



Departamento de computación

Facultad de Informática

UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Desarrollo de un sistema de gestión de ejercicios de programación para docencia

Estudiante: David Antelo del Río
Director/a/es/as: Óscar Fontenla Romero
Francisco Javier Bellas Bouza

A Coruña, 31 de agosto de 2019.

A mi familia y amigos

Agradecimientos

A los investigadores Francisco Javier Bellas Bouza y Óscar Fontenla Romero, por su guía y ayuda, tanto en la planificación del proyecto, como en su desarrollo.

A mi familia, y muy particularmente a mis padres, por su apoyo incondicional durante no solo el desarrollo de este proyecto, sino también el largo camino que ha resultado ser la carrera.

A todos mis amigos, y muy especialmente, a mis compañeros y ex-compañeros de piso, por todas las risas compartidas, y las largas noches de videojuegos.

Resumen

El objetivo de este proyecto, enmarcado en el campo del *desarrollo software*, ha sido crear una aplicación que permitiera, tanto a profesores como alumnos, visualizar y modificar ejercicios de alguna materia (aunque el foco concreto del proyecto hayan sido ejercicios de programación), así como los exámenes, prácticas, y prácticas evaluables en los que hayan sido utilizados. Por un lado, esta aplicación tiene como fin facilitar el trabajo de los docentes implicados, dándoles acceso a todo tipo de funcionalidades orientadas a la gestión y el control de los ejercicios y pruebas introducidas en el sistema, tales como el filtrado y ordenado de elementos por distintos criterios, la creación de bocetos de ejercicios que serán posteriormente refinados, o la descarga en de los enunciados de los ejercicios contenidos en alguna prueba, en un formato adecuado. Por otro, el sistema también está pensado para el uso por parte de alumnos, permitiendo que estos tengan acceso a aquellos recursos que los profesores hayan considerado apropiados, que les servirán de referencia de cara a las pruebas evaluatorias a las que se tendrán que enfrentar. Al igual que los profesores, dispondrán de numerosos criterios de filtrado y ordenado que garantizarán su control sobre los recursos, así como de opciones de descarga de los enunciados de estos ejercicios.

Teniendo en cuenta las características de esta aplicación, parece bastante evidente que lo que buscamos es un sistema **rápido, seguro, de fácil uso**, y que sea **accesible para el mayor número de usuarios posible**. Es por esto por lo que se ha decidido crear una **aplicación web responsive**, que siga las normas del diseño *Material*, y que implemente los mecanismos de seguridad pertinentes que garanticen que el *software* no tiene vulnerabilidades importantes, utilizando para todo ello herramientas y algoritmos apropiados que aseguren que el tiempo de respuesta de la aplicación es bajo.

Para realizar este proyecto, se ha seguido la metodología conocida como *Proceso Unificado de Desarrollo de Software*. Esta metodología propone el desarrollo de un esquema iterativo e incremental, en donde las iteraciones se centran en aspectos relevantes del *software*.

Para el desarrollo de la aplicación, nos hemos valido únicamente de herramientas de *software libre*, lo cual nos proporcionará numerosas ventajas. Además, se han realizado las pruebas que se han considerado necesarias para garantizar la calidad del *producto software* que ha sido entregado al cliente al concluir su desarrollo.

La aplicación final cumple con todos los requisitos, funcionales y no funcionales, establecidos al inicio del proyecto. Sin embargo, el *software* ha sido desarrollado con el objetivo de que pudiera ser fácilmente ampliado o modificado en algún momento en el futuro. Lo que aquí se presenta es una *plataforma de aprendizaje* muy simplificada, que solo cuenta con las funcionalidades esenciales que se esperarían de una aplicación de estas características. Si así

se decidiera, esta podría actuar de base para construir una *plataforma de aprendizaje* completa, equiparable a *Moodle* o a cualquiera de sus competidoras.

Palabras clave:

- Aplicación web
- Responsive
- Proceso Unificado de Desarrollo Software
- Base de datos relacional
- Patrón MVC
- Seguridad
- Software libre

Abstract

The goal of this project, framed in the *software development* field, has been to create an application that allowed students and teachers to visualize and modify exercises of some subject (although the main focus of the project have been programming exercises), as well as the exams, practices, and tests in which they have been used. On one hand, the main focus of this application is to facilitate the work of the teacher, giving them access to all kinds of features related to the management of these elements, like filtering and sorting exercises, creating sketches of exercises, or downloading these exercises in a readable format. On the other hand, the software system can also be used by students, allowing them to access resources that will be useful when getting ready for tests. They will also have available similar features to those of the teachers.

Taking into consideration the characteristics of the application, it should be obvious that we want the system to be **fast, secure, easy to use, and accesible for as many people as possible**. These are the reasons that made us decide to create a **responsive web application**, following the guidelines of the *Material* design philosophy, and that implements security features so that no important vulnerabilities are found in our software, using for this purpose adequate tools and algorithms that guarantee that the response time is low.

To develop this project, we have followed the *Unified Software Development Process* philosophy, which makes use of an iterative and incremental strategy, in which iterations focus on important aspects of the software.

To develop the application, we have exclusively used freeware, which will show many advantages later on. Besides, we have tested the application components in the ways we thought were necessary to guarantee the quality of the final product.

The final application fulfills all of the requirements the client provided, both functional and non-functional. However, the software was developed in such a way that it shouldn't be difficult to modify at some point in the future if necessary. What is presented in this document is a simplified version of a *learning platform*, that only provides the basic functionalities usually expected from this kind of applications.

Keywords:

- Web application
- Responsive
- Unified Software Development Process
- Relational database
- MVC pattern
- Security
- Freeware

Hardware y Software:

- Windows 7 Ultimate SP1
- Eclipse v2018-12 (4.10.0)
- JDK 1.8.0
- Apache Tomcat v9.0
- MySQL Server v8.0.13
- Hibernate 5.4.0.CR2
- Spring Framework (varios módulos)
- Spring Security (varios módulos)
- JUnit v4.12
- MaterializeCSS v1.0.0

Índice general

1	Introducción	1
1.1	Motivación	1
1.2	Objetivos	1
2	Antecedentes	5
2.1	Plataformas de aprendizaje	5
2.1.1	Moodle	5
2.1.2	TalentLMS	6
2.1.3	Docebo	6
2.2	Conclusiones	7
3	Fundamentos tecnológicos	9
3.1	Principios teóricos	9
3.1.1	Aplicación web	9
3.1.2	Modelo-Vista-Controlador	9
3.1.3	Seguridad de aplicaciones web	12
3.1.4	Bases de datos relacionales	12
3.1.5	Pruebas de software	14
3.1.6	Material	16
3.1.7	Diseño responsive	16
3.2	Tecnología utilizada	18
3.2.1	Entorno de desarrollo: Eclipse	18
3.2.2	Lenguaje de programación: Java	19
3.2.3	Gestor de base de datos: MySQL	20
3.2.4	Mapeador Objeto-Relacional: Hibernate	21
3.2.5	Pruebas de código: JUnit	21
3.2.6	Framework de estilo de la interfaz: MaterializeCSS	22
3.2.7	Funciones varias: Spring	23

4 Metodología	25
4.1 Proceso Unificado	25
4.1.1 Lenguaje Unificado de Modelado	26
4.2 Metodología a utilizar	27
4.3 Roles	27
5 Desarrollo	29
5.1 Fase de Inicio	29
5.2 Iteración 1	29
5.2.1 Requisitos	29
5.2.2 Análisis	32
5.2.3 Diseño	33
5.2.4 Revisión de la iteración	33
5.3 Fase de Elaboración	34
5.4 Iteración 2	34
5.4.1 Requisitos	34
5.4.2 Análisis	37
5.4.3 Diseño	37
5.4.4 Implementación	46
5.4.5 Pruebas	46
5.4.6 Revisión de la iteración	46
5.5 Fase de Construcción	46
5.6 Iteración 3	47
5.6.1 Requisitos	47
5.6.2 Análisis	47
5.6.3 Diseño	47
5.6.4 Implementación	47
5.6.5 Pruebas	48
5.6.6 Revisión de la iteración	48
5.7 Iteración 4	48
5.7.1 Requisitos	48
5.7.2 Análisis	49
5.7.3 Diseño	51
5.7.4 Implementación	53
5.7.5 Pruebas	54
5.7.6 Revisión de la iteración	55
5.8 Iteración 5	55
5.8.1 Requisitos	55

5.8.2	Análisis	55
5.8.3	Diseño	56
5.8.4	Implementación	56
5.8.5	Pruebas	58
5.8.6	Revisión de la iteración	58
5.9	Fase de Transición	58
5.10	Iteración 6	58
6	Planificación	61
6.1	Planificación inicial	61
6.2	Planificación de la fase de inicio	62
6.2.1	Iteración 1	62
6.3	Planificación de la fase de elaboración	62
6.3.1	Iteración 2	63
6.4	Planificación de la fase de construcción	64
6.4.1	Iteración 3	64
6.4.2	Iteración 4	65
6.4.3	Iteración 5	65
6.5	Planificación de la fase de transición	66
6.5.1	Iteración 6	67
6.6	Coste del proyecto	68
6.7	Revisión de la planificación	69
7	Funcionalidades destacadas	71
8	Pruebas	73
8.1	Pruebas backend	74
8.2	Pruebas ad-hoc	74
8.3	Pruebas de interfaz	74
8.4	Pruebas de compatibilidad de navegador	74
8.5	Pruebas de aceptación	75
8.6	Pruebas alpha	75
8.7	Pruebas beta	75
9	Conclusiones	77
9.1	Lecciones aprendidas	78

10 Trabajo futuro	79
10.1 Ampliar los datos de usuario	79
10.2 Crear un foro dentro de la web	79
10.3 Internacionalizar la aplicación	80
10.4 Integrar un editor de texto enriquecido	80
10.5 Validar y mejorar la seguridad de la página	80
A Glosario de acrónimos	83
B Terminología	85
C Manual de usuario	87
C.1 Descripción general del sistema	87
C.2 Usuarios	87
C.3 Barra de navegación	88
C.4 Ejercicios	88
C.4.1 Listado de ejercicios	88
C.4.2 Visualización de ejercicios	90
C.4.3 Edición de ejercicios	90
C.5 Conjuntos	90
C.5.1 Listado de conjuntos	90
C.5.2 Visualización de conjuntos	91
C.5.3 Modo edición de conjuntos	91
C.5.4 Edición de atributos de conjuntos	92
C.5.5 Selección de ejercicios del usuario	92
C.6 Funcionalidades específicas para Profesores	92
C.7 Funcionalidades específicas para Administradores	93
D Imágenes de la aplicación	95
Bibliografía	103

Índice de figuras

3.1	Diagrama de funcionamiento de aplicaciones MVC	11
3.2	Base de datos relacional	13
3.3	Diseño Material en una web	17
3.4	Filosofía de diseño <i>responsive</i>	17
5.1	Modelo de casos de uso (iteración 1)	31
5.2	Diagrama de secuencia del caso de uso <i>Crear Ejercicio</i> (iteración 1)	32
5.3	Diagrama de secuencia del caso de uso <i>Actualizar Ejercicio</i> (iteración 1)	33
5.4	Casos de uso relativos a Ejercicio (iteración 2)	36
5.5	Casos de uso del relativos a Conjunto (iteración 2)	36
5.6	Diagrama de secuencia del subsistema de gestión de documentos (iteración 2)	38
5.7	Diagrama de secuencia del subsistema de listado de ejercicios (iteración 2)	39
5.8	Diagrama <i>Entidad-Relación</i> de la base de datos (iteración 2)	40
5.9	Diagrama de paquetes de la aplicación (iteración 2)	41
5.10	Diagrama de clases del módulo <i>implementación</i> , dentro de la capa <i>modelo</i> (iteración 2)	42
5.11	Diagrama de clases del módulo <i>implementación</i> , dentro de la capa <i>rich-entity</i> (iteración 2)	43
5.12	Diagrama de clases del módulo <i>filtros</i> , dentro de la capa <i>rich-entity</i> (iteración 2)	43
5.13	Diagrama de clases del módulo <i>implementación</i> , dentro de la capa <i>controlador</i> (iteración 2)	44
5.14	Mockup de la vista de un ejercicio en un PC (iteración 2)	45
5.15	Mockup de la vista de un ejercicio en un móvil (iteración 2)	45
5.16	Diagrama de secuencia de la creación de un nuevo usuario (iteración 4)	50
5.17	Diagrama <i>Entidad-Relación</i> de la base de datos (iteración 4)	52
5.18	Diagrama de paquetes de la aplicación (iteración 4)	53
5.19	Mockup de la vista del listado de ejercicios en un PC (iteración 5)	56

5.20	Mockup de la vista del listado de ejercicios en una <i>tablet</i> (iteración 5)	57
5.21	Mockup de la vista del listado de ejercicios en un móvil (iteración 5)	57
6.1	Diagrama de <i>Gantt</i> inicial	62
6.2	Diagrama de <i>Gantt</i> (iteración 1)	63
6.3	Diagrama de <i>Gantt</i> (iteración 2)	63
6.4	Diagrama de <i>Gantt</i> al finalizar la iteración 2	64
6.5	Diagrama de <i>Gantt</i> (iteración 3)	65
6.6	Diagrama de <i>Gantt</i> al finalizar la iteración 3	65
6.7	Diagrama de <i>Gantt</i> (iteración 4)	66
6.8	Diagrama de <i>Gantt</i> al finalizar la iteración 4	66
6.9	Diagrama de <i>Gantt</i> (iteración 5)	67
6.10	Diagrama de <i>Gantt</i> al finalizar la iteración 5	67
6.11	Diagrama de <i>Gantt</i> (iteración 6)	67
6.12	Diagrama de <i>Gantt</i> al finalizar la iteración 6	68
8.1	Cobertura de los tests automatizados	73
D.1	Listado de ejercicios	96
D.2	Vista de un ejercicio	97
D.3	Vista de un ejercicio en un móvil	98
D.4	Vista de la barra de navegación desplegable en tablet o móvil	99
D.5	Listado de exámenes	100
D.6	Edición de los elementos que contiene un examen y sus posiciones	101
D.7	Selección de ejercicios del usuario	102

Índice de cuadros

5.1	Riesgos identificados al comienzo del proyecto	30
6.1	Trabajo previsto y actual del proyecto	68
6.2	Coste previsto y actual del proyecto	68

Introducción

1.1 Motivación

LA necesidad de realizar este proyecto surge de un problema intrínseco de cualquier sistema en el que se trabaje recurrentemente con información reaprovechable: necesitamos poder gestionar de alguna manera esta información, para así ahorrar tiempo, dinero, y poder tener en cuenta como están distribuidos los datos. En este caso, el problema particular del que debemos ocuparnos es el almacenamiento y manipulación de los ejercicios de programación, y su información asociada, diseñados para la asignatura de *Informática* de los grados de *Ingeniería Industrial* de la *UDC*.

La metodología utilizada en esta asignatura implica que se planteen 3 o 4 ejercicios diferentes cada semana de curso para cada grupo, lo que da lugar a unos 10 ejercicios distintos por semana. Ha llegado un momento en el que cada vez es más complicado encontrar ejercicios originales, y también lo es recurrir a los ya usados anteriormente en la asignatura, debido a la falta de documentación. Es por esto por lo que se requiere de un *software* especializado y adaptado a nuestro problema que permita almacenar, modificar, etiquetar, y ordenar los ejercicios de programación que se diseñen para esta asignatura, con sus respectivos recursos asociados, así como los exámenes, prácticas, y prácticas evaluables en los que hayan sido utilizados.

Hay que tener en cuenta que a pesar de que esta ha sido la motivación principal para desarrollar el proyecto, la aplicación final será lo suficientemente flexible como para poder ser utilizada en otros casos de índole similar.

1.2 Objetivos

El objetivo de este proyecto ha sido crear una aplicación capaz de simplificar el almacenamiento y la gestión de ejercicios de programación para su uso en la docencia de una asignatura oficial. Para lograr este objetivo, que es bastante abstracto en sí mismo, se han

planteado una serie de metas más concretas a alcanzar, que garantizarían la consecución del objetivo principal. Estas metas, a grandes rasgos, son:

- El sistema debe permitir el alta / baja de usuarios, que podrán ser alumnos, profesores, o administradores.
- El sistema debe permitir a los profesores la creación, edición, visualización, y borrado, de ejercicios de programación, con toda su información y recursos asociados, así como de exámenes, prácticas, y prácticas evaluables, con sus correspondientes ejercicios.
- El sistema debe permitir el almacenamiento de archivos como recursos asociados a los ejercicios.
- El sistema debe permitir la descarga de los enunciados de los ejercicios en algún formato adecuado.
- El sistema debe permitir hacer búsquedas de ejercicios o pruebas, utilizando para ello numerosos criterios de filtrado, que serán distintos según los privilegios del usuario.
- El sistema debe permitir la creación, edición, y borrado, de bocetos de ejercicio, que serán posteriormente refinados.

Además, existen una serie de requisitos que debíamos cumplir para que la aplicación estuviera a la altura de las expectativas del cliente. A grandes rasgos, estos son:

- **Que la aplicación sea de fácil uso:** dado el carácter didáctico de la aplicación, nos interesa que el manejo de la aplicación sea sencillo e intuitivo, y su visualización, clara, más que buscar un diseño excesivamente recargado y plagado de funcionalidades innecesarias.
- **Que la aplicación sea rápida:** por el mismo motivo que el punto anterior, una de las características más importantes de la aplicación debe ser la velocidad. Es por ello por lo que se debe elegir un diseño visual sencillo, que no ralentice el *software* con componentes innecesarios, así como seleccionar cuidadosamente los algoritmos que se utilizarán para obtener los datos que serán mostrados al usuario.
- **Que la aplicación abarque al mayor número de usuarios potenciales posible:** es importante que todos los alumnos y profesores involucrados en el ámbito para el que se desarrollará la aplicación puedan acceder a ella desde sus distintos dispositivos, sin importar cuáles sean.

- **Que la aplicación sea segura:** en una aplicación de estas características, es importante garantizar la seguridad e integridad de los datos, y asegurarse de que ningún usuario no autorizado tiene acceso a áreas restringidas.

Antecedentes

ESTE capítulo está dedicado al estado del arte. Aquí, se expondrán aquellas aplicaciones y trabajos relacionados con la temática del proyecto actual, con la intención de analizar sus características y extraer información que podría ser útil con vistas a nuestro proyecto.

2.1 Plataformas de aprendizaje

Las *plataformas de aprendizaje* son aplicaciones cuyo objetivo es proporcionar a educadores, administradores, y estudiantes, un sistema integrado único, robusto, y seguro, para crear ambientes de aprendizaje personalizados. En esta sección, hablaremos de algunas de las más importantes, aunque no entraremos en demasiado detalle dado que todas ellas son relativamente similares entre sí.

2.1.1 Moodle

*Moodle*¹ está construido por el proyecto *Moodle*, que está, a su vez, dirigido y coordinado por el *Cuartel General Moodle*, soportado financieramente por una red mundial de cerca de 80 compañías de servicio, también llamadas *Moodle Partners*.

Moodle se utiliza actualmente en decenas de miles de entornos de aprendizaje a nivel mundial, entre los cuales se encuentran algunos de los centros de enseñanza más prestigiosos del planeta, como pueden ser la *London School of Economics* o la *Universidad Estatal de Nueva York*. Además, es la plataforma de aprendizaje con mayor número de usuarios en todo el globo. Esto se debe, en parte, a que está disponible en numerosos idiomas.

La plataforma tiene a sus espaldas más de 10 años de desarrollo guiados por la *pedagogía de constructivismo social*, y tiene como principal objetivo el aprendizaje colaborativo entre el estudiante y el profesor.

¹<https://moodle.org/>

Posee una interfaz de usuario simple, con características *drag-and-drop*, y todas sus funcionalidades están cubiertas por la documentación. Esto hace de *Moodle* una plataforma fácil de aprender y de usar. La plataforma es gratuita y de código abierto. Esto tiene por consecuencia la existencia de numerosos *plugins* que cubren distintos apartados de los que carece la plataforma en su versión original.

La UDC tiene una página web ² **basada en la plataforma *Moodle***, utilizada tanto por alumnos como profesores para compartir recursos relacionados con las asignaturas de las distintas carreras que se imparten en la universidad. Ha sido usada durante años y se ha mantenido como una herramienta importante en el desarrollo de la enseñanza en las diferentes facultades de la UDC.

2.1.2 TalentLMS

TalentLMS es una *plataforma de aprendizaje* almacenada en la nube altamente configurable cuya finalidad es facilitar la tarea de impartir cursos y seminarios a través de internet. Posee un gran número de usuarios, y es utilizada por algunas de las compañías más importantes del mundo, como *LG* o *HarperCollins*. Las principales características de *TalentLMS* son las siguientes:

- Es flexible y puede ser adaptado al entorno concreto en el que se está utilizando.
- Gracias al almacenamiento en la nube, no existe necesidad de instalar o actualizar ningún *software*.
- Está preparado y optimizado para funcionar en todo tipo de dispositivos, y en particular aquellos de *Apple* y *Android*.
- Posee herramientas para fomentar la interactividad de los usuarios, que permiten, por ejemplo, crear pequeños "juegos" con fines didácticos.
- Utiliza una interfaz minimalista y sencilla que permite a los usuarios centrarse en el "qué" en lugar de el "cómo".

2.1.3 Docebo

Docebo es una *plataforma de aprendizaje online* completamente integrada y lista para su uso en dispositivos móviles, que está disponible a nivel mundial en más de 30 idiomas. Entre sus funcionalidades más importantes podemos destacar que permite realizar videoconferencias y que implementa funcionalidades de *blog*, de modo que los usuarios pueden crear

²<https://moodle.udc.es/>

publicaciones y comentar o puntuar las de otros, lo cual hace que el proceso de aprendizaje sea mucho más interactivo.

2.2 Conclusiones

Adaptar una *plataforma de aprendizaje* a nuestro caso particular plantea ciertos problemas. En primer lugar, se descartaron todas aquellas plataformas más orientadas a la utilización empresarial que docente, tales como *TalentLMS*, por el simple motivo de que existen mejores alternativas para el proyecto que se pretende desarrollar. De las restantes, se consideró que *Moodle* era la mejor opción, puesto que es la plataforma más utilizada en el mundo, y el hecho de que sea utilizada por nuestra universidad para una función similar a la que pretendemos darle a nuestra aplicación hace que además tengamos la garantía de que, muy probablemente, sirva como base. Sin embargo, esto no resuelve todos los problemas resultantes de reconvertir una plataforma existente para nuestro propósito.

Por un lado, la plataforma es demasiado **compleja** para nuestros intereses. Buscamos crear una aplicación sencilla y rápida. *Moodle* incluye demasiadas funcionalidades innecesarias, que lo único que harán será **ralentizar el software** y **entorpecer al usuario**. Además, *Moodle* no está pensado para almacenar las relaciones que nos interesa establecer entre las entidades de nuestro programa, tales como la existente entre un examen y sus ejercicios. Por tanto, tendríamos que modificar el *software* para incorporar estas funcionalidades, así como eliminar gran parte de código existente, que sería innecesario en nuestro proyecto. Evidentemente, esto, aunque es una opción a tener en cuenta, **está lejos de ser ideal**, puesto que estamos construyendo una aplicación a partir de otra con la que no comparte más que los fundamentos.

Podemos, por tanto, concluir que, aunque *Moodle* es una plataforma interesante para nuestro proyecto, dado que tiene ciertas características en común con él, **no tiene suficiente en común** como para justificar el ahorro en esfuerzo y coste que supondría construir una nueva aplicación desde cero. Sin embargo, sí nos será útil tener en cuenta las funcionalidades de esta plataforma, ya que podríamos valernos de algunas de ellas.

Fundamentos tecnológicos

EL objetivo de este capítulo es describir las tecnologías utilizadas y exponer los motivos de su elección para este proyecto concreto, así como describir brevemente las bases teóricas en las que se apoya el desarrollo de la aplicación.

3.1 Principios teóricos

Esta sección estará dedicada a explicar los principales conceptos teóricos en los que se apoya este proyecto. Al tratarse del desarrollo de una aplicación web, algo bastante común dentro del campo de la informática, la mayoría de ellos deberían ser familiares para cualquier persona letrada en la materia, sirviendo esta sección como recordatorio más que como material didáctico.

3.1.1 Aplicación web

En la *ingeniería de software*, se denomina **aplicación web** a una aplicación *cliente/servidor* que usa el navegador como cliente y proporciona un servicio interactivo conectándose con otros servidores en internet (o en una intranet) [1]. En otras palabras, es un programa que se codifica en un lenguaje interpretable por los navegadores web en el que se confía la ejecución al navegador. Aunque existen muchas variaciones posibles, una aplicación web está normalmente estructurada como una aplicación de tres capas que se corresponden con las del *patrón MVC*.

3.1.2 Modelo-Vista-Controlador

Modelo-Vista-Controlador, también conocido como *MVC*, es un **patrón de arquitectura de software** que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Este patrón se basa en las ideas de reutilización de código y la separación de conceptos, características que

buscan facilitar la tarea del desarrollo de aplicaciones, y su posterior mantenimiento. Este patrón propone la creación de tres componentes, independientes entre sí:

- **Modelo:** capa representación de la información con la que trabaja el sistema. Es, consecuentemente, la encargada de gestionar todos los accesos a esa información, tanto consultas como actualizaciones.
- **Controlador:** responde a eventos (normalmente peticiones de usuario), e invoca al *Modelo* cuando se solicita la información que este contiene. Actúa como intermediario entre el *Modelo* y la *Vista*.
- **Vista:** se encarga de presentar la información del *Modelo* en un formato legible y adecuado para la interacción del usuario. La separación entre componentes permite que la *Vista* se adapte al dispositivo en el que está siendo visualizada (concepto conocido como *responsive*) así como a las preferencias del usuario [1].

Estos elementos interactúan entre sí compartiendo y manipulando la información del sistema para atender las peticiones del usuario. El proceso que sigue el funcionamiento de una aplicación que implementa el *patrón MVC* es el siguiente:

1. El usuario interactúa con la interfaz proporcionada por la *Vista*.
2. El *Controlador* recibe la notificación de la acción solicitada por el usuario.
3. El *Controlador* delega en el *Modelo* para actualizar los datos en función de la petición del usuario, en caso de que fuera necesario.
4. El *Controlador* solicita al *Modelo* los datos a mostrar pertinentes en función de la petición de usuario anterior, y los devuelve a la *Vista*.
5. La *Vista* muestra los datos proporcionados por el controlador en un formato adecuado.
6. La *Vista* espera peticiones de usuario, comenzando de nuevo el ciclo.

En la [Figura 3.1](#) podemos ver una representación gráfica de la relación existente entre las capas.

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 ayudan a implementarlo. Este patrón tiene una serie de ventajas e inconvenientes ¹, que se describen a continuación.

¹<https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/>

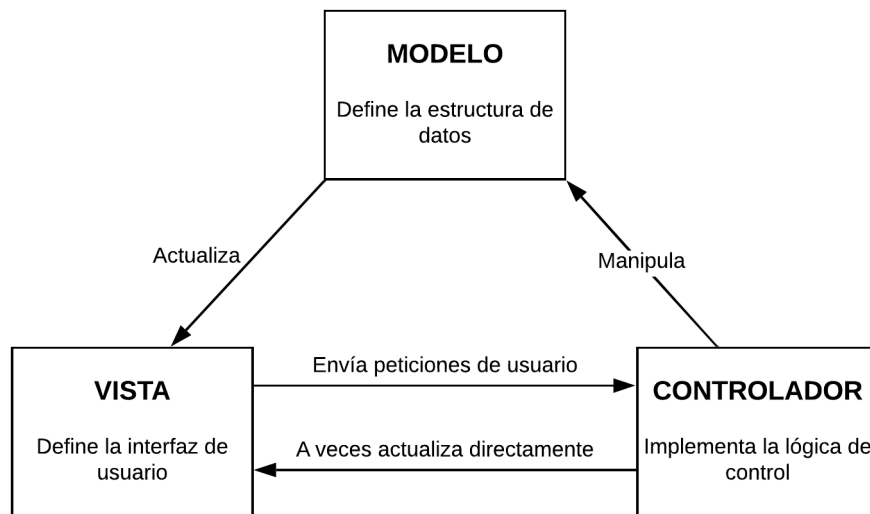


Figura 3.1: Diagrama de funcionamiento de aplicaciones MVC

Ventajas:

- **Proceso de desarrollo rápido:** por la desacoplabilidad entre componentes que aporta *MVC*, es sencillo trabajar en paralelo y desarrollar las tres capas simultáneamente.
- **Capacidad de proporcionar múltiples vistas para un mismo modelo**
- **Pequeñas modificaciones no afectan a todo el sistema:** cualquier modificación que se haga en cuanto a como se presenta la información al usuario es independiente del modelo que utiliza por debajo.
- **MVC devuelve los datos sin formato:** el *Controlador* envía los datos sin formatear a la *Vista*, lo cual permite que sea la *Vista* la que decida qué mostrar y cómo mostrarlo.

