Aplicación web progresiva para la realización de cuestionarios

Progressive web application for taking tests

Zakaria El Fakhri Ouajih
Pedro Martínez Gamero

GRADO EN INGENIERÍA INFORMÁTICA

FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin de Grado
Curso académico: 2021/22

Director: Manuel Montenegro Montes

Autorización de difusión

Autor

Fecha
Madrid, junio de 2022

Los abajo firmantes, matriculados en el Grado en Ingeniería informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Grado: "Aplicación web progresiva para la realización de cuestionarios", realizado durante el curso académico 2021-2022 bajo la dirección de Manuel Montenegro Montes, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Enlace al repositorio de github

Resumen

Qwizer es una aplicación web cuya finalidad es permitir a un alumno realizar cuestionarios. Lo que destaca de esta aplicación web con respecto a otras de este tipo, como Moodle, es que esta es progresiva, es decir, funciona incluso cuando no hay conexión a internet; aunque la funcionalidad está limitada a realizar cuestionarios de manera *o ine*. Cuando el usuario realice un cuestionario de manera *o ine*, se le generará un código QR de sus respuestas, el cual deberá mostrar al profesor. Una vez que el usuario vuelva a tener conexión a internet, las respuestas del cuestionario se enviarán automáticamente. Se pueden distinguir tres grupos de usuarios para la aplicación:

- Alumnos: Los alumnos tienen la posibilidad de realizar los cuestionarios de las asignaturas en las que estén matriculados. Una vez enviadas las respuestas, el cuestionario se corrige automáticamente y el alumno podrá ver su cuestionario corregido, además de su nota.
- Profesores: Los profesores cuentan con la posibilidad de matricular alumnos en cualquiera de las asignaturas que ellos impartan. Con respecto a los cuestionarios, pueden añadirlos a estas asignaturas tanto haciendo uso de una página de la propia aplicación como subiendo los cuestionarios mediante un fichero. También pueden revisar las respuestas dadas por un alumno, así como las notas que ha obtenido. Por último tienen acceso a un banco de preguntas, al cual pueden añadir nuevas preguntas mediante un fichero y también descargar las preguntas para su posterior uso.
- Administradores: Mediante el panel de control tienen la posibilidad de crear asignaturas, dar de alta a usuarios (tanto alumnos como profesores), matricular alumnos en una asignatura y asignar profesores a una asignatura.

La aplicación tiene un diseño adaptativo, lo que permite que se visualice bien, tanto en pantallas de ordenadores, como en la de dispositivos móviles.

Palabras clave

Aplicación web progresiva, React JS, Django, Service Worker, Cuestionarios offline

Abstract

Qwizer is a web application which allows students to take tests. What makes this web application stand out from others of this type, such as Moodle, is that it is progressive, which means that it works even when there is no internet connection; although the functionality is limited to perform offline questionnaires. When the user takes a test offline, a QR code of the answers will be generated, which must be shown to the teacher. Once the user recovers the internet connection, the answers will be sent automatically. We can distinguish three groups of users for our web application:

- Students: Students have the possibility to take tests of the subjects for which they are enrolled. Once the answers are submitted, the test will be corrected automatically. The students will be able to see the corrected test, in addition to their grade.
- Teachers: Teachers have the possibility to enroll students in any of the subjects they teach. Regarding the questionnaires, they can create them either using the web page or uploading them through a file. They can also review the answers given by students, as well as the grades. And they also have access to a question bank, to which they can add new questions by uploading a file, as well as downloading them for later use.
- Administrators: Through the control panel they have the possibility to create subjects, register users (both students and teachers), enroll students in a subject and assign teachers to a subject.

The application has an adaptive design, which allows it to adjust to both computer and mobile devices screens.

Keywords

Progressive web application, React JS, Django, Service Worker, Offline quizzes

Índice general

Índice I

Agradecimientos IV

Dedicatoria V

1. Introducción: antecedentes, objetivos y plan de trabajo 1
   1.1. Motivación del proyecto . . . 1
   1.2. Objetivos . . . 4
   1.3. Plan de trabajo . . . 5
      1.3.1. Fase 1: Familiarización con las tecnologías . . . 5
      1.3.2. Fase 2: Implementación de una prueba de concepto . . . 5
      1.3.3. Fase 3: Creación de la base de datos y manejo de sesiones . . . 5
      1.3.4. Fase 4: Implementación de las asignaturas y cuestionarios . . . 6
      1.3.5. Fase 5: Corrección automática de los cuestionarios . . . 6
      1.3.6. Fase 6: PWA y Service Workers . . . 6
      1.3.7. Fase 7: Generación de Códigos QR . . . 6
      1.3.8. Tareas llevadas a cabo a lo largo de todas las fases . . . 7

2. Introduction 8
   2.1. Background and motivation . . . 8
   2.2. Goals . . . 10
   2.3. Working plan . . . 11
      2.3.1. Phase 1: Familiarization with the technologies . . . 11
      2.3.2. Phase 2: Implementation of a proof of concept . . . 12
      2.3.3. Phase 3: Database creation and session management . . . 12
      2.3.4. Phase 4: Implementation of the subjects and quizzes . . . 12
      2.3.5. Phase 6: PWA and Service Workers . . . 12
      2.3.6. Phase 7: QR Code generation . . . 12
      2.3.7. Tasks carried out throughout all phases . . . 13

3. Metodología y tecnologías usadas 14
   3.1. Metodología usada . . . 14
   3.2. Tecnologías, lenguajes y frameworks . . . 15
      3.2.1. Tecnologías utilizadas . . . 15
      3.2.2. Tecnologías descartadas . . . 21

4. Descripción del back-end y la base de datos 23
4.1. Arquitectura global del sistema . . . 23
4.2. Base de datos . . . 24
4.2.1. Banco de preguntas . . . 25
4.2.2. Modelo relacional de la base de datos . . . 26
4.2.3. Tablas de la base de datos . . . 27
4.3. API de comunicación entre back-end y front-end . . . 30
4.3.1. test . . . 30
4.3.2. test-corrected . . . 31
4.3.3. response . . . 31
4.3.4. login . . . 32
4.3.5. logout . . . 32
4.3.6. register . . . 32
4.3.7. get-subjects . . . 33
4.3.8. get-all-subjects . . . 33
4.3.9. get-quizzes . . . 33
4.3.10. get-subject-info . . . 33
4.3.11. get-quiz-info . . . 34
4.3.12. get-subject-questions . . . 34
4.3.13. upload . . . 35
4.3.14. upload-questions . . . 35
4.3.15. create-quiz . . . 35
4.3.16. get-quiz-grades . . . 36
4.3.17. delete-question . . . 37
4.3.18. update-question . . . 37
4.3.19. get-students . . . 37
4.3.20. enroll-students . . . 38
4.3.21. insert-qr . . . 38
4.3.22. get-hashes . . . 39
4.4. Aspectos específicos de Django que se han utilizado . . . 39
4.4.1. Panel de administración de Django . . . 39
4.4.2. Librerías utilizadas . . . 40

5. Descripción del front-end 42
5.1. Funcionalidad básica desde el punto de vista del usuario . . . 42
5.1.1. Estudiantes . . . 42
5.1.2. Profesores . . . 43
5.1.3. Administrador . . . 43
5.2. Librerías utilizadas en el front-end . . . 44
5.3. Descripción de los componentes de React más importantes . . . 45
5.3.1. NavBar . . . 45
5.3.2. TarjetaAsignatura . . . 46
5.3.3. TarjetaCuestionario . . . 46

5.3.4. QuestionContainer . . . 48
5.3.5. DataTable . . . 52

6. Otras funcionalidades 60
6.1. Subir preguntas . . . 60
6.1.1. Descripción del formato YAML . . . 61
6.2. Subir cuestionarios . . . 62
6.2.1. Descripción del formato YAML . . . 63
6.3. Explicación del modo offline . . . 64
6.3.1. Caso de uso . . . 64
6.3.2. Service Worker . . . 65
6.3.3. Código QR . . . 67

