TRABAJO DE FIN DE GRADO

Gestión del ranking para una asociación canina de perros de exposición

Autora: **Déborah V. Alonso Hernández**

Tutores:
Carmelo Cuenca Hernández - Departamento de Informática y Sistemas Área de Arquitectura y Tecnología de Computadores
Francisca Quintana Domínguez - Departamento de Informática y Sistemas Área de Arquitectura y Tecnología de Computadores

Lugar de presentación: **Edificio de Informática y Matemáticas**

Fecha de presentación : **Enero de 2015**

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Índice general

Resumen

*[*ES] Esta aplicación web realizada en Ruby on Rails, tiene como objetivo principal la gestión del ranking para una asociación de perros de exposición. Dicha asociación es la "Asociación Española para el Fomento de la Raza Bulldog Francés".

En dicha asociación, los perros participan en exposiciones caninas y de acuerdo con una tabla y los resultados, les adjudican puntos a los perros. La tabla de asignación de puntos puede variar cada año. El perro que más puntos obtiene gana el ranking, hay un ganador hembra y otro ganador macho. Por otro parte, los posibles roles de usuarios son: el rol de administrador, los cuales pueden manejar todos los recursos; el rol de socio, que puede crear nuevos perfiles de perros, enviar resultados de sus perros y demás acciones posibles para este rol; el rol de usuarios registrado, que son usuarios que fueron socios y que ya no lo son. Dicho tipo de usuario puede modificar su perfil, pero ya no podría hacer cambios o eliminar sus perros de la aplicación. Y tampoco podría borrar resultados de sus perros que ya hubiesen sido enviados cuando era socio; y el rol de usuario no registrado que sólo pueden ver los listados de perros, ver sus perfiles y demás acciones que sólo sean ver pero no crear, modificar ni borrar nada

Como resultado final a este trabajo de fin de grado se ha obtenido una aplicación con los requisitos necesarios para cubrir las necesidades para la asociación de perros de exposición, en el cual se gestiona el ranking, también se gestionan los perfiles tanto de perros como de usuarios, control de accesos según el rol del usuario, envío de notificaciones, gestión de las exposiciones, etc.

*[*EN] The main goal of this web application made in Ruby on Rails is the management of the dog's ranking for an association of exhibition's dog. The association is the "Asociación Española para el Fomento de la Raza Bulldog Francés".

In this association, dogs participate in canine exhibitions and according to a table of points and the results, they award points to dogs. The points mapping table may vary every year. The dog that obtains more points, it wins the ranking. There is a female winner and a male winner. On the other hand, the possible roles of the users are: the administrator role, which can handle all resources; the cahoot role, which can create new dogs, send results of its dogs and others actions for this role; the user registered role, which are users that were cahoots but they are not any more. This kind of user can edit his profile, but they can't make any changes or eliminate his dogs of the application. Moreover, he can't delete his dogs' results, which were sent when he was a cahoot user; and finally, the non-registered user role, which can only see the ranking, the dogs list, the dogs' profiles and others actions that only imply reading information, but not creating, editing or deleting it.

As a final result in this project we have an application that fulfill the requirements of the association of exhibition's dog, in which different resources and managed: the ranking, both dogs and users profiles, access control according to the user's role, sending notifications, exhibitions, etc.

INTRODUCCIÓN

Este trabajo de fin de grado trata sobre la "Gestión del ranking para una asociación de perros de exposición". Comenzaremos introducciendo en qué consiste esta aplicación para a lo largo del informe ir explicando los pasos en los que se ha ido realizando la aplicación.

Los perros participan en exposiciones caninas y de acuerdo con una tabla y los resultados les adjudican puntos. La tabla de asignación de puntos varía cada año.

El perro que más puntos obtiene gana el ranking. De esos perros que más puntos obtienen, hay un ganador hembra y otro ganador macho.

La gestión del ranking consistirá en administrar las exposiciones, los perros, enviar resultados de los perros en las diversas exposiciones, etc.

Como resultado final a este trabajo de fin de grado se obtendrá una aplicación con los requisitos necesarios para cubrir las necesidades correspondientes a la gestión del ranking para la asociación de perros de exposición.

A lo largo de esta memoria se irán describiendo los pasos seguidos para capturar, analizar y describir los requisitos del sistema, además de ir describiendo en iteraciones el desarrollo del proyecto. Dicho proyecto será desarrollado en *Ruby on Rails*. Nos beneficiaremos de *Ruby* como lenguaje enfocado a la simplicidad y a la productividad, con una sintaxis elegante y natural. Y nos beneficiaremos de *Rails* como framework para el desarrollo de aplicaciones web, con *Rails* dispondremos de una serie de utilidades, de herramientas para crear aplicaciones web más rápidamente que haciéndolo desde cero. Usando dicho framework nos ahorramos mucho trabajo y podemos concentrarnos en lo que verdaderamente importa, es decir, concentrarnos en la lógica de la aplicación, y no en escribir una y otra vez los mismos formularios, scripts, etc.

Capítulo 1

ESTADO ACTUAL Y OBJETIVOS

En la actualidad, la "Asociación Española para el Fomento de la Raza Bulldog Francés" ya dispone de una aplicación web realizada en *PHP* , y lo que se persigue con este proyecto es que obtengan una aplicación web realizada en *Ruby on Rails*.

Dicha aplicación *PHP* funciona y es usable, pero se quiere a largo plazo sustituir el gestor de contenidos de *Joomla!* por una aplicación *Rails* . La aplicación obtenida al final de este TFG no podrá aún sustituir a la actual aplicación en *PHP* porque se necesitan más de 300 horas para completar la aplicación *Rails* para que pueda sustituir a dicha aplicación *PHP*.

Por lo que el objetivo principal que se persigue con este trabajo de fin de grado es conseguir una aplicación web para una asociación de perros de exposición, realizada en *Ruby on Rails* que permita la gestión de:

- Los perros. Con los que poder crear, editar o borrar perros según los privilegios. Además de poder ver el listado de perros de la aplicación y poder visualizar el perfil de cada perro.
- Los usuarios. Con los que poder crear, editar o eliminar usuarios de acuerdo a los privilegios correspondientes. Además, poder ver el listado de usuarios de la aplicación, ver sus perfiles y poder hacerlos socios o administradores o quitarles dichos roles.
- Las exposiciones caninas. Con las que siendo administrador poder crear, editar o eliminar exposiciones caninas. Además de poder ver el listado de exposiciones los usuarios de la aplicación .

- Resultados del ranking. Con el que poder visualizar el ranking de la asociación, con los puntos correspondientes a cada perro de acuerdo a los resultados validados hasta el momento de cada uno de los perros. Dicho ranking, es accesible a cualquier tipo de usuario, incluso a los usuarios que no están registrados.

Esta aplicación web a largo plazo sustituirá a la aplicación usada actualmente por dicha asociación y esa es la mayor motivación, el saber que el trabajo realizado será de provecho para una asociación real.

Capítulo 2

JUSTIFICACIÓN DE COMPETENCIAS ESPECÍFICAS CUBIERTAS

En este capítulo se justificará las competencias comunes a la Ingeniería informática cubiertas en este Trabajo de Fin de Grado.

- **CII01:** *Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.*

  En este trabajo de fin de grado se ha tenido previamente que realizar un diseño de la aplicación web, en el cual, se ha concluído entre otras cosas el desarrollo basado en pruebas para este proyecto, en el que cada iteración realizada conste de las siguientes partes:

  - Se escriben las pruebas
  - Se verifica que las pruebas fallan
  - Se implementa el código necesario
  - Se comprueba que las pruebas pasan satisfactoriamente
  - Se intenta refactorizar el código

- **CII02:** *Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social.*

  Antes de realizar el trabajo de fin de grado se ha planificado el trabajo, ya que de una buena planificación depende el que un proyecto salga adelante o no. Se ha desplegado

el entorno de trabajo para comenzar con lo que es la aplicación web, se ha puesto en marcha la aplicación y se ha ido mejorando conforme iban pasando las iteraciones durante la fase de desarrollo.

- **CII04:** *Capacidad para elaborar el pliego de condiciones técnicas de una instalación informática que cumpla los estándares y normativas vigentes.*

  El pliego de condiciones técnicas que se acuerda debe ser cumplido, en dicho acuerdo se tienen en cuenta duración del trabajo, requisitos técnicos, garantía y demás aspectos adicionales que se consideren oportunos.

- **CII18:** *Conocimiento de la normativa y la regulación de la informática en los ámbitos nacional, europeo e internacional.*

  En lo que se refiere a esta última competencia, se ha aplicado en mi trabajo de fin de grado en aspectos como es el tener en cuenta la Ley Orgánica de Protección de Datos de Carácter Personal (LOPD), que tiene como objeto garantizar y proteger, en lo referente al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas y en especial, su honor e integridad personal y familiar.

  Se ha cumplido la LOPD en mi TFG en aspectos como guardar en la base de datos las password de los usuarios encriptadas en md5, que usuarios que no estén registrados en la aplicación no puedan ver el correo de los usuarios registrados de la aplicación, etc.

  Además, en esta aplicación controlamos el que los datos estén disponibles, sean confidenciales y también controlamos su integridad. También, tenemos en cuenta cierto delitos informáticos, y por ello, controlamos el que sólo los usuarios autorizados accedan a determinada información y no sea toda la información accesible para todos lo usuarios. Por lo que controlamos que los usuarios no autorizados no puedan acceder o modificar información de manera ilícita intentando acceder directamente desde la url a recursos que no están disponibles para determinados usuarios ni introduciendo formularios con campos de los que no deben disponer algunos tipos de usuarios.

Capítulo 3

APORTACIONES

Una de las aportaciones que este Trabajo de Fin de Grado hace es a la "Asociación Española para el Fomento de la Raza Bulldog Francés", ya que dota a dicha asociación de una aplicación web con una interfaz bastante intuitiva, donde podrán manejar los socios de la asociación sus perfiles, sus perros y demás aspectos importantes. Además, tendrán esta aplicación y sin coste alguno, por lo que económicamente les es muy rentable.

Con la finalización de este proyecto no sustituirán directamente la aplicación *Rails* por la de *PHP* que hay actualmente porque con las 300 horas que implica el TFG no se podrá sustituir, pero ya se quedarán con un gran avance para luego ir mejorando y finalmente sustituir esa aplicación *PHP* que usan actualmente.

En cuanto a aportaciones técnicas a nuestro entorno, pues proporciona un ejemplo de uso de *Ruby on Rails* para la creación de aplicaciones web, una forma muy productiva de desarrollar aplicaciones web. En *Rails* , se pueden crear aplicaciones complejas sin que resulte todo tan complicado como con otros frameworks.

Y en cuánto a las aportaciones personales, me aporta una mayor formación en desarrollo web con *Rails* y experiencia conseguida, y es este el principal valor del proyecto. Además, he desarrollado una aplicación *Rails* que me servirá para demostrar mis conocimientos a la hora de salir al mercado laboral como graduada en ingeniería informática. No tiene la misma dificultad usar *Rails* que encajar piezas en un gestor de contenidos, encajar piezas en un gestor de contenidos es más sencillo. También me ha proporcionado este proyecto, el tener mayor experiencia elaborando la documentación del análisis y de los casos de uso.

Capítulo 4

CAPTURA, ANÁLISIS Y DESCRIPCIÓN DE LOS REQUISITOS DEL SISTEMA

4.1. Captura y Análisis de requisitos

Para lo referente a esta fase de captura y análisis de requisitos he usado Diagramas de casos de uso, mediante los cuales se muestra la relación entre los actores y los casos de uso en el sistema. Además, para la elicitación de requisitos también he realizado entrevistas directas a Carmelo Cuenca que es una de las personas pertenecientes a esta asociación de perros de exposición.

En las figuras 4.1, 4.2 y 4.3 se muestran los Diagramas de Casos de Uso resultantes. Los he dividido en 3 diagramas, para que esté más claro y no hayan muchos casos de usos liados usando un único diagrama, y para diferenciar entre los casos de uso que introducen información en el sistema, los casos de uso que son únicamente consultas de los usuarios sobre la información del sistema y los casos de uso que son de administración, es decir, que estos últimos son casos de uso que únicamente podrá ejecutar el administrador.

4.2. Descripción de requisitos

En esta sección, describiré cada uno de los casos de uso de los diagramas de casos uso de las figuras 4.1, 4.2 y 4.3.

En las tablas desde la 4.1 a la 4.7 se muestran las descripciones de los casos de uso referentes al Diagrama de Casos de Uso “Entrada de información en el sistema” (Figura 4.1).

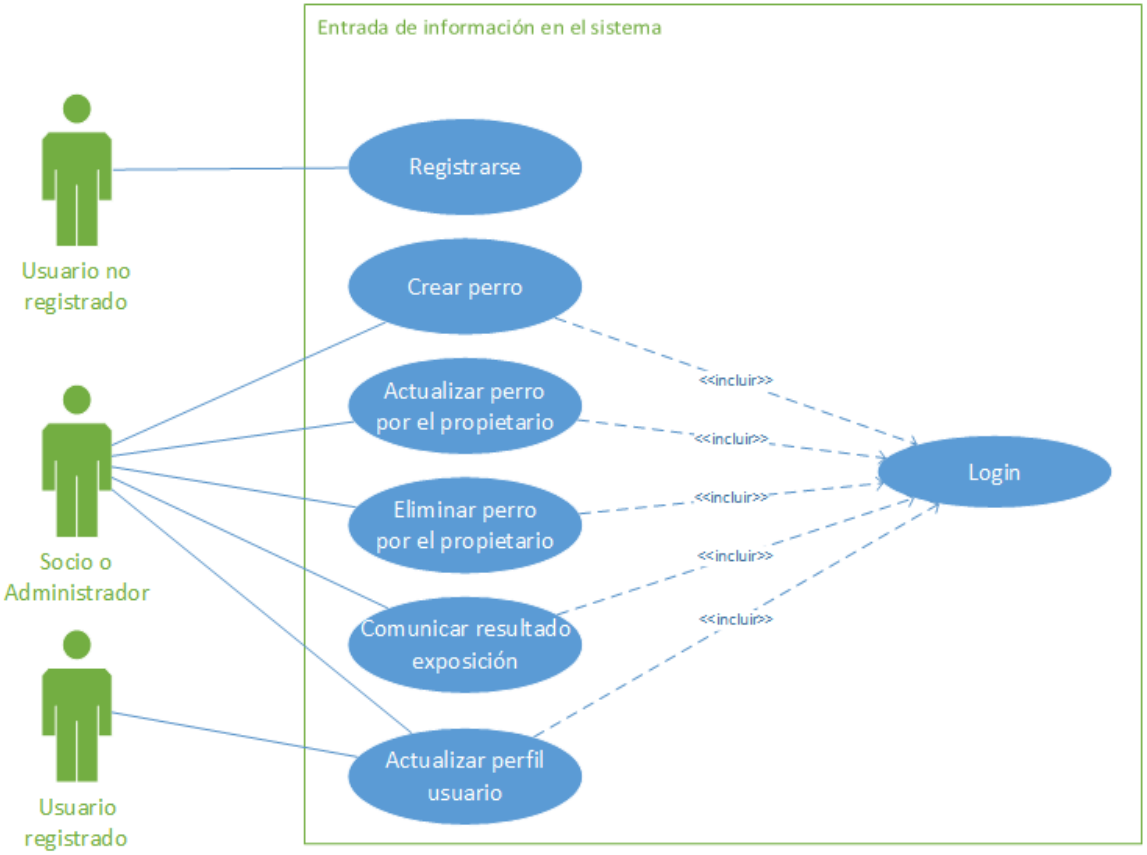

Figura 4.1: Diagrama Casos de Uso de “Entrada de información en el sistema”

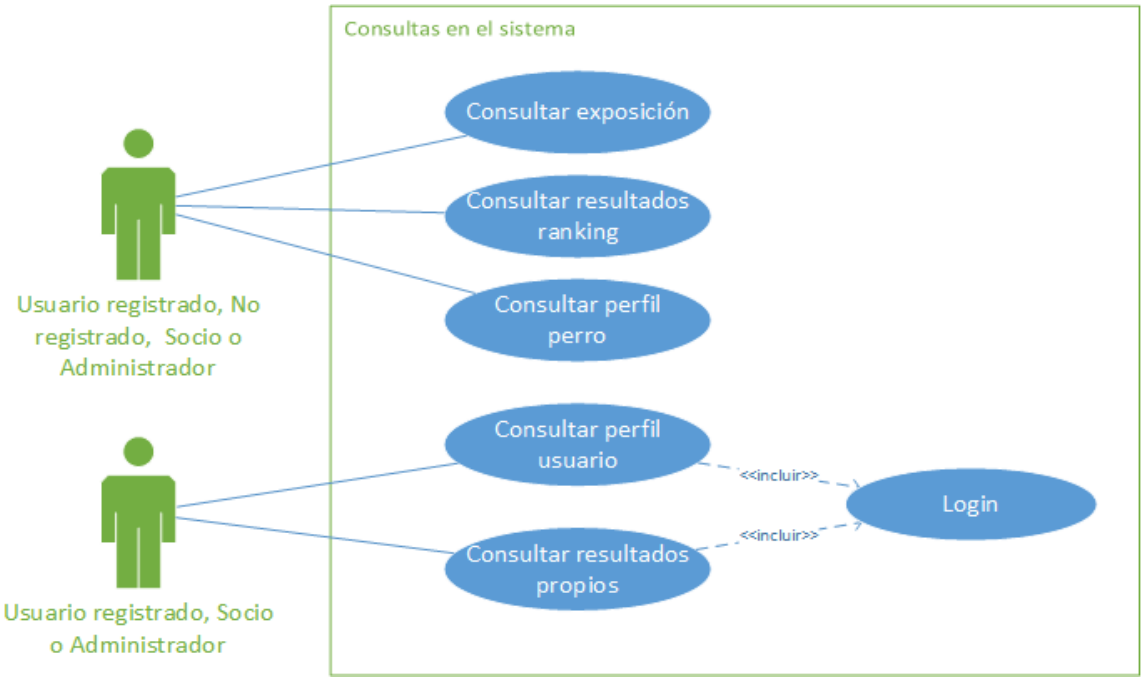

Figura 4.2: Diagrama Casos de Uso de “Consultas en el sistema”

A modo de ejemplo, explicaré la tabla 4.1. En dicha tabla podemos observar que primero especificamos el nombre del caso de uso, que es “Registrarse”. Después, especificamos por quién ha sido creada la descripción de este caso de uso. A continuación, se pone la fecha en la que se creó la descripción del caso de uso. Lo siguiente que se especifica es la descripción del caso de uso, es decir, se cuenta brevemente lo que el caso de uso hace. La descripción del fin de este caso de uso, es que un usuario no registrado se registra en el sistema. El siguiente elemento a especificar es el actor primario, es decir, se especifica quién inicia la comunicación con el caso de uso, que en este caso sería un usuario no registrado.

En las precondiciones especificamos las condiciones que se requieren sobre el estado del sistema antes de dar inicio al caso de uso, que como se observa en dicho caso de uso no hay precondiciones. Tenemos también las poscondiciones, que reflejan el estado en el que se queda el sistema una vez ejecutado el caso de uso, que en este caso de uso sería que un usuario nuevo se encuentra registrado en el sistema.

La siguiente parte a especificar en esta descripción del caso de uso sería el curso normal y alternativo. El curso normal representa el flujo exitoso más simple o habitual para el caso de uso. Y el curso alternativo comienza con el número del paso del escenario principal en el que la bifurcación se produce y la condición que da lugar a la bifurcación, y dicha condición debe ser una condición detectable por el sistema.

La última parte de la descripción del caso de uso serían las extensiones, en dicha parte se especifican los casos de uso que extienden del caso de uso que se esté describiendo.