Desventajas:

- Mayor complejidad
- Insuficiente acceso a datos desde la vista
- Dificultad a la hora de integrar el patrón *MVC* con las interfaces de usuario modernas
- Se necesitan múltiples programadores para que la paralelización del desarrollo de los componentes sea aprovechable
- Se requieren conocimientos de múltiples tecnologías

3.1.3 Seguridad de aplicaciones web

La **seguridad de aplicaciones web** es una rama de la Seguridad Informática que se encarga específicamente de la seguridad de sitios web, aplicaciones web, y servicios web.

Con la aparición de la *Web 2.0*, el intercambio de información a través de redes sociales y el crecimiento de los negocios en la adopción de la web como un medio para hacer negocios y ofrecer servicios, los sitios web son constantemente atacados. Los *hackers* buscan comprometer la red de la corporación o a los usuarios finales. Como resultado, la industria cada vez está prestando mayor atención a la seguridad de aplicaciones web. La seguridad web es un problema que le cuesta 60.000 millones de dólares al año a la industria *IT* estadounidense [2]. **OWASP** es el estándar emergente para la seguridad de aplicaciones Web.

OWASP

OWASP (*Open Web Application Security Project*) es un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro.

Los documentos con más éxito de **OWASP** incluyen la Guía **OWASP** y el ampliamente adoptado documento de autoevaluación **OWASP Top 10**². Las herramientas **OWASP** más usadas incluyen el entorno de formación *WebGoat*, la herramienta de pruebas de penetración *WebScarab* y las utilidades de seguridad para entornos *.NET* **OWASP DotNet**. **OWASP** cuenta con unos 50 capítulos locales por todo el mundo y miles de participantes en las listas de correo del proyecto. Son los creadores de la serie de conferencias *AppSec*³ para mejorar la construcción de la comunidad de seguridad de aplicaciones web.

3.1.4 Bases de datos relacionales

La **base de datos relacional** es un mecanismo de almacenamiento de información permanente que permite guardar datos y, opcionalmente, implementar funcionalidades respecto al tratamiento de dichos datos. Cumple con el **modelo relacional**, el modelo más utilizado actualmente para implementar las BD ya planificadas. Este tipo de bases de datos llevan en uso más de 35 años, y han hecho posible la creación de una industria multimillonaria que gira en torno a ellas [3] [4].

Un sistema de *software* utilizado para mantener las bases de datos relacionales es un **Relational Database Management System** (**RDBMS**). Prácticamente todos los sistemas de bases de datos relacionales utilizan **SQL** (*Structured Query Language*) para consultar y mantener la base de datos.

²https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

³https://www.owasp.org/index.php/Category:OWASP_AppSec_Conference

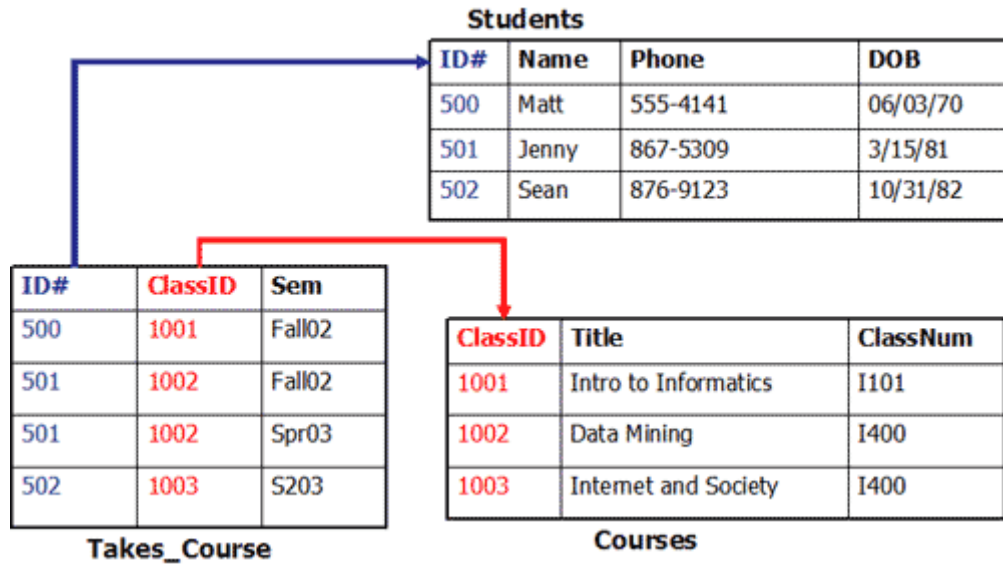


Figura 3.2: Base de datos relacional

Las bases de datos relacionales cumplen ciertas reglas de integridad, que aseguran que los datos contenidos en la tabla estén actualizados y sean siempre accesibles.

Reglas de integridad

Aseguran que la información que aparece en la base de datos sea la correcta, no aparezca por duplicado, y permanezca en todo momento actualizada y accesible. Podemos dividir los tipos de integridad de una base de datos relacional en [3]:

Integridad a nivel de tabla:

- No puede haber registros duplicados en una tabla.
- La clave primaria identifica inequívocamente un registro en una tabla.
- Los valores de la clave primaria no pueden ser nulos.

Integridad a nivel de campo:

- La identidad y propósito de un campo es clara, y es identificable en todas las tablas en las que aparece.
- La definición de un campo es consistente en todas las tablas de la base de datos.
- Los valores de un campo son consistentes y válidos.

- Los tipos de operaciones, comparaciones y modificaciones

Por último, tenemos la **integridad a nivel de relación**, que implica que la relación existente entre dos tablas es sólida y los registros que las relacionan están **sincronizados**, y responden consecuentemente a operaciones que afectan a uno de ellos.

Modelo relacional

El **modelo relacional** es un modelo de datos basado en la lógica de predicados y en la teoría de conjuntos.

Su idea fundamental es el uso de **relaciones**. Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados tuplas. Pese a que esta es la teoría de las bases de datos relacionales creadas por *Codd*, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar, pensando en cada relación como si fuese una tabla que está compuesta por registros y columnas [5]. Dada su flexibilidad, el *modelo relacional* se puede usar como base para la creación de bases de datos de propósito general.

Ventajas:

- Provee herramientas que garantizan evitar la duplicidad de registros.
- Garantiza la integridad referencial. Así, al eliminar un registro, elimina todos los registros relacionados dependientes.
- Favorece la normalización por ser más comprensible y aplicable.

Desventajas:

- Presenta deficiencias con datos gráficos, multimedia, *CAD*, y sistemas de información geográfica.
- Los bloques de texto como tipo de dato no son manipulados de forma eficiente.

3.1.5 Pruebas de software

Las **pruebas de software** son investigaciones empíricas que nos permiten comprobar si la aplicación alcanza unos determinados estándares de calidad. Permiten proporcionar información objetiva e independiente a la parte interesada o *stakeholder*. Es una actividad más en el **proceso de control de calidad** [6].

Podemos clasificar las pruebas, según cual sea su finalidad, en dos grupos: **funcionales**, y **no funcionales**. A continuación, se nombrarán algunas de las pruebas más habituales.

Pruebas funcionales

- **Unitarias:** comprueban el correcto funcionamiento de una unidad de código aislada.
- **De integración:** aquellas que se realizan una vez las pruebas unitarias han sido aprobadas y que tienen por objetivo comprobar que los componentes individuales que funcionan bien por separado también lo hacen cuando interactúan entre ellos.
- **De compatibilidad:** comprueban el comportamiento del *software* en distintos entornos de trabajo.
- **De aceptación:** las realiza el cliente, que evalúa si el software se adecúa a sus expectativas, y cubre los requisitos preestablecidos.

Pruebas no funcionales

- **De seguridad:** verifican que la aplicación cumple unos estándares de seguridad.
- **De stress:** prueban los límites de carga que puede soportar la aplicación.
- **De usabilidad:** pruebas que realizan usuarios finales del producto para verificar que el software cumple con sus expectativas.
- **De rendimiento:** determinan la velocidad con la que el software realiza una tarea bajo unas condiciones de trabajo concretas.
- **De mantenibilidad:** sirven para evaluar cuán fácil es mantener un sistema.

Algo que hay que tener en cuenta a la hora de decidir las pruebas de un proyecto, es **cuándo comenzar las pruebas y cuándo dejar de hacer pruebas** ⁴.

Cuándo comenzar a hacer pruebas

Un comienzo temprano reduce el coste, el tiempo de corrección, y mejora la calidad del producto que es entregado al cliente. Sin embargo, esto dependerá del modelo de desarrollo que se esté usando.

Cuándo dejar de hacer pruebas

Dado que las pruebas no pueden determinar con un 100% de certeza que el software sea correcto, determinar el momento de dejar de realizarlas es más complejo que el caso anterior. Sin embargo, hay una serie de parámetros que debemos considerar a la hora de tomar esta decisión.

⁴http://moodle.nccu.edu.tw/pluginfile.php/77731/mod_resource/content/1/software_testing%281%29.pdf pag. 2

- *Deadlines* de pruebas
- Completitud de los casos de ejecución
- Completitud del *coverage* del código.
- El porcentaje de bugs está por debajo de un cierto nivel y no se identifican bugs de alta prioridad
- Decisiones de la directiva

3.1.6 Material

Material⁵ es una normativa de diseño, creada en inicio para los sistemas *Android*, aunque que ya ha sido adaptada a todos los dispositivos dado que ha sido implementada por numerosas webs.

El diseño *Material* es más limpio que sus antecesores (*Hera*, *Quantum Paper*), y en él predominan animaciones y transiciones de respuesta, el relleno, y los efectos de profundidad tales como la iluminación y las sombras. *Material* tiene por objetivos:

- Crear un lenguaje visual que sintetice los principios básicos del buen diseño y los integre con las innovaciones que aparezcan.
- Desarrollar un único sistema que unifique la experiencia de usuario en todas las plataformas, y para los diferentes periféricos de entrada existentes.
- Proporcionar la flexibilidad suficiente como para que el diseño se ajuste a las necesidades del programador y del problema concreto.

Hoy en día existen varias librerías gráficas que ayudan a implementar el diseño *Material*, y ahorran al programador las tareas más tediosas en cuanto a la programación de la capa vista. Estas han ayudado a popularizar dicha filosofía de diseño. En la [Figura 3.3](#), podemos ver el aspecto de una web que implementa la filosofía *Material*.

3.1.7 Diseño responsive

Responsive es una filosofía de diseño y desarrollo cuyo objetivo es adaptar la apariencia de las páginas web al dispositivo que se esté usando para visualizarlas.

Esta filosofía surge de la necesidad de dar respuesta a la creciente variedad de dispositivos existentes que permiten acceso a internet, que además cuentan con diferentes modelos y formatos de pantalla. Debemos tener en cuenta que es extremadamente importante proporcionar

⁵<https://material.io/design/>

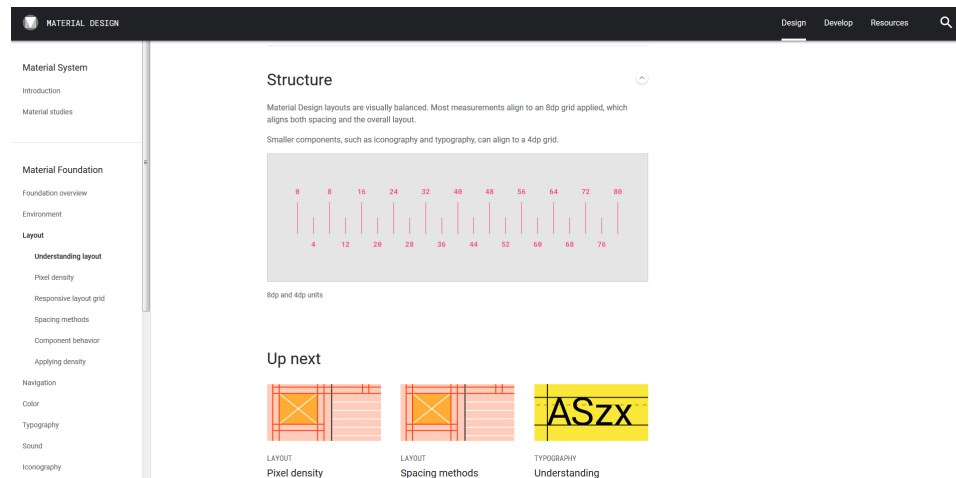


Figura 3.3: Diseño Material en una web

al usuario la mejor experiencia de navegación posible, sobre todo en páginas comerciales, y es este el motivo de que hoy sea imprescindible reescalar y manipular la interfaz para hacer que sea clara y legible para todos los usuarios [7] [8]. En la [Figura 3.4](#) podemos ver un boceto de como se vería una web *responsive* en distintos dispositivos.

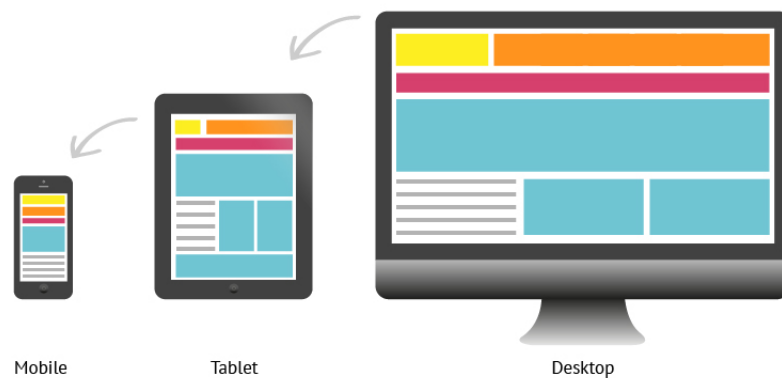


Figura 3.4: Filosofía de diseño *responsive*

Lo que, en principio, se necesita para crear una web *responsive*, es:

- Un *layout* flexible que utilice una cuadrícula
- Imágenes y demás *media* también flexibles

- *Media queries*, un módulo de la especificación CSS3

A nivel práctico, el diseño *responsive* es posible gracias a la introducción de **media queries**⁶ en las propiedades de los estilos CSS en su versión número 3. Estas son una serie de órdenes que se incluyen en la hoja de estilos que indican al documento *HTML* como debe comportarse según la resolución de pantalla.

3.2 Tecnología utilizada

Esta sección del capítulo estará dedicada exponer qué herramientas *software* se utilizaron y el porque de su elección.

A pesar de que cada componente de *software* ha sido escogido por unos motivos concretos, hay algunas características fundamentales que se han intentado buscar en todos ellos, y que, por tanto, aparecerán recurrentemente a lo largo de este apartado. Estas son: que el *software* fuera **libre**, que fuera lo más **sencillo** posible mientras siguiera permitiendo el desarrollo completo de la aplicación y no comprometiera sus funcionalidades, y que fuera **popular**, dado que esto habitualmente resulta en que muchos sistemas permitan la integración de este componente, y da lugar a la existencia de numerosos recursos de apoyo que serán de gran utilidad.

3.2.1 Entorno de desarrollo: Eclipse

Eclipse⁷ es un entorno de desarrollo, que soporta algunos de los lenguajes de programación más usados, como *Java*, *C#*, *C++*, *Erlang*... Es además uno de los entornos de desarrollo más utilizados del mundo⁸, y es este el motivo por el que también cuenta con innumerables recursos creados por la comunidad que ayudarán a acelerar y mejorar el proceso de desarrollo en distintas etapas.

La elección del entorno de desarrollo es una decisión compleja que no debe tomarse a la ligera, ya que, en cierta medida, condicionará la velocidad a la que el programador puede trabajar, y los recursos a los que este tendrá acceso. Los principales motivos que han llevado a elegir *Eclipse* por encima de otros *IDEs* tales como *IntelliJ* (que fue el candidato que más se tuvo en cuenta después de este) o *NetBeans*, han sido: el hecho de ser un producto gratuito y de código abierto, la simplicidad de la interfaz, la abundancia de plugins disponibles para este entorno, la numerosa cantidad de recursos de ayuda existentes, tales como foros, dedicados a él, y las opciones de ayuda de código que incorpora.

⁶https://www.w3schools.com/css/css_rwd_mediaqueries.asp

⁷<https://www.eclipse.org/eclipseide/>

⁸<https://pypl.github.io/IDE.html>

Los *plugins* ayudarán a mejorar las capacidades del entorno, dotándolo de todas las funcionalidades que un desarrollador pueda necesitar. Otros entornos, como *IntelliJ*, también tienen disponible una cantidad de *plugins* razonable, pero mucho menor que la de *Eclipse*, y con menor soporte. Además, la extensa comunidad de programadores que posee este *IDE* proporcionará el soporte que se requiera en caso de que fuera necesario. Esto, de nuevo, es mucho más difícil en el caso de otros *IDEs*, debido al reducido tamaño de sus respectivas comunidades. Por último, *Eclipse* cuenta con la opción de autocompletar aquello que esté escribiendo el programador, característica con la que cuentan muchos entornos, pero que *Eclipse* realiza de forma inteligente. Lo que quiere decir esto es que el entorno ofrece las opciones de autocompletación priorizando aquellas más probables en el entorno en el que se está trabajando.

3.2.2 Lenguaje de programación: Java

*Java*⁹ es un lenguaje de programación de propósito general, concurrente, orientado a objetos, y que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su objetivo es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo, concepto conocido como *WORA* (*Write Once, Run Anywhere*), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra.

Java es un lenguaje que se ajusta perfectamente a nuestro proyecto, por una serie de razones.

Por una parte, es un lenguaje **orientado a objetos**, paradigma que permite crear aplicaciones modulares y desacopladas muy fácilmente, justo lo que necesitamos para implementar de forma apropiada el **patrón MVC**. Implementarlo en lenguajes *imperativos de Von-Neumann* o *funcionales* es innecesariamente complicado, y la mayoría de los lenguajes que utilizan dichos paradigmas no están pensados, en principio, para desarrollar aplicaciones web.

Por otra, es un lenguaje **robusto**. Esto nos permite programar minimizando el número de errores en tiempo de ejecución, cuyas causas son normalmente más difíciles de averiguar que las de los errores en tiempo de compilación. Por este motivo se descartaron lenguajes como *Python*. Este argumento es, hasta cierto punto, subjetivo, ya que hay programadores que valoran la **flexibilidad** por encima de la **robustez**.

Otro aspecto importante es que *Java* es **multithread**. En otros lenguajes, es necesario llamar a funciones específicas del sistema operativo para utilizar dicha característica; sin embargo, en *Java*, la programación *multithread* está integrada en el propio lenguaje, y el programador puede despreocuparse de ella.

⁹<https://www.java.com/es/>

Java, además, está preparado para crear aplicaciones que funcionen en un **entorno distribuido**, y provee numerosas librerías de ayuda que aportan las funcionalidades necesarias para crear cualquier tipo de aplicación de este tipo. Este es también el motivo por el que implementa numerosos **mecanismos de seguridad** para proteger las aplicaciones.

Por último, *Java* es uno de los lenguajes más usados del mundo ¹⁰, y además es libre, lo cual se traduce en que existen numerosos recursos a nuestra disposición que podemos utilizar para mejorar tanto el proceso de desarrollo como el producto final.

3.2.3 Gestor de base de datos: MySQL

MySQL fue uno de los primeros **sistemas de gestión de bases de datos relacionales** en aparecer.

Dado que nuestra aplicación no precisa de ninguna funcionalidad especialmente compleja, elegir el sistema de gestión de bases de datos más popular del mundo (junto con *PostgreSQL*) parece, en principio, una buena idea, ya que seguramente cubra todas las funcionalidades que vamos a necesitar.

Sus bases de datos utilizan el **modelo relacional**, lo cual hace que diseñar el esquema de la BD sea sencillo. En otros *RDBMS*, como *MongoDB*, que, en este caso, almacena la información en formato *JSON*, la lógica de las queries es mucho más compleja, aportando eso a cambio una serie de ventajas (como el poder almacenar cualquier tipo de dato, o realizar búsquedas muy específicas con una sola instrucción, ahorrando así tiempo con respecto a una BD que utilice el modelo relacional), que nosotros, sin embargo, no necesitamos. Por tanto, es preferible no aumentar la complejidad de la lógica del programa a cambio de obtener un tiempo de respuesta ligeramente superior.

Otra característica importante de *MySQL* es que **permite roll-backs** de las transacciones, algo que será de gran ayuda, especialmente al hacer las pruebas de integración del software, en donde en cada *test* se modifica la información de la base de datos, y nos interesa devolverla a su estado original al acabar dicho *test*.

Y evidentemente, y como en el resto del *software* utilizado, el hecho de que sea **freeware** es algo que se ha valorado positivamente a la hora de la elección.

La desventaja principal de *MySQL*, y el motivo por el que no se usa en aplicaciones que trabajan con *big-data*, es que **no es capaz de manejar grandes cantidades de datos de forma eficiente**. Sin embargo, dada la naturaleza de nuestro proyecto, y que la cantidad de información que va a ser almacenada en la BD es bastante pequeña, este no es un tema que nos deba preocupar. Por este motivo, la eficiencia en cuanto al manejo de datos no ha sido el foco principal de la elección.

¹⁰<http://pypl.github.io/PYPL.html>

3.2.4 Mapeador Objeto-Relacional: Hibernate

Hibernate es un *mapeador objeto-relacional*. El *mapeo objeto-relacional* es la persistencia automatizada y transparente de los objetos en una aplicación por medio de la transferencia a una base de datos, usando *metadata* que describa cómo se produce ese mapeo [9].

En primer lugar, deberíamos preguntarnos: ¿por qué usar un *mapeador objeto-relacional*, en lugar de lanzar *queries* directamente sobre la base de datos? esta pregunta tiene una respuesta muy sencilla: **abstracción**. El utilizar un *ORM* permite al programador abstraerse de detalles de implementación, y centrarse en programar la lógica de negocio de la aplicación, algo bastante más importante que el cómo se almacenen los datos con los que se trabaja. Además, usando un *mapeador*, sería muy sencillo cambiar el *RDBMS* que la aplicación utiliza por debajo, dado que solo habría que cambiar el *driver* del *ORM*.

Ahora que ya hemos discutido el porqué es necesario un *mapeador objeto-relacional*, podemos preguntarnos: ¿por qué *Hibernate* en particular?. La respuesta es simple: es la implementación por defecto de *JPA* ¹¹ en *Spring Data JPA*, lo cual hace que tenga una **integración perfecta** con *Spring, framework* utilizado en este proyecto y que se describirá más adelante. Esto ahorrará muchos problemas a la hora de configurar los diversos *frameworks* que interactúan en la aplicación. Además, es **software libre, escalable, fácil de aprender**, e implementa el concepto de **Lazy Loading**, que permite a la aplicación cargar solo los datos pertinentes a una consulta concreta, sin obtener también sus tablas asociadas, lo cual ralentizaría la ejecución del programa.

3.2.5 Pruebas de código: JUnit

El objetivo de la **plataforma JUnit** es permitir a los desarrolladores lanzar *tests* en la máquina virtual de *Java* ¹². Además de *JUnit*, para los tests unitarios también se utilizará **Mockito**, que permite crear objetos falsos o *mocks*, que son clave para poder *testear* funciones de forma aislada.

Dado que nuestro proyecto es muy sencillo, nuestras pruebas también deben de ser sencillas. *JUnit* es la herramienta básica de creación de *tests* por excelencia: **sencilla y libre**. Dadas las características de nuestro proyecto, no necesitamos más potencia que la que este *framework* nos proporciona. Además, *JUnit* es una herramienta **altamente eficiente**, permitiendo escribir potentes *tests* en pocas líneas de código.

Existen otras *suites* para realizar *tests*, como *Arquillian*, *JTest*, o *JWalk*, pero, dado el caso, **son completamente innecesarias**, dado que las funcionalidades que aportan a mayores no

¹¹https://en.wikipedia.org/wiki/Java_Persistence_API

¹²<https://junit.org/junit5/docs/current/user-guide/>

van a ser utilizadas, y un correcto uso de *JUnit* nos permitirá cubrir las partes importantes del código y obtener unos resultados fiables sin necesidad de otras librerías.

3.2.6 Framework de estilo de la interfaz: MaterializeCSS

MaterializeCSS es un *framework* moderno e intuitivo para desarrollo *front-end*, similar a *Bootstrap* y *Foundation*, y basado en el diseño *Material*, expuesto anteriormente en esta memoria [10].

Antes de responder a por qué se ha usado este *framework* CSS en concreto, debemos responder a la pregunta de "¿Por qué usar un *framework* para el desarrollo *front-end*?" a secas. La respuesta es sencilla.

Si no utilizamos un *framework*, tendremos que crear nosotros mismos nuestros propios estilos CSS. A pesar de que crear estilos no es demasiado complicado, es un proceso extremadamente laborioso, ya que deben abarcar todos los elementos utilizados en la página. Además, los elementos seguirían sin tener vida si únicamente proporcionamos los estilos pero no código para mostrar animaciones que hagan que la página sea visualmente atractiva. Por último, y quizás el problema más complejo de resolver, es que necesitaríamos dar soporte a distintos tipos de pantalla sin una librería que nos facilitara la tarea. Estos son solo unos pocos de los problemas que supone programar la vista sin utilizar herramientas apropiadas para ello. En caso de querer crear un diseño con un toque personal, es más sencillo modificar los estilos de una librería existente, y aprovechar el resto del código.

En cuanto a la elección del *framework* en particular, existen varias razones que han llevado a la conclusión de que *MaterializeCSS* era el más adecuado para este proyecto. La más importante es que, dado que queremos seguir las pautas del diseño *Material*, la librería debe proporcionarnos herramientas que nos faciliten ese aspecto. Este punto descarta muchas librerías, pensadas con otro tipo de finalidades, ajenas a la que se intenta lograr en este proyecto: **claridad y legibilidad**. Entre las restantes se ha escogido *MaterializeCSS* por ser una librería bastante ligera y sencilla de utilizar, además de por el claro diseño visual de las vistas que se pueden crear con ella (esto es puramente subjetivo, y puede haber desarrolladores que discrepen en este punto), factor que llevó a descartar el *framework* *MUI* CSS.

Por último, está orientada al diseño de páginas **responsive**, algo crucial en nuestra aplicación, como ya ha sido mencionado más veces a lo largo de esta memoria. *Bootstrap*, la librería CSS por excelencia, fue descartada porque, aunque es posible implementar un diseño que siga la guía de *Material* con ella, y es bastante similar a *MaterializeCSS*, es innecesariamente compleja. Se podría decir que *MaterializeCSS* es una versión de *Bootstrap* específicamente creada para trabajar con el diseño *Material*.

3.2.7 Funciones varias: Spring

Spring¹³ es un *framework* orientado al desarrollo de aplicaciones y contenedor de inversión de control, **gratuito** y **de código abierto**, y disponible para la plataforma *Java*. Su función principal es proporcionar un *contexto de ejecución* (también denominado *Spring Application Context*) que se encarga de crear y manejar los distintos componentes de una aplicación. A cada uno de los componentes individuales se le denomina **bean**. Al hecho de conectar unos *beans* a otros se le denomina **inyección de dependencias** [11]. Provee numerosos módulos, que facilitan la implementación de diversos aspectos del software.

Spring, en su conjunto, ha sido escogido por considerarse una librería **casi imprescindible para el desarrollo de aplicaciones web** en *Java*. En 2018, un 74.5% de los programadores *Java* utilizaron *Spring*¹⁴. Por su popularidad, posee una **gran comunidad de desarrolladores**, lo cual hace que existan cuantiosos recursos *online*, así como libros, que permiten extraer el máximo potencial al *framework*. Además, cubre numerosos aspectos de la implementación del software, lo que **evita los problemas de integración** que surgen al trabajar con librerías de distintos desarrolladores que se encargan de tareas relacionadas en un mismo proyecto.

En este proyecto, se utilizarán varios módulos de *Spring* para facilitar la implementación de diferentes aspectos del software. A continuación, se describirán brevemente los más importantes, y se explicará por qué se han escogido para el desarrollo de esta aplicación.

