should see "Description has already been taken"

  @done
  Scenario: Creating a group without a name is bad
    When I fill in "Name" with ""
    And I press "Create Group"
    Then I should see "Group has not been created."
    And I should see "Name can't be blank"

  @done
  Scenario: Creating a group without a description is bad
    When I fill in "Description" with ""
    And I press "Create Group"
    Then I should see "Group has not been created."
    And I should see "Description can't be blank"
```

El primer cambio más significativo que vemos es la adición del tag "*@done*". Los tags se utilizan en Cucumber para definir escenarios de prueba específicos, de modo que las pruebas que ya estaban superadas las marcábamos el tag *"@done"*; las pruebas que estábamos testeando, lo hacíamos con el tag "*@actual*", y las pruebas pendientes por testear, las marcábamos con el tag "*@pending*". Con esta manera podíamos testear escenarios de pruebas independientes y no tener que correr todo el conjunto de testeo cada vez que queríamos comprobar alguna de ellas.

Otra de las características que podemos ver que se añadieron en las refactorizaciones, ha sido la modificación del background:

```
  Background:
    Given there are the following users:
      | email              | password | admin  |
      | admin@testing.com  | password | true   |
      | user@testing.com   | password | false  |

    Given I am signed in as "admin@testing.com"
    Given I am on the groups page
    And I follow "New Group"
```

Podemos ver cómo, a diferencia de lo mostrado en la primera iteración, ahora hay añadido una especificación que refleja la existencia de varios tipos de usuarios en el sistema,

en nuestro caso, diferenciados por el flag "*admin*". Y además vemos que para superar los escenarios de prueba de este conjunto debemos estar logueados como administrador:

```
Given I am signed in as "admin@testing.com"
```

Esto es debido a que la parte de gestión de los grupos, en este caso el crearlos, solamente puede ser administrada por un usuario de tipo administrador.

Para finalizar las diferencias respecto con el primer modelo, vemos que el número de escenarios ha crecido. Esto es debido a que el cliente quiso añadir nuevas características al modelo de los grupos:

```
  @done
  Scenario: Creating an existing group is bad
    Given there is a group called "Group I"
    When I fill in "Name" with "Group I"
    And I fill in "Description" with "Sheepdogs and Cattle Dogs"
    And I press "Create Group"
    Then I should see "Group has not been created."
    And I should see "Name has already been taken"

  @done
  Scenario: Creating a group with a description already taken is bad
    Given there is a group which description is "Terriers"
    When I fill in "Name" with "Group I"
    And I fill in "Description" with "Terriers"
    And I press "Create Group"
    Then I should see "Group has not been created."
    And I should see "Description has already been taken"
```

En las primeras reuniones con el cliente no se trataron estas dos características que debían poseer los grupos: Tanto el nombre de los grupos cómo la descripción de éstos no podían estar repetidos en el sistema. De modo que, cuando el administrador intentase crear un grupo con un nombre (o descripción) ya existente, debería aparecer un mensaje de error advirtiendo del fallo y no permitiendo realizar la acción. En sucesivas reuniones con el propietario se trató el tema y se llegó a la conclusión de que el sistema debía actuar así en estos casos. De este modo, mediante una refactorización de la prueba en otras iteraciones se añadieron estas características.

8.3.4.2- Viewing Groups

Veamos el fichero final "*.feature*" que refiere al see_groups visto en la sección 8.3.1:

```
Feature: Viewing groups
  In order to view groups
  As a user
  I want to see them on the groups' page

  Background:
    There are groups with a name and a description
    Given groups with these entries:
      |Name       |Description                                          |
      |Group I    |Sheepdogs and Cattle Dogs (except Swiss Cattle Dog)|
      |Group III  |Terriers                                             |
      |Group IX   |Companion and Toy Dogs                               |
Continúa en la página siguiente...
```

```
@done
Scenario: A visit wants see what groups are defined
  When the standards and nomenclature page is visited
  Then the standards and nomenclature page should content:
    |Name       |Description                                        |
    |Group I    |Sheepdogs and Cattle Dogs (except Swiss Cattle Dog)|
    |Group III  |Terriers                                           |
    |Group IX   |Companion and Toy Dogs                             |

@done
Scenario: Viewing a specific group properties
  When the standards and nomenclature page is visited
  And I follow "Group I"
  Then I should be on the group page for "Group I"
  And I should see "Group I" within "#groups h2"
```

El primer cambio apreciable es que se cambió el nombre de la *feature* a un nombre más aceptado y estándar: En lugar de "*see groups*", lo llamaríamos "*viewing groups*". Cambio poco significativo, pero cambio en la refactorización al fin y al cabo.

El cambio más significativo es que se elimina una columna en la visualización de los grupos que hay en el sistema; antes teníamos una definición así:

```
Background:
  There are groups with a name and a description
  Given groups with these entries:
    |GROUP ID|Name       |Description               |
    |1       |Group I    |Sheepdogs and Cattle Dogs |
    |2       |Group III  |Terriers                  |
    |3       |Group IX   |Companion and Toy Dogs    |

Scenario: A visit wants see what groups are defined
  When the standards and nomenclature page is visited
  Then the standards and nomenclature page should content:
    |GROUP ID|Name       |Description               |
    |1       |Group I    |Sheepdogs and Cattle Dogs |
    |2       |Group III  |Terriers                  |
    |3       |Group IX   |Companion and Toy Dogs    |
```

Lo que se aclaró con el propietario de la aplicación en posteriores reuniones a esta iteración es que, de cara a vista del usuario no se quería tener la columna "GROUP ID", ya que no aporta ninguna información relevante para un usuario del sistema. Por tanto el fichero quedó refactorizado como mostramos a principios de esta sección, eliminando la dicha columna de la visión del usuario:

```
@done
Scenario: A visit wants see what groups are defined
  When the standards and nomenclature page is visited
  Then the standards and nomenclature page should content:
    |Name       |Description               |
    |Group I    |Sheepdogs and Cattle Dogs |
    |Group III  |Terriers                  |
    |Group IX   |Companion and Toy Dogs    |
```

Y para finalizar los cambios en la especificación de "ver los grupos" respecto a la primera iteración del proyecto, vemos que en el último escenario se añadió una línea referente al aspecto visual:

```
  @done
  Scenario: Viewing a specific group properties
    When the standards and nomenclature page is visited
    And I follow "Group I"
    Then I should be on the group page for "Group I"
    And I should see "Group I" within "#groups h2"
```

Que indica, que querremos ver "**Group I**" en formato "***h2***" que esté dentro de algo que tenga un identificador "***#groups***".

8.3.4.3- Deleting Groups

En el caso del borrado de grupos, este es el fichero *"feature"* que tendremos en el sistema una vez terminado:

```
Feature: Deleting groups
  In order to remove groups
  As user
  I want to make it disappear when I'm an admin

  Background:
    Given there are the following users:
      | email              | password | admin  |
      | admin@testing.com  | password | true   |
      | user@testing.com   | password | false  |

    Given there is a group called "Group I"
    And there are sections for this group:
      | Section number | Description |
      | Section 1      | Sheepdogs   |
      | Section 2      | Cattledogs  |

  @done
  Scenario: Deleting groups as admin
    Given I am signed in as "admin@testing.com"
    And I am on the groups page
    When I follow "Group I"
    When I follow "Delete group"
    Then I should see "Group has been deleted."
    And I should not see "Group I"
    And all sections for this group should have been removed

  @done
  Scenario: Deleting groups as no admin is bad
    Given I am signed in as "user@testing.com"
    And I am on the groups page

    When I follow "Group I"
    And I try to delete the "group" "Group I"

    Then I should be redirected to the home page
    And I should see "You can't access to this page."
```

Como primer punto, en comparación con el fichero visto en la primera iteración (Sección 8.3.1), vemos que el documento ha crecido considerablemente. Se ha añadido la parte de comprobación de usuario, un nuevo escenario y se ha ampliado la definición del escenario que teníamos implementado. Además, vemos que en el **Background** se ha especificado un nuevo caso:

```
    And there are sections for this group:
      | Section number | Description |
      | Section 1      | Sheepdogs   |
      | Section 2      | Cattledogs  |
```

Hemos tenido que añadir esta precondición debido a la ampliación sobre el escenario que teníamos (marcado en rojo las líneas añadidas con respecto al anterior):

```
@done
Scenario: Deleting groups as admin
  Given I am signed in as "admin@testing.com"
  And I am on the groups page
  When I follow "Group I"
  When I follow "Delete group"
  Then I should see "Group has been deleted."
  And I should not see "Group I"
  And all sections for this group should have been removed
```

Como se observa, se ha añadido la comprobación de que estamos logueados como administrador, y además se ha añadido una nueva comprobación que requirió el propietario del sistema: “Cuando un grupo es borrado, debe de haber un borrado recursivo de las secciones que incluye grupo”. Comprobamos que el sistema actúa de esta manera mediante la última sentencia marcada en rojo en el cuadro.

El último escenario que se añadió corresponde a tests acerca de la seguridad del sistema:

```
@done
Scenario: Deleting groups as no admin is bad
  Given I am signed in as "user@testing.com"
  And I am on the groups page

  When I follow "Group I"
  And I try to delete the "group" "Group I"

