UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Gestión web del cuaderno de un profesor de secundaria

Proyecto fin de carrera

Universidad de Las Palmas de Gran Canaria

Autora: Matilde Gómez Ojeda
Tutor: D. Carmelo Cuenca Hernández
Titulación: Ingeniero en informática

Dedicado a mis hermannos, padres y mi pareja Álvaro; y en especial a mi padre, gracias papá por tu entrega y dedicación

Proyecto fin de carrera

Matilde Gómez Ojeda

9 de febrero de 2017

Agradecimientos

Estoy muy agradecida en general por haber tenido la oportunidad de terminar este proyecto, pero debo nombrar a varias personas que han hecho posible la realización del mismo.

En primer lugar me dirijo a mi tutor D. Carmelo Cuenca Hernández que ha sido para mi un referente de dedicación, constancia y profesionalidad durante todos los años que lo conozco. Ha sido y es un pilar importannte en la educación universitaria de la ULPGC, gracias Carmelo por haberme guiado en todo momento y gracias no sólo por mi sino por todas las personas que has ayudado en tu labor como docente.

Para mí es imprescindible agradecer a mi pareja y compañero Alvaro la dedicación y paciencia que ha tenido conmigo durante la realización del mismo y todo el amor que me da día a día para superarme y conseguir la lucidez mental que hay que tener para llevar a cabo tareas como esta.

La persona que me ha hecho seguir en todo momento adelante con este proyecto y la ilusión de entregarlo ha sido mi padre, sin él y mi pensamiento de amor y gratitud estoy segura que esto no hubiera sido posible. Gracias papá por estar siempre ahí y ser un gran padre.

A muchos compañeros de estudios, que han sido después también compañeros de trabajo y lo mas importante amigos. Gracias sobre todo a ti Juan Pablo por ser mi amigo y compartir tantas cosas conmigo.

Agradecer además a lo docentes que me han dando clase durante mi formación en la ULPGC ya que creo que he tenido la fortuna de tener una formación de calidad gracias a ellos. En especial a D. Carmelo Cuenca, D. Octavio Mayor, D. Fran Santana y D. Abraham Rodríguez.

Por último agradecer a todos mis alumnos las experiencias que he vivido con ellos y que hacen que me guste mi trabajo y profesión. Por ellos he dirigido el proyecto hacia la educación.

Contents

1 Introducción **22**

1.1 ¿Qué es el cuaderno del profesor? 22

2 Estructura del documento **23**

3 Estado del arte **24**

3.1 Cuadernos en papel 24

3.1.1 Additio 25

3.1.2 Anro 26

3.1.3 Cuadernos como hojas de cálculo 26

3.2 Aplicaciones existentes 27

3.2.1 Aplicaciones de escritorio 27

3.2.2 Aplicaciones en la web 31

4 Objetivos del proyecto **32**

5 Planificación temporal **36**

6 Estudio de herramientas **36**

6.1 Herramientas de desarrollo 37

6.1.1 Gestores de contenido 38

6.1.2 Jomla! 43
6.1.3 Ruby on Rails 44
6.1.4 Django 46
6.1.5 CakePHP 48
6.2 Plataformas de despliegue en la nube 50
6.2.1 Heroku 51
6.2.2 Windows Azure 52
6.2.3 OpenShift 55
6.3 Herramientas de automatización de prueba del software 57
6.3.1 Cucumber 59
6.3.2 RSpec 60
6.3.3 Minitest 62
6.3.4 Concordion 63
6.3.5 Jasmine 65
6.4 Herramientas escogidas para el proyecto. Motivación . 66
6.4.1 Herramientas de desarrollo 66
6.4.2 Plataforma de control de la nube 67
6.4.3 Herramientas de pruebas 68

7 Metodología de desarollo 69

7.1 Programación extrema (XP - eXtreme Programming) . 70
7.2 Metodología TDD 72
7.3 Metodología BDD 77
7.4 Desarrollo ATDD 80

8 Recursos utilizados 83

8.1 Recursos hardware 84
8.2 Recursos software 85
8.2.1 Sistemas operativos 85
8.2.2 Navegador Web 87
8.2.3 Entorno de desarrollo integrado 87
8.2.4 Sistema de control de versiones 88
8.2.5 Editores de texto 89
8.2.6 Editores de bocetos - Mockups 89
8.2.7 StarUML 90
8.3 Tecnologías utilizadas 91
8.3.1 Ruby / Ruby on Rails 91
8.3.2 Ruby Version Manager 92
8.3.3 Active Record 92
8.3.4 HTML5 93

8.3.5 CSS3 94
8.3.6 Bootstrap 95
8.3.7 RSpec 95
8.3.8 Heroku 96
8.3.9 Cloudinary 96

9 Análisis **96**

9.1 Especificación de requisitos de usuario 98
9.1.1 Descripción 98
9.1.2 Evidencias del cuaderno del profesor 99
9.1.3 Modelo del dominio 101
9.2 Especificación de requisitos software 102
9.2.1 Descripción 102
9.2.2 Actores y casos de uso del sistema 102
9.2.3 Diagramas UML de los casos de uso 104
9.2.4 Listado de casos de uso 108
9.2.5 Pruebas del software 109

10 Diseño **113**

10.1 Arquitectura software 113
10.1.1 Introducción a la arquitectura utilizada 114

10.1.2 Patron MVC 115
10.1.3 Active Record 120
10.1.4 Diagrama de clases 122
10.1.5 Diagrama de clases de los controladores 123
10.1.6 Transferencia de Estado Representacional (REST) 123
10.2 Portal web 126
10.2.1 Mookups 126
10.2.2 Modelo de despliegue 130
10.3 Esquema de la base de datos 136

11 Implementación 137

11.1 Estructura de ficheros 137
11.2 I18n 140
11.3 Subida de imágenes con la gema “Carrieware” 141
11.4 Cloudinary 144
11.5 Paginación 145
11.6 Gemas utilizadas en el proyecto 147

12 Resultados y conclusiones 148

12.1 Resultados 148
12.2 Conclusiones 149

13 Trabajo Futuro **150**

13.1 Mejoras 150

14 Anexos **153**

14.1 Anexo I: Descripción de los casos de uso 153

14.1.1 Usuario - Ver información web 153

14.1.2 Usuario - Registrarse 154

14.1.3 Usuario - Contactar administrador 155

14.1.4 Profesor - Crear portada 156

14.1.5 Profesor - Consultar portada 157

14.1.6 Profesor - Modificar portada 158

14.1.7 Profesor - Eliminar portada 159

14.1.8 Profesor - Crear alumno 160

14.1.9 Profesor - Consultar alumno 161

14.1.10 Profesor - Modificar alumno 162

14.1.11 Profesor - Eliminar alumno 163

14.1.12 Profesor - Crear horario 164

14.1.13 Profesor - Consultar horario 165

14.1.14 Profesor - Modificar horario 166

14.1.15 Profesor - Eliminar horario 167

14.1.16 Profesor - Crear sistema de evaluación 168

14.1.17 Profesor - Consultar sistema de evaluación . . . 169

14.1.18 Profesor - Modificar sistema de evaluación . . . 170

14.1.19 Profesor - Eliminar sistema de evaluación 171

14.1.20 Profesor - Crear nota 172

14.1.21 Profesor - Consultar nota 173

14.1.22 Profesor - Modificar nota 174

14.1.23 Profesor - Eliminar nota 175

14.1.24 Profesor - Crear clase 176

14.1.25 Profesor - Consultar clase 177

14.1.26 Profesor - Modificar clase 178

14.1.27 Profesor - Eliminar clase 179

Lista de Figuras

1 Additio cuaderno de todas las clases 25

2 Additio cuaderno duplex 25

3 Anro 26

4 Cuaderno digital del profesorado - Ramón José Gavilán 27

5 Gesticlas - Página principal 27

6 Gesticlas - El cuaderno 28

7 Labeldroid - Desarrollo de software 28

8 Labeldroid - Cuaderno del profesor - Descripción 29

9 Labeldroid - Cuaderno del profesor - Aplicación 29

10 Idoceo - Página principal 30

11 Idoceo - Manual 30

12 Idoceo - Cuaderno 31

13 Evalua - Información 31

14 Evalua - Aplicación - Miraya del Mar 32

15 Drupal 39

16 Joomla 39

17 WordPress 40

18 Joomla! 43

19 Ruby on Rails 44
20 Django 46
21 CakePHP 48
22 Heroku 51
23 Windows Azure 52
24 OpenShift 55
25 Cucumber 59
26 RSpec 60
27 Concordion 63
28 Jasmine 65
29 eXtreme Programming 70
30 Esquema ciclo TDD 74
31 Esquema BDD + TDD 78
32 Esquema ciclo ATDD 81
33 Equipo 84
34 Ubuntu 86
35 Windows 10 86
36 Mozilla Firefox 87
37 RubyMine 87
38 Git 88

39 LaTeX 89

40 Kile 89

41 Pencil 90

42 StartUML 90

43 Ruby 91

44 RVM - Ruby Version Mannager 92

45 HTML5 93

46 CSS3 94

47 Bootstrap 95

48 Modelo del dominio 101

49 Diagrama de caso de uso del sistema 103

50 Diagrama de caso de uso del actor usuario 104

51 Diagrama de caso de uso del actor profesor 104

52 Diagrama de caso de uso del actor profesor - Gestionar portada 105

53 Diagrama de caso de uso del actor profesor - Gestionar alumno 105

54 Diagrama de caso de uso del actor profesor - Gestionar horarios 106

55 Diagrama de caso de uso del actor profesor - Gestionar el sistema de evaluación 106

56 Diagrama de caso de uso del actor profesor - Gestionar notas . . . 107

57 Diagrama de caso de uso del actor administrador . . . 107

58 Arquitectura web . . . 114

59 Diagrama de la arquitectura MVC . . . 116

60 Estructura de las aplicaciones en Ruby o Rails . . . 118

61 Diagrama detallado de MVC para Rails . . . 119

62 Mapeo Objeto Relacional (ORM) . . . 121

63 Diagrama de clases . . . 122

64 Diagrama de clases de los controladores . . . 123

65 Recursos en las webs y sus URLs . . . 124

66 Mookups página inicio . . . 126

67 Mookups clases . . . 127

68 Mookups cuaderno . . . 127

69 Mookups alumno . . . 128

70 Mookups portada . . . 129

71 Mookups horario . . . 129

72 Mookups fotos . . . 130

73 Esquema general de arquitecura Cloud Computing . . . 131

74 Capas en Cloud Computing . . . 135

75 Esquema de la base de datos 136

76 Estructura de directorios 138

Lista de Tablas

1 Listado de actores y roles 103

2 Listado de actores y roles 109

3 Ejemplo de rutas REST 125

4 Descripción caso de uso: Usuario - Ver información web 153

5 Descripción caso de uso: Usuario - Registrarse 154

6 Descripción caso de uso: Usuario - Contactar administrador 155

7 Descripción caso de uso: Profesor - Crear portada . . . 156

8 Descripción caso de uso: Profesor - Consultar portada . 157

9 Descripción caso de uso: Profesor - Consultar portada . 158

10 Descripción caso de uso: Profesor - Eliminar portada . 159

11 Descripción caso de uso: Profesor - Crear alumno . . . 160

12 Descripción caso de uso: Profesor - Consultar alumno . 161

13 Descripción caso de uso: Profesor - Modificar alumno . 162

14 Descripción caso de uso: Profesor - Eliminar alumno . 163

15 Descripción caso de uso: Profesor - Crear horario . . . 164

16 Descripción caso de uso: Profesor - Consultar horario . 165

17 Descripción caso de uso: Profesor - Modificar horario . 166

18 Descripción caso de uso: Profesor - Eliminar horario . . 167

19 Descripción caso de uso: Profesor - Crear sistema de evaluación 168

20 Descripción caso de uso: Profesor - Consultar sistema de evaluación 169

21 Descripción caso de uso: Profesor - Modificar sistema de evaluación 170

22 Descripción caso de uso: Profesor - Eliminar sistema de evaluación 171

23 Descripción caso de uso: Profesor - Crear nota 172

24 Descripción caso de uso: Profesor - Consultar nota . . . 173

25 Descripción caso de uso: Profesor - Modificar nota . . . 174

26 Descripción caso de uso: Profesor - Eliminar nota . . . 175

27 Descripción caso de uso: Profesor - Crear clase 176

28 Descripción caso de uso: Profesor - Consultar clase . . 177

29 Descripción caso de uso: Profesor - Modificar clase . . 178

30 Descripción caso de uso: Profesor - Eliminar clase . . . 179

1 Introducción

El cuaderno del profesor es una de las herramientas esenciales con las que trabaja el docente durante todo un curso escolar. Se trata de un cuaderno que se utiliza a diario y que acompaña al docente durante toda su jornada laboral.

1.1 ¿Qué es el cuaderno del profesor?

Se trata de un **cuaderno donde se toman apuntes** de las notas de los exámenes y otras evidencias, datos del alumnado, control de asistencia, etc. En definitiva, donde se registra el trabajo que se realiza a lo largo del curso. Es una herramienta fundamental a la hora de evaluar. Se utiliza para:

- Llevar un **registro de todas las tareas** (realizadas y por hacer), exámenes, actitudes del alumno, observaciones durante el proceso de las clases didácticas.
- Hacer **informes individuales y colectivos** de calificaciones, del rendimiento académico, faltas de asistencia, etc., de los alumnos.
- **Planes de trabajos individualizados** (con adaptaciones colectivas o individuales).
- **Elección del método de evaluación** que más convenga.
- **Creación del horario** del docente.
- **Registro de todas las reuniones** (de departemento, de coordinación pedagógica, con tutores, etc.). Registro de todos los profesores que imparten clases a nuestros alumnos.
- Podemos añadir un **comentario a cada tarea realizada por cada alumno** (tanto de los exámenes como las tareas realizadas en casa).
- También podemos llevar un **registro de las entrevistas con los padres de los alumnos**.

- La **ubicación de todos los alumnos** (con nombre y foto) de nuestra clase en un mapa esquemático, donde podemos ubicar donde queramos a cada uno de nuestros alumnos.

2 Estructura del documento

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

Una vez realizada la introducción, en primer lugar veremos un estudio del estado del arte actual. Se analizarán aplicaciones actuales que están en funcionamiento y que realmente son utilizadas por los docentes.

Después se presentarán los objetivos fundamentales del proyecto y la planificación temporal del mismo.

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

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

El siguiente apartado que nos vamos a encontrar es el referente al análisis del sistema. Hablaremos de nuestro sitio web en prototipo, hablaremos de las pruebas realizadas en la fase previa al desarrollo de la plataforma, y especificaremos los requisitos de usuario y del software.

En lo que respecta al diseño, estudiaremos la arquitectura del sistema, que veremos que será una arquitectura basada en

Modelo-Vista-Controlador. Por otro lado, veremos opciones de diseño del portal web, presentando algunos mockups de cómo se desearía que se presentase de cara al cliente dicho portal. También estudiaremos el diseño y la estructura de la base de datos del sistema.

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

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

En los anexos del documento empezaremos mostrando los detalles de los casos de uso citados en el apartado de análisis.

3 Estado del arte

Antes del desarrollo propiamente dicho del proyecto, se ha hecho un estudio de algunos ejemplos los cuadernos del profesor que se utilizan por el profesorado.Para ello agruparemos el estudio en dos categorías: cuadernos en papel y aplicaciones existentes.

3.1 Cuadernos en papel

Los cuadernos de papel han sido antes de la llegada de lo digital la manera tradicional de anotar los profesores. Distintas editoriales venden en librerías especializadas su productos para que se utilicen por el profesorado.

3.1.1 Additio

http://www.additio.com/

Figura 1: Additio cuaderno de todas las clases

Figura 2: Additio cuaderno duplex

3.1.2 Anro

http://www.anroshop.es/es/oficina-libretas-y-organizacion/6911-cuaderno-del-profesor-s-nombre.html

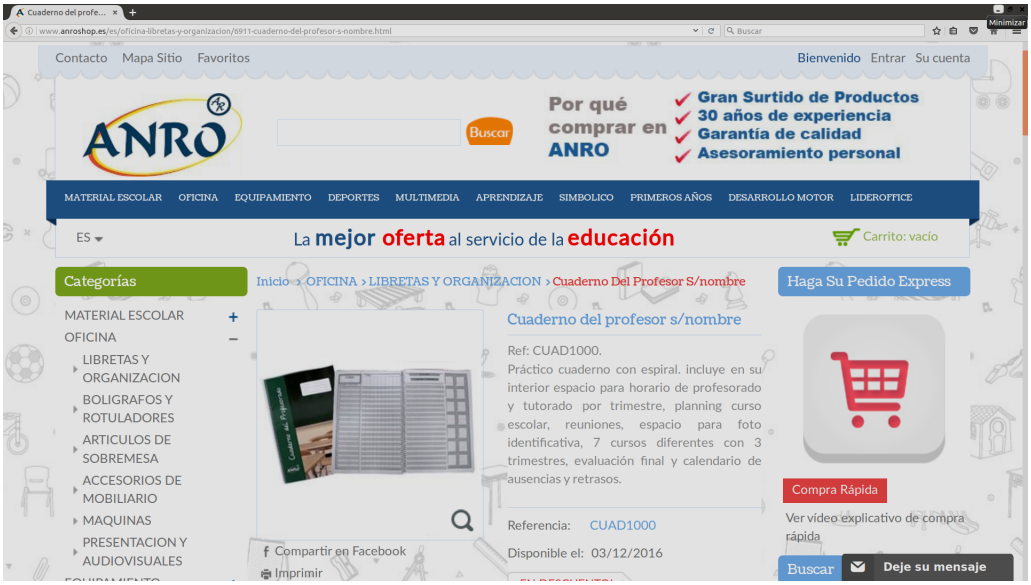

Figura 3: Anro

3.1.3 Cuadernos como hojas de cálculo

http://cuadernodigitaldelprofesorado.blogspot.com.es/

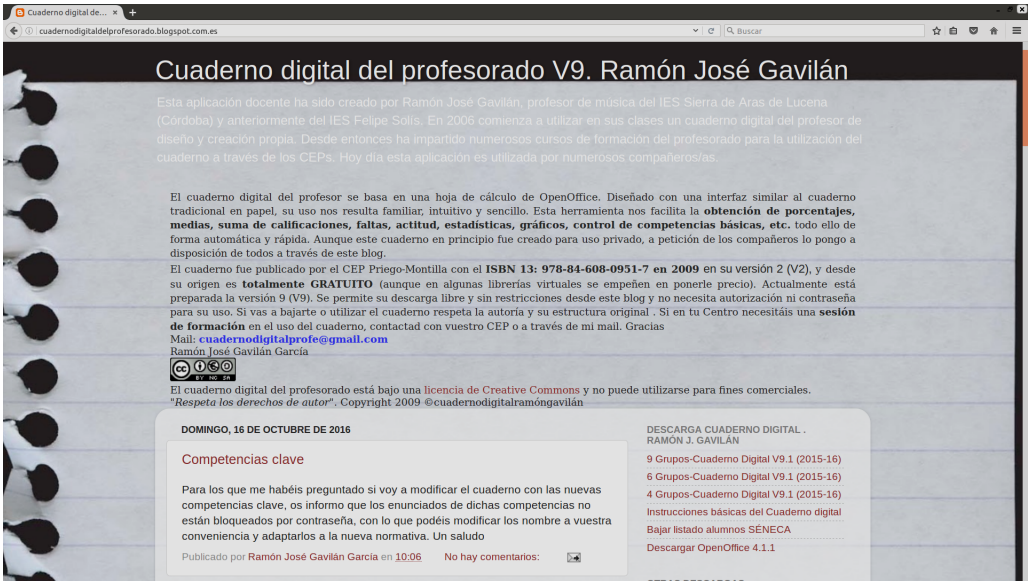

Figura 4: Cuaderno digital del profesorado - Ramón José Gavilán

3.2 Aplicaciones existentes

3.2.1 Aplicaciones de escritorio

Gesticlas

https://alfortiz.wordpress.com/

Figura 5: Gesticlas - Página principal

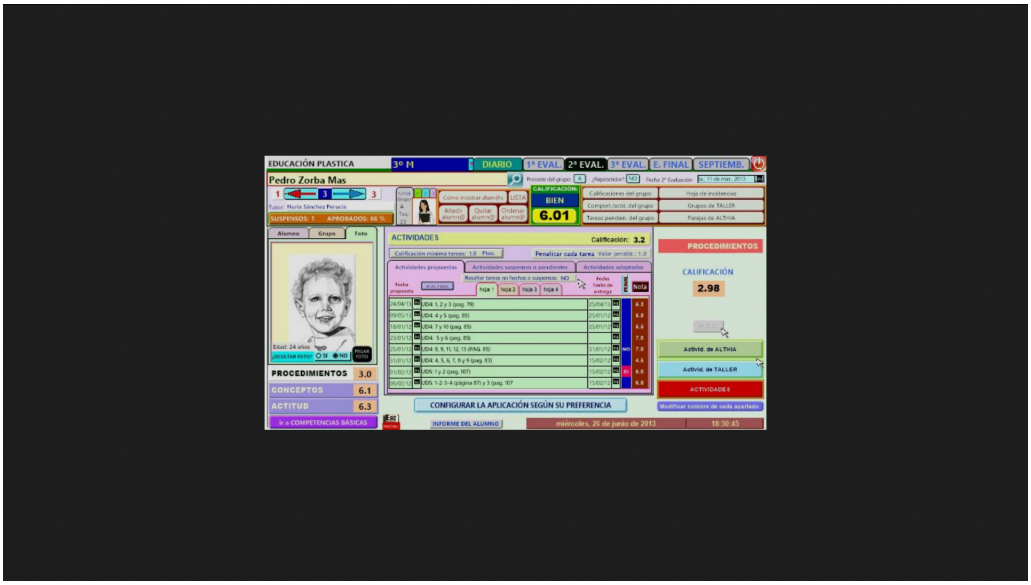

Figura 6: Gesticlas - El cuaderno

Labeldroid - Cuaderno del profesor http://labeldroid.com/

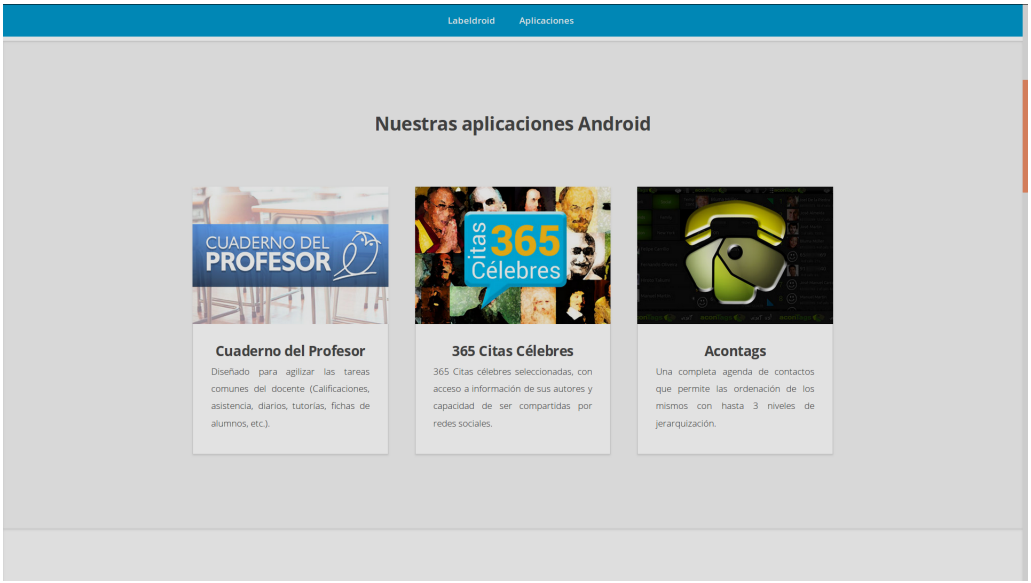

Figura 7: Labeldroid - Desarrollo de software

Figura 8: Labeldroid - Cuaderno del profesor - Descripción

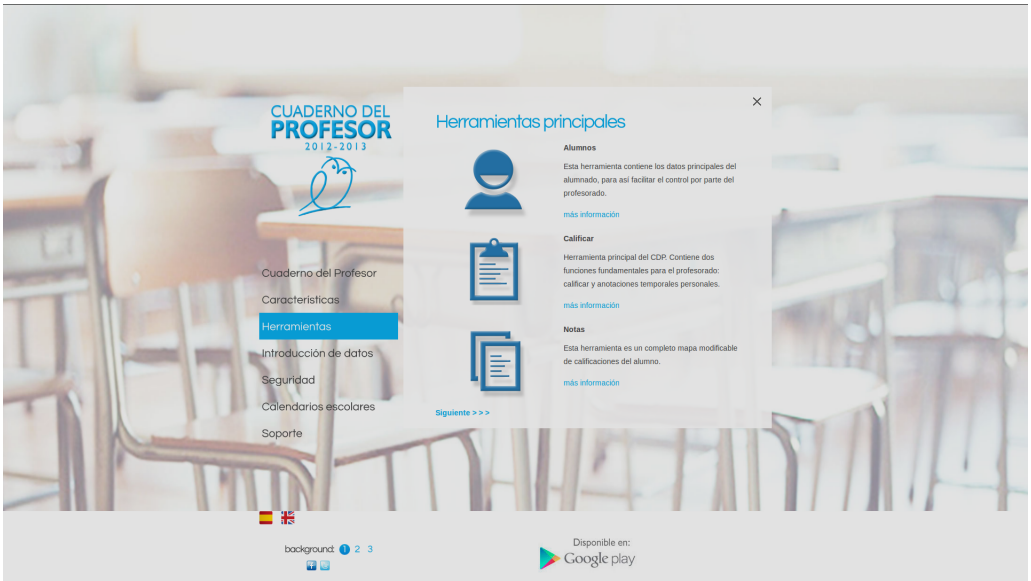

Figura 9: Labeldroid - Cuaderno del profesor - Aplicación

Idoceo

http://www.idoceo.es/index.php/es/

Figura 10: Idoceo - Página principal

Figura 11: Idoceo - Manual

Figura 12: Idoceo - Cuaderno

3.2.2 Aplicaciones en la web

EVALÚA

http://www.e-valua.org/

Figura 13: Evalua - Información

https://www.proevalua.net/v/index.aspx?ies=29700345

Figura 14: Evalua - Aplicación - Miraya del Mar

Por todo lo visto anteriormente podemos concluir que el cuaderno del profesor es una herramienta que esta a diario en las aulas y facilita el trabajo del docente. Esto únido a las nuevas tecnologías y el uso de ellas por parte de gran mayoría del profesorado hace que el desarrollo de una aplicación de estas características sea de nuestro interés.