Spring Core

Implementa la **inyección de dependencias** mencionada anteriormente. Se ha escogido para el proyecto porque permite al programador abstraerse de las interdependencias existentes entre los objetos de la aplicación, así como que se trabaje solo contra interfaces, eligiendo la implementación concreta de las mismas en algún otro sitio del código, desacoplando así a mayor nivel los objetos.

Spring ORM

Permite la integración de *Spring* con varias librerías de **mapeo objeto-relacional**, entre las que se encuentra *Hibernate*. Se ha escogido utilizar este módulo porque la integración de ambos *frameworks* nos proporciona la ventaja de poder aprovechar las características de *Spring*, tales como la **inyección de dependencias**, sobre las entidades manejadas por el *framework ORM*.

¹³<https://spring.io/>

¹⁴<https://www.baeldung.com/java-in-2017>

Spring MVC

Aunque el nombre real de este módulo es *Spring Web MVC*, se le conoce coloquialmente como *Spring MVC*. Se ha decidido utilizar este módulo porque proporciona la arquitectura básica de un proyecto web, y se encarga de numerosas tareas, tales como la resolución del mapeo de *URLs* a los controladores correspondientes, o dar soporte a la subida de archivos, así como de proporcionar ayuda a la hora de implementar el **patrón MVC**. Es un módulo imprescindible para el desarrollo de una aplicación web con *Spring*.

Spring Security

Spring Security se encarga de cubrir un amplio abanico de **necesidades de seguridad** de una aplicación, tales como la autenticación, la autorización, o la seguridad *API* [11]. Dado que necesitamos implementar medidas de seguridad en nuestra aplicación, y este *framework* se integra perfectamente con el resto de módulos *Spring* de nuestra aplicación, *Spring Security* es la librería ideal para implementar este apartado del proyecto.

Spring Test

Proporciona las herramientas necesarias para realizar *tests* en un contexto de *Spring*. En este caso, la elección de este módulo surge de una necesidad: debemos hacer *tests* de integración en un entorno que funciona bajo un *Spring Context*, y, por tanto, **necesitamos** este módulo.

Metodología

LA metodología utilizada para el desarrollo de este proyecto ha sido el *Proceso Unificado de Desarrollo de Software*. A continuación, se explicará más en detalle en qué consiste exactamente.

4.1 Proceso Unificado

Un **proceso de desarrollo de software** es el conjunto de actividades necesarias para transformar los requisitos de usuario en un sistema de *software*. Sin embargo, el *Proceso Unificado* es más que un simple proceso; es un **marco de trabajo genérico** que puede especializarse para una gran variedad de sistemas de software, diferentes áreas de aplicación, distintos tipos de organizaciones, diferentes niveles de aptitud, y varios tamaños de software [12].

El proceso de software está **basado en componentes**, lo cual quiere decir que el sistema *software* en construcción está formado por componentes *software* conectados a través de interfaces.

El proceso unificado utiliza el *Lenguaje Unificado de Modelado* o *UML* para preparar los esquemas de un sistema software.

Los aspectos definatorios del *Proceso Unificado* son tres:

- **Dirigido por casos de uso:** Un caso de uso es un fragmento de funcionalidad del sistema que proporciona al **usuario** (entendiendo usuario no necesariamente como una persona, sino también como otro sistema fuera del sistema actual) un resultado. Los casos de uso representan los **requisitos funcionales**. Sin embargo, los casos de uso no son solo útiles para especificar los requisitos de un sistema. También guían su diseño, implementación, y prueba. Es decir: **guían el proceso de desarrollo**.
- **Centrado en la arquitectura:** La arquitectura es una vista del diseño completo, con las características más importantes resaltadas, dejando de lado los detalles. Para hallar

una arquitectura adecuada para el proyecto, los arquitectos deben centrarse en los casos de uso claves. La arquitectura y los casos de uso deben **evolucionar en paralelo**.

- **Iterativo e incremental:** El desarrollo, al nivel más básico, se divide en **iteraciones**, de modo que cada iteración supone una mejora incremental sobre la anterior. En cada iteración, los desarrolladores identifican los casos de uso relevantes, los analizan, crean un diseño utilizando la arquitectura seleccionada como guía, implementan el diseño mediante componentes, y verifican que los componentes satisfacen los requisitos.

El Proceso Unificado se repite a lo largo de una serie de **ciclos** que constituyen la vida de un sistema. Cada ciclo concluye con una versión del producto para los clientes. Cada ciclo se compone de cuatro fases:

- **Inicio:** en esta fase se define el negocio: facilidad de realizar el proyecto, se presenta un modelo, visión, metas, deseos del usuario, plazos, costos, y viabilidad.
- **Elaboración:** en esta fase, se obtiene la visión refinada del proyecto a realizar, la implementación iterativa del núcleo de la aplicación, la resolución de riesgos altos, unos nuevos requisitos, y se ajustan las estimaciones.
- **Construcción:** esta fase abarca la evolución del *software* hasta que se convierte en un producto listo, incluyendo requisitos mínimos. También se afinan los detalles menores, como los diferentes tipos de casos, o los riesgos no críticos.
- **Transición:** al llegar a esta fase, el producto debe estar preparado para ser probado, instalado, y utilizado por el cliente. Una vez finalizada esta fase, solo queda pensar en futuras actualizaciones para la aplicación.

Cada una de estas fases se divide en una serie de **iteraciones**, ya que estas ayudan a reducir la dificultad del proyecto, dividiéndolo en esfuerzos más pequeños. Una iteración genérica, a su vez, se divide en una serie de etapas predefinidas, que son: **requisitos, análisis, diseño, implementación, y pruebas**.

4.1.1 Lenguaje Unificado de Modelado

El **Lenguaje Unificado de Modelado**¹, más conocido como **UML**, es un lenguaje estándar cuyas funciones son especificar, visualizar, construir, y documentar los artefactos de *software* de un sistema. UML fue creado por la organización conocida como *Object Management Group (OMG)*. Ha llegado a ser una parte fundamental de la *Ingeniería de Software*, debido a su capacidad de representar mucha información de manera clara y concisa [13].

¹<https://www.uml.org/>

4.2 Metodología a utilizar

Siguiendo las pautas marcadas por el **Proceso Unificado**, desarrollaremos la aplicación en base a dicha metodología, realizando un único ciclo de vida (dado que solo va a haber una versión de la aplicación, que será la que se obtenga como consecuencia del trabajo requerido en este proyecto), con las fases habituales, cada una compuesta por las iteraciones pertinentes.

Dado el limitado espacio del que se dispone en esta memoria, no se han incluido todos los diagramas realizados con vistas a la construcción del *software*. Sin embargo, sí aparecen aquellos que se han considerado significativos de cara a la presentación del proyecto a una persona que no haya estado involucrada en su desarrollo, ya sea porque describen de forma muy simple una parte muy amplia del proyecto, o porque están relacionados con módulos o casos de uso especialmente interesantes.

4.3 Roles

En el desarrollo de este sistema de software, mis tutores, Óscar Fontenla Romero, y Francisco Javier Bellas Bouza, han desempeñado el rol del **cliente**, así como de los **usuarios** finales de la aplicación mientras dure su desarrollo.

Yo, por otra parte, he desempeñado el rol del **equipo de desarrollo** de la aplicación. A pesar de que en un proyecto que siga las pautas del *Proceso Unificado* el equipo de desarrollo está compuesto por trabajadores que desempeñan, a su vez, distintas funciones, como la de **arquitecto**, la de **programador**, o la de **analista**, en este caso yo representaré todos estos papeles.

Capítulo 5

Desarrollo

COMO se ha mencionado en el capítulo anterior, la metodología que se ha decidido seguir para la realización del trabajo es la propuesta por el *Proceso Unificado*. En este capítulo, se explicará en detalle la implementación concreta que se ha hecho de dicha metodología en el proyecto, mediante la exposición de las fases por las que ha pasado, y sus correspondientes iteraciones.

5.1 Fase de Inicio

La primera fase del proyecto constará de **una única iteración**, dado que es lo habitual en la *fase de inicio*, y, además, el proyecto no alcanza una complejidad suficiente como para que merezca la pena plantear más iteraciones. Sin embargo, esta no será una iteración genérica, dado que las dos últimas etapas (*implementación* y *pruebas*) no existirán. Esto es porque, por las características del proyecto, **no se implementará ningún prototipo** durante esta fase, debido principalmente a que el proyecto es suficientemente sencillo y arquetípico como para poder garantizar al cliente que el proyecto es viable.

5.2 Iteración 1

Nuestro objetivo en esta iteración consiste en **garantizar la viabilidad del proyecto**, lo cual evitará invertir esfuerzos en una tarea sin futuro, y en **esbozar su arquitectura**, para tener una referencia en las siguientes fases de como estructurar el conjunto.

5.2.1 Requisitos

Lo primero que debemos hacer es **crear una lista de requisitos** en base a lo presentado por el cliente. Para esto, nos basaremos en la información expuesta por el enunciado del *TFG*, así como en una charla informal mantenida con el cliente días antes de comenzar propiamente

el desarrollo del proyecto. Esta es la etapa más importante de esta iteración en particular. Dada la temprana etapa del desarrollo en la que se encuentra el producto, los requisitos son bastante abstractos, y simplemente los utilizaremos para hacernos una idea de qué es lo que quiere el cliente. A continuación, procedemos a presentar aquellos **requisitos que consideramos clave**.

Requisitos funcionales:

- El sistema debe permitir **almacenar** y **manipular** ejercicios de programación con toda su información asociada, así como exámenes, prácticas, y prácticas evaluables.
- El sistema debe permitir **buscar los ejercicios** en base a ciertos criterios.
- El sistema debe permitir la **descarga de los enunciados** de los ejercicios en formato *LaTex*.

Requisitos no funcionales:

- El sistema debe ser rápido.
- El sistema debe ser sencillo.
- El sistema debe estar disponible para el mayor número de usuarios posible.

En la [Figura 5.1](#) se presenta una primera versión del **modelo de casos de uso** del proyecto, en base a los requisitos funcionales recogidos en esta primera iteración, que son únicamente aquellos que se han considerado esenciales a la hora de esbozar la arquitectura del mismo.

Riesgos

Es importante tener en cuenta cuáles son los riesgos que podrían retrasar o incluso poner en peligro el desarrollo del proyecto. En la [Tabla 5.1](#) se muestran los riesgos que se han detectado inicialmente, aunque esta tabla es susceptible de ser actualizada y cambiar conforme avance el proyecto y se obtenga nueva información.

Código	Descripción	Probabilidad	Impacto	Exposición
R1	<i>Skills</i> insuficientes	Alta	Medio	Alta
R2	Diseño inefectivo	Media	Alto	Media
R3	Nuevos requisitos	Alta	Medio	Alta
R4	Mala estimación en la planificación	Alta	Bajo	Muy alto

Tabla 5.1: Riesgos identificados al comienzo del proyecto

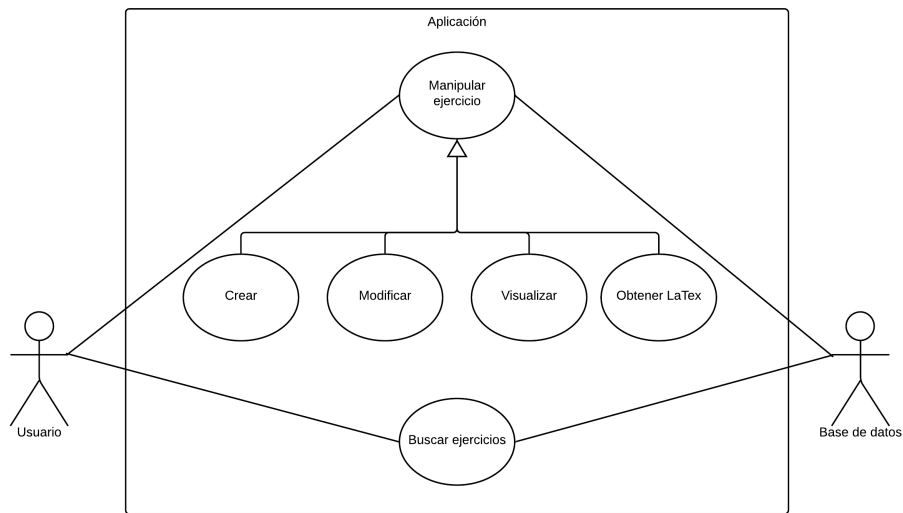


Figura 5.1: Modelo de casos de uso (iteración 1)

Prevención de riesgos

Los riesgos con un grado elevado de exposición pueden retrasar notablemente el proyecto, motivo por el que deben ser analizados con el fin de **prevenirlos**.

- **R1 - Skills insuficientes:** Dado que el equipo de desarrollo nunca ha realizado un proyecto de este tipo, al menos de estas dimensiones, la falta de conocimientos sobre las herramientas utilizadas podría resultar en un retraso de la consecución de los hitos establecidos, o en un producto final difícilmente mantenible. Para mitigar este riesgo, el equipo de desarrollo ha pasado tanto tiempo **familiarizándose con las herramientas de trabajo** como le ha sido posible. Además, en el caso de que, aún así, este riesgo se llegara a manifestar, las iteraciones se planificarán con **suficiente margen** de tiempo como para poder alcanzar las metas esperadas igualmente.
- **R3 - Nuevos requisitos:** En un proyecto "real", lo más probable es que el cliente quiera incluir nuevos requisitos en el proyecto a mitad de su desarrollo. Sin embargo, teniendo en cuenta la simpleza de la aplicación que se va a desarrollar, lo más probable es que estos requisitos sean **aditivos** y que no necesiten de un cambio en la estructura de la aplicación. Por tanto, la medida de prevención que se aplicará será **utilizar una arquitectura suficientemente genérica** como para poder añadir nuevas funcionalidades sin necesidad de rehacer módulos existentes.
- **R4 - Mala estimación en la planificación:** Si la planificación no es la adecuada, el desarrollo del proyecto se verá afectado y puede que se produzcan retrasos en la conse-

cución de los hitos establecidos. Para evitar esto, y dado que disponemos de una ventana de tiempo relativamente amplia para realizar el proyecto, todas las iteraciones se han planificado con un **margen de tiempo razonable** que permita completarlas en caso de que así fuera necesario.

Seguimiento de riesgos

A aquellos riesgos que no se consideran críticos por el momento, se les realizará un seguimiento para controlarlos y así asegurarse de que no se vuelven problemáticos más adelante.

- **R2 - Diseño inefectivo:** Por la falta de experiencia del equipo de desarrollo en el campo del desarrollo de aplicaciones web, es posible que las decisiones de diseño que se tomen no sean las óptimas. Por tanto, se realizará un seguimiento de la estructura del *software*, a partir de los *diagramas UML* que se desarrollen en cada una de las iteraciones, con el fin de garantizar que la aplicación está bien modulada, y que se adhiere a los patrones y arquetipos conocidos de la programación *orientada a objetos* en caso de ser necesario.

5.2.2 Análisis

A partir de los requisitos obtenidos en la etapa anterior, se ha tratado de modelar, todavía desde un punto de vista bastante abstracto, pero con un lenguaje más técnico, y pensando ya en orientar la salida de este paso a que sea utilizable por la etapa de *diseño*, la arquitectura de la aplicación. Para mostrar, de forma resumida, el trabajo realizado en esta etapa, se han adjuntado a la memoria un par de diagramas de secuencia, concretamente aquellos que se han considerado más ilustrativos de cara a un observador externo del proyecto. Podemos verlos en la [Figura 5.2](#) y en la [Figura 5.3](#).

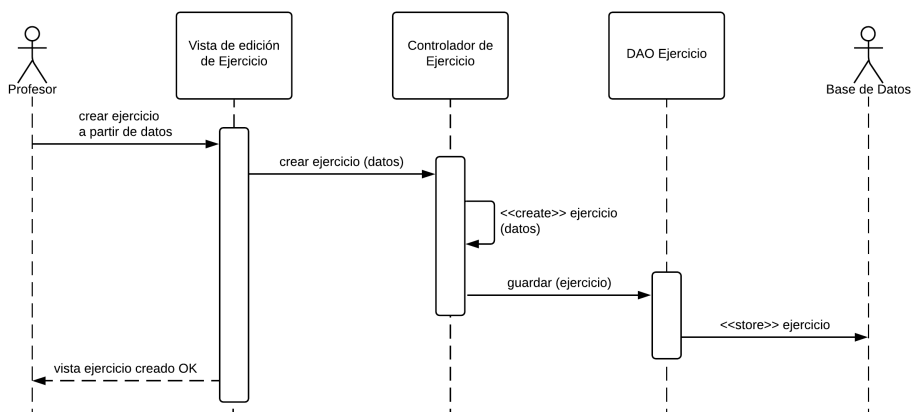


Figura 5.2: Diagrama de secuencia del caso de uso *Crear Ejercicio* (iteración 1)

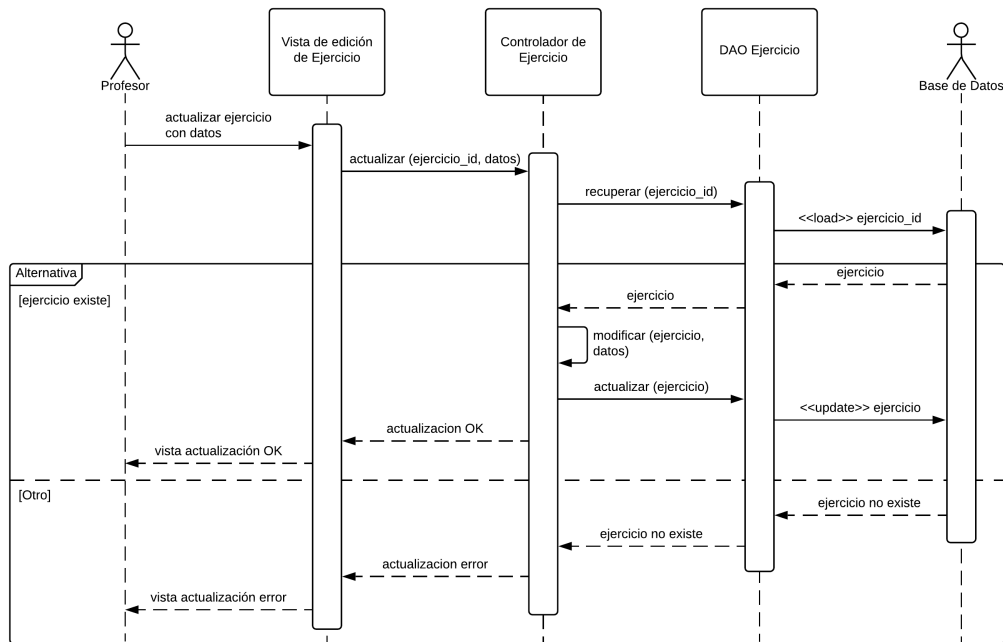


Figura 5.3: Diagrama de secuencia del caso de uso *Actualizar Ejercicio* (iteración 1)

5.2.3 Diseño

Dados los requisitos de usuario, y ahora que hemos analizado a un nivel muy superficial cómo tendría que funcionar el paso de mensajes para proporcionar las funcionalidades que se nos piden, podemos afirmar que la **arquitectura MVC** es adecuada para construir la aplicación. Por tanto, este patrón será usado como base, y todo el posterior proceso de diseño e implementación que se realizará a lo largo del desarrollo se apoyará en él.

Dado que no vamos a hacer una implementación de un prototipo en esta fase, y que estamos en una etapa tan temprana del desarrollo que seguramente los requisitos obtenidos todavía sean muy rudimentarios y poco específicos, no merece la pena esbozar la arquitectura más allá del pequeño párrafo anterior, por lo que no se ha realizado ningún diagrama de paquetes o clases para esta iteración.

5.2.4 Revisión de la iteración

En esta primera iteración, se han establecido los requisitos esenciales, se han modelado desde distintos puntos de vista, y se ha conseguido hallar una arquitectura que, en principio, podrá sostener el proyecto cuando se amplíe. Además, se han identificado los riesgos más significativos, y se han creado planes de prevención para ellos. Por estas razones, que se le han presentado al cliente, hemos concluido que el proyecto es **viable**, y, sumado a la planificación,

que se puede realizar **cumpliendo los *deadlines* propuestos.**

5.3 Fase de Elaboración

Es la segunda fase en un proyecto que se guía por las pautas del *Proceso Unificado*. Sus principales objetivos son: **acabar de recopilar la mayor parte de los requisitos que aún queden pendientes, establecer una base de la arquitectura sólida, continuar la observación y control de los riesgos críticos, y completar los detalles del plan de proyecto.**

Esta fase consistirá, de nuevo, en **una sola iteración**, en la que se abarcarán todos los aspectos mencionados anteriormente.

5.4 Iteración 2

5.4.1 Requisitos

En esta iteración, deberíamos ser capaces de identificar sobre el 80% de los casos de uso, así como de los requisitos no funcionales, de la aplicación final. A continuación, se explican, de forma breve y conceptual, cuáles son y en que consisten estos requisitos.

Requisitos funcionales:

- El sistema contemplará **tres tipos de usuarios: alumnos, profesores, y administradores**. Cada uno de ellos tendrá una serie de privilegios, en cuánto a qué información tendrá disponible, y qué parte de esa información podrá modificar. Los usuarios deben poder *loggearse* y *desloggearse*, como es de esperar.
- El sistema debe permitir a los profesores **crear, visualizar, actualizar, y eliminar** ejercicios.
- Un ejercicio deberá poder ser **visible o no** para los alumnos en un momento dado, según decida el profesor.
- Aparte de la dificultad estipulada por el profesor, los alumnos deberán poder **votar su opinión en cuánto a la dificultad** de un ejercicio.
- El sistema debe permitir **listar los ejercicios, y filtrarlos** en base a ciertos criterios.
- Los profesores podrán crear **ideas de ejercicio**, en las que dispondrán de las opciones de manipulación habituales.

- El sistema debe permitir almacenar **exámenes, prácticas, y prácticas evaluables**. Estos deben poder ser **creados, visualizados, actualizados, y eliminados**. Estos deben, también, poder contener un número indeterminado e ilimitado de ejercicios.
- Los conjuntos deben poder ser **listados y filtrados**, dentro de su respectivo tipo.
- Los conjuntos deberán poder ser **visibles o no** para los alumnos en un momento dado, y podrán también estar **abiertos o no**, estado que indica si un conjunto es modificable.
- Los conjuntos podrán **ser descargados** en formato *LaTeX* o *PDF*.
- Además de estos conjuntos, el sistema también permitirá almacenar una **selección de ejercicios de usuario**, que actuará como un carrito de la compra. Las diferencias de este conjunto con el resto residen en que no será visible para ningún usuario que no sea el propietario, y que no tendrá propiedades editables, más allá de los ejercicios que contenga.
- Las **categorías, dificultades, titulaciones, y usuarios**, solo podrán ser creadas, editadas, y eliminadas, **por un administrador**.
- Las distintas listas que se muestran en la aplicación estarán **paginadas**, con vistas a mejorar el rendimiento del producto *software*, al no tener que cargar todos los elementos de una sola vez.

Requisitos no funcionales:

- El sistema debe ser **rápido**.
- El sistema debe ser **sencillo y de fácil manejo**.
- El sistema debe estar **disponible para el mayor número de usuarios** posible.
- El sistema debe ser **seguro**.

Además, también se han realizado los diagramas de casos de uso pertinentes. En la memoria se adjuntan los más significativos: el de la [Figura 5.4](#), que representa los casos de uso relativos a Ejercicio, y el de la [Figura 5.5](#), que representa los casos de uso de los conjuntos.

Riesgos

En cuanto a los **riesgos**, mantenemos los reflejados en la [Tabla 5.1](#), dado que todavía estamos expuestos a todos ellos. De momento, ninguno se ha manifestado en un grado que haga peligrar la planificación preestablecida, por lo que continuaremos el desarrollo del proyecto siguiendo el plan previsto.

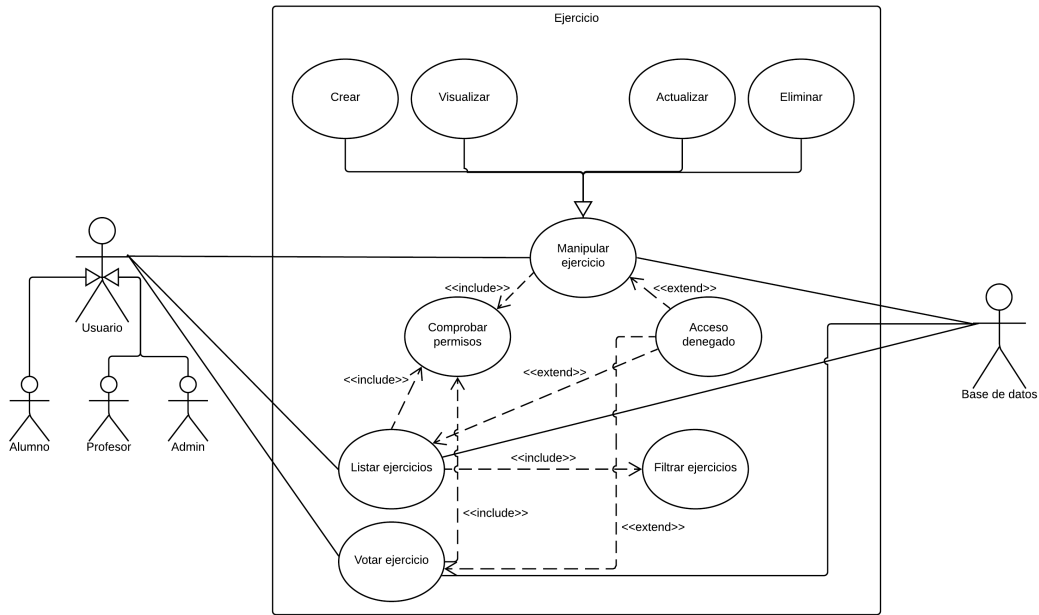


Figura 5.4: Casos de uso relativos a Ejercicio (iteración 2)

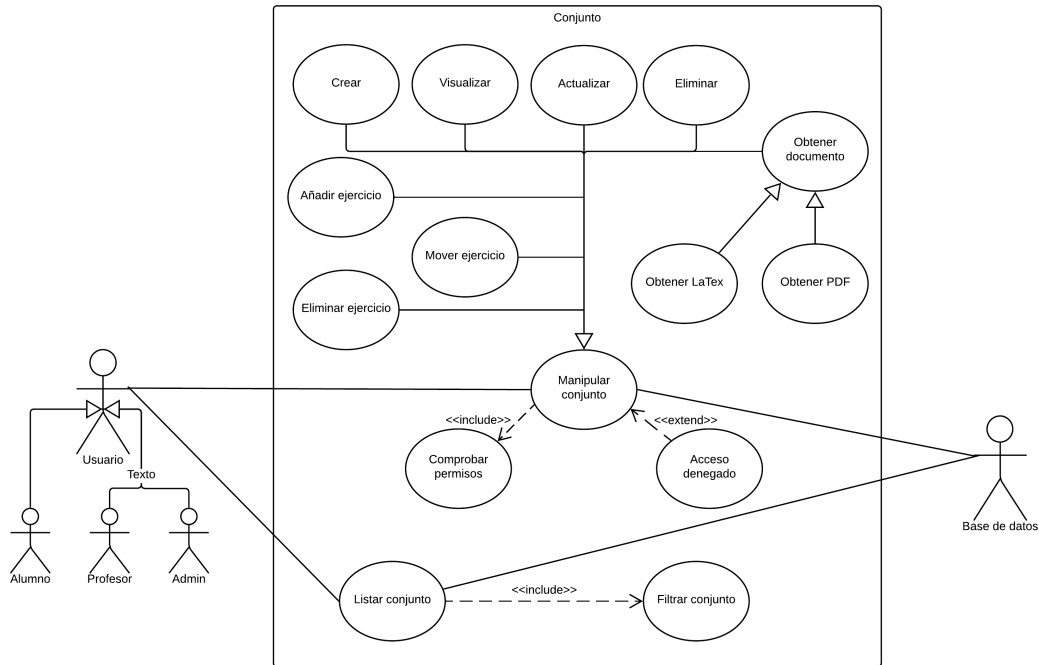


Figura 5.5: Casos de uso del relativos a Conjunto (iteración 2)

5.4.2 Análisis

En la etapa de análisis, se han refinado los requisitos obtenidos en el paso anterior para obtener una especificación más completa. Igual que en la fase anterior, se han realizado los diagramas de secuencia correspondientes, con el fin de ilustrar el comportamiento de los distintos componentes de cara al diseño. En la [Figura 5.6](#) podemos ver cómo la aplicación se encargará de generar el documento especificado a partir de la petición del usuario. Asimismo, en la [Figura 5.7](#), podemos observar la secuencia de acciones que realiza el subsistema de listado de ejercicios para obtener el conjunto final. Como podemos comprobar, a pesar del alto nivel de abstracción de los diagramas, la lógica de funcionamiento de la aplicación es más compleja que antes, y se tienen en cuenta más factores a la hora de estudiar cada caso de uso.