En las tablas desde la 4.8 a la 4.12 se muestran las descripciones de los casos de uso referentes al Diagrama de Casos de Uso “Consultas en el sistema” (Figura 4.2).

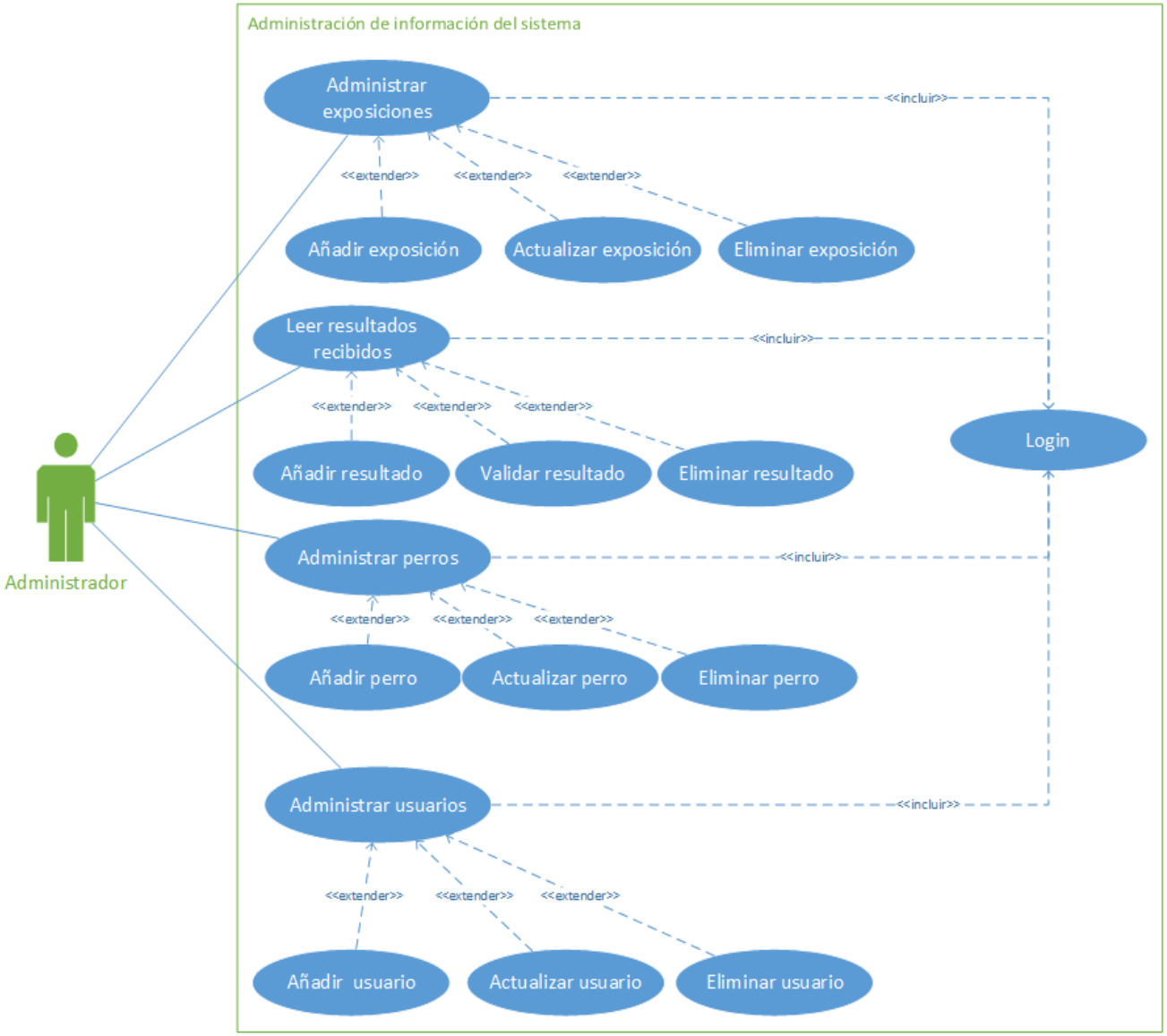

Figura 4.3: Diagrama Casos de Uso de “Administración de información en el sistema”

Nombre Caso de Uso: Registrarse	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Un usuario no registrado se registra en el sistema	
Actor primario: Usuario no registrado	
Precondiciones: -	
Poscondiciones: Un nuevo usuario se encuentra registrado en el sistema	
Curso Normal	**Curso Alternativo**
1. El usuario clicka en “Sign up” en el menú	
2. El usuario rellena los campos con sus datos	
3. El usuario clicka en el botón de “Sign up”	3.1. El usuario deja algún campo obligatorio vacío o con dato inválido y vuelve al paso 2
4. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.1: Caso de uso “Registrarse”

Nombre Caso de Uso: Crear perro	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Creación de un nuevo perro en el sistema cuyo owner es el usuario que lo crea	
Actor primario: Socio y Administrador	
Precondiciones: El usuario está logueado en el sistema	
Poscondiciones: En el sistema se encuentra un nuevo perro	
Curso Normal	**Curso Alternativo**
1. El usuario clicka en “Dogs” o en “My profile”	
2. El usuario clicka en “New Dog”	
3. El usuario rellena los campos con los datos del perro	
4. El usuario clicka en “Create Dog”	4.1. El usuario no rellena todos los campos obligatorios y vuelve al paso 3
5. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.2: Caso de uso “Crear perro”

Nombre Caso de Uso: Actualizar perro por el propietario	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Actualización de los datos de un perro del sistema cuyo propietario es el usuario actual	
Actor primario: Socio y Administrador	
Precondiciones: El usuario está logueado en el sistema y es propietario del perro a actualizar	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El usuario clicka en “Dogs” o en “My profile”	
2. El usuario clicka en “Edit” de uno de sus perros	
3. El usuario modifica los campos de los datos del perro	
4. El usuario clicka en “Update Dog”	4.1. El usuario deja algún campo obligatorio vacío y vuelve al paso 3
5. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.3: Caso de uso “Actualizar perro por el propietario”

Nombre Caso de Uso: Eliminar perro por el propietario	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Eliminación de un perro en el sistema cuyo propietario es el usuario actual	
Actor primario: Socio y Administrador	
Precondiciones: El socio está logueado en el sistema y es propietario del perro a eliminar	
Poscondiciones: En el sistema hay un perro menos y se han eliminado todos los resultados correspondientes de ese perro borrado	
Curso Normal	**Curso Alternativo**
1. El usuario clicka en “Dogs” o en “My profile”	
2. El usuario clicka en “Delete” de uno de sus perros	
3. El usuario confirma que quiere eliminar el ejemplar clickando en “Aceptar”	3.1. El usuario clica en “Cancelar” porque clickó por error en “Delete”
4. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.4: Caso de uso “Eliminar perro por el propietario”

Nombre Caso de Uso: Comunicar resultado exposición	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Un usuario comunica el resultado de una exposición al administrador	
Actor primario: Socio	
Precondiciones: El socio está logueado en el sistema	
Poscondiciones: Una nuevo resultado en el sistema para leer el administrador	
Curso Normal	**Curso Alternativo**
1. El usuario clicka en “My results”	
2. El usuario clicka en “New Result”	
3. El usuario rellena los campos correspondientes	
4. El usuario clicka en “Create Result”	4.1. El usuario deja algún campo obligatorio vacío y vuelve al paso 3
5. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.5: Caso de uso “Comunicar resultado exposición”

Nombre Caso de Uso: Actualizar perfil usuario	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Un usuario actualiza su perfil	
Actor primario: Socio, usuario registrado y administrador	
Precondiciones: El usuario está logueado en el sistema	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El usuario clicka en “My profile”	
2. El usuario clicka en “Edit User”	
3. El usuario modifica los campos correspondientes	
4. El usuario clicka en el botón de “Update User”	4.1. El usuario ha dejado vacío algún campo o lo ha rellenado erróneamente y vuelve al paso 3
5. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.6: Caso de uso “Actualizar perfil usuario”

Nombre Caso de Uso: Login	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Un usuario se loguea en el sistema	
Actor primario: Socio, usuario registrado y administrador	
Precondiciones: -	
Poscondiciones: Un nuevo usuario más logueado en el sistema	
Curso Normal	**Curso Alternativo**
1. El usuario clicka en "Sign in" en el menú	
2. El usuario introduce usuario y contraseña	
3. El usuario clicka en el botón de "Sign in"	3.1. El usuario ha introducido mal el usuario o la contraseña y vuelve al paso 2
4. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.7: Caso de uso "Login"

Nombre Caso de Uso: Consultar exposición	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Consultar una exposición del sistema	
Actor primario: Usuario registrado, Usuario no registrado, Socio y Administrador	
Precondiciones: -	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El usuario clicka en la imagen del menú de "aefrbf" o clicka en "Dogs"	
2. El usuario clicka en una de los perros	
3. El usuario clicka en una exposición concreta de las que aparecen en los resultados del perro que se está consultando	
4. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.8: Caso de uso "Consultar exposición"

Nombre Caso de Uso: Consultar resultados ranking	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Consultar los resultados del ranking del sistema	
Actor primario: Usuario registrado, Usuario no registrado, Socio y Administrador	
Precondiciones: -	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El usuario clicka en la imagen del menú de “aefrbf”	
2. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.9: Caso de uso “Consultar resultados ranking”

Nombre Caso de Uso: Consultar perfil perro	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Consulta del perfil de un perro del sistema	
Actor primario: Usuario no registrado, Usuario registrado, Socio y Administrador	
Precondiciones: -	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El usuario clicka en la imagen del menú de “aefrbf” o clicka en “Dogs”	
2. El usuario clicka en uno de los perros	
3. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.10: Caso de uso “Consultar perfil perro”

Nombre Caso de Uso: Consultar perfil usuario	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Consulta del perfil de un usuario del sistema	
Actor primario: Usuario registrado, Socio y Administrador	
Precondiciones: Usuario logueado en el sistema	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El usuario clicka en la imagen del menú de “aefrbf” o clicka en “Dogs”	
2. El usuario clicka en el email del owner de uno de los perros	
3. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.11: Caso de uso “Consultar perfil usuario”

Nombre Caso de Uso: Consultar resultados propios	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Consulta por parte de un usuario de sus resultados en el sistema	
Actor primario: Usuario registrado, Socio y Administrador	
Precondiciones: Usuario logueado en el sistema	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El usuario clicka en “My results”	
2. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.12: Caso de uso “Consultar resultados propios”

Nombre Caso de Uso: Administrar exposiciones	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: El administrador accede a la administración de las exposiciones del sistema	
Actor primario: Administrador	
Precondiciones: El administrador está logueado en el sistema	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El administrador clicka en “Exhibitions”	
2. Fin del caso de uso	
Extensiones	
1.A	El administrador clicka en “New Exhibition” (C.U. “Añadir exposición”)
1.B	El administrador clicka en “Edit” de uno de los perros (C.U. “Actualizar exposición”)
1.C	El administrador clicka en “Delete” de unos de los perros (C.U. “Eliminar exposición”)

Tabla 4.13: Caso de uso “Administrar exposiciones”

Nombre Caso de Uso: Añadir exposición	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: El administrado añade una nueva exposición en el sistema	
Actor primario: Administrador	
Precondiciones: El administrador está logueado en el sistema y clickó “New Exhibition”	
Poscondiciones: Hay una nueva exposición en el sistema	
Curso Normal	**Curso Alternativo**
1. El administrador rellena los campos correspondientes	
2. El administrador clicka en “Create Exhibition”	2.1. El administrador deja algún campo vacío o con datos inválidos y vuelve al paso 1
3. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.14: Caso de uso “Añadir exposición”

Y en las tablas desde la 4.13 a la 4.28 se muestran las descripciones de los casos de uso referentes al Diagrama de Casos de Uso “Administración de información en el sistema” (Figura 4.3).

Nombre Caso de Uso: Actualizar exposición	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Actualización de los datos de una exposición del sistema	
Actor primario: Administrador	
Precondiciones: El usuario está logueado en el sistema y ha clickado en "Edit" de uno de las exposiciones	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El administrador modifica los campos correspondientes de la exposición editada	
2. El usuario clicka en "Update Exhibition"	2.1. El administrador deja algún campo vacío o con datos inválidos y vuelve al paso 1
3. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.15: Caso de uso "Actualizar exposición"

Nombre Caso de Uso: Eliminar exposición	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Eliminación de una exposición del sistema	
Actor primario: Administrador	
Precondiciones: El administrador está logueado en el sistema y ha clickado en "Delete" de una de las exposiciones	
Poscondiciones: En el sistema hay una exposición menos y también se han eliminado todos los resultados correspondientes a esa exposición	
Curso Normal	**Curso Alternativo**
1. El administrador clicka en "Aceptar"	1.1. El administrador clicka en "Cancelar" porque clickó por error en "Delete"
2. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.16: Caso de uso "Eliminar exposición"

Nombre Caso de Uso: Leer resultados recibidos	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: El administrador lee los resultados recibidos por parte de los socios del sistema	
Actor primario: Administrador	
Precondiciones: El administrador está logueado en el sistema	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El administrador clicka en “Results”	
2. Fin del caso de uso	
Extensiones	
1.A	El administrador clicka en “New Result” (C.U. “Añadir resultado”)
1.B	El administrador clicka en “Edit” de uno de los resultados (C.U. “Validar resultado”)
1.C	El administrador clicka en “Delete” de uno de los resultados (C.U. “Eliminar resultado”)

Tabla 4.17: Caso de uso “Leer resultados recibidos”

Nombre Caso de Uso: Añadir resultado	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: El administrador añade un nuevo resultado enviado por un socio para actualizar el ranking	
Actor primario: Administrador	
Precondiciones: El administrador está logueado en el sistema y ha clickado en “New Result”	
Poscondiciones: Nuevo resultado en el sistema	
Curso Normal	**Curso Alternativo**
1. El administrador rellena los campos correspondientes	
2. El administrador clicka en “Create Result”	2.1. No rellena correctamente alguno de los campos o lo deja vacío y vuelve al paso 1
3. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.18: Caso de uso “Añadir resultado”

Nombre Caso de Uso: Validar resultado	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: El administrador valida un resultado enviado por un socio para actualizar el ranking	
Actor primario: Administrador	
Precondiciones: El administrador está logueado en el sistema y ha clickado en "Edit" de uno de los resultados	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El administrador modifica el campo "Status" por validated o rejected según verifique que ese resultado es cierto o no	
2. El administrador clicka en "Update Result"	
3. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.19: Caso de uso "Validar resultado"

Nombre Caso de Uso: Eliminar resultado	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: El administrador elimina un resultado enviado por un socio	
Actor primario: Administrador	
Precondiciones: El administrador está logueado en el sistema y ha clickado en "Delete" de uno de los resultados	
Poscondiciones: Un resultado menos en el sistema	
Curso Normal	**Curso Alternativo**
1. El administrador clicka en "Aceptar"	1.1. El administrador clicka en "Cancelar" porque clickó en "Delete" por error
2. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.20: Caso de uso "Eliminar resultado"

Nombre Caso de Uso: Administrar perros	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: El administrador accede a la administración de los perros del sistema	
Actor primario: Administrador	
Precondiciones: El administrador está logueado en el sistema	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El administrador clicka en “Dogs”	
2. Fin del caso de uso	
Extensiones	
1.A	El administrador clicka en “New Dog” (C.U. “Añadir perro”)
1.B	El administrador clicka en “Edit” de uno de los perros (C.U. “Actualizar perro”)
1.C	El administrador clicka en “Delete” de unos de los perros (C.U. “Eliminar perro”)

Tabla 4.21: Caso de uso “Administrar perros”

Nombre Caso de Uso: Añadir perro	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: El administrador añade un nuevo perro en el sistema	
Actor primario: Administrador	
Precondiciones: El administrador está logueado en el sistema y clickó “New Dog”	
Poscondiciones: Hay un nuevo perro en el sistema	
Curso Normal	**Curso Alternativo**
1. El administrador rellena los campos correspondientes	
2. El administrador clicka en “Create Dog”	2.1. El administrador deja algún campo vacío o con datos inválidos y vuelve al paso 1
3. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.22: Caso de uso “Añadir perro”

Nombre Caso de Uso: Actualizar perro	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Actualización de los datos de un perro del sistema	
Actor primario: Administrador	
Precondiciones: El usuario está logueado en el sistema y ha clickado en "Edit" de uno de los perros	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El administrador modifica los campos correspondientes del perro editado	
2. El usuario clicka en "Update Dog"	2.1. El administrador deja algún campo vacío o con datos inválidos y vuelve al paso 1
3. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.23: Caso de uso "Actualizar perro"

Nombre Caso de Uso: Eliminar perro	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Eliminación de un perro del sistema	
Actor primario: Administrador	
Precondiciones: El administrador está logueado en el sistema y ha clickado en "Delete" de uno de los perros	
Poscondiciones: En el sistema hay un perro menos y se han eliminado todos los resultados correspondientes de ese perro borrado	
Curso Normal	**Curso Alternativo**
1. El administrador clicka en "Aceptar"	1.1. El administrador clicka en "Cancelar" porque clickó por error en "Delete"
2. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.24: Caso de uso "Eliminar perro"

Nombre Caso de Uso: Administrar usuarios	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: El administrador accede a la administración de los usuarios del sistema	
Actor primario: Administrador	
Precondiciones: El administrador está logueado en el sistema	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El administrador clicka en “Users”	
2. Fin del caso de uso	
Extensiones	
1.A	El administrador clicka “New User” (C.U. “Añadir usuario”)
1.B	El administrador clicka en “Edit” (C.U. “Actualizar usuario”)
1.C	El administrador clicka en “Delete” (C.U. “Eliminar usuario”)

Tabla 4.25: Caso de uso “Administrar usuarios”

Nombre Caso de Uso: Añadir usuario	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Consulta del listado de todos los usuarios del sistema	
Actor primario: Administrador	
Precondiciones: El administrador está logueado en el sistema y clickó en “New User”	
Poscondiciones: Hay un nuevo usuario en el sistema	
Curso Normal	**Curso Alternativo**
1. El administrador rellena los campos correspondientes	
2. El administrador clicka en “Create User”	2.1. El administrador deja algún campo vacío o con datos inválidos y vuelve al paso 1
3. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.26: Caso de uso “Añadir usuario”

Nombre Caso de Uso: Actualizar usuario	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Actualización de los datos de un usuario del sistema	
Actor primario: Administrador	
Precondiciones: El usuario está logueado en el sistema y ha clickado en "Edit" de uno de los usuarios	
Poscondiciones: -	
Curso Normal	**Curso Alternativo**
1. El administrador modifica los campos correspondientes del usuario editado	
2. El usuario clicka en "Update User"	2.1. El administrador deja algún campo vacío o con datos inválidos y vuelve al paso 1
3. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.27: Caso de uso "Actualizar usuario"

Nombre Caso de Uso: Eliminar usuario	
Creado por: Déborah Alonso Hernández	
Fecha de Creación: 08/05/14	
Descripción: Eliminación de un usuario del sistema	
Actor primario: Administrador	
Precondiciones: El administrador está logueado en el sistema y ha clickado en "Delete" de uno de los usuarios	
Poscondiciones: En el sistema hay un usuario menos.	
Curso Normal	**Curso Alternativo**
1. El administrador clicka en "Aceptar"	1.1. El administrador clicka en "Cancelar" porque clickó por error en "Delete"
2. Fin del caso de uso	
Extensiones	
-	-

Tabla 4.28: Caso de uso "Eliminar usuario"

Capítulo 5

DISEÑO

En este capítulo explicaremos todo lo referente a la parte de diseño de este Trabajo de Fin de Grado.