4 Objetivos del proyecto

En este punto del documento, vamos a analizar los objetivos que se quieren cumplir con la realización del proyecto.

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

la aplicación puede abarcar una vez acabada.

Uno de los objetivos es hacer un estudio de las tecnologías de desarrollo web, tecnologías de despliegue en la nube y desarrollar parte de la aplicación del cuaderno del profesor.

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

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

se ha decidido incluir este tipo de metodologías en el desarrollo de la aplicación, de modo que el aprendizaje y manejo de metodologías de desarrollo guiadas por pruebas entre dentro de principales objetivos del proyecto.

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

El sector de la educación secundaria, es un sector en el que se invierte mucho tiempo en el seguimiento académico del alumno ya que se intenta en la medida de lo posible que sea personalizado y que se logren los objetivos con cada alumno. Hay que tener en cuenta que en los últimos años por un lado se ha aumentado el ratio de alumnos por profesor y por otro lado que las nuevas tecnologías han llegado a todo el conjunto del profesorado y al mundo en general, por todo esto se puede apreciar que hay un mercado en el que se puede fijar uno para plantearse un negocio.

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

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

De modo funcional, el objetivo se ha acotado a conseguir un sistema capaz de llevar el seguimiento en un conjunto de materias de sus alumnos por parte de su profesor. Para ello se deberá tener un sistema capaz de soportar usuarios (profesores). De este modo, un usuario registrado podrá crear un conjunto de asignaturas, matricular

alumnos en esas asignaturas y llevar un seguimiento de los mismos muy probado.

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

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

5 Planificación temporal

Esta es la planificación temporal propuesta al inicio del proyecto:

- **Fase 1: Análisis 220 horas**
  - Actividad 1.1 Documentación y herramientas
  - Actividad 1.2 Análisis de requisitos de usuario
  - Actividad 1.3 Análisis de requisitos de software
- **Fase 2: Diseño 160 horas**
  - Actividad 2.1 : Estudio herramientas
  - Actividad 2.2 : Diseño de módulo de bases de datos
  - Actividad 2.3 : Diseño de módulos de aplicación
  - Actividad 2.4 : Diseño de módulos interfaz Web
  - Actividad 2.5 : Diseño de instalación
- **Fase 3: Implementación 520**
  - Actividad 3.1 : Implementación de módulos de aplicación
  - Actividad 3.2 : Implementación de interfaz
  - Actividad 3.3 : Implementación de ensamblaje de módulos
- **Fase 4 : Pruebas del software 80**

Hay que tener en cuenta que en esta planificación inicial no se tuvo en cuenta la realización de la memoria.

6 Estudio de herramientas

El proyecto va a estar desarrollado con el framework Ruby on Rails pero con la finalidad de entender y estudiar diversos caminos por los que se podría haber optado para la evolución del proyecto **"Gestión web del cuaderno de un profesor de secundaria"**, se pasa a continuación a especificar y analizar distintas herramientas que pueden complacer para un eficiente desarrollo de dicho proyecto, así como las

necesidades que van a cubrir las mismas y las aportacines que podrán ofrecer.

Se estudiará cada una de las herramientas individualmente, excepto tres en particular (en la siguiente sección se explica el porqué de agrupar estas tres en un mismo conjunto), y se agrupan en **3 grandes bloques**. De modo que por un lado se estudia y compara las herramientas disponibles de desarrollo; por otro lado las plataformas que se pueden utilizar para el control de la nube; y para terminar se agrupa en otro bloque herramientas de testeo que se consideran que pueden ser competitivas para incluir en el proyecto. Al final de este apartado, y para concluir el bloque, se hace una valoración global y una explicación del porqué de las elecciones tomadas para realizar el proyecto.

6.1 Herramientas de desarrollo

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

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

- Ser de fácil utilización e intuitivas para administradores y usuarios.
- Que sean adaptables a los distintos requerimientos que puedan surgir durante el desarrollo y que no estén preestablecidos desde un primer momento.
- Ser de uso extendido, compatibles, y con una suficiente documentación disponible.
- Dichas herramientas deben de ser fiables y seguras.

- Que estén basadas en lenguajes de programación web de uso extendido.
- Que sea una herramienta que aporte conocimiento al alumno en el proyecto de fin de carrera.

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

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

6.1.1 Gestores de contenido

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

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

De este modo, si eres un desarrollador web profesional y tienes

un nivel de codificación avanzado, se recomienda la utilización de **Drupal**. Páginas como la de The Economist, MTV o The White House están construidas sobre esta plataforma.

Figura 15: Drupal

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

Figura 16: Joomla

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

Figura 17: WordPress

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

Drupal

- **Ventajas**
  - *Extremadamente flexible* – Permite desarrollar desde portales sencillos hasta un poderoso back-end que puedan soportar a millones de usuarios.
  - *Amigable con el desarrollador* – La instalación básica es esencial, animando a desarrolladores a crear sus propias soluciones.
  - *Grandes capacidades* ***SEO*** – Drupal fue diseñado para ser amigable con los motores de búsqueda.
  - *Amigable con las Empresas* – Fuerte control de versiones y capacidades **ACL** (Access Control List).
  - *Estabilidad* – Es escalable sin esfuerzo y estable incluso con miles de usuarios a la vez.
- **Inconvenientes**
  - *Gran curva de aprendizaje* – Cambiar de un **CMS** simple a Drupal puede ser muy complicado.
  - *Falta de plugins gratuitos* – La mayoría de los buenos plugins (llamados "módulos" en Drupal) no son gratuitos.
  - *Falta de temas* – Una instalación básica de Drupal es muy básica, y la falta de temas no mejora las cosas.

Joomla!

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

Wordpress

- **Ventajas**
  - *Múltiples autores.*
  - *Gran librería de Plugins* – Hay cientos de miles de plugins con infinidad de posibilidades.
  - *Amigable con el usuario* – La interfaz de uso es muy fácil de usar e intuitiva.
  - *Grandes capacidades **SEO*** – Con ciertos plugins se puede empezar a bloguear directamente sin preocuparse del **SEO** de la página.
  - *Fácil personalización* – El sistema de temas de WordPress está diseñado para una fácil personalización.
  - *Flexibilidad* – Se puede hacer que WordPress haga casi cualquier cosa.

- **Inconvenientes**
  - *Seguridad* – WordPress es a menudo diana de hackers y no es muy seguro.
  - *Incompatibilidad con plugins más antiguos* – Se sacan continuas actualizaciones sin arreglar problemas o agujeros de seguridad de las anteriores y a menudo son incompatibles con plugins más antiguos.
  - *Capacidades de gestión de contenido limitadas* – WordPress fue originalmente diseñado como plataforma de blogueo, por lo que esto ha afectado a su capacidad para manejar grandes cantidades de contenido.

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

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

6.1.2 Jomla!

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

Figura 18: Joomla!

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

- Hay muy buena documentación y soporte.
- Sigue el modelo **MVC**, lo que permite interactuar directamente con la parte vista de Joomla!, un desarrollo de componentes módulos y plugins basados en la arquitectura base del CMS, actualización de los elementos para cambios de versión con la plataforma funcionando.
- Excelente variedad de herramientas de personalización de contenidos, además de una gran disposición de módulos

modificables. Las posibilidades de ampliación de Joomla! no tiene límites.

- Limpieza y simplicidad en el desarrollo y administración: permite crear páginas ilimitadas y editarlas desde un sencillo editor, con lo que los contenidos de dichas páginas quedan organizados eficientemente en secciones y categorías. Además permite una cómoda administración, incluyendo la propia administración de usuarios, imágenes, módulos...
- Gran posibilidad de internacionalización del sitio web. Posee un soporte completo de **UTF-8**, soporte **RTL**, traducción de extensiones utilizando archivos **INI**.

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

6.1.3 Ruby on Rails

Figura 19: Ruby on Rails

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

Las principales características que presenta este framework son las siguientes:

- Incluye soporte para:
  - AJAX (Ajax on Rails).
  - Servicios Web (Action Web Service).
  - Patrón de diseño Modelo Vista Controlador.
  - Mapeo automático de objetos a modelo relacional.
  - RJS (Ruby to JavaScript compiler).
- Sigue la filosofía de desarrollo **DRY-COC**:
  - Don't Repeat Yourself (**DRY**).
    - Cuyo significado inluye que las definiciones deben hacerse una sola vez. Sigue un patrón de diseño Active Record, en el que, entre otras cosas, las definiciones de las clases no necesitan especificar los nombres de las columnas.
  - Convention Over Configuration (**COC**).
    - **COC** nos indica que el programador sólo necesita definir aquella configuración que no es convencional. Por ejemplo, en Ruby, una clase User en el modelo, corresponde con la tabla users de la base de datos.

- Rails incluye el concepto de migraciones, que nos permite definir o modificar la estructura de nuestra base de datos desde Ruby, y además de una manera independiente del motor de base de datos elegido.
- Sigue el paradigma de la arquitectura Modelo Vista Controlador (**MVC**), con el que se separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar las comunicaciones.

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

6.1.4 Django

Figura 20: Django

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

Django fue desarrollado por Adrian Holovaty, Simon Willison, Jacob Kaplan-Moss y Wilson Miner mientras trabajaban en World Online, y originalmente se utilizó para administrar tres sitios web de noticias: The Lawrence Journal-World, lawrence.com y KUsports.com.

Fue expuesto al público bajo licencia BSD (Berkeley Software Distribution) en el 2005. Su nombre Django viene haciendo alusión al guitarrista de jazz Django Reinhardt.

Django es muy utilizado en la administración de páginas de noticias, ya que proporciona una serie de características que facilitan el desarrollo rápido de páginas orientadas a contenidos. Por poner un ejemplo, en lugar de obligar a que los desarrolladores escriban controladores y vistas para las áreas de administración de la página, Django proporciona una aplicación incorporada para la administración de los contenidos, que puede incluirse como parte de cualquier página hecha con Django y permite administrar varias páginas desarrolladas con Django a partir de una misma instalación. Esta aplicación permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las accionas realizadas sobra cada uno, y proporciona una interfaz de administración de usuarios y grupos de usuarios. Las características principales de Django las listamos a continuación:

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

traducciones incorporadas de la interfaz de administración.

Vista la descripción general de en qué consiste Django, y qué características principales tiene, llegamos a la conclusión de que es bastante parecido a los otros dos frameworks analizados, con la peculiaridad de que este framework, por lo que parece, está más orientado a la creación de páginas de noticias que a otra cosa. Es por ello que Django tiene que ser otro framework a ser considerado a la hora de desarrollar una aplicación web.

6.1.5 CakePHP

Figura 21: CakePHP

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

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

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

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

Entre las particularidades de CakePHP podemos destacar las siguientes características:

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

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

del proyecto, por el principal motivo de la aportación que nos puede proporcionar en el sentido de aprendizaje.

6.2 Plataformas de despliegue en la nube

Para evitar futuros problemas de servidores con recursos estáticos, como el sobredimensionamiento, el crecimiento inesperado de usuarios, la necesidad de inversiones extras en servidores o problemas derivados de la utilización de dichos servidores, se decidió utilizar la nube (en la sección 10.2.2 de este mismo documento se detallará el porqué). Para la correcta utilización de la nube, necesitaremos una plataforma que nos permita controlar las posibilidades que nos ofrece la misma.

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

- Ser de fácil utilización y en parte intuitiva para el administrador.
- Que sean de uso extendido, compatibles, y con suficiente documentación.
- Que las plataformas sean seguras y fiables.
- Que nos proporcionen servicios de aprovisionamiento, almacenamiento de datos,y de soporte.
- Ser competitivas y transparentes, permitiendo buen control sobre las herramientas que nos ofrecen.
- Soportar los lenguajes que se vayan a escoger para programación.
- Que presenten una modalidad de plataforma como servicio (**PaaS**).

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

6.2.1 Heroku

Figura 22: Heroku

Heroku es un servicio de Hosting en la nube (el tiempo de procesamiento y el almacenamiento se le renta a un tercero), que soporta varios lenguajes. Es propiedad de Salesforce.com y es una de las primeras plataformas en la nube. Fue desarrollada en el 2.007, con el objetivo de soportar solamente el lenguaje de programación Ruby. Posteriormente se ha extendido a Java, Node.js, Python, Clojure, Scala y otros.

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

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

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

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

- Heroku es una plataforma como servicio.

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

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

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

6.2.2 Windows Azure

Figura 23: Windows Azure

Windows Azure, al igual que Heroku, es una plataforma ofrecida como servicio y, en este caso, Azure está alojada en los Data Centers de Microsoft. Sus inicios se remontan a cuándo se anunció en el

Professional Developers Conference de Microsoft (PDC) del 2008 en su versión beta, fase que pasó a comerciable el 1 de enero del 2009.

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

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

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

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

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

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

(blobs) que ofrecen un tipo de tablas con un lenguaje de consulta simple.

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

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

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

6.2.3 OpenShift

Figura 24: OpenShift

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

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

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

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

Algunas de las características de las que presume OpenShift son:

- **Velocidad:** Reduce el tiempo necesario para construir y desplegar nuestras aplicaciones, permitiéndonos así centrarnos en nuestro

código, en lugar de la administración de la infraestructura.

- **Versatilidad:** Ofrece la más amplia gama de lenguajes de programación, frameworks y tiempos de ejecución, incluyendo Java EE6 con JBoss EAP.
- **Código abierto:** Mediante su plataforma de código abierto nos ofrece asegurar la portabilidad de las aplicaciones.
- **Fácil manejo:** Posee herramientas de desarrollo integradas y una interfaz intuitiva, permitiéndonos trabajar rápidamente.

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

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

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

OpenShift parece una buena solución para subir nuestra aplicación a la nube: Sigue una visión de aplicación de código abierto, provee un

muy buen servicio de forma gratuita y nos ofrece facilidad de uso y buena documentación para su utilización.

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

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

6.3 Herramientas de automatización de prueba del software

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

- Cada una de ellas describe un escenario de ejecución o uso del sistema desde la perspectiva del cliente.
- Pueden estar asociadas a requisitos funcionales o no funcionales.
- Un requisito tiene una o más pruebas de aceptación asociadas.

- Una prueba de aceptación puede tener infinitas instanciaciones (ejecuciones con valores concretos). El diseño de las instanciaciones y su aplicación es trabajo del tester.

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

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

- Debe proporcionar soporte de testeo para la herramienta de desarrollo que se utilice.
- De manejo no complejo.
- Que se permita unas definiciones de pruebas que sean de una forma que el cliente pueda comprender, entender y evaluar.
- Que nos proporcione pruebas fiables.

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

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

6.3.1 Cucumber

Figura 25: Cucumber

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

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

Podemos utilizar Cucumber para validar código escrito en Ruby, Java, C#, Python... Solamente requiere un pequeño conocimiento en Ruby, por lo que aunque lo que se pretenda desarrollar no vaya a estar en Ruby, no se debe dejar de tener en cuenta esta herramienta de testeo.

Las características principales de Cucumber son las siguientes:

- Cucumber utiliza la sintaxis de Gherkin para definir las especificaciones.
- Es posible escribir los escenarios en distintos idiomas.
- Cucumber incorpora las etiquetas necesarias para relacionar los pasos con el código en muchos idiomas.

- Se evitan confusiones al transformar especificaciones en funcionalidades o tests unitarios.
- La estructura de definición de los tests es muy intuitiva y a la vez razonable.

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

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

6.3.2 RSpec

Figura 26: RSpec

RSpec es una framework para realizar tests en un desarrollo basado en el comportamiento **(BDD)**, y diseñada para programadores de Ruby. **BDD** es un acercamiento al desarrollo del software que combina un desarrollo basado en tests (**TDD** – Test-Driven Development) con

un diseño guiado por el dominio (**DDD** – Domain Driven Design) y una planificación basada en test de aceptación (**ATD** Planning – Acceptance Test-Driven Planning). RSpec está pensado para ayudar al desarrollador a cumplir la parte del **TDD**, enfocándose en aspectos de documentación y diseño del **TDD**.

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

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

Las características más sobresalientes de este framework son las siguientes:

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

- Permite especificaciones de forma legible sin ser engorrosas de escribir y nos permite realizar la misma tarea de varias formas.

Vistas las características que nos ofrece RSpec, además debemos mencionar algunos de los aspectos negativos que se han podido encontrar en el estudio de este framework de testeo. Por lo que se ha podido estudiar, no es un framework recomendable para empezar desde cero con la metodología de testeo, ya que se menciona de que es algo complejo de aprender y dominar, aunque una vez se ha aprendido es bastante claro y limpio.

Por lo analizado, RSpec es otra herramienta de prueba para un desarrollo guiado por pruebas muy a tener en cuenta, ya que los pros de este framework superan con creces a los contras. De hecho, lo peor que se ha podido estudiar acerca de RSpec es su complejidad a la hora de aprenderlo, algo con lo que podríamos coexistir si realmente la herramienta nos proporciona buenas funcionalidades.

6.3.3 Minitest

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

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

MiniTest/Unit es un pequeño e increíblemente rápido framework de testeo unitario, el cual posee un rico conjunto de declaraciones para

hacer las pruebas limpias y legibles.

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

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

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

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

6.3.4 Concordion

Figura 27: Concordion

Otro framework bastante utilizado y con buena aceptación en el mundo del desarrollo guiado por pruebas es el **Concordion**. Este

framework nos permite construir y ejecutar test de aceptación. Aporta visibilidad a los requisitos definidos por el propietario del producto que el desarrollador ha plasmado en las pruebas de aceptación.

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

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

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

Algunas de las características que proporciona este framework son:

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

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

Otra de las utilidades en las que puede usarse Concordion es en la creación de una documentación viva unida a los cambios de código. Si

nos planteamos la pregunta ¿Son los tests la auténtica documentación actualizada de nuestro código?, muchos desarrolladores responderían que sí. Por tanto... ¿Por qué no usarlos por medio de una fachada HTML cómo documentación?

6.3.5 Jasmine

Figura 28: Jasmine

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

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

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

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

A la hora de elegir un framework de testeo para JavaScript debemos tener en cuenta varias consideraciones. Jasmine es simplemente uno de los muchos participantes en el mundo del testeo de aplicaciones con JavaScript. Algunos utilizan *qUnit*, framework no oficial de testeo para *JQuery*, o YUI Test, el cual es parte de la librería YUI

de Yahoo. De hecho, la mayoría de las librerías JavaScript tienen su propio framework de testeo. Estos deberían ser considerados si vamos a ser parciales a una biblioteca específica JavaScript. Por otro lado, podríamos querer tener clases de prueba que no estén vinculadas a otro framework. Esto es algo importante en Jasmine. Otra consideración a tener en cuenta es que Jasmine se basa en el navegador o requiere la utilización de un entorno de pruebas de software basadas en la web, como Selenium. Hay otros como *RhinoUnit* que no requieren un navegador para funcionar, haciendo las pruebas mucho más automatizadas.

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

6.4 Herramientas escogidas para el proyecto. Motivación

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

6.4.1 Herramientas de desarrollo

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

queremos desarrollar.

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

Por otro lado no debemos olvidarnos que queremos un proyecto basado en pruebas, por lo que también hay que mirar la compatibilidad con herramientas que permitan realizar tests. Rails sigue una metodología de desarrollo guiado por tests, ya que el mundo RoR se guía bajo la metodología del testeo.

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

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

6.4.2 Plataforma de control de la nube

En nuestro proyecto en particular, la plataforma que más se adecúa a nuestro plan es Heroku. Heroku, como hemos mencionado anteriormente, nos provee un servicio de Hosting en la nube, sin tener que preocuparnos de la infraestructura, el tiempo de procesamiento y

el almacenamiento. Además tiene una capa gratuita, y es un servicio basado en la nube de **Amazon Web Services**.

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

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

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

6.4.3 Herramientas de pruebas

Vamos a utilizar RSpec, el conocer cómo se utiliza RSpec es importante en la actualizad para desarrollos guiados por pruebas, por lo que es interesante que el alumno conozca esta otra herramienta para pruebas de aplicaciones.

Hemos elegido **RSpec** frente a MiniTest, porque se considera que el conocimiento de realizar test con RSpec tiene más salida actualmente, y aunque es interesante aprender a realizar test con Minitest, la sintaxis entre ambos es muy parecida y hay que acotar en cierta

medida el estudio e inclusión de nuevas tecnologías en el proyecto.