5.4.3 Diseño

Durante esta etapa, el arquitecto de *software* debe centrarse en los casos de uso más representativos del sistema a construir, para poder obtener una arquitectura sobre la que poder construir.

Lo primero que se ha diseñado ha sido la **base de datos**, dado que es necesaria para cualquier implementación que se desee hacer de la aplicación. En la [Figura 5.8](#) se puede ver este primer diseño, que aún podrá sufrir cambios en lo que queda del proceso de desarrollo si se diesen las circunstancias.

Se ha decidido **añadir una nueva capa** que actúe como intermediara entre la capa *modelo* y la capa *controlador*. El motivo de la inclusión de esta nueva capa es que, por los nuevos requisitos establecidos, **ha aparecido nueva lógica de negocio que era necesario implementar**, pero que no encaja ni en las entidades del *modelo*, por no ser lógica que esté relacionada con la comunicación entre la base de datos y la aplicación, ni en el *controlador*, por ser lógica propia de cada una de las entidades individuales de la aplicación, y no de cómo interactúan entre sí. El mejor ejemplo de esto son los *filtros*. Por como se han diseñado, cada entidad que deba poder ser filtrada por algún criterio, debe implementar una interfaz concreta para ese criterio. Los ejercicios, por ejemplo, podrán ser filtrados por varios criterios, por lo que implementarán numerosas interfaces. Como se puede intuir, esta nueva lógica no se corresponde con la que implementan ninguna de las dos capas mencionadas anteriormente. Con esta capa, **lograremos un mayor desacoplamiento** dentro de los módulos de la aplicación. Se ha denominado a esta capa ***rich entity***, dado que sus objetos encapsulan entidades de la capa *modelo*, y añaden nuevas funcionalidades.

Se adjunta, para ilustrar el trabajo realizado en esta etapa, el diagrama de la estructura de los subsistemas que compondrán la aplicación final. Se podría decir que este diagrama permite

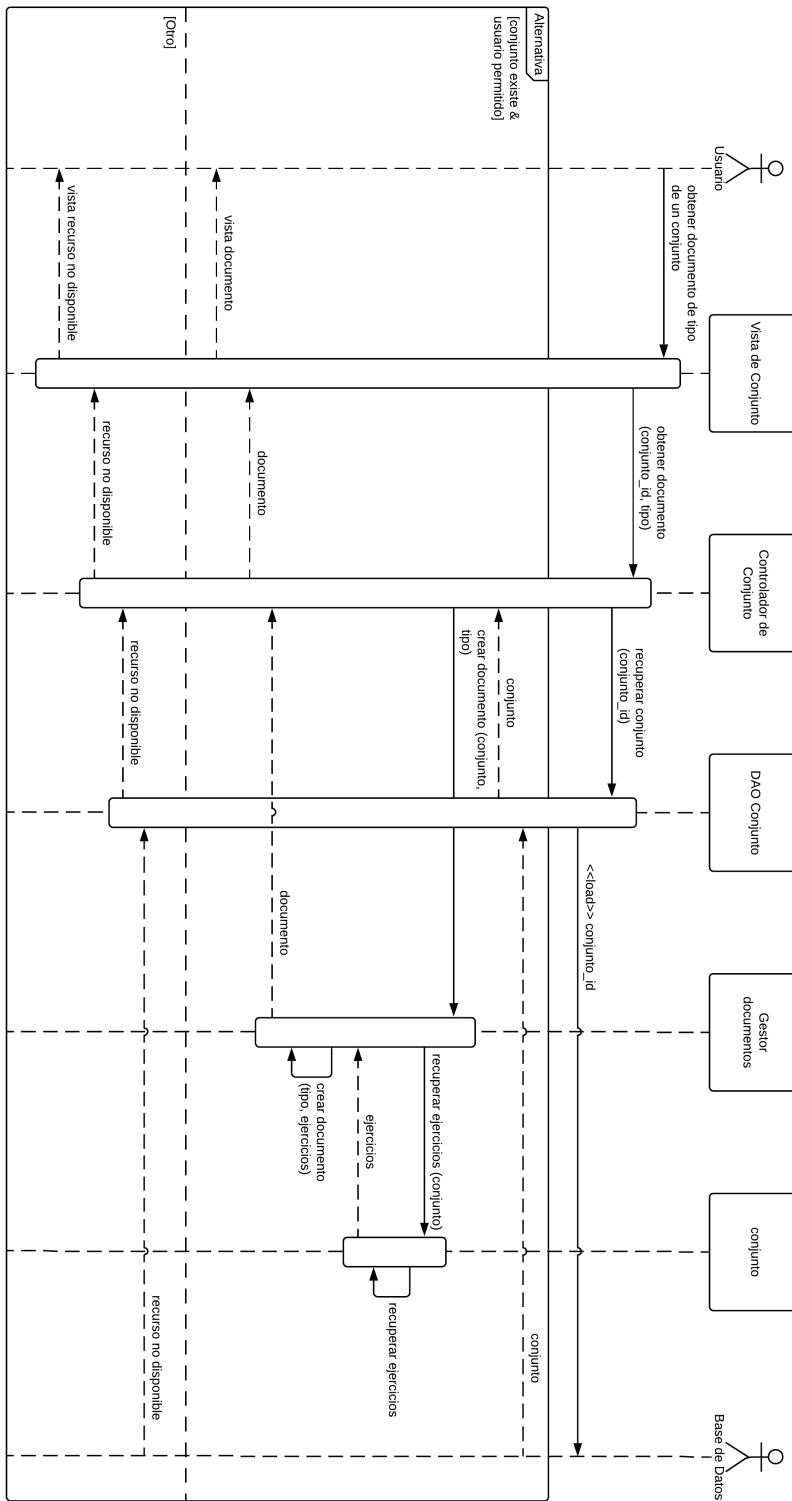


Figura 5.6: Diagrama de secuencia del subsistema de gestión de documentos (iteración 2)

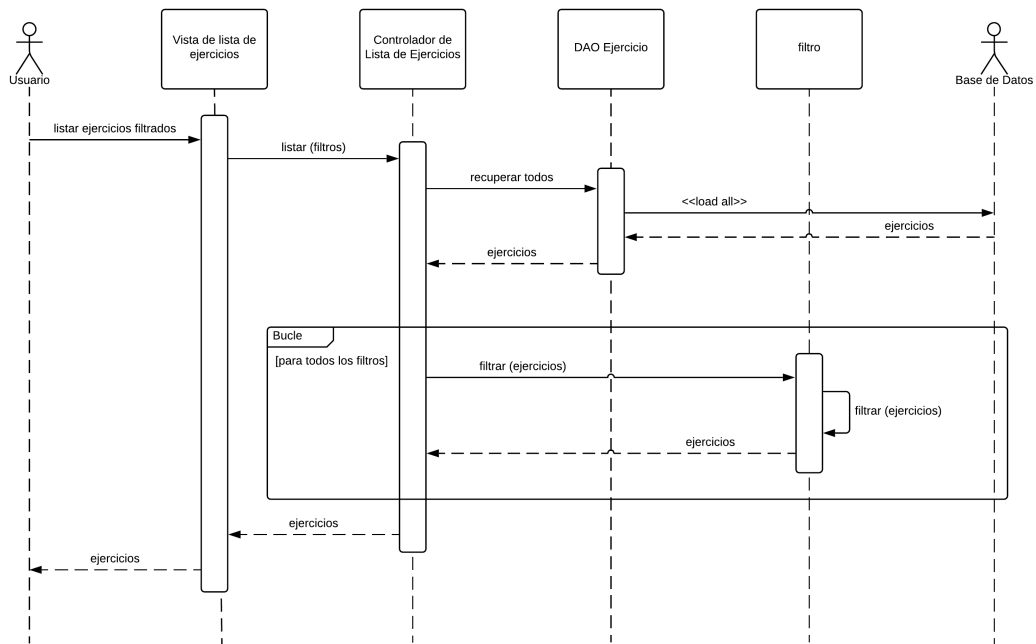


Figura 5.7: Diagrama de secuencia del subsistema de listado de ejercicios (iteración 2)

observar, a vista de pájaro, la arquitectura del sistema, de forma simplificada. En él, no se entra en detalles sobre cómo está compuesto cada subsistema, simplemente se muestra cuáles son las asociaciones que presentan entre sí. El diagrama aparece en la [Figura 5.9](#).

Además, también se han adjuntado diagramas que describen la estructura interna de algunos de los paquetes, aquellos que se han considerado importantes para entender la arquitectura de la aplicación en su conjunto. En la [Figura 5.10](#), podemos ver el paquete de la implementación del *modelo* en forma de diagrama de clases. En la [Figura 5.11](#), se puede observar la estructura del paquete *implementación* dentro del módulo *rich-entity*. Este paquete se puede comparar con el de la figura anteriormente mencionada para entender las diferencias, y las nuevas funcionalidades que añade esta capa. La más importante es que, a diferencia de la implementación de la capa *modelo*, las entidades de esta capa no tienen que tener una correspondencia directa con las tablas de la base de datos, lo que nos permite estructurar el paquete de forma más conveniente, utilizando clases abstractas para evitar repetir información. Ha de recordarse que en este diagrama no se han reflejado las relaciones de dependencia entre los objetos, por ser muy evidentes dados los nombres de las funciones, y hacer más innecesariamente complejo y difícil de estudiar el diagrama. En la [Figura 5.12](#), podemos ver la estructura interna del módulo *filtros*, de la capa *rich-entity*. Por último, en la [Figura 5.13](#), podemos ver como se ha estructurado la implementación de las clases controladoras.

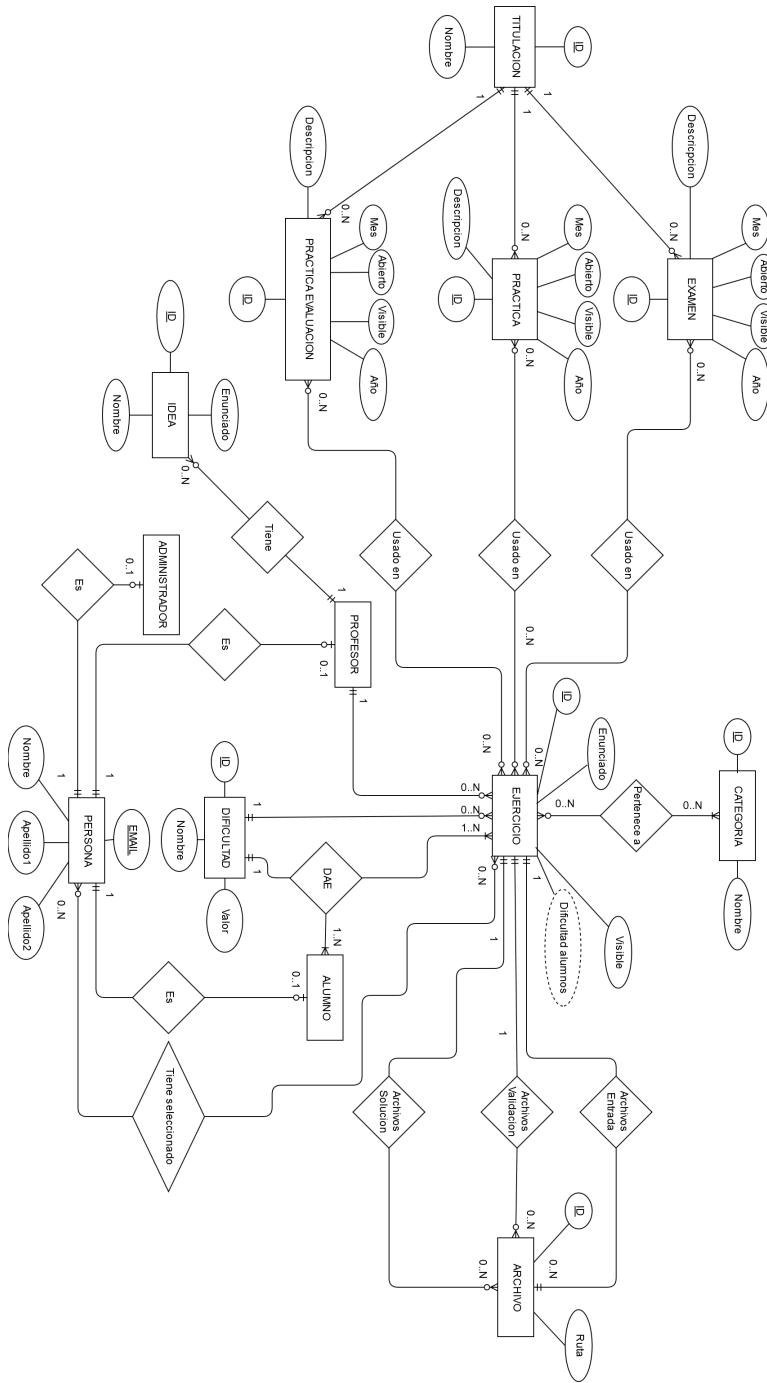


Figura 5.8: Diagrama Entidad-Relación de la base de datos (iteración 2)

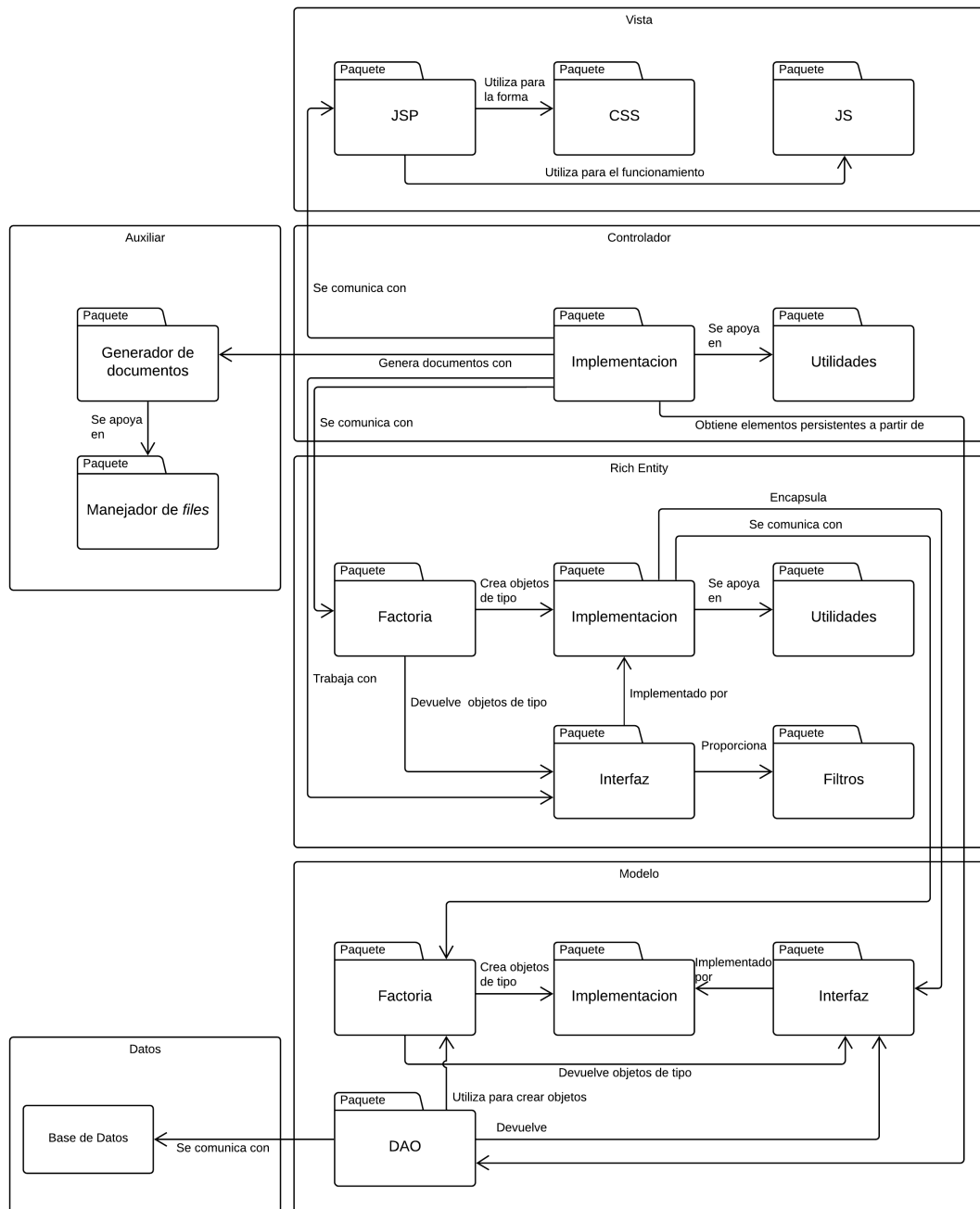


Figura 5.9: Diagrama de paquetes de la aplicación (iteración 2)

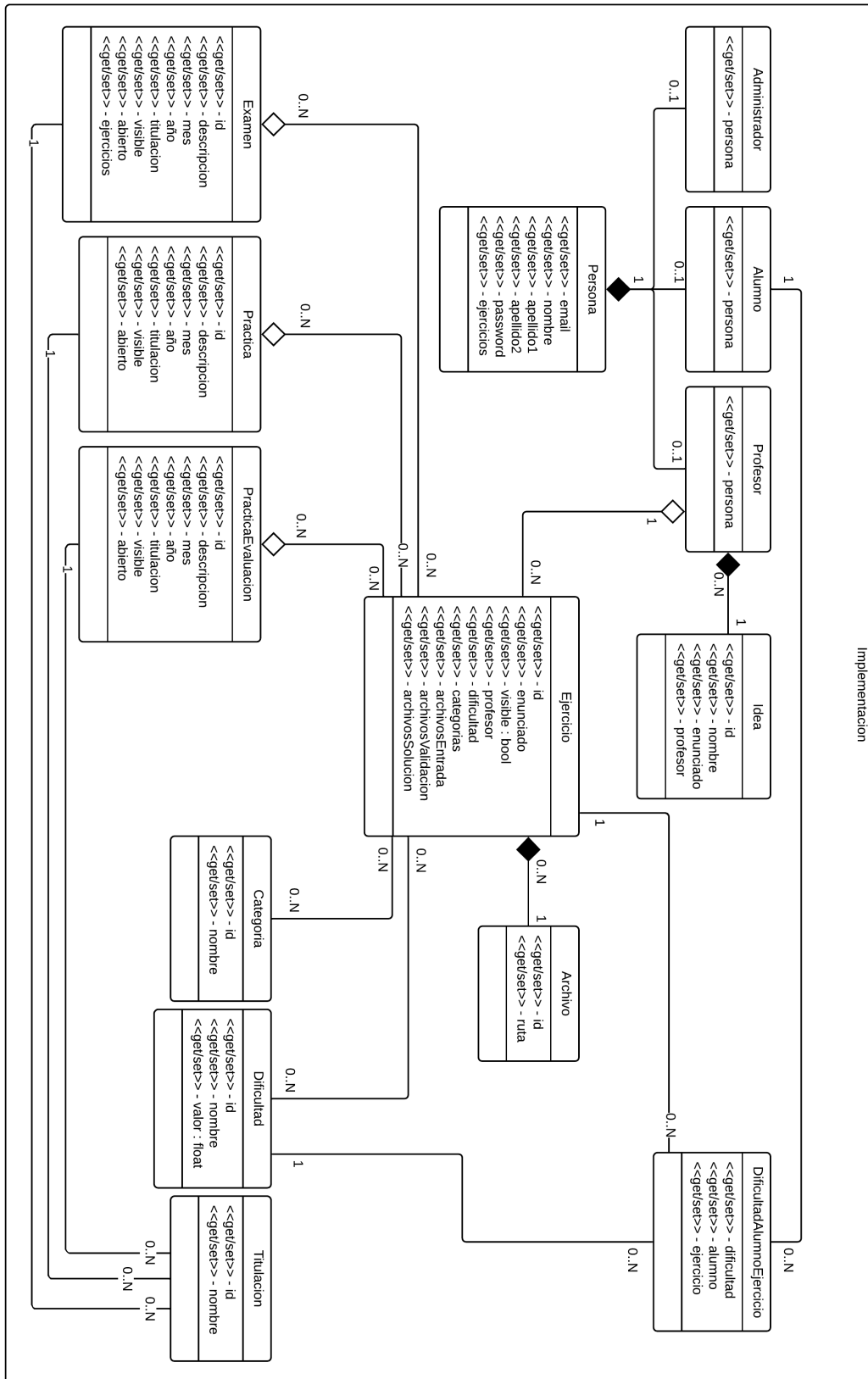


Figura 5.10: Diagrama de clases del módulo *implementación*, dentro de la capa *modelo* (iteración 2)

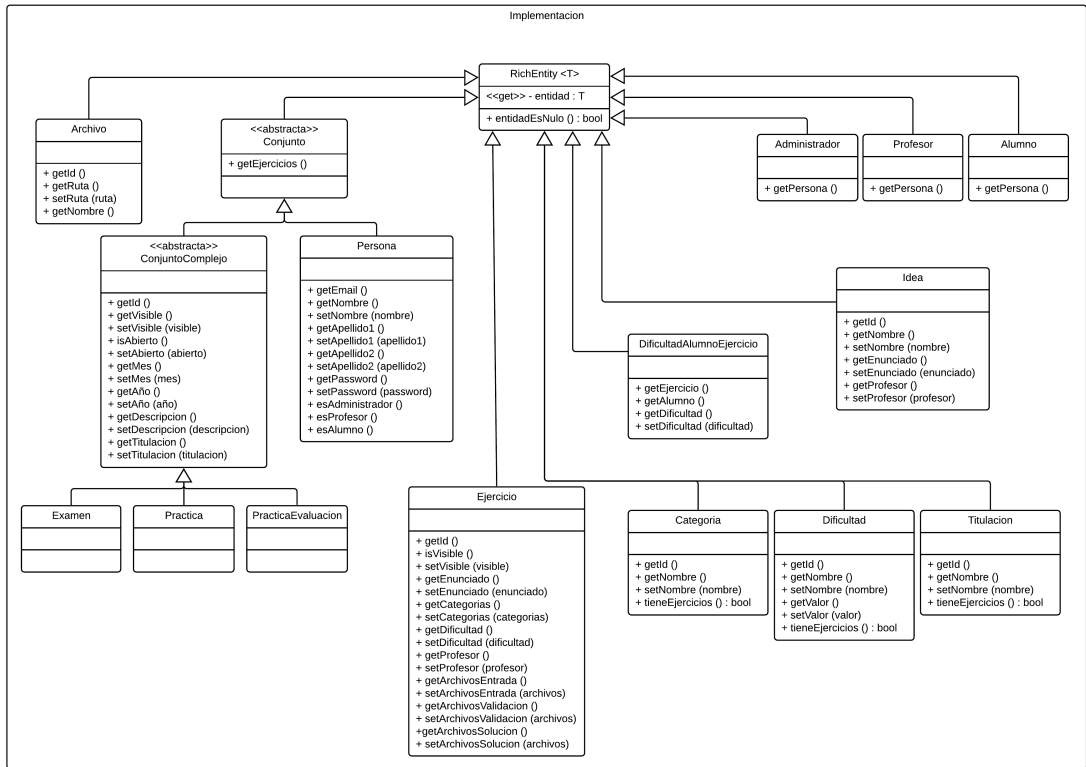


Figura 5.11: Diagrama de clases del módulo *implementación*, dentro de la capa *rich-entity* (ite-

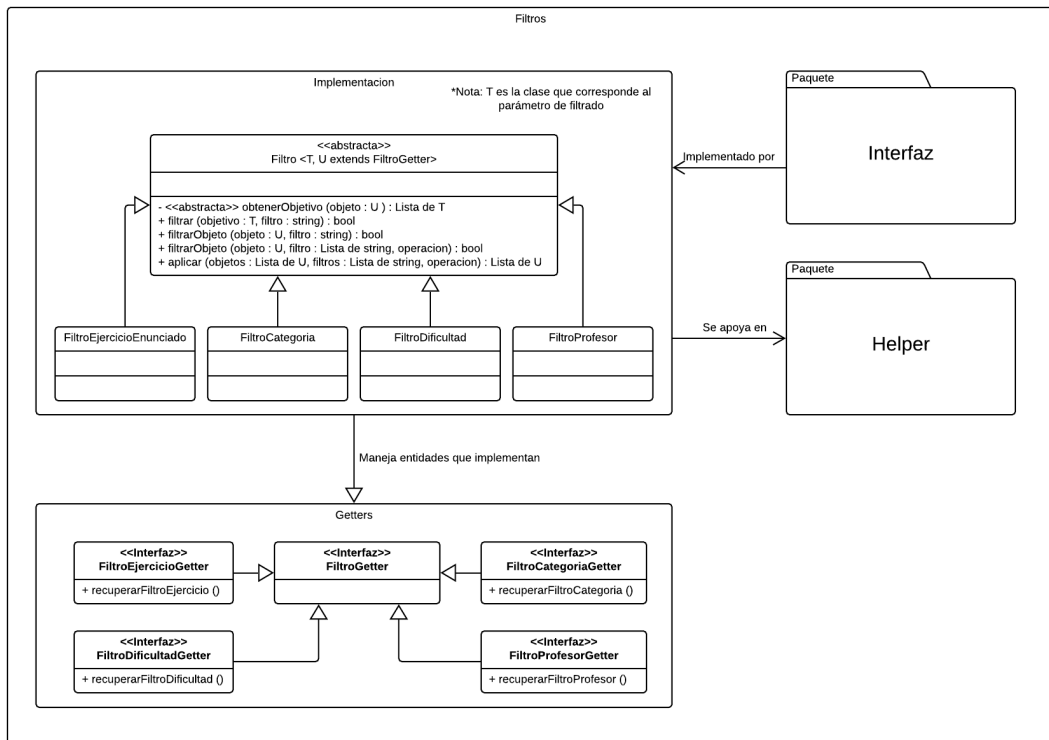


Figura 5.12: Diagrama de clases del módulo *filtros*, dentro de la capa *rich-entity* (iteración 2)

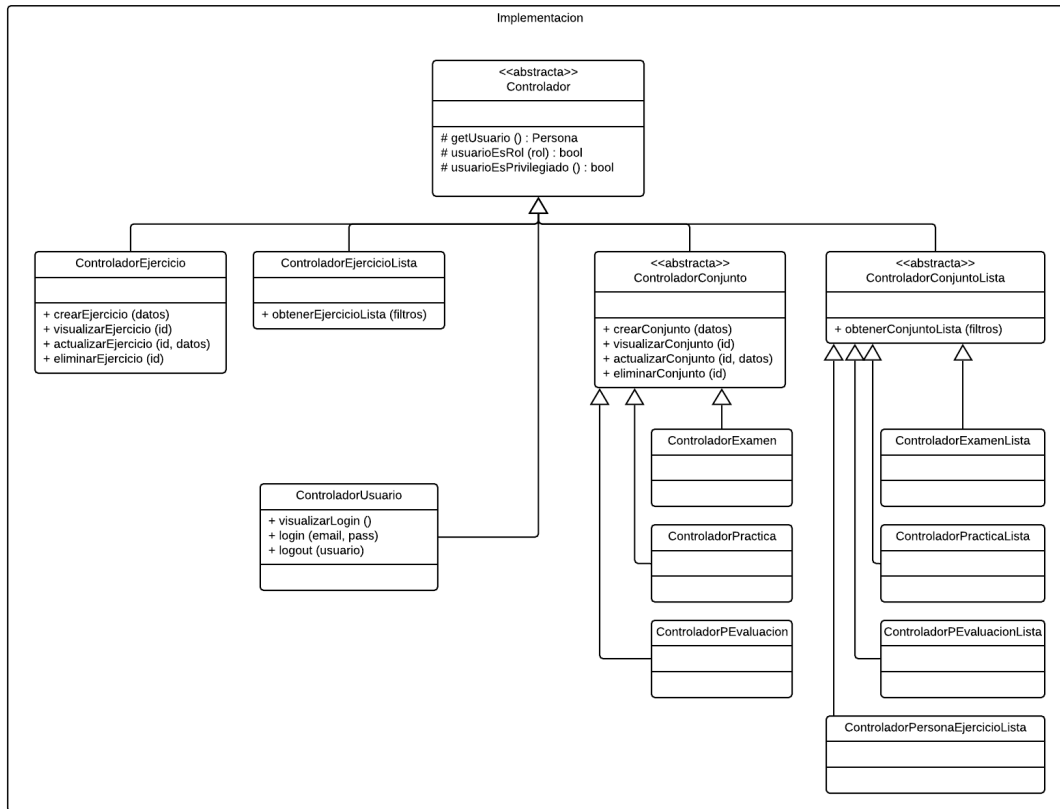


Figura 5.13: Diagrama de clases del módulo *implementación*, dentro de la capa *controlador* (iteración 2)

Mockups

Un *mockup* es un prototipo que implementa parcial o totalmente una funcionalidad del sistema a construir con el fin de orientar al cliente sobre el rumbo del desarrollo del mismo. En este caso concreto, la palabra *mockup* se refiere en particular al prototipo de una de las distintas vistas que conformarán la interfaz de usuario de la aplicación final.