7. Conclusiones y trabajo futuro 68
7.1. Objetivos conseguidos . . . 68
7.2. Dificultades durante el desarrollo . . . 69
7.2.1. React . . . 69
7.2.2. Modificación del modelo *user* de Django . . . 70
7.2.3. Integración de React con Django . . . 70
7.2.4. Cifrado de los cuestionarios . . . 70
7.2.5. PWA y Service Worker . . . 71
7.3. Trabajo futuro . . . 72

8. Conclusions and future work 75
8.1. Achieved goals . . . 75
8.2. Difficulties encountered . . . 76
8.2.1. React . . . 76
8.2.2. Modifying Django *user* model . . . 77
8.2.3. React integration with Django . . . 77
8.2.4. Encryption of questionnaires . . . 77
8.2.5. PWA and Service Worker . . . 78
8.3. Future work . . . 79

9. Contribución personal de cada autor 82
9.1. Zakaria El Fakhri Ouajih . . . 83
9.2. Pedro Martínez Gamero . . . 86

Bibliografía 90

Agradecimientos

En primer lugar nos gustaría agradecer a nuestro director de proyecto, Manuel Montenegro Montes, por habernos ayudado y guiado durante el desarrollo del Trabajo Fin de Grado. Sobre todo por su flexibilidad y paciencia.

En segundo lugar nos gustaría agradecer a todos aquellos profesores que a lo largo de todos estos años nos han acompañado y permitido que hoy estemos llevando a cabo este proyecto.

Por último pero no menos importante nos gustaría agradecer a nuestras familias el habernos apoyado y ayudado durante todos estos años. Sin ellos no habríamos llegado hasta aquí.

Zakaria El Fakhri Ouajih y Pedro Martínez Gamero

Dedicatoria

Me gustaría dedicar este proyecto en primer lugar a mis padres. Muchas gracias por todo el apoyo y cariño que he recibido por vuestra parte desde que soy pequeño. Por todas esas veces que me habéis ayudado en todo lo posible y por la paciencia que habéis tenido conmigo. No puedo estar mas orgulloso de vosotros.

En segundo lugar me gustaría dedicárselo a mis hermanos, Abel y Jaime. Al primero de ellos porque fue por él que comencé a estudiar esta carrera, ya que desde pequeño me interesé en la informática gracias a verle en su cuarto trasteando con ordenadores. Al segundo de ellos por todas las veces que nos hemos ayudado mutuamente desde pequeños.

También me gustaría dedicarle este trabajo a esa persona, que aunque ya no esté ahí me ha servido de motivación. Esto va por ti abuelo Basilio.

A Julia y Alicia, esas dos amigas que han estado ahí desde que era pequeño, ayudándome en todo lo que podían, escuchándome cada vez que tenía un problema o, simplemente, quedando para desconectar. Muchas gracias por estos años, que sin dudas han sido una gran aventura a vuestro lado. Sin vosotras estos años no habrían sido iguales. También me gustaría dedicárselo a esos amigos que también han estado presentes ahí cuando les necesitaba como Enrique ER, Daniel, Iván o Chozas. Muchas gracias.

Por último me gustaría dedicarle este proyecto a los amigos que he ido haciendo durante estos años en la facultad. He conocido a grandes personas como Arturo, Adri, Álvaro, Iker, Maikel, Pablo, o David entre muchos. Muchas gracias, mastodontes.

Pedro Martínez Gamero

Capítulo 1

Introducción: antecedentes, objetivos y plan de trabajo

1.1. Motivación del proyecto

Como estudiantes somos conscientes de que los cuestionarios en línea son una valiosa herramienta que nos ayudan, tanto a la hora de estudiar el contenido de una asignatura, permitiéndonos comprobar si lo hemos comprendido, o como método de evaluación usado por el profesor. Además, durante estos últimos años, debido a la pandemia provocada por el COVID-19, no hemos podido asistir a las aulas con una total normalidad, por lo que estos cuestionarios han sido incluso más importantes de lo que eran anteriormente, ya que muchos profesores comenzaron a usarlos para evaluar a sus alumnos. Algunas de las características más importantes de los cuestionarios en línea son las siguientes:

- Corrección automática: Esto permite que, tanto el alumno como el profesor, obtengan automáticamente las calificaciones. Esto es positivo, ya que en el caso del alumno podrá saber en qué parte del temario tendrá que esforzarse más, y en el caso del profesor se le facilitará la labor de corrección. Además, un alumno obtiene retroalimentación inmediata.

- Aleatorización de las preguntas y respuestas: Gracias a estos mecanismos el profesor puede dificultar los intentos de copia.

- Ahorro de papel: Debido a que los cuestionarios se realizan desde dispositivos electrónicos, el uso del papel se reduce, ya que solo sería necesario para aquellos alumnos que no dispusieran de un dispositivo electrónico o una conexión a internet.

Durante nuestra vida universitaria hemos usado la plataforma Moodle utilizada en el Campus Virtual de la UCM, la cual ha servido de inspiración para la realización de este proyecto. Esta plataforma permite a un profesor la creación de cuestionarios con distintos tipos de preguntas, como por ejemplo preguntas tipo test, multielección, o de respuesta corta entre otras. Además la aplicación permite que el profesor disponga de un *banco de preguntas* y también ofrece sistemas de aleatorización del orden de las preguntas.

Sin embargo esta aplicación puede presentar algunos problemas, los cuales hemos sufrido. Uno de los mayores inconvenientes que presenta es la necesidad de disponer de una conexión a internet durante todo el tiempo en el que se realiza el cuestionario. Esto es un problema, ya que no siempre es posible tener esa conexión durante todo ese tiempo. Algunas de las razones por las que no es posible son las siguientes:

- Inestabilidad de la conexión a internet: Este problema se da principalmente durante la realización de aquellos cuestionarios que se llevan a cabo de forma presencial. Esto es debido al gran número de alumnos que asisten a las clases, los cuales hacen que la red *WI-FI* se sature, provocando que, en muchos casos, esta no funcione correctamente. Además en nuestra facultad (Facultad de informática de la UCM) la cobertura no es buena en la mayoría de aulas, por lo que tampoco sería posible la realización de un cuestionario mediante la red de datos móviles.

- Sobrecarga de los servidores del Campus Virtual: Durante el proceso de cambio a la modalidad de enseñanza *online*, y especialmente al comienzo de este, los servidores no estaban preparados para que tantos usuarios hicieran uso de esta herramienta simultáneamente. Esto provocaba que en muchos casos no se pudieran realizar cuestionarios con normalidad.

La existencia de estos problemas de conexión acarrean una serie de desventajas sobre los cuestionarios en línea. El primero de ellos es la ansiedad que estos problemas pueden acarrear en el alumno, ya que tendrá la incertidumbre de saber si podrá realizar o no el test, y además estará nervioso debido al tiempo que pierde mientras se solucionan estos problemas. En segundo lugar, el profesor también tendrá que prever qué hacer en caso de que la conexión falle. Una de las soluciones podría ser imprimir copias del cuestionario para aquellos alumnos a los que les falle la conexión a internet, pero tendría que llevar tantas copias como alumnos haya, ya que no puede saber a cuántos les puede llegar a fallar la conexión a internet.

Una solución a estos problemas sería hacer uso de una ***aplicación web progresiva*** (***Progressive Web Application - PWA***) [2]. Una aplicación web progresiva es una página web que permite hacer uso de ella sin que el dispositivo tenga conexión a internet y desde el propio navegador web. Esto hace que no haya que instalarlas manualmente. Algunas de las PWA más conocidas son Google Docs, Google maps o Telegram.