En lo referente a la arquitectura de la aplicación desarrollada en este TFG es RESTFUL. REST define una serie de reglas que toda aplicación REST debe cumplir. Dichas reglas son las siguientes:

- **Arquitectura cliente-servidor:** consiste en una separación clara y concisa entre los 2 agentes básicos en un intercambio de información: el cliente y el servidor. Estos 2 agentes deben ser independientes entre sí, lo que permite una flexibilidad muy alta en todos los sentidos.

- **Stateless:** esto significa que nuestro servidor no tiene por qué almacenar datos del cliente para mantener un estado del mismo. Como sabemos, HTTP también cumple esta norma, por lo que estamos acostumbrados ya a hacer uso de protocolos stateless.

- **Cacheable:** esta norma implica que el servidor que sirve las peticiones del cliente debe definir algún modo de cachear dichas peticiones, para aumentar el rendimiento, escalabilidad, etc. HTTP implementa esto con la cabecera "Cache-control", que dispone de varios parámetros para controlar la cacheabilidad de las respuestas.

- **Sistema por capas:** nuestro sistema no debe forzar al cliente a saber por qué capas se tramita la información, lo que permite que el cliente conserve su independencia con respecto a dichas capas.

- **Interfaz uniforme:** esta regla simplifica el protocolo y aumenta la escabilidad y rendimiento del sistema. Esta regla nos garantiza que no importa quién haga las peticiones ni quién las reciba, siempre y cuando ambos cumplan una interfaz definida de antemano.

Otro aspecto relevante respecto al diseño de esta aplicación, es que además sigue el patrón de arquitectura Modelo-Vista-Controlador (MVC). Dicho patrón separa la lógica de la aplicación de la interfaz de usuario de manera que facilita la evolución por separado de ambos aspectos e incrementa la reutilización y la flexibilidad. Los elementos básicos de dicho patrón son:

- **Modelo:** representa la parte de la aplicación que implementa la lógica de la aplicación. En aplicaciones web sería la información almacenada en una base de datos o en XML junto con las reglas de la aplicación que transforman esa información.
- **Vista:** hace una presentación de los datos del modelo estando separada de los objetos del modelo. Es responsable del uso de la información de la cual dispone para producir cualquier interfaz de presentación de cualquier petición que se presente. En aplicaciones web sería la página HTML.
- **Controlador:** gestiona las peticiones de los usuarios. Es responsable de responder la información solicitada con la ayuda tanto del modelo como de la vista. En aplicaciones web sería el código que obtiene datos dinámicamente y genera el contenido HTML.

Y por último, referente al diseño de esta aplicación, me gustaría comentar las 3 capas en las que está estructurada. Dichas capas son:

- **Servidor de aplicaciones:** Tipo de servidor que permite el procesamiento de datos de una aplicación de cliente. Las principales ventajas de la tecnología de los servidores de aplicación es la centralización y la disminución de la complejidad del desarrollo de aplicaciones.
- **Servidor web:** este tipo de servidor procesa una aplicación del lado del servidor, realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente. El código recibido por el cliente suele ser compilado y ejecutado por un navegador web. Para la transmisión de todos estos datos suele utilizarse el protocolo HTTP.
- **Servidor de base de datos:** se utiliza para almacenar, recuperar y administrar los datos de una base de datos. El servidor gestiona las actualizaciones de datos, permite el acceso simultáneo de muchos servidores o usuarios web y garantiza la seguridad y la integridad de los datos.

Capítulo 6

DESARROLLO

En esta aplicación se ha optado por un desarrollo basado en pruebas (TDD – Test Driven Development), dicha práctica involucra otras dos prácticas que son:

- Escribir las pruebas primero
- Refactorización

En el desarrollo basado por pruebas generalmente se utilizan las pruebas unitarias, en las cuales se escribe primero una prueba, se verifica que la prueba falla, se implementa el código necesario para que pase satisfactoriamente la prueba y se refactoriza el código escrito. El objetivo que se persigue con el desarrollo basado en pruebas es conseguir un código limpio y que funcione.

Por otro lado, la aplicación será desarrollada en *Ruby on Rails*. *Ruby* es un lenguaje enfocado a la simplicidad y a la productividad, con una sintaxis elegante y natural. Y *Rails* es un buen framework para el desarrollo de aplicaciones web, con *Rails* dispondremos de una serie de utilidades, de herramientas para crear aplicaciones web más rápidamente que haciéndolo desde cero. Usando dicho framework nos ahorramos mucho trabajo y podemos concentrarnos en lo que verdaderamente importa , es decir, concentrarnos en la lógica de la aplicación, y no en escribir una y otra vez los mismos formularios, scripts, etc.

El desarrollo de la aplicación se realizará en 11 iteraciones, en cada una de esas iteraciones se realizarán las pruebas. Luego se implementará el código correspondiente. A

continuación, se verificará que las pruebas funcionen con el código implementado. Y finalmente se refactorizará el código en el caso de que sea posible.

Las iteraciones que seguirá el desarrollo de este trabajo de fin de grado son las siguientes:

- **Iteración 1:** Se realizarán los tests *Cucumber* básicos y posteriormente se ejecutarán para comprobar que fallan. Después, se realizará el modelado básico de perros y al final de la iteración se ejecután los tests para comprobar que se ha realizado correctamente el código de acuerdo a lo previsto para dicha iteración.

- **Iteración 2:** Se realizarán los tests *Cucumber* básicos necesarios para esta iteración y se ejecutarán para comprobar que fallan. Luego, se llevará a cabo el modelado básico de usuarios. Habrá un sólo tipo de usuario que puede hacerlo todo. Y finalmente, se ejecutarán los tests *Cucumber* para comprobar que se ha desarrollado correctamente todo el código.

- **Iteración 3:** Realizaremos los tests *Cucumber* y *RSpec* necesarios para esta iteración y se ejecutarán para comprobar que fallan. Después, modelaremos la relación usuario-perro. En esta iteración se completará el modelo del perro con su propietario y se introducirán las autorizaciones. Además, se distinguirá entre usuarios logueados y no logueados, de los cuales los usuarios logueados podrán hacer todo y los usuarios no logueados no podrán hacer nada, salvo leer. Y para acabar la iteración, ejecutaremos de nuevo los tests *Cucumber* y *RSpec* creados al comienzo de esta iteración.

- **Iteración 4:** Como en cada iteración, crearemos primero los tests necesarios, se modificará de los tests de iteraciones anteriores lo que sea necesario y se ejecutarán los tests para comprobar que fallan. En esta iteración se controlará que los usuarios logueados puedan manipular sólo a sus perros, es decir, que puedan editar y borrar sólo sus perros. Y finalmente, ejecutaremos los tests *Cucumber* y *RSpec* para comprobar que el código funciona de la manera esperada.

- **Iteración 5:** Primero, crearemos los tests necesarios para esta iteración, haremos la modificaciones necesarias a los tests realizados en las iteraciones anteriores y ejecutaremos los tests para comprobar que fallan. A continuación, se completará el modelo de usuarios con el usuario administrador. Se añadirán autorizaciones para el usuario administrador, para que el administrador entre otras cosas, pueda borrar los perros y borrar usuarios, cuyo matiz es que borrar un usuario implica poner el campo propietario del perro a nulo. Y finalmente, se ejecutarán los tests *Cucumber* y *RSpec* para comprobar que todo funciona correctamente.

- **Iteración 6:** Se realizarán los tests correspondientes a esta sexta iteración y se ejecutarán para comprobar que fallan. Después, se realizará el modelado de las exposiciones con su nombre, su fecha y el tipo de exposición que es. Además, se añadirán las autorizaciones correspondientes para que dichas exposiciones sólo las pueda gestionar el usuario administrador. Y por último, se ejecutarán los tests *Cucumber* y *RSpec* para comprobar que todo ha sido desarollado de manera correcta.

- **Iteración 7:** Comenzaremos realizando los tests para esta iteración y se ejecutarán para comprobar que fallan. Realizaremos las tablas correspondientes a los premios, a los tipos de exposición y la de puntos. La tabla de puntos será introducida directamente en la base de datos en el administrador, en vez de poder realizarlo a través de

vistas, para simplificar, ya que con todo lo planteado para el proyecto se excedía de las 300 horas que implica la realización del TFG . En la tabla de puntos, para cada Año-Tipo de exposición-Premio se tienen unos puntos determinados. Se realizará el modelado de la tabla de resultados, en la cual, para cada resultado de un perro que haya ganado un premio de un tipo de exposición determinado, tendrá un estado pendiente, validado o rechazado por el administrador. Sólo los usuarios logueados podrán mandar resultados de sus perros. El administrador validará o rechazará cada resultado. Además, los usuarios logueados podrán borrar sus propios resultados que estén en estado pendiente o rechazado. Sin embargo, el administrador podrá borrar incluso los resultados que están validados. Y finalmente, ejecutaremos los tests *Cucumber* y *RSpec* de nuevo para comprobar que todo va bien.

- **Iteración 8:** Primero, realizaremos los test necesarios para esta iteración y se ejecutarán para comprobar que fallan. A continuación, se realizará una modificación en el modelo de usuarios y en el modelo de perros, en el cual se añadirá un campo para almacenar la imagen de un usuario o de un perro respectivamente. No será obligatorio el que tengan una foto, por lo que dicho campo podrá estar vacío. Se hará uso de la gema *CarrierWave* para subir las imágenes a la aplicación. Y por último, se ejecutarán los tests.

- **Iteración 9:** Crearemos los tests requeridos para esta iteración y los ejecutaremos para comprobar que fallan. A continuación, se completará el modelo de los usuarios introduciendo los usuarios socios. Sólo los usuarios socios podrán crear perros, enviar resultados, etc. Los usuarios registrados son usuarios socios que han dejado de serlo, sus datos se conservarán en la aplicación pero a nivel de funcionalidad tiene los mismo privilegios que un usuario no registrado. Finalmente, se ejecutarán los test *Cucumber* y *RSpec* y así comprobaremos el que se haya desarrollado todo de manera correcta.

- **Iteración 10:** En esta iteración comenzaremos realizando unos tests sencillos sobres las vistas y los ejecutaremos para comprobar que fallan. En esta iteración, se introducirá Bootstrap en la aplicación y se ejecutará al final de la iteración los tests desarrollados sobre las vistas para comprobar que el estilo sea el que se quiere.

- **Iteración 11:** Y en esta última iteración, crearemos los tests necesarios, los ejecutaremos para comprobar que fallan, ocultaremos todos los links que no deben ver algunos tipos de usuarios. Y por último, ejecutaremos de nuevo los tests para comprobar que todo funciona de acuerdo a lo esperado.

En cada una de las siguientes secciones, se irá explicando cada iteración con más detalle resaltando las partes más importantes de cada una de dichas iteraciones. Pero antes de comenzar con las iteraciones del trabajo de fin de grado, se ha realizado la configuración de la aplicación y el despliegue en *Heroku* , todo ello se encuentra desarrollado en el Anexo A y el Anexo B respectivamente.

6.1. Iteración 1 - Gestión de los perros

En esta iteración, tenemos como objetivo principal realizar el modelado básico de los perros, en el cual crearemos el modelo “Dog”. Los perros de momento tendrán su nombre, su sexo, su fecha de nacimiento y los títulos que tenga el perro. Además, en esta iteración, hacemos frente a todo lo refente a la manipulación de perros, ya sea mostrar, crear, editar

o eliminar perros. Todo ese control de acciones sobre los perros los especificamos a través del controlador y posteriormente especificando las vistas correspondientes.

6.1.1. Tests

La primera parte de la iteración es la realización de los tests *Cucumber* . A continuación, mostraremos el test *Cucumber* "creating dogs", ya que es el más representativo. El "editing dogs" es muy parecido al "creating dogs", y el "viewing dogs" y el "deleting dogs" no revisten de mayor complicación. De todas formas todos los tests se pueden consultar en el anexo del código entregado en el CD.

```
Feature: Creating dogs
  In order to create a dog
  I want to create them easily
  Background:
    Given there are the following dogs:
      | Sex  | Titles | Name                    | Birth Date |
      | male |        | Samba y Fatiga Idilio   | 27/01/2006 |

    And I am on the dogs page
    When I follow "New Dog"


  Scenario: Creating a dog
    And I select "male" from "Sex"
    And I fill in "Titles" with "CH OF SPAIN"
    And I fill in "Name" with "Frael Valderrama"
    And I fill in "Birth date" with "01/01/2001"
    And I press "Create Dog"
    Then I should see "Dog has been created."
    And I should be on the dog page for "Frael Valderrama"

  Scenario: Creating a dog without sex
    And I fill in "Titles" with "CH OF SPAIN"
    And I fill in "Name" with "Frael Valderrama"
    And I fill in "Birth date" with "01/01/2001"
    And I press "Create Dog"
    Then I should see "Sex is not included in the list"

  Scenario: Creating a dog without name
    And I select "male" from "Sex"
    And I fill in "Titles" with "CH OF SPAIN"
    And I fill in "Birth date" with "01/01/2001"
    And I press "Create Dog"
    Then I should see "Name can't be blank"

  Scenario: Creating a dog without birth date
    And I select "male" from "Sex"
    And I fill in "Titles" with "CH OF SPAIN"
```

```
    And I fill in "Name" with "Frael Valderrama"
    And I press "Create Dog"
    Then I should see "Birth date can't be blank"

  Scenario: Creating a dog with birth date in the future
    And I select "male" from "Sex"
    And I fill in "Titles" with "CH OF SPAIN"
    And I fill in "Birth date" with "1/1/2016"
    And I fill in "Name" with "Frael Valderrama"
    And I press "Create Dog"
    Then I should see "Birth date can not be in the future"

  Scenario: Creating a dog with an invalid birth date format
    And I select "male" from "Sex"
    And I fill in "Titles" with "CH OF SPAIN"
    And I fill in "Birth date" with "1/15/2002"
    And I fill in "Name" with "Frael Valderrama"
    And I press "Create Dog"
    Then I should see "Birth date must be a valid date"
```

Como podemos observar en el "creating dogs", tenemos 6 escenarios definidos. En el primer escenario comprobamos que rellenando todos los campos se crea correctamente el nuevo perro mostrándonos el mensaje "Dog has been created".

En el segundo escenario comprobamos que si se deja vacío el "Sex", entonces el perro no se crea, ya que dicho campo es un campo obligatorio, por lo que se nos muestra el mensaje de error "Sex is not included in the list".

En el tercer escenario comprobamos que si se deja vacío el campo "Name", entonces el perro no se crea, ya que dicho campo es también obligatorio, por lo que se nos muestra el mensaje de error "Name can't be blank".

En el cuarto escenario comprobamos que si se deja vacío el campo "Birth date", entonces el perro no se crea, ya que dicha campo es también obligatorio, por lo que se nos muestra el mensaje de error "Birth date can't be blank".

En el quinto escenario comprobamos que si se rellena el campo "Birth date" con una fecha futura, entonces el perro no se crea, ya que como es lógico la fecha de nacimiento del perro debe ser una fecha del pasado y se nos muestra el mensaje de error "Birth date can not be in the future".

Y en el último escenario comprobamos que si se añade una fecha en un formato incorrecto, entonces el perro no se crea, y se muestra el mensaje de error "Birth date must be a valid date".

6.1.2. Desarrollo del código

En esta primera iteración, una de las partes más importantes es crear primero el modelo de los perros. En dicho modelo especificamos la lógica necesaria para no permitir al crear un perro que estén los campos de nombre, sexo ni fecha de nacimiento en blanco. Además de controlar que la fecha de nacimiento del perro esté en el formato correcto y no esté en el futuro. A continuación, mostramos como hicimos ese control de los campos en el modelo:

```
class Dog < ActiveRecord::Base
  validates_presence_of :name, :birth_date
```

```
  validate :future_birth_date
  SEX = ['male', 'female']
  validates_inclusion_of :sex, in: SEX

  validates_format_of :birth_date,
        with:  /[[19|20]+[0-9]+[0-9]]+-[0-9|1[0-2]]+-[[0-2+0-9]|3[0-1]]/,
                 message: 'Birth date must be a valid date'
  def future_birth_date
    if !birth_date.blank? and birth_date > Date.today
      errors.add(:birth_date, "can not be in the future")
    end
  end

end
```

Las otras partes importantes de esta iteración, serían el control de las operaciones CRUD sobre el perro a través del controlador y la creación de las vistas para poder acceder a las páginas de crear, editar, ver perros, etc.

En esta primera iteración, en lo referente al código no hubo grandes problemas. Fue más complicada la realización de los tests al comienzo de la iteración que la codificación de esta primera iteración, ya que yo nunca había realizado tests y tenía que aprender primero cómo hacer los tests *Cucumber* .

6.1.3. Reejecución de los tests

Una vez desarrollado el código necesario para la iteración, reejecutamos los tests *Cucumber* que creamos al principio de la iteración y vemos que todos los resultados nos salen en verde, es decir, que se ha codificado todo correctamente, de manera que funciona todo tal y como se esperaba.

6.2. Iteración 2 - Creación de los usuarios

En esta segunda iteración hacemos frente al modelado básico de los usuarios, para que los usuarios puedan realizar acciones básicas de “sign up”, “sign in” y “sign out”. En esta iteración, obtendremos un sólo tipo de usuario que puede hacerlo todo. Para la autenticación de usuarios haremos uso de la gema *devise*. Cada usuario tendrá su email, su contraseña y demás campos proporcionados por la gema *devise*.

6.2.1. Tests

En esta parte, mostraremos un test de los varios tests necesarios que se crean en el comienzo de esta iteración que servirán para verificar al final de la iteración que se ha desarrollado la gestión de los usuarios de manera adecuada. Mostraré a modo de ejemplo, el test *Cucumber* “signing in” realizado para comprobar que se realizada correctamente el inicio de sesión de los usuarios en la aplicación.

```
Feature: Signing in
  In order to use the site
```

```
As a user
I want to be able to sign in


Scenario: No signing in via confirmation
  Given there are the following users:
    | email            | password | unconfirmed |
    | user@example.com | password | true        |
   When "user@example.com" opens the email with subject "Confirmation instructions"
   And they click the first link in the email
   Then I should see "Your account was successfully confirmed"
   Then I should not see "Signed in as user@example.com"

@working
Scenario: Signing in via form
  Given there are the following users:
    | email            | password |
    | user@example.com | password |
  And I am signed in as them
  Then I should see "Signed in successfully"
  And I should see "Signed in as user@example.com"
```

Como podemos observar en el "signing in", el test consta de 2 escenarios, es un test bastante sencillo. En el primer escenario comprobamos que se inicia sesión correctamente al usuario clickar en el primer enlace que aparece en el email de confirmación, se confirma correctamente la cuenta, aparece el mensaje "Your account was successfully confirmed" y se loguea automáticamente apareciendo también el mensaje "Signed in as user@example.com".

Y en el segundo escenario, comprobamos que el usuario inicia sesión correctamente introduciendo los datos en el formulario de Sign in de la aplicación. Una vez rellena el formulario el usuario y lo envía, se inicia sesión y aparecen los mensajes "Signed in successfully" y "Signed in as user@example.com".

6.2.2. Desarrollo del código

A la hora de codificar en esta iteración, lo principal ha sido entender bien lo que hace la gema "devise" para luego poder instalarla y poder usarla de manera correcta y aprovechando todo lo posible lo que nos proporciona dicha gema. *Devise* gestiona la autenticación a todos los niveles. La arquitectura de *Devise* es modular y consta de once módulos cada uno de los cuales cubre un aspecto diferente de la autenticación. Por ejemplo, el módulo *Rememberable* recuerda la autenticación del usuario en una cookie mientras que otro módulo, *Recoverable*, se ocupa de reiniciar la clave del usuario enviando instrucciones por correo. Este enfoque hace que sea más fácil escoger exactamente las funcionalidades de autenticación que queramos utilizar.