Caso muy similar ocurre con Jasmine. Jasmine es una herramienta de testeo que en un principio estaba planeada utilizarse en el proyecto para testear el funcionamiento de la parte de JavaScript de la aplicación. Sin embargo se ha invertido mucho tiempo en el estudio de todas las demás herramientas utilizadas en el proyecto, ya que no se conocía gran parte de la tecnología (así como la metodología) utilizada el proyecto. Por lo tanto, vamos a proponer la inclusión de tests con Jasmine como una mejora del proyecto y el estudio de esta herramienta como una formación extra fuera de los marcos del proyecto.

7 Metodología de desarollo

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

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

En términos generales, las pruebas de aceptación del software o desarrollo son un conjunto de pruebas funcionales que se basan en metodologías ágiles que se realizan sobre el sistema completo; todos los programas tienen errores y la **ATDD** los descubre. Estas pruebas permiten al cliente o usuario final validar los requisitos, y hay dos tipos de pruebas las pruebas formales, y las informales. Antes de estudiar

en más detalle la metodología ATDD vamos a mirar unos escalones por arriba y vamos a estudiar sus orígenes.

7.1 Programación extrema (XP - eXtreme Programming)

Para hablar de los orígenes del desarrollo guiado por pruebas de aceptación debemos empezar hablando de la **metodología XP** – e**X**treme **P**rogramming. Esta metodología de desarrollo de la ingeniería del software, formulada por Ken Benk, es la más destacada de los procesos ágiles de desarrollo del software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradiciones principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de esta programación consideran que los cambios de requisitos sobre la marcha son un aspecto natural, que no se pueden evitar e incluso deseable dentro de los proyectos. Es necesario ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto, siendo una forma más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

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

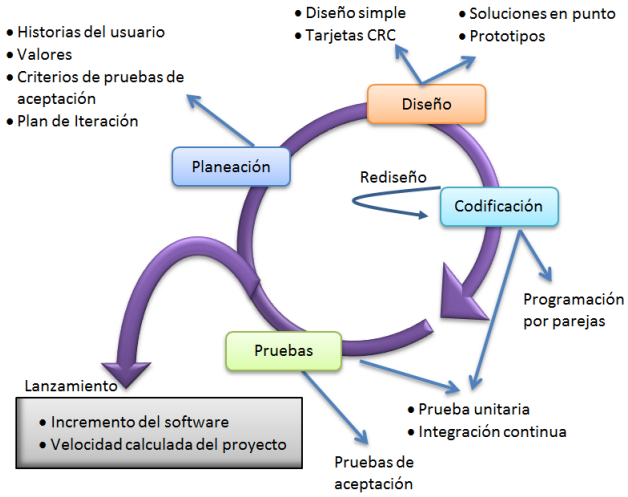

Figura 29: eXtreme Programming

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

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

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

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

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

7.2 Metodología TDD

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

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

Dicho esto, y una vez aclarada la importancia de las pruebas, vamos

a continuar con nuestro estudio de la metodología de desarrollo.

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

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

Con **TDD** se pasa de pensar en implementar tareas, a pensar en ejemplos certeros que eliminen la ambigüedad creada por el lenguaje natural. Hasta ahora estábamos acostumbrados a que las tareas, o los casos de uso, eran las unidades de trabajo más pequeñas sobre las que ponerse a desarrollar código. Intentamos traducir el caso de uso o tarea en X ejemplos, hasta que el número de ejemplos sea suficiente como para describir la tarea sin lugar a malinterpretaciones de ningún tipo.

En otras metodologías del software, primero nos preocupamos de definir cómo va a ser nuestra arquitectura. Pensamos en las clases de infraestructura que van a homogenizar la forma de trabajar en todos y cada uno de los casos, pensamos si vamos a usar un patrón u otro, una comunicación mediante eventos, o **DTOs**, y una clase central que va a hacer esto y aquello. ¿Y si resulta que luego no necesitamos todo esto? ¿Cuánto vamos a tardar en darnos cuenta de que no es necesario? ¿Cuánto dinero se va a malgastar? En **TDD** dejamos que la propia implementación de pequeños ejemplos, en constantes iteraciones, haga emerger la arquitectura que va a utilizar. No es que se olvide completamente las características técnicas de la aplicación a priori, es decir, lógicamente tendremos que tener en cuenta si el desarrollo es para un teléfono móvil, para una web o para un PC; más que nada porque tenemos que escoger las herramientas que se usarán para el desarrollo conforme las exigencias. Sin embargo nos limitamos a escoger el framework correspondiente y a usar su

arquitectura como base y lo que eliminamos son las arquitecturas encima de esas arquitecturas, las que intentan que todo se haga siempre igual y tal cómo se le ocurrió al "genio" de la empresa. A ser posible, esas que nos obligan a modificar siete ficheros para cambiar una cadena de texto. **TDD** produce una arquitectura que emerge de la no-ambigüedad de los tests automatizados, lo cual no exime de las revisiones de código entre compañeros ni de hacer preguntas a los desarrolladores veteranos.

El esquema básico del TDD lo podemos ver a continuación:

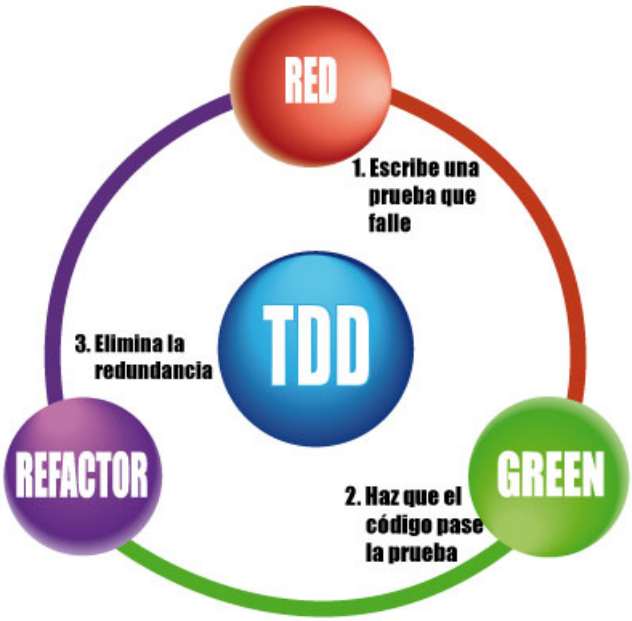

Figura 30: Esquema ciclo TDD

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

El **Ciclo Rojo** consta de *escribir la especificación del requisito*, el ejemplo, el test. Una vez tenemos claro cuál es el requisito, lo expresamos en forma de código. Y la gran pregunta puede ser ¿cómo

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

En **Ciclo Verde** se busca *implementar el código que hace funcionar el ejemplo.* Teniendo el ejemplo escrito, codificamos lo mínimo necesario para que se cumpla, para que el test pase. Típicamente, el mínimo código es el de menor número de caracteres, porque mínimo quiere decir el que menos tiempo llevó escribirlo. No importa que el código parezca feo o chapucero, eso lo vamos a enmendar con el siguiente paso y en las siguientes iteraciones.

En este paso, la máxima es no implementar nada más que lo estrictamente obligado a cumplir para la especificación actual. Estaremos tentados a escribir el código que gestiona comportamientos derivados de lo que estamos implementando, que harán falta en un futuro próximo; es aquí cuándo hay que contener el impulso y anotar todo ello en un margen, aparcándolo para una iteración próxima.

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

esta página: Catalog of Refactorings, y un buen libro es el de *Fowler* [**REF99**].

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

De una forma un poco más detallada, podríamos expandir estos tres ciclos en siete pasos más específicos para así determinar el ciclo de desarrollo guiado por pruebas, en el que, mediante los colores, asemejamos estos pasos a la Figura 30:

- **Elegir un requisito** – Se elige de una lista el requerimiento que se cree que nos dará mayor conocimiento del problema y de fácil implementación.
- Escribir una prueba – Se comienza escribiendo una prueba para el requisito.
- Verificar que la prueba falla – Si la prueba no falla es porque el requerimiento ya está implementado o porque la prueba es errónea.
- Escribir la implementación – Escribir el código más sencillo que haga que la prueba funcione.
- Ejecutar las pruebas automatizadas – Verificar si todo el conjunto de pruebas funciona correctamente.
- Eliminación de la duplicación – Se realiza la refactorización.
- **Actualización de la lista de requisitos** – Se actualiza la lista de requisitos eliminando el requisito implementado.

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

- La calidad del software aumenta.
- Incrementa la productividad.

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

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

7.3 Metodología BDD

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

BDD se basa en **TDD**, formalizando las mejores prácticas de **TDD**, clarificando cuales son y haciendo énfasis en ellas.

Algunos de los inconvenientes de utilizar **TDD** son el utilizarla cuando vamos a trabajar sobre tiempos de desarrollo reducidos, el que se requiera de conocer más técnicas o herramientas, los criterios sobre qué se constituye una buena prueba unitaria, las herramientas y sobre todo los malentendidos, ya que TDD no es puramente una técnica de pruebas, sino que más bien está en el marco de diseño y codificación.

En **BDD** no se prueban unidades independientes o clases, probamos escenarios y el comportamiento de las clases a la hora de cumplir dichos escenarios, los cuales pueden estar compuestos de varias clases. Vamos a ver de forma gráfica la integración de ambas técnicas (**TDD y BDD**):

El esquema básico del TDD lo podemos ver a continuación:

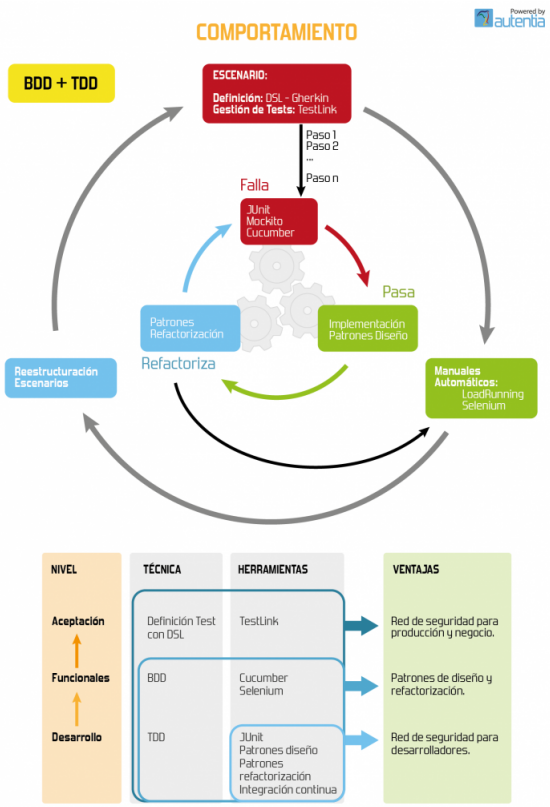

Figura 31: Esquema BDD + TDD

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

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

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

- 1- **Dado...: (Given)** Los pasos necesarios para poner el sistema en el estado que se desea probar.
- 2- **Cuando...: (When)** La interacción del usuario que acciona la funcionalidad que deseamos testear.
- 3- **Entonces...: (Then)** En este paso vamos a observar los cambios en el sistema y ver si son los deseados.

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

Dado que he introducido “2” en la calculadora
Y que he introducido “3” en la calculadora
Cuando pulso el botón “suma”

Entonces el resultado que aparece en pantalla debería ser "5"

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

7.4 Desarrollo ATDD

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

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

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

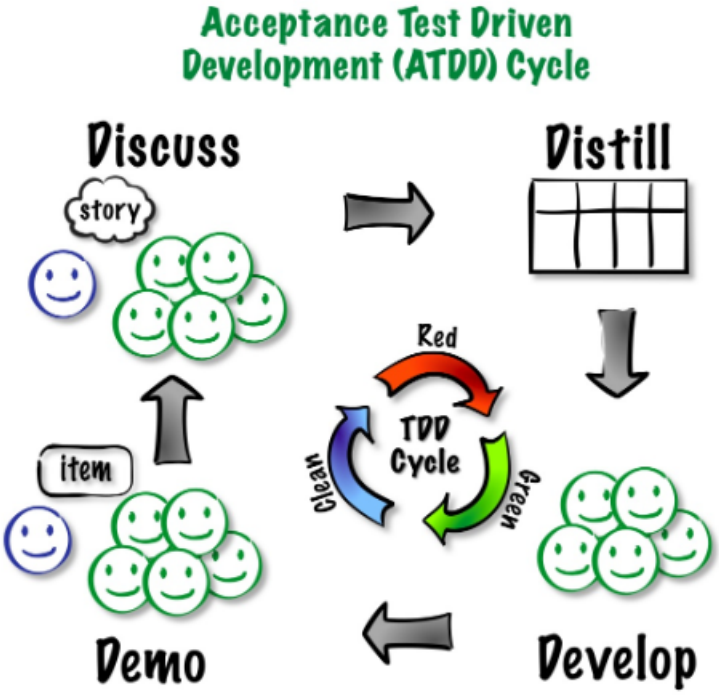

Figura 32: Esquema ciclo ATDD

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

En la Figura 32 aparece el concepto de "Historia" *(Story)* y el de "Item". Una historia de usuario posee similitudes con un caso de uso, salvando ciertas distancias. Por hacer una correspondencia entre historias y casos de uso, podríamos decir que el título de la historia se corresponde con el caso de uso tradicional. Sin embargo, la historia no pretende definir el requisito. Escribir una definición formal incurre en el peligro de la imprecisión y la malinterpretación, mientras que contarlo con ejemplos ilustrativos, transmite la idea sin complicaciones. En *ATDD* cada historia de usuario contiene una lista

de ejemplos que cuentan lo que el cliente quiere, con total claridad y ninguna ambigüedad (el "Item"). El enunciado de una historia es tan sólo una frase en lenguaje humano, de alrededor de cinco palabras, que resume lo que hay que hacer. Algunos ejemplos podrían ser:

- *Login en el sistema*
- *Añadir un estudiante en una asignatura*
- *Crear una asignatura*
- *Poner una nota a un alumno*

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

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

- ¿Qué hace el sistema si el alumno es borrado de una asignatura?
- ¿Qué sucede si un alumno se intenta añadir dos veces?
- ¿Se le indica al usuario que el alumno no ha podido ser matriculado y el motivo?

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

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

- Cuando el usuario añade un alumno en una asignatura correctamente, el sistema devuelve un mensaje que dice: "alumno añadido correctamente".
- Cuando queremos desmatricular a un alumno de una asignatura tenemos que dar al botón desmatricular.

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

- Hacer login "usuario2" produce: "Welcome usuario2"
- Añadir alumno muestra "alumno añadido correctamente"
- Clicar en "desmatricular" para desmatricular a un alumno.

Por lo tanto, podemos definir los tests de aceptación como afirmaciones en lenguaje humano que tanto el cliente, como también los desarrolladores y la máquina entienden, siendo labor del equipo de desarrollo conseguir que también la máquina lo entienda. Para ello existen varios frameworks, algunos de los cuales hemos estudiado en la sección 6.3 de este mismo documento.

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

8 Recursos utilizados

Una vez redactada la metodología de desarrollo, nos vamos a centrar en exponer los recursos que hemos utilizado para la

realización del proyecto. Empezaremos definiendo los **recursos hardware**, seguidamente veremos los **recursos software**, y finalmente definiremos brevemente las **tecnologías utilizadas** en el desarrollo. De esta última parte veremos una descripción breve de cada una de las tecnologías empleadas, ya que consideremos que no es necesario extenderse en un estudio de todas estas tecnologías.

8.1 Recursos hardware

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

Figura 33: Equipo

- **Procesador:** Intel® Core™ i7-6500U Dual-Core (4M Cache, 2.5GHz hasta 3.1GHz)
- **Memoria RAM:** 8GB (4GB + 4GB [ON BOARD]) DDR3L 1600MHz
- **Disco duro:** 1TB 5400rpm SATA
- **Pantalla:** 13.3" LED Retroiluminado / Utra Slim 350nits / Full HD (1920x1080/16:9) / Anti-Glare / NTSC:72
- **Controlador gráfico:** Intel® HD Graphics 520
- **Conectividad:**

  - 802.11ac (Dual band) + Intel® WiDi
  - Bluetooth 4.0
- **Cámara:** de portátil HD integrada
- **Micrófono:** Integrado
- **Batería:** 50WHrs, 3 celdas Li-ion polímeros
- **Conexiones:**
  - 3 x USB 3.0
  - 1 x Entrada/Salida línea audio (combo)
  - 1 x HDMI
  - 1 x mini DisplayPort
  - 1 x Entrada de Corriente
- **Lector de Tarjetas:** SD/MMC
- **Sistema operativo:** Windows® 10 Original (64Bits) y Ubuntu 16.04 LTS
- **Dimensiones:** (Ancho x Profundidad x Altura) 323 x 223 x 18 19,2 mm
- **Peso:** 1.45Kg
- **Color:** Marrón Ahumado

8.2 Recursos software

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

8.2.1 Sistemas operativos

Los sistemas operativos son la base para el funcionamiento de los programas que se han utilizado en el proyecto. En nuestro caso, hemos hecho uso de **Ubuntu 16.04 LTS** y **Windows 10**.

Ubuntu

Figura 34: Ubuntu

Ubuntu es un sistema operativo basado en GNU/Linux y que se distribuye como software libre, el cual incluye su propio entorno de escritorio denominado Unity. Su nombre viene de la ética homónima,en la que se habla de la existencia de uno mismo como cooperación de los demás. Está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario. Está compuesto de múltiple software normalmente distribuido bajo una licencia libre o de código abierto. Se ha considerado una buena elección a la hora de seleccionar un sistema operativo basado en Linux para trabajar. Será el sistema sobre el que trabajaremos para el desarrollo del proyecto.

Windows 10

Figura 35: Windows 10

Para la realización de algunos diagramas de este documento se han utilizado algunas aplicaciones basadas en **Windows**. **Windows 10** es el último sistema operativo desarrollado por Microsoft como parte de la familia de sistemas operativos Windows NT.5 Fue dado a conocer oficialmente en septiembre de 2014, seguido por una breve presentación de demostración en la conferencia Build 2014.

8.2.2 Navegador Web

Figura 36: Mozilla Firefox

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

8.2.3 Entorno de desarrollo integrado

Figura 37: RubyMine

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

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

8.2.4 Sistema de control de versiones

Figura 38: Git

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

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

8.2.5 Editores de texto

Figura 39: LaTeX

Para la realización de la memoria, se ha utilizado **LATEX** junto con el entrono integrado llamado **Kile**.

LaTeX es un sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica. Por sus características y posibilidades, es usado de forma especialmente intensa en la generación de artículos y libros científicos que incluyen, entre otros elementos, expresiones matemáticas.

Figura 40: Kile

Kile es un entorno integrado de LaTeX para el escritorio KDE. Kile le proporciona la habilidad de utilizar todas las funcionalidades de LaTex con una interfaz gráfica, proporcionándole un acceso sencillo, inmediato y personalizado a todas las herramientas para la compilación, procesamiento, depuración, conversión y visualización de LaTeX. También ofrece prácticos asistentes, una referencia de LaTeX, y potente administración de proyectos.

8.2.6 Editores de bocetos - Mockups

Los bocetos gráficos presentados en el documento se han realizado mediante la aplicación **Pencil**. Esta aplicación permite crear los mockups de sistemas de una forma rápida y vistosa.

Pencil se construyó con el fin de proporcionar una herramienta de prototipado GUI gratuita y de código abierto que la gente puede instalar y utilizar para crear maquetas en plataformas de escritorio populares fácilmente.

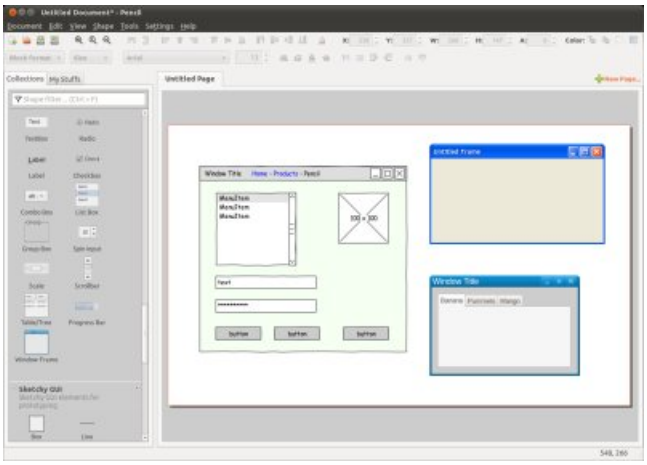

Figura 41: Pencil

8.2.7 StarUML

Figura 42: StartUML

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

StarUML se maneja con facilidad y permite dibujar los gráficos manualmente o mediante el uso de varias plantillas que contiene el archivo de instalación.

8.3 Tecnologías utilizadas

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

8.3.1 Ruby / Ruby on Rails

Figura 43: Ruby

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

Como mencionamos en la sección 6.1.3, **Ruby on Rails** es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación **Ruby**, siguiendo el paradigma Modelo Vista Controlador. En la sección 6.1.3 de este documento se ha descrito en detalle el framework Ruby on Rails, por lo que ahora no vamos a entrar en más detalle. Se puede consultar el citado apartado para más información acerca de la tecnología.

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

8.3.2 Ruby Version Manager

Figura 44: RVM - Ruby Version Mannager

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

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

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

8.3.3 Active Record

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

Su objetivo es simplificar enormemente la interacción con la base de

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

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

8.3.4 HTML5

Figura 45: HTML5

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

y una variante XHTML conocida como sintaxis XHTML5, que deberá ser servida como **XML**. Esta es la primera versión en la que

HTML y XHTML se han desarrollado en paralelo. Esta versión de HTML se publicó en octubre del 2014.

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

8.3.5 CSS3

Figura 46: CSS3

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

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

8.3.6 Bootstrap

Figura 47: Bootstrap

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

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

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

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

8.3.7 RSpec

Este framework de testeo ha sido estudiado anteriormente en este documento y la podemos encontrar en la sección 6.3.2 RSpec. En el proyecto se ha utilizado para poder llevar a cabo un desarrollo TDD.

8.3.8 Heroku

Heroku es otra herramienta estudiada en la sección 6.2.1 y en concreto es la plataforma en la nube en que se va a hospedar nuestra aplicación. Su mención en este apartado es meramente referencial a que es otra de las tecnologías que hemos integrado en nuestra aplicación.

8.3.9 Cloudinary

Cloudinary es una empresa de tecnología SaaS con sede en Sunnyvale, California , con una oficina en Israel. La compañía ofrece una solución de gestión de imágenes y video basada en la nube. Permite a los usuarios cargar, almacenar, administrar, manipular y entregar imágenes y videos para sitios web y aplicaciones. Cloudinary es utilizado por más de 120.000 desarrolladores de aplicaciones web y móviles en más de 3.000 empresas como Condé Nast, Dropbox, WeWork, Outbrain, Taboola y Answers.com.

En la sección XX veremos como la hemos incorporado a nuestro proyecto y para que nos ha servido en concreto para gestionar imágenes.

9 Análisis

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

Como hemos nombrado, a continuación vamos a ver qué características de los sitios web estudiados vamos a desear que contenga nuestro portal.

Analizando los diferentes sitios web referentes al cuaderno del profesor, estudiados en la sección 3, podemos sacar varios aspectos en común que deseáremos que tenga nuestro portal.

Por una parte, todos ellos tienen información organizada de los grupos/asignaturas que tiene un docente, y en cada grupo aparecen sus alumnos y las distintas notas que va obteniendo ese alumno para sus exámenes y trabajos. Es una idea visual que se repite mucho a lo largo de los sitios web que hemos estudiado, y que querremos que nuestro portal también posea.