Se han desarrollado algunos *mockups* muy primitivos de las interfaces más relevantes, que serán implementadas más adelante, para poder utilizar como referencia de cara al cliente, y así orientarlo en cuánto a la elección de los elementos que se les mostrarán a los usuarios en pantalla. Se han creado versiones de los *mockups* tanto para equipos sobremesa como para móviles, con vistas a la implementación que se le dará a la interfaz en la aplicación, que respetará las pautas del diseño *responsive*. Esto se puede ver en las figuras: [Figura 5.14](#), y [Figura 5.15](#). Ambas representan la vista de un ejercicio, pero en una de ellas, al ser diseñada específicamente para móviles, los elementos ocupan un porcentaje mayor de la pantalla, y la barra de navegación está oculta hasta que se pulsa el botón correspondiente.

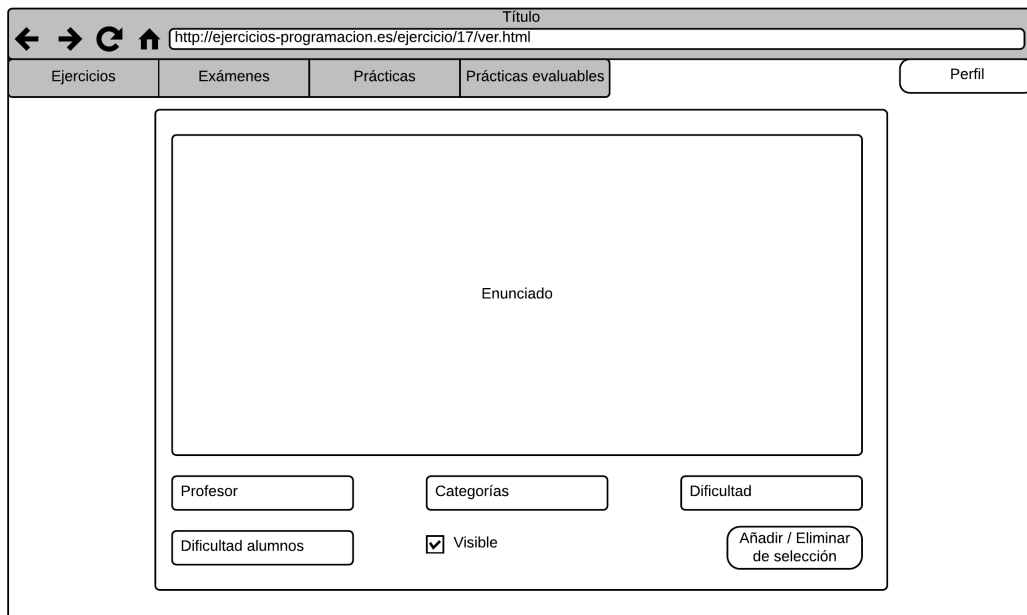


Figura 5.14: Mockup de la vista de un ejercicio en un PC (iteración 2)

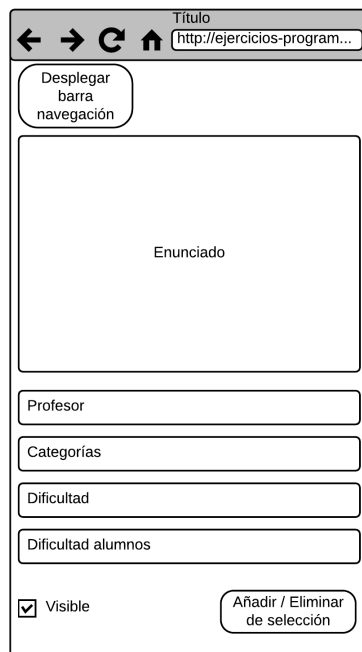


Figura 5.15: Mockup de la vista de un ejercicio en un móvil (iteración 2)

5.4.4 Implementación

La implementación de la aplicación que se debe hacer en esta fase es relativamente sencilla. Su objetivo principal es **establecer la línea base de la arquitectura**, algo que habitualmente requiere la consideración de menos del 10% de los casos de uso conocidos [12]. El prototipo construido, por tanto, es una pequeña muestra de la versión final de la aplicación, pero que contiene funcionalidades utilizables y fácilmente entendibles para los usuarios.

El prototipo consiste en una pequeña base de datos, que contiene solo los elementos esenciales para hacer funcionar la edición y el listado de ejercicios, en sus versiones más básicas, así como un sistema de gestión de usuarios que permite ejecutar las funcionalidades más básicas esperables de un sistema de este tipo.

5.4.5 Pruebas

Debido a que el prototipo diseñado suele ser muy simple, las pruebas todavía no son una parte esencial de esta fase normalmente. Sin embargo, sí es importante asegurarse de que las interacciones entre estos objetos elementales construidos funcionan como se espera, porque actuarán como base para la posterior implementación completa que se le dará al sistema.

Por este motivo, se han realizado **tests de base de datos**, para comprobar que la estructura de la *DB* es correcta, y suficiente para abarcar las entidades que pretendemos implementar en esta primera versión. También, como es de esperar, y como primer método de prueba, se han hecho **testeos ad-hoc**. Por último, se ha automatizado un pequeño conjunto de **pruebas de integración**, que no solo valdrán para comprobar que los componentes interactúan correctamente, sino que también sentarán la base de los *tests* automatizados, con vistas a ser ampliadas en la siguiente fase, y dar cobertura a todo el proyecto.

5.4.6 Revisión de la iteración

En esta iteración, se ha llevado a cabo el 80% del trabajo analítico y de diseño que implica el proceso de desarrollo. El prototipo resultante ha sido presentado al cliente, quien ha expresado la necesidad de implementar nuevas funcionalidades en el proyecto, que serán tomadas en consideración en la siguiente fase.

5.5 Fase de Construcción

El objetivo de esta fase es construir un producto software que **constituya la versión operativa inicial** del sistema creado a partir de los requisitos del cliente. Esta versión es conocida como **versión beta**.

Dado que, como veremos a continuación, han aparecido un número relativamente alto de nuevos requisitos para la fase en la que nos encontramos, nos hemos visto obligados a **replazar** esta fase. Realizaremos, por tanto, **tres iteraciones**, en vez de las dos preestablecidas. Las dos primeras estarán dedicadas a implementar todo aquello que no sean interfaces de usuario, y utilizarán una interfaz provisional para aquellas pruebas que requieran de ella. La tercera estará dedicada a implementar la vista final.

5.6 Iteración 3

El objetivo de esta iteración ha sido, centrándonos en los requisitos ya establecidos, y a partir del análisis y diseño ya realizado, implementar una aplicación funcional. Aunque para realizar este paso no partiremos del prototipo construido en la iteración anterior, sí lo usaremos como guía para facilitar el proceso.

5.6.1 Requisitos

Se mantienen los de la iteración anterior.

5.6.2 Análisis

Como parte de la revisión del análisis de la etapa anterior, se **han reevaluado los riesgos** existentes.

Riesgos

De los riesgos especificados en el [Tabla 5.1](#), el más significativo de cara al desarrollo de esta iteración es **R1 - Skills insuficientes**. Dado que esta fase está dedicada principalmente a la implementación del producto, la falta de experiencia con la tecnología utilizada debería ser nuestra mayor preocupación. Sin embargo, gracias a la planificación holgada que se ha hecho del proyecto, tendremos tiempo para solucionar los problemas que surjan por este motivo.

5.6.3 Diseño

Al igual que el paso anterior, tampoco reharemos el diseño en esta iteración.

5.6.4 Implementación

Se ha **implementado una versión funcional** de la arquitectura propuesta en la iteración anterior. Además, se ha implementado paralelamente una **interfaz de usuario provisional**, que servirá para que el programador pueda realizar pruebas a mano, además de como base para desarrollar la vista final en la iteración 5. Aparte de esto, no hay mucho más resaltable que

comentar de esta etapa, a pesar de su larga duración, puesto que la mayor parte del tiempo se ha estado ocupado con detalles de implementación que no son relevantes para esta memoria.

5.6.5 Pruebas

Se han automatizado las **pruebas de integración** que han sido consideradas pertinentes. Además, se han creado también **pruebas unitarias**. Ahora que el número de módulos es medianamente considerable, es importante garantizar no solo que funcionan bien en conjunto, sino que cada uno de ellos funciona bien por separado. El principal motivo de la importancia de estas pruebas en esta iteración es que, sabiendo que necesitaremos modificar numerosos componentes de la aplicación en la próxima iteración, también podemos imaginar que aparecerán nuevos errores en módulos que considerábamos estables. Si solo realizáramos pruebas de integración, encontrar el módulo origen del fallo en un determinado caso de uso sería una tarea ardua y compleja. De esta manera, sabremos qué módulo individual está causando problemas. Por último, se han realizado **pruebas *ad-hoc*** a partir de la interfaz provisional implementada.

5.6.6 Revisión de la iteración

Hemos llevado a cabo con éxito la implementación de los requisitos obtenidos en la fase de elaboración. Sin embargo, esta implementación deberá ser ampliada en la iteración siguiente, debido a los cambios en los requisitos. Con todo, gracias al sólido diseño obtenido en la iteración anterior, no debería ser demasiado complicado adaptar la aplicación a las nuevas necesidades de los clientes.

5.7 Iteración 4

En esta iteración, se ha llevado a cabo la implementación de los nuevos requisitos que ha pedido el usuario, tras la última charla con él, al acabar la iteración 2.

5.7.1 Requisitos

Los requisitos de la [etapa de recogida de requisitos de la iteración 2](#) se mantienen, por lo que no se volverán a listar aquí. Sin embargo, sí se expondrán los nuevos requisitos, así como los cambios que se tenga que hacer a los antiguos. Ha de decirse que estos nuevos requisitos han sido hallados, en parte, por las peticiones del cliente, y, por otro lado, por propuestas que el desarrollador ha formulado al cliente, dado el conocimiento que este tiene sobre su aplicación, y que el cliente ha aceptado. Estos requisitos son lo suficientemente importantes como para, a pesar de obligar a modificar ligeramente la estructuración de la aplicación, introducirlas en

esta iteración, dado que, además, no contamos con el tiempo suficiente como para desarrollar otro ciclo de vida. Los requisitos son:

- Los ejercicios, a partir de ahora, contarán con un **título**, que usaremos para identificarlos cuando sean listados.
- Un **sistema de correo** que se utilice para notificar al usuario cuando este sea creado, editado, o eliminado.
- Los usuarios podrán ser **creados a partir de una lista** en algún formato, para así ahorrar el trabajo a los administradores.
- La generación de una contraseña aleatoria, siguiendo ciertas restricciones, que se remitirá por email a los usuarios, tal como sean creados, de forma que el administrador no tenga que inventarse una contraseña por cada usuario que se cree.
- Un **sistema de restablecimiento de contraseña**.
- La implementación del **protocolo TLS** en la aplicación para garantizar la confidencialidad de los datos.
- Un sistema de *logeo* de información, que permita a los administradores monitorizar el sistema.
- Pedir confirmación para todas aquellas acciones que puedan realizar los usuarios que sean delicadas, tal como borrar un ejercicio.
- Los ejercicios deben poder **cambiarse de posición** dentro de un conjunto, por todos aquellos con privilegios para editar dicho conjunto.

Dado que estos casos de uso no cambian radicalmente la estructura de los casos de uso de la aplicación, no se incluirán los diagramas de casos de uso actualizados en esta sección.

5.7.2 Análisis

Para llevar a cabo el análisis de los requisitos detallados en el paso anterior, nos hemos valido de las herramientas habituales en este proceso de desarrollo. Se han realizado diagramas de secuencia de aquellos elementos que se han considerado relevantes, como el presentado en la figura [Figura 5.16](#), que representa la secuencia de acciones que realiza el *software* a la hora de crear un nuevo usuario, y que posiblemente es el más interesante de estos diagramas en lo referente a la memoria.

Además, se ha decidido utilizar el formato **JSON** como lenguaje para los ficheros que recibirá la aplicación para crear las listas de usuarios. El motivo principal de esta elección

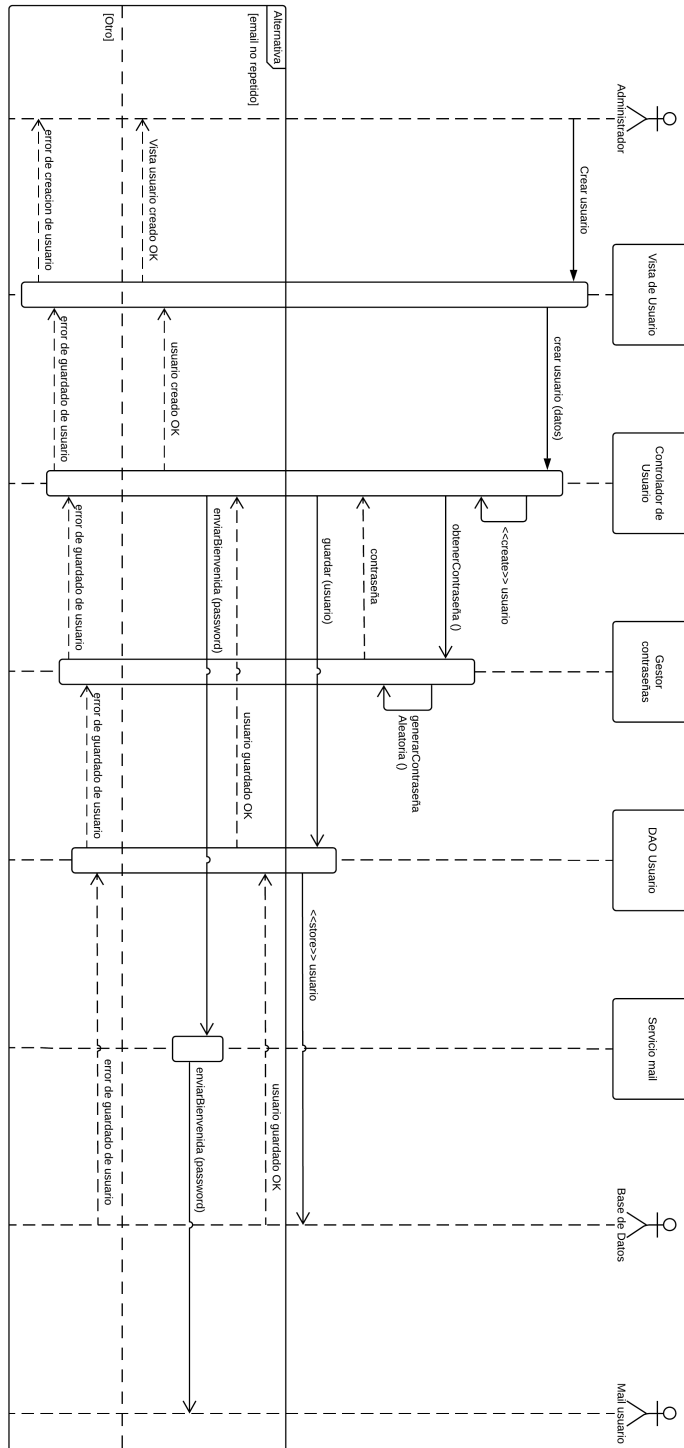


Figura 5.16: Diagrama de secuencia de la creación de un nuevo usuario (iteración 4)

es que es un lenguaje ampliamente conocido y con un enorme soporte, y que, aunque en teoría permite enviar cualquier tipo de información, está pensado específicamente para enviar *objetos Java*, lo cual es perfecto en este caso, dado que los usuarios que pretendemos añadir tienen una correspondencia directa con los *objetos Java* que crearemos.

Riesgos

Antes de proceder con la etapa de implementación de esta fase, se han revisado de nuevo los riesgos mostrados en el [Tabla 5.1](#). Al igual que en la iteración anterior, el más preocupante de estos riesgos es **R1 - Skills insuficientes**. La iteración anterior ya se retrasó por culpa de esto, y es posible que a esta iteración le ocurra lo mismo. Sin embargo, el impacto debería ser menor, puesto que, aunque todavía no se ha aprendido a manejar toda la tecnología con la que se construye el proyecto a un nivel adecuado, sí tenemos ya ciertos conocimientos sobre ella, lo cual debería minimizar el tiempo que se pierde por este motivo.

5.7.3 Diseño

Gracias a que la estructuración de la arquitectura de la fase anterior es aceptable, nos basaremos en ella y simplemente añadiremos y modificaremos componentes según sea necesario.

En primer lugar, debemos modificar la base de datos para adaptarla a los nuevos requisitos que han aparecido. Podemos ver su nueva estructura en el diagrama *entidad-relación* que aparece en la [Figura 5.17](#).

El cambio más llamativo que se puede apreciar es que ahora la relación entre ejercicio y los conjuntos tiene un atributo llamado *posición*. Esto nos obligará a cambiar la estructura de la capa *modelo* (y de la capa *rich entity* si queremos mantener la consistencia en el paralelismo que mantienen las dos capas), para añadir nuevas clases que representarán la relación entre un ejercicio y un conjunto, con el atributo anteriormente mencionado a mayores.

Sin embargo, como podemos ver en la [Figura 5.18](#), la estructura de paquetes del proyecto no cambia demasiado, y simplemente ha sido necesario añadir un par de subsistemas auxiliares. También se ha añadido un paquete **DAO** a la capa *rich entity*. El objetivo de este paquete es lograr el desacoplamiento absoluto de la capa *controlador* con respecto a la capa *modelo*. En la [Figura 5.9](#), que representa la estructura de paquetes propuesta en la iteración 2 del proyecto, podemos ver que la implementación de la capa *controlador* tiene que acceder a los DAOs de la capa *modelo* para comunicarse con la base de datos, y luego encapsular el objeto recibido en una clase *rich entity*. Esto, obviamente, no es deseable, puesto que obligamos al controlador a trabajar con elementos básicos y convertirlos a sus pertinentes clases cápsula en cualquier función en la que se desee utilizar entidades de la base de datos. Con este nuevo paquete de clases **DAO** en la capa *rich entity*, cuyas clases encapsulan a las de la capa *modelo*, y devuelven

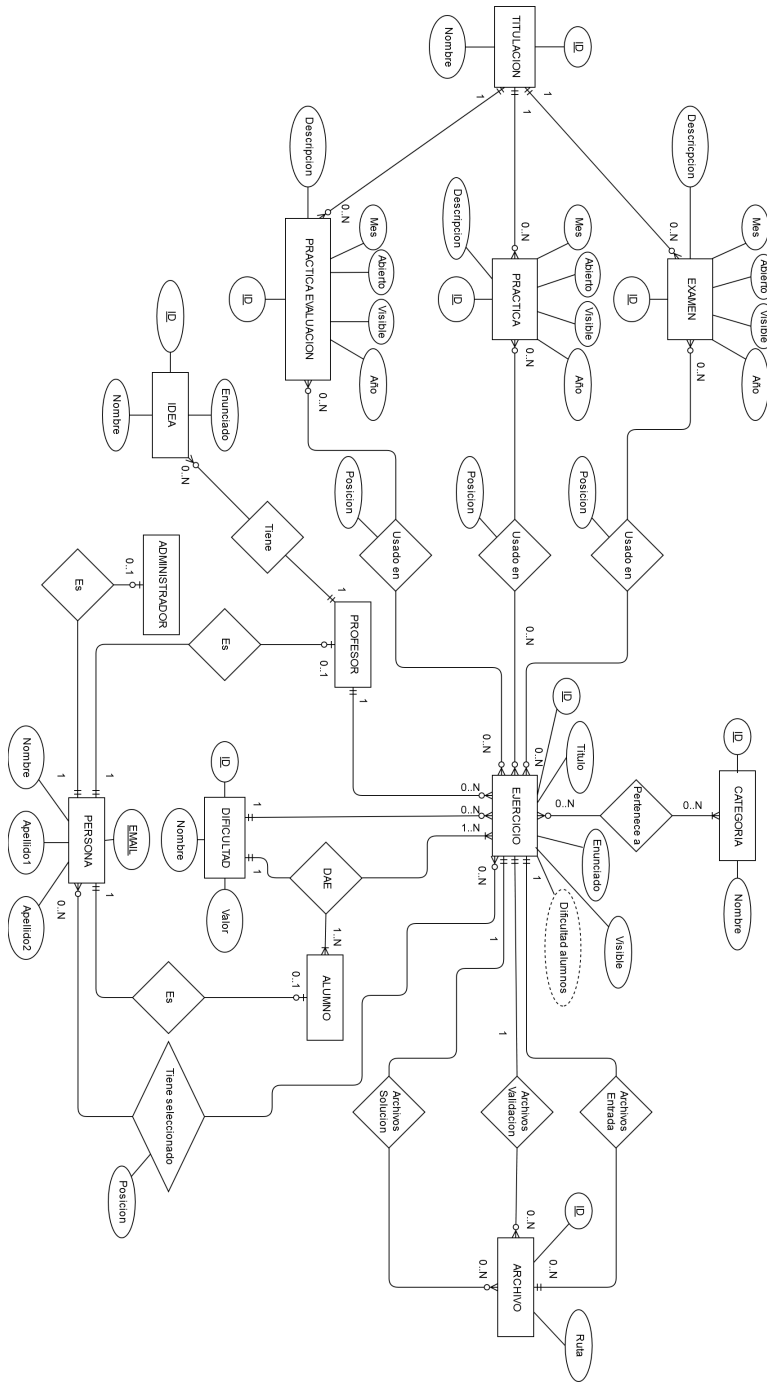


Figura 5.17: Diagrama Entidad-Relación de la base de datos (iteración 4)

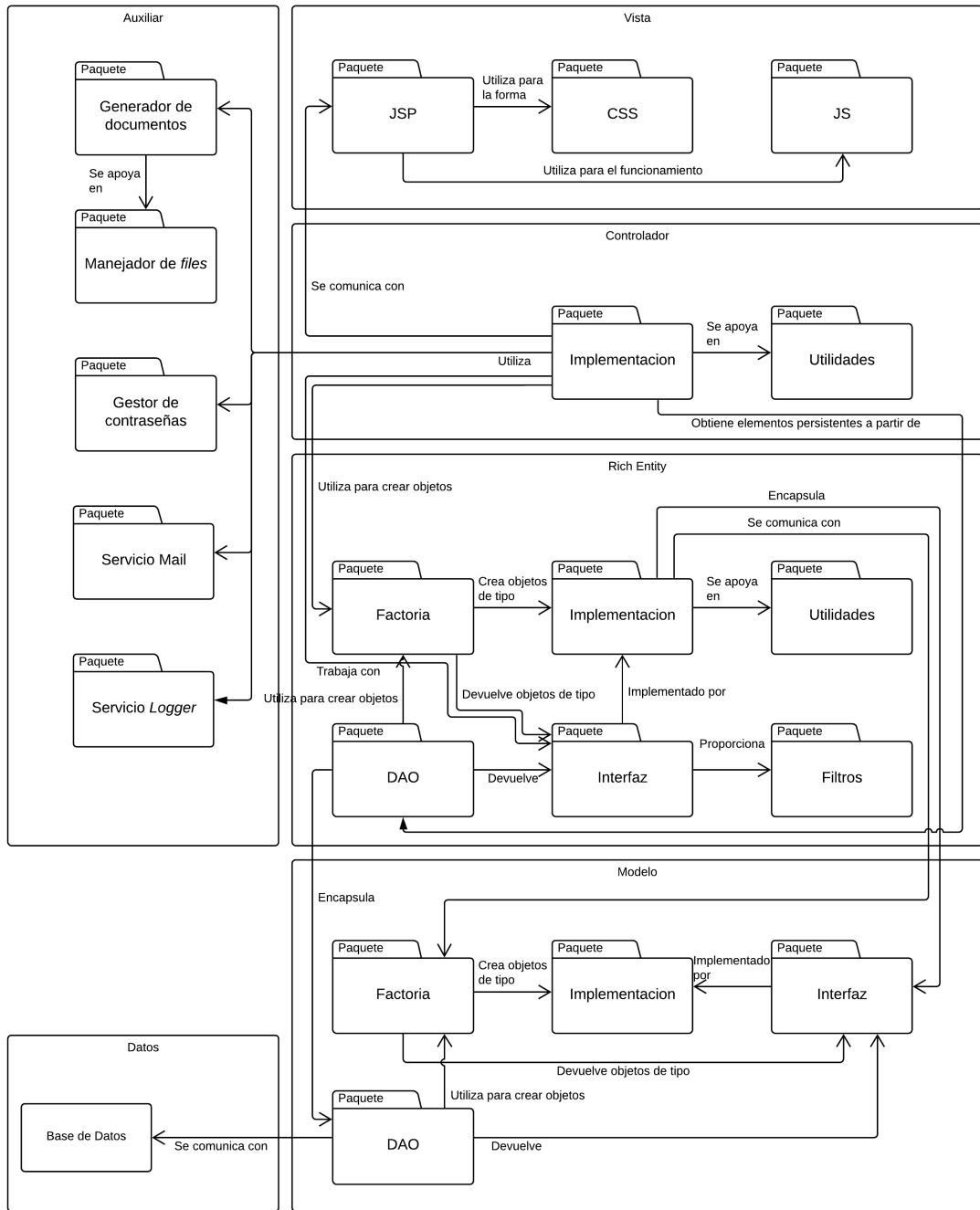


Figura 5.18: Diagrama de paquetes de la aplicación (iteración 4)

5.7.4 Implementación

En esta etapa, hemos llevado a cabo los cambios y mejoras propuestos para la aplicación, así como las nuevas funcionalidades pedidas, y, a la vez, se ha seguido desarrollando la **interfaz provisional**. A pesar de que, a nivel de análisis y diseño, esta iteración

no ha sido demasiado compleja, su implementación ha sido laboriosa, debido a la necesidad de añadir entidades al modelo y a la capa *rich entity*, lo cual ha obligado a cambiar la implementación de ciertos métodos en el controlador, algo no deseable pero necesario en este caso, debido a una previsión insuficiente. Por suerte, hemos conseguido mitigar el impacto que esto ha supuesto gracias al plan de riesgos establecido, dado que una de las medidas implementadas ha sido planificar cada fase e iteración con margen suficiente para resolver imprevistos. Las características más interesantes implementadas en esta iteración han sido:

Servicio de correo

La aplicación ahora posee un **servicio de correo** que permite enviar mensajes a los usuarios, informando así, entre otras cosas, de cambios en su perfil, o permitiendo recuperar la contraseña en caso de haberla olvidado.

Seguridad

El software web ahora utiliza el **protocolo TLS** para garantizar la seguridad de los datos enviados. En este momento, cuenta con un certificado autogenerado, por lo que el usuario podría, aún así, desconfiar de la aplicación. Sin embargo, esto es fácil de solucionar si se consigue un certificado proveniente de una *entidad emisora*.

Logger

El software ahora cuenta con un **servicio de loggeo** de información, fácilmente adaptable y ampliable, y que permite a los administradores conocer todo aquello que esté ocurriendo en la aplicación en cualquier momento. En principio viene preparado para registrar cualquier cambio que se haga en la base de datos, así como cualquier error que se produzca en la aplicación.