  Then I should be redirected to the home page
  And I should see "You can't access to this page."
```

Mediante esta comprobación lo que se busca es comprobar que el sistema no acepte que un usuario que no sea administrador borre algún grupo. De modo que si esto se intenta nos debe redireccionar a la página principal y mostrarnos el correspondiente mensaje de error.

Con este último cambio acabamos la parte de análisis en cuanto a la refactorización de las pruebas se refiere. Como hemos nombrado al principio de la sección, la refactorización no solo afectó a la parte de los grupos (ni solamente a las pruebas), sino que se fue realizando durante todas las iteraciones y sobre las partes del proyecto que iban necesitando de ella.

9- Diseño

Tras haber estudiado los sitios webs y los servicios que estos proporcionan, y que consideremos relevantes para nuestra aplicación, los cuáles hemos visto en el apartado 4 de este mismo documento, hemos analizado en profundidad y estudiado las herramientas consideradas útiles para el desarrollo del proyecto – apartado 5. De este modo se han concluido varios aspectos, en los que basaremos el diseño de nuestra aplicación web.

Cómo vimos en el apartado 6, vamos a utilizar una metodología de desarrollo ágil, que se basa en pruebas, y es conocida como **ATDD** – Acceptance Test Driven Development. Esta práctica nos obliga a realizar las pruebas antes del desarrollo del proyecto, de manera que durante el desarrollo del mismo conseguiremos un código limpio y que funcione. De este modo, nuestro desarrollo y diseño vendrá condicionado por una serie de pruebas las cuales hemos estudiado en la sección 8.3.

Este aparatado lo comenzaremos detallando la arquitectura que vamos a utilizar para que soporte los requisitos funcionales y no funcionales vistos anteriormente. A continuación nos centraremos en nuestro portal web, estudiando la estructura, algunos casos particulares de *mockups* y el modelo de despliegue. Acto seguido estudiaremos el diagrama de secuencia general que tendrá nuestro sistema, ya que en la sección 8.2.2.9 estudiamos los diagramas de secuencia particulares de ciertos casos de uso. La última sección que veremos en este apartado será referente al estudio de la estructura de la base de datos que va a poseer nuestro sistema.

9.1- Arquitectura

Cómo hemos mencionado, el primero punto que veremos en este apartado será el referente a la arquitectura de nuestra aplicación. Estudiaremos el patrón modelo vista controlador, ActiveRecord y el paradigma REST.

9.1.1- Patrón Modelo Vista Controlador

El patrón de arquitectura de software que se utiliza para nuestro sistema es el de **M**odelo **V**ista **C**ontrolador (**MVC**). Es un patrón que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Es un patrón frecuentemente utilizado en aplicaciones web, dónde la vista es la página HTML que se le muestra al usuario y el código que provee los datos dinámicos a la página; el modelo es el sistema de gestión de base de datos y la lógica del negocio; y el controlador es el responsable de recibir los eventos de entrada desde la vista.

De manera genérica, y un poco más en detalla, los componentes de **MVC** podrían definirse de la siguiente manera:

- **Modelo**: Es la representación de la información con la cual el sistema opera, por lo que gestiona todos los accesos a dicha información (consultas, actualizaciones…) e implementa también los privilegios de acceso que se han descrito en las especificaciones de la aplicación. Envía a la *vista* aquella parte de la información

que en cada momento se solicita para que sea mostrada. Las peticiones de acceso o manipulación de información llegan al *modelo* a través del *controlador*.

- **Vista:** La vista presenta el *modelo* en un formato adecuado para interactuar. Generalmente se refiere a la interfaz de usuario, por tanto requiere del *modelo* la información que debe representar como salida.
- **Controlador:** El controlador responde a eventos (generalmente acciones de los usuarios) e invoca peticiones al *modelo* cuándo se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su *vista* asociada si se solicita un cambio en la forma en que se presenta el modelo (por ejemplo, desplazamiento por un documento o por los diferentes registros de una base de datos. Por tanto, el *controlador* se podría decir que hace de intermediario entre la *vista* y el *modelo*.

El sistema desarrollado sigue el siguiente diagrama:

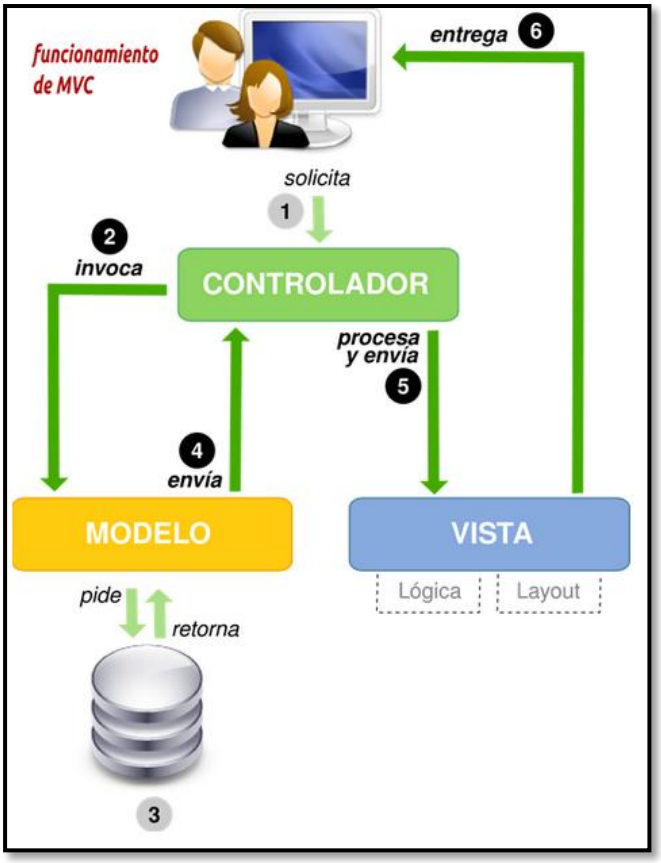


Imagen 9.1.1 – Diagrama de la arquitectura **MVC**

El comportamiento básico del diagrama mostrado en la imagen 9.1.1 es el siguiente:

1. El usuario interactúa con la interfaz de usuario de alguna forma (pulsa un botón, un enlace…). De esta manera el controlador recibe la notificación de la acción solicitada por el usuario y gestiona el evento que llega.
2. El controlador accede al modelo, actualizándolo (por ejemplo, mediante una llamada de retorno – *callback*).

3. El modelo será el encargado de interactuar con la base de datos. Puede ser de forma directa, con una capa de abstracción, un Web Service… y ésta retornará la información al controlador
4. El controlador recibe la información y la envía a la vista.
5. La vista procesa esta información, creando una capa de abstracción para la lógica (quien se encargará de procesar los datos) y otra para el diseño de la interfaz gráfica o GUI.
6. La lógica de la vista, una vez procesa los datos, los acomodará en base al diseño de la GUI – *layout* – y los entregará al usuario de forma legible para el usuario.

Este es el esquema general para un patrón Modelo Vista Controlador. Rails, obedece a este esquema, y a continuación vamos a ver un caso específico de acción en Rails para un administrador. En el caso que sigue, la acción que quiere hacer el administrador es visualizar todos los usuarios que hay en el sistema; veamos cómo sería el diagrama:

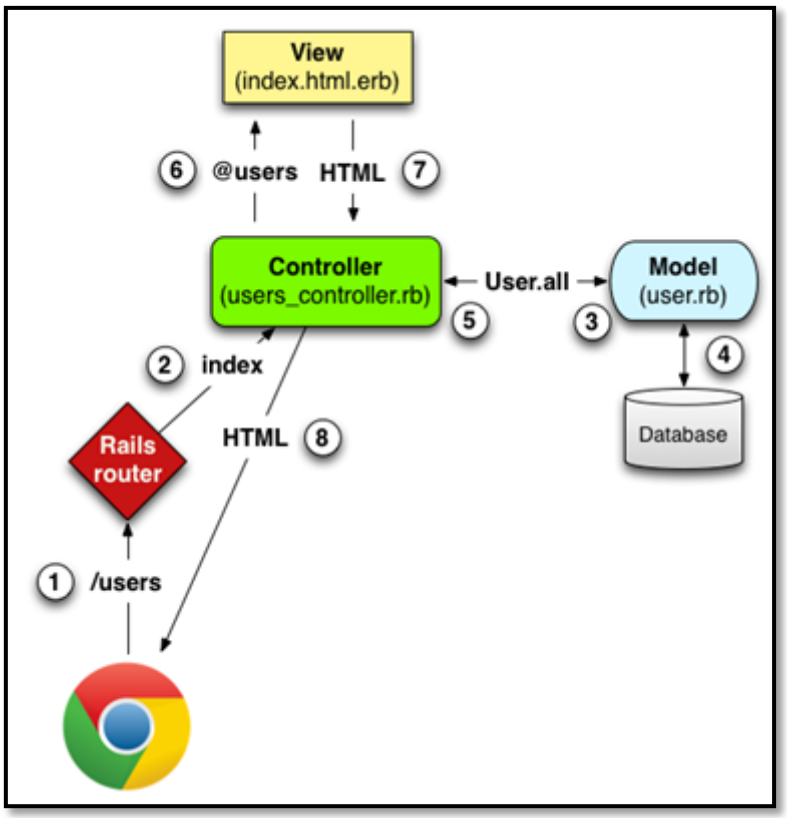


Imagen 9.1.2 – Diagrama detallado de MVC para Rails

1. El navegador emite una solicitud de URL para "/users".
2. Rails enruta "/users" a la acción *index* del controlador de los usuarios (*users_controller.rb*).
3. La acción *index* pide al modelo de usuarios para recuperar todos los usuarios (User.all).
4. El modelo *User* recoge todos los usuarios de la base de datos.
5. El modelo *User* devuelve la lista de usuarios al controlador.
6. El controlador capta los usuarios en la variable "@users", que es mandada a la vista *index*.

7. La vista utiliza Ruby para renderizar la página como HTML.
8. El controlador manda el HTML de nuevo al navegador.

Y de esta forma, actuaría nuestro patrón Modelo Vista Controlador en el sistema, para el caso concreto de que un administrador buscase listar todos los usuarios que hay registrados en la aplicación.

9.1.2- ActiveRecord

En nuestro sistema, gobernado por el framework Ruby on Rails, se utiliza para operar con las bases de datos un Mapeo Objeto Relacional (**ORM** – **O**bject-**R**elational **M**apping). Este mapeo es una técnica de programación utilizada para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación a objetos y la utilización de una base de datos relacional como motor de persistencia. En la práctica, esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

Por tanto, resumiendo, un **ORM** nos permite hacer consultas a las bases de datos a través de objetos. En el mercado existen paquetes comerciales y de uso libre que desarrollan este mapeo relacional, aunque hay programadores que prefieren crear sus propias herramientas ORM. En nuestro caso en particular vamos a hacer uso de la herramienta *ActiveRecord*, de Rails. ActiveRecord es la capa de Rails de mapeo relacional de objetos. Provee una interfaz orientada a objetos que se relacionan con una base de datos, para hacer el desarrollo más simple y amigable para el desarrollador. El ActiveRecord de Ruby, como ya hemos mencionado, es un **ORM**: hace las traducciones entre objetos Ruby y la base de datos, manejando registros y relaciones. La librería ActiveRecord de Ruby crea un modelo de negocio persistente desde los objetos de negocio y las tablas de datos, unificándose la lógica y los datos en un solo paquete. ActiveRecord nos agregará también herencia y asociaciones al patrón ActiveRecord común, dándole más potencia.

Ni que mencionar tiene que es una característica muy útil para nosotros, ya que nos permitirá abstraernos de utilizar lenguajes específicos para acceder a las bases de datos, y centrarnos sólo en qué es lo que queremos conseguir, sin necesidad de tener que estudiar otro lenguaje específico.

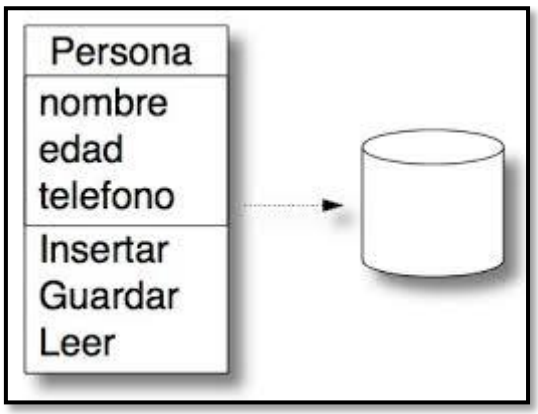


Imagen 9.1.3 – Mapeo Objeto Relacional (ORM)

9.1.3- Transferencia de Estado Representacional (REST)

REST (**RE**presentational **S**tate **T**ransfer) describe un paradigma de arquitectura para aplicaciones que solicitan y manipulan recursos en la web utilizando los métodos estándar de HTTP: GET, POST, PUT y DELETE.

En **REST** todo recurso es una entidad direccionable mediante una URL única con la que se puede interactuar a través del protocolo **HTTP.** Dichos recursos pueden representarse en diversos formatos como HTML, XML o RSS, según solicite el cliente. Al contrario que en las aplicaciones Rails tradicionales, la URL de un recurso no hace referencia a un modelo y su acción correspondiente, tan sólo hace referencia al propio recurso.

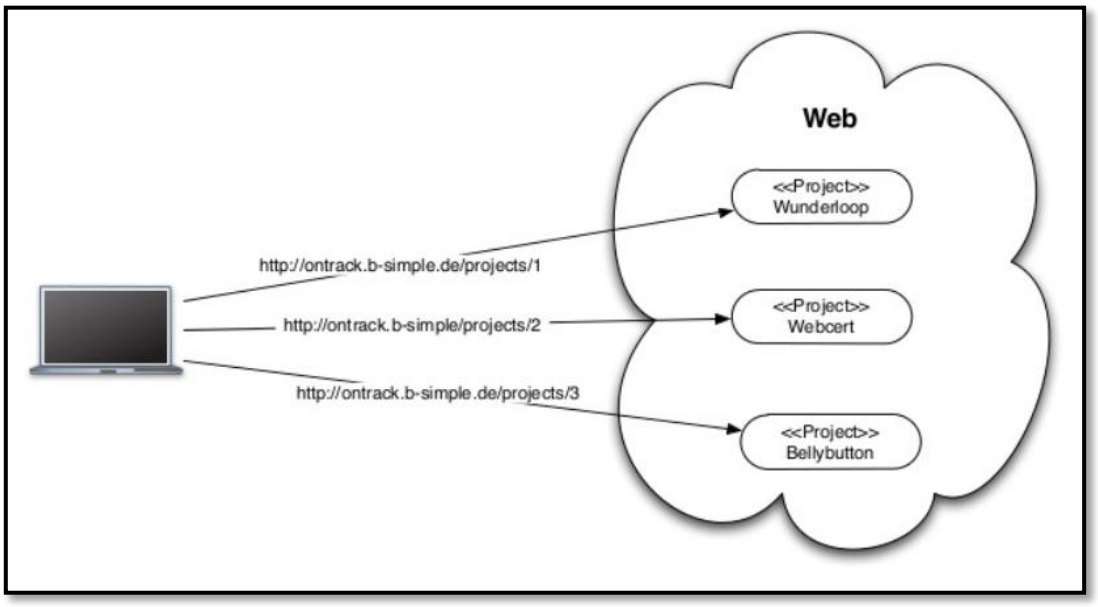


Imagen 9.1.4 – Recursos en la web y sus URLs

Los tres recursos de la imagen 9.1.4 están representados por URLs casi idénticas, seguidas por el identificador del recurso. Obsérvese que la URL no muestra qué se desea hacer con el recurso.

Un recurso, en el contexto de una aplicación Rails, es una combinación de un controlador dedicado y un modelo. De este modo, desde un punto de vista técnico, los recursos proyecto de la figura anterior son instancias de la clase ActiveRecord Project, combinadas con un controlador ProjectController, que es el responsable de manipular dichas instancias.

Entre otras características, utilizar un modelo de aplicación **REST**, aporta los siguientes beneficios:

- **URLs limpias**: En **REST**, las URLs representan recursos y no acciones, por lo tanto siempre tienen el mismo formato; primero aparece el controlador y luego el identificador del recurso.
- **Formatos de respuesta variados**: Los controladores **REST** están escritos de manera que las acciones pueden devolver sus resultados fácilmente en diferentes formatos de respuesta. Una misma acción puede entregar HTML, XML, RSS…

- **Menos código**: El desarrollar acciones únicas capaces de soportar múltiples clientes distintos evita repeticiones en el sentido DRY.
- **Controladores orientados a CRUD**: Los controladores y los recursos se funden en una única coas; cada controlador tiene como responsabilidad manipular un único tipo de recurso.
- **Diseño limpio**: El desarrollo **REST** produce un diseño de aplicación conceptualmente claro y más fácil de mantener.

Grosso modo, REST básicamente es un esquema de URLs y métodos HTTP que representan operaciones (leer, crear, actualizar…) sobre un recurso. Veamos un ejemplo de rutas para el recurso Raza:

Operación	Ruta	Método HTTP
Ingresar datos	/breed/new	GET
Crear registro	/breed	POST
Leer	/breed/1	GET
Modificar datos	/breed/1/edit	GET
Actualizar registro	/breed/1	PUT
Eliminar registro	/breed/1	DELETE
Listar todas	/breed	GET

Como podemos observar, las operaciones que requieren de una interfaz gráfica y no modifican el estado de un recurso (ingresar y actualizar datos, leer y listar) utilizan el método GET. Las demás operaciones que sí modifican el estado, utilizan el resto de los métodos.

9.2- Portal Web

9.2.1- Estructura

En esta sección representaremos el diseño de un portal web adecuado para la gestión de las exposiciones caninas. Para su elaboración se deberá tener en cuenta aspectos fundamentales de accesibilidad web, en especial interés, mostrar los contenidos que hemos estado estudiando a lo largo del documento. En particular, tendremos un portal web que contenga:

- Usuarios
  - Visitantes
  - Usuarios registrados
  - Administrador del sistema
- Información general
  - Grupos de clasificación de razas según la FCI
  - Secciones y sub-secciones de división de grupos según la FCI
  - Sistema de razas homologadas
  - Variedades y sub-variedades para las razas homologadas
- Sistema de exhibiciones
  - Zona de información

  - Fechas de inscripción
  - Detalles de exhibiciones
  - Precios
- Inscripciones
  - Administración de perros
  - Control de precios
  - Control de pagos para los usuarios
  - Control de pagos para el administrador
  - Control de inscripciones caninas
- Zona privada
  - Acceso a zona privada usuarios

Estas secciones son las que deberá contener y estar presentadas en el sitio web. A continuación presentamos unos Mockups realizados para reflejar cómo debe estar representado dicho portal de acuerdo al contenido que debe tener. Veremos los mockups de las secciones más importantes de la web, en los que incluimos: la página principal, información sobre una exposición y el ver las inscripciones que tiene un determinado usuario en una cierta exposición.

De este modo empecemos mostrando la vista que debería tener la página principal del portal:

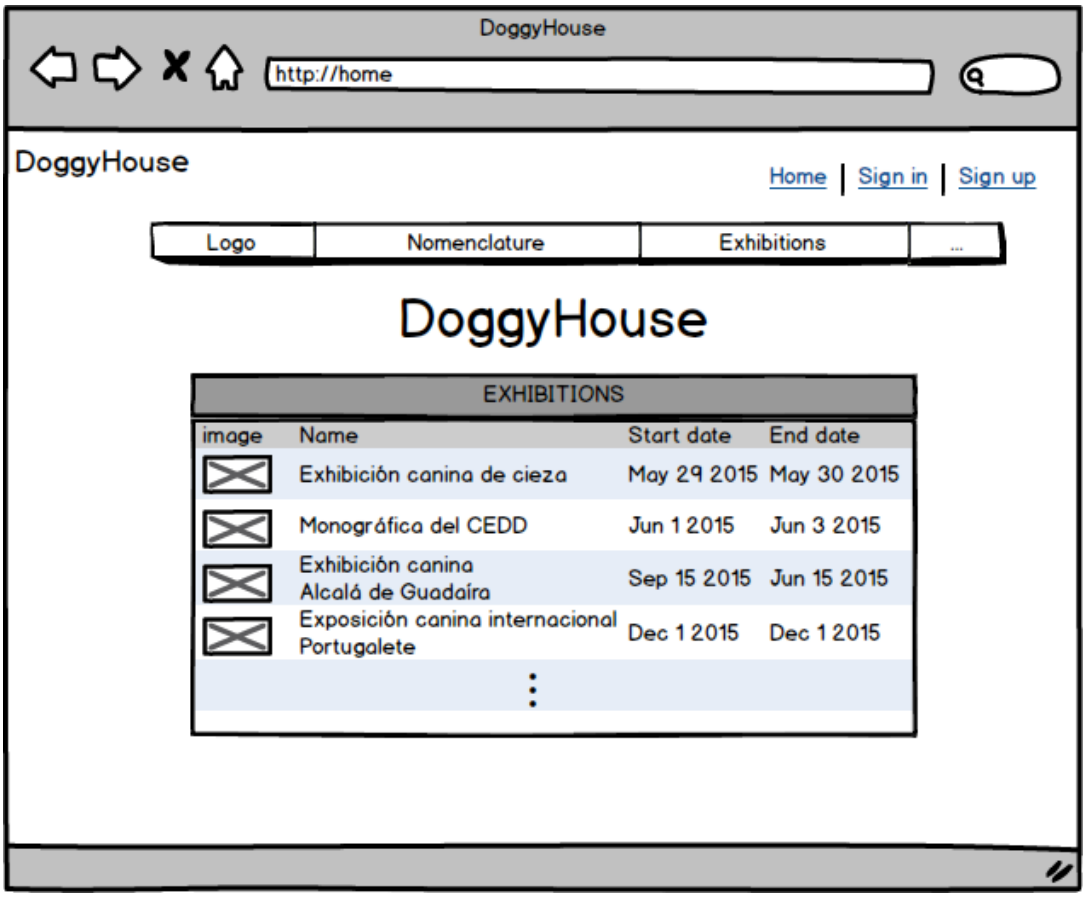


Imagen 9.2.1 – Mockup página principal

La página de inicio del portal debería ser algo parecido a lo que muestra la imagen 9.2.1. Deberemos mostrar las exhibiciones que hay actualmente en el sistema, y las posibilidades para loguearse o registrarse. Además, cuando el usuario esté logueado, el menú de la parte superior de la pantalla debería cambiar para que mostrara opciones acorde a las acciones que un usuario registrado/administrador puede hacer. En vez la barra que tenemos con 'home', 'sign in' y 'sign up', deberíamos cambiar la muestra y ofrecerle al usuario posibilidades cómo por ejemplo: 'home', 'sign out', 'profile', 'my exhibitions'… Algo que refleje mejor las acciones que puede realizar. Veamos un ejemplo de una visión para un usuario que ya se ha logueado en el sistema:

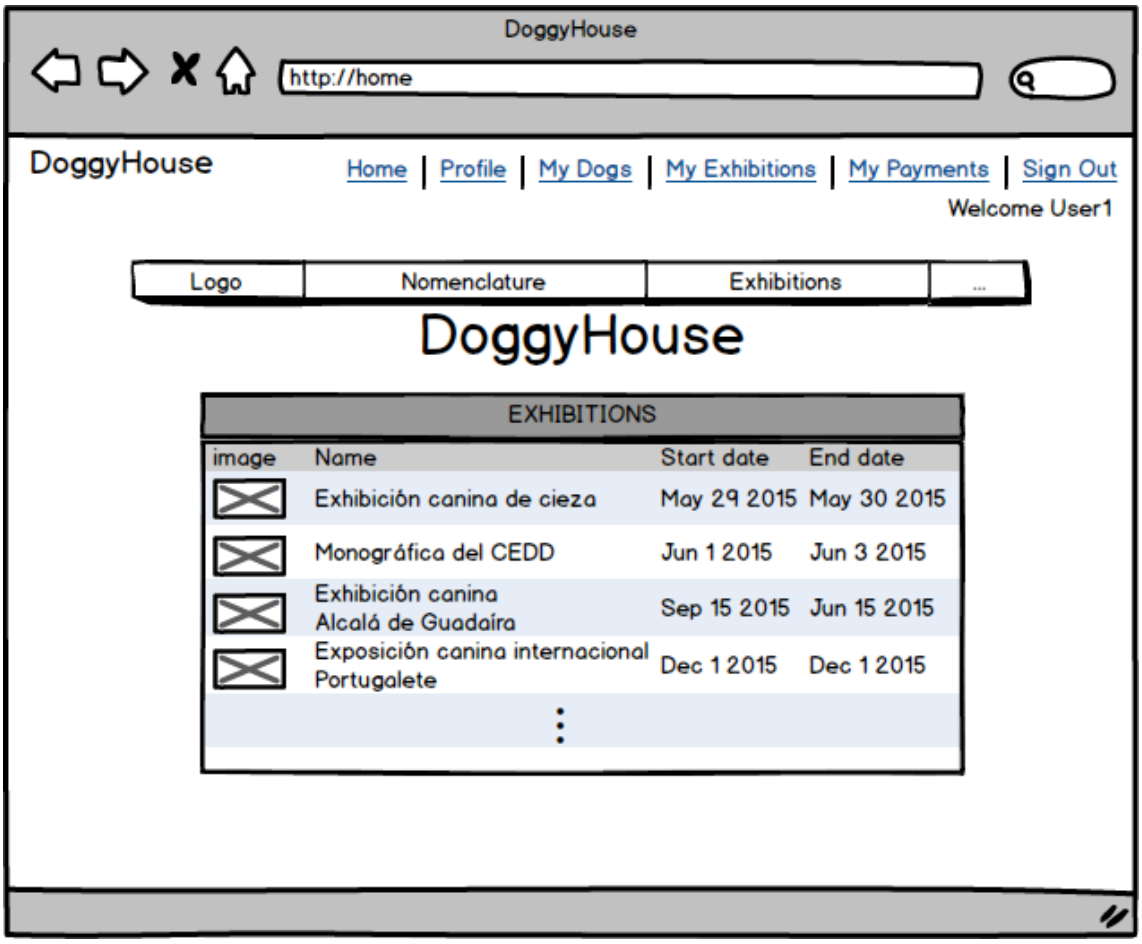


Imagen 9.2.2 – Mockup página principal (Visión Usuario Registrado)

En cambio, el menú del administrador en relación con el de un usuario registrado típico, debería cambiar y presentarse conforme a algo así:

Imagen 9.2.3 – Mockup página principal (Panel Administrador)

A la hora de entrar en la página de información de alguna de las exhibiciones, debería mostrar una presentación conforme al siguiente Mockup:

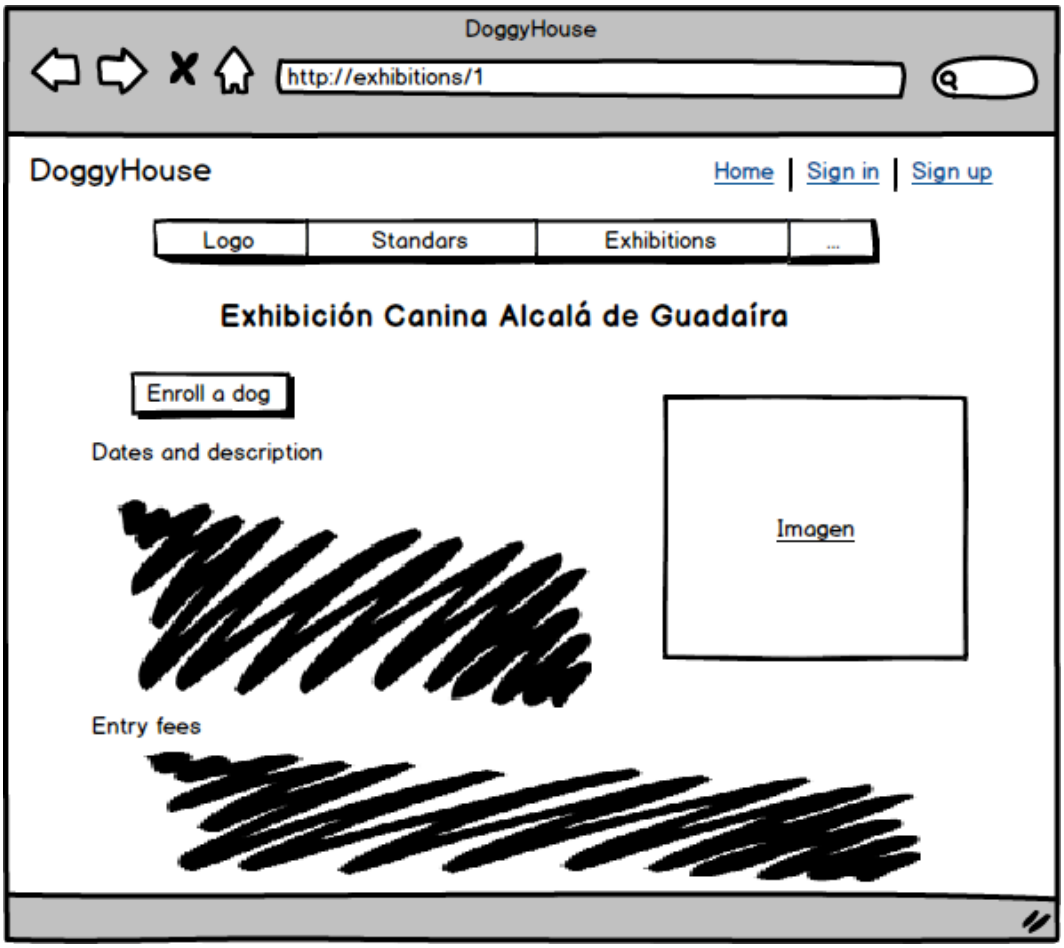


Imagen 9.2.4 – Mockup Detalles de exhibición

De modo que se mostrará la información relevante de la exhibición, así como la barra de navegación para poder visitar de una forma amena las distintas secciones del sitio web. Como podemos ver, aparece un botón de acceso directo a enrolar un perro. Dicho botón, sólo estará operativo en el caso de que el usuario esté logueado; en caso contrario se le redireccionará a la página de logueo. Además, en el caso de que el usuario sea un Administrador, deberán estar presentes las acciones disponibles para el mismo en cuanto a las exhibiciones (borrar y modificar). Veamos a modo de ejemplo este Mockup:

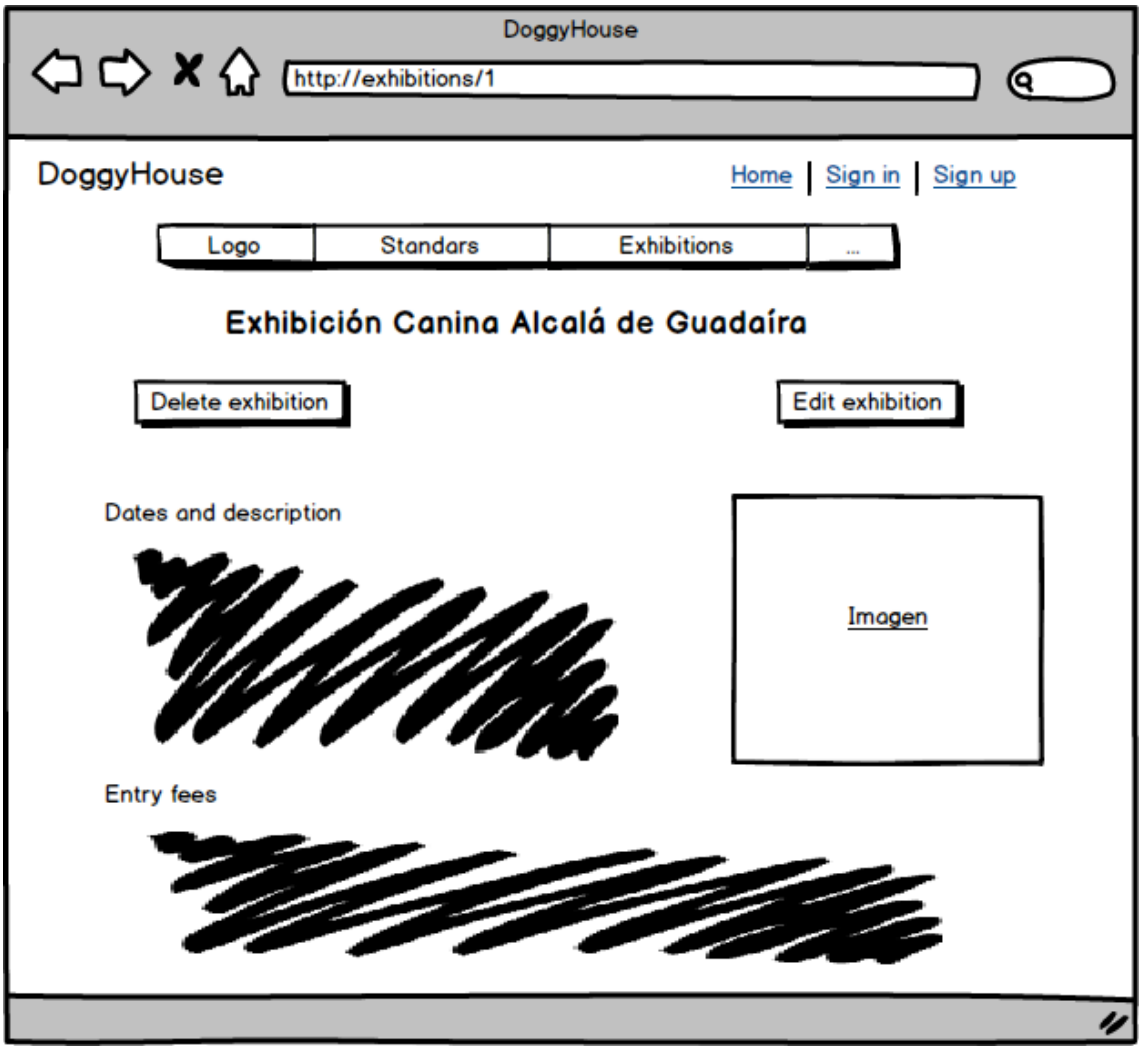


Imagen 9.2.5 – Mockup Detalles de exhibición (Visión Administrador)

Como observamos, la presentación original apenas varía excepto por el acceso a las habilidades propias del Administrador.

Para finalizar este bloque de Mockups, vamos a ver cómo sería una representación de cómo debería verse la página de los enrolments de un usuario. Se deberá diferenciar dos partes, una parte en la que salgan enrolments pendientes de abono, y otra en la que salgan enrolments que están ya pagados, la primera en la parte superior de la página, y la segunda justo debajo de la primera; veámoslo:

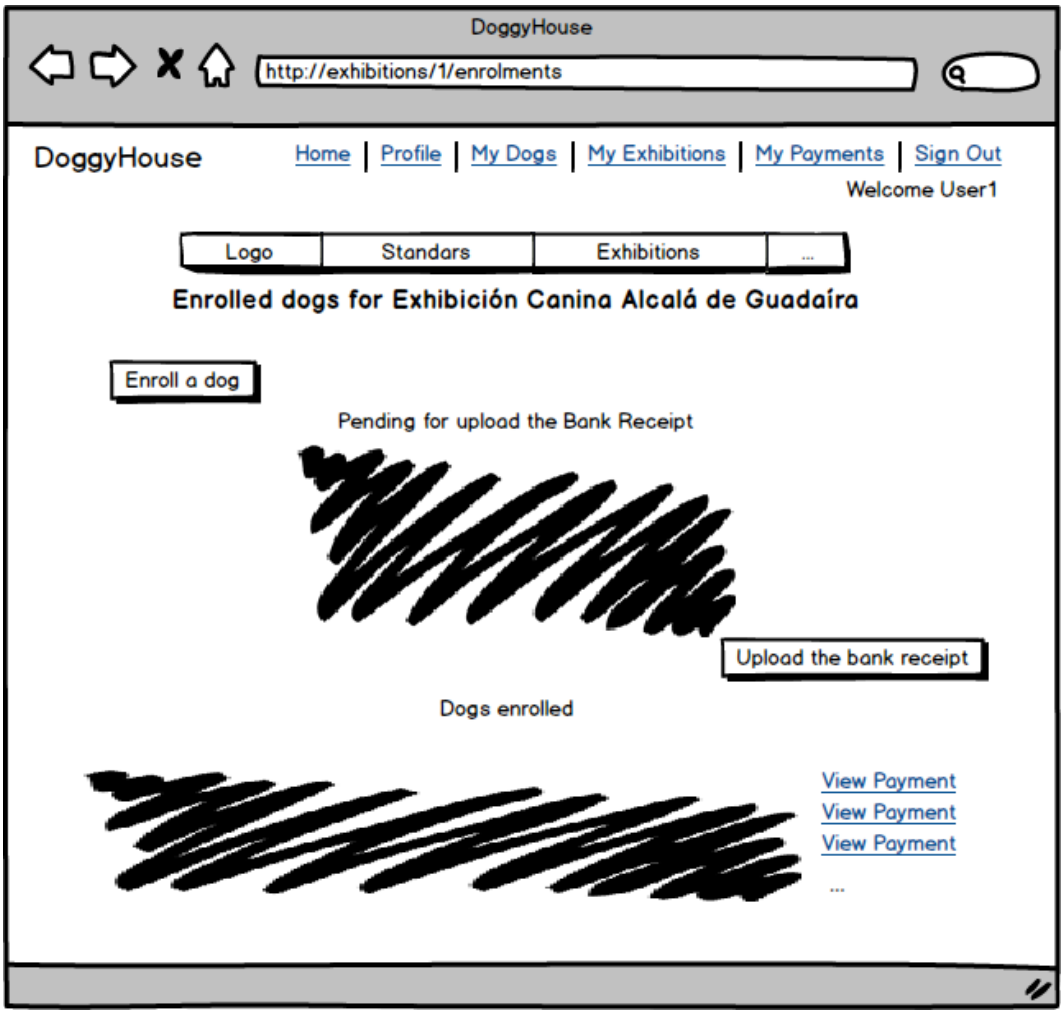


Imagen 9.2.6 – Mockup Enrolments para un usuario

9.2.2- Modelo de despliegue

Una vez el sistema esté implementado, ha de ser desplegado para que pueda ser utilizado por los usuarios finales. En nuestro caso en particular se ha utilizado el **Cloud Computing**, técnica que permite ofrecer servicios de computación en la nube a través de internet. Pasemos a estudiar esto en detalle:

En el modelo tradicional de implementación de Tecnologías de Información, las organizaciones destinan recursos materiales, humanos y tecnológicos, los cuales agrupan en un área encargada de solucionar los problemas relacionados con la infraestructura informática y el desarrollo de aplicaciones para la organización.

La mayoría de estas áreas se ven obligadas a dedicar gran parte de su tiempo en las tareas de implementar, configurar, mantener y actualizar proyectos relacionados con la infraestructura de su organización, lo cual, normalmente no supone valor añadido en el balance final de la producción de la misma.

Por otra parte, se puede observar que la distribución de servicios tales como energía eléctrica, agua potable o telefonía, dejan al proveedor la total responsabilidad de generar, organizar y administrar lo necesario para que el usuario final reciba lo acordado, pagando éste únicamente por el uso que hace de los mismo, mientras que realmente el proveedor se

encarga de precias los mecanismos por medio de los cuales determina el consumo por el que se genera el cobro.

De esta manera nos surge una pregunta: ¿por qué no implementar servicios o recursos de Internet bajo un esquema similar al descrito, dónde el proveedor solamente proporcione lo requerido y el usuario pague únicamente por el uso que hace?

Es por esto que dirigimos nuestra mirada hacia la tecnología mencionada anteriormente, el **Cloud Computing** (cómputo en la nube), la cual es capaz de minimizar el tiempo empleado en actividades de menor valor y permitirnos centrarnos realmente en lo importante en nuestro desarrollo.

En la siguiente imagen podemos apreciar un diseño arquitectónico general de cómo se comporta la nube, en la que los diferentes cliente interactúan con las distintas capas en las que puede dividirse la nube.

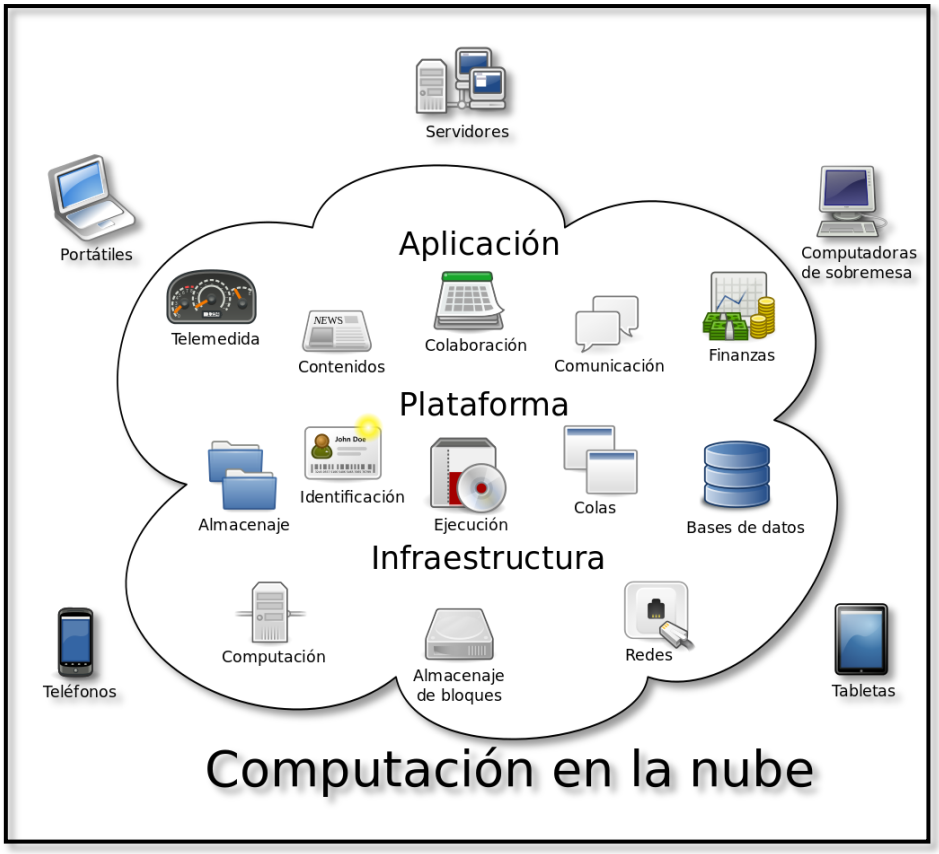


Imagen 9.2.7 – Esquema general de arquitectura Cloud Computing

Por lo general, el uso de este concepto se está extendiendo con una considerable velocidad, dando como resultados un incremento en el número de empresas que proporcionan servicios a través de esta tecnología. Los tipos de servicios que ésta nos puede proporcionar son extensos. El cliente paga a un proveedor por un recurso determinado (memoria, almacenamiento, procesamiento, software, bases de datos…) y éste le proporciona dicho servicio a través de internet.

Las ventajas que nos ofrece dicha tecnología se pueden simplificar en lo siguiente:

- **Costos:** Podría ser la ventaja más atractiva que presenta el cómputo en la nube, y si no lo es, al menos es la más evidente de todas las que ofrece esta tecnología. Al dejar la responsabilidad de implementación de la infraestructura al proveedor, el cliente no tiene por qué preocuparse por comprar equipos de cómputo, capacitar personal, mantenimiento…
- **Competitividad:** Al no tener que adquirir costos equipos, las pequeñas empresas pueden tener acceso a las más nuevas tecnologías a precios a su alcance, pagando únicamente por consumo.
- **Disponibilidad:** El proveedor está obligado a garantizar que el servicio esté disponible para el cliente. En este sentido, la virtualización juega un papel fundamental.
- **Abstracción de la parte técnica:** El cómputo en la nube permite al cliente la posibilidad de olvidarse de la implementación, configuración y mantenimiento de equipos.
- **Acceso desde cualquier punto geográfico:** El uso de las aplicaciones diseñadas sobre el paradigma del cómputo en la nube puede ser accesible desde cualquier equipo de cómputo en el mundo que esté conectado a internet.
- **Escalabilidad:** El cliente no tiene por qué preocuparse por actualizar el equipo de cómputo sobre el que se está corriendo la aplicación que utiliza, ni tampoco por la actualización de sistemas operativos o instalaciones de parches de seguridad, ya que es obligación del proveedor del servicio realizar este tipo de actualizaciones.
- **Concentración de esfuerzos en el negocio:** Como resultado de las ventajas antes mencionadas, el cliente puede concentrar más recursos y esfuerzos hacia un aspecto más estratégico y transcendente, que tenga un impacto directo sobre os procesos de negocio de la organización.

Sin embargo, también hay algunos puntos de desventaja en la utilización de esta tecnología; detallemos algunos de ellos:

- **Privacidad:** Es comprensible la percepción de inseguridad que genera una tecnología que pone la información (sensible en muchos casos), en servidores fuera de la organización, dejando como responsable de los datos al proveedor de servicio. El tema a tratar aquí, es el de la privacidad, ya que para muchos es extremadamente difícil el confiar su información sensible a terceros, y consideran que lo que propone el cómputo en la nube pone en riesgo la información vital para los procesos del negocio.