Todos ellos, tienen la idea muy clara en cuanto a la organización y estructura de contenidos. Hay varias secciones generalmente comunes a estos sitios:

- Información en general del cuaderno del profesor
- Posibilidad de registro
- Información de los alumnos de cada asignatura
- Información detallada de cada alumno
- Información del docente
- Horario del profesor

Debemos tener, al igual que existe en la mayoría de los sitios que hemos estudiado, un área en la que ofrezcamos información general referente al cuaderno del profesor y como utilizarlo.

A partir de aquí analizaremos nuestro propio portal a desarrollar en el proyecto. Primero especificaremos los requisitos del usuario, en los que, a modo de descripción, explicaremos qué beneficios obtiene el usuario de nuestro sistema y las restricciones bajo las que tiene que operar.

Sequidamente se verán los requerimientos del sistema, de una forma más extendida y técnica, ampliaremos los requisitos del usuario vistos anteriormente.

9.1 Especificación de requisitos de usuario

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

9.1.1 Descripción

El proyecto va a consistir diseñar un prototipo de un sistema final que deberá tener un repertorio completo de funcionalidad en cuanto al cuaderno de un profesor de secundaria.

Atendiendo a nuestro prototipo en particular, queremos diseñar una aplicación web que permita la gestión del cuaderno de un profesor. Al ser un prototipo y no un sistema totalmente completo, limitaremos la funcionalidad del mismo, de modo que habrá restricciones parciales en cuanto a la gestión del cuaderno del profesor.

La funcionalidad principal que va a tener el sistema que vamos a diseñar es permitir a un profesor la inscripción de asignaturas y alumnos que imparte a su cuaderno. Para ello, deberemos proveer de capacidad a dicho usuario para registrarse en el portal, añadir alumnos y asignaturas para poder gestionarlos.

Para que esto sea posible, también debemos proveer al profesor

la capacidad para administrar asignaturas. Será capaz de llevar la administración de dichas asignaturas. Tiene que tener potestad para poder crear nuevas asignaturas, modificar las asignaturas ya creadas, y borrar ciertas asignaturas.

Las asignaturas deben de tener una serie de características de consulta útiles para el usuario: "por ejemplo numero alumnos matriculados, notas de sus alumnos, ...".

De este modo, y en términos generales, tendremos un prototipo de sistema para la inscripción de asignaturas y poder matricular a los alumnos en las asignaturas. Este sistema estará accesible para el usuario vía web.

9.1.2 Evidencias del cuaderno del profesor

- Portada
  - Datos personales de profesor
  - Datos generales del centro
- Horarios
  - Horarios del profesor
  - Calendario de pruebas de evaluación y recuperación
  - Reuniones
  - Días no laborables
  - Reuniones con los padres
- Para cada grupo
  - Curso
  - Nombre del grupo
  - Materia que se imparte
  - Foto grupal de los alumnos y su nombre
  - Control de asistencia
- Para cada alumno
  - Nombre

- Curso
- Grupo
- Lugar y fecha de nacimiento
- Repetidor(s/n)
- Becario (s/n)
- Padre/Tutor
- Domicilio
- Teléfono
- Foto
- Asistencia
  - Faltas justificadas
  - Faltas injustificadas
  - Retrasos
  - Partes de disciplina
  - Expulsiones

9.1.3 Modelo del dominio

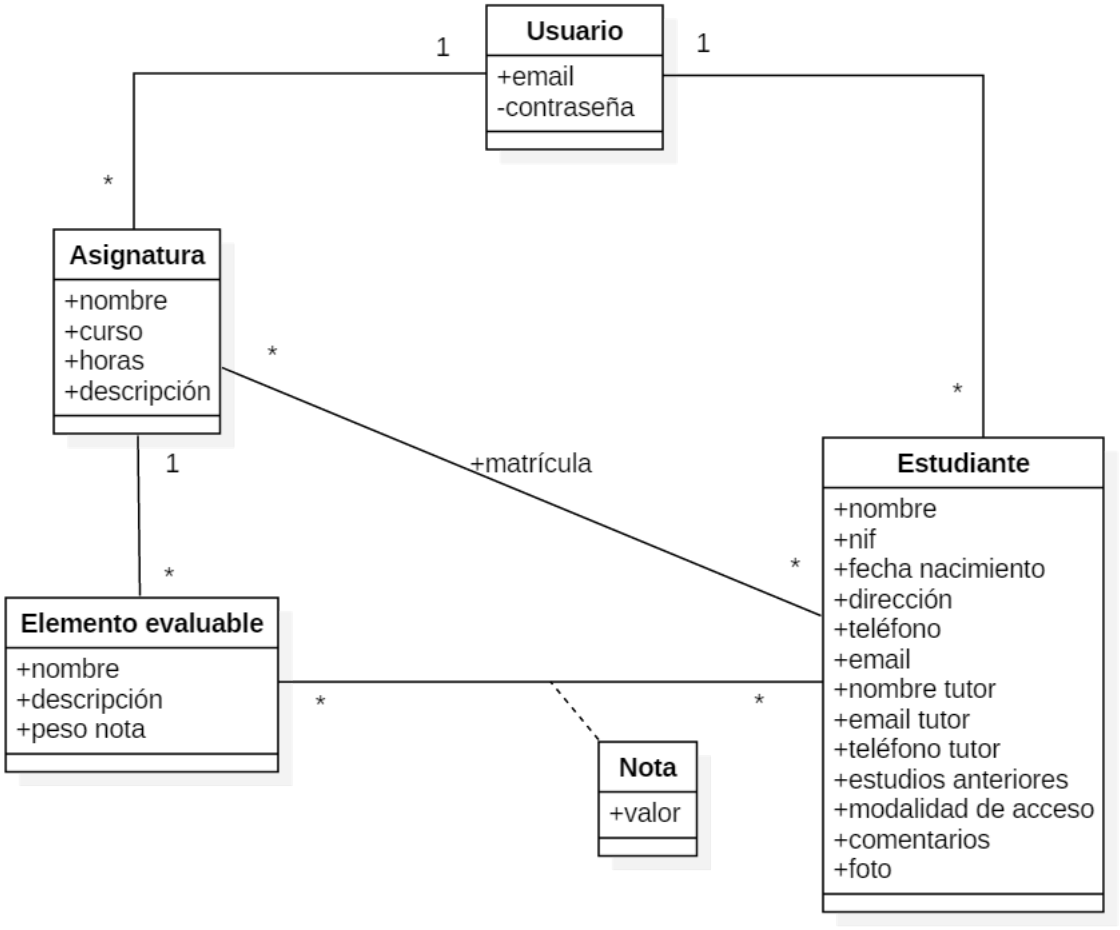

Figura 48: Modelo del dominio

9.2 Especificación de requisitos software

En la especificación de requisitos del software vamos a ver más técnica y detalladamente los requerimientos de nuestro sistema. Primero se hará descripción de los mismos. A continuación los siguientes puntos que veremos serán apartados explicativos referentes a los actores en el sistema (junto con sus roles y acciones/objetivos), el modelado de los casos de uso y un listado de los mismos.

9.2.1 Descripción

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

Las ventajas que nos proporciona la nube para alojar nuestro sistema son múltiples, y la ventaja más directa que podemos mencionar es la referente al coste. Cualquier sistema alojado en la nube abarata costes: No hay que comprar servidores, no hay que tener personal para mantenerlos, no hacen falta licencias, se paga sólo por lo que consumes. En la sección 8 de este documento se estudiará más acerca de este modelo de almacenamiento.

9.2.2 Actores y casos de uso del sistema

En nuestro sistema nos vamos a encontrar con **tres tipos de actores** interactuando con él.

Diagrama de caso de uso del sistema

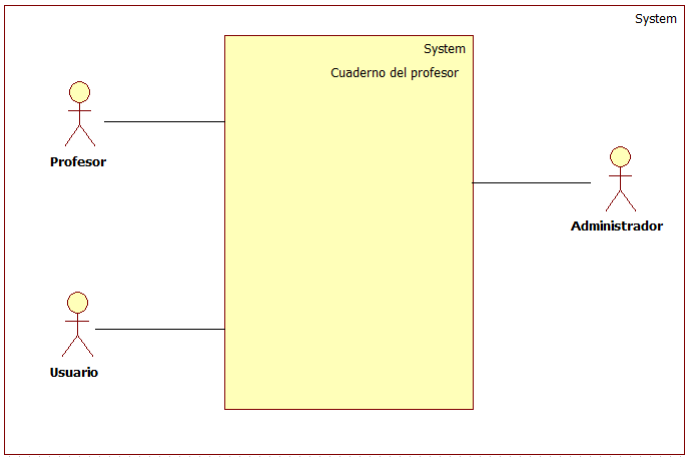

Figura 49: Diagrama de caso de uso del sistema

Listado de actores y roles

A continuación veremos una lista con los actores del sistema citados anteriormente y la definición de cada uno de ellos. Como dijimos, hay tres actores en nuestro sistema en concreto dos actores principales y uno secundario.

ACTOR	TIPO	DEFINICIÓN
Usuario	Secundario	Todo aquel visitante que tiene acceso a los servicios e informaciones esenciales que proporciona el sitio web. No se registran de él datos relevantes,aunque tiene la capacidad de registrarse, loguearse y convertirse en usuario Profesor.
Profesor	Principal	Es un usuario que se ha logueado y puede poner los datos del centro donde trabaja,localidad, su nombre,departemento, cursos, grupos y asignaturas. Puede introducir datos de los alumnos nombre, lugar, fecha de nacimiento, si es repetidor, becario, foto, telefono, correo web. Además de anotar las faltas del alumno, y las notas que va obteniendo.
Administrador	Principal	Usuario que posee las funciones de un administrador para gestionar todos los usuarios del sitio. Es decir crear los usuarios, modificarlos o eliminarlos.

Tabla 1: Listado de actores y roles

9.2.3 Diagramas UML de los casos de uso

Diagrama de caso de uso del actor usuario

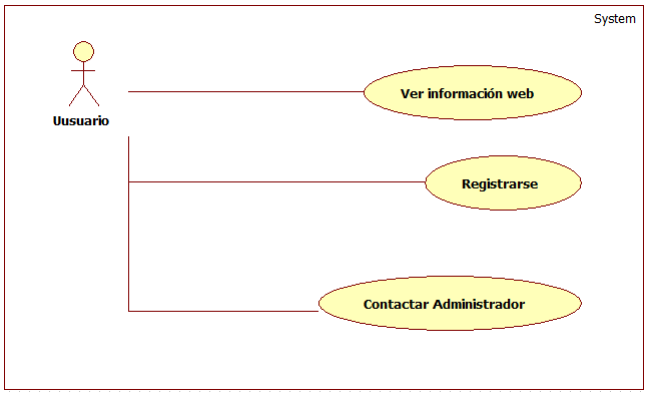

Figura 50: Diagrama de caso de uso del actor usuario

Diagrama de caso de uso del actor profesor

A continuación se muestrán los casos de uso correspondientes al actor profesor: destacar que por motivos de legibilidad se han dividido en varias pantallas.

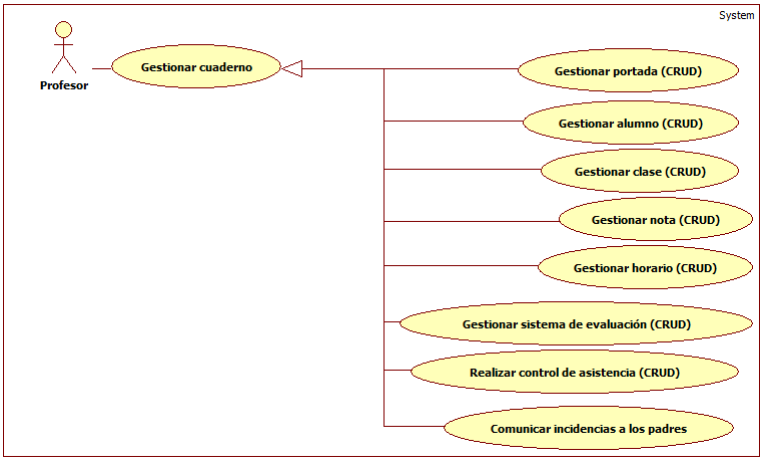

Figura 51: Diagrama de caso de uso del actor profesor

Diagrama de caso de uso del actor profesor - Gestionar portada

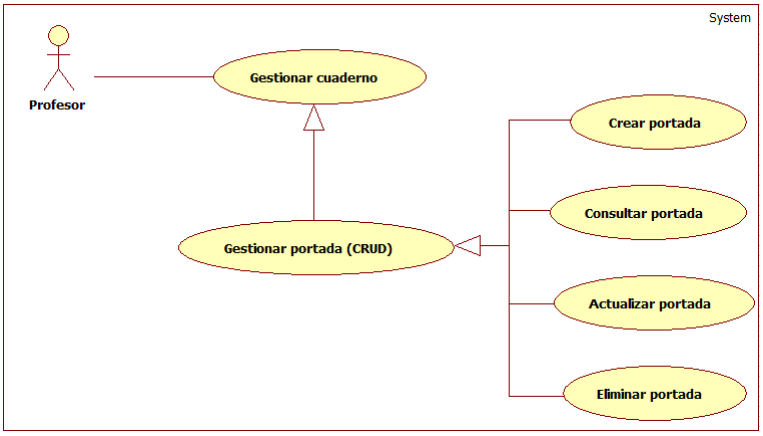

Figura 52: Diagrama de caso de uso del actor profesor - Gestionar portada

Diagrama de caso de uso del actor profesor - Gestionar alumno

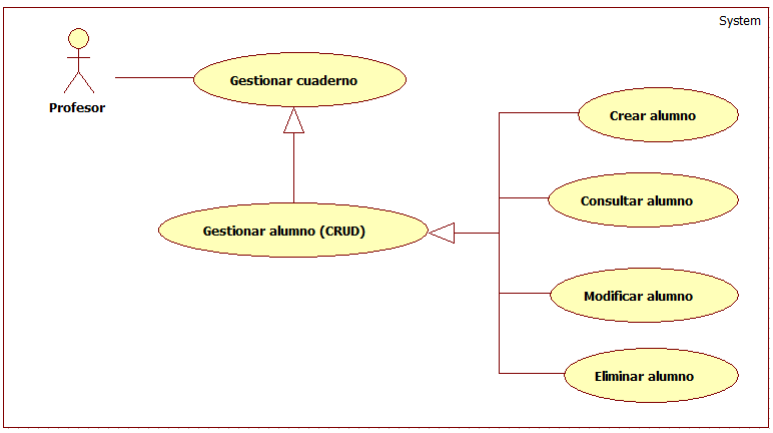

Figura 53: Diagrama de caso de uso del actor profesor - Gestionar alumno

Digagrama de caso de uso del actor profesor - Gestionar horarios

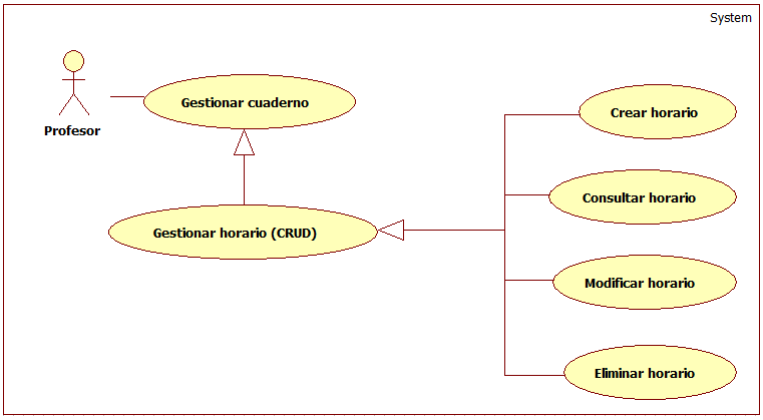

Figura 54: Diagrama de caso de uso del actor profesor - Gestionar horarios

Digagrama de caso de uso del actor profesor - Gestionar el sistema de evaluación

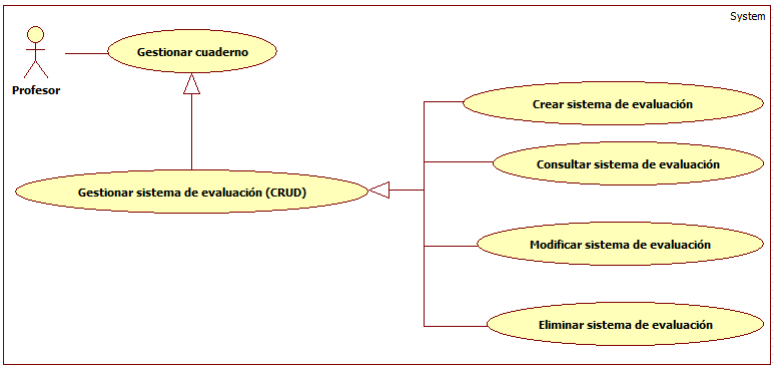

Figura 55: Diagrama de caso de uso del actor profesor - Gestionar el sistema de evaluación

Digagrama de caso de uso del actor profesor - Gestionar notas

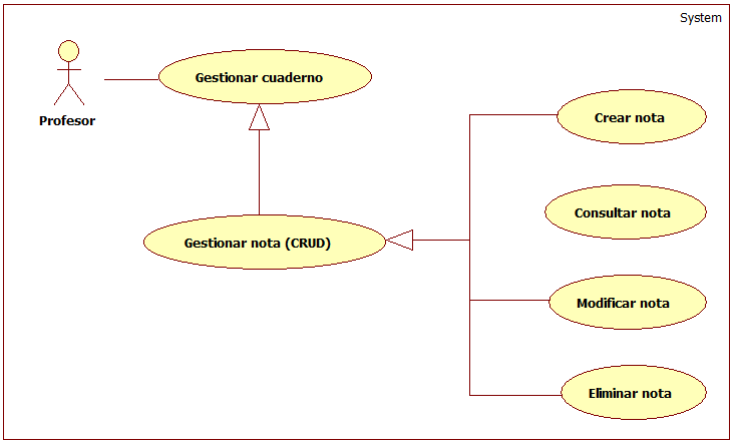

Figura 56: Diagrama de caso de uso del actor profesor - Gestionar notas

Diagrama de caso de uso del actor administrador

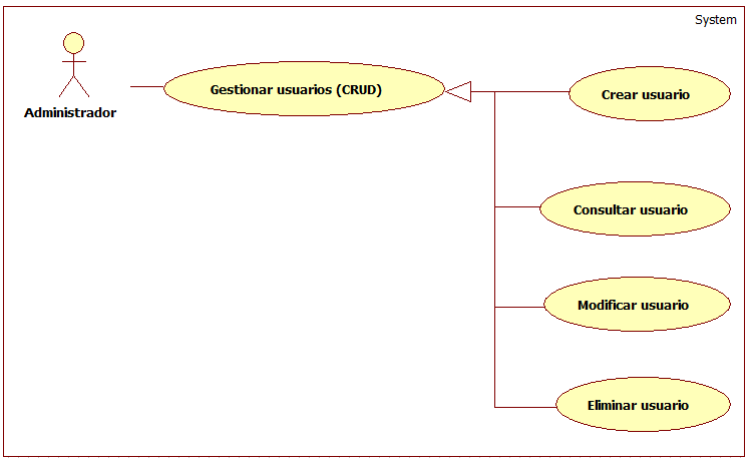

Figura 57: Diagrama de caso de uso del actor administrador

9.2.4 Listado de casos de uso

Actor	Caso de suso	Nº de CU
Usuario	Ver información web	1
	Registrarse	2
	Contactar administrador	3
Profesor	Gestionar cuaderno	4
	Gestionar portada (CRUD)	5
	Gestionar alumno (CRUD)	6
	Gestionar clase (CRUD)	7
	Gestionar nota (CRUD)	8
	Gestionar horario (CRUD)	9
	Gestionar sistema de evaluación (CRUD)	10
	Realizar control de asistencia (CRUD)	11
	Comunicar incidencias a los padres	12
	Crear portada	13
	Consultar portada	14
	Modificar portada	15
	Eliminar portada	16
	Crear alumno	17
	Consultar alumo	18
	Modificar alumno	19
	Eliminar alumno	20
	Crear horario	21
	Consultar horario	22
	Modificar horario	23
	Eliminar horario	24
	Crear sistema de evaluación	25
	Consultar sistema de evaluación	26
	Modificar sistema de evaluación	27
	Eliminar sistema de evaluación	28
	Crear nota	29
	Consultar nota	30
	Modificar nota	31
	Eliminar nota	32
	Crear clase	33
	Consultar clase	34
	Modificar clase	35

Actor	**Caso de suso**	**Nº de CU**
	Eliminar clase	36
Administrador	Gesionar usuarios (CRUD)	37
	Crear usuario	38
	Consultar usuario	39
	Modificar usuario	40
	Eliminar usuario	41

Tabla 2: Listado de actores y roles

9.2.5 Pruebas del software

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

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

A continuación vamos a describir una de las iteraciones en las que se ha dividido la creación de las pruebas, y del sistema en sí.

Ejemplo de iteración

Esta fase del proyecto se ha dedicado a la primera tarea que debemos plantearnos y que consiste en administrar los estudiantes que se guardarán en el cuaderno. Se ha decido implementar esta funcionalidad para ir cogiendo soltura con la metodología de desarrollo que estamos tratando.

Existen 4 aspectos claves que debemos poder hacer con los estudiantes: Crearlos, mostrarlos, modificarlos y poder borrarlos (CRUD – Create, Read, Update and Delete). Lo primero que se ha implementado son las pruebas de correcto funcionamiento que las acciones realizadas a los estudiantes deberán superar para asegurar que nuestro desarrollo va a ser fiable y sostenible.

Para ver esto vamos a utilizar RSPEC que nos va a permitir ver si el modelo se ha creado correctametne y describir el comportamiento esperado del sistema.

Para ver si el modelo de estudiante se crea correctamente para ello tenemos el fichero (spec/model/student.rb):

```
require 'spec_helper'

describe Student do

  before do
    @user = FactoryGirl.create(:user)
    @subject = FactoryGirl.create(:subject,user_id: @user.id)
    @student = FactoryGirl.build(:student, nif: '44703488k',
                                 user_id: @user.id)
  end

  subject { @student }
```

```
it { should respond_to(:name) }
it { should respond_to(:birthday) }
it { should respond_to(:nif) }
it { should respond_to(:address) }
it { should respond_to(:phone) }
it { should respond_to(:email) }
it { should respond_to(:guardian_name) }
it { should respond_to(:guardian_phone) }
it { should respond_to(:guardian_email) }
it { should respond_to(:former_studies) }
it { should respond_to(:access_mode) }
it { should respond_to(:remarks) }
it { should respond_to(:enroll?) }
it { should respond_to(:marks) }
it { should respond_to(:subjects) }
it { should respond_to(:user) }
it { should be_valid }

describe "when name is not present" do
  before { @student.name = " " }
  it { should_not be_valid }
end

describe "when user_id is not present" do
  before { @student.user_id = " " }
  it { should_not be_valid }
end

describe "when nif is not present" do
  before { @student.nif = " " }
  it { should_not be_valid }
end

describe "when email is not valid" do
  before { @student.email = "invalid" }
  it { should_not be_valid }
end
```

```
  describe "when guardian email is not valid" do
    before { @student.guardian_email = "invalid" }
    it { should_not be_valid }
  end

  describe "when it has not been enroled to the subject" do
    before { @student.save }
    it { expect(@student.enroll?(@subject)).to be_false }
  end

  describe "when it has been enroled to the subject" do
    before do
      @student.save
      @student.subjects << @subject
    end
    it { expect(@student.enroll?(@subject)).to be_true }
  end

  describe "when nif has been taken for the same user" do
    before do
      @another_student = FactoryGirl.create(:student, nif:
          '44703488k', user_id: @user.id)
    end
    it { should_not be_valid }
  end

  describe "when nif has been taken for a different user" do
    before do
      @another_student = FactoryGirl.create(:student,
                                            nif: '44703488k')
    end
    it { should be_valid }
  end
end
```

Una vez creado este test si intentamos pasar el test con:

```
rspec spec/models/student_spec.rb
```

nos va a dar una serie de errores, en este momento estamos en el ciclo rojo y tendremos que ir creando el modelo hasta conseguir pasar los test y pasar al ciclo verde.