1.2. Objetivos

Una vez analizado el problema y encontrada una posible solución, decidimos que el objetivo del proyecto sería desarrollar una aplicación web progresiva que permitiera que los alumnos pudieran realizar los cuestionarios sin tener que preocuparse de tener una conexión estable a internet, y que, los profesores a su vez, no tuvieran que preocuparse por los posibles problemas que conllevaría la pérdida de la conexión a internet por parte de uno o más alumnos. El funcionamiento de esta aplicación se basaría en los siguintes principios:

- Los estudiantes pueden descargarse un cuestionario desde el momento en el que un profesor lo crea. Estos cuestionarios han de estar encriptados, ya que si fuera de otra forma un alumno podría ver el cuestionario antes de hacerlo. Estos cuestionarios tienen una fecha de inicio y una fecha de fin, la cual puede ser vista por el alumno.
- En el momento de comenzar el cuestionario, el profesor compartirá con los alumnos la clave para poder desencriptar el cuestionario, y si se encuentran dentro del plazo marcado por la fecha de inicio y fin del cuestionario, podrán realizarlo. Las respuestas se irán guardando en el navegador por si el alumno cierra por error la aplicación. Además, estos cuestionarios contarán con preguntas de dos tipos:
  - Preguntas tipo "test", que muestran varias opciones, entre las cuales habrá que seleccionar una.
  - Preguntas tipo "texto", las cuales tendrán que ser respondidas escribiendo la respuesta en un cuadro de texto.
- En caso de que el alumno decida acabar el cuestionario o se acabe el tiempo las respuestas se intentarán enviar al servidor para que estas sean corregidas automáticamente y almacenadas en la base de datos. En caso de no disponer de una conexión a la red, aparecerá en la pantalla del alumno un *código QR*, el cual contendrá un identificador de las respuestas que ha dado el alumno. Esta huella se almacenará en la

base de datos y las respuestas, que aún permanecerían en el dispositivo del alumno, se intentarían enviar cuando el alumno recuperase la conexión a internet.

1.3. Plan de trabajo

Debido a que el proyecto hace uso de varias tecnologías que nunca habíamos usado previamente (React, Django, PWAs...) hemos atravesado varias etapas de aprendizaje y desarrollo. Además este plan giraba alrededor de las reuniones que manteníamos con nuestro tutor Manuel Montenegro, las cuales se realizaban cada dos semanas. La distribución temporal de las tareas fue la siguiente:

1.3.1. Fase 1: Familiarización con las tecnologías

Tal y se ha mencionado anteriormente este proyecto se basa en tecnologías nuevas para nosotros. A lo largo de esta etapa realizamos una labor de investigación y formación, en la cual nos familiarizamos con las herramientas y tecnologías necesarias.

1.3.2. Fase 2: Implementación de una prueba de concepto

En esta fase implementamos una prueba de concepto de la aplicación que cubriera la realización de un cuestionario. Esta prueba consistía en realizar el envío de un cuestionario desde el *back-end* al *front-end*. El cuestionario se cifraba en el *back-end* y era descifrado en el *front-end* (más información en las secciones 4.3 y 5.3.4). Una vez finalizada esta prueba de concepto, ya teníamos la funcionalidad más básica de la aplicación finalizada.

1.3.3. Fase 3: Creación de la base de datos y manejo de sesiones

En esta fase realizamos un modelo entidad-relación de la base de datos. También tuvimos que realizar una labor de investigación a lo largo de esta fase, ya que tuvimos que familiarizarnos con el *sistema de modelos de Django*, usado para gestionar la base de datos. Por último también implementamos el sistema de sesiones que usaría la aplicación web. Todo esto está explicado en la sección 4.2.

1.3.4. Fase 4: Implementación de las asignaturas y cuestionarios

A lo largo de esta fase se hizo la implementación de los apartados de la aplicación que hacían uso de las asignaturas y cuestionarios. Las llamadas a la API referentes a esto se pueden consultar en la sección 4.3. También se pueden ver los componentes de React relativos a esto en las secciones 5.3.2, 5.3.5.3 y 5.3.5.4. Además la base de datos sufrió modificaciones debido a las limitaciones que presentaba el primer *modelo entidad relación* que creamos. Esto último se recoge en la sección 4.2.

1.3.5. Fase 5: Corrección automática de los cuestionarios

En esta fase implementamos las funciones necesarias para que los cuestionarios que se recibían en el servidor fueran corregidos automáticamente y se almacenasen en la base de datos tanto las notas como las respuestas dadas por el alumno. Esto queda recogido tanto en las secciones 4.3.2, 4.3.3, 4.3.16 del capítulo 4 y en las secciones 5.3.3, 5.3.4.2, 5.3.5.2 del capítulo 5.

1.3.6. Fase 6: PWA y Service Workers

Esta fue una de las fases clave del desarrollo, puesto que en ella se implementó aquello que hacía a nuestra aplicación diferente de *Moodle*. Se implementaron los *Service Workers* necesarios para que nuestra aplicación fuera una PWA. Esto queda recogido en la sección 6.3.2.

1.3.7. Fase 7: Generación de Códigos QR

Esta fue la última fase de desarrollo del proyecto. En ella se implementó el sistema por el cual se generan los *códigos QR* en caso de que el alumno no tenga conexión a internet y el sistema con el cual estos códigos son escaneados y se almacenan la huellas de las respuestas del alumno en la base de datos. Esto queda recogido en las secciones 4.3.21, 4.3.22, 6.3.3.

1.3.8. Tareas llevadas a cabo a lo largo de todas las fases

A lo largo de todas las fases se llevó a cabo un intercambio de opiniones con nuestro tutor, permitiéndonos así mejorar diversos aspectos de la aplicación, como por ejemplo la interfaz de usuario, la cual fue variando en función de todas las fases de desarrollo.

Capítulo 2

Introduction

2.1. Background and motivation

As students we are aware that online quizzes are a valuable tool that help us study the contents of a subject, allowing us to check if we have understood it, and as an evaluation method used by the teacher. In addition, in recent years, due to the pandemic caused by the COVID-19, we have not been able to attend the classrooms with total normality, so these questionnaires have been even more important than they were previously, as many teachers began to use them to evaluate their students. Some of the most important features of the online quizzes are the following:

- Automatic correction: This allows both the student and the teacher to obtain automatically their califications. In the case of the students, they will be able to know in which part of the syllabus they have to put more effort. And in the case of teacher, it will facilitate the task of correction.
- Randomization of questions and answers: Thanks to these mechanisms, the teacher can prevent the students from cheating.
- Ease of changing an answer: For students it will be more comfortable changing an answer, since they won't have to cross it out.
- Paper saving: Due to the fact that questionnaires are carried out from electronic

devices, the use of paper is reduced, since it would only be necessary for those students who did not have an electronic device or an *Internet connection*.

During our university life we have used Moodle, which is a platform used in the Virtual Campus of the UCM. It has served as inspiration for the realization of this project. This platform allows a teacher to create questionnaires with different question types, such as multiple choice, or short answer questions among others. In addition, the application allows the teacher to have a question bank and it also offers randomization for the order of the questions.

However, this application can present some problems, which we have suffered. One of the biggest drawbacks it presents is the need to have a internet connection during the whole time in which the questionnaire is being resolved. This is a problem, since it is not always possible to have that connection during all that time. These are some of the reasons why it is not possible:

- Inestability of the internet connection: This problem occurs mainly during the realization of those questionnaires that are carried out in the university. This is due to the large number of students who attend the classes, which saturates the Wi-Fi network, causing it to not work properly in many cases. In addition, in our faculty (Faculty of Computer Science of the UCM) the coverage is not good in most classrooms, so it would not be possible to carry out a quiz using the mobile data network.
- Overload of Virtual Campus servers: During the change process to the online teaching modality, and especially at the beginning of this, the servers were not prepared for so many users to make use of this tool simultaneously. This meant that in many cases it was not possible to carry out quizzes normally.

The existence of these connection problems entail a series of disadvantages over the online quizzes. The first of these is the anxiety that these problems can cause in the students, since

they will have the uncertainty of knowing if they will be able to carry out or not the test. They will also be nervous due to the time they lose while solving these problems. Secondly, the teacher will also have to foresee what to do in case of the connection fails. One of the solutions could be to print copies of the quiz for those students who lost the internet connection. But the teachers would have to carry as many copies as the number of students that there are, since they can't know how many people would loose the internet connection.