En esta iteración, creamos el modelo de usuario, en el cual especificamos lo siguiente:

```
class User < ActiveRecord::Base
  has_many  :dogs
```

```
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable, :confirmable
end
```

Con lo especificado en el modelo usuario en la parte de "devise" cada uno de los elementos que le preceden significa lo siguiente:

- database authenticatable: cifra y almacena una contraseña en la base de datos para validar la autenticidad de un usuario mientras inicia sesión. La autenticación se puede realizar a través de las peticiones POST o autenticación básica HTTP.
- registerable: permite el manejo del registro de usuarios a través de un proceso de registro y además, les permite editar y borrar su cuenta.
- recoverable: se ocupa de reiniciar la clave del usuario enviando instrucciones por correo.
- rememberable: recuerda la autenticación del usuario en una cookie.
- trackable: podemos ver cuándo inició sesión un usuario, el número de veces que ha iniciado sesión y la dirección IP.
- validatable: proporciona validaciones de correo electrónico y contraseña.
- confirmable: envía mensajes de correo electrónico con instrucciones de confirmación y verifica si una cuenta ya está confirmado durante el logueo.

6.2.3. Reejecución de los tests

Una vez desarrollado el código necesario para la segunda iteración, reejecutamos los tests *Cucumber* que creamos al principio de la iteración y vemos que todos los resultados nos salen en verde, es decir, que se ha codificado todo correctamente, de manera que funciona todo tal y como se esperaba. En el caso de que alguno de los test salga en rojo habría que volver a modificar el código para luego volver a ejecutar los tests y comprobar que todo funciona de manera correcta para poder finalizar la iteración.

6.3. Iteración 3 - Gestión de permisos de usuarios sobre los perros

En esta iteración nos encargaremos del modelado de la relación usuario-perro, en el cual cada usuario podrá tener muchos perros y aparecerá para cada perro quién es su propietario. Además, diferenciaremos entre los usuarios logueados y los usuarios no logueados. Los usuarios logueados podrán hacer todo sobre todos los perros y los usuarios no logueados no podrán hacer nada, salvo leer.

6.3.1. Tests

Hasta esta iteración, únicamente hemos realizado tests *Cucumber* , pero en esta iteración introduciremos con los tests *RSpec*. Los test *Cucumber* los usaremos para controlar que funcione todo correctamente desde el punto de vista del usuario que entra en la aplicación y navega a través de los links por la aplicación, y con los test *RSpec* controlaremos

que usuarios sin privilegios accedan a través de la url a secciones de la aplicación que no pueden acceder y controlaremos que no nos intenten enviar formularios con campos que un usuario sin privilegios no pueda enviar.

En esta parte mostraremos los test más significativos realizados en esta iteración para verificar al final de dicha iteración que se ha desarrollado todo de manera adecuada conforme a los cambios que se realizan en esta iteración sobre la aplicación. Mostraremos el test *Cucumber* "creating dogs" para ver que se hemos añadido lo del propietario del perro, los otros 3 test *Cucumber* de los perros no los pondremos, ya que el "editing dogs" es muy similar al "creating dogs", y los test "viewing dogs" y "deleting dogs" son sencillos. Y de todas formas, todos los tests se pueden consultar en el anexo del código entregado en el CD.

```
Feature: Creating dogs
  In order to create a dog
  As a user
  I want to create them easily
  Background:
    Given there are the following dogs:
      | Sex  | Titles | Name                    | Birth Date | Owner              |
      | male |        | Samba y Fatiga Idilio   | 27/01/2006 | user1@example.com |

    Given there are the following users:
      | email            | password |
      | user@example.com | password |

    And I am signed in as them
    And I am on the dogs page
    When I follow "New Dog"

  @working
  Scenario: Creating a dog
    And I select "male" from "Sex"
    And I fill in "Titles" with "CH OF SPAIN"
    And I fill in "Name" with "Frael Valderrama"
    And I click in "Birth date"
    And I fill in "datepicker" with "01/01/2001"
    And I press "Create Dog"
    Then I should see "Dog has been created."
    And  I should see "Owned by user@example.com"
    And I should be on the dog page for "Frael Valderrama"

  Scenario: Creating a dog without sex
    And I fill in "Titles" with "CH OF SPAIN"
    And I fill in "Name" with "Frael Valderrama"
    And I click in "Birth date"
    And I fill in "datepicker" with "01/01/2001"
    And I press "Create Dog"
    Then I should see "Sex is not included in the list"
```

```
Scenario: Creating a dog without name
  And I select "male" from "Sex"
  And I fill in "Titles" with "CH OF SPAIN"
  And I click in "Birth date"
  And I fill in "datepicker" with "01/01/2001"
  And I press "Create Dog"
  Then I should see "Name can't be blank"

Scenario: Creating a dog without birth date
  And I select "male" from "Sex"
  And I fill in "Titles" with "CH OF SPAIN"
  And I fill in "Name" with "Frael Valderrama"
  And I press "Create Dog"
  Then I should see "Birth date can't be blank"

Scenario: Creating a dog with birth date in the future
  And I select "male" from "Sex"
  And I fill in "Titles" with "CH OF SPAIN"
  And I click in "Birth date"
  And I fill in "datepicker" with "01/01/2016"
  And I fill in "Name" with "Frael Valderrama"
  And I press "Create Dog"
  Then I should see "Birth date can not be in the future"

Scenario: Creating a dog with an invalid birth date format
  And I select "male" from "Sex"
  And I fill in "Titles" with "CH OF SPAIN"
  And I click in "Birth date"
  And I fill in "datepicker" with "30/02/2002"
  And I fill in "Name" with "Frael Valderrama"
  And I press "Create Dog"
  Then I should see "Birth date must be a valid date"
```

Como podemos observar el test "creating dogs" consta de 6 escenarios. En el primer escenario comprobamos que rellenando todos los campos se crea correctamente el nuevo perro mostrándonos el mensaje "Dog has been created". Y se muestra el mensaje más importante de este test en esta iteración que es "Owned by user@example.com", el objetivo principal de esta iteración era encargarnos de relacionar los modelos usuario y perro y así poder poner para cada perro quién es su propietario.

En el segundo escenario comprobamos que si se deja vacío el "Sex", entonces el perro no se crea, ya que dicho campo es un campo obligatorio, por lo que se nos muestra el mensaje de error "Sex is not included in the list".

En el tercer escenario comprobamos que si se deja vacío el campo "Name", entonces el perro no se crea, ya que dicho campo es también obligatorio, por lo que se nos muestra el mensaje de error "Name can't be blank".

En el cuarto escenario comprobamos que si se deja vacío el campo "Birth date", entonces el perro no se crea, ya que dicha campo es también obligatorio, por lo que se nos muestra el mensaje de error "Birth date can't be blank".

En el quinto escenario comprobamos que si se rellena el campo “Birth date” con una fecha futura, entonces el perro no se crea, ya que como es lógico la fecha de nacimiento del perro debe ser una fecha del pasado y se nos muestra el mensaje de error “Birth date can not be in the future”.

Y en el último escenario comprobamos que si se añade una fecha en un formato incorrecto, entonces el perro no se crea, y se muestra el mensaje de error “Birth date must be a valid date”.

Y de los tests *RSpec* pondremos el test sobre el controlador de los perros, en el cual veremos cómo controlamos las acciones que no puede realizar un usuario no logueado.

```
require 'spec_helper'

describe DogsController do

  let(:user) { FactoryGirl.create(:user) }

  context 'Guest users (not logged-in users)' do
    let(:valid_attributes) { {name: 'Samba y Fatiga Idilio', sex: 'male',
                              birth_date: '01/01/2001', user_id: user.id}  }

    before { @dog =  FactoryGirl.create(:dog) }

    it 'cannot access to the new action' do
      get :new
      response.should redirect_to(new_user_session_path)
    end

    it 'cannot access to the create action' do
      expect { post :create, dog: valid_attributes }.not_to change(Dog, :count)
      response.should redirect_to(new_user_session_path)
    end

    it 'cannot access to the edit action' do
      get :edit, id: @dog.to_param
      response.should redirect_to(new_user_session_path)
    end

    it 'cannot access to the update action' do
      put :update, {id: @dog.to_param, dog: valid_attributes}
      response.should redirect_to(new_user_session_path)
    end

    it 'cannot access to the delete action' do
      expect { delete :destroy, id: @dog.to_param }.not_to change(Dog, :count)
      response.should redirect_to(new_user_session_path)
    end
  end
end
```

En el test “dogs controller spec” podemos observar cómo comprobamos que un usuario no logueado no puede acceder a las acciones de “new”, “create”, “edit”, “update” ni “delete” sobre ninguno de los perros. En todas esas acciones que un usuario no logueado no puede realizar se le redirecciona a la página de Sign in, para que el usuario inicie sesión y así poder acceder a esas acciones deseadas.

6.3.2. Desarrollo del código

A la hora de codificar en esta iteración, lo principal ha sido entender bien lo que hace la gema *cancan* para luego poder instalarla y poder usarla de manera correcta y aprovechando todo lo posible lo que nos proporciona dicha gema. Dicha gema nos proporciona un manera más sencilla de autorización para Ruby on Rails que restringe a qué recursos se permite acceder a un usuario dado. Todos los permisos se definen en un solo lugar (la clase Ability) y no duplicados en los controladores, vistas y consultas de bases de datos.

En principio, en esta iteración se nos quedará la clase Ability sencilla, y ya se van añadiendo en las siguientes iteraciones las correspondientes autorizaciones. El fichero Ability en esta iteración nos queda de la siguiente manera:

```
class Ability
  include CanCan::Ability

  def initialize(user)

    user ||= User.new # guest user (not logged in)

    cannot :manage, :all

    if user.persisted?
      can [:new, :create], Dog
      can [:edit, :update, :destroy], Dog

    end
  end
end
```

Además, en esta iteración modificamos el controlador de los perros para controlar que cuando un usuario crea un perro, en el campo owner del perro se guarde el usuario que creo a dicho perro. Para ello, tuvimos que previamente que crear el campo owner en la tabla de los perros y hacer una migración.

6.3.3. Reejecución de los tests

Para comprobar que la iteración se ha realiza correctamente, testeamos todo lo realizado en dicha iteración ejecutando todos los tests obtenidos al principio de esta iteración. Una vez todo esos tests se ejecutan sin fallos, ya se puede pasar a la siguiente iteración.

6.4. Iteración 4 - Gestión de permisos de los usuarios sobre sus propios perros

En esta iteración se abordará todo lo que se refiere a controlar lo que puede realizar cada usuario sobre sus perros. Es decir, se controlará principalmente el que cada usuario logueado sólo pueda realizar acciones de editar o borrar sobre sus propios perros.

A diferencia de la iteración anterior, aquí controlaremos el que no todos los usuarios logueados puedan realizar las acciones CRUD sobre todos los perros, sino que únicamente puedan realizar dichas acciones sobre sus propios perros. Lo que si que podrán hacer todos los usuarios tanto logueados como no logueados es leer la información de los perros y ver el listado de perros del sistema.

6.4.1. Tests

En esta parte, mostraremos el test *Cucumber* "editing dogs" para ver cómo controlamos lo de que sólo pueda editar un perro su propietario. Sólo mostraremos dicho test de los tests *Cucumber* , ya que es el más representativo y complejo. El test "creating dogs" también es complejo, pero como es similar al "editing dogs", y en las iteraciones anteriores hemos mostrado el "creating dogs", en esta iteración mostraré el otro. De todas formas todos los tests realizados se encuentran en el anexo del código entregado en el CD.

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

  Background:
    Given there are the following users:
      | email             | password |
      | user1@example.com  | password |
      | user2@example.com  | password |

    And there are the following dogs:
      | Sex  | Titles | Name                    | Birth Date | Owner             |
      | male |        | Samba y Fatiga Idilio   | 27/01/2006 | user1@example.com |
      | male |        | Fortunato Hautacuperche | 15/05/2008 | user2@example.com |

    And I am signed in as "user1@example.com"
    And I am on the dogs page


  Scenario: Updating a dog with property
    When I follow "Edit" within "#dog_1"
    And I fill in "Name" with "Samba y Fatiga Idilio beta"
    And I press "Update Dog"
    Then I should see "Dog has been updated."
    Then I should be on the dog page for "Samba y Fatiga Idilio beta"

  @working
```

```
  Scenario: Updating a dog without property
    When I follow "Edit" within "#dog_2"
    Then I should see "You are not authorized to access this page"

  Scenario: Updating a dog without name
    When I follow "Edit" within "#dog_1"
    And I fill in "Name" with " "
    And I press "Update Dog"
    Then I should see "Name can't be blank"

  Scenario: Updating a dog without birth date
    When I follow "Edit" within "#dog_1"
    And I click in "Birth date"
    And I fill in "datepicker" with ""
    And I press "Update Dog"
    Then I should see "Birth date can't be blank"

  Scenario: Updating a dog with birth date in the future
    When I follow "Edit" within "#dog_1"
    And I click in "Birth date"
    And I fill in "datepicker" with "1/1/2016"
    And I press "Update Dog"
    Then I should see "Birth date can not be in the future"
```

Como se puede observar en el test de "editing dogs", dicho test consta de 5 escenarios. En el primer escenario se comprueba que se actualiza correctamente el perro cuando el usuario que lo intenta editar es propiedario del perro que quiere editar. Mostrando al final de la actualización del perro el mensaje "Dog has been updated".

En el segundo escenario se comprueba que no se puede editar un perro cuando el usuario que intenta editarlo no es propietario de dicho perro. Por lo que se muestra el mensaje "You are not authorized to access this page".

En el tercer escenario se comprueba que si al editar su perro un usuario, deja el campo "Name" vacío, no se actualizará el perro y saldrá el mensaje de error "Name can't be blank", ya que dicho campo es obligatorio.

En el cuarto escenario se comprueba que si al editar su perro un usuario, deja el campo "Birth date" vacío, no se actualizará el perro y se mostrará el mensaje de error "Birth date can't be blank", ya que dicho campo es obligatorio.

Y en el quinto y último escenario de este test, se comprueba que no se actualizará el perro si se introduce una fecha del futuro y aparecerá el mensaje "Birth date can not be in the future".

En cuanto a los test *RSpec*, mostraremos el test sobre el controlador de los perros que está más completo en esta iteración.

```
require 'spec_helper'

describe DogsController do

  let(:user) { FactoryGirl.create(:user) }

  let(:valid_attributes) { {name: 'Samba y Fatiga Idilio', sex: 'male',
```

```
                        birth_date: '01/01/2001', user_id: user.id}  }

context 'Guest users (not logged-in users)' do

  before { @dog =  FactoryGirl.create(:dog) }

  it 'cannot access to the new action' do
    get :new
    response.should redirect_to(new_user_session_path)
  end

  it 'cannot access to the create action' do
    expect { post :create, dog: valid_attributes }.not_to change(Dog, :count)
    response.should redirect_to(new_user_session_path)
  end

  it 'cannot access to the edit action' do
    get :edit, id: @dog.to_param
    response.should redirect_to(new_user_session_path)
  end

  it 'cannot access to the update action' do
    put :update, {id: @dog.to_param, dog: valid_attributes}
    response.should redirect_to(new_user_session_path)
  end

  it 'cannot access to the delete action' do
    expect { delete :destroy, id: @dog.to_param }.not_to change(Dog, :count)
    response.should redirect_to(new_user_session_path)
  end
end

context 'Non-creator users (logged-in users)' do

  let(:hacker) { FactoryGirl.create(:user) }


  before do
    @dog = FactoryGirl.create(:dog)
    sign_in(:user, hacker)
  end

  it 'cannot access to the edit action' do
    get :edit, id: @dog.to_param
    response.status.should == 403
    response.should render_template(file: "#{Rails.root}/public/403.html")
  end

  it 'cannot access to the update action' do
```

```
      put :update, {id: @dog.to_param, dog: valid_attributes}
      response.status.should == 403
      response.should render_template(file: "#{Rails.root}/public/403.html")
    end

    it 'cannot access to the delete action' do
      expect { delete :destroy, id: @dog.to_param }.not_to change(Dog, :count)
      response.status.should == 403
      response.should render_template(file: "#{Rails.root}/public/403.html")
    end

  end
end
```

La parte de los usuarios no logueados es lo mismo que explicamos en la iteración 3, lo nuevo es la parte de los usuarios no creadores del perro sobre el que se quiere realizar alguna acción. En esa segunda parte se puede observar que se comprueba que los usuarios no propietarios del perro al que se accede no pueden editar, actualizar ni borrar dicho perro.

6.4.2. Desarrollo del código

A la hora de codificar en esta iteración, la clave ha estado en modificar de manera correcta el fichero Ability para que sólo lo el propietario de cada perro sea el que pueda realizar las acciones de "edit", "update" y "delete" sobre su perro. A continuación podemos observar lo añadido en el fichero Ability:

```
class Ability
  include CanCan::Ability

  def initialize(user)
    user ||= User.new # guest user (not logged in)

    cannot :manage, :all

    if user.persisted?
      can [:new, :create], Dog
      can [:edit, :update, :destroy], Dog, ['dogs.user_id = ?', user.id] do |dog|
        dog.user_id == user.id
      end
    end

  end
end
```

En dicho fichero podemos observar como comprobamos que cuando se va a realizar la acción de "edit", "update" o "delete" sobre un perro, comprobamos que el propietario de dicho perro coincide con el usuario actual que está logueado e intentando realizar alguna de dichas 3 acciones sobre el perro.

6.4.3. Reejecución de los tests

Para comprobar que la iteración se realiza correctamente se testea todo lo que se va realizando en dicha iteración, ejecutamos todos los tests obtenidos al principio de la iteración. Comprobamos que todos los tests pasan correctamente, por lo que la aplicación al final de la iteración funciona tal y como se esperaba.

6.5. Iteración 5 - Gestión de permisos con los usuarios administradores

En esta iteración, abordaremos todo lo que se refiere a los usuarios administradores del sistema. Es decir, en esta iteración se añadirá el tipo de usuario administrador, y dicho tipo de usuario podrá realizar las operaciones CRUD tanto sobre los perros como sobre los usuarios de la aplicación. Lo único que no podrá realizar es borrarse a si mismo. Y además, controlaremos que al borrar un usuario no implica borrar sus perros, por lo que los perfiles de los perros permanecerán en la aplicación aunque su dueño se haga dado de baja en la aplicación.

6.5.1. Tests

En esta sección mostraremos el test “creating users”, ya que en dicho test se ven los escenarios más importantes, es de los tests más representativos de esta iteración. Todos los demás tests se pueden consultar en el anexo del código entregado en el CD. A continuación, mostraremos el test “creating users” y explicaremos.