- **Disponibilidad:** Si bien es cierto que en las ventajas se incluyó el concepto de disponibilidad, ésta queda como una responsabilidad que compete únicamente al proveedor del servicio, por lo que si su sistema de redundancia falla, y no logra mantener el servicio para el usuario, éste no puede realizar ninguna acción correctiva para restablecer el servicio.
- **Falta de control sobre recursos:** Al tener toda la infraestructura e incluso la aplicación corriendo sobre servidores que se encuentran en la nube, es decir, del

lado del proveedor, el cliente carece por completo de control sobre los recursos e incluso sobre su información una vez ésta es subida a la nube.

- **Dependencia:** En una solución basada en cómputo en la nube, el cliente se vuelve dependiente no sólo del proveedor del servicio, sino también de su conexión a internet.
- **Integración:** No siempre es fácil o práctica la integración de recursos disponibles a través de infraestructuras de cómputo en la nube con sistemas desarrollados de una manera tradicional, por lo que este aspecto debe ser tomado en cuenta por el cliente para ver qué tan viable resulta implementar una solución basada en la nube dentro de su organización.

En la sección 5.2 del documento, nombrábamos que nuestra plataforma de control de la nube debería poseer características **PaaS** (**P**latform **a**s **a S**ervice – Plataforma como Servicio). El cómputo de la nube se puede dividir en tres niveles en función de los servicios que ofrece a los proveedores. Lo que vamos a ver en los siguientes párrafos es la definición de estos tres niveles:

Infraestructura como Servicio (IaaS – Infraestructura as a Service)

La infraestructura como un servicio es un modelo de aprovisionamiento, en el cual una organización coloca "fuera de ella" el equipo usado para soportar operaciones, esto incluye el almacenamiento de la información, el hardware, servidores y componentes de redes.

La ventaja más evidente de utilizar **IaaS**, es la de transferir hacia el proveedor problemas relacionados con la administración de equipos de cómputo. Otra ventaja atractiva es la reducción de costos. Además, la infraestructura como servicio permite escalabilidad práctica automática y transparente al consumidor.

Plataforma como servicio (PaaS – Platform as a Service)

Podríamos definir el concepto de **PaaS** como una plataforma que permite crear y ejecutar aplicaciones personalizadas.

El proveedor, además de resolver problemas en la infraestructura de hardware, también se encargar del software. Los que hacemos uso de este tipo de aplicaciones no necesitaremos instalar, configurar ni dar mantenimiento a sistemas operativos, bases de datos y servidores de aplicaciones, que todo esto está integrado en la plataforma

Una plataforma **PaaS** resuelve más problemas si se compara con la infraestructura vista anteriormente, ya que presenta muchas limitaciones relacionadas con el entorno de ejecución. En éstas se encuentran el tipo de sistema, el lenguaje de programación, el manejador de bases de datos…

Empresas como Amazon, eBaby, Google e iTunes son algunas de las que emplean este modelo y hacen posible acceder a nuevas capacidades y nuevos mercados a través del navegador web.

Software como Servicios (SaaS –Software as a Service)

SaaS es el más conocido de los niveles de cómputo en la nube. El **SaaS** es un modelo de distribución del software que proporciona a los clientes el acceso a éste a trvés de la red

(generalmente Internet). De esta forma, ellos no tienen que preocuparse de las configuraciones, implementación o mantenimiento de las aplicaciones, ya que todas las labores se vuelven responsabilidad del proveedor. Las aplicaciones distribuidas a través de un modelo de Software como Servicio pueden llegar a cualquier empresa sin importar su tamaño o ubicación geográfica.

Este modelo tiene como objetivo al cliente final que utiliza el software para cubrir procesos de su organización. El Software como Servicio (**SaaS**) se puede describir como aquella aplicación consumida a través de internet, normalmente a través del navegador, cuyo pago está condicionado al uso de la misma y donde la lógica de la aplicación, así como los datos, residen en la plataforma del proveedor. Ejemplos de **SaaS** son Salesforce, Zoho y Google App.

La imagen que se muestra a continuación, podemos ver la estructura piramidal de los modelos que hemos estado detallando. Vemos como, en cierta parte, algunos niveles "heredan de los otros".

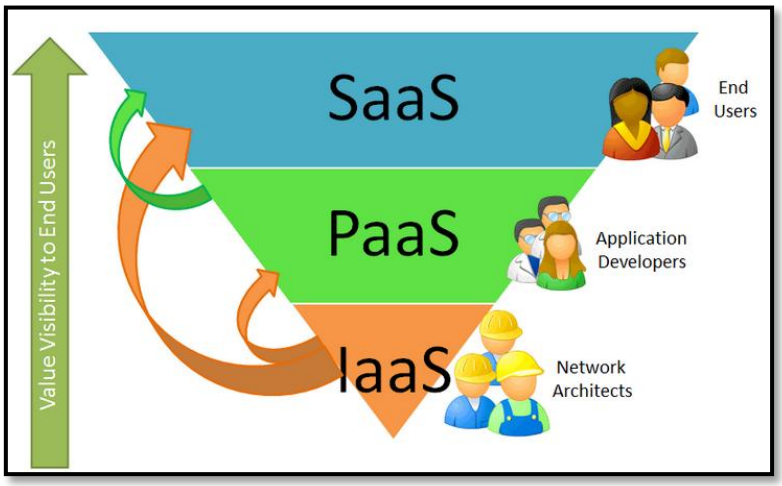


Imagen 9.2.8 – Capas en Cloud Computing

En nuestro caso en particular, como nombras anteriormente en el documento, vamos a utilizar Heroku como plataforma, y el dominio final del producto será: "http://doggyhouse.herokuapp.com/". También recordar, como hemos dicho, será una plataforma con características **PaaS**.

9.3- Diagrama de secuencia general

En la sección 8.2.2.9 estudiamos ciertos diagramas de secuencia específicos para los casos de uso de nuestro sistema. A su vez, en el mismo apartado que nos encontramos, en la sección 9.1.1, vimos la arquitectura del **patrón MVC**. En esta sección vamos a estudiar el diagrama general que debe tener un sistema basado en el paradigma Modelo-Vista-Controlador.

El diagrama que vamos a mostrar a continuación, sería el diagrama de secuencia general que las aplicaciones basadas en este paradigma deberían tener:

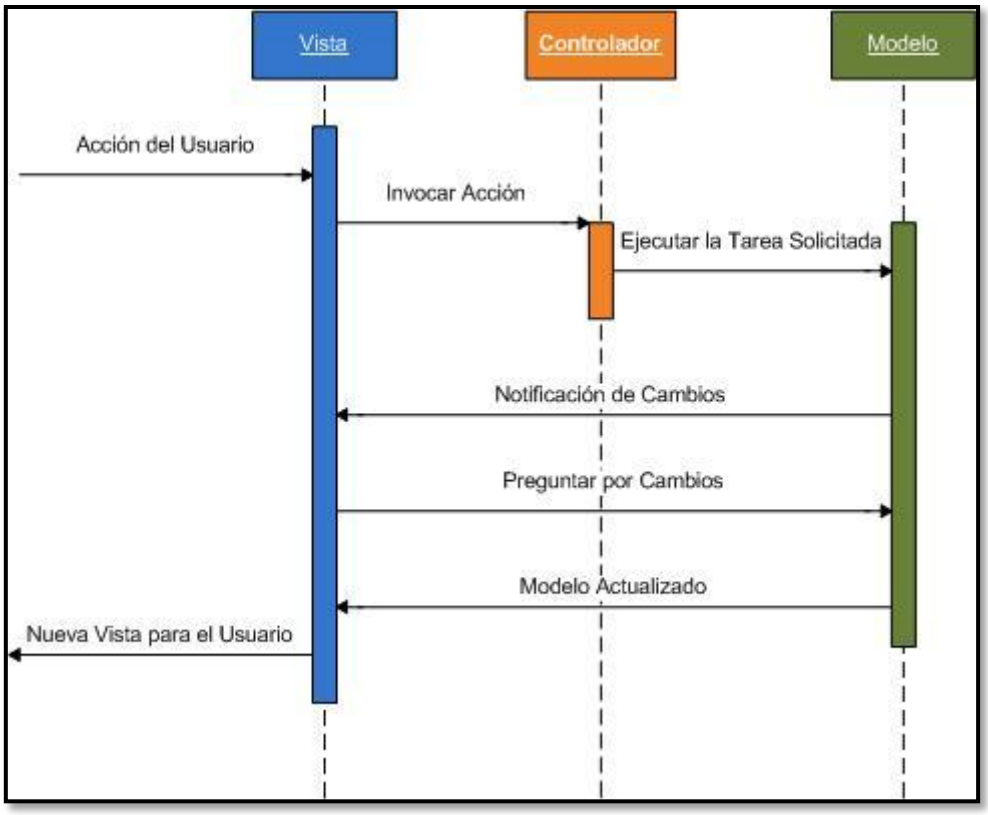


Imagen 9.3.1 – Iteración Modelo-Vista-Controlador estándar

Cómo vemos, el usuario interactúa con la vista, y las acciones (como por ejemplo clikear un botón) se capturan en la vista y se reenvían al controlador. El controlador decide qué hacer y lo hace a través de una iteración con el modelo. El modelo es esencialmente la capa empresarial de algunas capacidades adicionales. En concreto, el modelo notifica a la vista sobre los cambios que pueden requerir una actualización de la interfaz de usuario.

El modelo no conoce los detalles de la vista, pero entre el modelo y la vista hay una relación de "*observador*". En otras palabras, la vista contiene una referencia al modelo y se registra con el modelo para recibir notificaciones de cambios. Cuando se recibe la notificación, la vista obtiene datos actualizados desde el modelo y actualiza la interfaz de usuario. La figura anterior muestra el diagrama de secuencia. Hay casos de implementación de **MVC** en el que es el controlador el que notifica a la vista los cambios.

Una variante a esto, llamado comúnmente ***Model2***, es el esquema que presentamos en la imagen 9.3.2, en el que se expone la vista al usuario mediante el explorador. En la imagen 9.1.2 veíamos que aparecía un componente *Rails route* y un navegador en nuestro modelo, este es el caso particular del ***Model2*.**

El modelo **MVC,** visto anteriormente, se diseñó pensando en las aplicaciones de escritorio, pero debido a relativamente flexible formulación, se adapta fácilmente a la web. Una variación popular del patrón MVC es el nombrado ***Model2***.

El ***Model2***, como mencionamos, expone la vista a los usuarios a través del navegador. Las acciones de los usuarios son capturadas por el navegador y transformadas en una nueva petición HTTP. De este modo, las peticiones fluyen desde el navegador a un componente

especial (denominado *front controller*), que se estable en el servidor Web. El controlador coordina las peticiones que se hacen a la aplicación web.

La siguiente figura muestra el diagrama de secuencia típico de la iteración del ***Model2***, en el que mostramos los pasos generales de este modelo:

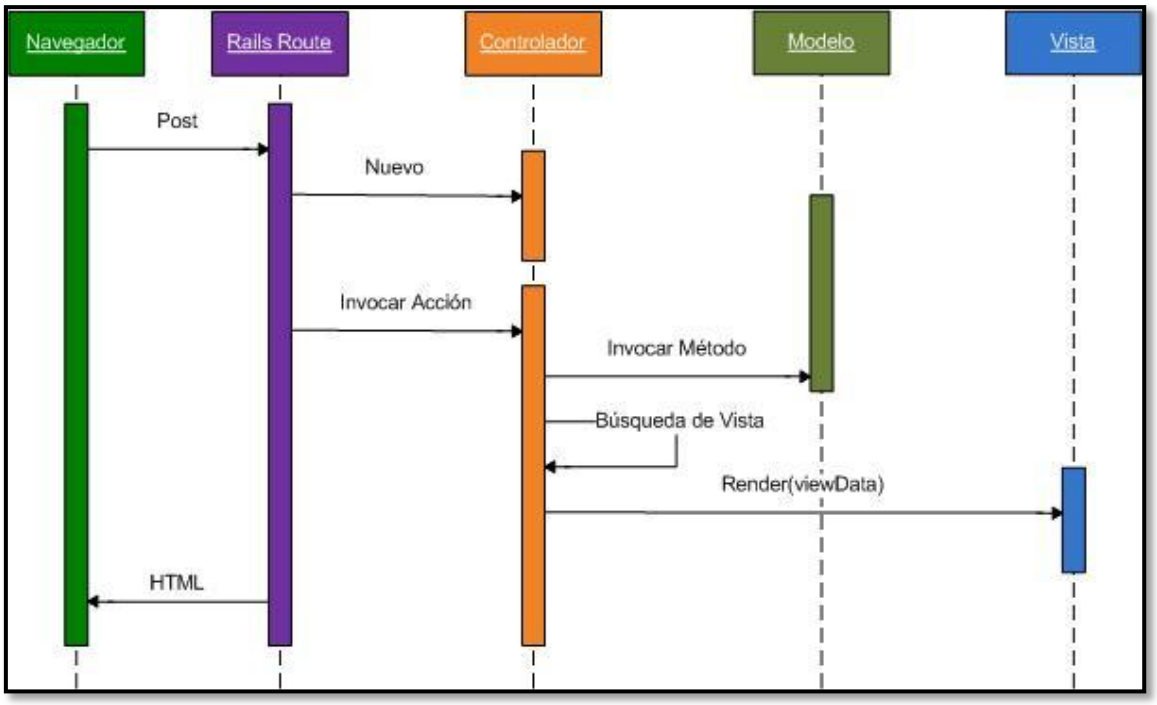


Imagen 9.3.2 – Iteración Modelo-Vista-Controlador Model2

Podemos observar varias diferencias entre el modelo expuesto en este diagrama y el anterior. Como vemos, no hay contacto entre la vista y el modelo, como había en la formulación original **MVC**, basada en la mencionada relación "*observador*". La acción del usuario no es captada y manejada por la vista; el controlador renderiza la vista y transmite el dato a mostrar explícitamente a ésta.

9.4- Base de datos

La última sección que veremos en cuanto al diseño, va a estar relacionada con la base de datos de nuestra aplicación. El esquema de una base de datos nos va a permitir describir la estructura de la base de datos. El lenguaje utilizado es un lenguaje relacional y nos da la posibilidad de definir cada tabla con sus campos y las relaciones entre ellas.

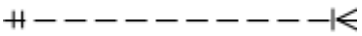
Las bases de datos permiten que los sistemas sean dinámicos, y son herramientas potentes, ampliamente utilizadas en todas las aplicaciones. Existen varias formas de representar el modelado de una base de datos y en nuestro caso en particular vamos a utilizar el modelo entidad-relación.

Un **modelo entidad-relación** es una herramienta muy utilizada para el modelado de datos que permite representar las entidad relevantes de un sistema de información, así como sus interrelaciones y propiedades.

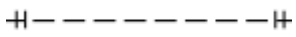
El procedimiento que se ha utilizado para el modelo ha sido, primeramente, elaborar el diagrama entidad-relación que mostramos en la imagen 9.4.1, y a continuación complementar el modelo con los atributos propios de cada una de las entidades, así como las restricciones que no se han podido reflejar en el diagrama.

Las relaciones que veremos en el diagrama de nuestro sistema van a ser 3:

- Relación 1 a muchos (**1:m**) – Esta relación es la más común. Cada registro de una tabla puede estar enlazado con varios registros de una segunda tabla, pero cada registro de la segunda sólo puede estar enlazado con un único registro de la primera; se representa mediante este dibujo:



- Relación 1 a 1 (**1:1**) – En este tipo de relación, un registro de la tabla uno sólo puede estar relacionado con un único registro de la tabla dos y viceversa. En nuestro diagrama se puede ver representado por este dibujo:



- Relación mucho a mucho (**n:m**) – Cada registro de la primera tabla puede estar enlazado con varios registros de la segunda y viceversa. Este tipo de relación implica la repetición de los campos de cada tabla. Para establecer relaciones de este tipo, es necesario crear una tabla intermedia que esté relacionada con las dos de uno a varios. Esta es la relación que guarda la tabla **Dog** y **Exhibition**, mediante **Enrolments**, que podemos observar en la imagen 9.4.1.

A continuación vamos a mostrar dicho diagrama y a estudiarlo, parándonos a detallar los puntos más relevantes de este diagrama. Veremos que tenemos un total de 12 tablas dónde guardarán relación directa entre ellas.

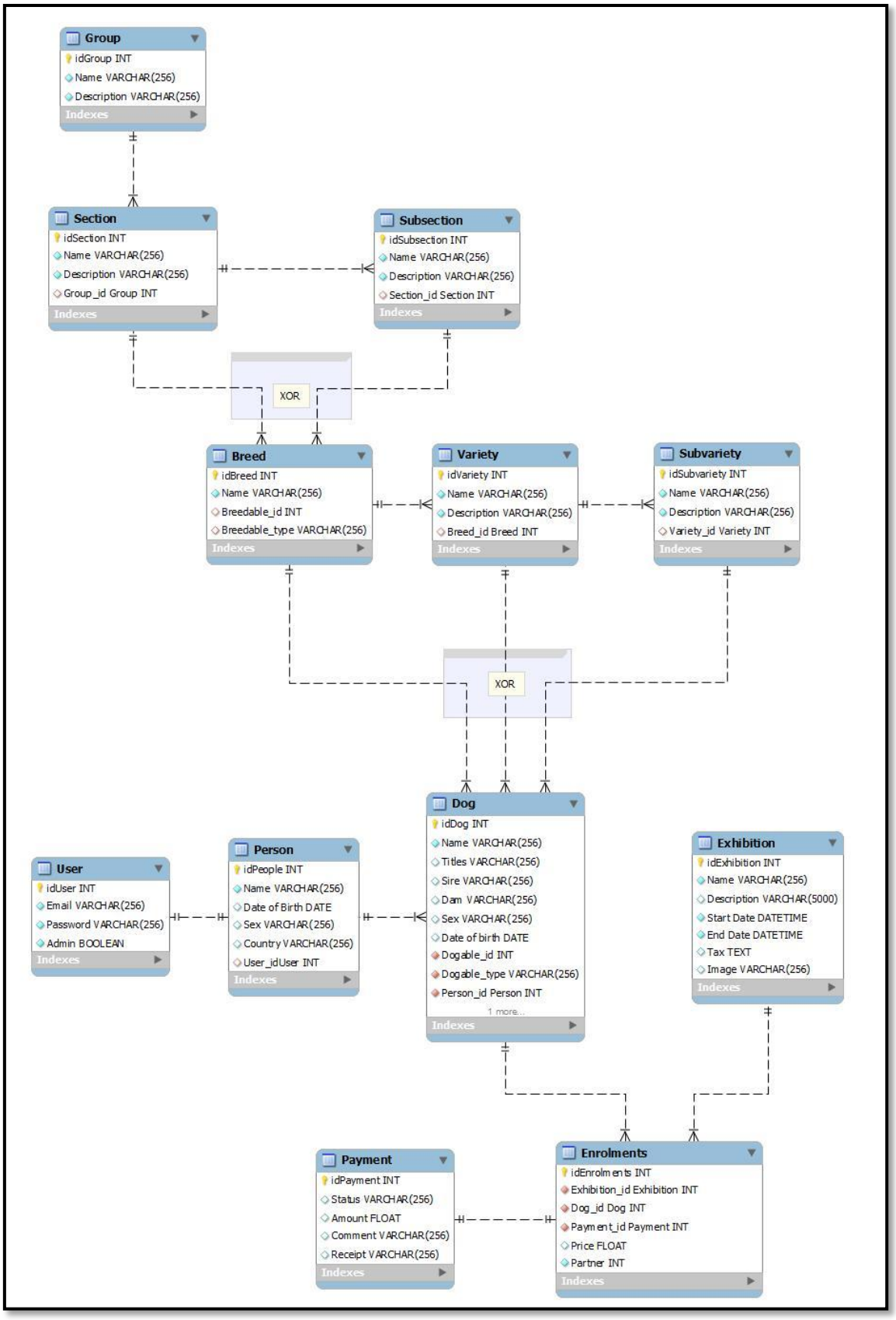


Imagen 9.4.1 – Diagrama Entidad – Relación de la base de datos

Cómo hemos mencionado, y podemos observar, tenemos un total de 12 tablas que vamos a describir a continuación:

- **Group –** Almacena los grupos generales de las razas de los perros.
- **Section –** Esta tabla está relacionada con **Group** y con **Subsection**, y contiene las secciones en que se puede dividir cada grupo. Guarda una relación 1:m con respecto a los grupos y a las subsecciones.
- **Subsection –** Contiene la información de las subsecciones de cada una de las secciones.
- **Breed –** En esta tabla se almacenan cada una de las razas que existen en el sistema. Más adelante detallaremos la relación que existe entre las razas, las secciones y las subsecciones.
- **Variety –** Contiene las variedades que puede tener cada una de las razas. Están relacionadas 1:m con las razas y con las subvariedades.
- **Subvariety –** Tabla que almacena información referente a cada subvariedad que pueda tener las variedades de las razas.
- **Dog –** Se utiliza para guardar los datos referentes a los perros de los usuarios. Está relacionado 1:m con la tabla **Person**. La relación que guarda con las tablas **Breed**, **Variety** y **Subvariety**, podemos observar que es a modo de Or-Exclusivo, hecho que detallaremos una vez terminada la exposición de las tablas.
- **Person –** La tabla **Person** guarda los datos personales de cada usuario y está relacionada con la tabla **User** 1:1.
- **User** – Esta tabla es la creada por la gema *Devise*, y lo que guarda son los datos del usuario en sí: email, contraseña, admin…
- **Exhibition** – Las exhibiciones que haya en el sistema se guardarán en esta tabla. Existe una relación n:m con la tabla **Dog** a través de la tabla **Enrolments**.
- **Enrolments** – En la tabla **Enrolments** se guardan las relaciones que hay entre las exhibiciones y los perros; esto es, que cada entrada de la tabla recoge una inscripción de un perro en una exhibición, junto con el precio y el checkeo de si es partner de la asociación o no.
- **Payment** – Esta tabla guarda una relación 1:1 con los **Enrolments**, y almacena la información respecto a los pagos de los mismos.

Mientras detallamos las tablas hemos comentado dos aspectos que resaltan en el diagrama entidad relación: los dos XOR que podemos ver en el esquema. Estos **Or-exclusivos** es una manera de reflejar gráficamente que las relaciones que solamente puede existir una de las relaciones que ocupan dichos **XOR.** Detallemos, por ejemplo, la relación **XOR** entre **Breed, Section** y **Subsection**:

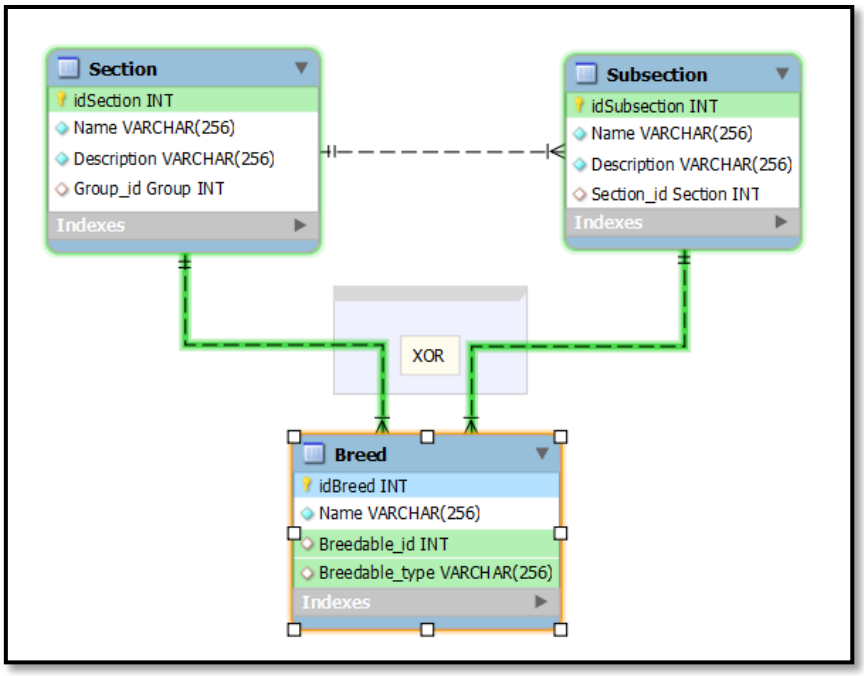


Imagen 9.4.2 – Relación Section-Subsection-Breed

En la imagen vemos que, en la tabla **Breed**, la referencia a las otras dos tablas está reflejada mediante los campos **Breedable_id** y **Breedable_type**. El primero de estos campos nos va a indicar el identificador de una de las entradas de alguna de las otras dos tablas (**Section** o **Subsection**), y el segundo, nos dirá si la relación es con una entrada de la tabla **Section** o una de la tabla **Subsection**. De este modo, una entrada de la tabla Breed tendría este aspecto:

Id	Name	Breedable_id	Breedable_type
1	Chow Chow	5	Section

Y lo que nos estaría indicando, es que la raza Chow Chow pertenece a la "Section" con identificado "5".

De la misma manera se realiza las relaciones entre las tablas **Breed, Variety, Subvariety** y **Dog**. Cada entrada de la tabla perro estará relacionada con una de las otras tablas, que identificaremos mirando el "*Dogable_type*".

Esta es la forma que ActiveRecord gestiona las llamadas asociaciones polimórficas, en las que un modelo puede pertenecer a más de un modelo en una sola asociación. En cuanto al resto del diagrama no hay nada más de relevancia que merezca interés de estudio.

10- Implementación

En este apartado del documento vamos a describir los aspectos de más interés en la fase de implementación de la aplicación. Concretamente vamos a hacer especial hincapié en la estructura de ficheros de la aplicación, en el aspecto del desarrollo pruebas del sistema, y el hash de precios en formato JSON que hemos mencionado varias veces a lo largo del documento.

10.1- Estructura de ficheros

Esta sección está destinada al estudio de la estructura de directorios del proyecto. Vamos a ver la estructura general de directorios, y detallar lo que sucede dentro de cada uno de los directorios relevantes en nuestra aplicación. En la siguiente imagen mostramos dicha estructura y a continuación detallaremos el contenido de estos directorios:

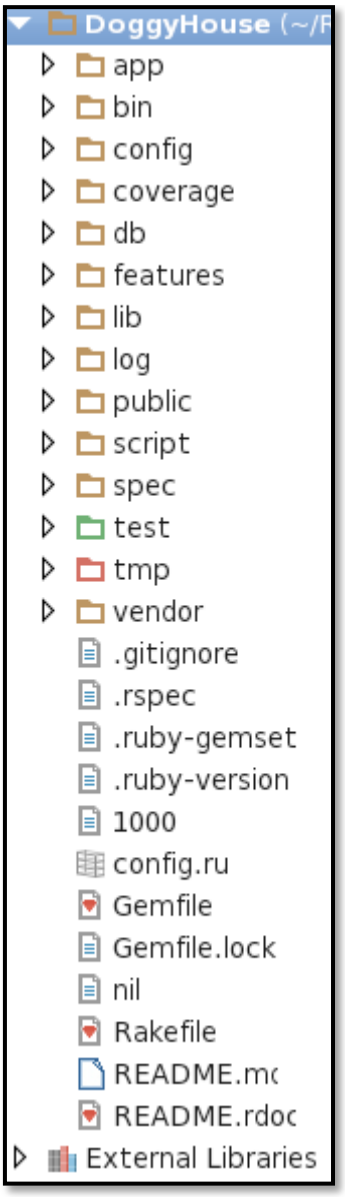
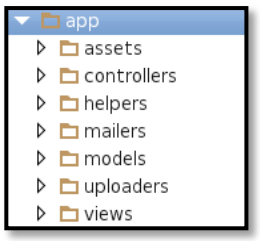


Imagen 10.1.1 – Estructura de directorios

- **App:** En este directorio se organiza la estructura general de la aplicación. Entre otros, tiene subdirectorios que contienen las vistas de la aplicación (*view y helpers*), los controladores (*controllers*) y los modelos de nuestro sistema (*models*).



  - **App/assets:** Contiene los ficheros de implementación de funciones *JavaScript*, CSS y las imágenes que el sistema va a mostrar.
  - **App/controllers:** Es el directorio dónde Rails busca los controladores de nuestro sistema.
  - **App/helpers:** En este directorio nos encontramos con los *helpers* que apoyan a las clases de los modelos, vistas y controladores. El objetivo fundamental de los ficheros que componen este directorio es mantener un código limpio, claro y conciso.
  - **App/models:** En esta carpeta podemos encontrar las clases modelo.
  - **App/uploaders:** En este directorio se almacenan los ficheros con información referente a las características de carga de ficheros.
  - **App/views:** Directorio en el cual están los archivos que contienen la implementación de las vistas de la aplicación. Contiene las plantillas con formato .ERB para completar con los datos de la aplicación, convertir a HTML y devolver al navegador del usuario.
- **Bin:** Contiene archivos binarios ejecutables.
- **Config:** En esta carpeta se guardan archivos que contienen configuración que requiere la aplicación: configuraciones del tester Cucumber (Cucumber.yml) y de la base de datos (database.yml), la estructura del enviroment de Rails (enviroment.rb), el fichero dónde se encuentra todo el enrutado del sistema (routes.rb)…
- **Coverage:** Este directorio contiene configuraciones y resultados acerca de las pruebas realizadas en el proyecto. En él se puede encontrar el archivo que nos da los resultados de cobertura de código que presentamos en el apartado de resultados y conclusiones de este documento.
- **Db:** En la carpeta "db" está el esquema de la base de datos y el fichero de semillas que se utiliza en la aplicación. También posee una subcarpeta en la que se hayan todas las migraciones realizadas durante el desarrollo del proyecto.
- **Features:** Podemos encontrar todos los ficheros correspondientes a las pruebas realizadas con *Cucumber* en este directorio. Cada uno de los ficheros contiene distintos conjuntos de tests de aceptación.

  - **Features/step_definitions:** En esta subcarpeta hallamos los archivos en los que se encuentran las definiciones de los *steps* realizados con *Cucumber*, es decir, la implementación de las pruebas que se encuentran en los archivos del directorio padre.
  - **Features/support:** Archivos de soporte para las pruebas de aceptación.
- **Lib:** Contiene módulos de bibliotecas específicas de la aplicación.

- **Log:** Archivos de log de la aplicación.
- **Public:** En esta carpeta se almacenan los datos accesibles para los usuarios (por ejemplo, los accesibles vía navegador), cómo las imágenes de las exhibiciones, hojas de estilo, los ficheros de pagos…
- **Script:** Aquí están los scripts para lanzar y administrar herramientas que se utilizan con Rails, como por ejemplo de *Cucumber*.
- **Spec:** Al igual que *Cucumber* tenía su propia carpeta para almacenar los ficheros de prueba realizados, *RSpec* utiliza esta carpeta para guardar sus ficheros. Hay subdirectorios para pruebas de controladores, modelos, vistas, helpers… En nuestro caso, solamente utilizamos *RSpec* con la finalidad de testear el fichero semilla de la aplicación.
- **Test**: Se utiliza para las pruebas realizadas con Minitest. En nuestro no hemos realizado pruebas con esta herramienta, por lo que no se ha modificado este directorio. Contiene subdirectorios para pruebas de controladores, helpers, mailers…
- **Vendor:** Códigos de terceros tales como plugins y gemas.
- **External Libraries:** Aquí se encuentran todas las librerías de las que hace uso la aplicación, como por ejemplo, cada una de las librerías de las gemas que utilizamos.

10.2- Desarrollo de las pruebas

En el apartado 8, el apartado en que hacíamos un análisis de lo que va a ser nuestro sistema, veíamos algunas de las pruebas que se han realizado sobre el software en cada una de las iteraciones que ha constado el proyecto. En este apartado detallaremos algunas de las pruebas más relevantes y más laboriosas que hemos realizado durante el desarrollo. Las pruebas que veíamos en el citado apartado (concretamente en la sección 8.3) eran pruebas realizadas para el cliente, en un lenguaje (**Gherkin**) que el cliente, sin ser especialista informático, debería poder leer y comprender. Cada una de las sentencias que se veían en estas pruebas, tienen una implementación interna que será realmente cómo se comprueba que la prueba debería pasar. En esta sección vamos a estudiar un poco acerca de estas implementaciones.

10.2.1- Cucumber

En la sección 8.3 estudiábamos ciertas pruebas que hacíamos durante las iteraciones en las que ha consistido el proyecto. Estas pruebas, realizadas con la herramienta Cucumber, eran la base en la que el sistema debía de irse construyendo. Lo que implementásemos respecto a nuestro sistema debía superar las pruebas que el cliente previamente había comprobado y aceptado. Cómo hemos hablado, estas pruebas estaban en un lenguaje que el cliente podía entender y comprender (**Gherkin**) y se realizaban mediantes los llamados "*steps*". Sin embargo, por debajo de estas sentencias había una implementación de dichas sentencias, que es la verdadera comprobación que el sistema debe superar.

Para empezar, vamos a ver una de las primeras implementaciones de los *steps* que tratábamos en la sección 8.3.4.2, la de **Viewing groups**. Vamos a recordar el fichero *.feature* que el cliente debía ser capaz de entender:

```
Feature: Viewing groups
  In order to view groups
  As a user
  I want to see them on the groups' page

  Background:
    There are groups with a name and a description
    Given groups with these entries:
      |Name       |Description                                          |
      |Group I    |Sheepdogs and Cattle Dogs (except Swiss Cattle Dog)|
      |Group III  |Terriers                                             |
      |Group IX   |Companion and Toy Dogs                               |

  @done
  Scenario: A visit wants see what groups are defined
    When the standards and nomenclature page is visited
    Then the standards and nomenclature page should content:
      |Name       |Description                                          |
      |Group I    |Sheepdogs and Cattle Dogs (except Swiss Cattle Dog)|
      |Group III  |Terriers                                             |
      |Group IX   |Companion and Toy Dogs                               |

  @done
  Scenario: Viewing a specific group properties
    When the standards and nomenclature page is visited
    And I follow "Group I"
    Then I should be on the group page for "Group I"
    And I should see "Group I" within "#groups h2"
```

Como vemos, a excepción del tag *@done* (del cuál hablábamos en la sección 8.4.3.1), es bastante comprensible e intuitivo para cualquier persona que no pertenezca al sector de la informática (y que sepa inglés).

La forma que tiene **Cucumber** de "*correr*" los tests, es, cuándo ejecutamos la sentencia en el terminal ("Cucumber --tag … ruta_del_fichero") se empieza a ejecutar el fichero de prueba que hemos indicado, y se va buscando la definición de cada paso (*step*) en su estructura interna y en los ficheros de definiciones realizados por el programador (dentro del directorio step_definitions). Si encuentra algo que no concuerde o definiciones repetidas, fallos de sintaxis, etc, nos mostrará el correspondiente mensaje de error.

Cómo hemos nombrado, **Cucumber** busca las definiciones de los *steps,* en su estructura interna, es decir que hay varias sentencias de definiciones de pasos que **Cucumber** nos ofrece ya implementadas. No son muchas, pero sí que nos ahorra esfuerzos, por ejemplo, en el caso de rellenar formularios o acciones como apretar botones, visitar links…; de este modo, algunas de las sentencias que Cucumber nos trae ya implementadas son:

```
I follow "..."
I should see "..."
I select "..."
I check "...", etc.
```

Sin embargo, la mayoría de los *steps* se deben definir de forma completa. En nuestro caso, particularizando en una iteración, y aprovechando que ya la hemos estudiado en la sección 8.3.4, vamos a ver el fichero de definición de los *steps* en cuanto a los grupos:

```
Given(/^groups with these entries:$/) do |table|
 table.map_headers!('Name' => :name, 'Description'=> :description)
 table.hashes.map{|hash| FactoryGirl.create(:group, hash)}
end

When(/^the standards and nomenclature page is visited$/) do
 visit 'groups'
end

Then(/^the standards and nomenclature page should content:$/) do |expected_table|
  rows = find('table#groups').all('tr')
  actual_table = rows.map{ |row| row.all('th, td').map{ |cell| cell.text} }
  expected_table.diff!(actual_table)
end

Given(/^there is a group called "(.*?)"$/) do |name|
  @group = FactoryGirl.create(:group, name: name)
end

Given(/^there is a group which description is "(.*?)"$/) do |description|
  @group = FactoryGirl.create(:group, description: description)
end

Then(/^all sections for this group should have been removed$/) do
  @group.sections.count.should eq 0
end

Given(/^I try to delete the "(.*?)" "(.*?)"$/) do |model, name|
  page.driver.submit :delete,
"#{model.capitalize.constantize.find_by_name(name).id}", {}
end
```

Lo primero que podemos notar, es que la sintaxis ya no es comprensible para cualquier persona o usuario, es una sintaxis, con sentencias más especializada para el gremio informático. Vemos en el cuadro, en el caso particular de los grupos, que en este fichero tenemos definidas las sentencias "*groups with these entries*", "*the standars and nomenclature page is visited*", "*the standars and nomenclature page should content*", "*there is a group called "…""*, "*there is a groups which description is "…""*… Cada una de estas definiciones están implementadas de acuerdo cómo queremos que la prueba sea exitosa. Por ejemplo, en el primer caso:

```
Given(/^groups with these entries:$/) do |table|
 table.map_headers!('Name' => :name, 'Description'=> :description)
 table.hashes.map{|hash| FactoryGirl.create(:group, hash)}
end
```

Estamos definiendo este *step* del **`Background`**:

```
Given groups with these entries:
  |Name       |Description                                          |
  |Group I    |Sheepdogs and Cattle Dogs (except Swiss Cattle Dog)|
  |Group III  |Terriers                                             |
  |Group IX   |Companion and Toy Dogs                               |
```

Y lo que estamos haciendo es, con la primera sentencia, mapear la tabla que recibimos para renombrar los campos *name* y *description*, y que coincidan con los nombres de los campos de nuestra base de datos. En segundo lugar, dado que este *step* lo que nos indica es "que existen grupos con estas entradas", mediante la gema **FactoryGirl** (nombrada en la sección 7.3.15) creamos registros "virtuales" de dichos grupos, para que, durante la prueba, estos grupos existan en el sistema y podamos probar el correcto funcionamiento del sistema en base a ellos. Estos registros se eliminarán una vez todas las pruebas del fichero han concluido.

En la segunda definición, lo que se hace es definir cómo se debe visitar la página "*standards and nomenclature*" (en nuestro caso la página "*standards and nomenclature*" corresponde al *index* de los grupos):

```
When(/^the standards and nomenclature page is visited$/) do
 visit 'groups'
end
```

Y esto corresponde a la implementación de la sentencia:

```
When the standards and nomenclature page is visited
```

La tercera definición es algo más compleja. Lo que buscamos es comprobar que realmente, dado que existen los grupos que hemos visto en el **Background**, estamos visualizando exactamente esa tabla con esos datos y no con otro contenido:

```
Then(/^the standards and nomenclature page should content:$/) do |expected_table|
  rows = find('table#groups').all('tr')
  actual_table = rows.map{ |row| row.all('th, td').map{ |cell| cell.text} }
  expected_table.diff!(actual_table)
end
```

Mediante un mapeo doble de la tabla que queremos obtener y una transposición de las filas, lo que conseguimos es que, si la tabla que estamos viendo no es la misma que la que se espera ver, nos muestre el error correspondiente, indicando que "algo" no va bien en nuestro sistema. Esta definición correspondería al *step*:

```
Then the standards and nomenclature page should content:
  |Name       |Description                                          |
  |Group I    |Sheepdogs and Cattle Dogs (except Swiss Cattle Dog)|
  |Group III  |Terriers                                             |
  |Group IX   |Companion and Toy Dogs                               |
```

Este sería el procedimiento a hacer con cada uno de los *steps* de cada uno de los ficheros que componen el sistema de pruebas. En el fichero de definiciones de pasos vemos

que hay algunas implementaciones que no corresponden a ninguna de las sentencia del fichero de prueba mostrado (*viewing groups*). Esto es porque en el fichero están recogidos todos los *steps* respecto a los grupos (*steps* de crear, borrar, ver y actualizar). Por poner un ejemplo, la definición:

```
Given(/^I try to delete the "(.*?)" "(.*?)"$/) do |model, name|
  page.driver.submit :delete,
"#{model.capitalize.constantize.find_by_name(name).id}", {}
end
```

Correspondería con la siguiente prueba de borrado de grupos (la cual está ubicada en el fichero *deleting_groups.feature*):

```
  @done
  Scenario: Deleting groups as no admin is bad
    Given I am signed in as "user@testing.com"
    And I am on the groups page

    When I follow "Group I"
    And I try to delete the "group" "Group I"

    Then I should be redirected to the home page
    And I should see "You can't access to this page."
```

Lo que comprobamos básicamente en esta prueba es que, cuándo mandamos la orden de borrado de grupos (*page.driver.submit*), debemos ser redireccionados a la página principal, ya que hemos indicado que no somos administradores del sistema (`Given I am signed in as user@testing.com`). Este *step*, al igual que la mayoría del código de la aplicación, ha sido definido en el "mandato" **DRY** de RoR ya que en concreto éste lo utilizamos en distintos *steps* de ficheros de pruebas; por ejemplo, para las exhibiciones también hacemos uso de esta misma definición:

```
  @done
  Scenario: Deleting exhibitions as no admin is bad
    Given I am signed in as "user@testing.com"
    And I am on the exhibition page for "9th Annual Dog Show"

    When I try to delete the "exhibition" "9th Annual Dog Show"

    Then I should be redirected to the home page
    And I should see "You can't access to this page."
```

Esta sentencia también invoca a la misma definición mostrada anteriormente, de forma que evitamos repetir código de testeo para acciones casi idénticas. De esta manera conseguimos un código más claro y conciso, evitando, como hemos dicho, repeticiones innecesarias.

Observando las implementaciones mostradas en este apartado podemos pensar que no es algo tan complicado el implementar los *steps* de las pruebas. Realmente, una vez familiarizado con el entorno, las características que nos ofrece, las limitaciones que tiene, el lenguaje propio de definiciones y demás aspectos claves, no es un tarea demasiado complicada (que no quita que sea tediosa). Sin embargo, en nuestro caso particular,

acordémonos de que lo que hemos estado viendo en esta sección es respecto a las primeras iteraciones del proyecto, en el que no es estábamos familiarizando con el entorno de desarrollo y la metodología; La evolución de las pruebas durante la fase de desarrollo es bastante notoria; de hecho, cómo nombramos anteriormente, en el ANEXO II podemos encontrar, a modo de ejemplo, uno de los ficheros de pruebas de la iteración 12 del proyecto (concretamente el correspondiente a las pruebas de *"ver enrolments"*), y en el ANEXO III, veremos el fichero de implementación de los *steps* correspondientes a las pruebas de los "*enrolments"*. Observando estos ficheros podemos percibir rápidamente la evolución de la complejidad de las pruebas con respecto a las primeras iteraciones del proyecto que hemos estudiado en esta sección.

10.2.2- Pruebas con Rspec

Rspec, cómo hemos estudiado, es otra herramienta de testeo que utilizamos en la elaboración del proyecto. En nuestro caso, **Rspec** se utilizó para testear que, en el fichero semilla (*db/seeds.rb*), esté correctamente definida la estructura de estándares y nomenclaturas de raza. El fichero de semillas es un fichero que se debe ejecutar antes de correr por primera vez la aplicación, y lo que contiene es la información básica que nuestro sistema debe poseer al iniciarse. En esta sección vamos a ver una pequeña porción de código que prueba la correcta implementación de este fichero. No lo veremos completamente ya que el fichero es bastante extenso y no tiene mayor interés el mostrarlo ni estudiarlo por completo.

El archivo lo podemos encontrar al completo en la ruta "*spec/features/seeds_spec.rb*", y en él, como hemos mencionado, están definidas las pruebas unitarias que deben comprobar que los estándares y nomenclaturas del fichero semilla están correctamente definidos. Debemos tener en cuenta que ya no hablamos de tests que deben ser entendidos por el usuario o propietario del sistema, no son tests de aceptación sino que son tests de validación del sistema. Al empezar a revisar este fichero, nos encontramos con las primeras sentencias que vamos a estudiar:

```
describe '#Groups' do
  specify 'should have 10 items' do
    Group.count.should eq 10
  end

  [['Group I', 2],
   ['Group II', 3],
   ['Group III', 4],
   ['Group IV', 1],
   ['Group V', 8],
   ['Group VI', 3],
   ['Group VII', 2],
   ['Group VIII', 3],
   ['Group IX', 11],
   ['Group X', 3],
  ].each do |name, count|
    it "#{name} should have #{count} sections" do
      Group.find_by(name: name).sections.count.should eq count
    end
  end
end
```

Con esta definición lo que comprobamos es que, en el fichero semilla, debe haber definido 10 grupos, en los que cada uno de ellos tiene que tener un determinado número de secciones: el *Group* I debe tener 2 secciones, el *Group* II 3, el *Group* III tiene que tener 4… Previamente a esto, se ha cargado el fichero semilla para la ejecución de las pruebas:

```
before(:all) { load Rails.root + 'db/seeds.rb' }
```

A continuación, lo que se comprueba es que, para cada sección existe el número correcto de subsecciones, razas, variedades y subvariedades. Por ejemplo, en el cuadro que vemos a continuación, indicamos que la *Section 1* del *Group II* debe tener 4 subsecciones, y que cada una de éstas debe tener 5 razas, 3 razas 1 raza y 1 raza respectivamente.

```
# GROUP II #
############
describe 'Group II' do
  # Section 1
  it 'Section 1 should have 4 subsections' do
    Group.find_by(:name => 'Group II').sections.find_by(:name => 'Section
    1').subsections.count.should eq 4
  end

  it 'Subsection 1 from Section 1 should have 5 breeds' do
    Group.find_by(:name => 'Group II').sections.find_by(:name => 'Section 1').
        subsections.find_by(:name => 'Subsection 1.1').breeds.count.should eq 5
  end

  it 'Subsection 2 from Section 1 should have 3 breeds' do
    Group.find_by(:name => 'Group II').sections.find_by(:name => 'Section 1').
        subsections.find_by(:name => 'Subsection 1.2').breeds.count.should eq 3
  end

  it 'Subsection 3 from Section 1 should have 1 breeds' do
    Group.find_by(:name => 'Group II').sections.find_by(:name => 'Section 1').
        subsections.find_by(:name => 'Subsection 1.3').breeds.count.should eq 1
  end

  it 'Subsection 4 from Section 1 should have 1 breeds' do
    Group.find_by(:name => 'Group II').sections.find_by(:name => 'Section 1').
        subsections.find_by(:name => 'Subsection 1.4').breeds.count.should eq 1
  end
```

Como explicación grosso modo del cuadro mostrado arriba, podemos ver que estamos describiendo la prueba para *Group II* (**`describe 'Group II' do`**) y, dentro de esta prueba, indicamos que la sección 1 debe contener 4 subsecciones (**`it 'Section 1 should have 4 subsections' do`**). Para comprobarlo lo que se hace es buscar el *Group II* (**`Group.find_by(:name => 'Group II')`**), la *Section 1* dentro de éste (**`sections.find_by(:name => 'Section1')`**) y que el conteo de las subsecciones de ésta debe de ser igual a 4 (**`subsections.count.should eq 4`**). La siguiente sentencia comprueba, de forma similar a la explicada, que las razas dentroe de la subsección 2 de la sección 1 deben ser 5.

De esta forma, y para cada grupo, comprobamos que todo está definido correctamente en el fichero de semillas.

También comprobamos que las variedades, subvariedades y razas están correctamente definidas y nombradas. Así pues, por ejemplo para el *Group VII*, esta sería la prueba correspondiente de la comprobación de las variedades:

```
# GROUP VII #
#############
describe 'Group VII' do
  context 'should have (4) varieties.' do
    # Subsection 1.1
    it 'In subsection 1.1 there are 4 varieties' do
      subsec_varieties =
      [
        ['Weimaraner',2],
        ['Bracco Italiano',2]
      ].inject(0) do |acc, (name, count)|
        Group.find_by!(:name => 'Group VII').sections.find_by!(:name => 'Section 1').
            subsections.find_by!(:name => 'Subsection 1.1').
            breeds.find_by!('name LIKE ?', "%#{name}%").varieties.count.should eq count
        acc + count
      end
      subsec_varieties.should eq 4
    end
  end
end
```

La comprobación, como vemos, la hacemos utilizando un hash que contiene la información que queremos cotejar, y vamos iterando sobre él. En cada iteración separamos el hash en dos campos que vamos a utilizar dentro de las mismas: el campo *name* (para indicar el nombre de la raza) y *count* (para indicar la cantidad de variedades que tiene que tener la raza).

```
subsec_varieties =
[
  ['Weimaraner',2],
  ['Bracco Italiano',2]
].inject(0) do |acc, (name, count)|
```

De este modo, buscamos en el fichero semilla el grupo correspondiente (**`Group.find_by!(:name => 'Group VII')`**), la sección y la subsección pertinentes (**`sections.find_by!(:name => 'Section 1').subsections.find_by!(:name => 'Subsection 1.1')`**), y el nombre de la raza que hemos indicado en el hash (**`breeds.find_by!('name LIKE ?', "%#{name}%")`**), cuyo campo ***name*** será el proporcionado por el primer campo del hash de cada iteración (`'Weimaraner'` y `'Bracco Italiano'`). Una vez encontrada la raza (y por tanto confirmado que los nombres están correctamente escritos, ya que si los encuentra es, lógicamente, porque están), comprobamos que las variedades de estas razas son iguales al segundo campo proporcionado por el hash (**`varieties. count.should eq count`**), que, en este caso, en ambas razas es 2. La última sentencia de la prueba comprueba que el número total de variedades en este grupo debe ser exactamente 4: "**`subsec_varieties.should eq 4`**".

Esta es la manera básica que comprobamos que los estándares y nomenclaturas están correctamente definidos en el fichero semilla. No corresponde alargar más la explicación del

fichero, ya que el concepto de la prueba respecto a esto ha quedado suficientemente estudiado con lo que hemos visto.

10.3- *Hash* de precios

En la especificación de requisitos del software, en la sección 8.2.2, nombrábamos que, para almacenar la tabla de precios de las exhibiciones caninas, hemos utilizado una estructura hash que almacenamos serializándola en formato JSON. Se ha decido almacenar las tablas de precios en formato JSON, principalmente, para no alargar el desarrollo del prototipo de la aplicación. Podríamos haber decido implementar esta estructura en un formato de tablas y relaciones, en el que todo quedase bien identificado y definido. Sin embargo, para nuestro fin actual, nos hace buena función el guardar esta información en un formato de texto plano, sin necesidad de tablas o relaciones entre las mismas.

Este sistema de almacenamiento de precios tiene sus convenientes y sus inconvenientes, como toda decisión. Por la parte positiva, tenemos la ventaja de que nos proporciona lo que queremos sin necesidad de estar ampliando el diseño de nuestra base de datos. Además, esto lo conseguimos de una forma clara y facilidad de uso, con la ventaja añadida de que es un formato independiente de cualquier lenguaje de programación, por lo que los servicios que comparten información por este método no necesitan "hablar" el mismo idioma, es decir, el emisor puede ser Java y el receptor PHP.

Uno de los inconvenientes principales en el uso de este estándar es que es necesario conocer la estructura interna del hash para poder utilizarlo. Es por ello que se ha decidido incluir en esta sección la representación de dicha estructura. El administrador del sistema, al crear una nueva exhibición, deberá introducir de forma manual y bajo el formato que vamos a especificar a continuación, la información correspondiente a los precios de la misma.

Realmente, lo ideal podría considerarse que no hiciese falta conocer la estructura interna de cómo almacenar la tabla de precios. Sin embargo, la intención de acotar el desarrollo del proyecto nos ha llevado por el camino de decidir utilizar una estructura JSON para almacenar esta tabla de precios. No obstante, pese a esto, el prototipo del sistema no deja de ser utilizable, y estar totalmente operativo y bien construido. Esto es, simplemente, una forma menos tediosa de desarrollar lo que se deseaba conseguir.

Dicho esto y aclarado el porqué de la elección de un formato JSON para almacenar los precios de las exhibiciones, vamos a estudiar la estructura que debe poseer el hash mediante un ejemplo real de precios de una exhibición canina.

Antes de nada, aclarar el concepto de hash. Un hash no es más que una estructura que contiene un par clave-valor, en el que dicho valor contiene más información (como pueden ser otros hashes). Nuestro hash lo vamos a definir como un hash típico en formato de Ruby. Luego, mediante la sentencia "*to_json*", lo almacenaremos en formato JSON en la base de datos y lo trataremos como tal. Este hash se va a dividir en tres "sub-hashes" principales: Un hash para indicar los **grupos** que hay en la exhibición, otro hash para indicar los ***deadlines***, y otro hash para indicar los **precios** en la exhibición. De este modo, en un mismo hash, tendremos toda la información necesaria para calcular los precios de inscripción de los perros.

Teniendo en cuenta que el campo de las exhibiciones que almacena esta información en la base de datos se llama "**tax**", de una manera simbólica tendríamos el siguiente hash principal:

@tax = {**groups**: [group_definition], **deadlines**: [deadline_definition], **prices**: [prices_definition]}

Y luego, cómo hemos nombrado, mediante un "@tax.to_json", almacenaríamos toda la información en un formato JSON en la base de datos, teniendo de esta manera todos los datos referentes a los precios en un único campo dentro la tabla de las exhibiciones.

A su vez, dentro de cada uno de estos 3 "sub-hashes", nos encontraremos definidas más estructuras internas a cada uno. Empecemos estudiando el hash de "**groups**". Este hash contendrá, en grupos, las clases de la exhibición que tienen el mismo precio de inscripción para los perros. Incluirá un hash por cada grupo de clases que exista; a modo de ejemplo, si tenemos esta tabla de precios de la **Exhibición Canina de Cieza**:

Primer Plazo		Segundo Plazo		Tercer Plazo	
Apertura	Cierre	Apertura	Cierre	Apertura	Cierre
08/01/2014	22/02/2014	23/02/2014	03/03/2014	04/03/2014	14/03/2014

La fecha del ingreso o pago de la inscripción determina la tarifa aplicable.

	PRECIOS INSCRIPCIONES					
	:: SOCIOS			:: NO SOCIOS		
Clase	Plazo 1	Plazo 2	Plazo 3	Plazo 1	Plazo 2	Plazo 3
1º PERRO C.J. C.I. C.A. C.T. C.CH.	24.00€	30.00€	36.00€	30.00€	37.50€	43.50€
2º PERRO C.J. C.I. C.A. C.T. C.CH.	19.00€	23.00	27.00€	25.00€	32.00€	37.00€
3º P. y siguientes C.J. C.I. C.A. C.T. C.CH.	14.00€	18.00€	21.00€	20.00€	25.00€	29.00€
1º PERRO CLASE VETERANOS	12.00€	15.00€	18.00€	15.00€	18.75€	21.75€
2º PERRO CLASE VETERANOS	9.50€	11.50€	13.50€	12.50€	16.00€	18.50€
3º P. CLASE VETERANOS y siguientes	7.00€	9.00€	10.50€	10.00€	12.50€	14.50€
1º CACHORRO	16.00€	20.00€	24.00€	21.00€	27.00€	31.00€
2º CACHORRO	12.00€	15.00€	18.00€	15.00€	19.00€	21.00€
3º CACHORRO y siguientes	9.00€	12.00€	15.00€	11.00€	14.00€	17.00€
CLASE PAREJAS	20.00€	25.00€	29.00€	25.00€	32.00€	37.00€
LOTE O GRUPO DE CRIA	Gratis	Gratis	Gratis	Gratis	Gratis	Gratis

Imagen 10.3.1 – Tabla de precios Exposición Canina de Cieza 2014

Tendremos 5 grupos de clases con mismos precios (y por tanto 5 hashes): El correspondiente a las clases "*C.J*, *C.I*, *C.A*, *C.T* y *C.CH*", el correspondiente a la clase "*veteranos*", el que corresponde a la clase "*cachorro*", y los grupos de precios para las "*clases parejas*" y "*grupo de cría*". Por lo tanto, el hash de "**groups**" sería:

```
groups: [
    {name: group1, classes: [Junior, Intermediate, Open, Working, Champion]},
    {name: group2, classes: [Veteran]},
    {name: group3, classes: [Puppy]},
    {name: group4, classes: [Couple]},
    {name: group5, classes: [Group Breeding]}
]
```

Obteniendo de esta manera una forma de representar cada uno de los grupos de clases que tienen los mismos precios en las exhibiciones.

El siguiente "sub-hash" que veíamos en la estructura inicial era el "**deadlines**". En este hash, representaremos los plazos de fechas que aparecen en la parte superior de la imagen 10.3.1 (Primer Plazo, Segundo Plazo y Tercer Plazo). Dentro de este hash también tendremos una estructura interna por cada plazo de inscripción. En nuestro ejemplo habrá 3 plazos de inscripción, por lo que tendremos 3 hashes internos:

```
deadlines:[
    {name: 1st entry deadline, start_date: 08/01/2014, end_date: 22/02/2014},
    {name: 2nd entry deadline, start_date: 23/02/2014, end_date: 03/03/2014},
    {name: 3rd entry deadline, start_date: 04/03/2014, end_date: 14/03/2014}
]
```

Y de esta manera tendremos organizados cada uno de los plazos de inscripción de la exhibición a tratar.

El último "sub-hash" de la estructura principal, el referente a los "**prices**" será el más complicado de especificar, ya que su estructura es un poco más compleja. Para empezar, teniendo en cuenta que en las exhibiciones puede haber varios conjuntos de precios dependiendo si eres socio (*partners*), o no (*nopartners*), se ha decidido crear dos grupos dentro del hash, idénticos, pero variando simplemente los precios. Por ello, el hash de **prices** empieza construyéndose así:

```
prices:[
    {partners: [...]},
    {nopartners: [...]}
]
```

Como ambos grupos son idénticos, vamos a detallar solamente el grupo de los **partners** (en el caso de que en las exhibiciones solamente haya un bloque de precios, se considera dicho bloque como "*nopartners*"). La estructura es la siguiente:

```
partners: [
    {group1: [ [24.00, 19.00, 14.00], [30.00, 23.00, 18.00], [36.00, 27.00, 21.00] ]},
    {group2: [ [12.00, 9.50, 7.00], [15.00, 11.50, 9.00], [18.00, 13.50, 10.50] ]},
    {group3: [ [16.00, 12.00, 9.00], [20.00, 15.00, 12.00], [24.00, 18.00, 15.00] ]},
    {group4: [ [20.00], [25.00], [29.00] ]},
    {group5: [ [0.00], [0.00], [0.00] ]}
]
```

Al principio parece complicado, pero vamos a estudiarla y veremos que es más fácil de entender de lo que parece. Básicamente no es más que la tabla de precios de la imagen 10.3.1, pero transpuesta por grupos. En el hash de **partners,** tendremos tantos "sub-hashes" como grupos de clases tenga el hash **groups**, y cada uno de estos "sub-hashes" corresponde a cada grupo del hash **groups**. Por tanto, group1 serán los precios para las clases [Junior, Intermediate, Open, Working, Champion]; group2 serán los precios para la clase [Veteran]; group3 para [Puppy], y así sucesivamente. Además, cada uno de estos sub-hashes representados en azul, vemos que tiene 3 arrays de valores. Cada vector de valores corresponde a los precios de cada una de las **deadlines** que tiene la exhibición, es decir:

[24.00, 19.00, 14.00] Corresponden con los precios que puede tener la 1st entry deadline.
[30.00, 23.00, 18.00] Corresponden con los precios que puede tener la 2nd entry deadline.
[36.00, 27.00, 21.00] Corresponden con los precios que puede tener la 2nd entry deadline.

Y del mismo modo para el group2, group3, group4 y group5. Cada elemento que contiene estos *arrays* corresponde al precio del perro i-*ésimo* que se inscribe para ese grupo; por ejemplo, en el primer *array* del group1:

[24.00, 19.00, 14.00] El primer perro que inscriba me costará 24 euros, el segundo perro me costará 19, y el tercero y sucesivos 14 euros.

En otras palabras, que si estoy en la 1st entry deadline (entre el día 08 enero y 22 de febrero del 2014), el primer perro que inscriba en la exhibición perteneciente a las clases Junior, Intermediate, Open, Working o Champion me costará 24 euros. El segundo perro me costará 23 euros y el tercer (y sucesivos) perros 14 euros. Si los inscribo en la 2nd entry deadline (si estamos entre los días 23 de febrero y 3 de marzo), me costará 30, 23 y 18 euros, el primer, segundo y tercer perro de estas clases respectivamente. Y así sucesivamente.

Podemos observar en el cuadro que el group4 y group5 difieren un poco a los 3 grupos anteriores. Se puede ver que, aunque también tienen 3 *arrays* (uno por cada **deadline**), solamente hay un elemento dentro de cada *array*. ¿Por qué? Si atendemos a lo que hemos explicado, cada elemento dentro los *arrays* corresponde al precio del perro i-*ésimo* que inscribo dentro de este grupo de clases. Si observamos los dos últimos grupos de la imagen 10.3.1:

CLASE PAREJAS	20.00€	25.00€	29.00€
LOTE O GRUPO DE CRIA	Gratis	Gratis	Gratis

Imagen 10.3.2 – Clase Parejas / Grupo de cría Exp. Canina Cieza

No hay perros i-*ésimos*, solamente hay un único precio para cada **deadline**. Da igual que inscriba a 1 o 10 perros, el perro por cada uno es exactamente el mismo. Por ello no hay más que un elemento dentro de cada array del group4 y group5:

```
{group4: [ [20.00], [25.00], [29.00] ]},
{group5: [ [0.00], [0.00], [0.00] ]}
```

Realmente esto es porque las clases parejas es para 2 perros, machos y hembra de la misma raza y variedad, y el grupo de cría es para 3 o más ejemplares de la misma raza o variedad, pero eso son aspectos de reglamentos internos a las exhibiciones de los que no viene a cuento mayor explicación.

La estructura de los no socios sería exactamente igual, pero con los precios para ellos.

De esta manera, el hash del tax para la Exposición Canina de Cieza sería tal que así:

```
tax ={
    groups: [
        {name: group1, classes: [Junior, Intermediate, Open, Working, Champion]},
        {name: group2, classes: [Veteran]},
        {name: group3, classes: [Puppy]},
        {name: group4, classes: [Couple]},
        {name: group5, classes: [Group Breeding]}
    ],
    deadlines: [
        {name: 1st entry deadline, start_date: 08/01/2014, end_date: 22/02/2014},
        {name: 2nd entry deadline, start_date: 23/02/2014, end_date: 03/03/2014},
        {name: 3rd entry deadline, start_date: 04/03/2014, end_date: 14/03/2014}
    ],
    prices: [
        {partners: [
            {group1: [ [24.00, 19.00, 14.00], [30.00, 23.00, 18.00], [36.00, 27.00, 21.00] ]},
            {group2: [ [12.00, 9.50, 7.00], [15.00, 11.50, 9.00], [18.00, 13.50, 10.50] ]},
            {group3: [ [16.00, 12.00, 9.00], [20.00, 15.00, 12.00], [24.00, 18.00, 15.00] ]},
            {group4: [ [20.00], [25.00], [29.00] ]},
            {group5: [ [0.00], [0.00], [0.00] ]}
        ]},
        {nopartners: [
            {group1: [ [30.00, 25.00, 20.00], [37.50, 32.00, 25.00], [43.50, 37.00, 29.00] ]},
            {group2: [ [15.00, 12.50, 10.00], [18.75, 16.00, 12.50], [21.75, 18.50, 14.50] ]},
            {group3: [ [21.00, 15.00, 11.00], [27.00, 19.00, 14.00], [31.00, 21.00, 17.00] ]},
            {group4: [ [25.00], [32.00], [37.00] ]},
            {group5: [ [0.00], [0.00], [0.00] ]}
        ]}
    ]
}
```

El formato final de la estructura del hash no sería exactamente el mostrado pero, para no complicar aún más la explicación, se ha optado por mostrarla utilizando esta sintaxis. La estructura general sí que es idéntica a cómo la hemos estudiado y en el Anexo IV se expone el formato y la sintaxis real de la estructura para esta misma exposición. Este formato se puede utilizar como formato base para cualquier definición de precios de inscripciones en exhibiciones caninas.

11- Resultados y conclusiones

En este apartado del documento vamos a recoger los resultados que se han obtenido de nuestro sistema, así como las conclusiones que derivan del trabajo realizado durante todo el trabajo realizado. En primer lugar detallaremos los resultados obtenidos, y para eso utilizaremos los datos que nos proporcionan aplicaciones (o gemas) de testeo que hemos utilizado en el desarrollo del mismo.

11.1- Resultados

Los resultados que vamos a ver a continuación son unos resultados obtenidos con un trabajo realizado en aproximadamente 1.100 horas a lo largo de todo el desarrollo el proyecto de final de carrera. Hay que tener en cuenta algo que se ha mencionado en varias ocasiones a lo largo de este documento, y es que la metodología de desarrollo, así como la mayoría de las herramientas utilizadas en el proyecto, eran totalmente desconocidas para el estudiante, por lo que el aprender a manejar estas herramientas, así como el manejo de metodologías de desarrollo basadas en pruebas o incluso el uso del paradigma **MVC,** ha llevado un tiempo más que considerable empleado en el estudio de esto. Además, se ha iniciado al alumno desde una programación a bajo nivel, es decir, por poner un ejemplo: Rails posee una capacidad de *scaffolding* que, mediante un simple comando (*rails generate scaffold* "…"), nos crea una estructura de modelo, vistas, controlador y carpetas de tests básicos, e implementa un sistema que permite operaciones **CRUD** básicas. Sin embargo, se ha denegado este uso al alumno y se le ha aconsejado que directamente implemente él cada uno de los modelos, vistas y controladores del sistema, de modo que de esta forma se ha conseguido que entienda la metodología Rails desde un aspecto de programación a bajo nivel (en el contexto de uso de un framework). Lo que se espera, y se ha conseguido con esto, es que, una vez conocidos los aspectos de programación a bajo nivel de Rails, el alumno esté preparado para afrontar y entender todas las posibilidades que Ruby on Rails ofrece, y que, una vez concluido el proyecto, cuándo haga uso de las herramientas *scaffolding* que Rails ofrece, sepa en todo momento lo que está haciendo y cómo se está haciendo.

Durante las más de 1.000 horas de trabajo se han desarrollado unas 9.000 líneas de código organizadas en más de 200 ficheros. Entre estas líneas de código, podemos diferenciar entre más de 4.500 líneas de código de aplicación y más de 4.000 líneas de código para pruebas, desarrolladas a lo largo de 2.500 pasos.

Antes de entrar en resultados de testeo de la aplicación, vamos a nombrar resultados a nivel de funcionalidad de la aplicación, desde un punto de vista de utilización del portal web. A continuación veremos los resultados de las pruebas realizadas en la aplicación.

11.1.1- Portal web

A pesar de ser un prototipo de aplicación, nuestro proyecto ha conseguido ser un sistema totalmente operativo (a nivel de prototipo) e integrado en la nube de internet. Es posible el acceso a él mediante el enlace http://doggyhouse.herokuapp.com/. Si se navega por el sistema, se observa que está totalmente operativo, que no terminado. Un usuario cualquier

es capaz de registrarse, de navegar por el sitio, de añadir nuevos perros, administrar su perfil y sus perros, enrolarse en exhibiciones que haya en el sistema, confirmar el pago por estas inscripciones mediante la subida de un recibo bancario… En definitiva capacidades básicas que el producto final debería tener y que se ha conseguido, en muy buena medida, con este prototipo. Hay muchos aspectos de mejoras y añadidos que el prototipo puede tener, pero estas mejoras las estudiaremos en el apartado 12.

También hay que mencionar, que al tener la aplicación abierta de cara al usuario y en un portal accesible por cualquier persona, se le han añadido aspectos visuales notorios mediante la utilización de *Bootstrap*. De este modo el portal web ha quedado amigablemente visible, y su utilización es bastante adecuada y cercana al sistema final que se desearía conseguir.

Es obvio, que los perfiles de usuarios están totalmente implementados, y que, en el sistema actual, se puede ingresar cómo un visitante, un usuario registrado, o en forma de administrador (con la cuenta creada para ello).

De este modo, hemos conseguido tener un portal web operativo, con buena funcionalidad para su fin y que se asemeja bastante a lo que se querría conseguir con la implantación de un sistema final totalmente acabado.

11.1.2- Pruebas

Cómo hemos mencionado, se han implementado muchas pruebas (tanto de aceptación, como unitarias) que comprueban el correcto funcionamiento de muchísimas características del sistema. De este modo, si el proyecto se desease ampliar en un futuro, y partir de lo que ya se tiene, se podría estar seguro que la base del sistema está correcta, y que si hay algún cambio negativo o de error en esta base, nos mostrará los errores correspondientes a la hora de correr las pruebas.

Se han utilizado varias herramientas para realizar las pruebas de la aplicación. En esta sección vamos a ver algunas estadísticas y datos reales de estas pruebas. Para comenzar, expondremos una imagen en la que mostramos el número total de tests realizados con Cucumber, los escenarios pasados y los *steps* totales que se han definido:

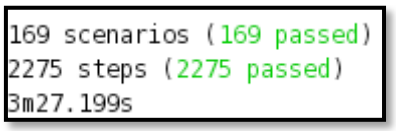
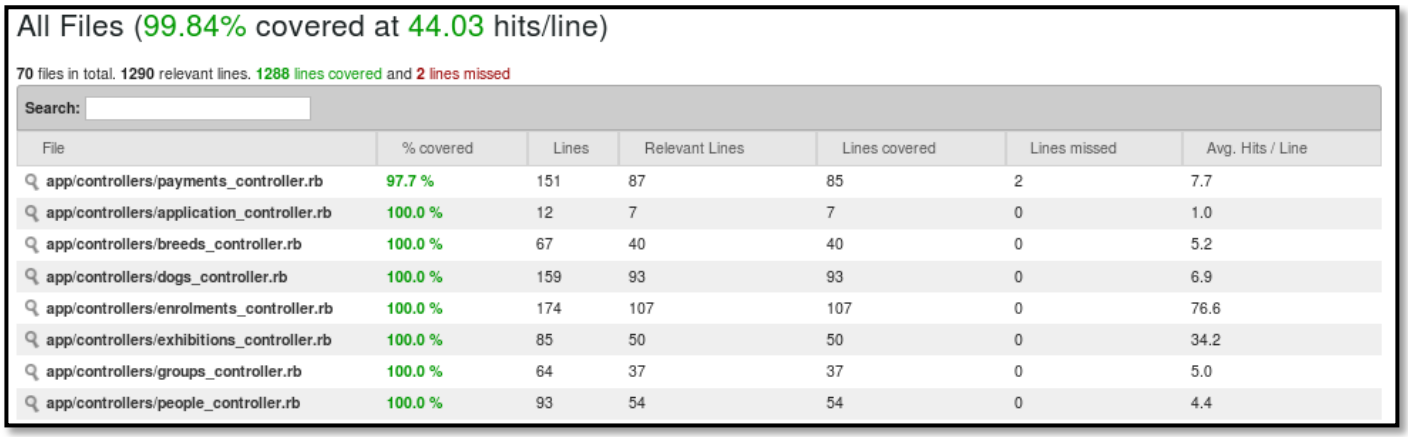


Imagen 11.1.1 – Estadísticas Cucumber

Como podemos observar, se han desarrollado un total de 169 escenarios, los cuáles divididos en 2.275 *steps*. Estos a su vez han tenido que ser implementados, por lo que la tarea de conseguir el porcentaje de cobertura de código que hemos conseguido, no ha sido nada fácil. El último número que muestra la imagen es simplemente el tiempo que ha tardado en realizarse toda la batería de pruebas.

A continuación veremos las estadísticas que nos proporciona la gema *simplecov*. Esta gema, nos indica el porcentaje de código cubierto mediante los tests realizados con Cucumber,

es decir, mediante los tests de aceptación. Evidentemente no cubre absolutamente todo el código, pero podemos ver una muy buena referencia de lo que se ha testeado. En la imagen de a continuación veremos los resultados proporcionados por esta gema:



All Files (99.84% covered at 44.03 hits/line)

70 files in total. 1290 relevant lines. 1288 lines covered and 2 lines missed

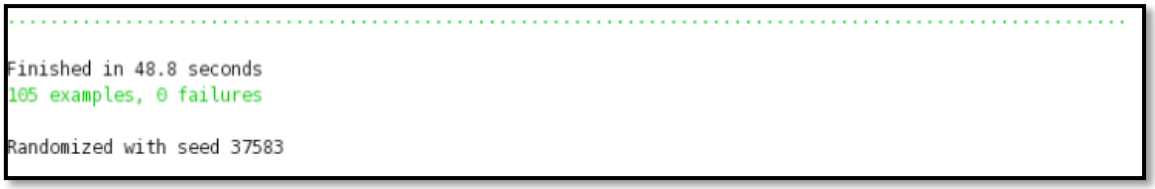
Search:

File	% covered	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
app/controllers/payments_controller.rb	97.7 %	151	87	85	2	7.7
app/controllers/application_controller.rb	100.0 %	12	7	7	0	1.0
app/controllers/breeds_controller.rb	100.0 %	67	40	40	0	5.2
app/controllers/dogs_controller.rb	100.0 %	159	93	93	0	6.9
app/controllers/enrolments_controller.rb	100.0 %	174	107	107	0	76.6
app/controllers/exhibitions_controller.rb	100.0 %	85	50	50	0	34.2
app/controllers/groups_controller.rb	100.0 %	64	37	37	0	5.0
app/controllers/people_controller.rb	100.0 %	93	54	54	0	4.4

Imagen 11.1.2 – Code Coverage

Lo que se busca conseguir con los tests, es cubrir el 100% del código de la aplicación; cómo observamos en la imagen anterior, el porcentaje de código cubierto mediante Cucumber es del 99,84%, es decir, casi la totalidad del código que la gema es capaz de comprobar. No hay que decir que esto es algo muy bueno, y que realmente, podemos considerar que está la totalidad del código cubierto. Es por esto, que para un trabajo futuro tendremos la certeza de que se va a partir de una base testeada, y que si algo se cambia y empieza a fallar, sabemos dónde encontrar el fallo.

El último aspecto que veremos, acerca de las pruebas, será el relacionado con las pruebas realizadas con RSpec. En la imagen siguiente mostramos las estadísticas que nos proporciona RSpec cuándo corremos los tests implementados con esta herramienta.



```
.....................................................................................................