10 Diseño

Tras haber estudiado los sitios webs y los servicios que estos proporcionan, y que consideremos relevantes para nuestra aplicación, los cuáles hemos visto en la sección 2 de este mismo documento, hemos analizado en profundidad y estudiado las herramientas consideradas útiles para el desarrollo del proyecto en la sección 6. De este modo se han concluido varios aspectos, en los que basaremos el diseño de nuestra aplicación web. Cómo vimos en la sección 7, vamos a utilizar una metodología de desarrollo ágil, que se basa en pruebas, y es conocida como ATDD – Acceptance Test Driven Development. Esta práctica nos obliga a realizar las pruebas antes del desarrollo del proyecto, de manera que durante el desarrollo del mismo conseguiremos un código limpio y que funcione. De este modo, nuestro desarrollo y diseño vendrá condicionado por una serie de pruebas las cuales hemos estudiado en la sección 9.2.3. Este aparatado lo comenzaremos detallando la arquitectura que vamos a utilizar para que soporte los requisitos funcionales y no funcionales vistos anteriormente. A continuación nos centraremos en nuestro portal web, estudiando la estructura, algunos casos particulares de mockups y el modelo de despliegue. La última sección que veremos en este apartado será referente al estudio del esquema de la base de datos que del sistema.

10.1 Arquitectura software

Cómo hemos mencionado, el primer punto que veremos en este apartado será el referente a la arquitectura de nuestra aplicación. Se realizará una introducción de la arquitectura del software del sistema y seguidamente estudiaremos el patrón modelo vista controlador, ActiveRecord y el paradigma REST.

10.1.1 Introducción a la arquitectura utilizada

La arquitectura utilizada para el diseño de la aplicación es una arquitectura **web**. Se dispone de tres capas: la capa de los **clientes**, la capa del **servidor** de aplicaciones y la capa de **base de datos**.

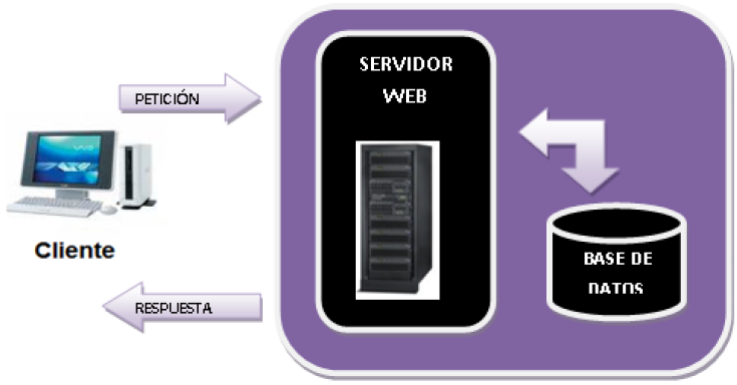

Figura 58: Arquitectura web

En la capa de cliente se sitúan como su nombre indica todas las máquinas cliente de la aplicación. Los clientes no contienen nada de la lógica de negocio de la aplicación. Toda la lógica de la aplicación esta situada en al segunda capa o nivel, es decir, en el servidor de la aplicación y en la base de datos se almacena toda la información que maneja la aplicación de forma permanente.

Cabe destacar, que los puestos de los clientes utilizan un modelo de cliene ligero. Debido a que toda la lógica de la aplicación se encuentra situada en el servidor de aplicaciones, los puestos de los clientes sólo deben tener el software necesario para que el usuario pueda invocar la funcionalidad del servidor de aplicaciones y pueda visualizar el resultado de la ejecución. Debido a que se ha utilizado el **protocolo HTTP**, este software será un navegador.

El patrón aquitectónico que se utiliza para la creación de la aplicación es el **Modelo Vista Controlador(MVC)**.

Es un patrón de arquitectura del software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para

ello MVC propone la construcción de tres componentes distintos que son el **modelo**, la **vista** y el **controlador**,es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.

Este patrón de diseño se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

10.1.2 Patron MVC

Descripción del patrón MVC

De manera genérica, los componentes de MVC se podrían definir como sigue:

- **El modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al modelo a través del controlador.
- **El controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.
- **La vista**: Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar por tanto requiere de dicho 'modelo' la información que debe representar como salida.

Interacción de los componentes en MVC

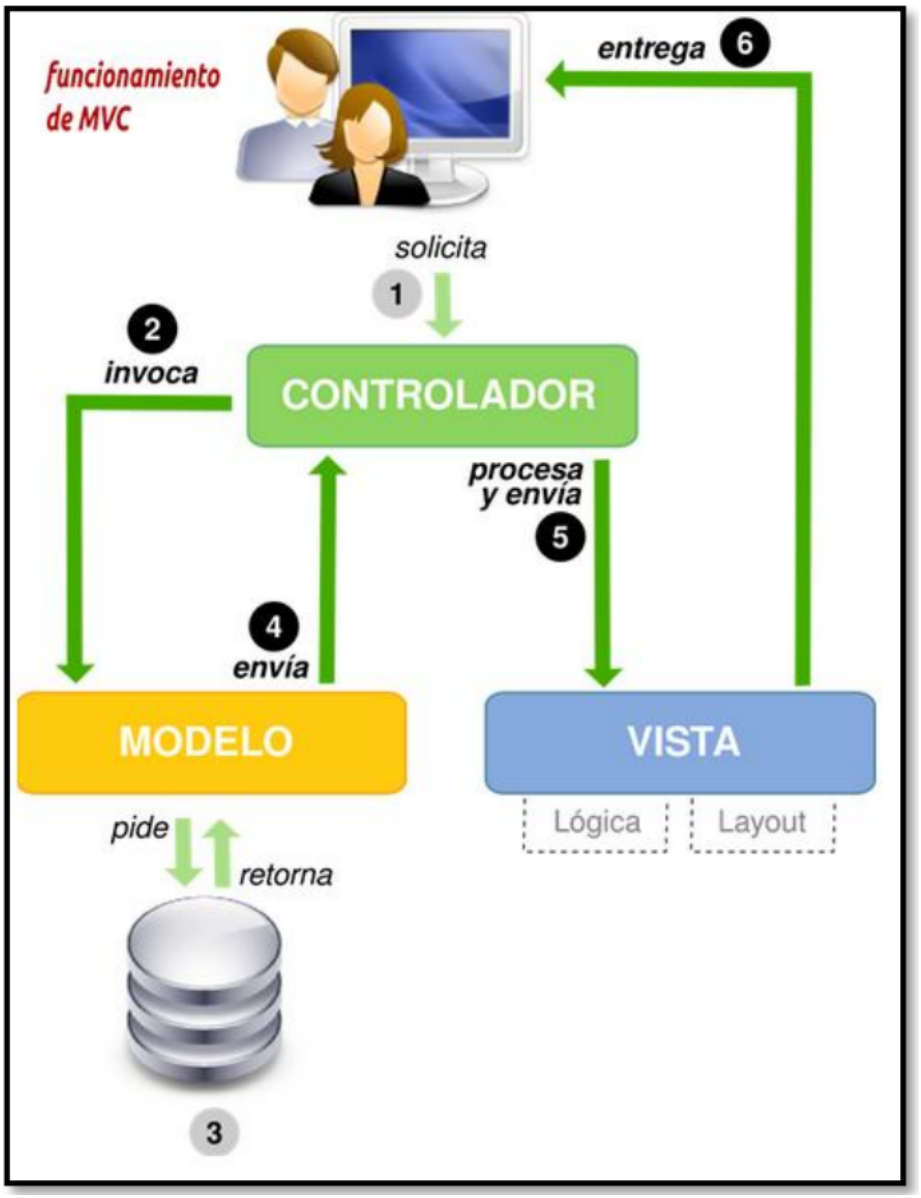

Figura 59: Diagrama de la arquitectura MVC

Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo de control que se sigue generalmente es el siguiente:

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

mediante una llamada de retorno – callback).

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

Uso en aplicaciones Web del MVC

Aunque originalmente MVC fue desarrollado para aplicaciones de escritorio, ha sido ampliamente adaptado como arquitectura para diseñar e implementar aplicaciones web en los principales lenguajes de programación. Se han desarrollado multitud de frameworks, comerciales y no comerciales, que implementan este patrón.

Ventajas de utilizar el MVC

- Clara separación entre interfaz, lógica de negocio y de presentación, que además provoca parte de las ventajas siguientes.
- Sencillez para crear distintas representaciones de los mismos datos.
- Reutilización de los commponentes.
- Simplicidad en el mantenimiento del sistema.
- Facilidad para desarrollar prototivos rápidos.
- Los desarrollos suelen ser más escalables.

Podemos ver las ventajas de utilzar lesta arquitectura del software con estos ejemplos:

Cambiar la interfaz gráfica de la aplicación web o aplicación móvil:sólo hay que modificar la vista. En el caso de una aplicación web en la vista sólo nos encontraremos con html y css. Sólo nos preocuparemos del diseño.

Extraer la información de otra base de datos: Sólo tenemos que ir al modelo donde nos encontraremos codigo para la base de datos que estemos utilizando. Seleccionar datos (seclect), agrupamientos (join), modificar datos(update). No nos encotraremos nada de html ni css.

MVC en Ruby on Rails

En concreto la aplicación se desarrollara con el Framework de aplicaciones web de código abierto escrito en lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador.

A continuación se muestra una imagen de la estructura de las aplicaciones en Ruby on Rails.

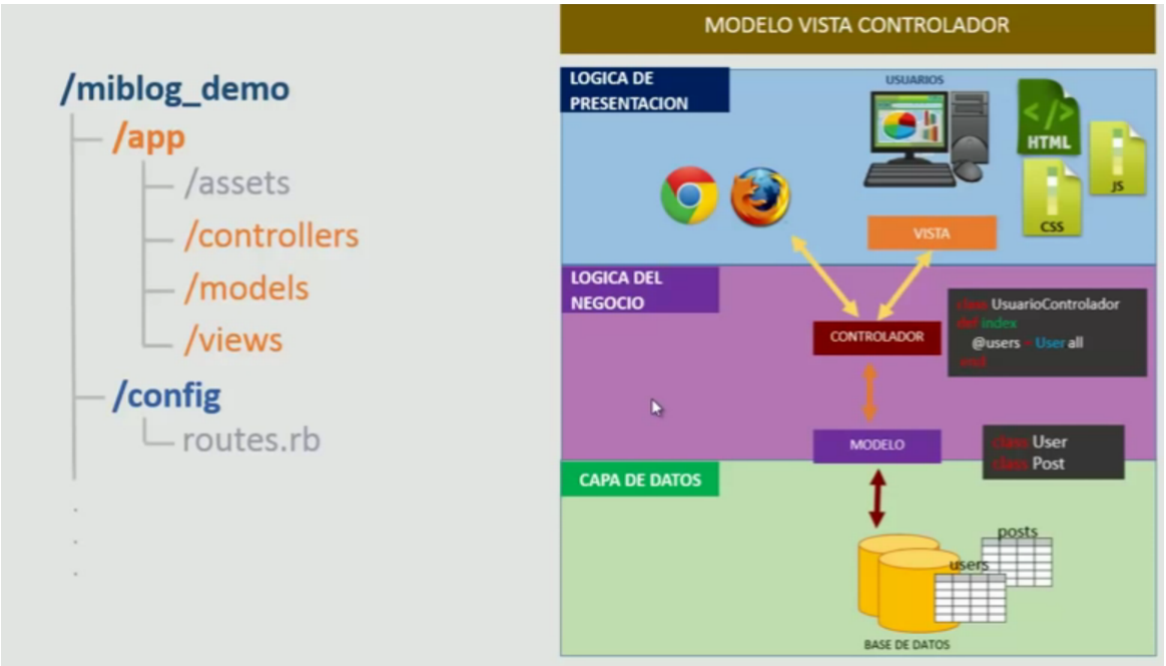

Figura 60: Estructura de las aplicaciones en Ruby o Rails

A continuación vamos a ver un caso específico de acción en Rails para un administrador. En el caso que sigue, la acción que quiere hacer el administrador es visualizar todos los usuarios que hay en el sistema; veamos cómo sería el diagrama:

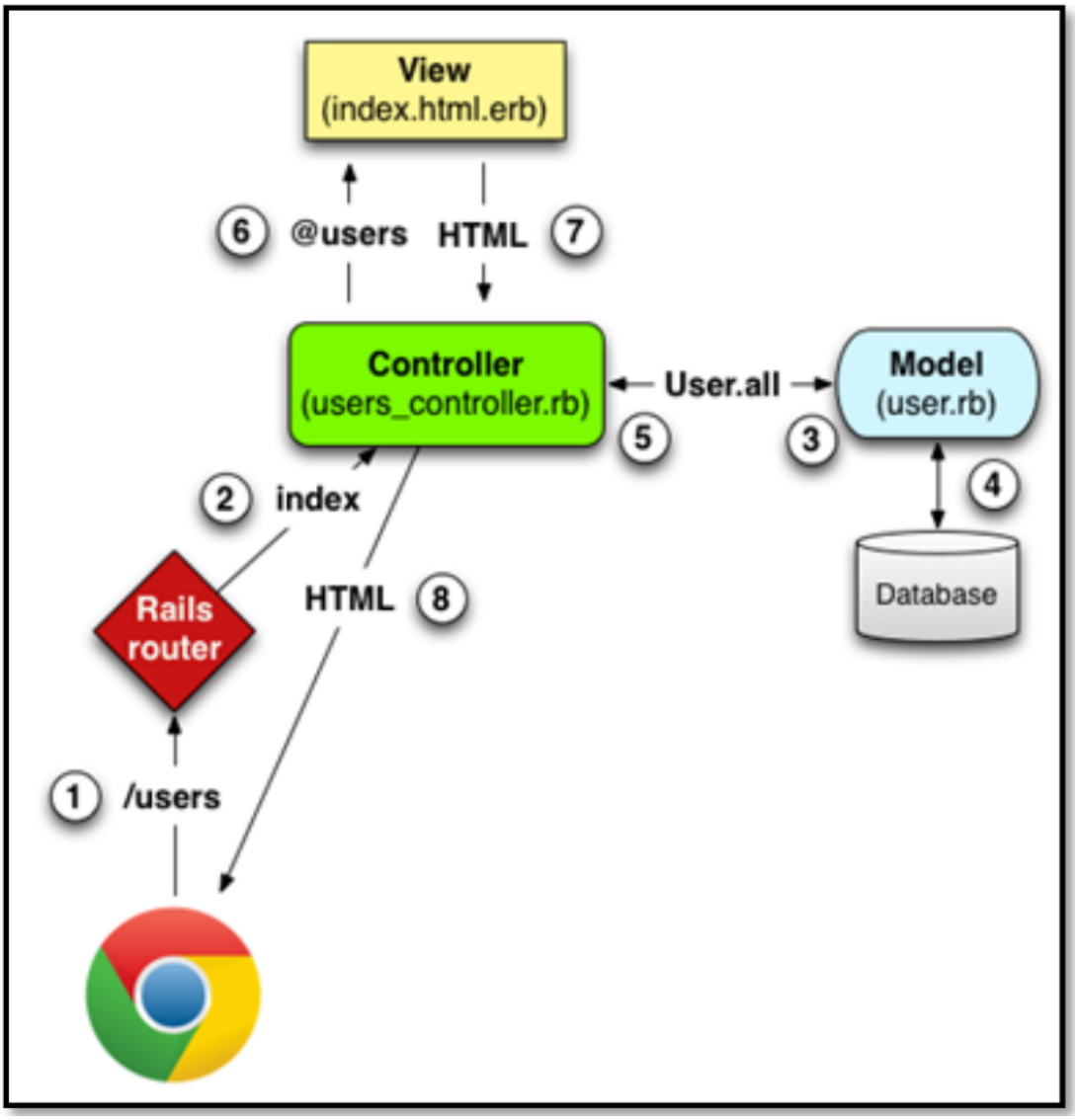

Figura 61: Diagrama detallado de MVC para Rails

1. El navegador emite una solicitud de URL para "/users".
2. Rails enruta "/users" a la acción index del controlador de los usuarios (users_controller.rb).
3. La acción index pide al modelo de usuarios para recuperar todos los usuarios (User.all).
4. El modelo User recoge todos los usuarios de la base de datos.

5. El modelo User devuelve la lista de usuarios al controlador.
6. El controlador capta los usuarios en la variable "@users", que es mandada a la vista index.
7. La vista utiliza Ruby para renderizar la página como HTML.
8. El controlador manda el HTML de nuevo al navegador.

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

10.1.3 Active Record

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

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

y los datos en un solo paquete. ActiveRecord nos agregará también herencia y asociaciones al patrón ActiveRecord común, dándole más potencia.

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

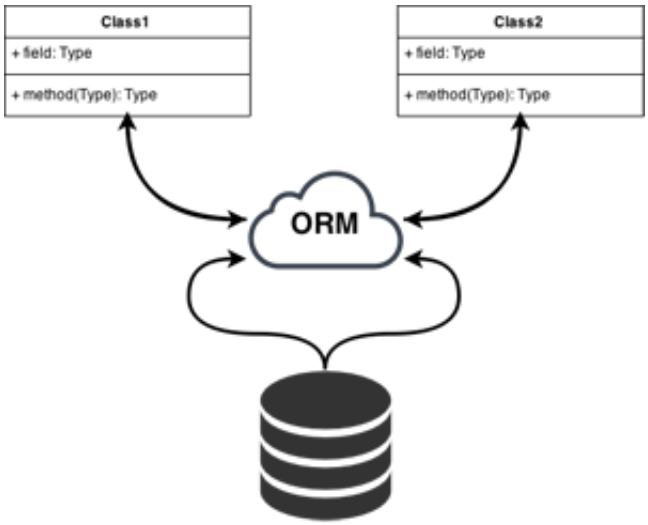

Figura 62: Mapeo Objeto Relacional (ORM)

10.1.4 Diagrama de clases

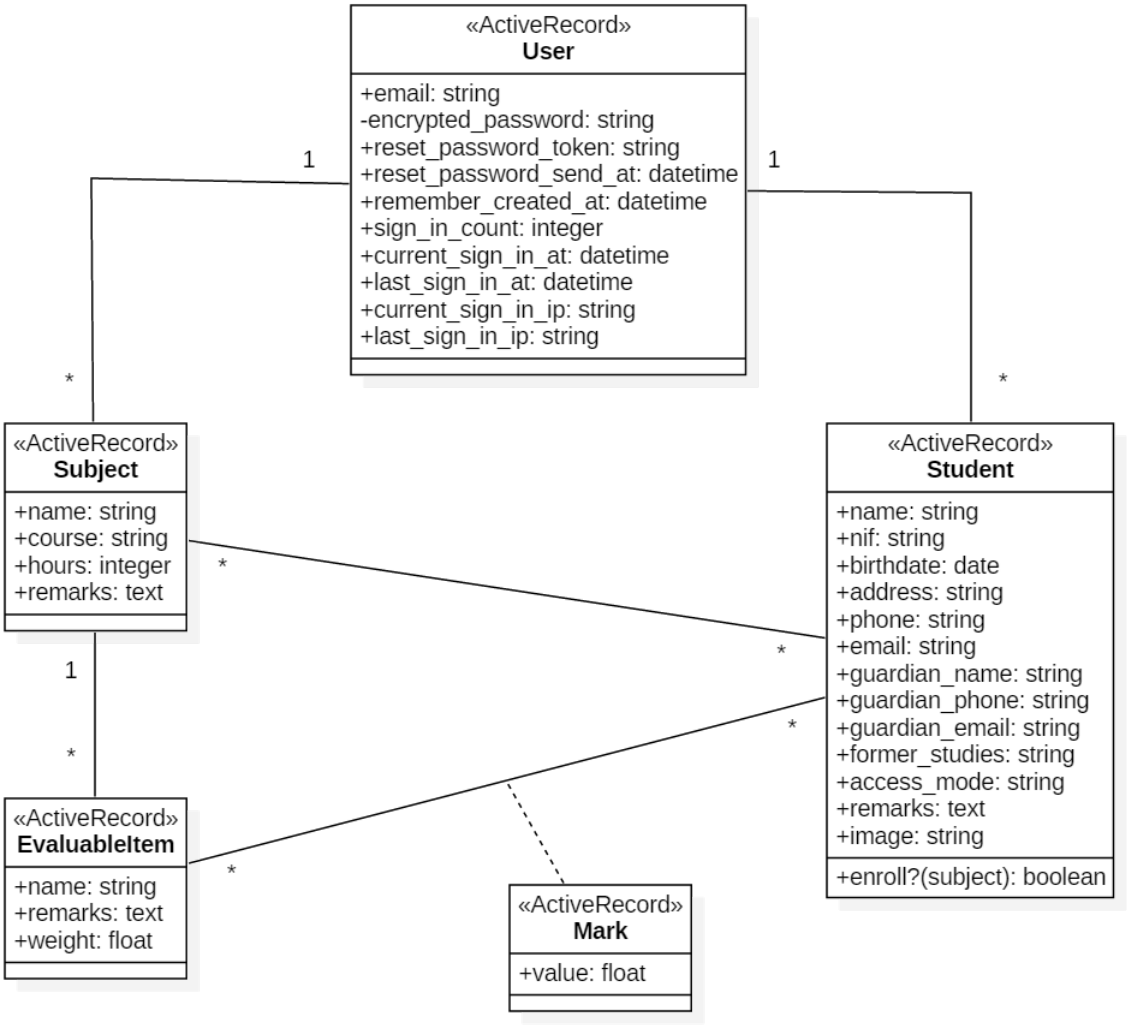

Figura 63: Diagrama de clases

10.1.5 Diagrama de clases de los controladores

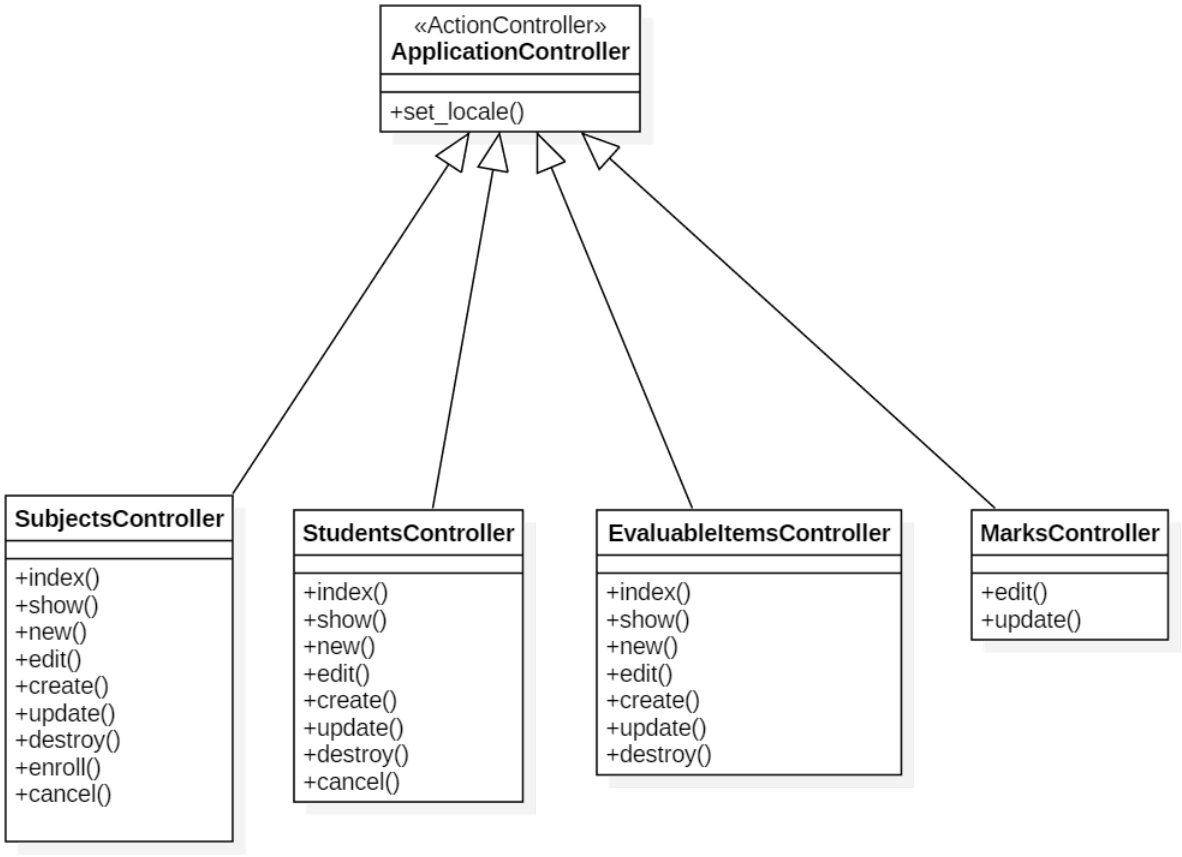

Figura 64: Diagrama de clases de los controladores

10.1.6 Transferencia de Estado Representacional (REST)

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

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

referencia a un modelo y su acción correspondiente, tan sólo hace referencia al propio recurso.