```
Feature: Creating Users
  In order to create a user
  As an admin
  I want to create them easily

  Background:
    Given there are the following users:
      | email              | password | admin |
      | user1@example.com  | password | false |
      | user2@example.com  | password | false |
      | admin@example.com  | password | true  |

    And I am signed in as "admin@example.com"
    And I am on the users page


  Scenario: Creating a user
    When I follow "New User"
    And I fill in "Email" with "user3@example.com"
    And I fill in "Password" with "password"
    And I fill in "Password confirmation" with "password"
    And I press "Create User"
    Then I should see "User has been created"
    And I should see "user3@example.com (User)"
```

```
Scenario: Creating an admin user
  When I follow "New User"
  When I fill in "Email" with "admin2@example.com"
  And I fill in "Password" with "password"
  And I fill in "Password confirmation" with "password"
  And I check "Is an admin?"
  And I press "Create User"
  Then I should see "User has been created."
  And I should see "admin2@example.com (Admin)"

Scenario: Creating a user without password
  When I follow "New User"
  And I fill in "Email" with "user3@example.com"
  And I press "Create User"
  Then I should see "User has not been created"
  And I should see "Password can't be blank"

Scenario: Creating a user without email
  When I follow "New User"
  And I fill in "Password" with "password"
  And I fill in "Password confirmation" with "password"
  And I press "Create User"
  Then I should see "User has not been created"
  And I should see "Email can't be blank"

Scenario: Creating a user with a different password and confirmation password
  When I follow "New User"
  And I fill in "Email" with "user3@example.com"
  And I fill in "Password" with "password"
  And I fill in "Password confirmation" with "pasword"
  And I press "Create User"
  Then I should see "User has not been created"
  And I should see "Password confirmation doesn't match Password"
```

Como se puede observar, el test de "creating users" contiene 5 escenarios. El primer escenario comprueba que un usuario administrador puede crear un usuario normal correctamente rellenando los campos correspondientes sin marcar la casilla de que el nuevo usuario sea administrador, cuyo mensaje final es "User has been created".

En el segundo escenario se comprueba que un usuario administrador puede crear otro usuario administrador rellenando todos los campos correspondientes y marcando la casilla de "Is an admin?", cuyo mensaje final es "User has been created".

En el tercer escenario se comprueba que no se crea el usuario si el administrador deja vacío el campo "Password" del nuevo usuario, cuyos mensajes de error serán "User has not been created" y "Password can't be blank".

En el cuarto escenario se comprueba que no se crea el usuario si el administrador deja vacío el campo "Email" del nuevo usuario, cuyos mensajes de error serán "User has not been created" e "Email can't be blank".

Y en el quinto y último escenario, se comprueba que no se crea el usuario si el administrador introduce la contraseña introducida no es igual en los campos "Password" y

"Password confirmation", apareciendo los mensajes de error "User has been created" y "Password confimation doesn't match Password".

En lo referente a los test *RSpec* sobre los usuarios, mostraremos los tests sobre los usuarios administradores (admin users controller spec), ya que son los más importantes de esta iteración cuyo objetivo principal era añadir a el rol de usuario administrador.

```
require 'spec_helper'

describe Admin::UsersController do

  let(:user){ FactoryGirl.create(:user) }
  valid_attributes = {email: 'user2@example.com', password: "password",
                      password_confirmation: "password"}
  let(:admin){FactoryGirl.create(:admin)}

  context "admin user actions on himself" do
    it "can access the edit action" do
      sign_in(:user, admin)
      get :edit, id: admin.to_param
      response.should be_success
    end

    it "can access the update action" do
      sign_in(:user, admin)
      put :update, {id: admin.to_param, user: valid_attributes}
      response.should redirect_to(admin_user_path)
      flash[:notice].should eql("User has been updated.")
    end

    it "can access the show action" do
      sign_in(:user, admin)
      get :show, id: admin.to_param
      response.should be_success
    end

    it "cannot access the destroy action" do
      sign_in(:user, admin)
      delete :destroy, id: admin.to_param
      response.should redirect_to(admin_users_path)
      flash[:alert].should eql("You cannot delete yourself")
    end
  end

  context "admin user actions on other users" do
    it "can access the index action" do
      sign_in(:user, admin)
      get :index
      response.should be_success
    end
```

```
    it "can access the new action" do
      sign_in(:user, admin)
      get :new
      response.should be_success
    end

    it "can access the create action" do
      sign_in(:user, admin)
      post :create, user: valid_attributes
      response.should redirect_to(admin_users_path)
      flash[:notice].should eql("User has been created.")
    end

    it "can access the edit action" do
      sign_in(:user, admin)
      get :edit, id: user.to_param
      response.should be_success
    end

    it "can access the update action" do
      sign_in(:user, admin)
      put :update, {id: user.to_param, user: valid_attributes}
      response.should redirect_to(admin_user_path)
      flash[:notice].should eql("User has been updated.")
    end

    it "can access the show action" do
      sign_in(:user, admin)
      get :show, id: user.to_param
      response.should be_success
    end

    it "cann access the destroy action" do
      sign_in(:user, admin)
      delete :destroy, id: user.to_param
      response.should redirect_to(admin_users_path)
      flash[:notice].should eql("User has been deleted.")
    end
  end

end
```

En la primera parte del test de “users controller spec” que se refiere a las acciones que puede realizar un usuario administrador sobre sí mismo, podemos observar que se comprueba que el usuario administrador puede realizar cualquier acción sobre sí mismo, excepto eliminarse. Si dicho usuario se intenta eliminar le aparece el mensaje de error de “You cannot delete yourself”.

En la segunda parte de dicho test, que se refiere a las acciones que puede realizar un usuario administrador sobre los demás usuarios del sistema, se comprueba que los usuarios administradores pueden realizar cualquier tipo de acción sobre los otros usuarios del sistema.

6.5.2. Desarrollo del código

En la codificación de esta iteración, comentaremos las partes más relevantes. Lo primero que hacemos es crear un nuevo espacio de nombre del sitio llamado admin, con el propósito de separar un controlador de la zona principal del sitio y así poder garantizar que los usuarios que accedan a este controlador en particular tienen el campo admin a true.

Se empieza por la generación de un controlador de espacio de nombres con una acción index vacío mediante el siguiente comando:

```
$ rails g controller admin/users index
```

Después, ya se van realizando las demás acciones a parte del index. Luego, añadimos el campo admin en la tabla de los usuarios, que ponemos por defecto a false.

Y también configuramos en el fichero "routes.rb" para los recursos del espacio de nombre "admin" como se muestra a continuación:

```
RankingApp::Application.routes.draw do
  get "users/index"
  devise_for :users
  root 'dogs#index'

  resources :dogs
  resources :users

  namespace :admin do
    resources :users
  end

end
```

6.5.3. Reejecución de los tests

Como en cada iteración realizada, la última parte es testear que todo funciona como tiene que funcionar. Para ello, se testea todo lo realizado, ejecutando los tests obtenidos al principio de la iteración y si ninguno de esos tests falla queda comprobado que todo funciona bien y que la iteración ha finalizado.

6.6. Iteración 6 - Gestión de exposiciones

En esta iteración se realizará el modelado de las exposiciones, de las cuales se especificará su nombre, su fecha y el tipo de exposición que es. Dichas exposiciones sólo podrán ser gestionadas por los usuarios de tipo administrador.

Todos los demás usuarios lo único que podrán hacer sobre las exposiciones es ver el listado de las exposiciones y ver la información de cada una de las exposiciones.

6.6.1. Tests

De los test *Cucumber* de esta iteración, mostraremos el "editing exhibitions", ya que en todas las iteraciones el "creating" y el "editing" han sido los test más importantes y similares entre ellos, ya sea para los perros, para los usuarios o para las exposiciones.

```
Feature: Editing Exhibitions
  In order to update exhibition information
  As an admin
  I want to be able to do that through an interface

  Background:
    Given there are the following exhibitions:
      | name         | date       | type                  |
      | exhibition 1 | 27/03/2015 | MONOGRÁFICA NACIONAL  |
      | exhibition 2 | 15/07/2015 | Punto Obligatorio     |

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

    And I am on the dogs page

  Scenario: Updating a exhibition like an admin
    When I am signed in as "admin@example.com"
    And I follow "Exhibitions"
    And I follow "Edit" within "#exhibition_1"
    And I fill in "Name" with "exhibition beta 1"
    And I press "Update Exhibition"
    Then I should see "Exhibition has been updated."
    And I should see "exhibition beta 1"
    And I should not see "exhibition 1"

  Scenario: Updating a exhibition like an other user
    When I am signed in as "user@example.com"
    And I follow "Exhibitions"
    And I follow "Edit" within "#exhibition_1"
    Then I should see "You must be an admin to do that"

  Scenario: Updating a exhibition without name
    When I am signed in as "admin@example.com"
    And I follow "Exhibitions"
    And I follow "Edit" within "#exhibition_1"
    And I fill in "Name" with ""
    And I press "Update Exhibition"
    Then I should see "Exhibition has not been updated."
    And I should see "Name can't be blank"

  Scenario: Updating a exhibition without date
```

```
    When I am signed in as "admin@example.com"
    And I follow "Exhibitions"
    And I follow "Edit" within "#exhibition_1"
    And I click in "Date"
    And I fill in "datepicker" with ""
    And I press "Update Exhibition"
    Then I should see "Exhibition has not been updated."
    And I should see "Date can't be blank"

  Scenario: Updating a exhibition with date at the past
    And I am signed in as "admin@example.com"
    And I follow "Exhibitions"
    When I follow "Edit" within "#exhibition_1"
    And I click in "Date"
    And I fill in "datepicker" with "15/05/2004"
    And I press "Update Exhibition"
    Then I should see "Exhibition has not been updated."
    Then I should see "Date can not be at the past"
```

Como se puede observar en el test de "editing exhibitions", dicho test consta de 5 escenarios. En el primer escenario se comprueba que un usuario administrador puede actualizar una exposición rellenando los campos correspondientes de dicha exposición. El mensaje final que le aparecería sería "Exhibition has been updated".

En el segundo escenario se comprueba que no se puede editar una exposición como otro tipo de usuario que no sea administrador, ya que al intentar editar una exposición apacería el mensaje de error "You must be an admin to do that".

En el tercer escenario se comprueba que no se actualiza la exposición si el usuario administrador deja el campo "Name" vacío, ya que aparecerían los mensajes de error "Exhibition has not been updated" y "Name can't be blank".

En el cuarto escenario se comprueba que no se actualiza la exposición si el usuario administrador deja el campo "Date" vacío, ya que aparecerían los mensajes de error "Exhibition has not been updated" y "Date can't be blank".

Y en el quinto escenario se comprueba que no se pueda añadir una exposición con fecha en el pasado, de tal forma que se mostrarían los mensajes de error "Exhibition has not been updated" y "Date can not be at the past".

Y de los test *RSpec*, mostraremos el nuevo test creado para las exposiciones, ya que es el test *RSpec* más importante por el objetivo que se persigue en esta iteración.

```
require 'spec_helper'

describe ExhibitionsController do

  let(:user) { FactoryGirl.create(:user) }

  let(:valid_attributes) { {name: 'Exhibition 1', date: '28/05/2015',
                            type: 'MONOGRÁFICA NACIONAL'}  }

  context 'Guest users about exhibitions' do

    before { @exhibition =  FactoryGirl.create(:exhibition) }
```

```
    it 'cannot access to the new action' do
      get :new
      response.should redirect_to(new_user_session_path)
    end

    it 'cannot access to the create action' do
      expect {post :create, exhibition: valid_attributes}.not_to change(Exhibition,
                                                                        :count)
      response.should redirect_to(new_user_session_path)
    end

    it 'cannot access to the edit action' do
      get :edit, id: @exhibition.to_param
      response.should redirect_to(new_user_session_path)
    end

    it 'cannot access to the update action' do
      put :update, {id: @exhibition.to_param, exhibition: valid_attributes}
      response.should redirect_to(new_user_session_path)
    end

    it 'cannot access to the delete action' do
      expect {delete :destroy, id: @exhibition.to_param}.not_to change(Exhibition,
                                                                      :count)
      response.should redirect_to(new_user_session_path)
    end

    it 'cannot access to the show action' do
      get :show, id: @exhibition.to_param
      response.should be_success
    end

    it 'cannot access to the index action' do
      get :index
      response.should be_success
    end

  end

  context 'Standard users about exhibitions' do

    before { @exhibition =  FactoryGirl.create(:exhibition) }

    it 'cannot access to the new action' do
      sign_in(:user, user)
      get :new
      response.should redirect_to(root_path)
    end
```

```
  it 'cannot access to the create action' do
    sign_in(:user, user)
    expect {post :create, exhibition: valid_attributes}.not_to change(Exhibition,
                                                                      :count)
    response.should redirect_to(root_path)
  end

  it 'cannot access to the edit action' do
    sign_in(:user, user)
    get :edit, id: @exhibition.to_param
    response.should redirect_to(root_path)
  end

  it 'cannot access to the update action' do
    sign_in(:user, user)
    put :update, {id: @exhibition.to_param, exhibition: valid_attributes}
    response.should redirect_to(root_path)
  end

  it 'cannot access to the delete action' do
    sign_in(:user, user)
    expect { delete :destroy, id: @exhibition.to_param }.not_to change(Exhibition,
                                                                      :count)
    response.should redirect_to(root_path)
  end

  it 'cannot access to the show action' do
    sign_in(:user, user)
    get :show, id: @exhibition.to_param
    response.should be_success
  end

  it 'cannot access to the index action' do
    sign_in(:user, user)
    get :index
    response.should be_success
  end

end


context 'Admin users about exhibitions' do

  before { @exhibition =  FactoryGirl.create(:exhibition) }
  let(:admin){FactoryGirl.create(:admin)}

  it "can access the index action" do
    sign_in(:user, admin)
```

```
      get :index
      response.should be_success
    end

    it "can access the new action" do
      sign_in(:user, admin)
      get :new
      response.should be_success
    end

    it "can access the create action" do
      sign_in(:user, admin)
      post :create, exhibition: valid_attributes
      response.should redirect_to(Exhibition.find_by_name('Exhibition 1'))
      flash[:notice].should eql("Exhibition has been created.")
    end

    it "can access the edit action" do
      sign_in(:user, admin)
      get :edit, id: @exhibition.to_param
      response.should be_success
    end

    it "can access the update action" do
      sign_in(:user, admin)
      put :update, {id: @exhibition.to_param, exhibition: valid_attributes}
      response.should redirect_to(exhibition_path(@exhibition))
      flash[:notice].should eql("Exhibition has been updated.")
    end

    it "can access the show action" do
      sign_in(:user, admin)
      get :show, id: @exhibition.to_param
      response.should be_success
    end

    it "cann access the destroy action" do
      sign_in(:user, admin)
      delete :destroy, id: @exhibition.to_param
      response.should redirect_to(exhibitions_path)
      flash[:notice].should eql("Exhibition has been deleted.")
    end

  end

end
```

En la primera parte del test de “exhibitions controller spec” se observa que los usuarios no registrado no pueden realizar las acciones “new”, “create”, “edit” ni “update” sobre las exposiciones y se les redirecciona a la página principal.

En la segunda parte de dicho test, se puede ver que los usuarios registrados normales tampoco pueden realizar las acciones “new”, “create”, “edit” ni “update” sobre las exposiciones y se les redirecciona a la página principal.

Y en la última parte del test, se puede observar como los usuarios administradores pueden realizar cualquier tipo de acción sobre las exposiciones.

6.6.2. Desarrollo del código

En esta parte comentaremos las partes más relevantes a la hora de codificar en esta iteración. Lo primero que hacemos es crear el modelo de las exposiciones y especificar que los 3 campos que tiene no puedan estar vacíos a la hora de crear una exposición, es decir, que no puedan estar vacíos ni el nombre, ni la fecha ni el tipo de exposición.

Los tipos de exposiciones los añadimos en una tabla, por lo que el campo con el tipo de exposición perteneciente al formulario de la exposición lo que guardará será una referencia a un elemento de la tabla de tipos de exposición según el tipo de exposición correspondiente. También creamos el controlador de las exposiciones y las vistas correspondientes.

Y además, debemos autorizar sólo a los usuarios administradores a crear, editar o borrar exposiciones, para ello añadimos en el controlador de las exposiciones la siguiente línea:

```
before_filter :authorize_admin!, except: [:show, :index]
```

Esa función “authorize admin” está definida en el “application controller” de la siguiente manera:

```
def authorize_admin!
  authenticate_user!
  unless current_user.admin?
    flash[:alert] = 'You must be an admin to do that'
    redirect_to root_path
  end
end
```

6.6.3. Reejecución de los tests

Como última parte de la iteración, ejecutamos todos los tests y de esta manera comprobamos que se ha desarrollado todo de manera correcta de acuerdo a los objetivos planteados al principio de la iteración.

6.7. Iteración 7 - Gestión de tabla de puntos de cada exposición y de tabla de resultados

Realizaremos el modelado de la tabla de puntos, de las cuales para cada Año-Tipo de exposición-Premio tienen unos puntos determinados. En esta iteración, también realizaremos el modelado de la tabla de resultados en la cual para cada resultado de un perro que haya ganado un premio de un tipo de exposición determinado, tendrá un estado pendiente, validado o rechazado por el administrador. Sólo los usuarios logeados podrán mandar

resultados de sus perros. De esos resultados que envían los usuarios de sus perros, el administrador validará o rechazará cada uno de esos resultados. Los usuarios logueados podrán borrar sus propios resultados que estén en estado pendiente o rechazado. Sin embargo, el administrador podrá borrar resultados aunque estén validados.

En esta iteración, se ha decidido priorizar otro tipo de aspectos más importantes del trabajo de fin de grado y restar algo más de tiempo en lo que se refiere a la introducción de la tabla de puntos porque el número de horas del proyecto ya excedía de 300 horas. Por ello, el manejo de los puntos no se hará a través de interfaz gráfica, sino que se introducirán esas tablas de puntos directamente en la base de datos.

6.7.1. Tests

De los test *Cucumber* de esta iteración, mostraremos el "creating results", que es el más importante de los test de resultados.

```
Feature: Creating results
  In order to create an result
  As a user
  I want to create them easily
  Background:
    Given there are the following results:
      | exhibition   | award | dog                   | status  |
      | exhibition 1 | BOS   | Samba y Fatiga Idilio | Pending |

    And there are the following users:
      | email             | password | admin |
      | user1@example.com | password | false |
      | user2@example.com | password | false |
      | admin@example.com | password | true  |

    And there are the following dogs:
      | Sex  | Titles | Name                   | Birth Date | Owner             |
      | male |        | Samba y Fatiga Idilio  | 27/01/2006 | user1@example.com |
      | male |        | Fortunato Hautacuperche| 15/05/2008 | user2@example.com |

    Given there are the following exhibitions:
      | name         | date       | type                 |
      | exhibition 1 | 27/03/2015 | MONOGRÁFICA NACIONAL |
      | exhibition 2 | 15/07/2015 | Punto Obligatorio    |

    Given there are the following awards:
      | award            |
      | B.I.S.           |
      | 2º B.I.S.        |
      | 3º B.I.S.        |
      | 1º GRUPO         |
      | 2ºGRUPO          |
      | 3º GRUPO         |
      | MEJOR DE RAZA    |
```

```
    | BOS                |
    | CACIB              |
    | R.CACIB            |
    | RAPPEL CAC         |
    | CAC                |
    | R.CAC              |
    | Mejor Joven        |
    | CCJ/Exc.1 (Joven)  |

  And I am on the dogs page


Scenario: Creating a result like an admin
  When I am signed in as "admin@example.com"
  And I follow "Administrar Resultados"
  And I follow "New Result"
  And I select "exhibition 2" from "Exhibition"
  And I select "MEJOR DE RAZA" from "Award"
  And I select "Fortunato Hautacuperche" from "Dog"
  And I select "Validated" from "Status"
  And I press "Create Result"
  Then I should see "Result has been created.
      A message with the result's link has been sent to your email address"

@working
Scenario: Creating a result like a user
  When I am signed in as "user2@example.com"
  And I follow "My Results"
  And I follow "New Result"
  And I select "exhibition 2" from "Exhibition"
  And I select "MEJOR DE RAZA" from "Award"
  And I select "Fortunato Hautacuperche" from "Dog"
  And I press "Create Result"
  Then I should see "Result has been created.
      A message with the result's link has been sent to your email address"

Scenario: Creating a result without exhibition
  When I am signed in as "user2@example.com"
  And I follow "My Results"
  And I follow "New Result"
  And I select "Exhibition..." from "Exhibition"
  And I select "MEJOR DE RAZA" from "Award"
  And I select "Fortunato Hautacuperche" from "Dog"
  And I press "Create Result"
  Then I should see "Result has not been created."
  And I should see "Exhibition can't be blank"

Scenario: Creating a result without award
  When I am signed in as "user2@example.com"
```

```
    And I follow "My Results"
    And I follow "New Result"
    And I select "exhibition 2" from "Exhibition"
    And I select "Award..." from "Award"
    And I select "Fortunato Hautacuperche" from "Dog"
    And I press "Create Result"
    Then I should see "Result has not been created."
    And I should see "Award can't be blank"

  Scenario: Creating a result without dog
    When I am signed in as "user2@example.com"
    And I follow "My Results"
    And I follow "New Result"
    And I select "exhibition 2" from "Exhibition"
    And I select "MEJOR DE RAZA" from "Award"
    And I select "Dog..." from "Dog"
    And I press "Create Result"
    Then I should see "Result has not been created."
    And I should see "Dog can't be blank"
```

Como se puede observar en el test de "creating results", el test consta de 5 escenarios. En el primer escenario se comprueba que el administrador puede crear un resultado rellenando los campos correspondientes. Al final le aparecerá el mensaje "Result has been created. A message with the result's link has been sent to your email address".

En el segundo escenario se comprueba que un usuario normal puede crear un resultado de manera correcta pero sin aparecerle el campo "Status" que sólo le aparece al usuario administrador. El mensaje final que aparecería al crear el resultado sería "Result has been created. A message with the result's link has been sent to your email address".

En el tercer escenario se comprueba que no se crea el resultado cuando el usuario no selecciona una exposición y se muestran los mensajes de error "Result has not been created" y "Exhibition can't be blank".

En el cuarto escenario se comprueba que no se crea el resultado cuando el usuario no selecciona un premio y se muestran los mensajes de error "Result has not been created" y "Award can't be blank".

Y en el quinto y último escenario, se comprueba que no se crea el resultado cuando el usuario no selecciona un perro y se muestran los mensajes de error "Result has not been created" y "Dog can't be blank".

Y de los tests *RSpec*, mostraremos el nuevo test creado para los resultados, que es de gran importancia en esta iteración.