Finished in 48.8 seconds
105 examples, 0 failures

Randomized with seed 37583
```

Imagen 11.1.3 – Estadísticas Rspec

Como vemos, se han ejecutado un total de 105 *steps* y no ha habido ningún fallo; todos los tests han resultado exitosos, aspecto que es deseable.

11.2- Conclusiones

Como conclusión final del proyecto, debemos decir que todas las horas invertidas en el desarrollo del mismo han sido totalmente provechosas. Consideramos que gracias a todos los estudios realizados, el material consultado, los libros leídos, el trabajo desarrollado, y demás, se ha adquirido un nivel importante en la materia, tanto en programación Ruby on Rails cómo en metodologías de desarrollo basadas en tests.

Al ser un área desconocida para nosotros nos ha conducido a tener que dedicar más tiempo de estudio de herramientas y disciplinas de desarrollo, consultando mucha documentación y llevándonos a entender, para nosotros, un nuevo modo de desarrollar aplicaciones. Sin duda es un punto muy positivo a tener en cuenta en el ámbito de un proyecto de final de carrera, ya que, realmente, todo este tiempo invertido nos va a proporcionar una mejoría y especialización en ciertas materias vistas durante la carrera, y, en nuestro caso particular, el aprendizaje de una metodología y entorno de desarrollo actualmente muy solicitados en el mundo laboral.

En cuanto al resultado final de la aplicación, hemos conseguido un prototipo de aplicación muy interesante, y que se mantiene abierta a muchas posibilidades. Son muchas las ampliaciones que se pueden realizar con esta base de la aplicación, y las oportunidades que ofrece son bastante amplias. En el siguiente apartado veremos aspectos de mejoras y ampliaciones que podemos conseguir utilizando como base nuestro proyecto. Como hemos nombrado, gracias a los tests de la aplicación que hemos ido realizado durante todo el desarrollo, tenemos la certeza de que un trabajo futuro, utilizando como base este mismo proyecto, no se va a permitir hacer modificaciones que dejen nuestra aplicación inservible, ya que cualquier cambio que produzca resultados erróneos en la aplicación será detectado y advertido mediante estos tests.

Por otro lado, el prototipo está totalmente operativo y, bajo la idea de que es un prototipo y no una aplicación comercial, el trabajo se puede dar por finalizado para este proyecto. El prototipo resultante del desarrollo del proyecto ha sido bastante deseable y cercano a una parte de un producto final que al cliente le gustaría tener. Teniendo en cuenta que hemos tenido que formarnos básicamente desde cero en muchos aspectos de la materia, hay que considerar que, aunque es cierto que las aspiraciones iniciales del proyecto eran algo distintas y se pretendía conseguir una aplicación totalmente acabada y operativa, el resultado obtenido es muy satisfactorio. No hay que guiarnos por la idea de buscar una aplicación comerciable para un proyecto de final de carrera, sino la de conseguir una formación final para el alumno que le impulse a iniciarse en el mundo laboral de la informática, y, en estos términos, con este proyecto se ha logrado. Si realmente se buscase obtener una aplicación comerciable, tal vez la podríamos haber conseguido durante el tiempo de trabajo que ha llevado el proyecto: podríamos haber utilizado herramientas para mejorar la productividad en los tests, podríamos haber no seguido a rajatabla la metodología de desarrollo, utilizar muchas más gemas y *scaffolds* para la creación de modelos y controladores… aspectos que hubieran ayudado a conseguir una aplicación comerciable, pero que no hubieran ayudado en cuanto a la formación del alumno. Guiados por los pasos que hemos seguido en el desarrollo, lo que hemos conseguido es entender, desde la parte más baja del mundo Rails, en lo que consiste empezar a elaborar una aplicación comerciable y que, realmente, hay todo un mundo por debajo de un mero *scaffold* o de una prueba de aceptación. En un futuro habrá muchas más ocasiones para empezar las cosas desde un punto no tan a bajo nivel como en el que se hecho en el proyecto y de esta forma conseguir mayores resultados a nivel comercial (eso sí, conoceremos y sabremos de dónde provienen todos los “atajos” que usemos).

12- Trabajo futuro

El proyecto está abierto a una gran cantidad de ampliaciones y mejoras ya que, al ser un prototipo de aplicación, nuestro proyecto da posibilidades de grandes ampliaciones y de futuros proyectos en base a este. En este apartado vamos a nombrar cuales son estas ampliaciones y las mejoras que debería tener el prototipo para poder empezarse a considerar una aplicación comercial. Empezaremos por estas, las mejoras necesarias al prototipo para conseguir una aplicación comercial. A continuación expondremos una serie de ampliaciones que pueden considerarse en base a este sistema y que se podrían acoplar a nuestra aplicación una vez esté acabada.

12.1- Mejoras

En las dos siguientes secciones veremos las mejoras que nuestro prototipo debería tener para conseguir adaptarlo a ser una aplicación comerciable. Primero veremos las mejoras necesarias de cara al usuario y a continuación las generales de cara al sistema.

12.1.1- Usuarios

En referente a los usuarios al sistema, habría que considerar varias posibilidades antes de comercializar la aplicación. Algunas de los aspectos que tenemos que considerar son los siguientes:

- **Registro:** A la hora de registrarse en el sitio, actualmente sólo permite un registro rellenando los datos e-mail e introduciendo una contraseña. Los datos restantes de un perfil de usuario (nombre, fecha de nacimiento, ciudad…) no son accesibles a la hora de registrarse en el portal, una vez el usuario entra en su perfil es cuándo puede personalizar estos datos. Sería interesante modificar esto para que, de entrada, el usuario pudiese introducir sus datos personales y quedasen guardados en el sistema a la hora de registrarse.
- **Logueo:** La manera de hacer log-in en el sistema es mediante la introducción de su e-mail. Una posibilidad útil para el usuario es ampliar esta capacidad y que no solamente se pudiese loguear en el sistema con el e-mail, sino también con su nombre propio o un posible “nombre de usuario”.
- **Gravatar:** Gravatar (*Globally Recognized Avatar*), es un servicio que ofrece un avatar único globalmente. En Gravatar los usuarios pueden registrar una cuenta basada en su correo electrónico, y subir un avatar para que sea asociado a la cuenta. La idea sería utilizar este servicio, e integrarlo en la aplicación final. De este modo conseguiremos un sistema más amigable para el cliente, ampliando de este modo la profesionalidad.
- **Cookies:** Las cookies son muy útiles en cualquier sistema. Cuando iniciamos sesión en cualquier servicio web y cuando, por ejemplo, cerramos esta página y al cabo de un tiempo la volvemos a abrir, percibiremos que generalmente la sesión continúa iniciada. Esto es porque el servicio web ha leído información de la cookie que se guardó la primera vez que introducimos los credenciales. En nuestro portal

web también ocurre esto, sin embargo hay una Ley de cookies que indica que, dependiendo las cookies a utilizar, algunas requieren autorización del uso por parte del usuario. En nuestro sistema actual no están contempladas estas opciones, por lo que habría que añadir tareas de confirmación de cookies por parte del usuario, así cómo aviso legal y demás características derivadas.

- **Mejoras de amigabilidad:** El prototipo actual también es susceptible a mejorar la amigabilidad en ciertos aspectos de diálogo con el usuario. Por ejemplo, a la hora de ver las exposiciones en las que un usuario está inscrito, mostrarlas en un desplegable tipo acordeón en lugar de en una tabla, la posibilidad de añadir la foto del perro de un usuario, un sistema de navegabilidad por el site más avanzado, añadir documentos *pdf* con las características estándares de cada raza… Es decir, ciertos aspectos que pueden hacer la aplicación más competitiva.
- **Búsqueda de perros/exhibiciones/usuarios:** No hay implementado ningún sistema de búsqueda en nuestro prototipo. En la aplicación final sería deseable que el usuario tuviera la posibilidad de buscar perros, exhibiciones, usuarios… y que de este modo se pueda ver información de quién está registrado en el sistema y con qué perros. Con esto se crearía cierta competitividad entre los propios usuarios y motivaría el uso del portal.

12.1.2- Sistema

A parte de opciones interesantes y necesarias de cara al usuario, hay otras capacidades que deben ser tenidas en cuenta desde el punto de vista de funcionalidad del sistema, y de las cuales nuestro prototipo actual carece. Estas características que se deberían implementar para llegar a ser un sistema comercial serían las siguientes:

- **Internacionalización (i18n):** Se conoce como la internacionalización de una aplicación (cuyo acrónimo es i18n) al proceso de diseñar el software de manera que pueda adaptarse a diferentes idiomas y regiones. El sistema actual está íntegramente en el idioma inglés, por lo que un aspecto a tener en cuenta de cara a la comercialización del mismo, es la internacionalización de éste.
- **Reasignación de razas:** A la hora del borrado de las secciones y sub-secciones, de la manera que está actualmente implementado el sistema, hay un borrado en cascada de razas, variedades y sub-variedades. Una mejora del sistema actual sería el no realizar un borrado en cascada, sino permitir la reasignación de estas supuestas razas de perros “fantasmas” que se generarían al borrar una de las secciones o sub-secciones. De este modo, mediante alguna interfaz gráfica, si existiesen en el sistema razas que no están asignadas a alguna sección o sub-sección, se diera la posibilidad al administrador para reasignarlas a alguna otra.
- **Correos masivos:** Añadir al sistema capacidad de envío de correos masivos es algo casi fundamental en una aplicación comerciable de este tipo. Es muy útil, por ejemplo, posibilitar la opción de que al cambiar precios en una exhibición se le avise a todos los clientes que están inscritos en ella, o tal vez cuándo se fuese acercando las fechas de las *deadlines* de pago. Otra utilidad de esta integración podría ser el aviso a clientes que se consideren potencialmente interesados en una nueva exhibición añadida al sistema (por ejemplo, por proximidad, si el cliente vive

en el país en el que se va a celebrar la exhibición, mandarle un email de información).

- **Interfaz para añadir precios de exhibiciones:** Hemos nombrado que la manera actual de ingresar las tablas de precios de las exposiciones caninas es mediante un hash en formato JSON que se debe insertar mediante consola. Una aplicación final y comerciable, debería proveer una interfaz gráfica que permitiese esta inserción de precios de una forma más amigable. De esta forma, también habría que terminar la implementación de las distintas clases de perros que se pueden incluir en una exhibición. Vimos en el apartado de análisis de la aplicación que las clases *Couple*, *Working* y *Group Breeding* no están actualmente implementadas en el sistema, por lo que a la hora de implementar la interfaz para añadir los precios de las exhibiciones, también habría que considerar integrar la definición y la comprobación de estos tres tipos de clase en el sistema.
- **Localización geográfica:** Sería interesante tener en un sistema final la posibilidad de incluir geolocalización a la hora de definir exhibiciones. Hay varias gemas que permiten incluir esta característica.
- **CanCan:** Actualmente, el sistema de autorización para el acceso a determinadas secciones de la web (secciones de administración, secciones de usuarios registrados, perfiles de usuario…) está implementado de forma manual, utilizando una solución personal para restringir los accesos dependiendo del tipo de usuario que esté utilizando el sistema. Esto puede resultar (aunque efectivo) algo engorroso a la hora de leer el código por futuros desarrolladores, por lo que no se puede dejar de proponer como mejora al sistema la utilización de la gema "**CanCan**", la cual nos ofrece una solución amigable y limpia para aspectos relacionados con restricciones de accesos a los diferentes sitios del portal.
- **Paginación de tablas:** Nuestro sistema no ofrece una paginación de tablas para una vista amigable de las mismas, es decir, que la tabla se muestra como un bloque entero en una sola página. La gema **Kaminari** sería la gema indicada para permitirnos paginar las tablas. En vista a una aplicación comerciable, habría que incluir la utilización de esta gema en el desarrollo.
- **CRUD de usuarios**: Incluir capacidades para crear, leer, actualizar y borrar usuarios, no solo perfiles o "personas".
- **Nuevos actores**: La base para la inclusión de nuevos actores en el sistema está implementada (esto es el modelo "*person*"). En un sistema final debería haber nuevos actores, como por ejemplo organizadores de exhibiciones, responsables de cobros, personal de contacto… Actores que, sin ser directamente usuarios del sistema, deberían tener un perfil de contacto en la aplicación y que serían administrados por el propio administrador.
- **Métodos de pago:** El método de pago implementado actualmente es algo rudimentario para los tiempos que corren. Para nuestra causa funciona, pero para un sistema comercial se deberían añadir métodos de pago mediante Paypal, tarjetas bancarias, mediante móvil… Aspecto que nos lleva a nuestra siguiente característica de mejora:
- **Seguridad:** La seguridad actual del sistema es bastante pobre; la inexperiencia en este aspecto seguramente nos haya ocasionado dejar bastantes puertas abiertas a

muchos tipos de hackeos de sistema. Es por ello que, para una aplicación comerciable, habría que revisar y dejar cerrados aspectos fundamentales en la seguridad de aplicaciones web. Controlamos ciertas características relacionadas con la seguridad, cómo la de no permitir acceder a secciones que no le corresponden al usuario o el crear/borrar/actualizar información privilegiada del sistema sin ser administrador, pero aspectos a rasgos mayores como los temas relacionados con hackeos no es algo que esté controlado actualmente.

- **Relaciones reflexivas para perros:** De forma que se puedan relacionar perros entre sí y poder crear crear una estructura de padres e hijos de perros. De este modo un perro podrá tener descendencia (o ascendencia) genealógica.
- **Mejorar aspectos visuales:** Aunque en términos generales estamos bastante contentos con la apariencia visual mostrada, por ejemplo, el entorno visual no lo tenemos optimizado para móviles. Por ello, otros aspectos de mejora de cara a una aplicación comerciable serían el mejorar el diseño visual, o el integrar características concretas para poder correr la aplicación desde dispositivos móviles.
- **Jasmine:** Como hemos visto a lo largo del documento, una de las características que tiene nuestro sistema es la buena cobertura de testeo que tiene. Sin embargo, aspectos relacionados con JavaScript no están totalmente cubiertos. Para ello, se recomienda la adición de la herramienta **Jasmine**. Jasmine es un framework de testeo para JavaScript que funciona de manera muy similar a RSpec con funciones útiles para organizar los tests como "*describe*" e "*it*". De este modo conseguiríamos una cobertura mayor en cuanto al testeo de la aplicación.

12.2- Ampliaciones

Hemos nombrado en alguna ocasión durante la memoria que nuestro sistema es una pequeña parte de algo mucho más grande que sería muy interesante desarrollar. La aplicación totalmente montada que se puede llegar a conseguir mediante asociaciones de proyectos es un sistema capaz de englobar todas las peculiaridades del mundo de las exhibiciones caninas, es decir, un conjunto de aplicaciones integradas y que funcionen conjuntamente. Estas aplicaciones pueden ser numerosas, ya que el campo es amplio, y la capacidad de negocio que aporta, también.

De este modo, con nuestro proyecto hemos conseguido un prototipo de aplicación para la gestión de exposiciones caninas. Lo que hemos logrado es posibilitar a usuarios que inscriban a perros en exposiciones reales, en base a ciertas normas y precios, y que demuestren el abono de las inscripciones mediante la subida del recibo bancario al portal. A esto se le ha añadido cierta información cinológica general respecto a la organización de las razas según la FCI. A este sistema, una vez lo hayamos modificado para lograr una aplicación comerciable en base a lo que hemos expuesto en la sección anterior de mejoras, se le pueden añadir otros sistema que podrían dar fruto a otros trabajos de final de carrera. Por ejemplo, uno de estos sistemas podría ser un sistema de información avanzado acerca de la cinología, es decir que tenga una estructura de grupos, secciones, razas… más avanzado que el que tiene el nuestro: mucha más información respecto a ello, estándares de cada raza en formato *pdf*,

imágenes de cada una de las razas… Una gran cantidad de información complementaria que empezaría a acrecentar nuestro sistema base.

Otra ampliación o proyecto futuro que se podría integrar en nuestro sistema, es un sistema de pedigrís como el que tiene Ingrus (estudiado en la sección 4.1.2). Si bien es cierto que el sistema de Ingrus está muy bien construido, es muy bueno y muy extenso, algo que imite a esta gran base de datos podría ser un buen proyecto futuro y que también podría ser candidato para integrar en nuestro sistema. La base para empezarlo ya la tenemos con nuestro prototipo.

Un sistema capaz de guardar los resultados y otros datos que se generan en cada una de las exhibiciones que se celebran, sería otra aplicación para considerar en un proyecto de final de carrera y que, también, podría ser integrado en la base del sistema que hemos desarrollado en este proyecto. En cada celebración de exhibiciones caninas hay una gran cantidad de datos que resultan de ellas y que se generan para ellas: número de perros participantes, ranking de perros, personal participante, ventas de entradas, asociaciones responsables… La integración de otra aplicación que recogiese y organizase todo esto, añadiría aún más características a nuestro sistema para obtener un sistema final muy potente, en el que cualquier información con respecto a todo esto quedase recogida y organizada. Y esto es lo que se querría conseguir con nuestro portal, con **DoggyHouse**, un portal de consulta y de referencia mundial para las exhibiciones caninas, en el que cualquier información con respecto a ellas pudiera ser accedida desde él.

Bibliografía

[ING10] Roger S. Pressman. 2010. "*Ingeniería del Software. Un enfoque práctico*". Séptima edición. McGraw-Hill.

[LAR03] Larman, Craig. 2003. "*UML Y PATRONES. Una introducción al análisis y diseño orientado a objetos y al proceso unificado*". Segunda edición. Prentice Hall.

[ELL00] James Rumbaugh; Ivar Jacobson; Grady Booch. 2000. "*El lenguaje unificado de modelado. Manual de referencia*". Addison Wesley.

[ING95] Amescua Seco, Antonio. 1995. "*Ingeniería del software de gestión: Análisis y diseño de aplicaciones*". Primera edición. Paraninfo.

[DIS10] Blé Jurado, Carlos. 2010. "Diseño Ágil con TDD". Primera edición. Edición digital

[AGI13] Sam Ruby; Dave Thomas; David Heinemeir Hansson. 2013. "*Agile Web Development with Rails 4*". Primera edición. The Pragmatic Programmers.

[RAI08] Derel DeVries; Mike Naberezny. 2008. "*Rail for PHP Developers*". Primera edición. The Pragmatic Bookself.

[LEA13] Hafiz; Nia Mutiara; Giovanni Sakti. 2013. "*Learning Devise for Rails*". Primera edición. Packt Publishing.

[THE13] Fernandez, Obed; Faustino, Kevin. 2014. "*The Rails 4 Way*". Primera edición. Addison-Wesley.

[REF99] Martin Fowler. 1999. "*Refactoring: Improving the Design of Existing Code*". Addison-Wesley.

[RUB10] Hartl, Michael. 2010. "*Ruby on Rails 3 Tutorial: Learn Rails by Example*". Formato digital (www.railstutorial.org). Último año de consulta: 2014.

Anexos

Anexo I – Detalle de casos de uso

Visitante

Nombre	Consultar Documentación	**ID**	01
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por	Raúl José Alcañiz de la Fuente	**Fecha Modif.**	10-01-2015
Actor Principal			
• **Visitante**			
Personal Involucrado o Intereses			
• Visitante: El usuario busca poder acceder a información general del sistema.			
Descripción			
El usuario puede navegar por el sitio, visitando las distintas secciones de información general cinológica que hay en el portal.			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. Navegar por el sitio web.			
Flujo Alternativo			
Excepción			
1.* La búsqueda de la información falla - Se le indica al usuario que en estos momentos no es posible realizar la consulta deseada.			
Includes			
Requisitos Especiales			
Notas			
En este caso de uso, se incluirían los casos de uso número **35, 39, 43, 47, 51 y 55** correspondiente al visionado de los **grupos**, **secciones**, **sub-secciones**, **sub-variedades**, **razas** y **variedades** respectivamente. No obstante, por las características **CRUD** de estos elementos del sistema, se ha decidido definir independientemente, y en la sección del administrador de este mismo anexo, los casos de uso referentes al visionado de estos elementos.			

Nombre	Consultar Exposición	**ID**	02
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por	Raúl José Alcañiz de la Fuente	**Fecha Modif.**	10-01-2015
Actor Principal			
• **Visitante**			
Personal Involucrado o Intereses			
• Visitante: El usuario busca poder acceder a información referente a exposiciones.			
Descripción			
Se presenta al usuario la información de las exposiciones que hay en el sistema			
Trigger			
Hacer clic en **Exhibitions**.			

Precondición
Postcondición
Flujo Normal
1. El usuario accede a la página principal de las exhibiciones. 2. Se le muestra información general de las exhibiciones que hay en el sistema.
Flujo Alternativo
2.* No hay exhibiciones en el sistema - Se muestra el mensaje correspondiente de que no existen actualmente exhibiciones en el sistema.
Excepción
Includes
Requisitos Especiales
Notas

Nombre	Darse de alta	**ID**	03
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por	Raúl José Alcañiz de la Fuente	**Fecha Modif.**	21-01-2015
Actor Principal			
• **Visitante**			
Personal Involucrado o Intereses			
• Visitante: El usuario pretende introducir sus datos en el sistema para quedar registrado en el mismo y poder acceder a más características.			
Descripción			
El usuario no registrado tiene la posibilidad de rellenar un formulario de registro; dicho registro será confirmado por e-mail y el usuario quedará incluido en el sistema. Éste formulario constará de datos personales, nombre de usuario, contraseña, correo electrónico, etc.			
Trigger			
Hacer clic en el botón de **Sign Up**, en el panel superior.			
Precondición			
Postcondición			
• Se envía un e-mail de confirmación al correo electrónico del usuario.			
Flujo Normal			
1. El usuario entra en la sección de registro. 2. El consumidor rellena el formulario de registro. 3. El usuario valida los datos introducidos. 4. El sistema corrobora los datos. 5. Se envía un email de confirmación al e-mail proporcionado.			
Flujo Alternativo			
4.a Algún dato introducido por el usuario es incorrecto (dirección de correo repetida, contraseña inválida…). 4.a.1. Se informa al cliente del dato/s inválido/s y se le da la opción que lo/s ingrese de nuevo, volviendo al paso 2.			
Excepción			
Includes			
Se utiliza el caso de uso con ID **05 – Validar registro**.			
Requisitos Especiales			
Notas			

Nombre	Mostrar detalles de exposición	**ID**	04
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por	Raúl José Alcañiz de la Fuente	**Fecha Modif.**	21-01-2014
Actor Principal			
• **Visitante**			
Personal Involucrado o Intereses			
• Visitante: El usuario quiere conocer más a fondo algunos datos de interés de una determinada exposición ubicada en el site.			
Descripción			
El consumidor pretende informarse sobre alguna exposición que tenemos alojada en el portal web, ya sea un evento con el plazo de inscripción abierto, o un evento ya finalizado.			
Trigger			
Hacer clic en la exposición que desea obtener la información.			
Precondición			
• Haber entrado en el menú de exposiciones.			
Postcondición			
• El usuario obtiene la información que existe acerca de la exposición seleccionada.			
Flujo Normal			
1. El usuario hace clic en la exposición de la cuál desea obtener información. 2. Se muestra la información que haya disponible acerca del evento seleccionado.			
Flujo Alternativo			
Excepción			
2.* La información del evento no está disponible. - Se informa al usuario que la consulta realizada no está disponible en estos momentos.			
Includes			
Requisitos Especiales			
Notas			

Nombre	Validar registro	**ID**	05
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por	Raúl José Alcañiz de la Fuente	**Fecha Modif.**	10-01-2015
Actor Principal			
• **Visitante**			
Personal Involucrado o Intereses			
• Visitante: Se pretende comprobar que el e-mail introducido es correcto y está operativo.			
Descripción			
Una vez relleno y aceptado el formulario de registro, el sistema manda un e-mail al correo electrónico del cliente. Dicho correo contendrá un enlace, que debe cliquear el cliente, y que sirve tanto para confirmar la dirección de correo, como para completar el registro en el portal web.			
Trigger			
El evento ocurre cuando el usuario acepta y se da por correcto el formulario de registro.			
Precondición			
• Haber rellenado el formulario de registro correctamente y pulsar el botón de **Aceptar registro**.			
Postcondición			

• El usuario queda registrado en el sistema.
Flujo Normal
1. El usuario recibe un e-mail en su correo. 2. Abre el e-mail recibido y cliquea en el enlace de confirmación. 3. El sistema valida el registro de usuario.
Flujo Alternativo
2.a El usuario no acepta/confirma el e-mail de confirmación. 2.a.1. Se desecha la petición de registro tras 24h.
Excepción
1.* El e-mail de confirmación de registro no le llega al cliente. - Se requiere clickear en “Didn’t receive confirmation instructions”. - Se indica el email y se reenvía la información. 3.* El sistema no puede validar el registro. - Se indica al usuario que no ha sido posible validar el registro.
Includes
Notas

Usuario registrado

Nombre	Añadir can	**ID**	06
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Usuario Registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: Quiere añadir un can a su perfil de usuario, quedando registrado como propietario del mismo.			
Descripción			
Opción que permite al cliente añadir un can a su lista de ejemplares. De esta forma queda reflejado que es propietario del mismo.			
Trigger			
Hacer clic en el botón **Add a dog**.			
Precondición			
• Encontrarse en el perfil del usuario.			
Postcondición			
• El animal queda inscrito en el registro del sistema como propiedad del cliente.			
Flujo Normal			
1. Se cliquea en el botón de **Add a dog**. 2. Se rellena el formulario. 3. Se confirman los datos introducidos. 4. El sistema almacena los datos del ejemplar vinculando éste al usuario.			
Flujo Alternativo			
4.a Identificador del perro ya existente. 4.a.1. Se le indica al usuario el mensaje de error correspondiente y se vuelve al paso 2.			
Excepción			
Includes			
Requisitos Especiales			
Notas			

Nombre	Borrar inscripción	**ID**	07
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	21-01-2014
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Usuario Registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: Se desea eliminar algún ejemplar de alguna exposición en la que está inscrito.			
Descripción			
El cliente va a eliminar a uno de sus ejemplares de una competición en la que ha sido enrolado.			
Trigger			
Hacer clic en **Delete enrolment**.			
Precondición			
• Estar en la sección *My enrolments* para una exhibición. • Tener al can inscrito en una competición. • No haber abonado el importe de la inscripción.			
Postcondición			
• El ejemplar queda borrado de la lista de canes inscritos en la competición.			
Flujo Normal			
1. Se cliquea en el botón de **Delete enrolment**. 2. Se confirma el borrado. 3. El sistema elimina el vínculo existente entre el ejemplar actual y la competición seleccionada.			
Flujo Alternativo			
Excepción			
3.* No se puede eliminar el ejemplar de la competición. - El sistema avisa al cliente y, en el caso que proceda, se le comunica que se ponga en contacto con el administrador del sistema o con el organizador del evento para llevar a cabo la acción requerida.			
Includes			
Requisitos Especiales			
Notas			

Nombre	Cancelar cuenta	**ID**	08
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	10-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Usuario Registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: Quiere darse de baja en el sistema.			
Descripción			
Opción que permite al cliente eliminar su cuenta del sistema. Una vez terminada la operación, la cuenta quedará totalmente borrada del sistema.			
Trigger			
Hacer clic en el botón **Cancel account**.			
Precondición			
• Encontrarse en el perfil del usuario.			
Postcondición			

• La cuenta queda eliminada del sistema.
Flujo Normal
1. Se cliquea en el botón de **Cancel account**. 2. Se confirma la acción. 3. El sistema actualiza los datos, quedando el usuario eliminado del mismo.
Flujo Alternativo
3.a No se confirma la acción. 3.a.1. El sistema no varía respecto a cómo estaba antes de empezar la operación.
Excepción
4.* No se puede eliminar la cuenta del sistema. - El sistema avisa al cliente y se le comunica que se ponga en contacto con el administrador del sistema para informar sobre el error ocurrido. - No se varía ninguna información en el sistema.
Includes
Requisitos Especiales
Notas

Nombre	Editar perfil	**ID**	09
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Usuario Registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: Quiere cambiar información personal almacenada en el sistema y/o borrarse del mismo.			
Descripción			
Se cambia información personal del usuario, almacenada en el sistema, a interés de dicho usuario.			
Trigger			
Hacer clic en **Edit profile**.			
Precondición			
• Estar en el menú del perfil del usuario.			
Postcondición			
• El perfil del cliente queda modificado a sus intereses.			
Flujo Normal			
1. Se cliquea en el botón de **Edit profile**. 2. El cliente cambia los datos que desee. 3. Se confirman los cambios realizados. 4. El sistema acepta los cambios y actualiza la información en el mismo.			
Flujo Alternativo			
4.a El sistema no acepta los cambios. 4.a.1. Hay datos erróneos en las modificaciones (contraseña incorrecta, e-mail inválido, datos erróneos…). Se le notifica al usuario y se vuelve al paso 2 sin realizar modificaciones en los datos del usuario.			
Excepción			
Includes			
Requisitos Especiales			
Notas			

Nombre	Eliminar can	**ID**	10
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Usuario Registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: Quiere eliminar del sistema algún can vinculado a su cuenta de usuario.			
Descripción			
Opción que permite al cliente eliminar un can de su lista de ejemplares. De esta forma se elimina (y se desvincula) el animal del cliente.			
Trigger			
Hacer clic en el botón **Delete dog**.			
Precondición			
• Tener algún can vinculado al usuario. • Estar visionando la ficha del ejemplar a eliminar en cuestión.			
Postcondición			
• El animal queda desvinculado del cliente y eliminado del registro del sistema.			
Flujo Normal			
1. Se cliquea en el botón de **Eliminar can**. 2. Se confirma la acción. 3. El sistema actualiza los datos, quedando dicho ejemplar eliminado del mismo.			
Flujo Alternativo			
Excepción			
Includes			
Requisitos Especiales			
Notas			

Nombre	Inscribir can en exposición	**ID**	11
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por	Raúl José Alcañiz de la Fuente	***Fecha Modif.***	10-01-2015
Actor Principal			
• **Usuario Registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: Desea inscribir a uno o varios ejemplares a una competición			
Descripción			
Se le permite al cliente añadir a un can a una competición que tiene el plazo de inscripciones abiertas.			
Trigger			
Hacer clic en el botón **Enroll a new dog**.			
Precondición			
• Encontrarse en la página principal de la exhibición a la que va a enrolarse el perro			
Postcondición			
• Se crea un vínculo temporal (a espera de la confirmación del pago de las tasas) entre el ejemplar y la competición seleccionados.			
Flujo Normal			
1. Se cliquea en el botón **Enroll a new dog**. 2. Se selecciona el can a inscribir. 3. Se selecciona la clase a la que pertenece el perro.			

4. Confirmar los datos seleccionados. 5. El sistema comprueba los datos y muestra el precio. 6. Se acepta el precio mostrado por el sistema. 7. El enrolment queda registrado en el sistema a espera de la subida del pago.
Flujo Alternativo
Excepción
5.* El sistema no permite la inscripción. - Por algún motivo no se permite la operación y el sistema notifica al usuario del error correspondiente, y dándole la opción a corregirlo. - No se varía ninguna información en el sistema. 6.* No se acepta el precio por la inscripción. - No se varía ninguna información en el sistema.
Includes
Requisitos Especiales
Notas

Nombre	Modificar can	**ID**	12
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	10-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Usuario Registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: Quiere cambiar información respecto a algún perro registrado en su cuenta.			
Descripción			
Se cambia información referente a algún perro del usuario, almacenada en el sistema.			
Trigger			
Hacer clic en **Edit dog**.			
Precondición			
• Tener algún can vinculado al usuario. • Estar visionando la ficha del ejemplar a editar en cuestión.			
Postcondición			
• El dato del perro queda modificado a interés del usuario.			
Flujo Normal			
1. Se cliquea en el botón de **Edit dog**. 2. El cliente cambia los datos que desee. 3. Se confirman los cambios realizados. 4. El sistema acepta los cambios y actualiza la información en el mismo.			
Flujo Alternativo			
4.a El sistema no acepta los cambios. 4.a.1. Hay datos erróneos en las modificaciones. Se le notifica al usuario y se vuelve al paso 2 sin realizar modificaciones en los datos del perro.			
Excepción			
Includes			
Requisitos Especiales			
Notas			

Nombre	Modificar pagos	**ID**	13
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	10-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Usuario Registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: Quiere cambiar información referente a algún pago que ha realizado			
Descripción			
Se cambia información referente a algún pago realizado por el usuario.			
Trigger			
Hacer clic en **Edit payment**.			
Precondición			
• Tener algún pago realizado. • Estar en la página principal de pagos. • Que el pago no haya sido aceptado por el administrador del sistema.			
Postcondición			
• La información nueva del pago queda almacenada en el sistema.			
Flujo Normal			
1. Se cliquea en el botón de **Edit payment**. 2. El cliente cambia los datos que desee. 3. Se confirman los cambios realizados. 4. El sistema acepta los cambios y actualiza la información en el mismo.			
Flujo Alternativo			
4.a El sistema no acepta los cambios. 4.a.1. Hay datos erróneos en las modificaciones. Se le notifica al usuario y se vuelve al paso 2 sin realizar modificaciones en los datos del pago.			
Excepción			
Includes			
Requisitos Especiales			
Notas			

Nombre	Pagar tasas	**ID**	14
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por	Raúl José Alcañiz de la Fuente	**Fecha Modif.**	10-01-2015
Actor Principal			
• **Usuario registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: El cliente desea realizar algún abono por la inscripción de algún can a una competición.			
Descripción			
El cliente va a realizar el pago para formalizar la inscripción de algún can a una competición. El sistema actualizará la información en el mismo y quedará reflejado el pago correspondiente.			
Trigger			
Hacer clic en **Upload the Bank Receipt**.			
Precondición			
• Tener algún can registrado inscrito en una competición. • Que haya alguna tarifa pendiente de abonar.			

• Estar en la sección de enrolments.
Postcondición
• El sistema queda actualizado con el pago realizado. • El ejemplar o ejemplares quedan registrados cómo participantes en la competición. • La competición queda actualizada con el nuevo ejemplar participante incluido.
Flujo Normal
1. Se cliquea el botón de **Upload the Bank Receipt.** 2. Se selecciona el archivo a adjuntar. 3. Se añade algún comentario (opcional). 4. Confirmar la acción. 5. El sistema se actualiza con el pago realizado y se le notifica al usuario.
Flujo Alternativo
4.a El sistema no acepta los cambios. 4.a.1. Hay datos erróneos el pago (formato de fichero incorrecto). Se le notifica al usuario y se vuelve al paso 2 sin realizar modificaciones en los datos del pago.
Excepción
5.* No se puede realizar el pago. - El sistema no se actualiza, y se le informa al usuario del error y se le comunica que se ponga en contacto con el administrador del sistema.
Includes
Requisitos Especiales
Notas

Nombre	Ver canes propios	**ID**	15
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Usuario registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: El cliente desea revisar los canes que tiene vinculados a su cuenta.			
Descripción			
Se le muestra al usuario una lista con los ejemplares disponibles registrados en el sistema a su nombre.			
Trigger			
Hacer clic en el botón de **My dogs**.			
Precondición			
Postcondición			
• Se presenta en pantalla la información solicitada por el cliente.			
Flujo Normal			
1. Se cliquea en el botón **My dogs.** 2. El sistema recupera la información y se la muestra al usuario en una tabla.			
Flujo Alternativo			
2.a No hay ejemplares registrados a su cuenta. 2.a.1. No se le muestra información al usuario.			
Excepción			
Includes			
Requisitos Especiales			
Notas			

Nombre	Ver detalles de can	**ID**	16
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Usuario registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: Desea conocer información detallada de alguno de sus ejemplares.			
Descripción			
Se muestra al cliente información detallada de alguno de los ejemplares que tiene registrados a su cuenta de usuario.			
Trigger			
Hacer clic en algún can mostrado en la tabla de ejemplares propios.			
Precondición			
• Tener, como mínimo, un ejemplar inscrito en su cuenta de usuario.			
Postcondición			
• Se le muestra al cliente más información acerca del ejemplar seleccionado.			
Flujo Normal			
1. Se cliquea en el can deseado**.** 2. El sistema recupera la información y la muestra al usuario.			
Flujo Alternativo			
Excepción			
2.* El sistema no puede acceder a la información. - Se comunica al cliente que la operación ha fracasado, que lo intente de nuevo, y si persiste el problema, que se ponga en contacto con el administrador de la web.			
Includes			
Requisitos Especiales			
Notas			

Nombre	Ver detalles de pago	**ID**	17
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Usuario registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: El cliente desea conocer más detalladamente un pago realizado para la inscripción de una competición.			
Descripción			
Se le permite al usuario conocer más detalles acerca de pagos efectuados.			
Trigger			
Hacer clic en algún pago realizado, en el menú de perros enrolados.			
Precondición			
• Encontrarse en el menú de perros enrolados en una exhibición. • Tener algún pago realizado.			
Postcondición			
• El cliente tiene en pantalla más información acerca del pago seleccionado.			
Flujo Normal			
1. Se cliquea en el pago deseado**.** 2. El sistema recupera la información y la muestra al usuario.			
Flujo Alternativo			

Excepción
2.* El sistema no puede acceder a la información. - Se comunica al cliente que la operación ha fracasado, que lo intente de nuevo, y si persiste el problema, que se ponga en contacto con el administrador de la web.
Includes
Requisitos Especiales
Notas

Nombre	Ver exposiciones inscritas	**ID**	18
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Usuario registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: El cliente busca conocer las exposiciones en las que está o ha estado inscrito alguno de sus ejemplares.			
Descripción			
Se le da la posibilidad al usuario de que conozca las competiciones en las que hay algún can inscrito.			
Trigger			
Hacer clic en **My exhibitions**.			
Precondición			
• Desplegar el desplegable de opciones de usuario o estar en el perfil del usuario.			
Postcondición			