A solution to these problems would be to make use of a progressive web app (Progressive Web Application - PWA) [2]. A progressive web app is a web page that allows one to use it without an internet connection and from the web browser itself. Another advantage is that it installs automatically. Some of the best known PWAs are Google Docs, Google maps or Telegram.

2.2. Goals

After analyzing the problem and finding a possible solution, we decided that the objective of the project would be to develop a progressive web application that would allow students to solve the quizzes without having to worry about having a stable internet connection, and that, in turn, the teachers will not have to worry about the possible problems that the loss of the internet connection can produce. The operation of this application would be based on the following principles:

- Students can download a quiz from the moment that a teacher creates it. These quizzes must be encrypted, since if they were not, a student could see the quiz before doing it. These questionnaires might have a start date and a end date, which can be seen by the student.
- The teacher will share with the students the key to decrypt the quiz, and if they are within the period marked by the start and end date of the questionnaire they

will be able to solve them. The answers will be saved in the browser in case the student accidentally closes the application. In addition, these questionnaires will count with two types of questions:

  - "Test" type questions, they show several options, among which it will be necessary to select one.
  - Text" type questions, which will have to be answered by writing the answer.

- In case the student decides to finish the quiz or it runs out of time the answers will be sent to the server so that they are corrected automatically and stored in the database. If the user do not have an internet connection, a QR code will appear on the screen, which will contain an identifier of the answers given by the student. This identifier will be stored in the database. The responses, which would still remain on the student device, will be sent when they recover the internet connection.

2.3. Working plan

Because the project makes use of several technologies that we had never used previously (React, Django, PWAs...) we have gone through several stages of learning about theese technologies. Furthermore, this plan evolved around the meetings we held with our tutor Manuel Montenegro, which were held every two weeks. The distribution task schedule was as follows:

2.3.1. Phase 1: Familiarization with the technologies

As mentioned above, this project is based on new technologies for us. Throughout this stage we carried out a research and training work, in which we becomed familiar with the necessary tools and technologies for this project.

2.3.2. Phase 2: Implementation of a proof of concept

In this phase we implemented a proof of concept of the application that covered the completion of a questionnaire. This test consisted of sending a questionnaire from the *back-end* to the *front-end*. The questionnaire was encrypted at the *back-end* and decrypted at the *front-end* (more information in sections 4.3 and 5.3.4). After this test was completed, we had achieved the most basic functionality of the application.

2.3.3. Phase 3: Database creation and session management

In this phase we made an entity-relationship model of the database. We also had to carry out investigation work throughout this phase, since we had to familiarize with *Django's model system*, used to manage the database. Finally, we also implemented the session system that the web application would use. All of this is explained in section 4.2.

2.3.4. Phase 4: Implementation of the subjects and quizzes

Throughout this phase, we implemented the sections of the web application that make use of the subjects and questionnaires. API calls relating to this can be found in section 4.3. You can also see the related React components in sections 5.3.2, 5.3.5.3 and 5.3.5.4. In addition, the database suffered some modifications due to the limitations of the first entity-relationship model we have created. There is more information in the section 4.2.

2.3.5. Phase 6: PWA and Service Workers

This was one of the key phases of the development, since here we implemented what made our application stand out from *Moodle*. The necessary service workers were implemented for our application to be a *PWA*. This is covered in section 6.3.2.

2.3.6. Phase 7: QR Code generation

This was the last phase of development in the project. Along it, we implemented the functions necessary to generate QR Codes when a student does not have internet connection.

We also implemented the functions to store the identifier of the answers in the database. This is covered in sections 4.3.21, 4.3.22, 6.3.3.

2.3.7. Tasks carried out throughout all phases

Along all the phases an exchange of opinions was carried out with our tutor, allowing us to improve various aspects of the application, such as the user interface, which was changing in all the development phases.

Capítulo 3

Metodología y tecnologías usadas

A lo largo de este apartado explicaremos tanto las metodologías como las tecnologías usadas para desarrollar el proyecto.

3.1. Metodología usada

A la hora de desarrollar el proyecto decidimos usar una metodología de desarrollo ágil similar a la seguida por el marco de trabajo *Scrum*. Decimos seleccionar esta metodología debido a que nos permitía desarrollar el proyecto de forma escalonada, agregándole más módulos conforme el mismo avanzaba. A esto hay que sumarle que inicialmente no conocíamos algunas de las tecnologías que más tarde usaríamos, y esta metodología nos permitía aumentar la dificultad de las tareas conforme se completaban los hitos y aumentaba nuestro conocimiento de las tecnologías involucradas.

Por una parte, las reuniones con el director del proyecto (Manuel Montenegro) se realizaban cada dos semanas. En estas se comunicaba el avance del desarrollo de la aplicación, se nos indicaba cómo mejorar lo que teníamos, se discutían diferentes posibilidades de implementación de algunas funcionalidades y se establecían los objetivos a conseguir para la siguiente reunión. Por otro lado, las reuniones entre nosotros (autores del TFG) se realizaban al menos una vez a la semana. El objetivo de estas reuniones era hacer juntos algunas funcionalidades o, en los casos donde nos repartíamos las tareas, para comunicarnos nuestro

avance y ayudarnos en caso de que alguno necesitase ayuda.

3.2. Tecnologías, lenguajes y frameworks

Durante el desarrollo de este proyecto hemos hecho uso de un gran número de tecnologías. La mayoría pertenecen al ámbito de las aplicaciones web. A lo largo de esta sección enumeraremos aquellas tecnologías que consideramos primordiales para poder llevar a cabo este proyecto.

3.2.1. Tecnologías utilizadas

A lo largo de esta sección haremos un repaso de las tecnologías más importantes que fueron utilizadas en el proyecto.

3.2.1.1. JavaScript

Javascript [12] es un lenguaje de programación que es usado como base por la biblioteca *React*, en la cual está basado el *front-end* del proyecto. También es la base de los *Service Workers* que son uno de los componentes esenciales de las aplicaciones web progresivas. Además este lenguaje también ha sido usado para la creación de pequeños elementos que forman parte de la interfaz de la aplicación web, como por ejemplo los contadores de tiempo que se muestran a la hora de realizar un test. Otro de los componentes esenciales de las aplicaciones web progresivas es el *Local Storage*. El *Local Storage* es una funcionalidad de *Javascript* que permite almacenar datos en pares clave-valor en el navegador del cliente. Entre los diferentes datos que se han almacenado en el *Local Storage*, están los tests que se haya descargado el usuario, lo cual le permite realizarlos de forma offline, y también los datos de sesión, que son necesarios para poder identificar al usuario. Otra de las funcionalidades que ofrece *Javascript* es `fetch()`, que es una función que permite realizar peticiones HTTP sin necesidad de instalar ninguna librería adicional.

3.2.1.2. React

React [11] es una biblioteca de *Javascript* usada para desarrollar el *front-end* de la *aplicación web*. Decidimos seleccionar esta biblioteca debido a la facilidad con la que se podía crear una *Single-page application*, es decir, una aplicación web en la que solo hace uso de una única página, haciendo que esta se asimile a una aplicación de escritorio. Además seleccionamos esta tecnología debido a la novedad que la misma representaba para nosotros, ya que ninguno de los dos había usado nunca esta biblioteca. Otra de las ventajas que ofrece React es la posibilidad de reutilizar código gracias a los *componentes*, que son partes de la vista de la aplicación que tienen una lógica. Los componentes se pueden definir mediante clases o funciones. En nuestro caso los hemos definido como clases porque nos era más familiar. En estas clases hay dos elementos importantes. El primer elemento es el contructor, este recibe las *props*, que son los párametros que le haya pasado otro componente, e inicializa el estado del componente. El segundo elemento es el método `render()`. Aquí se escribe el código JSX que se muestra al renderizar el componente. JSX es código que mezcla sintaxis de HTML con *JavaScript*. Los navegadores no entienden código JSX, por lo que existe un compilador llamado *Babel* que convierte ese código a *JavaScript*. Babel no solo traduce código JSX a *JavaScript*, también traduce las nuevas versiones de *JavaScript*, como ECMAScript 2015, a *JavaScript* que es compatible con los navegadores. Este compilador viene configurado al crear una aplicación de React con el comando *npx create-react-app* al igual que Webpack. Webpack se encarga de juntar en un único archivo *JavaScript* todos aquellos módulos que necesita la aplicación para poder funcionar.