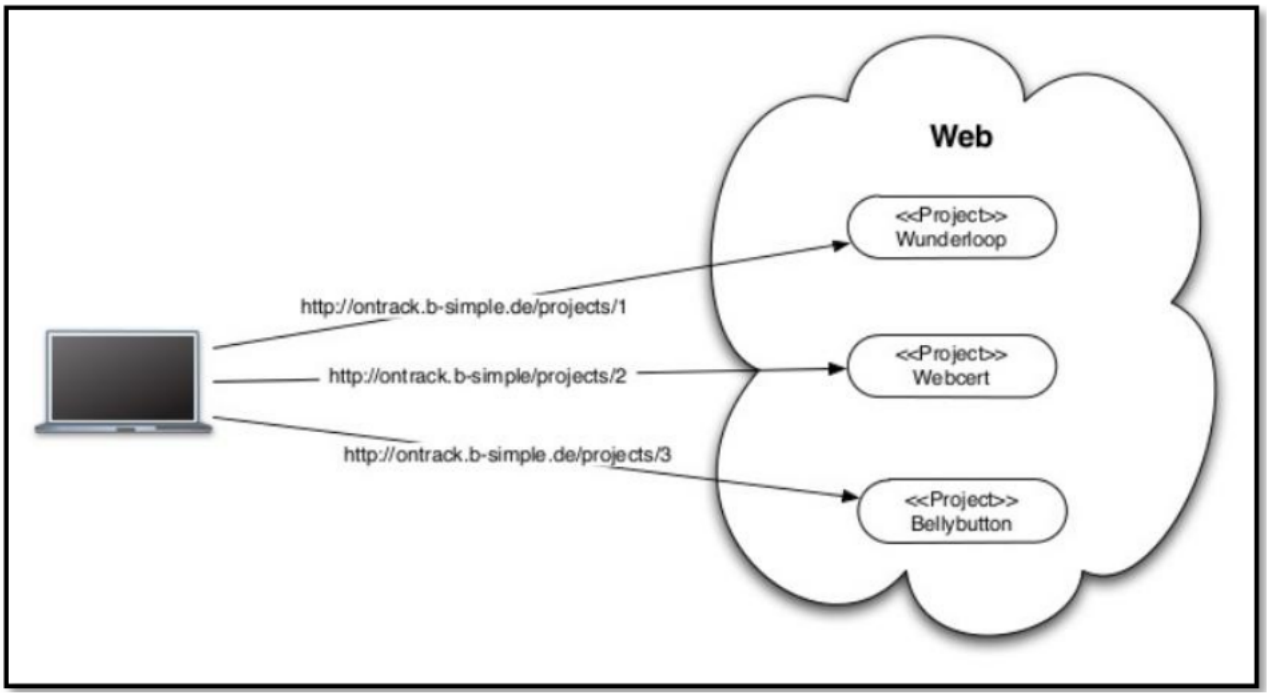

Figura 65: Recursos en las webs y sus URLs

Los tres recursos de la Figura 65 están representados por URLs casi idénticas, seguidas por el identificador del recurso. Obsérvese que la URL no muestra qué se desea hacer con el recurso.

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

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

- **URLs limpias:** En **REST**, las URLs representan recursos y no acciones, por lo tanto siempre tienen el mismo formato; primero

aparece el controlador y luego el identificador del recurso.

- **Formatos de respuesta variados:** Los controladores **REST** están escritos de manera que las acciones pueden devolver sus resultados fácilmente en diferentes formatos de respuesta. Una misma acción puede entregar HTML, XML, RSS...
- **Menos código:** El desarrollar acciones únicas capaces de soportar múltiples clientes distintos evita repeticiones en el sentido DRY.
- **Controladores orientados a CRUD:** Los controladores y los recursos se funden en una única coas; cada controlador tiene como responsabilidad manipular un único tipo de recurso.
- **Diseño limpio:** El desarrollo **REST** produce un diseño de aplicación conceptualmente claro y más fácil de mantener.

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

Operación	**Ruta**	**Método HTTP**
Ingresar datos	/students/new	GET
Crear registro	/students	POST
Leer	/students/1	GET
Actualizar registro	/students/1	PUT
Eliminar registro	/studens/1	DELETE
Listar todas	/students	GET

Tabla 3: Ejemplo de rutas REST

10.2 Portal web

10.2.1 Mookups

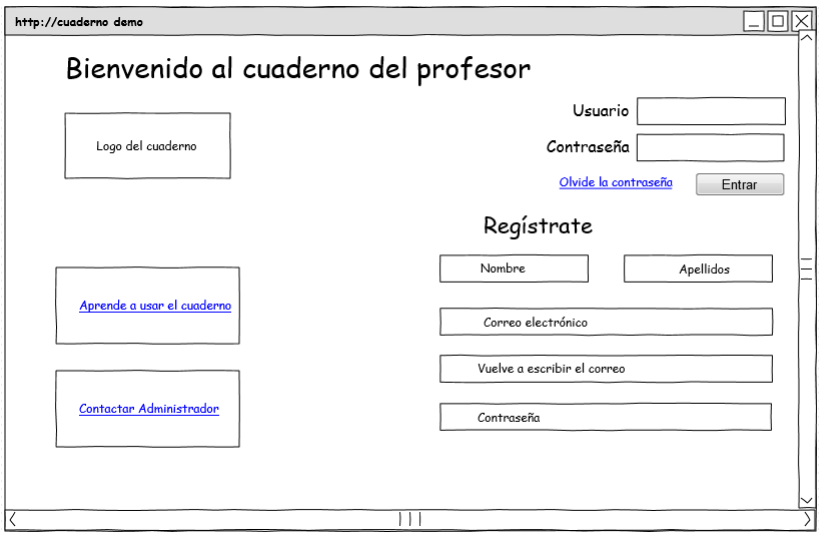

Figura 66: Mookups página inicio

Este es el mookup que se carga al iniciar la aplicación. En el aparece un logo de la aplicación un enlace a ayuda de como utilizar la aplicación y el usuario puede entrar al sistema con su usuario y contraseña. Si olvidó su contraseña puede entrar a un formulario para que le envien la nueva contraseña a su correo.

Si hay un usuario que no está registrado puede solicitar registrarse, contactar con el administrador y ver informacion de las prestaciones del cuaderno.

Mookups clases

Figura 67: Mookups clases

Una vez el profesor entra al sistema le aparece este segundo Mookup. En el se podrán crear clases nuevas,editar las que ya existen o eliminar una clase. Si queremos entrar en una clase ya creada nos ponemos sobre ella y la seleccionamos.

Mookups cuaderno

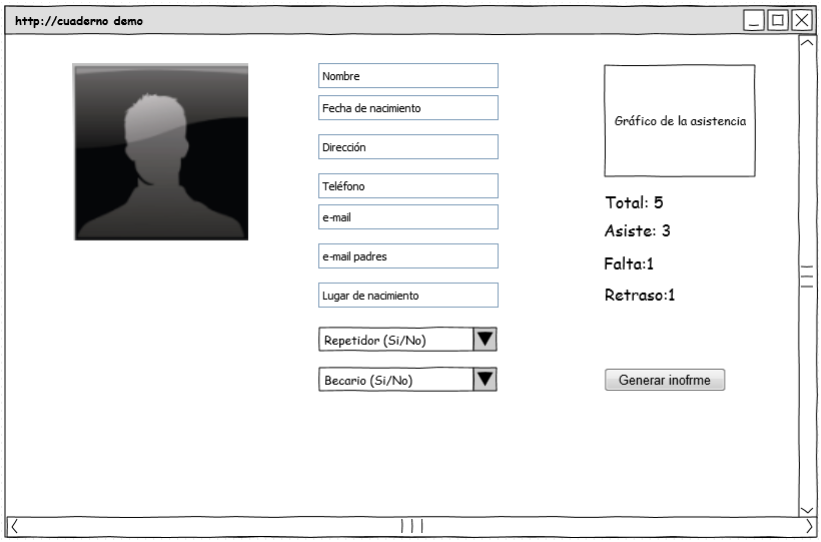

Figura 68: Mookups cuaderno

En este mookup hay tres áreas: las pestañas, las columnas y las

celdas del cuaderno. Las pestañas son páginas del cuaderno.Pueden añadirse pestañas. Se pueden crear pestañas por trimestre, otra pestaña para para la asistencia, para los test, los examenes etc.

Se pueden crear columna hacliendo click en la ultima columna cada columna tendrá un título y su tipo de nota. El tipo de nota decide como se calcularán las notas de esa columna, es decir, solo tendrá efecto si esa columna interviene en un cálculo.

Para obtener informácion de un alumno solo tenemos que hacer click sobre el.

Si queremos añadir un alumno en esta clase le damos a la pestaña añadir alumno.

Mookups alumno

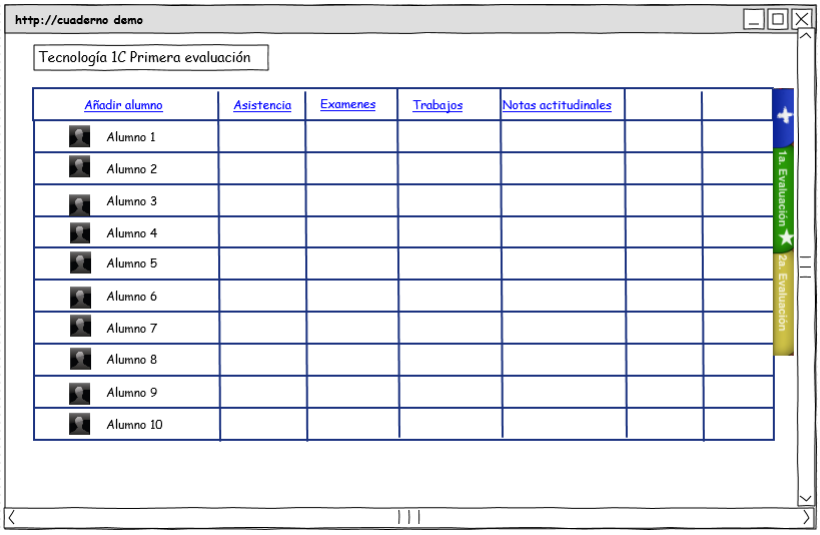

Figura 69: Mookups alumno

Para introducir o modificar los datos de un alumno, el resumen de faltas y se pueden generar informes.

Mookups portada

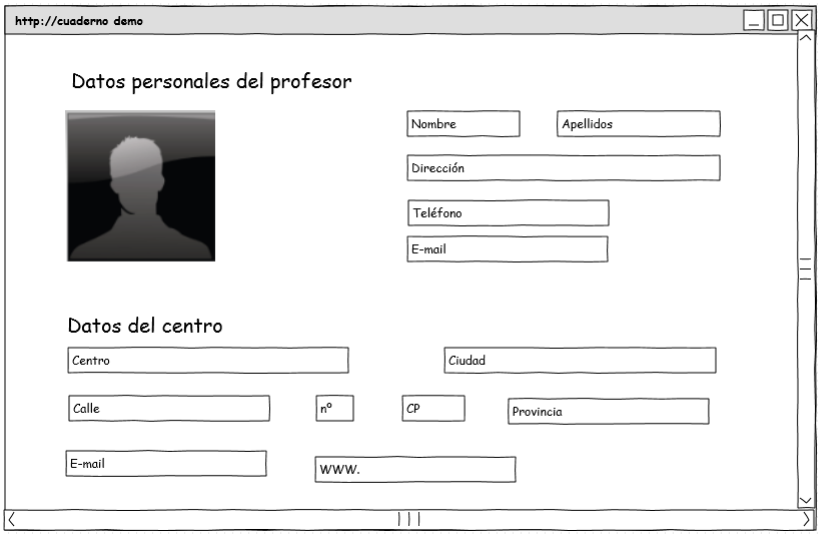

Figura 70: Mookups portada

Se introducen los datos del centro y del docente.

Mookups horario

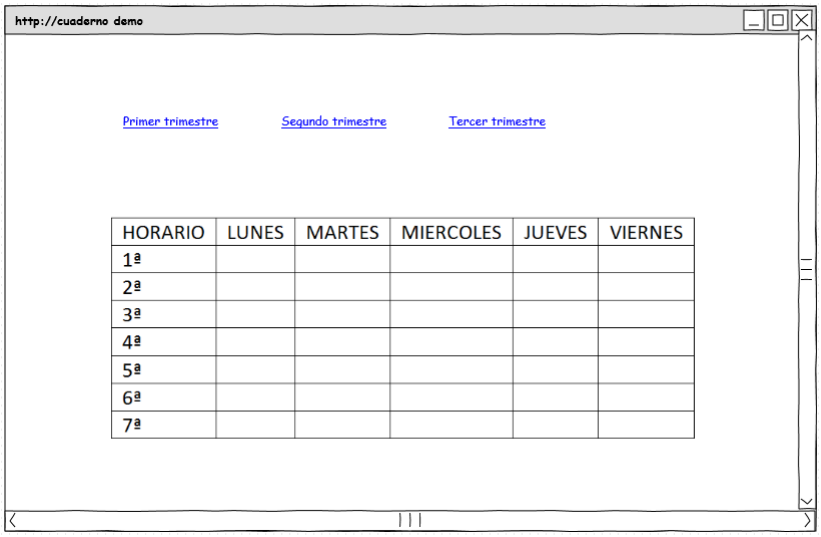

Figura 71: Mookups horario

El horario que va a tener el docente durante la primera evaluación, segunda evaluación o tercera evaluación.

Mookups fotos

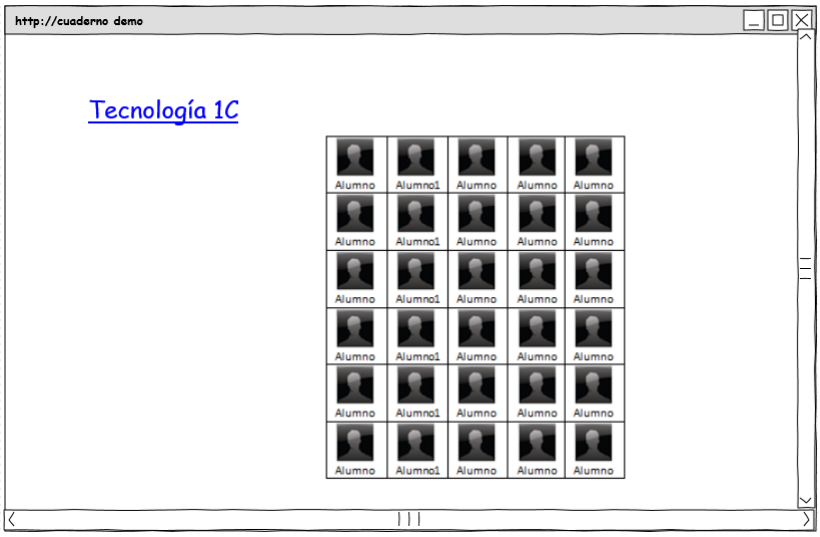

Figura 72: Mookups fotos

Es muy util para ver la foto de los alumnos.

10.2.2 Modelo de despliegue

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

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

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

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

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

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

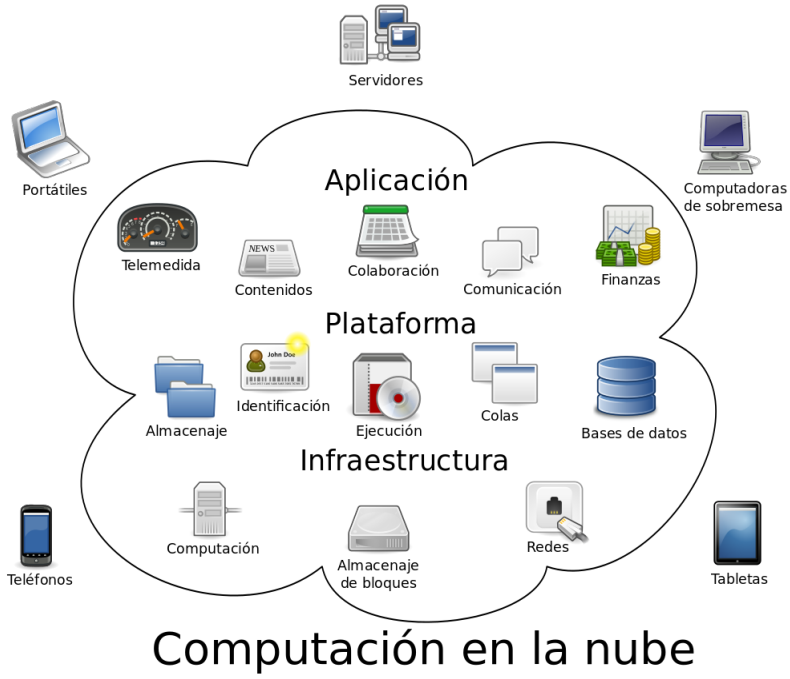

Figura 73: Esquema general de arquitecura Cloud Computing

Por lo general, el uso de este concepto se está extendiendo con una considerable velocidad, dando como resultados un incremento en

el número de empresas que proporcionan servicios a través de esta tecnología. Los tipos de servicios que ésta nos puede proporcionar son extensos. El cliente paga a un proveedor por un recurso determinado (memoria, almacenamiento, procesamiento, software, bases de datos...) y éste le proporciona dicho servicio a través de internet.

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

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

- **Concentración de esfuerzos en el negocio:** Como resultado de las ventajas antes mencionadas, el cliente puede concentrar más recursos y esfuerzos hacia un aspecto más estratégico y transcendente, que tenga un impacto directo sobre os procesos de negocio de la organización.

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

- **Privacidad:** Es comprensible la percepción de inseguridad que genera una tecnología que pone la información (sensible en muchos casos), en servidores fuera de la organización, dejando como responsable de los datos al proveedor de servicio. El tema a tratar aquí, es el de la privacidad, ya que para muchos es extremadamente difícil el confiar su información sensible a terceros, y consideran que lo que propone el cómputo en la nube pone en riesgo la información vital para los procesos del negocio.
- **Disponibilidad:** Si bien es cierto que en las ventajas se incluyó el concepto de disponibilidad, ésta queda como una responsabilidad que compete únicamente al proveedor del servicio, por lo que si su sistema de redundancia falla, y no logra mantener el servicio para el usuario, éste no puede realizar ninguna acción correctiva para restablecer el servicio.
- **Falta de control sobre recursos:** Al tener toda la infraestructura e incluso la aplicación corriendo sobre servidores que se encuentran en la nube, es decir, del lado del proveedor, el cliente carece por completo de control sobre los recursos e incluso sobre su información una vez ésta es subida a la nube.
- **Dependencia:** En una solución basada en cómputo en la nube, el cliente se vuelve dependiente no sólo del proveedor del servicio, sino también de su conexión a internet.
- **Integración:** No siempre es fácil o práctica la integración de recursos disponibles a través de infraestructuras de cómputo en la nube con sistemas desarrollados de una manera tradicional, por lo que este aspecto debe ser tomado en cuenta por el cliente para ver qué tan viable resulta implementar una solución basada en la nube dentro de su organización.

En la sección 6.2, nombrábamos que nuestra plataforma de control de la nube debería poseer características **PaaS** (**P**latform **a**s **a** **S**ervice – Plataforma como Servicio). El cómputo de la nube se puede dividir en tres niveles en función de los servicios que ofrece a los proveedores. Lo que vamos a ver en los siguientes párrafos es la definición de estos tres niveles:

Infraestructura como Servicio (IaaS – Infraestructure as a Service)

La infraestructura como servicio es un modelo de aprovisionamiento, en el cual una organización coloca "fuera de ella" el equipo usado para soportar operaciones, esto incluye el almacenamiento de la información, el hardware, servidores y componentes de redes. La ventaja más evidente de utilizar **IaaS**, es la de transferir hacia el proveedor problemas relacionados con la administración de equipos de cómputo. Otra ventaja atractiva es la reducción de costos. Además, la infraestructura como servicio permite escalabilidad práctica automática y transparente al consumidor.

Plataforma como servicio (PaaS – Platform as a Service)

Podríamos definir el concepto de **PaaS** como una plataforma que permite crear y ejecutar aplicaciones personalizadas. El proveedor, además de resolver problemas en la infraestructura de hardware, es también el que se encargar del software. Los que hacemos uso de este tipo de aplicaciones no necesitaremos instalar, configurar ni dar mantenimiento a sistemas operativos, bases de datos y servidores de aplicaciones, que todo esto está integrado en la plataforma. Una plataforma **PaaS** resuelve más problemas si se compara con la infraestructura vista anteriormente, ya que presenta muchas limitaciones relacionadas con el entorno de ejecución. En éstas se encuentran el tipo de sistema, el lenguaje de programación, el manejador de bases de datos,etc. Empresas como Amazon, eBaby, Google e iTunes son algunas de las que emplean este modelo y hacen posible acceder a nuevas capacidades y nuevos mercados a través del navegador web.

Software como Servicios (SaaS –Software as a Service)

SaaS es el más conocido de los niveles de cómputo en la nube. El SaaS es un modelo de distribución del software que proporciona a los clientes el acceso a éste a trvés de la red (generalmente Internet). De esta forma, ellos no tienen que preocuparse de las configuraciones, implementación o mantenimiento de las aplicaciones, ya que todas las labores se vuelven responsabilidad del proveedor. Las aplicaciones distribuidas a través de un modelo de Software como Servicio pueden llegar a cualquier empresa sin importar su tamaño o sin importar su ubicación geográfica.