5.7.5 Pruebas

Se han ampliado las pruebas comenzadas en la iteración anterior, aumentando el número de casos de uso *testeados*, y cubriendo también las nuevas funcionalidades, completando así el conjunto de **pruebas unitarias** y **de integración** de la aplicación. De nuevo, se han continuado realizando **pruebas ad-hoc**, como forma de eliminar los errores más evidentes. Además, se ha comprobado que los tiempos de carga están dentro de lo esperable de una página de estas características, y, si la vista final no supone una carga demasiado pesada para la aplicación, deberían ser aceptables para el cliente.

5.7.6 Revisión de la iteración

En esta iteración, todas las funcionalidades pedidas por el cliente, más aquellas propuestas por el equipo de desarrollo y aceptadas por este, han sido implementadas y *testeadas*, hasta llegar a un punto en el que se ha considerado que el estado de la aplicación era aceptable para dar por cerrada la iteración. También se han ampliado algunas de las funcionalidades ya existentes. Por ejemplo, se han implementado nuevos filtros, para ofrecerle al usuario una experiencia de navegación adecuada, ya que la búsqueda de ejercicios y conjuntos es una característica esencial de nuestro sistema.

5.8 Iteración 5

Ahora que contamos con un sistema de *software* que cumple todos los requisitos funcionales preestablecidos, y cuyas funcionalidades han sido *testeadas* de diversas formas, se plantea esta quinta iteración, la tercera en la fase de construcción, con el fin de darle al producto **una interfaz de usuario adecuada**.

5.8.1 Requisitos

Dado que ya se dispone de todos los requisitos que determinarán el funcionamiento de la aplicación final, solo recordaremos aquellos que sean importantes para el desarrollo de las vistas. Estos son **requisitos no funcionales**, dado que no cambian las funcionalidades de la aplicación. Por tanto, necesitamos:

- Que el sistema sea **rápido**.
- Que la interfaz sea **de fácil uso**.

Dado que en una gran parte el primer requisito viene determinado por los módulos de *software* construidos en las dos iteraciones anteriores, llevaremos a cabo esta iteración **centrándonos en conseguir cumplir el segundo**, y el primero solo será tomado en consideración si en algún punto del desarrollo percibimos un deterioro notable en el rendimiento de la aplicación.

5.8.2 Análisis

Dado que la mayor parte de la lógica de funcionamiento que aporta la vista es notablemente simple, debido a que su función principal consiste en recibir información del controlador, *parsearla* de una determinada manera, mostrarla al usuario, y recibir las peticiones que este haga sobre dicha información, **no se ha realizado una etapa de análisis en esta iteración**.

5.8.3 Diseño

El objetivo de este paso ha sido terminar de dar forma a la interfaz de usuario. **No se han realizado diagramas de paquetes o de clases**, más allá del diagrama mostrado en la Figura 5.18, dado que, por la simpleza de la aplicación, no tendremos un número de pantallas demasiado alto. Sin embargo, **sí se han creado cuantiosos *mockups***, poniendo especial empeño en aquellos que representarían las vistas más complejas del sistema. Estos *mockups* **guiarán la implementación** que se realizará en esta iteración, y de todos ellos se han creado tres versiones: una orientada a la visualización en PC, otra a *tablet*, y otra a móvil. Para ejemplificar esto, se han incluido en la memoria la Figura 5.19, la Figura 5.20, y la Figura 5.21, que ilustran el diseño que tendrá la interfaz final en los distintos dispositivos para la vista de listado de ejercicios. Como podemos observar, estos diseños son más elaborados y específicos que los realizados en la iteración 2.

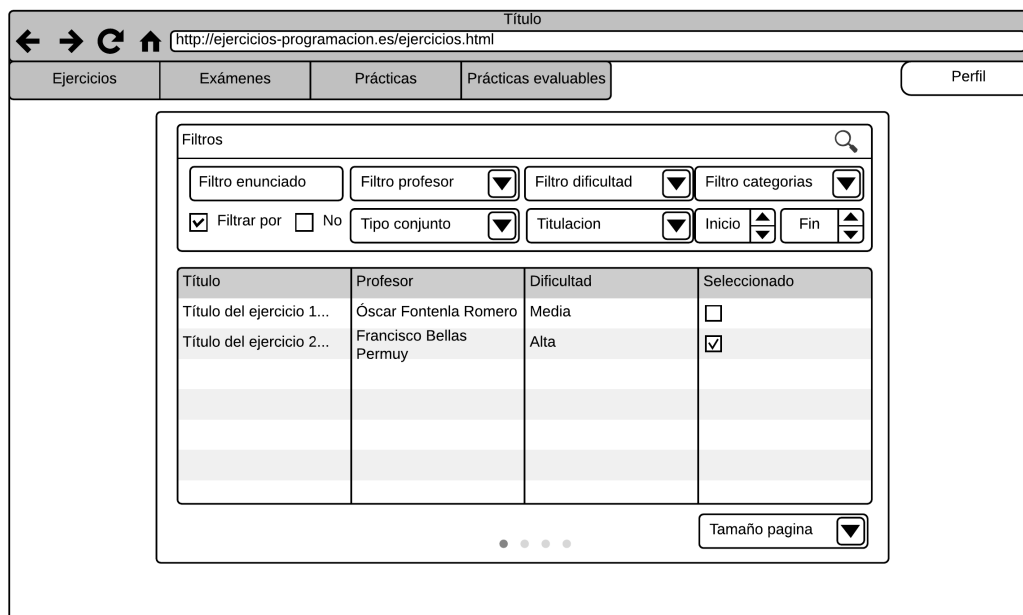


Figura 5.19: Mockup de la vista del listado de ejercicios en un PC (iteración 5)

5.8.4 Implementación

En este paso, se ha procedido a la implementación de la interfaz de usuario, a partir de los diseños anteriormente descritos, y tomando como base la vista provisional desarrollada durante las dos iteraciones anteriores.

El principal cambio que se ha producido ha sido la **introducción de la librería *MaterializeCSS***, dado que la vista provisional solamente se valía de los elementos básicos propor-

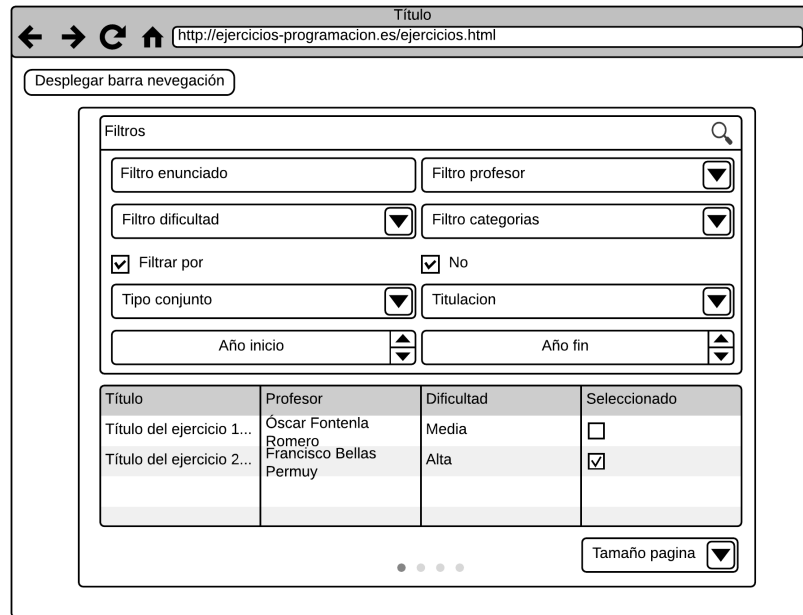


Figura 5.20: Mockup de la vista del listado de ejercicios en una *tablet* (iteración 5)

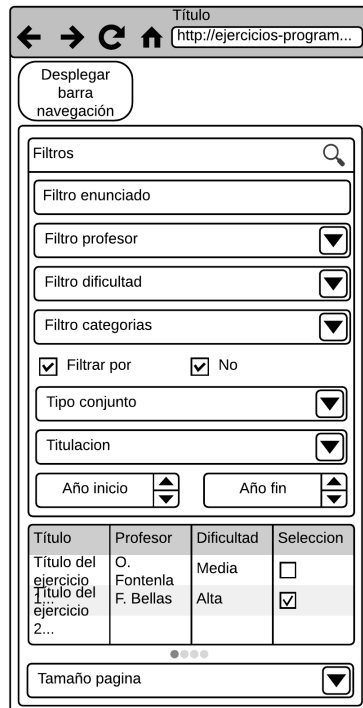


Figura 5.21: Mockup de la vista del listado de ejercicios en un móvil (iteración 5)

cionados por la tecnología *JSP*. A pesar de que ha habido un pequeño tiempo de adaptación al *framework*, esta iteración se ha realizado con relativa rapidez, y no se han encontrado demasiados obstáculos que interrumpieran su desarrollo.

5.8.5 Pruebas

Las pruebas unitarias y de integración se completaron en la iteración anterior. Sin embargo, el proceso de *testeo* todavía no ha finalizado, dado que esta última capa también utiliza código, aunque trabaje con métodos muy simples.

En primer lugar, se han realizado **pruebas de interfaz**, para comprobar que la vista se ajusta a los *mockups* y da acceso a todas las funcionalidades acordadas entre el cliente y el desarrollador.

También se han realizado **pruebas de compatibilidad**, para garantizar que la experiencia de navegación es adecuada para todos los usuarios.

Por último, se han realizado las pruebas más importantes de esta fase, las **pruebas de aceptación**. Una vez terminado el desarrollo de esta iteración, el cliente ha podido probar que la aplicación web cumple sus expectativas, aunque todavía en el entorno de ejecución del desarrollador.

5.8.6 Revisión de la iteración

En esta iteración, ha concluido el principal esfuerzo en cuanto a la implementación del programa pedido. Se ha terminado la implementación de los requisitos exigidos, se han hecho las pruebas pertinentes, y se ha obtenido el visto bueno del usuario. Se da por terminada, por tanto, la fase de construcción de *software* del proyecto.

5.9 Fase de Transición

UNA vez que el proyecto entra en la **fase de transición**, el sistema ha alcanzado la capacidad operativa inicial. El jefe de proyecto ha considerado que el sistema ofrece la confianza suficiente como para operar en el entorno del usuario, aunque no sea necesariamente perfecto [12]. Esta fase está compuesta únicamente de una iteración, que se describirá a continuación.

5.10 Iteración 6

Esta iteración ha resultado ser muy sencilla, como cabía esperar, dada la fase en la que nos encontramos, y las características de nuestro proyecto. Dada la simpleza de la misma, se

ha creído preferible explicarla en su conjunto en lugar de desglosarla como se ha hecho con el resto de iteraciones.

Lo primero que se ha hecho, ha sido **estudiar los bugs** encontrados por los usuarios en las pruebas de aceptación realizadas en la iteración anterior, así como las nuevas funcionalidades pedidas. Los nuevos requisitos son muy simples y no cambian la estructura del proyecto, por lo que no necesitamos realizar otra iteración para implementarlos. De estos, el más destacable ha sido el conseguir que el sistema **permita al usuario ordenar las listas de ejercicios y conjuntos por varios criterios**. Asimismo, los *bugs* identificados, mayormente relacionados con la codificación de la web y los caracteres especiales admitidos, han sido corregidos.

Tras esto, la aplicación ha sido presentada a los usuarios finales, con el fin de realizar un nuevo tipo de *testeo* conocido comúnmente como **pruebas beta**. La principal diferencia de estas pruebas con respecto a las pruebas de aceptación es que **estas las realizan los usuarios** (aunque en nuestro caso, evidentemente, esto es irrelevante, dado que nuestros usuario son también nuestros clientes), y además **se realizan en el entorno de ejecución de los propios usuarios**.

Una vez los usuarios probaron el sistema, reportaron, de nuevo, los *bugs* encontrados, que constituían ya un conjunto muy pequeño debido a que los más importantes ya habían sido previamente identificados y abordados por el desarrollador. Sin necesidad de ningún tipo de análisis más avanzado, se procedió a la corrección de dichos *bugs*.

Una vez finalizado este paso, se dio por finalizada la fase de transición, **finalizando con ella el ciclo de vida**, y, en consecuencia, **terminando el proceso de desarrollo de la aplicación**.

El código de la aplicación desarrollada está disponible en el repositorio *Github*¹ público del proyecto.

¹<https://github.com/DaveDarkLion/TFG>

Planificación

ESTE capítulo estará dedicado a explicar cuál fue la planificación que siguió el proyecto, y por qué se eligió dicha planificación, así como todos los cambios que ha sufrido durante el desarrollo.

Como herramienta de ayuda para esta tarea, se ha utilizado *Microsoft Project*. Se ha considerado que era adecuada para realizar este trabajo porque el equipo de desarrollo estaba ya familiarizado con ella de antemano, de modo que no se ha tenido que invertir un tiempo innecesario en conocer el entorno.

6.1 Planificación inicial

Una vez obtenida la descripción inicial del sistema, se puede proceder a realizar una primera planificación, que servirá para orientar las planificaciones que se harán más adelante. Dado que aún carecemos de suficiente información como para hacer una valoración precisa, de momento simplemente esbozaremos, a grandes rasgos, la estructura de la planificación del proyecto, intentando dejar un margen suficiente como para que cuando sea desglosada en partes más pequeñas y concretas, estas tengan asignadas un tiempo razonable para que sean desarrolladas. En la [Figura 6.1](#) se puede observar esta planificación inicial.

Como podemos ver, el proyecto se desarrollará en **un único ciclo de vida**, compuesto por **las fases habituales** de un proyecto que sigue la metodología anteriormente estipulada. La estimación del tiempo que durará cada iteración, a falta de más información, ha sido realizada a partir del estudio bibliográfico que se ha hecho del **Proceso Unificado** [12], así como de la escasa experiencia del desarrollador. Se ha intentado dar un margen razonable a todas las iteraciones, con vistas a la aparición de imprevistos, dado que, en teoría, **disponemos de suficiente tiempo con respecto a los deadlines**. Además, se ha incluido y tenido en cuenta el tiempo que durará el **desarrollo de la memoria**, dado que forma parte del proyecto pues **representa la documentación** formal del mismo.

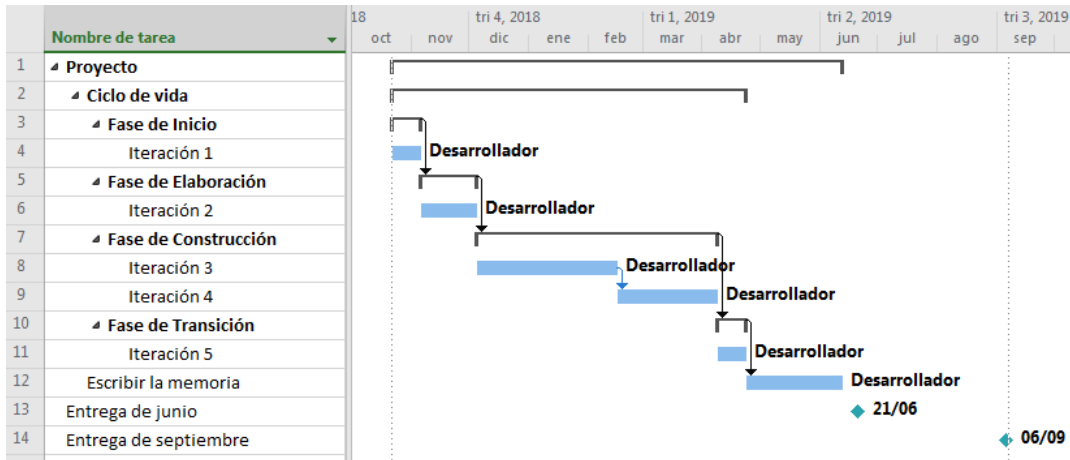


Figura 6.1: Diagrama de Gantt inicial

El objetivo a alcanzar con esta planificación es **finalizar el proyecto antes de la entrega de junio**, para así entregar en esta fecha. Si esto no fuera posible, disponemos de un segundo *deadline*, en septiembre, el cual deberíamos poder respetar sin problemas.

En principio, se trabajará **un total de 7 horas por día, y se descansará el fin de semana**.

6.2 Planificación de la fase de inicio

En esta fase se ha decidido realizar una sola iteración, dado que es lo habitual en esta metodología, y no existe ninguna razón que nos impulse a realizar un trabajo más complejo.

6.2.1 Iteración 1

En esta iteración, se ha decidido realizar únicamente los tres primeros pasos de una iteración genérica, dado que no se considera necesaria la construcción de un prototipo para demostrar la viabilidad al cliente. En la figura [Figura 6.2](#), podemos ver el diagrama de gantt correspondiente a esta primera iteración.

Dado que esta iteración ha sido completada en el tiempo estimado, no se mostrará el diagrama de Gantt en el que se compara con la línea base.

6.3 Planificación de la fase de elaboración

En esta fase, de nuevo, se ha planteado únicamente una iteración, dado que la complejidad de los requisitos no es suficientemente elevada como para necesitar un desglose de tareas mayor.

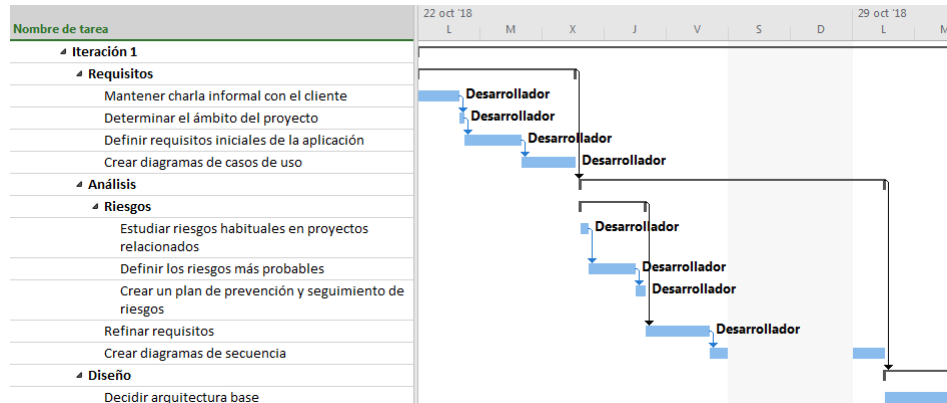


Figura 6.2: Diagrama de *Gantt* (iteración 1)

6.3.1 Iteración 2

La segunda iteración de nuestro proyecto se encargará de dar forma al sistema, puesto que su principal objetivo es desarrollar el análisis y el diseño del sistema a construir.

Dado que el cliente no podía concertar la reunión necesaria para dar comienzo a esta iteración hasta un cierto día, y a pesar de que la primera iteración ha sido llevada a cabo en el tiempo previsto, la segunda iteración empezará con algo de retraso. La planificación que se ha realizado aparece reflejada en la [Figura 6.3](#).

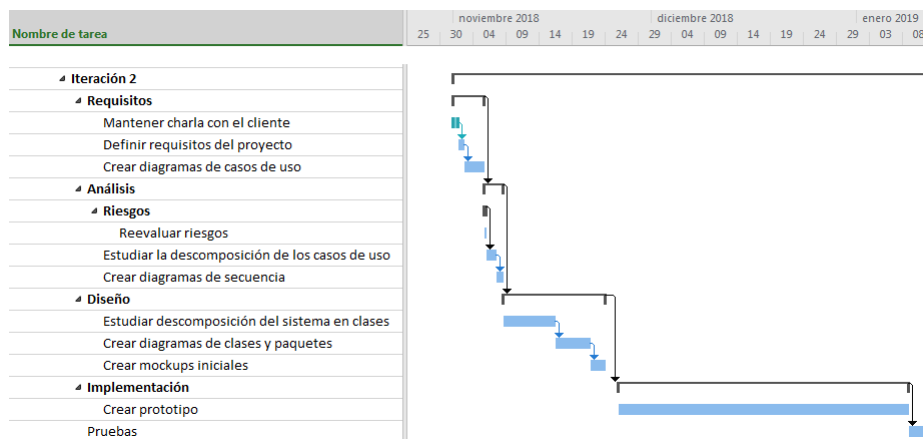


Figura 6.3: Diagrama de *Gantt* (iteración 2)

En la [Figura 6.4](#), podemos observar la diferencia entre la planificación y el transcurso real del proyecto.

Como podemos ver, esta segunda iteración ha sido completada con retraso, debido a que la implementación del prototipo resultó más compleja de lo previsto, principalmente porque muchas de las tecnologías que se comenzaron a utilizar en este paso eran desconocidas para

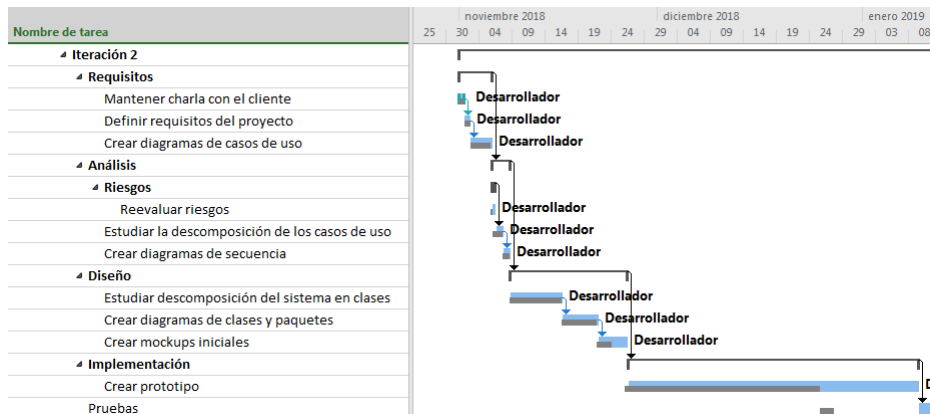


Figura 6.4: Diagrama de Gantt al finalizar la iteración 2

el desarrollador. A pesar de esto, el plan original no cambia, y, de momento, nos fijaremos el primer *deadline* como objetivo.

6.4 Planificación de la fase de construcción

A pesar de que solo habíamos anticipado realizar dos iteraciones en esta fase, ha aparecido la necesidad de replanificar esta etapa del desarrollo, dado que han aparecido numerosos nuevos requisitos. Por lo tanto, esta fase estará finalmente **compuesta por tres iteraciones**, donde la iteración extra será en la que se implementen los nuevos requisitos. Dada la inclusión de esta nueva iteración, sumada a los potenciales retrasos con respecto a la planificación que se produzcan durante lo que queda de proyecto, se ha decidido **ignorar el primer *deadline***, y **centrarse en el segundo**.

6.4.1 Iteración 3

Esta iteración estará destinada a implementar todos los requisitos obtenidos y analizados en la iteración anterior, a excepción de la vista, a la que se le dedicará una iteración entera. Para poder hacer pruebas, y tener algo que mostrar en caso de que así fuera necesario, se implementará una vista provisional. Los nuevos requisitos serán abordados en la iteración siguiente.

Como se puede apreciar en el diagrama de la [Figura 6.5](#), esta iteración será algo más larga de lo que se había previsto en inicio. Esto se debe a que ahora disponemos de más información acerca del proyecto a desarrollar que cuando empezamos, por lo que las estimaciones deberían aproximarse más a la realidad que las primeras.

En la [Figura 6.6](#), podemos observar la diferencia entre la planificación y el transcurso real del proyecto.

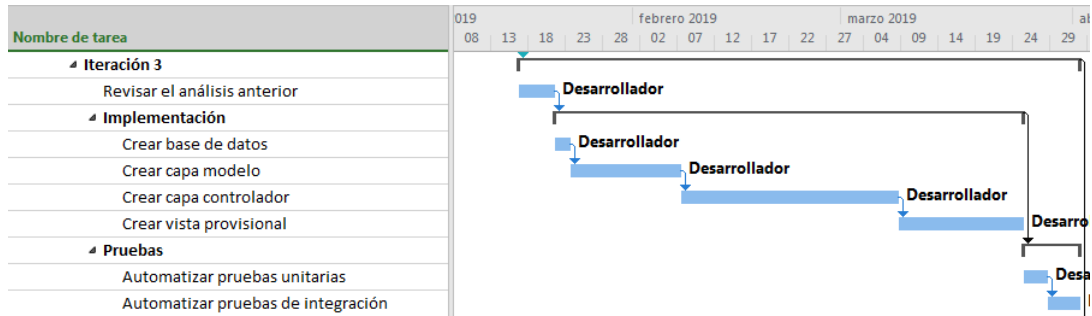


Figura 6.5: Diagrama de Gantt (iteración 3)

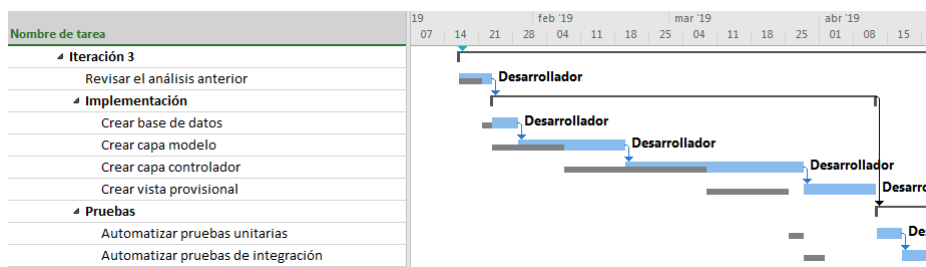


Figura 6.6: Diagrama de Gantt al finalizar la iteración 3

De nuevo, podemos comprobar que el desarrollo no se ajusta al plan que habíamos establecido. A pesar de que esto no es óptimo, ahora que el *deadline* ha cambiado, no debería de ser un problema excesivamente importante.

6.4.2 Iteración 4

Esta iteración estará dedicada a implementar los nuevos requisitos especificados por el usuario al comienzo de la fase. Dado que estos requisitos son completamente nuevos, se ha tenido que realizar el correspondiente análisis y diseño asociado, lo que ha ralentizado la iteración. A pesar de que esto no es idóneo, se ha considerado la única solución aceptable, a falta de tiempo para desarrollar otro ciclo de vida, que quizás hubiera sido lo mejor.

En la [Figura 6.7](#), podemos ver la planificación que se ha realizado para esta iteración.

En la [Figura 6.8](#), podemos observar la diferencia entre la planificación y el transcurso real del proyecto.

En esta iteración han aparecido nuevos retrasos, algo que, aunque es de esperar en un proyecto, nunca es deseable.

6.4.3 Iteración 5

El objetivo de esta iteración será el de desarrollar la vista final de la aplicación. En la [Figura 6.9](#), podemos ver como se ha planificado esta etapa, que es algo más simple que las dos

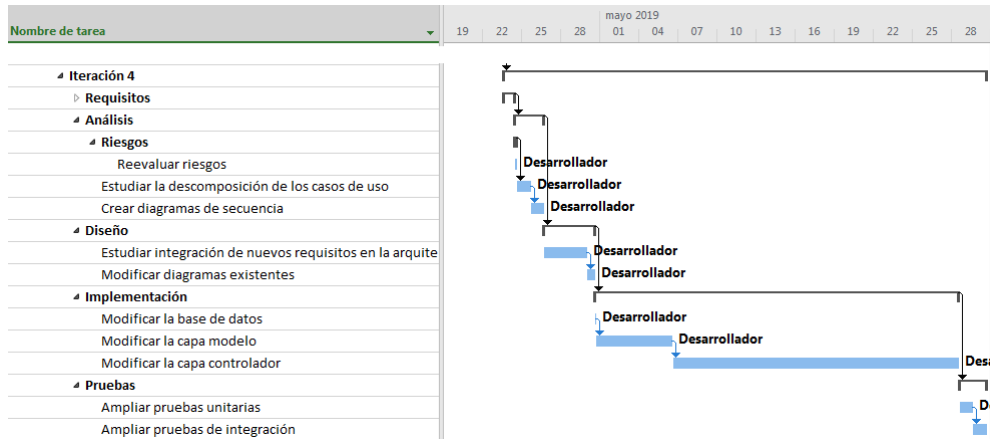


Figura 6.7: Diagrama de Gantt (iteración 4)

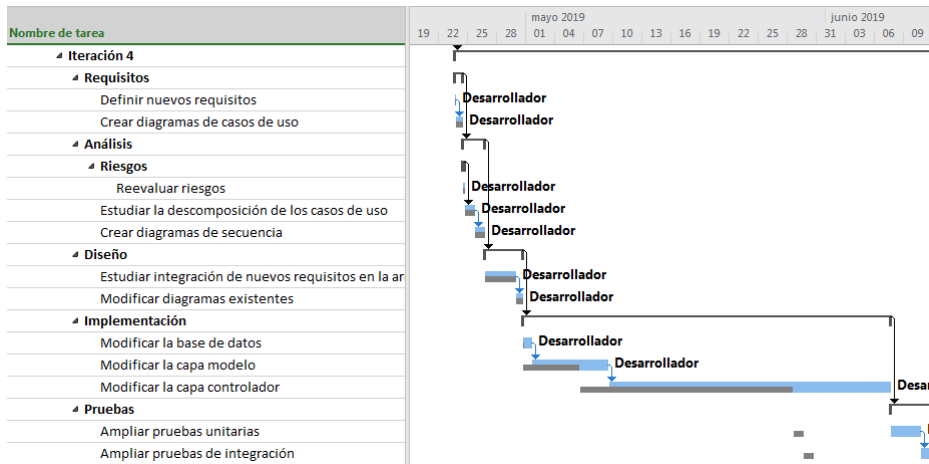


Figura 6.8: Diagrama de Gantt al finalizar la iteración 4

iteraciones anteriores.