3.2.1.3. Python

Python [14] es un lenguaje de programación usado como base por el *framework web Django*, en el cual está basado el *back-end* del proyecto. Seleccionamos este lenguaje debido a su facilidad de uso y su versatilidad.

3.2.1.4. Django

Django [6] [4] es un *framework* que usa el lenguaje de programación Python usado para desarrollar el *back-end* de la aplicación web. Decidimos usar este *framework* debido a que ofrece seguridad ante los ataques más comunes: inyecciones SQL, XSS, XSRF, entre otros y una autenticación de usuarios simple y robusta. Además, Django cuenta con un panel de administrador que facilita la gestión de la base de datos sin necesidad de tener que hacerlo a través de sentencias SQL. Otra característica de Django es su sistema de modelos, los cuales son un tipo especial de objetos que se guardan en la base de datos. Estos modelos nos permiten tanto crear las tablas de la base de datos como acceder posteriormente a ellas con facilidad. A la hora de crear un modelo tendremos que declarar los atributos que tendrá la tabla y otras características como, por ejemplo, la o las *claves primarias*, *claves externas* o *campos unique*. Por último, Django también ofrece la posibilidad de crear vistas, que son métodos que devuelven el contenido correspondiente a cada ruta de la API a la que se haya accedido.

3.2.1.5. Service Worker

El *Service Worker* [1] permite que una aplicación web sea progresiva. Es un script escrito en *JavaScript* que se ejecuta en segundo plano, es decir, aunque se cierre la aplicación web, este se sigue ejecutando. Cuando el usuario visita la página web por primera vez, el *Service Worker* se instala. Una vez instalado, este es capaz de interceptar las peticiones a internet que hace el cliente, es decir, actúa como un *Proxy* entre el cliente e internet.

La funcionalidad principal del *Service Worker* es el *cacheo*, es decir guardar recursos en la memoria caché del navegador web. Por ello lo primero que hace al instalarse por primera vez, es cachear todos los archivos estáticos esenciales de la aplicación web. Esto es conocido como *pre-cache*. Los archivos estáticos suelen ser archivos de CSS, *JavaScript*, HTML y también imágenes.

A la hora de programar un *Service Worker* tenemos que saber qué recursos queremos cachear y qué estrategia de cacheo vamos a utilizar. Existen un par de estrategias a la hora de cachear recursos. El uso de una u otra dependerá de cómo deseemos que se comporte la aplicación web progresiva. Algunas de estas estrategias son:

- Cache First: En esta estrategia lo primero que se hace es buscar si el recurso está en la caché. Si está lo devuelve; si no, lo busca en el servidor y lo devuelve a la vez que lo almacena en la caché. De esta manera la próxima vez que se busque dicho recurso, este estará en la caché. Esto es especialmente óptimo para conexiones lentas.

- Network First: Esta estrategia hace lo contrario a la *Cache First*. Primero se busca el recurso en el servidor, si hay conexión lo devuelve y lo cachea, si no, lo busca en la caché. Con esta estrategia, el cliente siempre tiene la última versión de la aplicación.

Cuando se realiza una petición al servidor existe la posibilidad de fallo. Esto puede ocurrir porque el servidor no esté disponible en ese momento o porque el cliente no tenga conexión a internet. Aquí entra otra de las funcionalidades que proporcionan los *Service Workers*, que es el *Background-Sync*. El *Background-Sync* permite que el *Service Worker* vuelva a enviar la petición. En nuestra aplicación web progresiva, se usa cuando el usuario realiza un test y lo envía, pero se encuentra sin conexión. El *Service Worker* detecta cuándo el cliente vuelve a tener conexión y vuelve a enviar la petición.

La caché disponible en el navegador es limitada. Por ello hay que gestionarla. Esta gestión suele ser complicada. Por ello hemos usado *Workbox* [7], que es una librería que se encarga de gestionar la caché, además de simplificar la programación de los *Service Workers*.

3.2.1.6. Bootstrap

Bootstrap [5] es una biblioteca CSS que hemos usado para dar estilo a toda la aplicación web. Decidimos usarla debido a que ya estábamos familiarizados con ella, ya que la habíamos usado previamente en asignaturas como *Aplicaciones Web*. Además, esta biblioteca nos permitía hacer que la aplicación fuera adaptable (*responsive*), es decir, que su interfaz se adaptase en función de las dimensiones del dispositivo en el que se visualizara la aplicación (teléfonos móviles o tablets), cosa que considerábamos primordial en una aplicación de este estilo.

3.2.1.7. JQuery

JQuery [8] es una biblioteca de JavaScript que hemos usado principalmente para actualizar distintos elementos del DOM sin tener que recargar la página. Por ejemplo, hemos usado esta librería para cambiar el contenido de algunos textos en función de qué queríamos mostrar, cosa que hubiera sido más engorrosa de hacer con React.

3.2.1.8. YAML

YAML [15] es un formato de serialización de datos, el cual se inspira en lenguajes como JSON o XML. Una de las ventajas que presenta este formato es que es bastante legible a simple vista, por lo que un usuario que no sea muy experto podría hacer uso de estos ficheros sin problema. Debido a esta razón, hemos decidido usarlo para la creación de cuestionarios y preguntas en ficheros y también para la descarga de preguntas, ya que es fácil de entender para los profesores que no tengan conocimientos informáticos.

```
1    lenguajes:
2      - nombre: "JavaScript"
3        fecha_creacion: "Diciembre de 1995"
4      - nombre: "Python"
5        fecha_creacion: "Diciembre de 1989"
6
```

Ejemplo de fichero YAML

3.2.1.9. JSON

JSON [9] es un formato de fichero de texto utilizado para el intercambio de datos. Se ha utilizado este formato ya que resulta muy práctico para intercambiar información a través de la API. Su sintaxis es sencilla, lo que facilita la decodificación y codificación de los datos. Además, el formato JSON es compatible con cualquier lenguaje de programación.

3.2.1.10. SQL

SQL [13] es lenguaje de programación usado para acceder a las bases de datos. Algunas de las consultas realizadas a la base de datos están hechas usando este lenguaje, ya que en ocasiones las consultas realizadas mediante los modelos de *Django* se traducían en varias consultas SQL, mientras que con *SQL* puro, podíamos obtener toda la información necesaria a partir de una única consulta. Además, gracias al uso de SQL hemos solucionado el *problema de las n+1 consultas*.

3.2.1.11. Laragon

Laragon es un entorno de desarrollo que se ha usado para iniciar el servidor de la base de datos en la máquina local. Lo hemos elegido debido a que es más moderno, ligero y rápido comparado con XAMPP. Además, ofrece una interfaz gráfica que es muy fácil de usar.

3.2.1.12. Visual Studio Code

Visual Studio Code es el IDE con el que hemos desarrollado nuestro TFG. Decidimos usarlo debido a la versatilidad del mismo. Hemos utilizado las siguientes extensiones:

- Simple React Snippets: Simple React Snippets permite crear entre otros el código de una clase y una función de React con atajos. Gracias a esta extensión podemos escribir código repetitivo rápidamente.
- GitHub: Permite realizar operaciones de Git como *commit, push, etc* con Visual Studio Code, así como mostrar con un color distinto los ficheros que se han modificado

y añadido desde el último *commit* que se haya realizado.

- Python: Permite, entre muchas otras cosas, depurar y formatear código escrito en Python.

3.2.1.13. Git

Como sistema de control de versiones hemos usado Git. Esto nos ha permitido trabajar sin tener conflictos a la hora de trabajar, separando el desarrollo en distintas ramas que cada uno usábamos para avanzar en determinados módulos de la aplicación. Como repositorio hemos hecho uso de Github.