• Se muestra una tabla al usuario con las competiciones en las cuáles ha inscrito a alguno de sus canes (actuales o ya concluidas).			
Flujo Normal			
1. Se cliquea en **My exhibitions.** 2. El sistema recupera la información y se la muestra al usuario en una tabla.			
Flujo Alternativo			
2.a No hay exposiciones en las que se está inscrito. 2.a.1. No se muestra ninguna información			
Excepción			
Includes			
Requisitos Especiales			
Notas			

Nombre	Ver históricos de pagos	**ID**	19
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Usuario registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: El interesado desea tener información acerca de temas relacionados con los pagos de inscripción a competiciones.			
Descripción			
En el perfil del usuario se presenta una opción para que el cliente lleve un control de los gastos realizados en su cuenta.			

Trigger
Hacer clic en **My payments**
Precondición
• Desplegar el desplegable de opciones de usuario o estar en el perfil del usuario
Postcondición
• Se muestra una tabla al usuario con información sobre los pagos realizados.
Flujo Normal
1. Se cliquea en el botón **My payments**. 2. El sistema recupera la información y la muestra al usuario.
Flujo Alternativo
2.a No hay ningún pago realizado. 2.a.1. No se muestra información al respecto.
Excepción
Includes
Requisitos Especiales
Notas

Nombre	Ver perfil	**ID**	20
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2014
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Usuario registrado**			
Personal Involucrado o Intereses			
• Usuario registrado: El cliente busca ver su perfil de usuario.			
Descripción			
Opción que permite al usuario ver íntegramente su perfil.			
Trigger			
Hacer clic en **Profile**.			
Precondición			
Postcondición			
• El cliente tiene a la vista su perfil de usuario.			
Flujo Normal			
1. Se cliquea en la opción **Profile.** 2. El sistema recupera la información y la muestra al usuario.			
Flujo Alternativo			
Excepción			
2.* El sistema no puede acceder a la información. - Se comunica al cliente que la operación ha fracasado, que lo intente de nuevo, y si persiste el problema, que se ponga en contacto con el administrador de la web.			
Includes			
Requisitos Especiales			

Administrador

Modulo usuarios

Nombre	Añadir can	**ID**	21
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	11-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Quiere añadir un can al perfil de algún usuario, quedando registrado éste como propietario del perro.			
Descripción			
Opción que permite al administrador añadir un can la lista de ejemplares de un usuario. De esta forma queda reflejado que el usuario propietario del mismo.			
Trigger			
Hacer clic en el botón **Add a dog**.			
Precondición			
• Encontrarse en el perfil del usuario al cual quiere añadirle el perro.			
Postcondición			
• El animal queda inscrito en el registro del sistema como propiedad del cliente.			
Flujo Normal			
1. Se cliquea en el botón de **Add a dog**. 2. Se rellena el formulario. 3. Se confirman los datos introducidos. 4. El sistema almacena los datos del ejemplar vinculando éste al usuario.			
Flujo Alternativo			
4.b Identificador del perro ya existente. 4.b.1. Se le indica al administrador el mensaje de error correspondiente y se vuelve al paso 2.			
Excepción			
Includes			
Se utiliza el caso de uso con ID **27 – Ver perfil usuario**.			
Requisitos Especiales			
Notas			

Nombre	Cancelar cuenta	**ID**	22
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	11-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Quiere cancelar la cuenta de un usuario.			
Descripción			
Opción que permite al administrador eliminar la cuenta de un usuario del sistema. Una vez terminada la operación, la cuenta quedará totalmente borrada del sistema.			
Trigger			
Hacer clic en el botón **Cancel account**.			
Precondición			

• Encontrarse en el perfil del usuario.
Postcondición
• La cuenta queda eliminada del sistema.
Flujo Normal
1. Se cliquea en el botón de **Cancel account**. 2. Se confirma la acción. 3. El sistema actualiza los datos, quedando el usuario eliminado del mismo.
Flujo Alternativo
3.a No se confirma la acción. 3.a.1. El sistema no varía respecto a cómo estaba antes de empezar la operación.
Excepción
2.* La cuenta pertenece al propio administrador. - Se indica al administrador que no es posible borrar su propia cuenta 4.* No se puede eliminar la cuenta del sistema. - No se varía ninguna información en el sistema.
Includes
Se utiliza el caso de uso con ID **27 – Ver perfil usuario**.
Requisitos Especiales
Notas

Nombre	Eliminar can	**ID**	23
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	11-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Quiere eliminar del sistema algún can vinculado a la cuenta de un usuario.			
Descripción			
Opción que permite al administrador eliminar un can de la lista de ejemplares de un usuario. De esta forma se elimina (y se desvincula) el animal del cliente.			
Trigger			
Hacer clic en el botón **Delete dog**.			
Precondición			
• El usuario debe tener algún can vinculado al usuario. • Estar visionando la ficha de detalles del ejemplar a eliminar en cuestión.			
Postcondición			
• El animal queda desvinculado del cliente y eliminado del registro del sistema.			
Flujo Normal			
1. Se cliquea en el botón de **Eliminar can**. 2. Se confirma la acción. 3. El sistema actualiza los datos, quedando dicho ejemplar eliminado del mismo.			
Flujo Alternativo			
Excepción			
Includes			
Se utiliza el caso de uso con ID **27 – Ver perfil usuario**.			
Requisitos Especiales			
Notas			

Nombre	Modificar can	**ID**	24
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	11-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Quiere cambiar información respecto a algún perro registrado en la cuenta de un usuario.			
Descripción			
Se cambia información referente a algún perro del usuario seleccionado, que está almacenada en el sistema.			
Trigger			
Hacer clic en **Edit dog**.			
Precondición			
• Tener algún can vinculado al usuario en cuestión. • Estar visionando la ficha de detalles del ejemplar a editar.			
Postcondición			
• El dato del perro queda modificado a interés del administrador.			
Flujo Normal			
1. Se cliquea en el botón de **Edit dog**. 2. El administrador cambia los datos que desee. 3. Se confirman los cambios realizados. 4. El sistema acepta los cambios y actualiza la información en el mismo.			
Flujo Alternativo			
4.a El sistema no acepta los cambios. 4.a.1. Hay datos erróneos en las modificaciones. Se le notifica al administrador y se vuelve al paso 2 sin realizar modificaciones en los datos del perro.			
Excepción			
Includes			
Se utiliza el caso de uso con ID **27 – Ver perfil usuario**.			
Requisitos Especiales			
Notas			

Nombre	Modificar usuario	**ID**	25
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	11-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Quiere cambiar información personal almacenada en el sistema.			
Descripción			
Se le otorga potestad al Administrador del sistema para cambiar información personal de un usuario registrado, almacenada en el sistema.			
Trigger			
Hacer clic en **Edit person**.			
Precondición			
• Estar en el menú del perfil del usuario.			
Postcondición			
• El perfil del cliente queda modificado a intereses del administrador.			

Flujo Normal
1. Se cliquea en el botón de **Edit person**. 2. El cliente cambia los datos que desee. 3. Se confirman los cambios realizados. 4. El sistema acepta los cambios y actualiza la información en el mismo.
Flujo Alternativo
4.a El sistema no acepta los cambios. 4.a.1. Hay datos erróneos en las modificaciones (contraseña incorrecta, e-mail inválido, datos erróneos…). Se le notifica al administrador y se vuelve al paso 2 sin realizar modificaciones en los datos del usuario.
Excepción
Includes
Se utiliza el caso de uso con ID **27 – Ver perfil usuario**.
Requisitos Especiales
Notas

Nombre	Ver detalles de can	**ID**	26
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	11-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Desea conocer información detallada de alguno de los perros de un cliente.			
Descripción			
Se muestra al administrador información detallada de alguno de los ejemplares que tiene registrados en la cuenta de usuario.			
Trigger			
Hacer clic en algún can mostrado en la tabla de ejemplares del cliente.			
Precondición			
• Que haya, como mínimo, un ejemplar inscrito en su cuenta del usuario.			
Postcondición			
• Se le muestra al administrador más información acerca del ejemplar seleccionado.			
Flujo Normal			
1. Se cliquea en el can deseado**.** 2. El sistema recupera la información y la muestra al administrador.			
Flujo Alternativo			
Excepción			
2.* El sistema no puede acceder a la información. - Se comunica al administrador que la operación ha fracasado y el motivo.			
Includes			
Se utiliza el caso de uso con ID **27 – Ver perfil usuario**.			
Requisitos Especiales			
Notas			

Nombre	Ver perfil usuario	**ID**	27
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	11-01-2015
Modif. Por		***Fecha Modif.***	

Actor Principal
• **Administrador**
Personal Involucrado o Intereses
• Administrador: El administrador busca ver el perfil de un usuario.
Descripción
Opción que permite al administrador ver íntegramente el perfil de un usuario.
Trigger
Hacer clic en el usuario a mostrar.
Precondición
• Estar en la sección de administración de usuarios.
Postcondición
• El administrador tiene a la vista del perfil del usuario seleccionado.
Flujo Normal
1. Se cliquea en el usuario que se quiere ver. 2. El sistema recupera la información y la muestra al usuario.
Flujo Alternativo
Excepción
Includes
Requisitos Especiales
Notas

Módulo personas/perfiles

Nombre	Crear persona	**ID**	28
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	11-01-2015
Modif. Por		**Fecha Modif.**	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: El administrador pretende introducir datos de una nueva persona en el sistema para que quede registrado en el mismo.			
Descripción			
El administrador tiene la posibilidad de rellenar un formulario de registro para añadir a una nueva persona al sistema; el perfil quedará incluido en el sistema.			
Trigger			
Hacer clic en el botón de **New person**, en el panel principal de gestión de usuarios.			
Precondición			
• Estar en el apartado de usuarios del panel de administrador, en el portal web.			
Postcondición			
• Existe un perfil de una nueva persona.			
Flujo Normal			
1. Se cliquea en el botón de **New person**. 2. Se rellena el formulario de registro. 3. El administrador confirma los datos. 4. El sistema valida los datos.			
Flujo Alternativo			
4.a Algún dato introducido por el Administrador es incorrecto (nombre de usuario incorrecto, no seleccionar el sexo…). 4.a.1. Se informa al administrador del dato/s inválido/s y se le da la opción que lo/s			

ingrese de nuevo, volviendo al paso 2.
Excepción
Includes
Requisitos Especiales
Notas

Nombre	Eliminar Persona	**ID**	29
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	11-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Quiere cancelar la cuenta de una persona.			
Descripción			
Opción que permite al administrador eliminar la cuenta de personas en el sistema. Una vez terminada la operación, la cuenta quedará totalmente borrada del sistema.			
Trigger			
Hacer clic en el botón **Cancel account**.			
Precondición			
• Encontrarse en el perfil de la persona.			
Postcondición			
• La cuenta queda eliminada del sistema.			
Flujo Normal			
1. Se cliquea en el botón de **Cancel account**. 2. Se confirma la acción. 3. El sistema actualiza los datos, quedando el usuario eliminado del mismo.			
Flujo Alternativo			
3.a No se confirma la acción. 3.a.1. El sistema no varía respecto a cómo estaba antes de empezar la operación.			
Excepción			
2.* La cuenta pertenece al propio administrador. - Se indica al administrador que no es posible borrar su propia cuenta 4.* No se puede eliminar la cuenta del sistema. - No se varía ninguna información en el sistema.			
Includes			
Requisitos Especiales			
Notas			

Nombre	Modificar persona	**ID**	30
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	11-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Poder modificar información personal referente a una persona que exista en el sistema.			
Descripción			
El administrador es capaz de cambiar la información de cualquier persona o perfil que haya			

en el sistema.
Trigger
Hacer clic en el botón **Edit person**.
Precondición
• Estar en el menú del perfil del usuario.
Postcondición
• El perfil de la persona queda modificado a intereses del administrador.
Flujo Normal
1. Se cliquea en el botón de **Edit person**. 2. El administrador modifica los datos requeridos. 3. Se confirman los cambios realizados. 4. El sistema acepta los cambios y actualiza la información en el mismo.
Flujo Alternativo
4.a El sistema no acepta los cambios. 4.a.1. Hay datos erróneos en las modificaciones (contraseña incorrecta, e-mail inválido, datos erróneos…). Se le notifica al administrador y se vuelve al paso 2 sin realizar modificaciones en los datos del usuario.
Excepción
Includes
Requisitos Especiales
Notas

Módulo exposiciones

Nombre	Añadir exposición	**ID**	31
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	11-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Definir los requisitos de una exposición.			
Descripción			
Un administrador podrá crear una nueva exposición en el sistema con los datos necesarios. Los datos quedarán almacenados en el mismo y la exposición quedará publicada en el apartado de exhibiciones.			
Trigger			
El caso se inicia al cliquear en el botón **New exhibition**.			
Precondición			
• Encontrarse en la sección de las exhibiciones			
Postcondición			
• Existe en el sistema una nueva exposición con la información que se haya especificado.			
Flujo Normal			
1. Cliquear en el botón **New exhibition**. 2. Rellenar los campos necesarios para la creación de una nueva exposición. 3. Confirmar los datos. 4. El sistema verifica los datos. 5. El sistema se actualiza con la nueva información introducida.			
Flujo Alternativo			
Excepción			

4.* Los datos introducidos son erróneos. - Se indica al administrador que hay errores en los datos introducidos, y el motivo del error (nombre incorrecto, fechas incorrectas, fichero de imagen erróneo…)
Includes
Requisitos Especiales
Notas

Nombre	Eliminar exposición	**ID**	32
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	11-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Quiere eliminar del sistema alguna exposición que ha creado.			
Descripción			
Opción que permite al administrador eliminar una exposición del sistema. Una vez terminada la operación, la exposición quedará totalmente borrada del sistema.			
Trigger			
Hacer clic en el botón **Delete exhibition**.			
Precondición			
• Estar en el menú de detalles de exhibición. • Que el deadline de los pagos no haya comenzado.			
Postcondición			
• La exposición queda eliminada del registro del sistema.			
Flujo Normal			
1. Se cliquea en el botón de **Delete exhibition**. 2. Se confirma la acción. 3. El sistema actualiza los datos, quedando el evento eliminado del mismo.			
Flujo Alternativo			
3.b No se confirma la acción. 3.b.1. El sistema no varía respecto a cómo estaba antes de empezar la operación.			
Excepción			
3.* No se puede eliminar la exposición del sistema. - Si el periodo de pagos ha comenzado no es posible eliminar la exhibición del sistema. Se le notifica al administrador con el correspondiente mensaje.			
Includes			
Se utiliza el caso de uso con ID **04 – Mostrar detalles de exposición**.			
Requisitos Especiales			
Notas			

Nombre	Modificar exposición publicada	**ID**	33
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	11-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: El administrador desea modificar una exposición pública, existente en el sistema.			

Descripción
Se le permite al administrador modificar cualquier exposición que haya en el sistema y cuya periodo de pago no haya comenzado o terminado.
Trigger
Clickear en **Edit exhibition**.
Precondición
• Encontrarse en los detalles de la exhibición. • Que la deadline de pagos no ha empezado o concluido.
Postcondición
• La información existente en el sistema, respecto a una exposición concreta, queda modificada y actualizada.
Flujo Normal
1. Se cliquea en el botón de **Edit exhibition**. 2. Se realizan los cambios deseados. 3. El administrador confirma los datos. 4. El sistema confirma los datos. 5. El sistema guarda los cambios.
Flujo Alternativo
Excepción
2.* La exhibición ya ha empezado el periodo de pagos. - Se le muestra al administrador un mensaje indicando que no es posible la modificación de esta exposición.
Includes
Se utiliza el caso de uso con ID **04 – Mostrar detalles de exposición**.
Requisitos Especiales
Notas

Módulo razas

Nombre	Crear grupo (Create)	**ID**	34
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Crear un nuevo grupo de razas que se pretende almacenar en el sistema.			
Descripción			
El Administrador del sistema puede crear los grupos que necesite para la correcta definición de las razas.			
Trigger			
El caso se inicia al cliquear en el botón **New group**.			
Precondición			
• Encontrarse la página de las nomenclaturas y estándares.			
Postcondición			
• En el sistema existe un nuevo grupo al que se le puede asignar secciones.			
Flujo Normal			
1. Cliquear en el botón **New group**. 2. Rellenar los campos necesarios para la creación del nuevo grupo.			

3. Aceptar los datos. 4. El sistema valida los datos. 5. El sistema se actualiza con la nueva información introducida.
Flujo Alternativo
4.a El sistema no acepta los cambios. 4.a.1. Hay datos erróneos introducidos (Nombre repetido, no se define descripción…). Se le notifica al administrador y se vuelve al paso 2 sin realizar la creación del grupo en cuestión.
Excepción
Includes
Requisitos Especiales
Notas

Nombre	Leer grupo (Read)	**ID**	35
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador, Visitante**			
Personal Involucrado o Intereses			
• Administrador y visitante: Ver los grupos de razas que existen en el sistema.			
Descripción			
Se puede navegar por el sitio web para ver los grupos que existen en el mismo.			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. Cliquear para entrar en la sección de estándares y nomenclaturas. 2. El sistema visualiza el contenido.			
Flujo Alternativo			
Excepción			
Includes			
Requisitos Especiales			
Notas			
Es un caso incluido en el caso de uso **01 – Consultar documentación**, que se ha decidido incluir en la sección de administrador por las capacidades CRUD (Create – Read – Update – Delete) que deben tener los grupos.			

Nombre	Modificar grupo (Update)	**ID**	36
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Alterar la información pertinente de uno de los grupos que existen en el sistema			
Descripción			
Cuando sea necesario, el administrador del sistema podrá cambiar el nombre o la descripción de alguno de los grupos existentes en el mismo.			

Trigger
El caso se inicia al cliquear en el botón **Edit group**.
Precondición
• Encontrarse visionando la página del grupo en cuestión.
Postcondición
• El grupo queda modificado de acuerdo con las características que el administrador ha definido.
Flujo Normal
1. Cliquear en el botón **Edit group.** 2. Rellenar el formulario correspondiente. 3. Aceptar los cambios. 4. El sistema verifica los datos. 5. El sistema almacena los nuevos datos referenciados al grupo modificado.
Flujo Alternativo
4.a El sistema no acepta los cambios. 4.a.1. Hay datos erróneos introducidos (Nombre repetido, no se define descripción…). Se le notifica al administrador y se vuelve al paso 2 sin realizar modificaciones en los datos del grupo.
Excepción
Includes
Requisitos Especiales
Notas

Nombre	Borrar grupo (Delete)	**ID**	37
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Eliminar del sistema un grupo de razas existente.			
Descripción			
Cuando sea necesario, el administrador del sistema va a tener posibilidad de eliminar grupos de razas almacenados en el sistema (Con la consecuencia de un borrado en cascada y el borrado de las secciones, sub-secciones y toda la rama por debajo de este grupo).			
Trigger			
El caso se inicia al cliquear en el botón **Delete group**.			
Precondición			
• Encontrarse visionando la página del grupo en cuestión.			
Postcondición			
• El grupo queda eliminado y borrado del sistema			
Flujo Normal			
1. Cliquear en el botón **Delete group.** 2. Aceptar el borrado del grupo. 3. El sistema elimina del sistema la información del grupo y de las secciones relacionadas con el mismo.			
Flujo Alternativo			
Excepción			
Includes			

Requisitos Especiales
Notas

Nombre	Crear sección (Create)	**ID**	38
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Crear una nueva sección para un grupo de razas.			
Descripción			
El Administrador del sistema puede crear las secciones que sean necesarias dentro de cada grupo para la correcta definición de las razas.			
Trigger			
El caso se inicia al cliquear en el botón **New section**.			
Precondición			
• Encontrarse la página del grupo al que se le va a vincular la sección.			
Postcondición			
• En el sistema existe una nueva sección perteneciente a un determinado grupo.			
Flujo Normal			
1. Cliquear en el botón **New section**. 2. Rellenar los campos necesarios para la creación de la nueva sección. 3. Aceptar los datos. 4. El sistema valida los datos. 5. El sistema se actualiza con la nueva información introducida.			
Flujo Alternativo			
4.b El sistema no acepta los cambios. 4.b.1. Hay datos erróneos introducidos (Nombre repetido, no se define descripción…). Se le notifica al administrador y se vuelve al paso 2 sin crear la nueva sección.			
Excepción			
Includes			
Requisitos Especiales			
Notas			

Nombre	Leer sección (Read)	**ID**	39
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador, Visitante**			
Personal Involucrado o Intereses			
• Administrador y visitante: Ver las secciones de razas que existen en el sistema.			
Descripción			
Se puede navegar por el sitio web para ver las secciones que existen en el mismo.			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. Cliquear en el grupo del cual se desean ver las secciones.			

2. El sistema visualiza el contenido.
Flujo Alternativo
Excepción
Includes
Requisitos Especiales
Notas
Es un caso incluido en el caso de uso **01 – Consultar documentación**, que se ha decidido incluir en la sección de administrador por las capacidades CRUD (Create – Read – Update – Delete) que deben tener las secciones.

Nombre	Modificar sección (Update)	**ID**	40
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Alterar la información pertinente de una de las secciones que existen en el sistema			
Descripción			
Cuando sea necesario, el administrador del sistema podrá cambiar el nombre o la descripción de alguna de las secciones existentes en el mismo.			
Trigger			
El caso se inicia al cliquear en el botón **Edit section**.			
Precondición			
• Encontrarse visionando la página de la sección en cuestión.			
Postcondición			
• La sección queda modificada de acuerdo con las características que el administrador ha definido.			
Flujo Normal			
1. Cliquear en el botón **Edit section.** 2. Rellenar el formulario correspondiente. 3. Aceptar los cambios. 4. El sistema verifica los datos. 5. El sistema almacena los nuevos datos referenciados a la sección modificada.			
Flujo Alternativo			
4.b El sistema no acepta los cambios. 4.b.1. Hay datos erróneos introducidos (Nombre repetido, no se define descripción…). Se le notifica al administrador y se vuelve al paso 2 sin realizar modificaciones en los datos de la sección.			
Excepción			
Includes			
Requisitos Especiales			
Notas			

Nombre	Borrar sección (Delete)	**ID**	41
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			

• **Administrador**
Personal Involucrado o Intereses
• Administrador: Eliminar del sistema una sección existente.
Descripción
Cuando sea necesario, el administrador del sistema va a tener posibilidad de eliminar secciones almacenadas en el sistema (Con la consecuencia de un borrado en cascada y el borrado de las sub-secciones y toda la rama por debajo de esta sección – razas, variedades y sub-variedades).
Trigger
El caso se inicia al cliquear en el botón **Delete section**.
Precondición
• Encontrarse visionando la página de la sección a borrar.
Postcondición
• La sección queda eliminada y borrada del sistema
Flujo Normal
1. Cliquear en el botón **Delete section.** 2. Aceptar el borrado de la sección. 3. El sistema elimina del sistema la información de la sección y las sub-secciones y demás datos relacionados con la misma.
Flujo Alternativo
Excepción
Includes
Requisitos Especiales
Notas

Nombre	Crear sub-sección (Create)	**ID**	42
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Crear una nueva sub-sección para una sección existente en el sistema.			
Descripción			
El Administrador del sistema puede crear las sub-secciones que sean necesarias dentro de cada sección para la correcta definición de las razas.			
Trigger			
El caso se inicia al cliquear en el botón **New subsection**.			
Precondición			
• Encontrarse la página de la sección a la que se le va a vincular la sub-sección.			
Postcondición			
• En el sistema existe una nueva sub-sección perteneciente a una determinada sección.			
Flujo Normal			
1. Cliquear en el botón **New subsection**. 2. Rellenar los campos necesarios para la creación de la nueva sub-sección. 3. Aceptar los datos. 4. El sistema valida los datos. 5. El sistema se actualiza con la nueva información introducida.			
Flujo Alternativo			
4.c El sistema no acepta los cambios.			

4.c.1. Hay datos erróneos introducidos (Nombre repetido, no se define descripción…). Se le notifica al administrador y se vuelve al paso 2 sin crear la nueva subsección.
Excepción
Includes
Requisitos Especiales
Notas

Nombre	Leer sub-sección (Read)	**ID**	43
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador, Visitante**			
Personal Involucrado o Intereses			
• Administrador y visitante: Ver las sub-secciones de razas que existen en el sistema.			
Descripción			
Se puede navegar por el sitio web para ver las sub-secciones que existen en el mismo.			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. Cliquear en la sección de la cual se desean ver las sub-secciones. 2. El sistema visualiza el contenido.			
Flujo Alternativo			
Excepción			
Includes			
Requisitos Especiales			
Notas			
Es un caso incluido en el caso de uso **01 – Consultar documentación**, que se ha decidido incluir en la sección de administrador por las capacidades CRUD (Create – Read – Update – Delete) que deben tener las subsecciones.			

Nombre	Modificar sub-sección (Update)	**ID**	44
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Alterar la información pertinente de una de las sub-secciones que existen en el sistema			
Descripción			
Cuando sea necesario, el administrador del sistema podrá cambiar el nombre o la descripción de alguna de las sub-secciones existentes en el mismo.			
Trigger			
El caso se inicia al cliquear en el botón **Edit subsection**.			
Precondición			
• Encontrarse visionando la página de la sub-sección en cuestión.			
Postcondición			
• La sub-sección queda modificada de acuerdo con las características que el			

administrador ha definido.
Flujo Normal
1. Cliquear en el botón **Edit subsection.** 2. Rellenar el formulario correspondiente. 3. Aceptar los cambios. 4. El sistema verifica los datos. 5. El sistema almacena los nuevos datos referenciados a la sub-sección modificada.
Flujo Alternativo
4.c El sistema no acepta los cambios. 4.c.1. Hay datos erróneos introducidos (Nombre repetido, no se define descripción…). Se le notifica al administrador y se vuelve al paso 2 sin realizar modificaciones en los datos de la sub-sección.
Excepción
Includes
Requisitos Especiales
Notas

Nombre	Borrar sub-sección (Delete)	**ID**	45
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Eliminar del sistema una sub-sección existente.			
Descripción			
Cuando sea necesario, el administrador del sistema va a tener posibilidad de eliminar sub-secciones almacenadas en el mismo (Con la consecuencia de un borrado en cascada y el borrado de la rama por debajo de esta sub-sección – razas, variedades y sub-variedades).			
Trigger			
El caso se inicia al cliquear en el botón **Delete subsection**.			
Precondición			
• Encontrarse visionando la página de la sub-sección a borrar.			
Postcondición			
• La subsección queda eliminada y borrada del sistema			
Flujo Normal			
1. Cliquear en el botón **Delete subsection.** 2. Aceptar el borrado de la sub-sección. 3. El sistema elimina del sistema la información de las sub-secciones y demás datos relacionados con la misma.			
Flujo Alternativo			
Excepción			
Includes			
Requisitos Especiales			
Notas			

Nombre	Crear sub-variedad (Create)	**ID**	46
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	

Actor Principal
• **Administrador**
Personal Involucrado o Intereses
• Administrador: Crear una nueva sub-variedad para una variedad de raza existente en el sistema.
Descripción
El Administrador del sistema puede crear las sub-variedades que sean necesarias dentro de cada sección para la correcta definición de las razas.
Trigger
El caso se inicia al cliquear en el botón **New subvariety**.
Precondición
• Encontrarse la página de la variedad a la que se le va a vincular la sub-variedad.
Postcondición
• En el sistema existe una nueva sub-variedad perteneciente a una determinada variedad.
Flujo Normal
1. Cliquear en el botón **New subvariety**. 2. Rellenar los campos necesarios para la creación de la nueva sub-variedad. 3. Aceptar los datos. 4. El sistema valida los datos. 5. El sistema se actualiza con la nueva información introducida.
Flujo Alternativo
4.d El sistema no acepta los cambios. 4.d.1. Hay datos erróneos introducidos (Nombre repetido). Se le notifica al administrador y se vuelve al paso 2 sin crear la nueva sub-variedad.
Excepción
Includes
Requisitos Especiales
Notas

Nombre	Leer sub-variedad (Read)	**ID**	47
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador, Visitante**			
Personal Involucrado o Intereses			
• Administrador y visitante: Ver las sub-variedades de razas que existen en el sistema.			
Descripción			
Se puede navegar por el sitio web para ver las sub-variedades que existen en el mismo.			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. Cliquear en la sección de la cual se desean ver las sub-variedades. 2. El sistema visualiza el contenido.			
Flujo Alternativo			
Excepción			
Includes			
Requisitos Especiales			

Notas
Es un caso incluido en el caso de uso **01 – Consultar documentación**, que se ha decidido incluir en la sección de administrador por las capacidades CRUD (Create – Read – Update – Delete) que deben tener las sub-variedades.

Nombre	Modificar sub-varieadad (Update)	**ID**	48
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Alterar la información pertinente de una de las sub-variedades que existen en el sistema			
Descripción			
Cuando sea necesario, el administrador del sistema podrá cambiar el nombre de alguna de las sub-variedades existentes en el mismo.			
Trigger			
El caso se inicia al cliquear en el botón **Edit subvariety**.			
Precondición			
• Encontrarse visionando la página de la sub-variedad en cuestión.			
Postcondición			
• La sub-variedad queda modificada de acuerdo con las características que el administrador ha definido.			
Flujo Normal			
1. Cliquear en el botón **Edit subvariety.** 2. Rellenar el formulario correspondiente. 3. Aceptar los cambios. 4. El sistema verifica los datos. 5. El sistema almacena los nuevos datos referenciados a la sub-variedad modificada.			
Flujo Alternativo			
4.d El sistema no acepta los cambios. 4.d.1. Hay datos erróneos introducidos (Nombre repetido, no se define descripción…). Se le notifica al administrador y se vuelve al paso 2 sin realizar modificaciones en los datos de la sub-variedad.			
Excepción			
Includes			
Requisitos Especiales			
Notas			

Nombre	Borrar sub-variedad (Delete)	**ID**	49
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Eliminar del sistema una sub-variedad existente.			
Descripción			
Cuando sea necesario, el administrador del sistema va a tener posibilidad de eliminar sub-			

variedades de raza almacenadas en el mismo.
Trigger
El caso se inicia al cliquear en el botón **Delete subvariety**.
Precondición
• Encontrarse visionando la página de la sub-variedad a borrar.
Postcondición
• La sub-variedad queda eliminada y borrada del sistema
Flujo Normal
1. Cliquear en el botón **Delete subvariety.** 2. Aceptar el borrado de la sub-variedad. 3. El sistema elimina del sistema la información de la sub-variedad.
Flujo Alternativo
Excepción
Includes
Requisitos Especiales
Notas

Nombre	Crear raza (Create)	**ID**	50
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Crear una nueva raza en el sistema.			
Descripción			
El Administrador del sistema puede crear las razas que sean necesarias dentro de cada sección o sub-sección.			
Trigger			
El caso se inicia al cliquear en el botón **New breed**.			
Precondición			
• Encontrarse la página de la sección o sub-sección al que se le va a vincular la raza.			
Postcondición			
• En el sistema existe una nueva raza perteneciente a una determinada sección o sub-sección.			
Flujo Normal			
1. Cliquear en el botón **New breed**. 2. Rellenar los campos necesarios para la creación de la nueva raza. 3. Aceptar los datos. 4. El sistema valida los datos. 5. El sistema se actualiza con la nueva información introducida.			
Flujo Alternativo			
4.e El sistema no acepta los cambios. 4.e.1. Hay datos erróneos introducidos (Nombre repetido). Se le notifica al administrador y se vuelve al paso 2 sin crear la nueva raza.			
Excepción			
Includes			
Requisitos Especiales			
Notas			

Nombre	Leer raza (Read)	**ID**	51
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador, Visitante**			
Personal Involucrado o Intereses			
• Administrador y visitante: Ver las razas que existen en el sistema.			
Descripción			
Se puede navegar por el sitio web para ver las razas que existen en el mismo.			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. Cliquear en la sección o sub-sección en el que se desean ver las razas. 2. El sistema visualiza el contenido.			
Flujo Alternativo			
Excepción			
Includes			
Requisitos Especiales			
Notas			
Es un caso incluido en el caso de uso **01 – Consultar documentación**, que se ha decidido incluir en la sección de administrador por las capacidades CRUD (Create – Read – Update – Delete) que deben tener las razas.			

Nombre	Modificar raza (Update)	**ID**	52
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Alterar la información pertinente de una de las razas que existen en el sistema			
Descripción			
Cuando sea necesario, el administrador del sistema podrá cambiar el nombre de alguna de las razas existentes en el mismo.			
Trigger			
El caso se inicia al cliquear en el botón **Edit breed**.			
Precondición			
• Encontrarse visionando la página de la raza en cuestión.			
Postcondición			
• La raza queda modificada de acuerdo con las características que el administrador ha definido.			
Flujo Normal			
1. Cliquear en el botón **Edit breed.** 2. Rellenar el formulario correspondiente. 3. Aceptar los cambios. 4. El sistema verifica los datos. 5. El sistema almacena los nuevos datos referenciados a la raza modificada.			
Flujo Alternativo			

4.e El sistema no acepta los cambios. 4.e.1. Hay datos erróneos introducidos (Nombre repetido). Se le notifica al administrador y se vuelve al paso 2 sin realizar modificaciones en los datos de la raza.
Excepción
Includes
Requisitos Especiales
Notas

Nombre	Borrar raza (Delete)	**ID**	53
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Eliminar del sistema una raza existente.			
Descripción			
Cuando sea necesario, el administrador del sistema va a tener posibilidad de eliminar razas almacenadas en el sistema (Con la consecuencia de un borrado en cascada y el borrado de las variedades y sub-variedades relacionadas con la misma).			
Trigger			
El caso se inicia al cliquear en el botón **Delete breed**.			
Precondición			
• Encontrarse visionando la página de la raza a borrar.			
Postcondición			
• La raza queda eliminada y borrada del sistema			
Flujo Normal			
1. Cliquear en el botón **Delete breed.** 2. Aceptar el borrado de la raza. 3. El sistema elimina del sistema la información de la raza y las variedades y sub-variedades relacionadas con la misma.			
Flujo Alternativo			
Excepción			
Includes			
Requisitos Especiales			
Notas			

Nombre	Crear variedad (Create)	**ID**	54
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Crear una nueva variedad para una raza.			
Descripción			
El Administrador del sistema puede crear las variedades que sean necesarias dentro de cada raza para la correcta definición de las mismas.			
Trigger			

El caso se inicia al cliquear en el botón **New variety**.
Precondición
• Encontrarse la página de la raza al que se le va a vincular la variedad.
Postcondición
• En el sistema existe una nueva variedad perteneciente a una determinada raza.
Flujo Normal
1. Cliquear en el botón **New variety**. 2. Rellenar los campos necesarios para la creación de la nueva variedad. 3. Aceptar los datos. 4. El sistema valida los datos. 5. El sistema se actualiza con la nueva información introducida.
Flujo Alternativo
4.f El sistema no acepta los cambios. 4.f.1. Hay datos erróneos introducidos (Nombre repetido). Se le notifica al administrador y se vuelve al paso 2 sin crear la nueva variedad.
Excepción
Includes
Requisitos Especiales
Notas

Nombre	Leer variedad (Read)	**ID**	55
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador, Visitante**			
Personal Involucrado o Intereses			
• Administrador y visitante: Ver las variedades de razas que existen en el sistema.			
Descripción			
Se puede navegar por el sitio web para ver las variedades de razas que existen en el mismo.			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. Cliquear en la raza de la cual se desean ver las variedades. 2. El sistema visualiza el contenido.			
Flujo Alternativo			
Excepción			
Includes			
Requisitos Especiales			
Notas			
Es un caso incluido en el caso de uso **01 – Consultar documentación**, que se ha decidido incluir en la sección de administrador por las capacidades CRUD (Create – Read – Update – Delete) que deben tener las variedades.			

Nombre	Modificar variedad (Update)	**ID**	56
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	

Actor Principal
• **Administrador**
Personal Involucrado o Intereses
• Administrador: Alterar la información pertinente de una de las variedades de razas que existen en el sistema
Descripción
Cuando sea necesario, el administrador del sistema podrá cambiar el nombre de alguna de las variedades existentes en el mismo.
Trigger
El caso se inicia al cliquear en el botón **Edit variety**.
Precondición
• Encontrarse visionando la página de la variedad en cuestión.
Postcondición
• La variedad queda modificada de acuerdo con las características que el administrador ha definido.
Flujo Normal
1. Cliquear en el botón **Edit variety.** 2. Rellenar el formulario correspondiente. 3. Aceptar los cambios. 4. El sistema verifica los datos. 5. El sistema almacena los nuevos datos referenciados a la variedad modificada.
Flujo Alternativo
4.f El sistema no acepta los cambios. 4.f.1. Hay datos erróneos introducidos. Se le notifica al administrador y se vuelve al paso 2 sin realizar modificaciones en los datos de la variedad.
Excepción
Includes
Requisitos Especiales
Notas

Nombre	Borrar variedad (Delete)	**ID**	57
Creado por	Raúl José Alcañiz de la Fuente	**Fecha**	13-01-2015
Modif. Por		***Fecha Modif.***	
Actor Principal			
• **Administrador**			
Personal Involucrado o Intereses			
• Administrador: Eliminar del sistema una variedad existente.			
Descripción			
Cuando sea necesario, el administrador del sistema va a tener posibilidad de eliminar variedades de razas almacenadas en el sistema (Con la consecuencia de un borrado en cascada y el borrado de las sub-variedades dependientes de esta variedad).			
Trigger			
El caso se inicia al cliquear en el botón **Delete variety**.			
Precondición			
• Encontrarse visionando la página de la variedad a borrar.			
Postcondición			
• La variedad queda eliminada y borrada del sistema			
Flujo Normal			
1. Cliquear en el botón **Delete variety.**			

2. Aceptar el borrado de la variedad. 3. El sistema elimina del sistema la información de la variedad y las sub-variedades relacionadas con la misma.
Flujo Alternativo
Excepción
Includes
Requisitos Especiales
Notas

Anexo II – Fichero de prueba de iteración 12 - Viewing enrolments