En la Figura 6.10, podemos observar la diferencia entre la planificación y el transcurso real del proyecto.

A pesar de que, al igual que en las iteraciones anteriores, no se ha cumplido el plan establecido, el retraso existente es lo suficientemente pequeño como para considerar exitoso el desarrollo de esta iteración en particular.

6.5 Planificación de la fase de transición

El objetivo de esta fase es atar los cabos que queden sueltos de cara al lanzamiento de la aplicación, y dejarla preparada para el siguiente ciclo de vida. Esta fase suele estar compuesta por una única iteración, que será lo que se hará en nuestro proyecto.

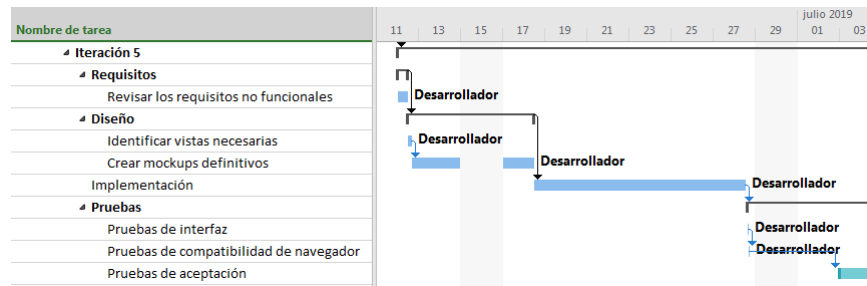


Figura 6.9: Diagrama de Gantt (iteración 5)

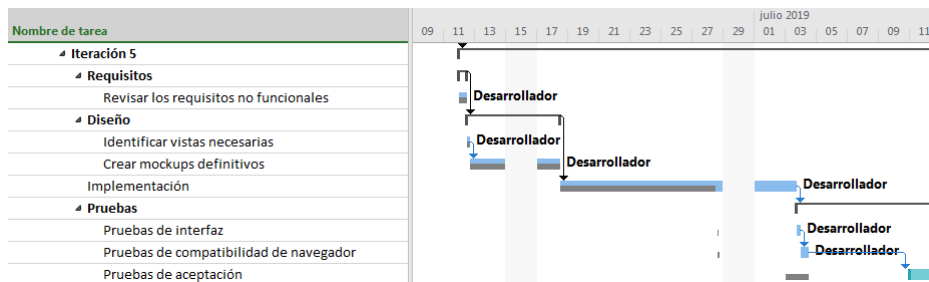


Figura 6.10: Diagrama de Gantt al finalizar la iteración 5

6.5.1 Iteración 6

En la Figura 6.11, podemos ver la planificación que se ha preparado para esta última iteración.

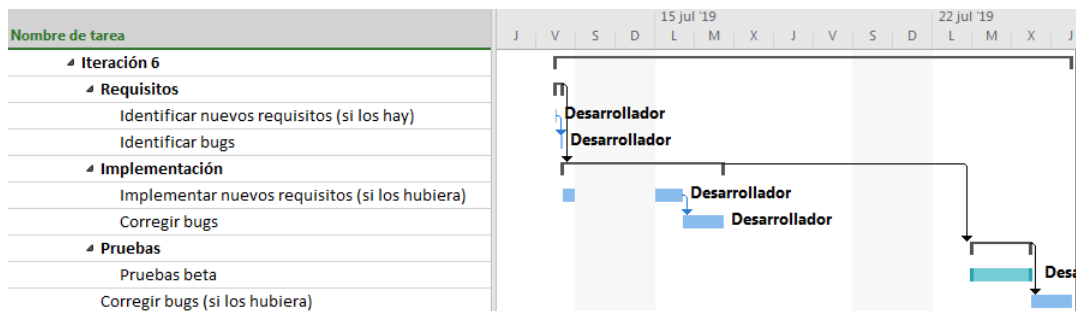


Figura 6.11: Diagrama de Gantt (iteración 6)

En la Figura 6.12, podemos observar la diferencia entre la planificación y el transcurso real del proyecto.

Esta iteración, a diferencia de las anteriores, ha acabado **antes del día previsto**, lo cual es todo un logro dentro de un proyecto de *software*. Sin embargo, dada la pequeña diferencia de tiempo que existe entre la planificación y la realidad, este hecho no tendrá una repercusión importante en el desarrollo del proyecto.

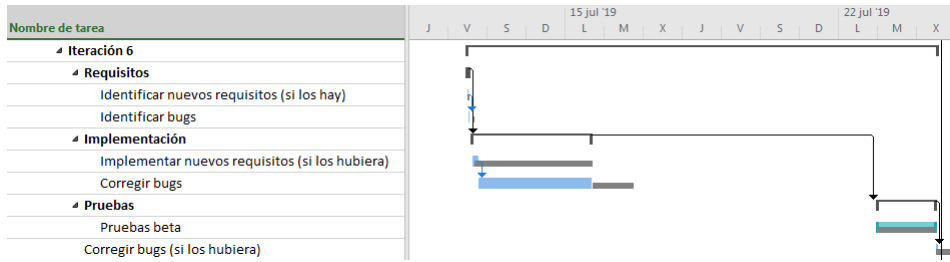


Figura 6.12: Diagrama de Gantt al finalizar la iteración 6

6.6 Coste del proyecto

La [Tabla 6.1](#) muestra las diferencias en cuanto a trabajo entre el plan de proyecto original sumando cada una de las iteraciones, y el desarrollo final, mientras que la [Tabla 6.2](#) muestra las diferencias entre el coste original y el esperado del proyecto. El coste del ingeniero ha sido calculado en base al *XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública (resolución de 22 de febrero de 2018, de la Dirección General de Empleo)*¹, que indica que el sueldo medio bruto de un analista programador es de 24.157.27 € al año, con una jornada de trabajo de 8 horas, lo cual nos da un total de 12,60 € la hora. Para simplificar la situación, la estimación se realizará suponiendo que el ingeniero de este proyecto ha cobrado 12 € / hora.

Previsto	Actual
847 h	1096 h

Tabla 6.1: Trabajo previsto y actual del proyecto

Elemento	Previsto	Actual
Ingeniero	10164,00 €	13.152,00 €
Hardware	1.100,00 €	1100,00 €
Licencia SO	300,00 €	300,00 €
Licencia RDBMS	0 €	0 €
Licencia IDE	0 €	0 €
Licencia Lenguaje desarrollo	0 €	0 €
Licencia frameworks	0 €	0 €
Licencia herramientas de pruebas	0 €	0 €
Material de aprendizaje	200,00 €	50,00 €
Total	11.814,00 €	14.602,00 €

Tabla 6.2: Coste previsto y actual del proyecto

¹<https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf>

6.7 Revisión de la planificación

Una vez terminada la aplicación, y a falta de acabar la memoria, vemos que, a pesar de que la planificación no ha sido óptima, y las estimaciones, en muchos casos, han estado lejos de la realidad (principalmente debido a la inexperiencia del desarrollador, lo cual dificultó la tarea, especialmente en los pasos de implementación), **hemos cumplido el objetivo marcado por el segundo *deadline***, por lo que se considera que **el desarrollo del proyecto ha sido llevado a cabo con éxito**.

Funcionalidades destacadas

ESTE capítulo está dedicado a exponer y explicar las funcionalidades más importantes del proyecto realizado.

La principal funcionalidad que implementa la aplicación desarrollada es el **almacenamiento de los ejercicios de programación** introducidos por los profesores, con toda su información asociada. Estos ejercicios pueden ser **creados, modificados, eliminados**, y, por supuesto, **visualizados**. Todos los ejercicios están siempre disponibles para los profesores, pero estos pueden **restringir su visibilidad** para los alumnos, de modo que es sencillo preparar de antemano ejercicios que podrían ser utilizados en alguna prueba evaluable en un futuro cercano. A los ejercicios **se les pueden adjuntar archivos**, que muy posiblemente, aunque no están limitados a ello, sean fragmentos de código. Aparte de la dificultad asignada por el profesor, los alumnos pueden **votar la dificultad** de un ejercicio. Cuando el ejercicio sea visualizado, la **dificultad media** calculada a partir de estos votos, así como el número de votos, aparecerá en pantalla.

Los profesores pueden crear **ideas**, que son bocetos de ejercicios que se almacenarán en la aplicación hasta que el profesor decida eliminarlas. Permiten crear directamente ejercicios a partir de ellas, para mayor comodidad del usuario.

También se podrán guardar en la aplicación agrupaciones de ejercicios, que se corresponden con **exámenes, prácticas, y prácticas evaluables**. A partir de ahora, nos referiremos grupalmente a ellas con el nombre de **conjuntos**, en lugar de tratar cada una de ellas individualmente, dado que todas ellas son iguales a nivel de funcionalidades.

Los **conjuntos** pueden ser, al igual que los ejercicios, **creados, modificados, eliminados, y visualizados**. Los ejercicios que contienen pueden ser **ordenados** arbitrariamente. Los **conjuntos**, al igual que los ejercicios, pueden ser o no **visibles** para los alumnos. Además, los **conjuntos** solo se podrán editar si están **abiertos**, que es una propiedad que estos poseen que indica si su estado es final y, en principio y en condiciones normales, no deberían ser modificados. En caso contrario, habrá que reabrir el conjunto primero si se desea modificar de

alguna forma. Los conjuntos ofrecen una opción de **descarga**, ya sea en formato *PDF* o *LaTeX*, que genera un fichero y lista los ejercicios contenidos en el mismo para crear un boletín, que luego puede ser imprimido en caso necesario.

Tanto los ejercicios como los conjuntos pueden ser **filtrados** y **ordenados** por un elevado número de criterios. Gracias a esto, se pueden hacer búsquedas tipo "ejercicios que hayan aparecido en exámenes en los últimos cinco años ordenados inversamente por dificultad". El poder realizar este tipo de búsquedas es una característica que se considera esencial, dado el objetivo de nuestro proyecto.

Los datos que utiliza la aplicación, tales como dificultades, categorías, titulaciones, o usuarios, pueden ser creados, modificados, y eliminados, por un **administrador**.

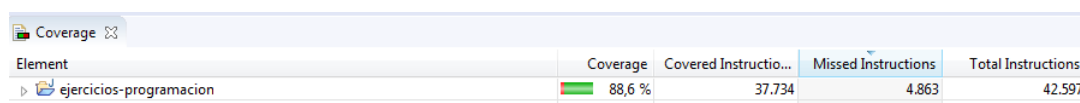
Además, los usuarios podrán **actualizar su contraseña**, o **restablecerla** en caso de haberla olvidado.

Capítulo 8

Pruebas

EL objetivo de este capítulo es exponer cuáles han sido las pruebas que nos han permitido verificar la validez del *software* desarrollado, así como la finalidad por la que se han realizado.

Debemos empezar este capítulo recalcando una noción que ha estado presente durante toda la memoria: la búsqueda de la simplicidad siempre que fuera posible. Para los *tests* automatizados utilizaremos **JUnit**, uno de los *frameworks* de pruebas más conocidos para *Java*, cuya principal característica es ser **user-friendly**. Además, también se contará con la ayuda de **Mockito**, un *framework* de creación de objetos falsos cuyo fin es aislar el funcionamiento de los distintos componentes de la aplicación, que permite escribir un código limpio y legible, cuando se requiera de ella. También hemos utilizado **Cobertura**, un *framework* que permite comprobar el porcentaje de código que cubren los *tests*. A pesar de que esto no es en ningún caso una prueba definitiva de una realización adecuada de las pruebas, si nos puede ayudar a hacernos una idea de cuánto de la aplicación se prueba realmente. En nuestro caso, tenemos **cerca de un 90% de coverage** por parte de los *tests automatizados*, como podemos ver en la [Figura 8.1](#), lo cual consideramos un porcentaje aceptable, dado que el otro 10% corresponde principalmente a módulos auxiliares difíciles de testear de forma automatizada, como el módulo de *loggeo* o el servicio de correo electrónico.



Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
ejercicios-programacion	88,6 %	37.734	4.863	42.597

Figura 8.1: Cobertura de los tests automatizados

Hay que añadir que, para los *tests* automatizados, se ha creado una **base de datos de prueba**, en la que se hace un **roll-back de la información** al final de cada test individual, de modo que los tests se pueden ejecutar incluso con la aplicación en marcha, dado que no modificará la base de datos principal, así como tantas veces como se desee, porque al final de

la ejecución la base de datos de prueba habrá vuelto a su estado original. Podemos encontrar el código de los *tests* automatizados en el repositorio *Github* ¹ público del proyecto.

Las pruebas realizadas han sido las siguientes:

8.1 Pruebas backend

El *backend testing* nos permite comprobar que la base de datos tiene una estructura adecuada y funciona como se espera. Esto se consigue conectándose a la *DB* directamente, sin interfaces de por medio, con algún programa, tipo *HeidiSQL* (el que se ha usado en este proyecto), y ejecutando unas cuantas *queries* para comprobar que todo está en orden. Este proceso se ha realizado nada más tener la estructura de la base de datos en un archivo *.sql*, al inicio de la programación del código de la aplicación, para garantizar que la *BD* sobre la que trabajarán el resto de componentes de la aplicación es la correcta.

8.2 Pruebas ad-hoc

El *ad-hoc testing* es aquel que se realiza en cualquier momento del desarrollo, sin referencias ni un plan de pruebas, simplemente ejecutando la aplicación y lanzando un determinado camino del flujo del programa. Se ha utilizado como un primer método de testeo para cualquier funcionalidad implementada, actuando de filtro para los errores más evidentes y fácilmente solucionables, antes de hacer cualquier otro tipo de prueba.

8.3 Pruebas de interfaz

Son aquellas llevadas a cabo por el desarrollador, para comprobar que la interfaz de la aplicación se ajusta a los *mockups* que sirvieron de guía para crearla. A pesar de que existen aplicaciones que pueden automatizar este proceso, y que se han tenido en cuenta, tales como *Squish* ², se ha preferido no utilizarlas, dado que el tiempo invertido en el aprendizaje y la configuración de dichas herramientas es demasiado alto y difícilmente merecerá la pena en una aplicación con un número de vistas tan limitado.

8.4 Pruebas de compatibilidad de navegador

Aquellas que sirven para comprobar si la aplicación web funciona como se espera en distintas combinaciones de navegadores y sistemas operativos. Se han realizado al final de la

¹https://github.com/DaveDarkLion/TFG/tree/master/src/test/java/rio/antelodel/david/ejercicios_programacion

²<https://www.froglogic.com/squish/>

programación del código, para garantizar que el número de usuarios potenciales de la aplicación sea el mayor posible.

8.5 Pruebas de aceptación

Las **pruebas de aceptación** las realiza el cliente, antes de la entrega final del producto, y su finalidad es garantizar que la aplicación cubre las necesidades de este. En nuestro proyecto, el cliente ha prestado su colaboración para comprobar que las funcionalidades del *software* son las esperadas.

Las pruebas de aceptación del proyecto se realizaron al concluir la fase de construcción, y el *feedback* recibido por parte del cliente fue utilizado en la siguiente iteración para modificar el *software*. La característica más destacable que pidió el cliente fue la inclusión de criterios de ordenación para los ejercicios y los conjuntos a los que pertenecen, con el fin de facilitar la elección de ejercicios para los distintos boletines y pruebas que se realizarán. Además, el cliente descubrió un *bug* que aparecía al introducir ciertos caracteres en los nombres y descripciones de la aplicación, relacionado con la codificación de la página y de la base de datos. Las características fueron implementadas y los *bugs* corregidos al inicio de la fase de transición.

8.6 Pruebas alpha

Las lleva a cabo el desarrollador, y su finalidad es identificar todos los posibles problemas del *software* antes de mostrárselo a los usuarios. Se han realizado con la intención de detectar *bugs*, para así poder corregirlos antes de hacérselo llegar al usuario.

8.7 Pruebas beta

El objetivo de las pruebas *beta* es comprobar el funcionamiento del software en un entorno real. Las realizan los usuarios, antes de la salida del producto al mercado. En nuestro caso, tras las pruebas de aceptación, y los cambios pertinentes que provocaron, los usuarios se han prestado a probar, desde sus dispositivos, una versión "final" de nuestro producto, con la intención de identificar los defectos del mismo, antes de su implementación definitiva en un servidor.

Conclusiones

EN este capítulo, se compararán los objetivos iniciales planteados, con los resultados obtenidos tras la realización del proyecto. Cabe aclarar que, dado que varios de los aspectos mencionados a continuación son, hasta cierto punto, subjetivos, se ha utilizado la opinión del cliente como baremo para comprobar su validez.

El principal objetivo de este proyecto, a grandes rasgos, era crear una aplicación que permitiera la gestión de una base de datos de ejercicios de programación, que sirviera de ayuda a los profesores a la hora de preparar ejercicios para las clases, crear exámenes, boletines de ejercicios... Esto se ha cumplido en su totalidad. La aplicación no solo permite almacenar y modificar estos elementos, junto con sus recursos e información asociada, sino que además cuenta con numerosos filtros y criterios de ordenación, destinados a ofrecer a los usuarios una forma sencilla de gestionar y visualizar esta información.

Otro objetivo importante planteado en este proyecto era que la aplicación estuviera disponible para el mayor número posible de usuarios. Esto, de nuevo, se ha logrado en su totalidad, gracias a dos características clave de la aplicación desarrollada. En primer lugar, se trata de una **aplicación web**, algo que garantiza que cualquier usuario con acceso a internet podrá usarla. Sin embargo, el hecho de que sea accesible desde cualquier dispositivo, no asegura que la experiencia de navegación vaya a ser igual de buena independientemente de la pantalla utilizada. Esto también ha sido solventado, gracias a la adopción de la aproximación **responsive**, como aspecto fundamental del diseño de la interfaz de usuario.

Otro aspecto que debíamos tener en cuenta era que la interfaz de usuario debía ser **clara, intuitiva, y de fácil manejo**. Esto ha sido posible gracias al seguimiento que se ha hecho de las pautas marcadas por la filosofía de diseño **Material**. Además, el cliente ha verificado que la interfaz es adecuada y se adapta a sus expectativas, durante la realización de las pruebas de aceptación.

Algo que no debemos olvidar es que la aplicación debía ser **rápida**. Esta es una meta que también se ha alcanzado. El cliente ha quedado satisfecho con los tiempos de respuesta que

ha mostrado la aplicación durante las pruebas de aceptación, y ha confirmado su aprobación del rendimiento de la aplicación en las pruebas beta, ya en un entorno real de trabajo.

Por último, queríamos una aplicación que fuese segura. ¿Se ha cumplido este objetivo? **Mayormente sí.** Se ha utilizado una de las librerías auxiliares de seguridad más populares (*Spring Security*) que existen como apoyo para implementar todo el sistema de **autenticaciones**, evitando así los ataques de suplantación de identidad más comunes. También se ha implementado el protocolo **TLS**, que controla la transferencia de información en toda la web, evitando así que los datos vayan en claro, siendo así fácil presa de numerosos ataques. Ahora bien, esto no nos garantiza que no existan vulnerabilidades en la aplicación, dado que es literalmente imposible garantizar tal cosa. Sin embargo, consideramos que el nivel de seguridad que ofrece la aplicación llegados a este punto es más que aceptable, teniendo en cuenta las características de la misma y, en especial, el hecho de que en ella no se trabaja con datos sensibles.

9.1 Lecciones aprendidas

Dado que la creación de una aplicación web es un trabajo bastante prototípico del empleo desempeñado por un desarrollador de *software*, existen numerosas lecciones que podemos extraer de los aciertos y los errores cometidos en el desarrollo de este proyecto.

Por una parte, se ha visto la **importancia de la planificación** en un proyecto *software*, y que puede ser la diferencia entre que el proyecto sea finalmente llevado a cabo o tenga que ser cancelado. Una mala planificación puede aumentar el coste del proyecto, así como hacer que no se cumplan los plazos de entrega, influyendo negativamente en la consideración en la que nos tiene el cliente y, consecuentemente, en la opinión pública, lo cual, a largo plazo, supondrá más pérdidas.

También se ha puesto en evidencia la importancia de **realizar un buen diseño**. Un buen diseño permitirá que la aplicación se adapte a los nuevos requisitos que desee incluir el cliente en medio de su desarrollo, sin necesidad de someter a grandes cambios la arquitectura de la aplicación.

Por último, y enlazando con el punto anterior, uno de los aspectos más importantes del desarrollo *software* es **mantener un nivel de comunicación adecuado con el cliente**. Dado que la aplicación final debe cumplir sus expectativas, lo que se debe hacer es mantener al cliente informado del rumbo que está tomando el desarrollo, para poder corregir cualquier desviación que se produzca de la ruta deseada, además de poder incluir los nuevos requisitos que este reclame una vez comenzado el proceso.

Trabajo futuro

PARA concluir el contenido de la memoria, dedicaremos este capítulo a discutir el trabajo futuro que se podría realizar si se deseara ampliar este proyecto.

Con el tiempo necesario y herramientas adecuadas, se podría llegar a construir una **plataforma de aprendizaje** a partir de la pequeña aplicación que ha dado como resultado este proyecto. Esta, en principio, sería una versión muy simplificada de otras plataformas de aprendizaje profesionales, dado que estas llevan bastantes años en constante evolución y cuentan con comunidades muy grandes de desarrolladores, algo de lo que nosotros no disponemos. Sin embargo, a pesar de que sería complicado que nuestro *software* se convirtiese en un competidor serio para los gigantes de este sector, tal vez podría llegar a ser lo suficientemente complejo como para poder adoptarlo en nuestra universidad como alternativa a *Moodle*. Aquí se muestran algunos de los primeros pasos a dar en caso de querer encaminar el proyecto en esta dirección:

10.1 Ampliar los datos de usuario

Tal como está ahora la aplicación, solo conocemos la información más básica de sus usuarios, tal como *email*, nombre, y apellidos. En una versión ampliada del software, debería de poder guardarse más información sobre los usuarios. Como mínimo, los usuarios deberían de poder **proporcionar una foto de perfil**, con el fin de que sean más fáciles de identificar, sobre todo si se decide implementar el punto siguiente.

10.2 Crear un foro dentro de la web

Podría ser interesante crear un foro dentro de la aplicación, para que tanto profesores como alumnos pudieran, entre otras cosas, debatir diferentes soluciones a un ejercicio concreto, preguntar sobre una herramienta determinada, o tal vez buscar un grupo de trabajo para una

práctica. Si se decidiera abordar este aspecto, también sería interesante **implementar un sistema de notificaciones**, que informara a los usuarios de cualquier actualización en los temas que sigan.

10.3 Internacionalizar la aplicación

En el momento de la entrega de este proyecto, la web está disponible únicamente en castellano. Esto se debe principalmente a que, en origen, la aplicación estaba destinada a ser usada únicamente dentro del entorno de la *UDC*. Obviamente, esto sería un problema si se quisiera que la aplicación fuera usada en todo el mundo. Sin embargo, añadir nuevos idiomas no debería de ser complicado, debido principalmente a que existen recursos, como los *Resource Bundles*, destinados a esta función. Los *Resource Bundles* son pares clave-valor, cuya clave es siempre la misma, independientemente del idioma, pero su valor depende del idioma del usuario que esté visualizando la página, y se corresponde con el texto que el usuario verá en su pantalla.

10.4 Integrar un editor de texto enriquecido

La aplicación almacena los enunciados de los ejercicios en texto plano. Esto está bien si la aplicación se va a destinar a la finalidad por la cual se ha realizado el proyecto, dado que los ejercicios a almacenar deberían de ser, por lo general, muy simples. Además, siempre podríamos editar el archivo *LaTeX* una vez descargado para adaptarlo al formato deseado. Sin embargo, si se piensa ampliar la aplicación, muy seguramente vayamos a querer guardar ejercicios relativamente complejos, que serán utilizados en diversos exámenes, prácticas... y no es eficiente estar cambiando su formato cada vez que aparece en algún conjunto de ejercicios. Es por esto por lo que sería una buena idea integrar un pequeño **editor de texto enriquecido** en la web, de modo que los profesores puedan guardar el enunciado en formato de texto enriquecido directamente.

10.5 Validar y mejorar la seguridad de la página

Como ya se ha expuesto anteriormente en esta memoria, las pruebas que se han realizado para identificar las vulnerabilidades existentes no se consideran suficientes. Si pretendiéramos ampliar esta aplicación, uno de los puntos clave que habría que abordar sería el de realizar las pruebas de seguridad de forma apropiada. Para ello habría que contratar a un equipo de especialistas en seguridad que se encargaran de probar a fondo el *software*, para posteriormente corregir las vulnerabilidades detectadas.

Apéndices

Glosario de acrónimos

API *Application Programming Interface*: Conjunto de métodos claros y bien definidos que ofrece una aplicación para permitir la comunicación entre componentes o con otros sistemas.

BD, DB *Base de datos o Database*: Conjunto de datos pertenecientes al mismo contexto y almacenados sistemáticamente.

CSS *Cascading Style Sheet*: Lenguaje utilizado para dar formato a la visualización de un archivo *HTML* o *XML*.

DAO *Data Access Object*: Objeto que proporciona una interfaz abstracta entre la aplicación y una base de datos o cualquier otro mecanismo de persistencia de información que esta utilice.

HTML *Hypertext Markup Language*: Lenguaje estándar de los documentos que se quiere que sean visualizables en un navegador web.

MVC *Modelo-Vista-Controlador*: Patrón de diseño de *software* habitualmente utilizado en el desarrollo de aplicaciones web, que propone la creación de tres capas: la capa *modelo*, la capa *vista*, y la capa *controlador*.

ORM *Object-Relational Mapper*: Herramienta encargada de transformar la información de una base de datos en objetos de algún tipo y viceversa, con el fin de facilitar la comunicación del programador con la base de datos.

OWASP *Open Web Application Security Project*: Comunidad cuyo principal objetivo es fomentar la seguridad de aplicaciones web.

RDBMS *Relational Database Management System*: Sistema de gestión de bases de datos relacionales.

SQL *Structured Query Language*: Lenguaje utilizado para la gestión de información almacenada en bases de datos relacionales.

TFG *Trabajo de Fin de Grado*.

TLS *Transport Layer Security*: Protocolo criptográfico encargado de proporcionar seguridad de comunicaciones entre los equipos de una red.

UDC *Universidade Da Coruña*.

UML *Unified Modeling Language*: Lenguaje de modelado de propósito general que se enmarca dentro del campo del desarrollo *software* y cuya finalidad es proporcionar una forma estándar de visualizar el diseño de un sistema.

URL *Uniform Resource Locator*: Referencia a un recurso web que especifica su localización dentro de una red de equipos, y que se utiliza como mecanismo para recuperar dicho recurso.

USDP *Unified Software Development Process*: Marco de desarrollo iterativo e incremental centrado en los casos de uso.

WORA *Write Once, Run Anywhere*: Slogan creado por *Sun Microsystems* para ilustrar las ventajas multiplataforma que tiene el lenguaje de desarrollo *Java*.

XML *eXtensible Markup Language*: Lenguaje que define una serie de reglas para codificar documentos en un formato que sea tanto legible por humanos como por máquinas.

Terminología

COMO se habrá podido comprobar a estas alturas, esta memoria incluye numerosos términos en inglés. Los motivos que nos han llevado a utilizar estos vocablos son varios.