Este modelo tiene como objetivo al cliente final que utiliza el software para cubrir procesos de su organización. El Software como Servicio **(SaaS)** se puede describir como aquella aplicación consumida a través de internet, normalmente a través del navegador, cuyo pago está condicionado al uso de la misma y donde la lógica de la aplicación, así como los datos, residen en la plataforma del proveedor. Ejemplos de **SaaS** son Salesforce, Zoho y Google App. La imagen que se muestra a continuación, podemos ver la estructura piramidal de los modelos que hemos estado detallando. Vemos como, en cierta parte, algunos niveles "heredan de los otros".

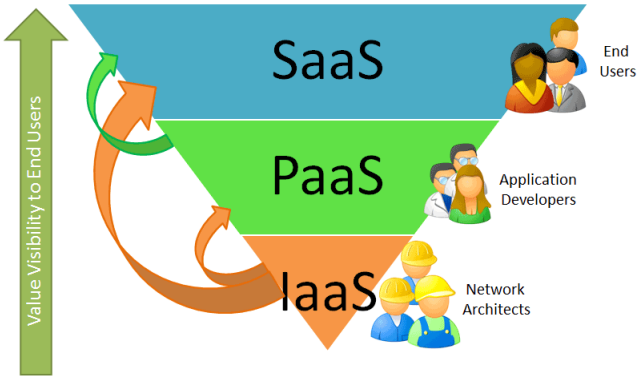

Figura 74: Capas en Cloud Computing

10.3 Esquema de la base de datos

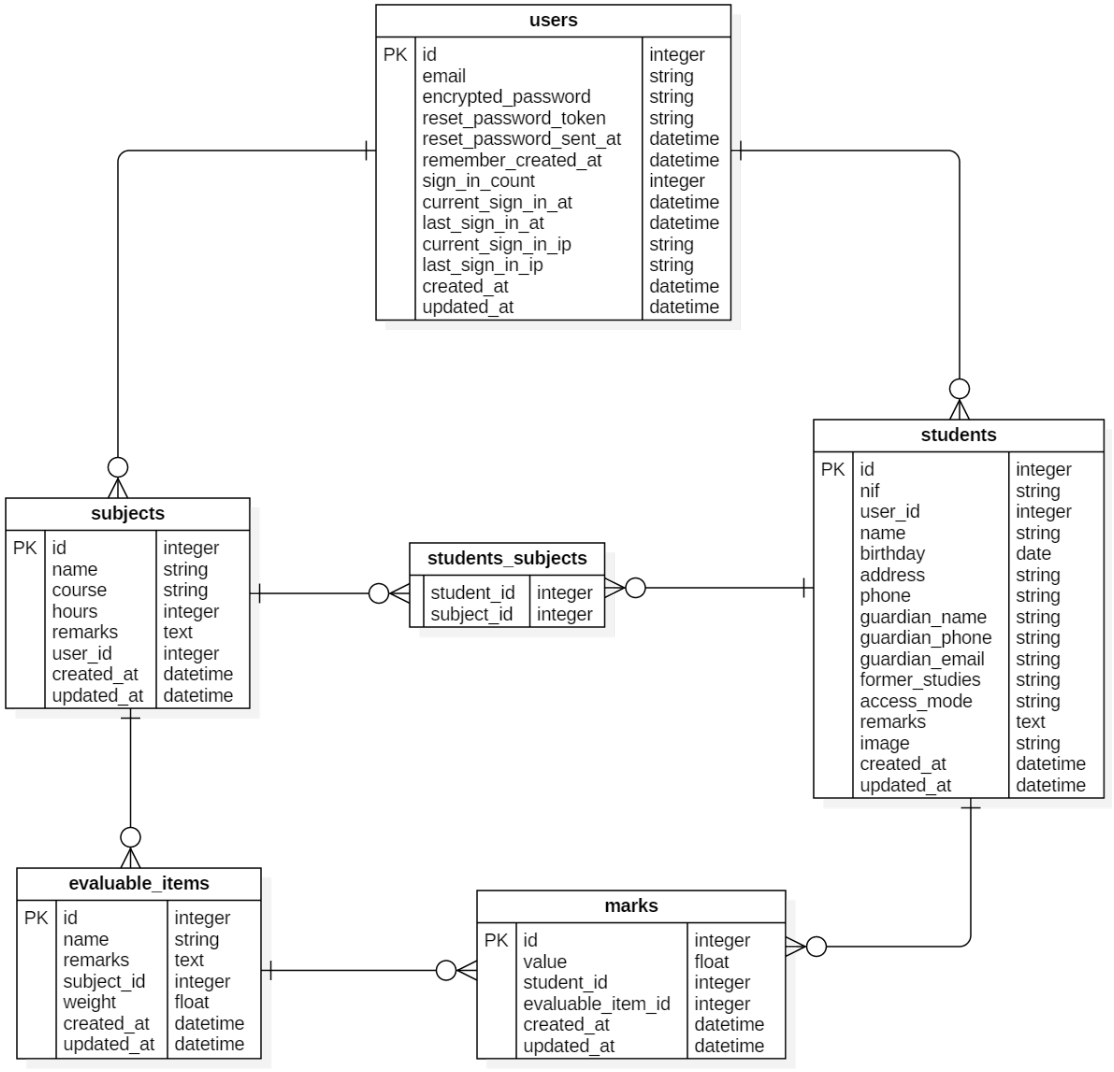

Figura 75: Esquema de la base de datos

11 Implementación

En este apartado del documento vamos a describir los aspectos de más interés en la fase de implementación de la aplicación. Concretamente vamos a hacer especial hincapié en la estructura de ficheros de la aplicación, en el aspecto del desarrollo pruebas del sistema.

11.1 Estructura de ficheros

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

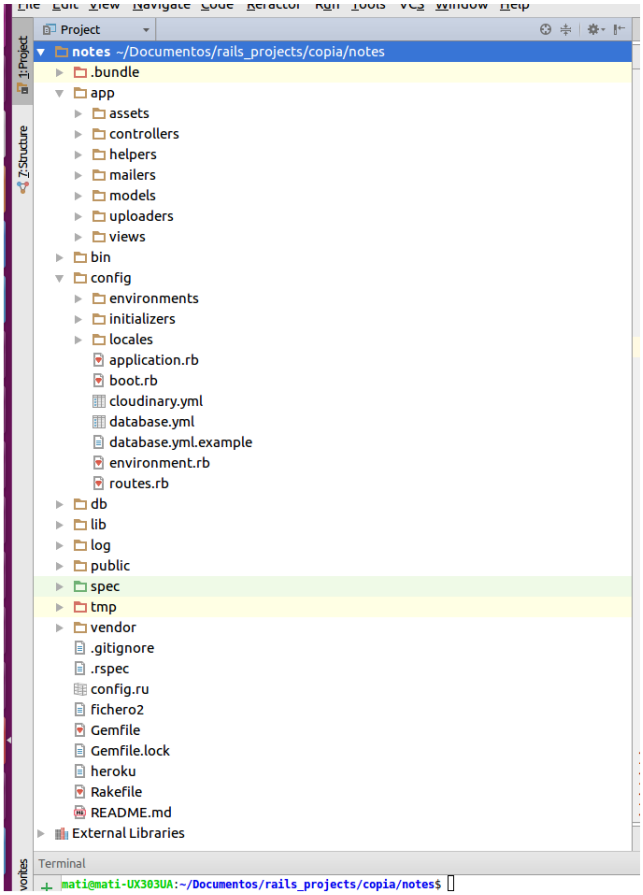

Figura 76: Estructura de directorios

- **app:** En este directorio se organiza la estructura general de la aplicación. Entre otros, tiene subdirectorios que contienen las vistas de la aplicación (view y helpers), los controladores (controllers) y los modelos de nuestro sistema (models).
  - **app/assets:** Contiene los ficheros de implementación de funciones JavaScript, CSS y las imágenes que el sistema va a mostrar.
  - **app/controllers:**Es el directorio dónde Rails busca los controladores de nuestro sistema.
  - **app/herlpers:** En este directorio nos encontramos con los helpers que apoyan a las clases de los modelos, vistas y controladores. El objetivo fundamental de los ficheros que componen este directorio es mantener un código limpio, claro y conciso.
  - **app/mailers**

  - **app/models:** En esta carpeta podemos encontrar las clases modelo.
  - **app/views:** Directorio en el cual están los archivos que contienen la implementación de las vistas de la aplicación. Contiene las plantillas con formato .ERB para completar con los datos de la aplicación, convertir a HTML y devolver al navegador del usuario.

- **bin:** Contiene archivos binarios ejecutables.
- **config:** Contiene lo poco de código de configuración que necesitará su aplicación, incluyendo la configuración de la base de datos (database.yml), la estructura del ambiente Rails (environment.rb) y el direccionamiento de los Web requests entrantes (routes.rb).
- **db:** En la carpeta "db" está el esquema de la base de datos y el fichero de semillas que se utiliza en la aplicación. También posee una subcarpeta en la que se hayan todas las migraciones realizadas durante el desarrollo del proyecto.
- **lib:** Contiene módulos de bibliotecas específicas de la aplicación.
- **log:** Aquí van los logs de peticiones y errores. (development.log, test.log y production.log).
- **public:** En esta carpeta se almacenan los datos accesibles para los usuarios (por ejemplo, los accesibles vía navegador), cómo las imágenes, hojas de estilo, ...
- **spec:** En este directorio se almacenan los casos de prueba realizados con la herramienta RSpec. Hay subdirectorios para pruebas de controladores, modelos, vistas, helpers...
- **tmp:** Rails usa este directorio para almacenar archivos temporales para procesamiento inmediato.
- **vendor:** Códigos de terceros tales como plugins y gemas.
- **External Libraries:** Aquí se encuentran todas las librerías de las que hace uso la aplicación, como por ejemplo, cada una de las librerías de las gemas que utilizamos.

11.2 I18n

La gema Ruby I18n (abreviatura de internacionalización) que se entrega con Ruby on Rails (a partir de Rails 2.2) proporciona un marco fácil de usar y extensible para traducir una aplicación a un único idioma personalizado distinto del inglés o para proporcionar idiomas multilingües Soporte en su aplicación.

El proceso de "internacionalización" significa generalmente abstraer todas las cadenas y otros bits específicos de la configuración regional (como formatos de fecha o moneda) de la aplicación. El proceso de "localización" significa proporcionar traducciones y formatos localizados para estos bits.

La aplicación ahora se puede visualizar tanto en inglés como en español y los pasos que se ha seguido para ello se muestran a continuación:

1. **Leer la siguiente documentación:**
   http://guides.rubyonrails.org/i18n.html#looking-up-translations.
2. **Modificar app/controllers/application_controller.rb**

```
before_action :set_locale
def set_locale
  I18n.locale = "es"
  #I18n.locale = params[:locale] || I18n.default_locale
end
```

3. **Modificar config/aplication_controller.rb (por defecto español)**

```
config.i18n.default_locale = :es
```

4. **Modificar config/locales/en.yml (para las traducciones en inglés) y he creado el config/locales/es.yml (para las traducciones en español)**

5. **Invocar al método t en varios lugares de la aplicación como por ejemplo en app/views/students/index.htm.erb**

```
...
<th><%= t(:name) %></th>
<th><%= t(:birthday) %></th>
<th><%= t(:nif) %></th>
...
```

6. **Para las vistas generadas por devise, que es para la gestion de usuarios, logueo, registro, etc. instalar la gema:**

```
gem 'devise-i18n'
```

7. **Instalar de las dependencias**

```
bundle install
```

11.3 Subida de imágenes con la gema "Carrieware"

La subida de los ficheros a la aplicación podría ser realizada utilizando los recursos básicos proporcionados por el lenguaje "html".

Toda la información de esta gema en la url https://github.com/carrierwaveuploader/carrierwave y facilita la integración con la aplicación, proporcionando funcionalidades extras (comprobación de tipos y tamaños de los fichero, post-procesos...).

Los pasos que he seguido para poder realizar esto en el proyecto en concreto para poder subir la imagen del alumno han sido los que se muestran a continuación:

1. **Leer la siguiente documentación:** https://github.com/carrierwaveuploader/carrierwave.
2. **Instalar las dependencias de ubuntu para que funcione la gema:**
   sudo apt-get install libmagickwand-dev imagemagick libmagickcore-dev
   sudo apt-get install libgmp-dev

3. **Instalar la gema en el sistema**

```
gem install carrierwave -v "1.0.0.rc"
```

4. **Añadir la gema en el Gemfile:**

```
...
gem 'carrierwave', '>= 1.0.0.rc', '< 2.0'
...
```

5. **Instalar las dependencias**

```
bundle install
```

6. **Seguir las instrucciones de la web de carrierwave. Para empezar, generar una clase uploader:**

```
rails g uploader Avatar
```

7. **Añadir la siguiente linea al principio de la clase del modelo Student:**

```
require 'carrierwave/orm/activerecord'
```

8. **Crear migración para incluir columna nueva en estudiantes, para almacenar la ruta donde está la imagen guardada. Las imagenes se guardan en un directorio del disco en principio. Luego hay que cambiarlo para que se guardaren en la nube.**

```
rails g migration add_image_to_students image:string
rake db:migrate
```

9. **Añadir esta linea en el modelo Student ("montamos" el "uploader")**

```
mount_uploader :image, AvatarUploader
```

10. **Instalar el imagemagick para transformar las imagenes subidas**

```
sudo apt-get install imagemagick
gem install mini_magick
```

11. **Añadir esta linea en el Gemfile, antes de la linea de carrierwave**

```
gem 'mini_magick'
```

12. **Instalar de las dependencias**

```
bundle install
```

13. **Descomentar en el avatar_uploader.rb la linea que incluye minimagick y las lineas que crean version :thumb , para generar version miniatura de la imagen. También se tiene que descomentar la whitelist para permitir solo ficheros de las extensiones que se le indiquen.**

```
include CarrierWave::MiniMagick
version :thumb do
process resize_to_fit: [50, 50]
end
def extension_whitelist
%w(jpg jpeg gif png)
end
```

14. **Añadir el campo nuevo en el formulario de estudiantes**

```
<div class="field">
<%= f.label :image %><br>
<%= image_tag @student.image_url if @student.image? %>
<%= f.file_field :image %>
<%= f.hidden_field :image_cache %>
</div>
```

15. **Añadir atributo :image_cache al modelo Student**

```
attr_accessor :image_cache
```

16. **Incluir el nuevo parámetro :image en el strong params del controlador de Students**

```
def student_params
params.require(:student).permit(
                :name, :birthday,
                :nif, :address,
                :phone, :email,
                :guardian_name,
                :guardian_phone, :guardian_email,
                :former_studies,
                :access_mode, :remarks, :image)
 end
```

17. **Añadir la imagen en la vista show del estudiante**

```
<p>
<strong><%= t('activerecord.attributes.student.image')
%>:</strong>
<%= image_tag @student.image_url if @student.image? %>
</p>
```

18. **Añadir al listado de estudiantes la imagen en miniatura (app/views/students/index.html.erb)**

```
<td><%= image_tag student.image_url(:thumb)
if student.image? %></td>
```

19. **Hacer lo mismo para la lista de estudiantes matriculados en una asignatura (app/views/subjects/show.html.erb)**

```
<td><%= link_to(image_tag(student.image_url(:thumb)),
student) if student.image? %></td>
```

20. **Añadir public/uploads al .gitignore para no versionarlo**

11.4 Cloudinary

Para poder hacer que las imágenes de los alumnos estuvieran en Heroku hemos seguido los siguientes pasos:

1. **Utilizar la documentacion:**
   http://cloudinary.com/documentation/rails_carrierwave.
   http://cloudinary.com/documentation/rails_integration#getting_started_guide.
   https://github.com/carrierwaveuploader/carrierwave.
2. **Declarar las variables de entorno con las credenciales de Cloudinary (para evitar subir las credenciales a GitHub (las credenciales se obtienen en la web de Cloudinary al loguearse)**
3. **Para evitar declarar las variables de entorno cada vez, se pueden definir dentro del fichero .bashrc para que se inicien siempre al encender el ordenador.**

4. **Instalar la gema cloudinary. Para ello instalar también la dependencia**
   sudo apt-get install g++
   gem install cloudinary
5. **Añadir la gema en el Gemfile:**

```
...
gem 'cloudinary'
...
```

6. **La gema minimagick ya no es necesaria, quitarla del Gemfile**

```
#gem 'mini_magick'
```

7. **Instalar las dependencias**

```
bundle install
```

8. **Crear fichero config/cloudinary.yml para que lea nuestras variables de entorno**
9. **Modificar el avatar_uploader.rb para que use Cloudinary y genere de antemano las versiones de las imagenes.**
10. **Para subir a Heroku, hay que declarar las variables de entorno en Heroku antes de hacer el git push**
11. **Para comprobar que se han definido bien las variables de entorno en Heroku utilizamos:**

```
heroku config
```

11.5 Paginación

La necesidad de la paginación de los contenidos aparece recurrentemente en la programación Web. Por ejemplo, no tiene sentido cargar una página con muchos clientes, ni tan siquiera hacer el query a la base de datos.

Paginar los datos con código propio no resulta difícil, pero existen gemas que nos ahorran esta codificación.

Los pasos que he seguido para poder realizar la paginación por ejemplo para los alumnos en el proyecto han sido los que se muestran a continuación:

1. **Instalar la gema en el sistema**

```
gem install kaminari
```

2. **Añadir la gema en el Gemfile:**

```
...
gem 'kaminari'
...
```

3. **Instalación de las dependencias**

```
bundle install
```

4. **Poner la acción index del controlador el metodo page y per**

```
#app/controllers/students_controller.rb
def index
@students = Student.where(user: current_user).
          page(params[:page]).per(10)
end
```

5. **En la vista del listado, poner usar el helper de vista para mostrar el paginador**

```
#app/views/students/index.html.erb
<%= paginate @students %>
```

6. **Añadir recla CSS para que el paginador aparezca a la derecha de la tabla.**

```
.pagination{
float: right;
}
```

7. **Añadir las traducciones necesarias para el paginador en en.yml y es.yml**

11.6 Gemas utilizadas en el proyecto

Hemos hecho uso de la cualidad que posee RoR para la utilización de las gemas. Una gema es un plugin y/o código añadido a nuestro proyecto que nos permite nuevas funcionalidades, nuevas funciones o nuevas herramientas para el desarrollo. Es responsabilidad del desarrollador no sobrecargar su trabajo con gemas y utilizarlas en buena medida. En nuestro caso en particular hemos hecho uso de las siguientes gemas, incluidas en el fichero "Gemfile" de nuestro proyecto:

- sqlite3
- rspec-rails
- test-unit
- selenium-webdriver
- capybara
- factory_girl_rails
- devise
- devise-i18n
- bootstrap-sass
- mini_magick
- carrierwave
- cloudinary
- kaminari

12 Resultados y conclusiones

En esta sección de la memoria hablaremos de los resultados y conclusiones obtenidas tras realizar el proyecto final de carrera.

12.1 Resultados

Los resultados que vamos a ver a continuación son unos resultados obtenidos con un trabajo realizado en aproximadamente 1.100 horas a lo largo de todo el desarrollo el proyecto de final de carrera. Hay que tener en cuenta que la **metodología de desarrollo**, así como la mayoría de las **herramientas utilizadas** en el proyecto, eran totalmente desconocidas para la estudiante, por lo que el aprender a manejar estas herramientas, así como el manejo de **metodologías de desarrollo basadas en pruebas** o incluso el uso del **paradigma MVC**, ha llevado un tiempo más que considerable empleado en el estudio de esto.

También se debe tener en cuenta que la realización de este documento se ha realizado con una heramienta muy potente como es **LaTeX** pero hay que tener en cuenta que como esta herramienta era desconocida también para la alumna ha tenido una curva de aprendizaje elevada. Se optó por esta obción porque el documento sigue fielmente una estructura concreta. LaTeX entre sus ventajas tiene:

- Asegura que los títulos tendrán siempre el mismo formato.
- Las tabulaciones que hay en los párrafos serán siempre iguales.
- Gestión de referencias cruzadas.
- Listas de figuras.
- Lista de tablas.
- Bibliografía.
- Etc

A pesar de ser un prototipo de aplicación, nuestro proyecto ha conseguido ser un sistema totalmente operativo (a nivel de prototipo) e integrado en la nube de internet. Es posible el acceso a él mediante el enlace http://limitless-waters-13812.herokuapp.com/users/sign_in la aplicación está abierta en un portal accesible por cualquier persona. Si se navega por el sistema, se observa que el prototipo está totalmente operativo, que no terminado. Un usuario cualquiera es capaz de registrarse, de navegar por el sitio, de añadir nuevas asignaturas, administrar su perfil y sus alumnos, poner notas a sus alumnos.En definitiva capacidades básicas que el producto final debería tener y que se ha conseguido con este prototipo. Hay muchos aspectos de mejoras y añadidos que el prototipo puede tener, pero estas mejoras las estudiaremos en la sección 13.

12.2 Conclusiones

Como conclusión final del proyecto, debo decir que todas las horas invertidas en el desarrollo del mismo han sido totalmente provechosas. Considero que gracias a todos los estudios realizados, el material consultado, los libros leídos, el trabajo desarrollado, y demás, se ha adquirido un nivel importante en la materia, tanto en programación Ruby on Rails cómo en metodologías de desarrollo basadas en tests.

Al ser un área desconocida para mi me ha conducido a tener que dedicar más tiempo de estudio de herramientas y disciplinas de desarrollo, consultando mucha documentación y llevándome a entender, para mi, un nuevo modo de desarrollar aplicaciones. Sin duda es un punto muy positivo a tener en cuenta en el ámbito de un proyecto de final de carrera, ya que, realmente, todo este tiempo invertido me va a proporcionar una mejoría y especialización en ciertas materias vistas durante la carrera, y, en mi caso particular, el aprendizaje de una metodología y entorno de desarrollo actualmente muy solicitados en el mundo laboral.

En cuanto al resultado final de la aplicación, he conseguido un prototipo de aplicación, y que se mantiene abierta a muchas posibilidades.

Por otro lado, el prototipo está totalmente operativo y, bajo la idea de que es un prototipo y no una aplicación comercial, el trabajo se puede dar por finalizado para este proyecto.

A nivel personal la realización de este proyecto significa un paso más en el camino recorrido en el mundo de la informática.

13 Trabajo Futuro

El proyecto está abierto a una gran cantidad de ampliaciones y mejoras ya que, al ser un prototipo de aplicación, nuestro proyecto da posibilidades de grandes ampliaciones y de futuros proyectos en base a este. En este apartado vamos a nombrar cuales son estas ampliaciones y las mejoras que debería tener el prototipo para poder empezarse a considerar una aplicación comercial.

13.1 Mejoras

Usuarios

En referente a los usuarios al sistema, habría que considerar varias posibilidades antes de comercializar la aplicación. Algunas de los aspectos que tenemos que considerar son los siguientes:

- **Registro:** La manera de hacer log-in en el sistema es mediante la introducción de su e-mail. Una posibilidad útil para el usuario es ampliar esta capacidad y que no solamente se pudiese loguear en el sistema con el e-mail, sino también con su nombre propio o un posible "nombre de usuario".
- **Gravatar:** Gravatar (Globally Recognized Avatar), es un servicio que ofrece un avatar único globalmente. En Gravatar los usuarios pueden registrar una cuenta basada en su correo electrónico, y subir un avatar para que sea asociado a la cuenta. La idea sería utilizar este servicio, e integrarlo en la aplicación final. De este

modo conseguiremos un sistema más amigable para el cliente, ampliando de este modo la profesionalidad.