3.2.1.14. LaTeX

Como herramienta para la elaboración de la memoria del TFG hemos usado LaTeX. Lo hemos utilizado porque a diferencia de Word permite generar documentos con una apariencia profesional sin mucho esfuerzo. [10]

3.2.2. Tecnologías descartadas

En esta sección se mencionarán las tecnologías que han sido descartadas.

3.2.2.1. Redux

Es una librería para gestionar el estado de React. Cada componente de React tiene un estado, por lo tanto el estado de la aplicación esta distribuido entre sus componentes. Algunos componentes necesitan la información del estado de otros componentes. Para ello esta información se debe pasar a dichos componentes para que estos puedan usarla. Un componente puede estar compuesto por varios componentes hijos que a su vez estos pueden tener otros componentes hijos. Por ejemplo si queremos pasar la información del estado del componente padre al componente nieto, esta información primero se tiene que pasar del padre al hijo, y después del hijo al nieto. En una aplicación grande, pasar esta información de estado a través de varios componentes complicaría el escalado y mantenimiento de la

aplicación. Por ello aquí entra Redux, que gestiona toda la información de estado en un único lugar accesible a todos los componentes. De esta manera no hay necesidad de que los componentes se pasen información de estado entre sí. Sin embargo usar Redux en nuestra aplicación web no era necesario ya que no es de un tamaño muy grande. Además, usarlo solo añadiría complejidad innecesaria a la aplicación.

Capítulo 4

Descripción del back-end y la base de datos

A lo largo de este apartado explicaremos tanto el funcionamiento del *back-end* como la estructura de la base de datos. El *back-end* es el componente del proyecto que permite la comunicación entre el *front-end* y la *base de datos*. Es el encargado de realizar varias operaciones, como por ejemplo el *login* de los usuarios, o las distintas verificaciones a la hora de insertar elementos en la *base de datos*, como por ejemplo, que únicamente un profesor que imparta cierta asignatura pueda insertar cuestionarios para la misma. La base de datos se encarga de almacenar la información necesaria para que la aplicación pueda funcionar, como por ejemplo, los usuarios, las asignaturas y los cuestionarios.

4.1. Arquitectura global del sistema

Nuestra aplicación está compuesta por un *front-end* y un *back-end*. El *front-end* es la parte que ve el usuario, es decir la interfaz gráfica de la aplicación. El *back-end* es la parte que no ve el usuario. En esta reside la lógica del negocio y se encarga tanto de almacenar datos como devolverlos y ofrecer mecanismos de seguridad.

4.2. Base de datos

Para implementar la base de datos decidimos usar el sistema de gestión de base de datos MySQL. Decidimos usar esta tecnología debido a que nos pareció lo suficientemente potente como para poder cumplir con nuestras necesidades, y además ya teníamos experiencia con su uso. También la escogimos debido a que, tal y como indica su nombre, hace uso del lenguaje SQL, el cual hemos usado a lo largo de la carrera y que, en el caso de Pedro, lo ha estado usando durante 5 meses en sus prácticas en empresas.

También cabe destacar que Django ofrece un tipo especial de objeto llamado *modelo*. Estos objetos permiten implementar con facilidad la base de datos, ya que a través de ellos podemos declarar los propios modelos de la base de datos. También nos permiten declarar características como *foreign keys*, el tipo del campo o los campos *unique* del modelo.
A continuación se muestra un modelo de Django utilizado para representar cuestionarios:

```
class Cuestionarios(models.Model):
    titulo = models.CharField(blank=True, max_length=100, verbose_name='titulo')
    idProfesor = models.ForeignKey(User, on_delete=models.CASCADE)
    idAsignatura = models.ForeignKey('Asignaturas', on_delete=models.CASCADE)
    duracion = models.IntegerField(default=10, verbose_name='duracion')
    secuencial =  models.IntegerField(default=1, verbose_name='secuencial')
    password = models.CharField(blank=True, max_length=300, verbose_name='password')
    fecha_apertura = models.DateTimeField(blank=False, verbose_name='fecha_apertura')
    fecha_cierre = models.DateTimeField(blank=False,  verbose_name='fecha_cierre')

    def __str__(self):
        return self.titulo

    class Meta:
        ordering = ['titulo']
        db_table = "cuestionarios"
        unique_together = ('idAsignatura', 'titulo')
        #No puede haber dos cuestionarios con el mismo nombre para una asignatura

```

La base de datos ha sufrido varias modificaciones a lo largo del desarrollo del proyecto. En un comienzo el modelo entidad está representado en la Figura 4.1:

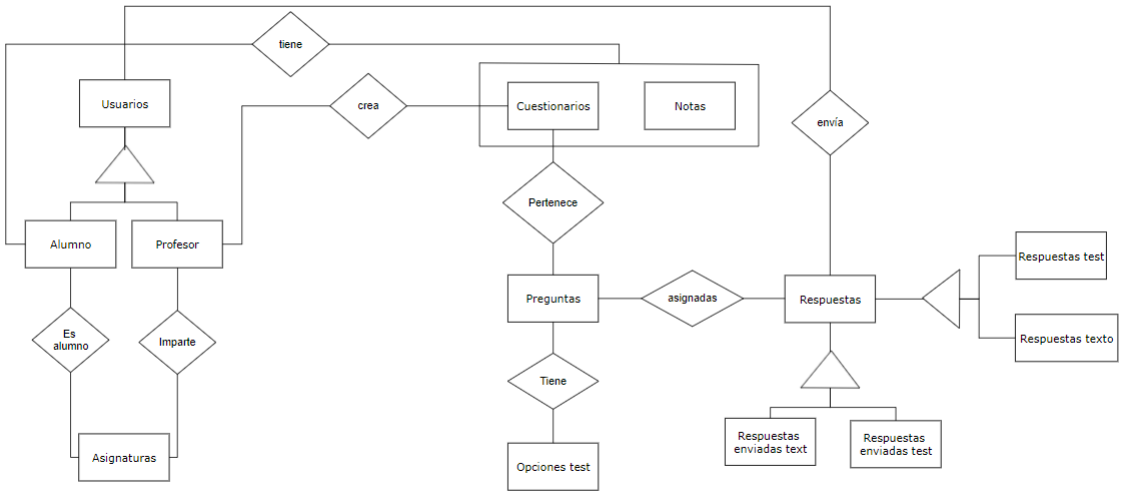

Figura 4.1: *Primer modelo entidad relación de la base de datos.*

Conforme el proyecto fue avanzando nos dimos cuenta de que este modelo contaba con ciertas limitaciones. Por ejemplo, a la hora de implementar la función de *envíos o ine* decidimos crear un nuevo modelo, el cual almacenaría los *hashes* usados para verificar que un alumno no ha alterado las respuestas a la hora de enviarlas. Además de crear nuevos modelos, también tuvimos que añadir atributos a los modelos que ya existían.

4.2.1. Banco de preguntas

La aplicación cuenta con bancos de preguntas. Estos bancos son únicos por asignatura, y facilitan la labor del profesor a la hora de generar cuestionarios, ya que no tiene que añadir preguntas que ya haya incluido en otros cuestionarios anteriores. Para poder lograr esto decidimos establecer relaciones entre distintas tablas de la base de datos que permitieran esto.

4.2.2. Modelo relacional de la base de datos

A continuación se muestra el **modelo relacional** generado a partir de la base de datos. (ver Figura 4.2). El contenido de las tablas que se muestran en este modelo se explicará en la siguiente sección.