Por una parte, muchas de las palabras usadas en la memoria son **tan utilizadas en castellano que han pasado a formar parte de la lengua común**, llegando algunas de ellas a **estar recogidas por la RAE**. Este es el caso de la palabra *software*, por ejemplo. Sin embargo, algunas de las palabras utilizadas no están recogidas por la *Real Academia* ni forman parte de la jerga común. El motivo de la utilización de estas palabras es que sí son **de uso muy común en el ámbito de la informática**. El mejor ejemplo de esto es la palabra *framework*. Por último, también se han utilizado términos que poseen una traducción en castellano, pero que **no tiene las implicaciones y connotaciones de la palabra original**. Un buen ejemplo de esto sería el término *parse*, cuya traducción según *Google Translator* es *analizar gramaticalmente*. Como podemos ver, la traducción consta de varias palabras, por lo que es innecesariamente larga, y ni siquiera se ajusta demasiado al significado del término original, careciendo de sus connotaciones en el ámbito de la informática.

Manual de usuario

ESTE anexo constituye el manual de usuario de la aplicación, que intentará abarcar todos aquellos aspectos necesarios para que alguien ajeno al proyecto pueda usar el *software* sin tener problemas.

C.1 Descripción general del sistema

El producto desarrollado es una **aplicación web responsive**, cuyo principal objetivo es **permitir el almacenamiento de ejercicios de programación**, con todos sus recursos asociados, **así como los exámenes, prácticas, y prácticas evaluables a los que estos pertenecen**.

C.2 Usuarios

Lo primero que debemos hacer al abrir la web es **entrar con un usuario**, puesto que, de otra manera, no tendremos acceso a ninguna sección de la aplicación, más que al índice. Por tanto, seleccionaremos la opción **Entrar**, en la barra de navegación, o accederemos al enlace proporcionado en el índice.

Una vez en la vista de *login*, podemos **introducir las credenciales** para acceder o, en caso de que hayamos olvidado la contraseña, también podemos **restablecer el *password***, en cuyo caso nos será pedida una dirección de correo, a donde se enviará una nueva contraseña generada aleatoriamente, si existe una cuenta asociada a esta dirección. Para evitar el abuso de esta funcionalidad, se ha puesto un límite: no se podrá restablecer la contraseña más de una vez cada cinco minutos para una cuenta concreta.

Los usuarios podrán cambiar su contraseña cuando lo deseen. Las contraseñas, sin embargo, deben respetar unas ciertas normas: deben tener entre 6 y 10 caracteres, y solo pueden estar formadas por letras, números, y el caracter ”_”.

El sistema contempla **tres roles de usuario: Alumno, Profesor, y Administrador**. Un usuario puede tener más de un rol asignado.

C.3 Barra de navegación

La barra de navegación es la principal herramienta de la que disponen los usuarios para moverse por la página. Contiene enlaces a todas las vistas importantes de la aplicación, y sus elementos son distintos según los roles del usuario que esté utilizando la aplicación.

Lo primero que podemos ver es el botón **Volver**, que nos permitirá regresar a la página anterior. Después, tenemos los botones **Ejercicio**, **Examen**, **Práctica**, y **Práctica Evaluable**, que permitirán listar dichos elementos, y, si el usuario es un *Profesor* o un *Administrador*, crear nuevas entidades de los elementos, mediante los botones **Listar** y **Nuevo**.

Por último, en el lado derecho de la barra de navegación, podemos observar un icono con forma de carro, que se corresponde con **la selección de ejercicios del usuario en cuestión**, y el botón que contiene las opciones propias del usuario, que son **Perfil**, desde donde podremos cambiar la contraseña, y **Salir**, que es el botón de *logout* de la aplicación. En caso de que el usuario sea *Profesor*, tendrá también la opción de listar las *ideas* de profesor, que se corresponde con el botón **Mis ideas**, y también podrá crear una nueva *idea*, gracias al botón **Crear idea**.

C.4 Ejercicios

Los ejercicios son el centro de la aplicación, y todo está construido en torno a ellos.

C.4.1 Listado de ejercicios

Para listar los ejercicios, pulsaremos el botón correspondiente en la barra de navegación. Se mostrará una lista con los ejercicios correspondientes al número de página en el que nos encontramos, dado que las listas de la aplicación están paginadas. El tamaño de página puede ser elegido, de entre unos valores predeterminados.

Si pulsamos sobre el nombre de un ejercicio, podremos visualizar dicho ejercicio. Si pulsamos sobre el icono con forma de carro a la derecha de cada una de las filas de la tabla, añadiremos o borraremos el ejercicio al que corresponda de nuestra selección personal de ejercicios. Nótese que el icono del carro será distinto según cuál sea la acción a realizar.

Hay que tener en cuenta que si el usuario es un *Alumno*, solo podrá ver aquellos ejercicios que sean *visibles*.

Ordenación

Podemos pulsar sobre los nombres de las distintas cabeceras de la tabla para **ordenar los ejercicios**. Si se pulsa una única vez sobre una cabecera, los ejercicios serán ordenados por el criterio correspondiente en orden natural. Si se vuelve a pulsar sobre dicha cabecera, los ejercicios serán ordenados por el mismo criterio, pero en orden inverso. La ordenación por defecto del listado es la que utiliza el *título* del ejercicio como criterio, y los ordena de forma natural.

Filtros

Si desplegamos el contenedor superior, llamado **Filtros**, tendremos acceso a los filtros que ofrece la aplicación para los ejercicios. Los filtros básicos, a los que tiene acceso cualquier usuario, son:

- **Título o Enunciado:** Filtra los ejercicios por la coincidencia, ignorando mayúsculas y minúsculas, del título o enunciado, con el texto introducido. Si introducimos varias cadenas de texto separadas por comas, el sistema las interpretará como filtros distintos. Si alguna coincide con el ejercicio que está siendo filtrado, este será aceptado en la búsqueda.
- **Profesor:** Filtra los ejercicios por el profesor asociado al ejercicio.
- **Dificultad:** Filtra los ejercicios por la dificultad asignada por el profesor.
- **Categorías:** Filtra los ejercicios por sus categorías asociadas. Un ejercicio solo será aceptado si este posee todas las categorías seleccionadas en el filtro (es decir, las categorías actúan como un *AND*).

Los *Profesores* y *Administradores* tendrán acceso a un filtro más que los *Alumnos*. Este filtro permite seleccionar los ejercicios por aparición en un determinado conjunto en un rango concreto de tiempo. El filtro está compuesto por varios fragmentos, que se describen a continuación.

- **Filtrar por:** casilla utilizada para activar el filtro. Si esta casilla no está marcada, el filtro no entrará en acción.
- **No:** casilla utilizada para negar el resultado del filtro. En vez de, por ejemplo, recuperar los ejercicios que aparecieron en exámenes entre 2010 y 2015, obtendríamos aquellos que **NO** hubieran aparecido en ningún examen en esas fechas.
- **Tipo de conjunto:** Tipo de conjunto por el que filtrar, que puede ser *Examen*, *Práctica*, o *Práctica Evaluable*.

- **Titulación:** titulación a la que tienen que pertenecer los conjuntos que se van a comparar.
- **Año inicial:** Año mínimo al que tiene que pertenecer el conjunto para ser tenido en cuenta en el filtrado.
- **Año final:** Año máximo al que tiene que pertenecer el conjunto para ser tenido en cuenta en el filtrado.

Para filtrar los ejercicios una vez elegidos los valores de los filtros, simplemente hay que pulsar el icono con forma de lupa en la esquina superior derecha del contenedor.

Los filtros **interactúan entre sí como un AND**, lo cual quiere decir que **un ejercicio solo será aceptado en un filtrado si cumple todos los filtros que estén activados**.

C.4.2 Visualización de ejercicios

En esta pantalla podremos **ver el ejercicio** con toda su información relevante, incluidos los *archivos* que tiene asociados, que pueden ser *de entrada*, *de validación*, o *archivos solución*.

Desde esta vista podemos añadir o borrar el ejercicio de la selección personal, y, si se es *Profesor* o *Administrador*, acceder a la pantalla de edición de ejercicio. Además, los *Alumnos* podrán votar su opinión con respecto a la dificultad del mismo. La media de todos los votos realizados también será visible en esta pantalla.

C.4.3 Edición de ejercicios

Desde esta pantalla se podrá **editar toda la información** que posee el ejercicio, así como añadir o eliminar sus archivos asociados, que tendrán un tamaño máximo de 50 MB. También se podrá **eliminar el ejercicio** si así se desea.

C.5 Conjuntos

Los *exámenes*, *prácticas*, y *prácticas evaluables*, son, a nivel de funcionamiento, **iguales**. Para no repetirnos, se explicarán una única vez, usando el concepto de **conjunto**, que representa a cualquiera de las entidades anteriormente mencionadas.

C.5.1 Listado de conjuntos

Para listar los elementos de un conjunto, pulsaremos en el botón de la barra de tareas correspondiente al tipo de conjunto con el que queramos trabajar, y después elegiremos la opción **Listar**.

En la tabla de *conjuntos*, podemos pulsar en el nombre de cualquier *conjunto* para visualizarlo, o en el botón con el icono en forma de flecha, en la parte derecha de la tabla, para descargarlo en *PDF*. Si el usuario es *Profesor*, también tendrá un botón en forma de lápiz, que le permitirá acceder directamente al modo edición del *conjunto*, sin necesidad de visualizarlo primero.

El ordenado funciona exactamente igual que en el caso de los ejercicios, con la única diferencia de que no se pueden ordenar *conjuntos* por el criterio de la columna *dificultad*, por razones de implementación ajenas al ámbito de este manual de usuario.

El filtrado también funciona de igual manera que en caso del listado de ejercicios, con la diferencia de que los filtros son ligeramente distintos.

C.5.2 Visualización de conjuntos

Cuando se accede a la pantalla de visualización de un *conjunto*, se verá una tabla que contiene los ejercicios contenidos en dicho *conjunto*, con las propiedades habituales del listado de ejercicios, salvo que esta lista no se podrá filtrar ni ordenar, dado que los *conjuntos* almacenan los ejercicios en un orden concreto, que no tiene por que cumplir ningún criterio de ordenación.

En la parte inferior izquierda de la pantalla, además, tenemos el botón que nos permitirá descargar el *conjunto* en formato *LaTeX* o *PDF*, según se prefiera.

C.5.3 Modo edición de conjuntos

Si el *conjunto* está *abierto*, y el usuario es *Profesor* o *Administrador*, se mostrará un botón dentro del contenedor *Opciones* llamado **Modo edición**. En el modo edición se mostrará, además de la tabla anteriormente mencionada, la tabla de listado de ejercicios habitual, con sus correspondientes filtros, y que se puede ordenar por columnas. A partir de este momento, el icono con forma de carro a la derecha de cada uno de los elementos de las tablas, adquiere una nueva función: gestionar los ejercicios del *conjunto*. En vez de, como anteriormente, servir para añadir o eliminar ejercicios de la selección de usuario, ahora se utilizará para añadir o eliminar ejercicios del *conjunto* con el que estamos trabajando.

Además, en la tabla que contiene los ejercicios actualmente pertenecientes al *conjunto*, habrán aparecido dos nuevos botones: uno representado por una flecha apuntando hacia arriba, y otro representado por una flecha que apunta hacia abajo. Estos nos permitirán mover los ejercicios de posición dentro del *conjunto*, para obtener el orden que nosotros deseemos.

C.5.4 Edición de atributos de conjuntos

Dentro del contenedor *Opciones*, si el usuario es *Profesor* o *Administrador*, estará disponible el botón *Editar atributos*, que dará acceso a la pantalla de edición de atributos del *conjunto* en cuestión. El más destacable de estos atributos es **abierto**, que indica si la estructura de ejercicios que contiene el *conjunto* es modificable o no. Un *conjunto* que no está *abierto* se podrá reabrir, pero el sistema verificará si el usuario está seguro de la decisión, dado que esto, en principio, solo debería hacerse en caso de error al cerrarlo, pero, por lo general, un *conjunto* cerrado debería permanecer cerrado. Además, desde esta pantalla, también se podrá **eliminar** el *conjunto*, de ser necesario.

Una característica importante de la aplicación es que si un ejercicio *no visible* está en un *conjunto visible*, el ejercicio se volverá temporalmente *visible* hasta que el *conjunto* se vuelva invisible o sea eliminado. Si un *Alumno* decide, en ese periodo de visibilidad, añadir el ejercicio a su selección personal, una vez el ejercicio se vuelva *no visible* de nuevo, aunque seguirá estando en su selección, este no podrá verlo, descargarlo, o interactuar con él de forma alguna. Si un ejercicio *no visible* es temporalmente *visible*, la pantalla de edición del ejercicio lo especificará.

C.5.5 Selección de ejercicios del usuario

La **selección de ejercicios del usuario** es un *conjunto* que tiene la particularidad de que sus atributos no son editables, y de que tampoco posee *modo edición*, puesto que, cuando no está activo el *modo edición* de ningún otro *conjunto*, los ejercicios que se añadirán o eliminarán con el botón con el icono que representa el carro, son los de la selección de usuario.

C.6 Funcionalidades específicas para Profesores

Los *Profesores* podrán listar, crear, modificar, y eliminar, **ideas de ejercicio**. Estas actuarán como bocetos de ejercicios que se desean añadir a la aplicación pero que todavía no se han terminado de definir completamente. Para acceder al listado o añadir una *idea*, simplemente pulsaremos sobre el botón correspondiente en la barra de navegación, dentro del desplegable de usuario.

En el listado, en principio, solo se mostrarán las *ideas* creadas por el usuario actual, pero esto se puede cambiar activando el filtro *Mostrar todas*.

En la pantalla de edición, además de las opciones habituales, también podremos **convertir la idea en un ejercicio**, lo cual nos llevará a la pantalla de adición de ejercicios, asignando el texto de la *idea* al campo *enunciado*.

C.7 Funcionalidades específicas para Administradores

Los *Administradores* tendrán acceso a una serie de funcionalidades específicas que les permitirán configurar la aplicación para garantizar la correcta experiencia de uso del resto de usuarios.

Los *Administradores* podrán listar, crear, modificar, y eliminar, las **categorias, dificultades, y titulaciones** con las que se trabaja en la aplicación. Para acceder a estas opciones, el *Administrador* deberá desplegar la barra lateral de opciones de administrador. Una vez ahí, tendrá acceso a todos estos elementos, que son iguales entre sí conceptualmente, con la diferencia de que las dificultades, además de un *nombre*, tienen un *valor* numérico entre 0 y 10, que indicará el grado exacto de dificultad que implica dicha *dificultad*.

Además, los *Administradores* podrán listar, crear, y eliminar, *usuarios*. Esta funcionalidad está implementada de igual forma que en el caso de los anteriormente mencionados, excepto por una particularidad: **los usuarios también se podrán añadir desde un archivo**, para facilitar la tarea a los *Administradores*. El archivo **utilizará el formato JSON**, y cada uno de los usuarios estará formateado tal y como se muestra a continuación:

```
{"apellido2": "Alcantara", "apellido1": "Marquez",  
"nombre": "Jorge", "email": "jmarquez@hotmail.com",  
"roles_id": ["ROLE_ADMINISTRADOR",  
"ROLE_PROFESOR", "ROLE_ALUMNO"]}
```

El archivo final contendrá un conjunto de usuarios separados por comas, estando el conjunto encerrado entre los caracteres "[]".

Imágenes de la aplicación

EN este anexo, mostraremos algunos pantallazos de la aplicación final, con el objetivo de que, sin necesidad de tener que acceder a ella, el lector pueda hacerse una idea de la apariencia del producto *software* obtenido como consecuencia del desarrollo de este proyecto.

En la [Figura D.1](#), la [Figura D.2](#), la [Figura D.5](#), la [Figura D.6](#), y la [Figura D.7](#), podemos ver las que posiblemente sean las vistas más significativas de nuestra aplicación. Además, también se ha decidido incluir en este apéndice la [Figura D.3](#), que muestra la vista de visualización de ejercicio en un móvil, para ilustrar el concepto de *responsive*, tan mencionado en la memoria, y como lo implementa nuestra aplicación. Por último, y a modo de curiosidad, en la [Figura D.4](#), podemos ver la barra de navegación desplegable que el usuario tiene disponible en móvil y *tablet*, cuya existencia se debe a la falta de espacio en la parte superior de la pantalla.

[Volver](#)
[Ejercicios](#)
[Exámenes](#)
[Prácticas](#)
[Prácticas evaluables](#)
[Admin](#)

Pablo Vazquez

Ejercicios

Filtros

Título o enunciado:

Filtrar por No

Profesor: Fuentes Torres, Antonio

 Dificultad: **Cualquiera**

Tipo: Examen

 Titulación: **Cualquiera**

Desde: **2000**
 Hasta: **2020**

Título	Profesor	Dificultad	Acciones
Ejercicio 1	Fuentes Torres, Antonio	Baja	<input type="button" value="👍"/>
Ejercicio 3	Fuentes Torres, Antonio	Muy alta	<input type="button" value="👎"/>
Ejercicio 4	Fuentes Torres, Antonio	Muy baja	<input type="button" value="👍"/>

⏪
⏴
1
⏵
⏩

10

Figura D.1: Listado de ejercicios

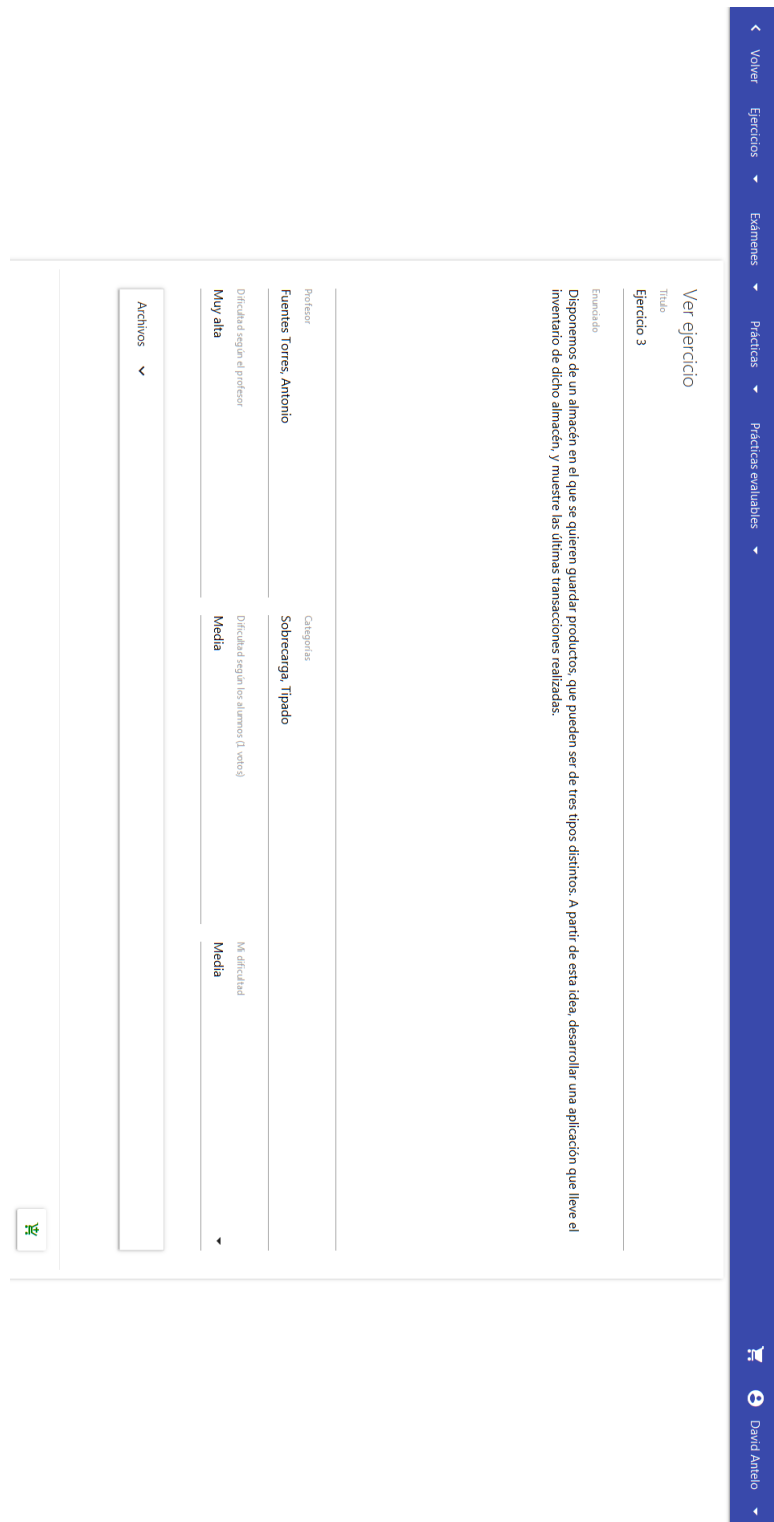


Figura D.2: Vista de un ejercicio



Ver ejercicio

Título

Ejercicio 3

Enunciado

Disponemos de un almacén en el que se quieren guardar productos, que pueden ser de tres tipos distintos. A partir de esta idea, desarrollar una aplicación que lleve el inventario de dicho almacén, y muestre las últimas transacciones realizadas.

Profesor

Fuentes Torres, Antonio

Categorías

Figura D.3: Vista de un ejercicio en un móvil

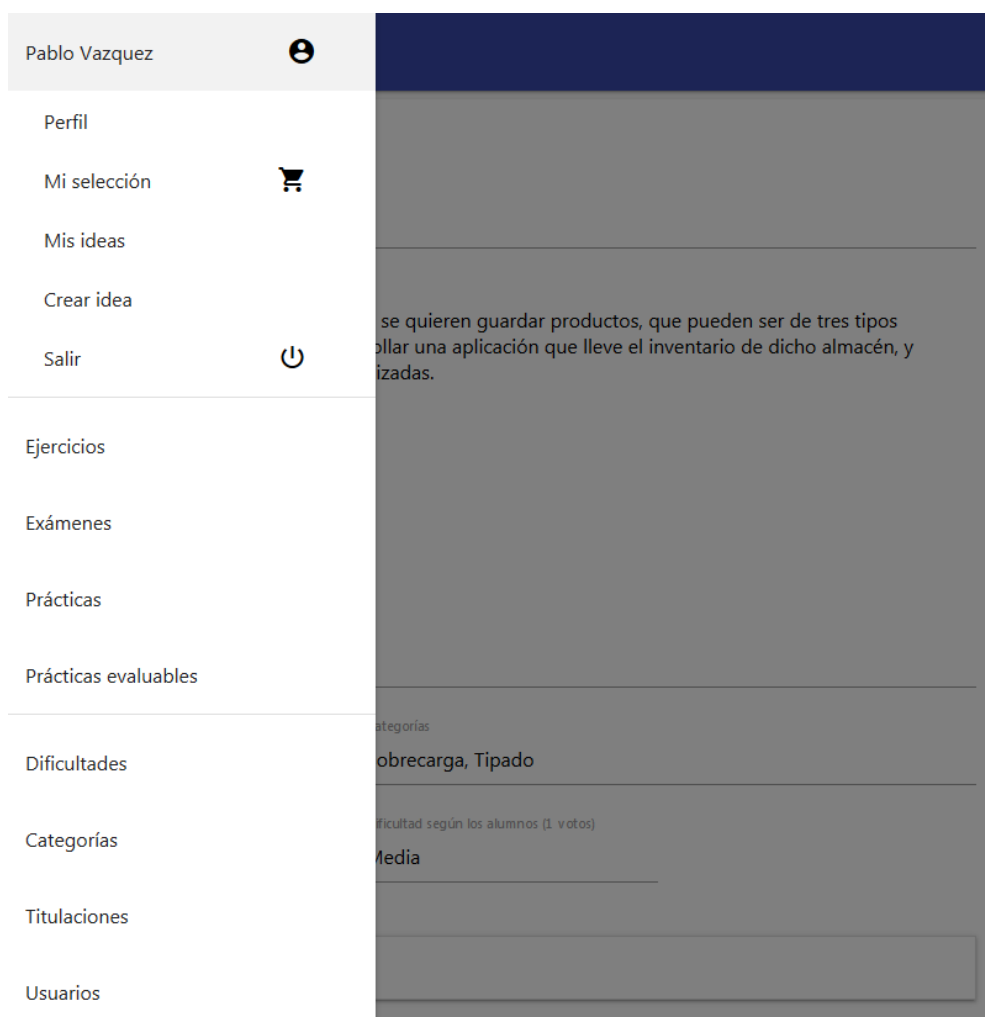


Figura D.4: Vista de la barra de navegación desplegable en tablet o móvil

Volver Ejercicios Exámenes Prácticas Prácticas evaluables Admin

Pablo Vazquez

Listado de exámenes

Filtros

Profesor: Cualquiera

Titulación: Cualquiera

Descripción: 2000

Hasta: 2020

Descripción	Titulación	Mes	Año	Dificultad
Examen de matemáticas de la primera convocatoria	Titulación en matemáticas	1	2016	Muy baja
Examen de química de la segunda convocatoria	Titulación en químicas	7	2012	Media

10

Figura D.5: Listado de exámenes

[Volver](#)
[Ejercicios](#)
[Exámenes](#)
[Prácticas](#)
[Prácticas evaluables](#)
[Admin](#)
Pablo Vazquez

Examen

Opciones

Título	Profesor	Dificultad	
Ejercicio 1	Fuentes Torres, Antonio	Baja	<input type="button" value="←"/> <input type="button" value="→"/> <input type="button" value="✖"/>
Ejercicio 4	Fuentes Torres, Antonio	Muy baja	<input type="button" value="←"/> <input type="button" value="→"/> <input type="button" value="✖"/>

Filtros

Título	Profesor	Dificultad	
Ejercicio 1	Fuentes Torres, Antonio	Baja	<input type="button" value="✖"/>
Ejercicio 2	Casado Perez, Eduardo	Media	<input type="button" value="✔"/>
Ejercicio 3	Fuentes Torres, Antonio	Muy alta	<input type="button" value="✔"/>
Ejercicio 4	Fuentes Torres, Antonio	Muy baja	<input type="button" value="✖"/>
Ejercicio 5	Casado Perez, Eduardo	Media	<input type="button" value="✔"/>

Figura D.6: Edición de los elementos que contiene un examen y sus posiciones

[Volver](#)
[Ejercicios](#)
[Exámenes](#)
[Prácticas](#)
[Prácticas evaluables](#)
[Admin](#)


 Pablo Vazquez

Mi selección

Título	Profesor	Dificultad		
Ejercicio 3	Fuentes Torres, Antonio	Muy alta	<input type="button" value="←"/> <input type="button" value="→"/>	<input type="button" value="✖"/>
Ejercicio 1	Fuentes Torres, Antonio	Baja	<input type="button" value="←"/> <input type="button" value="→"/>	<input type="button" value="✖"/>
Ejercicio 5	Casado Perez, Eduardo	Media	<input type="button" value="←"/> <input type="button" value="→"/>	<input type="button" value="✖"/>

Documento PDF

Figura D.7: Selección de ejercicios del usuario

Bibliografía

- [1] Leon Shklar, Richard Rosen, *Web Application Architecture: Principles, Protocols and Practices*. Wiley, 2003.
- [2] Bryan Sullivan, Vincent Liu, *Web Application Security, a Beginner's Guide*. McGraw Hill, 2011.
- [3] M. J. Hernandez, *Database Design for Mere Mortals*. Addison-Wesley Professional, 2003.
- [4] S. M. M. Tahaghoghi, *Learning MySQL*, 1st ed. O'Reilly Media, 2006.
- [5] E. F. Codd, *The Relational Model for Database Management, Version 2*. Addison-Wesley Professional, 1990.
- [6] P. C. Jorgensen, *Software Testing: A Craftsman's Approach*, 4th ed. CRC Press, 1995.
- [7] E. Marcotte, *Responsive Web Design*. Jeffrey Zeldman, 2011.
- [8] B. Frain, *Responsive Web Design with HTML5 and CSS3*. Packt Publishing, 2012.
- [9] Christian Bauer, Gavin King, Gary Gregory, *Java Persistence with Hibernate*, 2nd ed. Manning, 2006.
- [10] Anirudh Prabhu, Aravind Shenoy, *Introducing Materialize*. Apress, 2016.
- [11] Craig Walls, *Spring in Action*, 5th ed. Manning, 2005.
- [12] Ivar Jacobson, *The Unified Software Development Process*, 1st ed. Addison-Wesley, 1999.
- [13] Russ Miles, Kim Hamilton, *Learning UML 2.0*, 1st ed. O'Reilly Media, 2006.