```
@time_travel
Feature: Viewing enrolments
  In order to view enrolments
  As user
  I want to see them on the enrolments page

  Background:
    Given today is "10-12-2013"
    Given there are the following users:
      | email              | password |
      | user@testing.com   | password |
    And I am signed in as "user@testing.com"

    Given "user@testing.com" is owner for some dogs:
      | Name       | Date of birth | Sire | Dam   | Sex  | Titles           |
      | Snoopy     | 12/12/2012    | Rex  | Missy | Male | PORTO WINNER Jr. |
      | Scooby Doo | 27/01/2008    | Skip | Peggy | Male |                  |

    Given there are exhibitions with these entries:
      | Name                   | Description     | Start date  | End date    |
      | Exp. Canina de Cieza   | www.example.org | Sep 1, 2014 | Sep 3, 2014 |
      | Amsterdam Winner Show  | www.example.org | Sep 1, 2014 | Sep 3, 2014 |

    Given there are these entry deadlines for "Exp. Canina de Cieza":
      | Name                | Start date | End date   |
      | 1st entry deadline | 08-01-2014 | 22-02-2014 |
      | 2nd entry deadline | 23-02-2014 | 03-03-2014 |
      | 3rd entry deadline | 04-03-2014 | 14-03-2014 |

    Given there are these payments for "partners" in "Exp. Canina de Cieza":
      | Class          | Dogs             | 1st entry deadline|2nd entry deadline| 3rd entry deadline|
      | J. I. O. W. CH. | 1st dog          | 24.00             |30.00             | 36.00             |
      | J. I. O. W. CH. | 2nd dog          | 19.00             |23.00             | 27.00             |
      | J. I. O. W. CH. | 3rd and following| 14.00             |18.00             | 21.00             |

    Given there are these payments for "nopartners" in "Exp. Canina de Cieza":
      | Class          | Dogs             | 1st entry deadline|2nd entry deadline| 3rd entry deadline|
      | J. I. O. W. CH. | 1st dog          | 24.00             | 30.00            | 36.00             |
      | J. I. O. W. CH. | 2nd dog          | 19.00             | 23.00            | 27.00             |
      | J. I. O. W. CH. | 3rd and following| 14.00             | 18.00            | 21.00             |

    And I have "Scooby Doo" enrolled in "Exp. Canina de Cieza" in "Open" class on "11-02-2014"
    And I have sent the payment of "Scooby Doo" for "Open" class
    And the payment status of "Scooby Doo" for "Open" class is "accepted"
    Given I have "Snoopy" enrolled unpartnered in "Exp. Canina de Cieza" in "Champion" class on "10-02-2014"
    And I have not sent the payment of "Snoopy" for "Champion" class

  @done
  Scenario: Viewing enrolments from the exhibitions page
    Given I am on the exhibitions page
    When I follow "My enrolments" for "Exp. Canina de Cieza"
    Then I should be on the enrolments page for "Exp. Canina de Cieza"
    And I should see "Snoopy" and "Champion"
    And I should see "TOTAL" and "24.00"
    And I should see "Scooby Doo" and "Open"
    And I should see "19.00"
    And I should see "Status" and "Accepted"
    And I should see "See Payment" for the payment of "Scooby Doo"
```

Continúa en la página siguiente…

```
@done
 Scenario: Viewing enrolments from the profile
   Given I am on my profile page
   When I follow "My enrolments"
   Then I should be on the exhibitions page

   When I follow "My enrolments" for "Exp. Canina de Cieza"
   Then I should be on the enrolments page for "Exp. Canina de Cieza"
   And I should see "Snoopy" and "Champion"
   And I should see "TOTAL" and "24.00"
   And I should see "Scooby Doo" and "Open"
   And I should see "19.00"
   And I should see "Status" and "Accepted"
   And I should see "See Payment" for the payment of "Scooby Doo"

 @done
 Scenario: Viewing enrolments from the page of one exhibition
   Given I am on the exhibition page for "Exp. Canina de Cieza"
   Then I should see "My enrolments for Exp. Canina de Cieza"
   When I follow "My enrolments for Exp. Canina de Cieza"
   Then I should be on the enrolments page for "Exp. Canina de Cieza"
   And I should see "Snoopy" and "Champion"
   And I should see "TOTAL" and "24.00"
   And I should see "Scooby Doo" and "Open"
   And I should see "19.00"
   And I should see "Status" and "Accepted"
   And I should see "See Payment" for the payment of "Scooby Doo"

 @done
 Scenario: Trying to view enrolments in which I haven't dogs enrolled should be alerted
   Given I am on the enrolments page for "Amsterdam Winner Show"
   Then I should see "You haven't dogs enrolled in this exhibition."
   And I should see "Enroll a new dog" button

 @done
 Scenario: Enrolments that have receipts pending to upload in exhibitions
           already finished should not be displayed

   Given today is "04-09-2014"
   Then the exhibition "Exp. Canina de Cieza" has already finished

   Given I am on the exhibitions page
   When I follow "My enrolments" for "Exp. Canina de Cieza"
   Then I should be on the enrolments page for "Exp. Canina de Cieza"
   And I should not see "Snoopy" and "Champion"
   But I should see "Scooby Doo" and "Open"
   And I should see "Status" and "Accepted"
   And I should not see "Upload the Bank Receipt"
```

Anexo III – Fichero de implementación de las pruebas de los enrolments

```
Then(/^"(.*?)" should( not)? be enrolled for "(.*?)"$/) do |dog_name, negate, exhibition_name|
  if negate
    Enrolment.count.should eq 0
  else
    @enrolment = Enrolment.find_by(dog_id: Dog.find_by(name: "#{dog_name}").id,
                           exhibition_id: Exhibition.find_by(name: "#{exhibition_name}").id)
    @enrolment.should_not be_nil
  end
end

When(/^I follow "(.*?)" for "(.*?)" enrolled in "(.*?)" class$/) do |action, dog_name, dog_class|

  enrol_id = Enrolment.find_by(dog_id: Dog.find_by_name("#{dog_name}").id, dog_class: dog_class).id
  exhib_id = (Exhibition.find_by_name 'Exp. Canina de Cieza').id

  page.all('a', text: "#{action}").each do |link|
    link.click if link[:href] == "/exhibitions/#{exhib_id}/enrolments/#{enrol_id}"
  end

end

Given(/^I have "(.*?)" enrolled( unpartnered)? in "(.*?)" in "(.*?)" class on "(.*?)"$/) do
|dog_name, nopartner, exhibition_name, dog_class, date|
  steps %{
    Given I am on the exhibitions page
    And I follow "#{exhibition_name}"
    Given today is "#{date}"
    And I press "Enroll a new dog"
    When I select "#{dog_name}" from "enrolment_dog_id"
    And I select "#{dog_class}" from "enrolment_dog_class"
    }
    step 'I check "Partner"' if !nopartner
  steps %{
    When I press "Create Enrolment"
    And I press "Create Enrolment"
    Then "#{dog_name}" should be enrolled for "#{exhibition_name}"
    Given I am on the exhibitions page
    And I follow "#{exhibition_name}"
    }
end

Then(/^I should see "(.*?)" for the payment of "(.*?)"$/) do |text, dog_names|
  flag = false
  dog_names.gsub(/and/,',').split(',').each do |dog_name|
    Enrolment.where(dog_id: Dog.find_by_name("#{dog_name.strip}").id).each do |enrolment|
      if Enrolment.where(payment_id: enrolment.id).count  == dog_names.gsub(/and/,',').
                                                              split(',').count
        page.all('a', text: "#{text}").each do |link|
          flag = true if link[:href] == "/payments/#{enrolment.payment_id}?
                                        exhibition_id=#{enrolment.exhibition_id}"
        end
      end
    end
  end
  flag.should be true
end
```

Continúa en la página siguiente…

```
Then(/^I should see "(.*?)" button$/) do |name|
  find_button(name).should_not be_nil
end

Given(/^the exhibition "(.*?)" has already finished$/) do |exhibition|
  (Exhibition.find_by_name exhibition).end_date.should < Date.today
end

Then(/^the last deadline ended for "(.*?)"$/) do |exhibition|
  JSON.parse((Exhibition.find_by_name exhibition).tax)['deadlines'].last['end_date'].should
  < Date.today.strftime('%d/%m/%Y')
end

When(/^I try to "(.*?)" the enrolments for "(.*?)" in "(.*?)"$/) do |action, user, exhibition|
  case action
    when 'view'
      visit ("people/#{Person.find_by_name(user).id}/exhibitions/#
             {Exhibition.find_by_name(exhibition).id}/enrolments")
    when 'create'
      visit ("people/#{Person.find_by_name(user).id}/exhibitions/#
              {Exhibition.find_by_name(exhibition).id}/enrolments/new")
  end
end

When(/^I try to access to a user enrolments page "(.*?)" as admin$/) do |exhibition|
  visit ("/exhibitions/#{Exhibition.find_by_name(exhibition).id}/enrolments")
end

Given(/^there are sent erroneous payments data for "(.*?)"$/) do |exhibition|
  step "there are these payments for \"partners\" in \"#{exhibition}\":", table(%{
    | Class           | Dogs              | 1st entry deadline |
    | J. I. O. W. CH. | 1st dog           | 24.00              |
  })
end
```

Anexo IV – Estructura hash para Exposición Canina de Cieza

```
@tax = {
    :groups=>[
        {:name=>"group1", :classes=>["Junior", "Intermediate", "Open", "Working", "Champion"]},
        {:name=>"group2", :classes=>["Veteran"]},
        {:name=>"group3", :classes=>["Puppy"]},
        {:name=>"group4", :classes=>["Couple"]},
        {:name=>"group5", :classes=>["Group Breeding"]}
    ],
    :deadlines=>[
        {:name=>"1st entry deadline", :start_date=> '08/01/2014', :end_date=> '22/02/2014' },
        {:name=>"2nd entry deadline", :start_date=> '08/01/2014', :end_date=> '22/02/2014' },
        {:name=>"3rd entry deadline", :start_date=> '08/01/2014', :end_date=> '22/02/2014' }
    ],
    :prices=>[
        {:partners=>[
            {"group1"=>[["24.00", "19.00", "14.00"], ["30.00", "23.00", "18.00"], ["36.00", "27.00", "21.00"]]},
            {"group2"=>[["12.00", "9.50", "7.00"], ["15.00", "11.50", "9.00"], ["18.00", "13.50", "10.50"]]},
            {"group3"=>[["16.00", "12.00", "9.00"], ["20.00", "15.00", "12.00"], ["24.00", "18.00", "15.00"]]},
            {"group4"=>[["20.00"], ["25.00"], ["29.00"]]},
            {"group5"=>[["0.00"], ["0.00"], ["0.00"]]}
        ]},
        {:nopartners=>[
            {"group1"=>[["30.00", "25.00", "20.00"], ["37.50", "32.00", "25.00"], ["43.50", "37.00", "29.00"]]},
            {"group2"=>[["15.00", "12.50", "10.00"], ["18.70", "16.00", "12.50"], ["21.75", "18.50", "14.50"]]},
            {"group3"=>[["21.00", "15.00", "11.00"], ["27.00", "19.00", "14.00"], ["31.00", "21.00", "17.00"]]},
            {"group4"=>[["25.00"], ["32.00"], ["37.00"]]},
            {"group5"=>[["0.00"], ["0.00"], ["0.00"]]}
        ]}
    ]
}.to_json
```

Anexo V – Referencias de Internet

Último año de consulta: 2015

[Googl] Google. www.google.es

[Youtu] Youtube. www.youtube.es

[Wikip] Wikipedia. www.wikipedia.es

[Slide] Slideshare. www.slideshare.net

[Prezi] Prezi. www.prezi.com

Tutoriales Uml

- www.humbertocervantes.net
- http://users.dcc.uchile.cl/~psalinas/uml
- http://arodm.blogspot.com.es

Información Joomla, Wordpress, Drupal

- www.joomlaspanish.org
- www.openlabs.com.mx
- www.edujoomla.es
- www.baquia.com
- www.dlogica.com

Información Django, Python, CakePHP

- www.django.es
- http://book.cakephp.org
- www.coplec.org
- www.tufuncion.com

Información Cucumber, Rspec, MiniTests, Jasmine, Concordio

- www.methodsandtools.com
- http://arjanvandergaag.nl
- www.cheezyworld.com
- www.relishapp.com
- www.rubydoc.info/gems
- www.reddit.com/r/ruby
- http://docs.seattlerb.org/minitest
- www.rubyinside.com
- http://blog.crowdint.com
- http://jasmine.github.io
- http://concordion.org

Información Ruby on Rails

- http://rubytutorial.wikidot.com
- http://api.rubyonrails.org

- www.imaginanet.com
- www.rubysur.org/introduccion.a.rails
- www.rubisobrerieles.blogspot.com.es
- www.railsguides.net
- http://api.rubyonrails.org
- www.railscasts.com
- www.apidock.com/ruby
- www.ruby-doc.org
- www.xyzpub.com
- www.rubyjunky.com
- www.zittlau.ca
- www.easyactiverecord.com

Información Heroku, OpenShift, Windows Azure

- www.openshift.com
- www.azure.microsoft.com
- www.heroku.com
- http://blog.rodrigopuente.com

Información Desarrollo Ágil

- www.pmoinformatica.com
- www.pruebasdesoftware.com
- http://danieldejesustorres.blogspot.com.es
- www.pmoinformatica.com
- www.dosideas.com
- http://rbblog.foxandxss.net

Información Cloud Computing

- www.agpd.es
- www.imaginar.org
- www.proweblatam.com
- https://msdn.microsoft.com

Información exhibiciones y perros

- www.westminsterkennelclub.org
- www.ingrus.net
- www.perrospedia.com
- www.fci.be
- www.crufts.org.uk
- www.doglle.com

Maquetación Web

- http://bootstrapbay.com
- www.w3schools.com
- www.twbscolor.smarchal.com

- www.cssdeck.com
- www.plugolabs.com
- www.webdesign.about.com
- www.flaticon.es
- www.bootsnipp.com
- www.jsfiddle.net
- www.coffeescript.org
- www.librojquery.com

Consultas Varias

- www.genbetadev.com
- www.librosweb.es
- www.petermac.com