- **Cookies:** Las cookies son muy útiles en cualquier sistema. Cuando iniciamos sesión en cualquier servicio web y cuando, por ejemplo, cerramos esta página y al cabo de un tiempo la volvemos a abrir, percibiremos que generalmente la sesión continúa iniciada. Esto es porque el servicio web ha leído información de la cookie que se guardó la primera vez que introducimos los credenciales. En nuestro portal web también ocurre esto, sin embargo hay una Ley de cookies que indica que, dependiendo las cookies a utilizar, algunas requieren autorización del uso por parte del usuario. En nuestro sistema actual no están contempladas estas opciones, por lo que habría que añadir tareas de confirmación de cookies por parte del usuario, así cómo aviso legal y demás características derivadas.
- **Mejoras de amigabilidad:** El prototipo actual también es susceptible a mejorar la amigabilidad en ciertos aspectos de diálogo con el usuario.

Sistema

A parte de opciones interesantes y necesarias de cara al usuario, hay otras capacidades que deben ser tenidas en cuenta desde el punto de vista de funcionalidad del sistema, y de las cuales nuestro prototipo actual carece. Estas características que se deberían implementar para llegar a ser un sistema comercial serían las siguientes:

- **Correos masivos:** Añadir al sistema capacidad de envío de correos masivos es algo casi fundamental en una aplicación comerciable de este tipo.
- **Nuevos actores:** La base para la inclusión de nuevos actores en el sistema.
- **Seguridad:** La seguridad actual del sistema es bastante pobre; la inexperiencia en este aspecto seguramente nos haya ocasionado dejar bastantes puertas abiertas a muchos tipos de hackeos de sistema. Es por ello que, para una aplicación comerciable,

habría que revisar y dejar cerrados aspectos fundamentales en la seguridad de aplicaciones web.

- **Mejorar aspectos visuales:** Habría que mejorar la apariencia visual mostrada, por ejemplo, el entorno visual no lo tenemos optimizado para móviles. Por ello, otros aspectos de mejora de cara a una aplicación comerciable serían el mejorar el diseño visual, o el integrar características concretas para poder correr la aplicación desde dispositivos móviles.

14 Anexos

14.1 Anexo I: Descripción de los casos de uso

14.1.1 Usuario - Ver información web

Caso de uso	**1**	Ver información web
Descripción	El usuario puede navegar por la web con acceso limitado	
Actores	Usuario	
Precondiciones		
Flujo normal	**Paso**	**Accion**
	1	El usuario navega por las distintas secciones informativas habilitadas en la web
	2	Se muestra la información precisa
Postcondiciones	Se muestra información requerida	
Variaciones	**Paso**	**Accion**
	1	No tiene permiso para acceder
	2	Se indica al cliente que esta información es restringida para usuarios registrados
Excepciones		
Observaciones		

Tabla 4: Descripción caso de uso: Usuario - Ver información web

14.1.2 Usuario - Registrarse

Caso de uso	**2**	Registrarse
Descripción	El usuario introduce sus datos en el sistema para quedar registrado	
Actores	Usuario	
Precondiciones		
Flujo normal	**Paso**	**Accion**
	1	El usuario entra en la sección de registro
	2	El usuario rellena el formulario de registro
	3	Se envía un e-mail de confirmación al e-mail proporcionado
Postcondiciones	Se envía un e-mail de confirmación al correo electrónico del usuario	
Variaciones	**Paso**	**Accion**
	1	Algún dato introducido por el usuario es incorrecto (dirección correo repetida, contraseña inválida, nombre de usuario incorrecto...)
	2	Se informa al cliene del los datos inválidos y se le da la opción que los ingrese de nuevo
Excepciones		
Observaciones		

Tabla 5: Descripción caso de uso: Usuario - Registrarse

14.1.3 Usuario - Contactar administrador

Caso de uso	**3**	Contactar administrador
Descripción	El usuario busca contactar con el administrador del sistema	
Actores	Usuario	
Precondiciones	RELLENAR	
Flujo normal	**Paso**	**Accion**
	1	Se pulsa el botón de contactar con el administrador
	2	El usuario rellena el formulario contacto
	3	Se envía un mensaje vía e-mail al administrador del sistema
	4	El administrador recibe ese mensaje
	5	La respuesta proporcionada por el administrador llega en forma de e-mail al correo electrónico facilitado por el usuario en el momento de la consulta
Postcondiciones	Se envía un e-mail al administrador	
Variaciones	**Paso**	**Accion**
	1	El envío de e-mail al administrador falla (se le indica al usuario que en estos momentos no es posible realizar la consulta deseada)
	2	El envío de e-mail al usuario falla (se comprueba que el e-mail indicado no tiene ninguna errata simple)
Excepciones		
Observaciones		

Tabla 6: Descripción caso de uso: Usuario - Contactar administrador

14.1.4 Profesor - Crear portada

Caso de uso	**13**	Crear portada
Descripción	Permite rellenar los datos personales del profesor y del centro	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autenticado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de rellenar la portada
	2	El profesor elige rellenar la portada
	3	El sistema muestra los campos a rellenar para crear la portada
	4	El profesor rellena los campos
	5	El sistema confirma que la portada ha sido creada
	6	El actor sistema vuelve al paso 1
Postcondiciones	Los datos de la portada son guardados en el sistema	
Variaciones	**Paso**	**Accion**
Excepciones		
Observaciones	Hay que rellenar todos los campos obligatorios	

Tabla 7: Descripción caso de uso: Profesor - Crear portada

14.1.5 Profesor - Consultar portada

Caso de uso	**14**	Consultar portada
Descripción	Permite consultar la portada	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de consultar la portada
	2	El profesor escoge consultar la portada
	3	El sistema muestra los campos con la información actual de la portada
	4	El profesor cuando termina de consultarla le da a aceptar
	5	El sistema vuelve al paso 1
Postcondiciones	Se muestran los datos de la portada	
Variaciones	**Paso**	**Accion**
Excepciones		
Observaciones		

Tabla 8: Descripción caso de uso: Profesor - Consultar portada

14.1.6 Profesor - Modificar portada

Caso de uso	**15**	Modificar portada
Descripción	Permite modificar los datos de la portada	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de modificar la portada
	2	El profesor elige modificar la portada
	3	El sistema muestra en la pantalla los campos con la información actual de la portada para poder modificarla
	4	El profesro modifica los campos que desee
	5	El sistema informa que los datos han sido modificados
	6	El sistema vuelve al paso 1
Postcondiciones	Los datos son modificados en el sistema	
Variaciones	**Paso**	**Accion**
Excepciones		
Observaciones		

Tabla 9: Descripción caso de uso: Profesor - Consultar portada

14.1.7 Profesor - Eliminar portada

Caso de uso	**16**	Eliminar portada
Descripción	Permite eliminar los datos de la portada	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la opción de eliminar los datos de la portada
	2	El profesor elige eliminar la portada
	3	El sistema informa de que los datos han sido eliminados
	4	El sistema vuelve al paso 1
Postcondiciones	Los datos son eliminados del sistema	
Variaciones	**Paso**	**Accion**
Excepciones	4	No se pueden eliminar los datos de una portada que han sido introducidos en el sistema
Observaciones		

Tabla 10: Descripción caso de uso: Profesor - Eliminar portada

14.1.8 Profesor - Crear alumno

Caso de uso	**17**	Crear alumno
Descripción	Perminte añadir un nuevo alumno en la aplicación	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autenticado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de dar de alta a un nuevo alumno
	2	El profesor elige dar de alta un nuevo alumno
	3	El sistema muestra los campos a rellenar para dar de alta al alumno
	4	El profesor rellena los campos
	5	El sistema confirma que el alumno se ha dado de alta en el sistema
	6	El actor sistema vuelve al paso 1
Postcondiciones	Los datos son dados de alta en el sistema	
Variaciones	**Paso**	**Accion**
Excepciones	6	No se puede dar de alta a un alumno si ya estaba dado de alta en el sistema
Observaciones	Hay que rellenar todos los campos obligatorios.	

Tabla 11: Descripción caso de uso: Profesor - Crear alumno

14.1.9 Profesor - Consultar alumno

Caso de uso	**18**	Consultar alumno
Descripción	Permite consultar un alumno	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de consultar un alumno
	2	El profesor elige consultar un alumno
	3	El sistema muestra los campos con la información del alumno
	4	El profesor cuando termina de consultar el alumno le da aceptar
	5	El sistema vuelve al paso 1
Postcondiciones	Se muestran los datos de un alumno	
Variaciones	**Paso**	**Accion**
Excepciones	5	No se puede consultar un alumno que no esté dado de alta en el sistema
Observaciones		

Tabla 12: Descripción caso de uso: Profesor - Consultar alumno

14.1.10 Profesor - Modificar alumno

Caso de uso	**19**	Modificar alumno
Descripción	Permite actualizar un alumno	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autenticado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de modificar un alumno
	2	El profesor elige modificar a un alumno
	3	El sistema muestra en pantalla los campos con la información actual del alumno para poder modificarla
	4	El profesor modifica los campos que desea
	5	El sistema informa que los datos han sido modificados
	6	El sistema vuelve al paso 1
Postcondiciones	Los datos del alumno son modificados en el sistema	
Variaciones	**Paso**	**Accion**
Excepciones	6	No se puede modificar un alumno que no esté dado de alta en el sistema
Observaciones	Hay que rellenar todos los campos obligatorios	

Tabla 13: Descripción caso de uso: Profesor - Modificar alumno

14.1.11 Profesor - Eliminar alumno

Caso de uso	**20**	Eliminar alumno
Descripción	Permite eliminar un alumno del sistema	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de eliminara un alumno
	2	El profesor elige eliminar un alumno
	3	El sistema informa que los datos han sido eliminados
	4	El sistema vuelve al paso 1
Postcondiciones	Los datos son eliminados del sistema	
Variaciones	**Paso**	**Accion**
Excepciones	4	No se puede eliminar un alumno que no exista en el sistema
Observaciones		

Tabla 14: Descripción caso de uso: Profesor - Eliminar alumno

14.1.12 Profesor - Crear horario

Caso de uso	**21**	Crear horario
Descripción	Permite rellenar el horario del profesor	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autenticado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de rellenar el horario
	2	El profesor elige rellenar el horario
	3	El sistema muestra los campos a rellenar para crear el horario
	4	El profesor rellena los campos
	5	El sistema confirma que el horario se ha dado de alta en el sistema
	6	El actor sistema vuelve al paso 1
Postcondiciones	Los datos son guardados en el sistema	
Variaciones	**Paso**	**Accion**
Excepciones		
Observaciones	Hay que rellenar todos los campos obligatorios	

Tabla 15: Descripción caso de uso: Profesor - Crear horario

14.1.13 Profesor - Consultar horario

Caso de uso	**22**	Consultar horario
Descripción	Permite consultar el horario	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de consultar el horario
	2	El profesor elige consultar el horario
	3	El sistema muestra los campos con la información del horario
	4	El profesor cuando termina de consultar el horario le da a aceptar
	5	El sistema vuelve al paso 1
Postcondiciones	RELLENAR	
Variaciones	**Paso**	**Accion**
Excepciones		
Observaciones		

Tabla 16: Descripción caso de uso: Profesor - Consultar horario

14.1.14 Profesor - Modificar horario

Caso de uso	**23**	Modificar horario
Descripción	Permite acutalizar el horario	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autenticado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de modificar el horario
	2	El profesor elige modificar el horario
	3	El sistema muestra en pantalla los campos con la información actual del horario para poder modificarla
	4	El profesor modifica los campos que desea
	5	El sistema informa que los datos han sido modificados
	6	El sistema vuelve al paso 1
Postcondiciones	El horario queda modificado en el sistema	
Variaciones	**Paso**	**Accion**
Excepciones		
Observaciones		

Tabla 17: Descripción caso de uso: Profesor - Modificar horario

14.1.15 Profesor - Eliminar horario

Caso de uso	**24**	Eliminar horario
Descripción	Permite eliminar el horario	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra la posibilidad de eliminar el horario
	2	El profesor elige eliminar el horario
	3	El sistema informa que los datos han sido eliminados
	4	El sistema vuelve al paso 1
Postcondiciones	Los datos son eliminados del sistema	
Variaciones	**Paso**	**Accion**
Excepciones	4	No se puede eliminar el horario sino existe en el sistema
Observaciones	RELLENAR	

Tabla 18: Descripción caso de uso: Profesor - Eliminar horario

14.1.16 Profesor - Crear sistema de evaluación

Caso de uso	**25**	Crear sistema de evaluación
Descripción	Permite añadir un nuevo item de evaluación	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autenticado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de añadir un nuevo item al sistema de evaluación
	2	El profesor elige dar de alta un nuevo item al sistema de evaluación
	3	El sistema muestra los campos a rellenar para añadir el nuevo item de evaluación
	4	El profesor rellena los campos
	5	El sistema confirma que el item de evaluación se ha dado de alta en el sistema
	6	El actor sistema vuelve al paso 1
Postcondiciones	Los datos son dados de alta en el sistema	
Variaciones	**Paso**	**Accion**
Excepciones		
Observaciones	Hay que rellenar todos los campos obligatorios	

Tabla 19: Descripción caso de uso: Profesor - Crear sistema de evaluación

14.1.17 Profesor - Consultar sistema de evaluación

Caso de uso	**26**	Consultar sistema de evaluación
Descripción	Permite consultar el sistema de evaluación	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de consultar el sistema de evaluación
	2	El profesor elige conusultar el sistema de evaluación
	3	El sistema muestra los campos con el sistema de evaluación
	4	El profesor cuando termina de consultar el sistema de evaluación le da a aceptar
	5	El sistema vuelve al paso 1
Postcondiciones	Se muestra el sistema de evaluación	
Variaciones	**Paso**	**Accion**
Excepciones	7	
Observaciones		

Tabla 20: Descripción caso de uso: Profesor - Consultar sistema de evaluación

14.1.18 Profesor - Modificar sistema de evaluación

Caso de uso	**27**	Modificar sistema de evaluación
Descripción	Permite modificar un item de evaluación	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de modificar un item de evaluación
	2	El profesor elige modificar un item de evaluación
	3	El sistema muestra en pantalla el item de evaluación con su información actual
	4	El profesor modifica el item de evaluación como desea
	5	El sistema informa que el item de evaluación ha sido modificado
	6	El sistema vuelve al paso 1
Postcondiciones	El sistema de evaluación queda modificado en el sistema	
Variaciones	**Paso**	**Accion**
Excepciones	6	No se puede modificar un item de evaluación sino existe en el sistema
Observaciones	RELLENAR	

Tabla 21: Descripción caso de uso: Profesor - Modificar sistema de evaluación

14.1.19 Profesor - Eliminar sistema de evaluación

Caso de uso	**28**	Eliminar sistema de evaluación
Descripción	Permite eliminar un item de evaluación	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sitema muestra al profesor la posibilidad de eliminar un item del sistema de evaluación
	2	El profesor elige eliminar un item del sistema de evaluación
	3	El sistema informa que los datos han sido eliminados
	4	El sistema vuelve al paso 1
Postcondiciones	Los datos son eliminados del sistema	
Variaciones	**Paso**	**Accion**
Excepciones		
Observaciones		

Tabla 22: Descripción caso de uso: Profesor - Eliminar sistema de evaluación

14.1.20 Profesor - Crear nota

Caso de uso	**29**	Crear nota
Descripción	Permite añadir una nueva nota en la aplicación	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autenticado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de dar de alta una nueva nota
	2	El profesor elige dar de alta una nueva nota
	3	El sistema muestra los campos a rellenar para añadir la nota
	4	El actor rellena los campos
	5	El sistema confirma que la nota se ha introducido en el sistema
	6	El actor sistema vuelve al paso 1
Postcondiciones	Los datos son dados de alta en el sistema	
Variaciones	**Paso**	**Accion**
Excepciones		
Observaciones	Hay que rellenar todos los campos obligatorios	

Tabla 23: Descripción caso de uso: Profesor - Crear nota

Profesor - Consultar nota

14.1.21 Profesor - Consultar nota

Caso de uso	**30**	Consultar nota
Descripción	Permite consultar una nota	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de consultar una nota
	2	El profesor elige consultar una nota
	3	El sistema muestra los campos con la información de la nota
	4	El profesor cuando termina de consultar la nota le da aceptar
	5	El sistema vuelve al paso 1
Postcondiciones	Se muestra el valor de una nota	
Variaciones	**Paso**	**Accion**
Excepciones		
Observaciones		

Tabla 24: Descripción caso de uso: Profesor - Consultar nota

14.1.22 Profesor - Modificar nota

Caso de uso	**31**	Modificar nota
Descripción	Permite modificara una nota	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autenticado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de modificar una nota
	2	El profesor elige modificar una nota
	3	El sistema muestra en pantalla la información actual de la nota para poder modificarla
	4	El profesor modifica la nota
	5	El sistema informa que los datos han sido modificados
	6	El sistema vuelve al paso 1
Postcondiciones	Los datos de la nota son modificados en el sitema	
Variaciones	**Paso**	**Accion**
Excepciones	6	No se puede modificar una nota que no existe en el sistema
Observaciones		

Tabla 25: Descripción caso de uso: Profesor - Modificar nota

14.1.23 Profesor - Eliminar nota

Caso de uso	**32**	Eliminar nota
Descripción	Permite eliminar una nota	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la opción de eliminar una nota
	2	El profesor elige eliminar una nota
	3	El sistema informa que los datos han sido eliminados
	4	El sistema vuelve al paso 1
Postcondiciones	Los datos son eliminados del sistema	
Variaciones	**Paso**	**Accion**
Excepciones	4	No se puede eliminar una nota que no esta en el sistema
Observaciones		

Tabla 26: Descripción caso de uso: Profesor - Eliminar nota

14.1.24 Profesor - Crear clase

Caso de uso	**33**	Crear clase
Descripción	Permite añadir una nueva clase en la aplicación	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autenticado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de dar de alta a un nueva nueva clase
	2	El profesor elige dar de alta una nueva clase
	3	El sistema muestra los campos a rellenar para dar de alta al alumno
	4	El profesor rellena los campos
	5	El sistema confirma que la clase se ha dado de alta en el sistema
	6	El actor sistema vuelve al paso 1
Postcondiciones	Los datos son dados de alta en el sistema	
Variaciones	**Paso**	**Accion**
Excepciones	6	No se puede dar de alta una clase que ya está dada de alta
Observaciones	Hay que rellenar todos los campos obligatorios	

Tabla 27: Descripción caso de uso: Profesor - Crear clase

14.1.25 Profesor - Consultar clase

Caso de uso	**34**	Consultar clase
Descripción	Permite consultar una clase	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de consultar una clase
	2	El profesor elige consultar una clase
	3	El sistema muestra en pantalla los campos con la información actual de la clase
	4	El profesor cuando termina de consultarla le da a aceptar
	5	El sistema vuelve al paso 1
Postcondiciones	Se muestran los datos de una clase	
Variaciones	**Paso**	**Accion**
Excepciones	5	No se puede consultar una clase que no esté dada de alta en el sistema
Observaciones		

Tabla 28: Descripción caso de uso: Profesor - Consultar clase

14.1.26 Profesor - Modificar clase

Caso de uso	**35**	Modificar clase
Descripción	Permite modificara una clase	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de modificar una clase
	2	El profesor elige modificar una clase
	3	El sistema muestra en pantalla los campos con la información actual de la clase para poder modificarla
	4	El profesor modifica los campos que desea
	5	El sistema informa que los datos han sido modificados
	6	El sistema vuelve al paso 1
Postcondiciones	Los datos de la clase son modificados en el sistema	
Variaciones	**Paso**	**Accion**
Excepciones	6	No se puede modificar una clase que no esté dada de alta en el sistema
Observaciones	Hay que rellenar los campos obligatorios	

Tabla 29: Descripción caso de uso: Profesor - Modificar clase

14.1.27 Profesor - Eliminar clase

Caso de uso	**36**	Eliminar clase
Descripción	Permite eliminar una clase	
Actores	Profesor	
Precondiciones	El profesor tiene que estar autentificado	
Flujo normal	**Paso**	**Accion**
	1	El sistema muestra al profesor la posibilidad de eliminar una clase
	2	El profesor elige eliminar una clase
	3	El sistema informa que los datos han sido eliminados
	4	El sistema vuelve al paso 1
Postcondiciones	Los datos son eliminados del sistema	
Variaciones	**Paso**	**Accion**
Excepciones	4	No se puede eliminar una clase que no está dada de alta en el sistema
Observaciones		

Tabla 30: Descripción caso de uso: Profesor - Eliminar clase

Bibliografía y referencias

[1] Roger S. Pressman.*Ingeniería del Software. Un enfoque práctico*. Séptima edición. McGraw-Hill. 2010.

[2] Larman, Craig.*UML Y PATRONES. Una introducción al análisis y diseño orientado a objetos y al proceso unificado*. Segunda edición. Prentice Hall. 2003.

[3] James Rumbaugh; Ivar Jacobson; Grady Booch.*El Proceso Unificado de desarrollo de software.* Addison Wesley.2000.

[4] James Rumbaugh; Ivar Jacobson; Grady Booch.*El lenguaje unificado de modelado. Manual de referencia.* Addison Wesley.2000.

[5] Amescua Seco, Antonio.*Ingeniería del software de gestión: Análisis y diseño de aplicaciones*. Primera edición. Paraninfo. 1995.

[6] Blé Jurado, Carlos.*Diseño Ágil con TDD*. Edición digital. 2010.

[7] Universidad Pólitécnica de Madrid. Manuales Ingenería del Software. http://is.ls.fi.upm.es/docencia/is2/documentacion/. Ultima consulta: Enero 2017.

[8] Hartl, Michael. Ruby on Rails 5 Tutorial. Learn Web Development with Rails. https://www.railstutorial.org/book. Ultima consulta: Diciembre 2016.

[9] Ruby on Rails Guides (v5.0.1). http://guides.rubyonrails.org/v5.0.1/. Ultima consulta: Enero 2017.

[10] Ruby on Rails Guides. Rails Internationalization (I18n) API. http://guides.rubyonrails.org/i18n.html. Ultima consulta: Enero 2017.

[11] Repositorio GitHub. https://github.com/carrierwaveuploader/carrierwave. Ultima consulta: Enero 2017.

[12] Cloudinary. http://cloudinary.com/documentation/rails_carrierwave. Ultima consulta: Enero 2017.

[13] Behaviour Driven Development for Ruby. Making TDD Productive and Fun. http://rspec.info/. Ultima consulta: Octubre 2016.

[14] Computación en la nube - Master en Ingeniería Informática. Desarrollo de aplicaciones para la nube. http://calm-coast-85106.herokuapp.com/frontmatter. Ultima consulta: Enero 2017.

[15] Heroku Dev Center. https://devcenter.heroku.com/. Ultima consulta: Enero 2017.

[16] Bootstrap 3, el manual oficial. Libros web. http://librosweb.es/libro/bootstrap_3/. Ultima consulta: Julio 2016.

[17] Leslie Lamport *LaTex : A document Preparation System*. Addison-Wesley, 1986.

[18] Christian Rolland *LaTex guide pratique*. Addison-Wesley, 1993.

[19] Miniejercicios con LaTex. http://minisconlatex.blogspot.com.es/. Ultima consulta: Enero 2017.

[20] Preparación de textos con LaTex. http://www.uv.es/~ivorra/Latex/LaTeX.pdf. Ultima consulta: Enero 2017.

[21] Introducción a la edición de textos con latex. http://www.emis.de/journals/RCE/IntroLatex/IntroLaTeX.html. Ultima consulta: Enero 2017.

[22] Curso de Introcción a LaTex. http://www.sindominio.net/~xabier/curso_latex/. Ultima consulta: Enero 2017.

[23] Sharelatex. https://es.sharelatex.com/learn/Main_Page. Ultima consulta: Enero 2017.