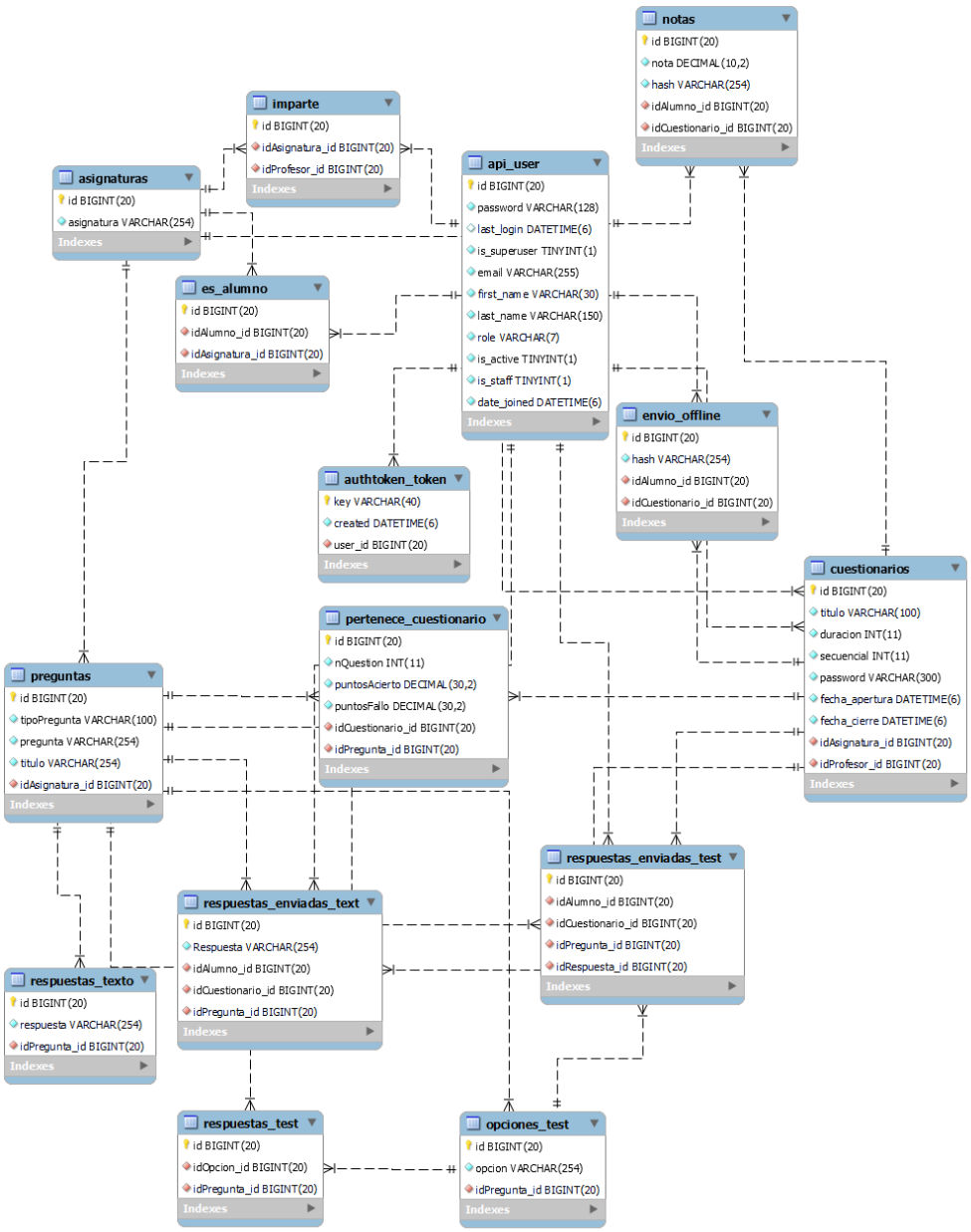

Figura 4.2: *Modelo relacional*

4.2.3. Tablas de la base de datos

Todas las tablas de la base de datos contienen un campo *id*, el cual es usado para identificar cada fila de la tabla. Para simplificar el listado no lo mencionaremos en ninguna de descripciones de las tablas que vienen a continuación.

4.2.3.1. *api_user*

Esta tabla es usada para la *gestión de usuarios* dentro de la aplicación. La tabla está basada en un modelo de *Django*, al cual le añadimos atributos, como por ejemplo el rol del usuario, el cual nos permite diferenciar entre estudiantes, profesores y administradores. Contiene atributos como el nombre y los apellidos, el correo electrónico y la contraseña del usuario, entre otros. Además, esta tabla es la usada por *Django* para autenticar a los usuarios. Para llevar a cabo esta acción inicialmente se hacía uso de los campos *username* y *password*, pero la modificamos para que hiciera uso del campo *email* en vez del *username*.

4.2.3.2. *asignaturas*

Esta tabla es la encargada de almacenar las asignaturas. Solo consta de un atributo, que es el nombre de la asignatura.

4.2.3.3. *imparte*

Esta tabla nos permite relacionar un usuario con una asignatura, estableciendo que el usuario es *profesor de esa asignatura*. Consta de dos atributos principales, los cuales son *foreign keys*. Uno de ellos apunta a la tabla *api_user* y el otro a la tabla *asignaturas*.

4.2.3.4. *es_alumno*

Tabla similar a la anterior. Permite relacionar un usuario con una asignatura, estableciendo que el usuario es *alumno de esa asignatura*. Consta de dos atributos principales, los cuales son *foreign keys*. Uno de ellos apunta a la tabla *api_user* y el otro a la tabla *asignaturas*.

4.2.3.5. *cuestionarios*

Esta tabla es la encargada de almacenar información sobre los cuestionarios, como por ejemplo, el título de los mismos o su duración. Entre sus atributos más destacados tenemos el título, la duración, la contraseña del mismo, su fecha de apertura, su fecha de cierre, una *foreign key* que apunta a la tabla *asignaturas*, la cual nos permite identificar a qué asignatura pertenece el cuestionario, y otra *foreign key* que apunta a la tabla *api_user*, siendo este usuario el *profesor* que creó el cuestionario.

4.2.3.6. *notas*

Esta tabla es la encargada de almacenar las notas de los alumnos. Contiene un atributo encargado de guardar la nota, otro encargado de almacenar el *hash* de las respuestas que han llegado al servidor por parte del alumno (nos es útil para comprobar que no se hayan modificado las respuestas del alumno) y dos *foreign keys*. Una de ellas apunta a la tabla *api_user*, que será el alumno al que le pertenece la nota, y la otra hace referencia al cuestionario al que pertenece la nota.

4.2.3.7. *preguntas*

Esta tabla es la encargada de almacenar las preguntas de todos los cuestionarios. Debido a que la aplicación contiene un banco de preguntas, este modelo no contiene ninguna *foreign key* que haga referencia a un cuestionario. Como atributos contiene la propia pregunta, un título para la misma, el tipo de la pregunta y una *foreign key* que hace referencia a la asignatura a la que pertenece la pregunta.

4.2.3.8. *pertenece_cuestionario*

Esta tabla es la encargada de establecer una relación entre una pregunta y un cuestionario, estableciendo la relación *"una pregunta pertenece a uno (o varios) cuestionarios"*. Esta relación nos permite además que una misma pregunta tenga distintas puntuaciones en función del cuestionario. Como atributos cuenta con el número de pregunta,

los puntos que sumará al cuestionario en caso de que la pregunta se acierte y los puntos que restará al cuestionario en caso de que la pregunta se falle, una *foreign key* que referencia al cuestionario al que pertenece la pregunta, y otra *foreign key* que referencia a la propia pregunta.

4.2.3.9. *respuestas_texto*

Esta tabla es la encargada de almacenar las respuestas correctas para las preguntas del tipo *"texto"*. Cuenta con un atributo encargado de almacenar la respuesta como tal y con una *foreign key* que referencia a la pregunta a la cual pertenece esta respuesta.

4.2.3.10. *respuestas_test*

Esta tabla es la encargada de almacenar las respuestas correctas para las preguntas del tipo *"test"*. Cuenta con una *foreign key* que referencia a la opción correcta de la pregunta y otra *foreign key* que referencia a la pregunta a la cual pertenece esta respuesta.

4.2.3.11. *opciones_test*

Esta tabla es la encargada de almacenar las distintas opciones que tendrá una pregunta del tipo *"test"*. Cuenta con un atributo encargado de almacenar el texto de la opción y una *foreign key* que referencia a la pregunta a la que pertenece la opción.