```
require 'spec_helper'

describe ResultsController do
  let(:user) { FactoryGirl.create(:user) }

  let(:exhibition) { FactoryGirl.create(:exhibition)}

  let(:award) { FactoryGirl.create(:award)}

  let(:dog) { FactoryGirl.create(:dog)}
```

```
let(:valid_attributes) {{exhibition_id: exhibition, award_id: award, dog_id: dog}}

let(:valid_attributes2) { {exhibition_id: '1', award_id: '8', dog_id: '1',
                           status: 'Validated'}  }

let(:valid_attributes3) { {exhibition_id: '1', award_id: '8', dog_id: '1',
                           status: 'Rejected'}  }

context 'Guest users about results' do

  before { @result =  FactoryGirl.create(:result) }

  it 'cannot access to the new action' do
    get :new
    response.should redirect_to(new_user_session_path)
  end

  it 'cannot access to the create action' do
    expect {post :create, result: valid_attributes}.not_to change(Result, :count)
    response.should redirect_to(new_user_session_path)
  end

  it 'cannot access to the edit action' do
    get :edit, id: @result.to_param
    response.should redirect_to(new_user_session_path)
  end

  it 'cannot access to the update action' do
    put :update, {id: @result.to_param, result: valid_attributes}
    response.should redirect_to(new_user_session_path)
  end

  it 'cannot access to the delete action' do
    expect {delete :destroy, id: @result.to_param}.not_to change(Result, :count)
    response.should redirect_to(new_user_session_path)
  end

  it 'cannot access to the show action' do
    get :show, id: @result.to_param
    response.should redirect_to(new_user_session_path)
  end

end


context 'Standard users about results' do

  before {
```

```
    @result =  FactoryGirl.create(:result)

    @result_validated = FactoryGirl.create(:result, valid_attributes2)

    @result_rejected = FactoryGirl.create(:result, valid_attributes3)

  }

  it 'can access to the new action' do
    sign_in(:user, user)
    get :new
    response.should be_success
  end

  it "can access the create action" do
    sign_in(:user, user)
    post :create, result: valid_attributes
    response.should redirect_to(results_path)
    flash[:notice].should eql("Result has been created. A message with the
                               result's link has been sent to your email address")
  end

  it 'cannot access to the edit action' do
    sign_in(:user, user)
    get :edit, id: @result.to_param
    response.should redirect_to(root_path)
  end

  it 'cannot access to the update action' do
    sign_in(:user, user)
    put :update, {id: @result.to_param, result: valid_attributes}
    response.should redirect_to(root_path)
  end

  it 'can access to the delete action with results pending ' do
    sign_in(:user, user)
    delete :destroy, id: @result.to_param
    response.should redirect_to(results_path)
    flash[:notice].should eql("Result has been deleted.")
  end

  it 'cannot access to the delete action with results validated ' do
    sign_in(:user, user)
    expect {delete :destroy,id: @result_validated.to_param}.not_to change(Result,
                                                                          :count)
    response.should redirect_to(results_path)
  end

  it 'can access to the delete action with results rejected ' do
```

```
      sign_in(:user, user)
      delete :destroy, id: @result_rejected.to_param
      response.should redirect_to(results_path)
      flash[:notice].should eql("Result has been deleted.")
    end

    it 'can access to the show action' do
      sign_in(:user, user)
      get :show, id: @result.to_param
      response.should be_success
    end

    it 'can access to the index action' do
      sign_in(:user, user)
      get :index
      response.should be_success
    end

  end

end
```

Como se puede observar en el test "results controller spec", en la primera parte del test se comprueba que los usuarios no registrados no podrían acceder a ningún tipo de acción sobre los resultados.

Y en la segunda parte de dicho test, se comprueba que los usuarios registrados no pueden realizar las acciones de "edit" ni "updated" sobre sus resultados, ni tampoco pueden eliminar los resultados con estado "Validated".

Además, de los tests *RSpec* mostraremos el test realizado sobre el seed, ya que la tabla de puntos no la hemos realizado sobre vistas, sino directamente se añade la tabla de puntos en la base de datos. Esto se ha realizado así por lo dicho anteriormente, por simplificar porque el número de horas del proyecto ya excedía de 300 horas.

```
require 'spec_helper'
feature "Seed Data" do

  scenario "The basics" do
    load Rails.root + "db/seeds.rb"
    User.where(email: "admin@example.com").first!
  end

  scenario "Types" do
    load Rails.root + "db/seeds.rb"
    Type.all.count.should eq 6
  end

  scenario "Awards" do
    load Rails.root + "db/seeds.rb"
    Award.all.count.should eq 15
  end
```

```
 scenario "Exhibitions" do
  load Rails.root + "db/seeds.rb"
  Exhibition.all.count.should eq 3
  Exhibition.where(type_id: Type.where(name:'MONOGRÁFICA NACIONAL')).count.should eq 2
  Exhibition.where(type_id: '3').count.should eq 1
 end

  scenario "MONOGRÁFICA NACIONAL" do
    load Rails.root + "db/seeds.rb"
    Point.where(type_id: '1').count.should eq 7
  end

  scenario "CONCURSO MONOGRÁFICO NACIONAL O REGIONAL" do
    load Rails.root + "db/seeds.rb"
    Point.where(type_id: '2').count.should eq 4
  end

  scenario "Especial A.E.F.R.B.F." do
    load Rails.root + "db/seeds.rb"
    Point.where(type_id: '3').count.should eq 14
  end

  scenario "Punto Obligatorio" do
    load Rails.root + "db/seeds.rb"
    Point.where(type_id: '4').count.should eq 14
  end

  scenario "EXPOSICIÓN INTERNACIONAL" do
    load Rails.root + "db/seeds.rb"
    Point.where(type_id: '5').count.should eq 13
  end

  scenario "EXPOSICIÓN NACIONAL" do
    load Rails.root + "db/seeds.rb"
    Point.where(type_id: '6').count.should eq 14
  end

  scenario "Points" do
    load Rails.root + "db/seeds.rb"
    Point.where(year: '2014').count.should eq 66
  end
end
```

En el test sobre el fichero “seed”, lo que comprobamos es que se crean correctamente los tipos de exposiciones, los premios, las exposiciones y la tabla de puntos especificados en el fichero “seed”.

6.7.2. Desarrollo del código

En esta parte comentaremos las partes más relevantes a la hora de codificar en esta iteración. Lo primero a destacar es que comentamos anteriormente, en esta iteración la tabla de puntos la vamos a añadir directamente en la base de datos para simplificar ya que el TFG ya se excede de 300 horas.

Con los modelos de premios, puntos, tipo de exposición y resultados creados, pasamos a los controladores de resultados para controlar según sea un administrador u otro usuario poder realizar diferentes acciones sobre las exposiciones. Por ejemplo, a la hora de crear un resultado, un usuario normal no dispone del campo "Status" sino que cuando envía el resultado el estado se pone por defecto a "Pending". Sin embargo, el administrador si que dispone del campo "Status" a la hora de crear nuevos resultados y también para editar los resultados enviados por los usuarios y ponerlos a "Validated" o "Rejected" según considere oportuno el administrador.

En el controlador de los perros añadimos una nueva acción llamada "ranking", dicha acción nos servirá para controlar el ranking de los perros de la asociación según sus resultados obtenidos en las diferentes exposiciones. De los resultados correspondientes a cada perro, sólo se tienen en cuenta en el ranking los resultados validados, los demás no se tienen en cuenta a la hora de obtener la puntuación de cada perro en el ranking. Conseguir realizar la tabla del ranking, sumando los puntos de los resultados de cada perro obteniendo las cosas de varias tablas me costó bastante tiempo para realizarlo correctamente, pero finalmente conseguí que funcionase bien.

También creamos la clase "ResultMailer" donde definimos los correos que se enviarán en cada caso cuando se crea un resultado por un usuario el correo que se enviaría a ese mismo usuario y al administrador, y cuando el administrador valida o rechaza un resultado el correo que se envía al usuario que envió dicho resultado. A continuación, se muestra el contenido de la clase "ResultMailer":

```
class ResultMailer < ActionMailer::Base
  default from: "notifications@example.com"

  def create_result(user, result)
    @user = user
    @url  = result_path(result)
    mail(to: @user.email, subject: 'The result has been created')
  end

  def create_result_to_admin(user, result, admins)
    @user = user
    @url  = edit_admin_result_path(result)
    emails = admins.collect(&:email).join(",")
    mail(from: @user.email, to: emails,
         subject: "A result has been created by #{@user.email}")
  end

  def edit_result_by_admin(user, result, admin)
    @user = user
    @url  = result_path(result)
    mail(from: admin.email, to: @user.email,
         subject: 'The response to the result sent')
```

```
  end

end
```

Y ponemos en los controladores de resultados las creaciones de esos mailers en los casos correspondientes. Por ejemplo, cuando el administrador edita un resultado para validarlo o rechazarlo, ponemos lo siguiente para que se envíe ese correo:

```
ResultMailer.edit_result_by_admin(@user,@result,current_user)
```

6.7.3. Reejecución de los tests

Y como en cada iteración, la última parte siempre es ejecutar los tests *Cucumber* y *RSpec* creados al principio de esta iteración para comprobar que la aplicación funciona de manera correcta, de acuerdo a los aspectos planteados al comienzo de dicha iteración.

6.8. Iteración 8 - Gestión de tabla de resultados

En esta iteración, realizaremos modificaciones en el modelo de usuarios y en el modelo de perros, en el cual se añadirá un campo nuevo para almacenar la imagen de un usuario o de un perro respectivamente. No será obligatorio el que tengan una foto, por lo que dicho campo imagen podrá estar vacío. Se hará uso de la gema CarrierWave para subir las imágenes a la aplicación.

6.8.1. Tests

A modo de visualizar que se ha introducido la imagen tanto para perros como para usuarios, mostraremos el test *Cucumber* "viewing dogs" en el que se ven las tablas de usuario y perro con su imagen.

```
Feature: Viewing dogs
  In order to view the dogs
  I want to see them on that dog's page

  Background:
    Given there are the following users:
      | email             | password |  image     |
      | user@example.com  | password | user.jpeg |


    And "user@example.com" is the owner of the following dogs:
      | Sex  | Titles | Name                    | Birth Date | Image      |
      | male |        | Samba y Fatiga Idilio   | 27/01/2006 | perro.jpeg |
      | male |        | Fortunato Hautacuperche | 15/05/2008 |            |

  Scenario: Viewing dogs
    When I am on the dogs page
    Then I should see "Samba y Fatiga Idilio"
```

```
    And I should see "Fortunato Hautacuperche"
    And I should see "user@example.com"
```

Como se puede observar en el test de "viewing dogs", es un test corto de un único escenario. En dicho escenario lo que se comprueba es que se vea el listado de los perros de la aplicación. Y en la tabla de los perros se puede ver que está añadido su campo imagen.

6.8.2. Desarrollo del código

Explicaremos ahora lo más relevante en esta iteración a la hora de codificarla. En esta iteración, en la codificación el aspecto principal fue saber usar correctamente la gema "CarrierWave". Dicha gema facilita de carga y manipulación de imágenes desde el cliente. Iremos explicando paso a paso lo realizado para configurar todo y que se subiesen las imágenes correctamente.

Primero instalamos la gema "CarrierWave":

```
$ gem carrierwave
```

A continuación, generamos un "uploader" ejecutando lo siguiente:

```
$ rails generate uploader Imagen
```

El siguiente paso es vincular el uploader con un campo del modelo que lo utilizará, que en nuestro caso lo usan el modelo "user.rb" y el modelo "dog.rb". Para ello, añadimos tanto en el modelo "user.rb" como en el modelo "dog.rb" lo siguiente:

```
mount_uploader :image, ImageUploader
```

Incluimos un campo de tipo file tanto en el formulario de los usuarios como en el de los perros, como el fragmento que se muestra a continuación:

```
<p>
  <%= f.label :image %>
  <%= f.file_field :image%>
</p>
```

En cada uno de esos formularios debemos especificar "multipart:true" e impedir el comportamiento ajax de rails. A continuación se muestran como ejemplo la cabecera de formulario de los perros:

```
<%= form_for @dog, html: {multipart: true, "data-ajax" => "false"} do |f|  %>
```

Y finalmente, añadimos en el fichero que vayamos a usar la imagen subida, algo como lo que se muestra a continuación que es un fragmento del "show.html.erb" de los perros:

```
<%= image_tag(@dog.image_url) if @dog.image.present? %>
```

Con todo lo explicado, ya tendríamos realizado el objetivo principal de esta iteración, que era poder añadir una imagen a los perros y a los usuarios y luego mostrarlas en las diferentes partes de la aplicación donde nos haga falta mostrar dichas imágenes.

6.8.3. Reejecución de los tests

En el final de la iteración ejecutamos los test *Cucumber* y *RSpec* creados al comienzo de la iteración y comprobamos que todo funciona correctamente, es decir, que la aplicación al final de la iteración funciona como tiene que funcionar y se puede dar paso al comienzo de la siguiente iteración.

6.9. Iteración 9 - Gestión de los usuarios socios

En la novena iteración, completaremos el modelo de los usuarios introduciendo los usuarios socios. Debemos tener en cuenta que, sólo los usuarios socios podrán crear perros, enviar resultados, etc. Los usuarios registrados son usuarios socios que han dejado de serlo, sus datos se conservarán en la aplicación pero y que a nivel de funcionalidad tienen los mismo privilegios que un usuario no registrado.

6.9.1. Tests

De esta iteración mostraremos sólo el test *Cucumber* de "creating users", en el cual se podrá observar el que se han añadido los usuarios socios en la aplicación y la manera en que el usuario administrador puede hacer socio o quitarle esa opción de socio a los usuarios de la aplicación.

```
Feature: Creating Users
  In order to create a user
  As an admin
  I want to create them easily

  Background:
    Given there are the following users:
      | email              | password | admin | image      | cahoot |
      | user1@example.com  | password | false | user.jpeg | true   |
      | user2@example.com  | password | false |            | true   |
      | user3@example.com  | password | false |            | false  |
      | admin@example.com  | password | true  |            | true   |

    And I am signed in as "admin@example.com"
    And I am on the users page


  Scenario: Creating a user
    When I follow "New User"
    And I attach the file "images/user/user.jpeg" to "Image"
    And I fill in "Email" with "user4@example.com"
    And I fill in "Password" with "password"
    And I fill in "Password confirmation" with "password"
    And I press "Create User"
    Then I should see "User has been created"
    And I should see "user4@example.com (User)"
@working
  Scenario: Creating a cahoot user
    When I follow "New User"
    And I attach the file "images/user/user.jpeg" to "Image"
    And I fill in "Email" with "user4@example.com"
    And I fill in "Password" with "password"
    And I fill in "Password confirmation" with "password"
    And I check "Is a cahoot?"
```

```
  And I press "Create User"
  Then I should see "User has been created"
  And I should see "user4@example.com (User) (Cahoot)"

Scenario: Creating an admin user
  When I follow "New User"
  And I attach the file "images/user/user.jpeg" to "Image"
  When I fill in "Email" with "admin2@example.com"
  And I fill in "Password" with "password"
  And I fill in "Password confirmation" with "password"
  And I check "Is an admin?"
  And I press "Create User"
  Then I should see "User has been created."
  And I should see "admin2@example.com (Admin)"

Scenario: Creating a user without password
  When I follow "New User"
  And I attach the file "images/user/user.jpeg" to "Image"
  And I fill in "Email" with "user4@example.com"
  And I press "Create User"
  Then I should see "User has not been created"
  And I should see "Password can't be blank"

Scenario: Creating a user without email
  When I follow "New User"
  And I attach the file "images/user/user.jpeg" to "Image"
  And I fill in "Password" with "password"
  And I fill in "Password confirmation" with "password"
  And I press "Create User"
  Then I should see "User has not been created"
  And I should see "Email can't be blank"

Scenario: Creating a user with a different password and confirmation password
  When I follow "New User"
  And I attach the file "images/user/user.jpeg" to "Image"
  And I fill in "Email" with "user4@example.com"
  And I fill in "Password" with "password"
  And I fill in "Password confirmation" with "pasword"
  And I press "Create User"
  Then I should see "User has not been created"
  And I should see "Password confirmation doesn't match Password"
```

En el test de "creating users" podemos observar que hay 6 escenarios. En el primer escenario se comprueba que un usuario administrador puede crear un usuario normal correctamente rellenando los campos correspondientes sin marcar la casilla de que el nuevo usuario sea administrador, cuyo mensaje final es "User has been created".

En el segundo escenario se comprueba que un usuario administrador puede crear un usuario socio rellenando todos los campos correspondientes y marcando la casilla de "Is a cahoot?", cuyo mensaje final es "User has been created".

En el tercer escenario se comprueba que un usuario administrador puede crear otro usuario administrador rellenando todos los campos correspondientes y marcando la casilla de “Is an admin?”, cuyo mensaje final es “User has been created”.

En el cuarto escenario se comprueba que no se crea el usuario si el administrador deja vacío el campo “Password” del nuevo usuario, cuyos mensajes de error serán “User has not been created” y “Password can’t be blank”.

En el quinto escenario se comprueba que no se crea el usuario si el administrador deja vacío el campo “Email” del nuevo usuario, cuyos mensajes de error serán “User has not been created” e “Email can’t be blank”.

Y en el sexto y último escenario, se comprueba que no se crea el usuario si el administrador introduce la contraseña introducida no es igual en los campos “Password” y “Password confirmation”, apareciendo los mensajes de error “User has been created” y “Password confimation doesn’t match Password”.

6.9.2. Desarrollo del código

Explicaremos ahora lo más relevante en esta iteración a la hora de codificarla. En esta iteración, en la codificación uno de los aspectos principales fue crear un nuevo campo en la tabla de los usuarios llamado “cahoot”, el cual si está a true quiere decir que es un usuario socio y si está a false quiere decir que es un usuario registrado que era socio pero que ya no lo es.

Además, en esta iteración hay que modificar las autorizaciones, ya que los usuarios registrados únicamente podrán ver los diferentes recursos de la aplicación y modificar su perfil. Los socios los que podrán crear, editar y borrar perros y resultados y el usuario administrador seguirá con las mismas autorizaciones que tenía al comienzo de esta iteración, ya que en esta iteración no hay que cambiar las autorizaciones del administrador. El fichero Ability tras cambias las autorizaciones correspondientes de acuerdo a lo dicho anteriormente, nos queda de la siguiente manera:

```
class Ability
  include CanCan::Ability

  def initialize(user)
    user ||= User.new # guest user (not logged in)

    cannot :manage, :all

    can :index, User
    can :index, Dog

    if user.persisted?
      can [:index, :show], Dog
      can [:edit, :update], User, ['user.id = ?', user.id] do |u|
        u.id == user.id
      end
      can [:index, :show], Result, ['dog_id = ?', user.id] do |r|
        r.dog_id.user_id == user.id
      end
    end
```

```
    if user.cahoot?
      can [:index, :show], Dog
      can [:new, :create], Dog
      can [:edit, :update, :destroy], Dog, ['dogs.user_id = ?', user.id] do |dog|
        dog.user_id == user.id
      end
      can [:edit, :update], User, ['user.id = ?', user.id] do |u|
        u.id == user.id
      end
      can [:new, :create], Result
      can [:index, :show, :destroy], Result, ['dog_id = ?', user.id] do |r|
        r.dog_id.user_id == user.id
      end
    end

    if user.admin?
      can :index, :all
      can [:edit, :update, :destroy], Dog
      can [:new, :create, :edit, :update, :destroy], User
      can [:new, :create, :edit, :update, :destroy], Result
    end

  end
end
```

Como se puede observar, al añadir las autorizaciones necesarias para esta iteración, ya el fichero ability va cogiendo una estructura algo más compleja que en las iteraciones anteriores.

6.9.3. Reejecución de los tests

Finalmente, ejecutaremos los tests *Cucumber* y *RSpec* obtenidos previamente al principio de la iteración tanto los creados en esta iteración como los modificados de las iteraciones anteriores. De dicha ejecución, obtenemos como conclusión que al no haber fallos, la aplicación funciona correctamente y esta iteración se da por finalizada.

6.10. Iteración 10 - Vistas introduciendo Bootstrap

En esta iteración, introduciremos Bootstrap en la aplicación, para dar un mejor aspecto visual a la aplicación. Boostrap hace que el desarrollo web front-end sea más rápido y menos complicado.

6.10.1. Tests

De esta iteración mostraremos un test de los varios tests *RSpec* que se han creado de manera sencilla para comprobar al final de la iteración que se está aplicando el estilo que queremos en las vistas de la aplicación. El test que mostraremos será el “in-

dex.html.erb_spec.rb" de los perros , ya que como los perros son el elemento principal de la aplicación, qué mejor que mostrar un test sobre la vista index de los perros.

```
require 'spec_helper'
include Devise::TestHelpers

describe "dogs/index.html.erb" do
  after(:all) { DatabaseCleaner.clean_with(:truncation) }
  it "display all dogs" do
    cahoot = FactoryGirl.create(:cahoot)
    admin = FactoryGirl.create(:admin)
    sign_in :user, cahoot
    assign(:dogs, [
        Dog.create!(name: 'Samba y Fatiga Idilio', sex: 'male',
                    birth_date: '01/01/2001', user_id: cahoot.id,
                    image: File.open(File.join(Rails.root,
                          'spec/support/attachments/images/dog/perro.jpeg'))),
        Dog.create!(name: 'Fortunato Hautacuperche', sex: 'male',
                    birth_date: '01/02/2007', user_id: admin.id,
                    image: File.open(File.join(Rails.root,
                         'spec/support/attachments/images/dog/perro.jpeg')))
    ])
    render :template => "dogs/index.html.erb"
    expect(rendered).to match /class="table table-hover table-striped"/
    expect(rendered).to match /class="dog"/
    expect(rendered).to match /class="imageDog img-rounded"/
    expect(rendered).to match /class="glyphicon glyphicon-plus link"/
  end
end
```

En el test sobre la vista del index de los perros, se puede ver que es un test sencillo, los test sobre las vistas son muy complejos, pero en este caso sólo queremos comprobar que el estilo que se aplicar es el correcto. En dicho test se puede ver como se hace un render de la vista del index de los perros y se especifican las clases que tienen que aparecer en el estilo de la vista.

6.10.2. Desarrollo del código

Explicaremos ahora lo más relevante en esta iteración a la hora de introducir *Bootstrap* en el código de la aplicación. Como hemos resaltado anteriormente, *Boostrap* hace que el desarrollo web front-end sea más rápido y menos complicado.

Iré mostrando y comentando algunas partes del código donde se ha introducido *Bootstrap* pero no todo, sobre todo comentaré el "application.html.erb" que es la vista fundamental. De todas formas, el código se puede consultar en el CD entregado.

Para añadir *Bootstrap* a nuestra aplicación debemos introducir en el fichero "application.html.erb" en el "head" de la página lo siguiente:

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0/css/bootstrap.min.css">
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0/css/bootstrap-theme.min.css">
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0/js/bootstrap.min.js"></script>
```

En este mismo fichero añadimos el yield dentro de un div de clase container. como vemos en el siguiente fragmento:

```
<div class="container">
  <%= yield %>
</div>
```

A continuación, en el siguiente fragmento mostramos cómo añadimos en "application.html.erb" una barra de navegación usando las clases que nos proporciona Bootstrap para ello como es "navbar" por ejemplo:

```
<nav class="navbar navbar-default navbar-fixed-top" role="navigation">
 <div class="container-fluid bigBar">
  <div id="bar" class="container-fluid">
   <div id="header1" class="navbar-header">
    <a class="navbar-brand aefrbf" href="/"><%= image_tag "aefrbf.jpg"%></a>
   </div>
   <div id="header2" class="collapse navbar-collapse">
    <ul id="menu" class="nav navbar-nav">
    <% if user_signed_in? %>
     <% if current_user.admin='true' %>
      <li id=<%= @adminUsers%>>
        <%= link_to "Users management", admin_users_path, class:'textHeader' %>
      </li>
      <li id=<%= @adminResults%>>
        <%=link_to "Results management", admin_results_path, class:'textHeader'%>
      </li>
     <% end %>
     <li id=<%= @theRanking%>>
        <%= link_to "Ranking", dogs_ranking_path, {class:'textHeader'}  %>
     </li>
     <li id=<%= @myProfile%>>
        <%= link_to "My profile", user_path(current_user), class:'textHeader' %>
     </li>
     <li id=<%= @myResults%>>
        <%= link_to "My results",results_path, class:'textHeader' %>
     </li>
     <li id=<%= @allExhibitions%>>
        <%= link_to "Exhibitions", exhibitions_path, class:'textHeader' %>
     </li>
     <li>
        <%= link_to "Sign out", destroy_user_session_path, method: :delete, class:'textHeader' %>
     </li>
     <li>
        <p class="navbar-text navbar-right textSign">Signed in as <%= current_user.email %></p>
     </li>
    <% else %>
     <li id=<%= @signUp%>>
        <%= link_to "Sign up", new_user_registration_path, class:'textHeader' %>
     </li>
     <li id=<%= @signIn%>>
        <%= link_to "Sign in", new_user_session_path, class:'textHeader' %>
     </li>
    <% end %>
    </ul>
   </div>
  </div>
 </div>
</nav>
```

En el siguiente fragmento de código, vemos cómo añadimos un footer a la página de "application.html.erb":

```
<footer>
  <div class="container">
    <%= image_tag "bulldogs.gif", :id => "gif"%>
  </div>
</footer>
```

También hemos dado estilo a las tablas como se puede observar en el siguiente fragmento del "index.html.erb" de los perros:

```
<table class="table table-hover table-striped">
  <tr>
    <th>Photo</th>
    <th>Name</th>
    <th>Owner</th>
    <th></th>
    <th></th>
    <th></th>
  </tr>
  <tbody class="searchable">
    <%= render @dogs %>
  </tbody>
</table><br>
```

Además de todos los elementos de *Bootstrap* usados, me gustaría resaltar también algunos elementos de CSS añadidos. Por ejemplo, la imagen que se observa al acceder al perfil de un perro, al pasar el ratón por encima de la imagen del perro, dicha imagen se mueve hacia el frente, aumenta un poco y se gira un poco hacia la izquierda . Este efecto lo he conseguido añadiendo el siguiente código CSS siendo .imgDog la clase que le he puesto a la imagen del perfil del perro:

```
.imgDog {
  position:relative;
  overflow:visible;
  border: 4px 4px 4px;
  width: 300px;
  min-height:250px;
  max-height: 300px;
  margin-top: 40px;
  margin-bottom:20px;
  margin-left:30px;
  -webkit-transition: -webkit-transform 0.5s ease-in-out;
  -moz-transition: -moz-transform 0.5s ease-in-out;
  transition: transform 0.5s ease-in-out;
}
.imgDog:hover {
  position:relative;
  overflow:visible;
  transform: scale(1,1.05) skew(5deg, -5deg) translate(-5px,-5px);
  -webkit-transform: scale(1,1.05) skew(5deg, -5deg) translate(-5px,-5px);
  -moz-transform: scale(1,1.05) skew(5deg, -5deg) translate(-5px,-5px);
```

```
  -webkit-box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.5);
  -moz-box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.5);
  box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.5);
}
```

Lo último que me gustaría resaltar respecto al estilo añadido es el añadir a los links iconos con Bootstrap. Por ejemplo, en el fichero "index.html.erb" de los perros mostramos que se le ha añadido un inoco con un "+" al link de "New Dog" como se muestra en la siguiente línea de código:

```
<%= link_to "New Dog", new_dog_path, class: "glyphicon glyphicon-plus link" %>
```

6.10.3. Reejecución de los tests

Al final de la iteración, ejecutamos los tests *RSpec* sencillos creados al comienzo de esta iteración para comprobar que se está aplicando el estilo deseado en la aplicación. Al ejecutarse con éxito todos los tests, quiere decir que todo funciona correctamente y que ya podemos pasar a la última iteración del proyecto.

6.11. Iteración 11 - Enlaces ocultos

En esta última iteración, ocultaremos todos los links que no deben ver algunos tipos de usuarios. Esta iteración no reviste mayor complicación, es una iteración sencilla, en la que lo único que hay que tener es cuidado para ocultar todos los enlaces que algunos tipos de usuarios no deben ver, y modificar los test correspondientes que comprobaban que esos links que algunos usuarios no podían ver, al acceder a ellos les salía un mensaje de que no podían acceder a dichos enlaces.

6.11.1. Tests

En esta iteración, como de los test de iteraciones anteriores lo único que se ha hecho es eliminar casos en los que se comprobaba que no tuviese acceso a determinados links algunos usuarios, mostraremos el test *Cucumber* importante creado en esta iteración que es "hidden links", en el que como hemos dicho anteriormente comprobamos que hayamos ocultado esos links correctamente para determinados tipos de usuarios sin privilegios para visualizar determinados links de la aplicación.

```
Feature: Creating dogs
  In order to create a dog
  As a user
  I want to create them easily
  Background:
    Given there are the following users:
      | email              | password | admin | image     | cahoot |
      | user1@example.com  | password | false | user.jpeg | true   |
      | user2@example.com  | password | false |           | true   |
      | user3@example.com  | password | false |           | false  |
      | admin@example.com  | password | true  |           | true   |
```

```
And there are the following dogs:
  |Sex |Titles|Name                    |Birth Date|Owner            |Image     |
  |male|      |Samba y Fatiga Idilio   |27/01/2006|user1@example.com|perro.jpeg|
  |male|      |Fortunato Hautacuperche|15/05/2008|user2@example.com|          |
  |male|      |Frael Valderrama        |01/01/2001|user3@example.com|          |

Given there are the following types:
  | name                                  |
  | MONOGRÁFICA NACIONAL                  |
  | CONCURSO MONOGRÁFICO NACIONAL O REGIONAL |
  | Especial A.E.F.R.B.F.                 |
  | Punto Obligatorio                     |
  | EXPOSICIÓN INTERNACIONAL              |
  | EXPOSICIÓN NACIONAL                   |

And there are the following awards:
  | award             |
  | B.I.S.            |
  | 2° B.I.S.         |
  | 3° B.I.S.         |
  | 1° GRUPO          |
  | 2°GRUPO           |
  | 3° GRUPO          |
  | MEJOR DE RAZA     |
  | BOS               |
  | CACIB             |
  | R.CACIB           |
  | RAPPEL CAC        |
  | CAC               |
  | R.CAC             |
  | Mejor Joven       |
  | CCJ/Exc.1 (Joven) |

Given there are the following exhibitions:
  | name         | date       | type                  |
  | exhibition 1 | 27/03/2015 | MONOGRÁFICA NACIONAL  |
  | exhibition 2 | 15/07/2015 | Punto Obligatorio     |
  | exhibition 3 | 12/03/2015 | EXPOSICIÓN NACIONAL   |

And there are the following results:
  | exhibition | award | dog | status    |
  | 1          | 8     | 1   | Pending   |
  | 2          | 7     | 2   | Rejected  |
  | 3          | 12    | 1   | Validated |
  | 1          | 7     | 3   | Pending   |
  | 2          | 4     | 3   | Rejected  |
  | 3          | 2     | 3   | Validated |
```

```
  And I am on the dogs page

Scenario: Dogs link is hidden for non-signed-in users
  Then I should not see the "Dogs" link

Scenario: New dog link is hidden for registered users
  When I am signed in as "user3@example.com"
  And I follow "Dogs"
  Then I should not see the "New Dog" link

Scenario: Exhibitions link is hidden for non-signed-in users
  Then I should not see the "Exhibitions" link

Scenario: New exhibition link is hidden for cahoot users
  When I am signed in as "user1@example.com"
  And I follow "Exhibitions"
  Then I should not see the "New Exhibition" link

Scenario: New exhibition link is hidden for registered users
  When I am signed in as "user3@example.com"
  And I follow "Exhibitions"
  Then I should not see the "New Exhibition" link

Scenario: My results link is hidden for non-signed-in users
  Then I should not see the "My results" link

Scenario: New result link is hidden for registered users
  When I am signed in as "user3@example.com"
  And I follow "My results"
  Then I should not see the "New Result" link

Scenario: Users management link is hidden for non-signed-in users
  Then I should not see the "Users" link

Scenario: Users management link is hidden for registered users
  When I am signed in as "user3@example.com"
  Then I should not see the "Users" link

Scenario: Users management link is hidden for cahoot users
  When I am signed in as "user2@example.com"
  Then I should not see the "Users" link

Scenario: Delete dog link is hidden for cahoot users without property
  When I am signed in as "user1@example.com"
  And I follow "Dogs"
  Then I should not see "Delete" within "#dog_2"
```

```
Scenario: Delete dog link is hidden for registered users with property
  When I am signed in as "user3@example.com"
  And I follow "Dogs"
  Then I should not see "Delete" within "#dog_3"

Scenario: Delete dog link is hidden for registered users without property
  When I am signed in as "user3@example.com"
  And I follow "Dogs"
  Then I should not see "Delete" within "#dog_1"

Scenario: Delete an exhibition link is hidden for cahoot users
  When I am signed in as "user1@example.com"
  And I follow "Exhibitions"
  Then I should not see "Delete" within "#exhibition_1"

Scenario: Delete an exhibition link is hidden for registered users
  When I am signed in as "user3@example.com"
  And I follow "Exhibitions"
  Then I should not see "Delete" within "#exhibition_1"

Scenario: Delete a Pending result link for registered users
  When I am signed in as "user3@example.com"
  And I follow "My results"
  Then I should not see "Delete" within "#result_4"

Scenario: Delete a Rejected result link for registered users
  When I am signed in as "user3@example.com"
  And I follow "My results"
  Then I should not see "Delete" within "#result_5"

Scenario: Delete a Validated result link for registered users
  When I am signed in as "user3@example.com"
  And I follow "My results"
  Then I should not see "Delete" within "#result_6"

Scenario: Results management link is hidden for non-signed-in users
  Then I should not see the "Results" link

Scenario: Results management link is hidden for registered users
  When I am signed in as "user3@example.com"
  Then I should not see the "Results" link

Scenario: Results management link is hidden for cahoot users
  When I am signed in as "user2@example.com"
  Then I should not see the "Results" link

Scenario: Edit a dog link for cahoot users without property
  And I am signed in as "user1@example.com"
  And I follow "Dogs"
```

```
  Then I should not see "Edit" within "#dog_2"

Scenario: Edit a dog link for registered users with property
  And I am signed in as "user3@example.com"
  And I follow "Dogs"
  Then I should not see "Edit" within "#dog_3"

Scenario: Edit a dog link for registered users  without property
  And I am signed in as "user3@example.com"
  And I follow "Dogs"
  Then I should not see "Edit" within "#dog_1"

Scenario: Edit an exhibition for cahoot users
  When I am signed in as "user1@example.com"
  And I follow "Exhibitions"
  Then I should not see "Edit" within "#exhibition_1"

Scenario: Edit an exhibition for registered users
  When I am signed in as "user3@example.com"
  And I follow "Exhibitions"
  Then I should not see "Edit" within "#exhibition_1"

Scenario: Edit a result link for cahoot users
  When I am signed in as "user1@example.com"
  And I follow "My results"
  Then I should not see "Edit" within "#result_1"

Scenario: Edit a result link for registered users
  When I am signed in as "user3@example.com"
  And I follow "My results"
  Then I should not see "Edit" within "#result_4"

Scenario: Edit other user profile link for  non-signed-in users
  And I follow "user2@example.com"
  Then I should not see the "Edit User" link

Scenario: Edit other user profile link for cahoot users
  When I am signed in as "user1@example.com"
  And I follow "Dogs"
  And I follow "user2@example.com"
  Then I should not see the "Edit User" link

Scenario: Edit other user profile link for registered users
  When I am signed in as "user3@example.com"
  And I follow "Dogs"
  And I follow "user2@example.com"
  Then I should not see the "Edit User" link
```

Se puede observar que el test “hidden links” es bastante largo pero no es complejo. En dicho test lo único que se hace es comprobar que los links que antes aparecían pero daban mensajes de error porque no se tenían privilegios para acceder a dichos links pues en este test se comprueba que dichos links no accesibles a ciertos usuarios no aparezcan al finalizar esta última iteración.

6.11.2. Desarrollo del código

Explicaremos ahora lo más relevante en esta iteración a la hora de ocultar todos los links que cierto tipo de usuarios no deben ver. Por ejemplo, al link de “New Dog” que aparece en la página de index de los perros, le añadimos una condición comprobando si ese usuario tiene permiso para crear un perro. Si el usuario tiene permiso para crear un perro le aparecerá el link “New Dog” y sino tiene permiso para ello, pues no le aparecerá dicho link. En el siguiente fragmento de código se muestra como se comprueba lo dicho anteriormente sobre ese link de “New Dog”:

```
<% if can? :new, Dog  %>
  <%= link_to "New Dog", new_dog_path, class: "glyphicon glyphicon-plus link" %>
<% end %>
```

A continuación, mostramos un fragmento del “application.html.erb” en el que aparece el menú de navegación, para comentar luego los links que hemos ocultado:

```
<nav class="navbar navbar-default navbar-fixed-top" role="navigation">
  <div class="container-fluid bigBar">
    <div id="bar" class="container">
       <div id="header1" class="navbar-header">
          <a class="navbar-brand aefrbf" href="/"><%= image_tag "aefrbf.jpg"%></a>
       </div>
       <div id="header2" class="collapse navbar-collapse">
         <ul id="menu" class="nav navbar-nav">
            <li id=<%= @theDogs%>>
              <%= link_to "Dogs", dogs_path, {class:'textHeader'}  %>
            </li>
            <% if user_signed_in? %>
              <% if current_user.try(:admin?) %>
                <li id=<%= @adminUsers%>>
                  <%= link_to "Users", admin_users_path, class:'textHeader' %>
                </li>
                <li id=<%= @adminResults%>>
                  <%= link_to "Results", admin_results_path, class:'textHeader' %>
                </li>
              <% end %>
              <li id=<%= @myProfile%>>
                <%= link_to "My profile", user_path(current_user), class:'textHeader' %>
              </li>
              <li id=<%= @myResults%>>
                <%= link_to "My results",results_path, class:'textHeader' %>
              </li>
              <li id=<%= @allExhibitions%>>
                <%= link_to "Exhibitions", exhibitions_path, class:'textHeader' %>
              </li>
              <li>
                <%= link_to "Sign out", destroy_user_session_path, method: :delete, class:'textHeader' %>
              </li>
              <li>
                <p class="navbar-text navbar-right textSign">Signed in as <%= current_user.email %></p>
```

```
            </li>
          <% else %>
            <li id=<%= @signUp%>>
              <%= link_to "Sign up", new_user_registration_path, class:'textHeader' %>
            </li>
            <li id=<%= @signIn%>>
              <%= link_to "Sign in", new_user_session_path, class:'textHeader' %>
            </li>
          <% end %>
        </ul>
      </div>
    </div>
  </div>
</nav>
```

Como podemos observar en el código anterior, en dicho código controlamos que si un usuario no está logueado se le muestra en el menú la imagen de “aefrbf” (que enlaza con el ranking), “Dogs”, “Sign up” y “Sign in”. Si el usuario está logueado ya sea como socio o como usuario registrado, se le muestra en el menú la imagen de “aefrbf” (que enlaza con el ranking), “Dogs”, “My profile”, “My results”, “Exhibitions” y “Sign out”. Y si el usuario es el administrador, se le muestra en el menú la imagen de “aefrbf” (que enlaza con el ranking), “Dogs”, “Users”, “Results”, “My profile”, “My results”, “Exhibitions” y “Sign out”.

De esta manera vamos ocultando todos los links necesarios a determinados usuarios de la aplicación. Todo el código se puede consultar en el CD entregado.

6.11.3. Reejecución de los tests

Al final de la iteración, comprobamos que hemos ocultado todos los links correctamente, ejecutando el nuevo test creado al comienzo de la iteración, en el cual comprobamos que hayamos ocultado esos links correctamente para determinados tipos de usuarios sin privilegios para visualizar determinados links de la aplicación. Y además, ejecutamos los tests modificados al comienzo de la iteración de los tests realizados en las iteraciones anteriores. Una vez ejecutamos todos estos tests y comprobamos que todos se ejecutan exitosamente, ya podemos dar por finalizada la última iteración del proyecto.

Capítulo 7

NORMATIVA Y LEGISLACIÓN

En este Trabajo de Fin de Grado, hay algo que es de vital importancia y es tener en cuenta la protección de datos y la seguridad. Por ello de las Normativas y legislaciones vigentes, lo principal que se ha tenido en cuenta es la Ley Orgánica de Protección de Datos (LOPD), que tiene como objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas y en especial, su honor e integridad personal y familiar.

En este proyecto hemos tenido en cuanta la protección de los datos personales, en aspectos como guardar en la base de datos las password de los usuarios encriptadas en md5, que usuarios que no estén registrados en la aplicación no puedan ver el correo de los usuarios registrados de la aplicación, etc. Todos los datos almacenados en la base de datos de esta aplicación son como mucho de nivel básico, ya que de los usuarios únicamente almacenamos el email y la contraseña.

En lo referente al ámbito informático, también tenemos el Real Decreto 994/1999 de Medidas de Seguridad de los ficheros automatizados que contengan datos de carácter personal de 11 de junio de 1999 (RMS) : Es un reglamento que desarrolla la Ley Orgánica 5/1992, de 29 de octubre, de Regulación del Tratamiento Automatizado de los Datos de Carácter Personal (LORTAD), regula las medidas técnicas y organizativas que deben aplicarse a los sistemas de información en los cuales se traten datos de carácter personal de forma automatizada.

La normativa y regulación de la informática en el ámbito europeo, hasta ahora centra el esfuerzo principalmente en regular el tema de los delitos 21 de noviembre de 2001. Este documento fue firmado por los representantes de cada país miembro del Consejo de

Europa, aunque su eficacia depende de su posterior refrendo por los órganos nacionales de cada país firmante. El “Convenio sobre la Ciberdelincuencia”permitió la definición de los delitos informáticos y algunos elementos relacionados con éstos, tales como “sistemas informáticos”, “datos informáticos”, o “proveedor de servicios”.

Estos delitos informáticos fueron clasificados en cuatro grupos:

- Delitos contra la confidencialidad, la integridad y la disponibilidad de los datos y sistemas informáticos.
- Delitos informáticos.
- Delitos relacionados con el contenido
- Delitos relacionados con infracciones de la propiedad intelectual y derechos afines

De estos 4 aspectos, en esta aplicación, el que más afecta es el primero, ya que controlamos el que los datos sean estén disponibles, sean confidenciales y también controlamos su integridad. En esta aplicación controlamos el que sólo los usuarios autorizados accedan a determinada información y no sea toda la información accesible para todos lo usuarios. Por lo que controlamos que los usuarios no autorizados no puedan acceder o modificar información de manera ilícita intentando acceder directamente desde la url a recursos que no están disponibles para determinados usuarios ni introduciendo formularios con campos de los que no deben disponer algunos tipos de usuarios.

Capítulo 8

CONCLUSIONES Y TRABAJOS FUTUROS

Una vez finalizado el Trabajo de Fin de Grado, una de las conclusiones que he obtenido es que desde mi punto de vista *Ruby on Rails* es una combinación muy buena de lenguaje-framework. En la asignatura DAW de 4º del grado, empezamos a ver lo que es este mundo de *Ruby on Rails*, pero con la realización de este trabajo he obtenido muchísimos nuevos conocimientos, que en dicha asignatura no obtuve sobre todo por los problemas de tiempo de las asignaturas en general. Gracias a esa asignatura fomenté mis ganas para seguir aprendiendo *Ruby on Rails* y realizar un proyecto en el que lo utilizara.

Si bien es cierto que en *Ruby on Rails* es complicado al principio entender la lógica, la orgranización de carpetas y el funcionamiento en general, con el paso del tiempo vas entendiendo cada vez más cosas y si se llega a conocer bien, llega a ser muy potente para el desarrollo de aplicaciones web.

Posibles trabajos futuros de este TFG podría ser la mejora de la vistas de la aplicación web realizada para la "Asociación Española para el Fomento de la Raza Bulldog Francés". Otro trabajo futuro sería el que las tablas de puntos se puedan introducir de manera gráfica a través de vistas y no como está ahora mismo que es introduciéndo el administrador dichas tablas de puntos directamente en la base de datos. Además, también como trabajo futuro sería hacer el porting de usuarios de la antigua aplicación a esta nueva aplicación, para que los usuarios no tuviesen que volverse a registrar, registrar a sus perros, etc.

De los trabajos futuros más importantes, estarían sobre todos los correspondientes a completar del todo la aplicación para poder sustituir a la actual aplicación *PHP* de la asociación. Dichos trabajos futuros serían la exportación de los usuarios, la exportación de los perros, etc. Además, de poder dotar a la aplicación de almacenamiento en la nube, gestión

de álbumes de fotos de los perros, fotografías de las exposiciones para que la web fuera un lugar de referencia.

Como hemos comentado en este proyecto no se han podido añadir todos estos aspectos debido a que en 300 horas no se podían realizar, pero en un período más largo de tiempo si se podría completar del todo esta aplicación para que sea operativa.

Como conclusión final, estoy contenta de haber sacado adelante esta aplicación web, que sea de utilidad para una asociación, y también estoy contenta del trato recibido por parte de mis dos tutores del TFG.

ANEXO A

CONFIGURACIÓN DE LA APLICACIÓN

En este capítulo, se explicarán los pasos seguidos para realizar la configuración de la aplicación. La aplicación será desplegada en *CentOS*, utilizando como herramientas de apoyo *Cucumber* y *RubyMine*. Primero, empezaremos por la definición de un gemset que llamaremos "ranking_app". Para definir dicho gemset, ejecutamos el siguiente comando en la consola de comandos:

```
$ rvm use 2.1.1@ranking_app --create
```

Una vez, tenemos definido el gemset `ranking_app`, procedemos a la instalación de *Rails* en dicho gemset. Para ello, ejecutamos el siguiente comando en la consola:

```
$ gem install rails -version=4.0.4
```

Ahora nos creamos un proyecto rails llamado `ranking_app`. Para ello, ejecutamos el siguiente comando:

```
$ rails new ranking_app
```

Creamos el fichero `ranking_app/.ruby-version` y ponemos lo siguiente para que utilice esa versión de ruby:

```
$ 2.1.1
```

También nos creamos el fichero `ranking_app/.ruby-gemset`, para que utilice el gemset indicado:

```
$ ranking_app
```

Modificamos el fichero `Gemfile` e introducimos todas las gemas que en principio hacen falta para la realización de este proyecto, quedando de la siguiente manera:

```
source 'https://rubygems.org'
ruby '2.1.1'
#ruby-gemset=ranking_app

# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.0.4'

# Use sqlite3 as the database for Active Record
group :development, :test do
  gem 'sqlite3', '1.3.8'
  gem 'rspec-rails', '2.13.1'
end

group :test do
  gem 'factory_girl_rails'
  gem 'selenium-webdriver', '2.35.1'
  gem 'capybara', '2.1.0'
  gem 'cucumber-rails', :require => false
  gem 'cucumber-websteps'
  gem 'database_cleaner'
  gem 'factory_girl'
  gem 'email_spec'
end


# Use SCSS for stylesheets
gem 'sass-rails', '~> 4.0.2'

# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'

# Use CoffeeScript for .js.coffee assets and views
gem 'coffee-rails', '~> 4.0.0'

# See https://github.com/sstephenson/execjs#readme for more supported runtimes
# gem 'therubyracer', platforms: :ruby

# Use jquery as the JavaScript library
gem 'jquery-rails'

# Turbolinks makes following links in your web application faster. Read more: https://g
gem 'turbolinks'

# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 1.2'
```

```
group :doc do
  # bundle exec rake doc:rails generates the API under doc/api.
  gem 'sdoc', require: false
end

group :production do
  gem 'pg', '0.15.1'
  gem 'rails_12factor', '0.0.2'
end


# Use ActiveModel has_secure_password
# gem 'bcrypt', '~> 3.1.7'

# Use unicorn as the app server
# gem 'unicorn'

# Use Capistrano for deployment
# gem 'capistrano', group: :development

# Use debugger
# gem 'debugger', group: [:development, :test]
```

A continuación, prodecemos a instalar las gemas ejecutando el siguiente comando:

```
$ bundle install --without production
```

La opción `--without production`, la utilizamos para que no se instalen gemas de producción, ya que no las necesitamos ahora mismo.

Procedemos ahora a la ejecución de los siguientes comandos:

```
$ rails generate rspec:install
$ rails generate cucumber:install
```

Usar `rspec` nos sirve para preparar los archivos necesarios para *RSpec* y usamos `cucumber` para preparar todos los archivos necesitados por *Cucumber* .

Modificamos el fichero `ranking_app/.gitignore` para configurar los ficheros que queremos que se suban al Git:

```
# See https://help.github.com/articles/ignoring-files for more about ignoring files.
#
# If you find yourself ignoring temporary files generated by your text editor
# or operating system, you probably want to add a global ignore instead:
#   git config --global core.excludesfile '~/.gitignore_global'

# Ignore bundler config.
/.bundle

# Ignore the default SQLite database.
```

```
/db/*.sqlite3
/db/*.sqlite3-journal

# Ignore all logfiles and tempfiles.
/log/*.log
/tmp

# Ignore other unneeded files.
database.yml
doc/
*.swp
*~
.project
.DS_Store
.idea
.secret
```

ANEXO B

DESPLIEGUE EN HEROKU DE LA APLICACIÓN

Desplegaremos la aplicación *Rails* en *Heroku* , que es una plataforma construída específicamente para el despliegue de *Rails* y otras aplicaciones Web.*Heroku* hace el despliegue de aplicaciones un poco más absequible, siempre y cuando el código fuente se encuentre bajo control de versiones con *Git* .

A continuación, iremos describiendo los pasos necesarios para el despliegue de la aplicación en *Heroku* .

Heroku utiliza la base de datos *PostgreSQL*, lo que significa que tenemos que añadir la gema *pg* para permitir a *Rails* comunicarse con *Postgres*. Además, debemos añadir la gema *rails_12factor*, que es utilizada por *Heroku* para servir activos estáticos como imágenes y hojas de estilo. Por todo ello, debemos tener en el `Gemfile` lo siguiente:

```
group :production do
  gem 'pg', '0.15.1'
  gem 'rails_12factor', '0.0.2'
end
```

Ejecutamos lo siguiente:

```
$ bundle install --without production
```

A continuación, debemos realizar la precompilación y hacer un commit en *Git* :

```
$ rake assets:precompile
$ git add .
$ git commit -m "Add precompiled assets for Heroku"
```

Ahora tenemos que crear y configurar una nueva cuenta de *Heroku* . El primer paso es registrarse para *Heroku* . Después de comprobar nuestro correo electrónico para completar la creación de la cuenta, instalamos el software necesario *Heroku* utilizando el Heroku Toolbelt.

A continuación, utilizamos el comando heroku para conectarse conectarse en el shell:

```
$ heroku login
```

Ahora vamos hasta el directorio que contiene la aplicación *Rails* y creamos una aplicación en *Heroku* :

```
$ cd ~/rails_projects/ranking_app
$ heroku create
```

El comando heroku crea un nuevo subdominio sólo para nuestra aplicación, disponibles para el inmediato visionado.

Desplegamos el código en *Heroku* ejecutando lo siguiente en el shell:

```
$ git push heroku master
```

Ahora pasamos a migrar manualmente la base de datos ejecutando lo siguiente:

```
$ heroku run rake db:migrate
```

Podemos renombrar nuestra aplicación en *Heroku* de la siguiente manera:

```
$ heroku rename ranking_app
```

Pasemos a asegurarnos de tener un banco de pruebas de ejecutar el tipo de proceso de la web:

```
$ heroku ps:scale web=1
```

Podemos comprobar el estado de los bancos de pruebas de la aplicación. El comando `heroku ps` lista los comandos de pruebas que están corriendo:

```
$ heroku ps
=== web (1X): `bin/rails server -p $PORT -e $RAILS_ENV`
web.1: idle 2014/07/26 13:08:27 (~ 1m ago)
```

Como se puede observar, se está ejecutando un banco de pruebas. Ahora podemos visitar la aplicación en nuestro navegador con `heroku open`:

```
$ heroku open
Opening ranking-app... done
```

Una vez que se ha desplegado con éxito, *Heroku* ofrece una buena interfaz para administrar y configurar la aplicación.

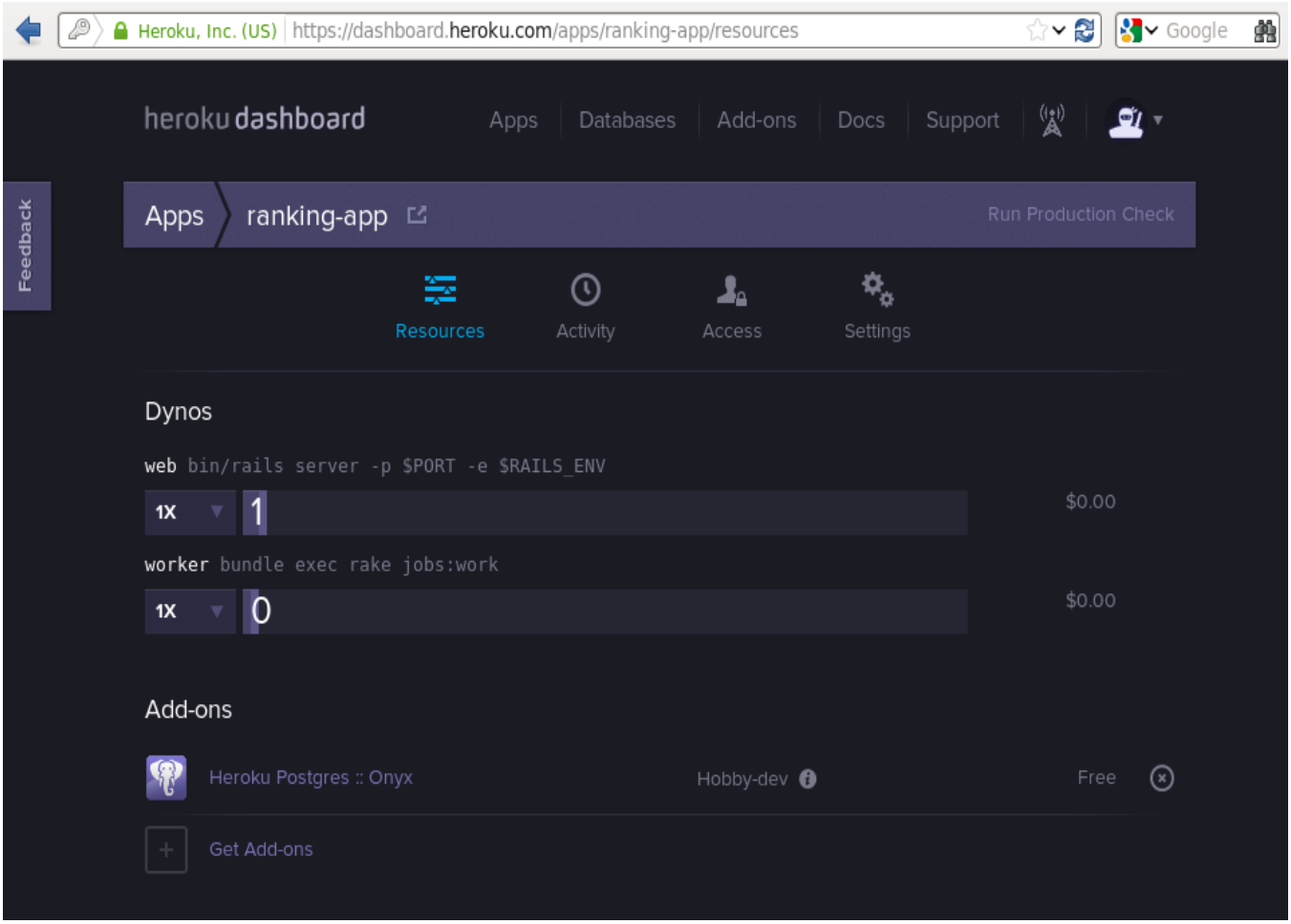

Figura B.1: Proyecto desplegado en Heroku

REFERENCIAS

[1] Hartl Michael, *Ruby on rails tutorial : learn Web development with rails* , Second edition, Addison-Wesley, 2013.

[2] Feduke Charles, *Instant RSpec test-driven development how-to learn RSpec and redefine your approach towards software development*, Packt Pub., 2013.

[3] `http://rubyonrails.org/`

[4] `http://guides.rubyonrails.org/`

[5] `http://codehero.co/`

[6] `http://railscasts.com/`

[7] `http://www.w3schools.com/`