4.2.3.12. *respuestas_enviadas_test*

Esta tabla es la encargada de almacenar las respuestas que ha enviado un alumno para una pregunta del tipo *"test"*. Como atributos cuenta con una *foreign key* que referencia a el alumno que ha mandado la respuesta, otra *foreign key* que referencia al cuestionario al que ha respondido, otra *foreign key* que referencia a la pregunta a la que se ha respondido y por último otra *foreign key* que referencia a la tabla *opciones_test* para indicar la opción que se ha marcado.

4.2.3.13. *respuestas_enviadas_text*

Esta tabla es la encargada de almacenar las respuestas que ha enviado un alumno para una pregunta del tipo *"texto"*. Es muy similar a la tabla anteriormente mencionada, con la diferencia de que en vez de contener una *foreign key* que referencia a la tabla *opciones_test* contiene un atributo que almacena la respuesta que ha dado el alumno.

4.2.3.14. *envio_offline*

Esta tabla es la encargada de almacenar los *hashes* de las respuestas originales de un alumno. Esta tabla es usada cuando un alumno realiza una entrega las respuestas más tarde de la fecha de finalización del test debido a problemas de conexión. Cuenta con un atributo encargado de almacenar el *hash* de las respuestas que ha dado el alumno durante la clase, una *foreign key* que referencia al alumno que ha enviado las respuestas, y otra *foreign key* que referencia al cuestionario al que pertenecen las respuestas.

4.3. API de comunicación entre back-end y front-end

Para poder desarrollar la aplicación tuvimos que crear una API que permitiera la comunicación entre el *front-end* y el *back-end*. Para ello creamos un conjunto de métodos, los cuales son llamados desde el *front-end* y que cumplen distintos propósitos, como por ejemplo la autenticación de los usuarios, la obtención de asignaturas que imparte un profesor o en las que está matriculado un alumno o la obtención de las notas los alumnos para un cuestionario determinado. Todos los métodos que explicarán a continuación son de tipo POST a excepción de los métodos *logout* y *get-subjects*, que son de tipo GET.

4.3.1. test

Este método es el encargado de, dado el identificador de un cuestionario, devolver el cuestionario encriptado. Lo devuelve encriptado debido a que este cuestionario se almacenará en el *local storage* del usuario, y este no debería poder acceder al test hasta que el profesor

compartiera la clave con él. Para encriptar el cuestionario usamos el algormito AES en el modo CFB. La información que devuelve es:

- Identificador del cuestionario.
- *Hash* de la contraseña usada para encriptar el cuestionario.
- Vector de inicialización usado para encriptar el cuestionario.
- Cuestionario encriptado: contiene las preguntas y las opciones para las preguntas del tipo *"test"*.
- Título del cuestionario.
- Duración del cuestionario.
- Fechas de apertura y cierre del cuestionario en dos formatos.

4.3.2. test-corrected

Este método es el encargado de, dado el identificador de un cuestionario, devolver el cuestionario corregido para que el usuario pueda revisarlo. La información que devuelve es:

- Preguntas del cuestionario.
- Respuesta dada por el alumno.
- Respuesta correcta.

4.3.3. response

Este método es usado para almacenar las respuestas que ha dado un usuario a un cuestionario en la base de datos. A su vez llama a otro método llamado *calcularNota*, el cual se encargará de corregir las respuestas que haya enviado el usuario y calcular la nota del mismo.

4.3.4. login

Este método es el encargado de autenticar un usuario. Recibe el correo y la contrseña hasheada del usuario. Para realizar la autenticación se utiliza un método de Django llamado *authenticate*. Una vez autenticado al usuario se genera un *token*, el cual será usado para identificar a un usuario con una sesión activa.
La información que devuelve el método es:

- Respuesta: Indica si el login se ha realizado con éxito o no.
- En caso de que el login se haya realizado correctamente, además se enviarán los siguientes datos:
  - Username: Es el correo electrónico del usuario que ha hecho login
  - Token: Token que permitirá al usuario hacer uso de las demás funciones de la API.
  - Rol: Rol del usuario.
  - Identificador del usuario.

4.3.5. logout

Este método es el encargado de cerrar la sesión de un usuario. Para ello hace uso de un método de Django llamado *logout*. Únicamente devuelve un mensaje indicando que la sesión se ha cerrado correctamente.

4.3.6. register

Este método es usado para registrar usuarios sin hacer uso de la consola de comandos de Django.
El método recibe los siguientes parámetros:

- Email del nuevo usuario.

- Nombre del nuevo usuario.
- Apellidos del nuevo usuario.
- Contraseña del usuario.
- Rol del nuevo usuario (puede ser “student”, “teacher” o “admin”).

Como respuesta envía un mensaje indicando que se ha registrado correctamente al usuario.

4.3.7. get-subjects

Este método sirve para obtener las asignaturas en las que, o bien el usuario actualmente identificado está matriculado (en el caso de que el usuario sea un alumno), o bien imparte (en el caso de que el usuario sea un profesor). El método no recibe ningún parámetro. Como resultado devuelve una lista con el nombre de la asignatura y el identificador de la misma.

4.3.8. get-all-subjects

Este método sirve para recuperar todas las asignaturas que hay dadas de alta dentro de la aplicación.

4.3.9. get-quizzes

Este método sirve para, dado el identificador de una asignatura, obtener todos los cuestionarios que están asociados a ella. Como entrada recibe únicamente el identificador de la asignatura.

El método devuelve dos listas, una con los nombres de los cuestionarios y otra con los identificadores de los cuestionarios.

4.3.10. get-subject-info

Este método sirve para obtener información relativa a una asignatura. Como parámetros recibe el identificador de la asignatura para la que se quiere obtener los datos.

Finalmente, el método devuelve la siguiente información:

- Número de cuestionarios de la asignatura.
- Número de cuestionarios corregidos para esa asignatura.
- Número de cuestionarios pendientes de realizar para esa asignatura.

4.3.11. get-quiz-info

Este método sirve para obtener información relativa a un cuestionario. Como parámetros recibe el identificador del cuestionario para el que se quiere obtener los datos.
Finalmente el método devuelve la siguiente información:

- Duración del cuestionario.
- Fecha de apertura del cuestionario en dos formatos. Uno de estos formatos es el predefinido por Django, y el otro es un formato que más tarde usamos para mostrar las fechas ("día mes año, hora:minuto:segundo").
- Fecha de cierre del cuestionario en dos formatos, que son los mismos que en el campo anterior.
- Indicación de si el cuestionario está corregido o no.
- Nota obtenida en el cuestionario.

4.3.12. get-subject-questions

Este método sirve para obtener aquellas preguntas que pertenecen a el banco de preguntas de una asignatura determinada. Como parámetros recibe el identificador de la asignatura.
Este método lo pueden usar únicamente los profesores y los administradores, ya que los alumnos no deben acceder al banco de preguntas de una asignatura.
El método devuelve una lista con todas las preguntas pertenecientes al banco de esa asignatura con la siguiente información de cada una de ellas:

- Identificador de la pregunta.
- Texto de la pregunta.
- Título de la pregunta.
- Tipo de pregunta.
- En caso de que la pregunta sea del tipo *"test"* se envía también una lista con las posibles opciones y la opción correcta.
- En caso de que la pregunta sea del tipo *"texto"* se envía la respuesta correcta.

4.3.13. upload

Método usado para guardar un cuestionario que ha sido subido a la aplicación mediante un fichero en formato YAML. El método recibe una cadena en formato JSON en el que se encuentra una cadena de caracteres con el contenido del fichero YAML. Esta cadena es procesada y las preguntas extraídas son almacenadas en la base de datos.

4.3.14. upload-questions

Método usado para guardar un conjunto de preguntas que han sido subidas a la aplicación mediante un fichero en formato YAML. El método recibe un JSON en el que se encuentra una cadena de caracteres con el contenido del fichero YAML, además del identificador de la asignatura a la que pertenecen las preguntas. Esta cadena es procesada y las preguntas se almacenan en la base de datos.

4.3.15. create-quiz

Este método es usado para crear los cuestionarios en la base de datos. El método solo puede ser usado por aquellos usuarios que sean administradores o profesores de esa asignatura.

El método recibe los siguientes datos como parámetros:

- Título del cuestionario.
- Contraseña del cuestionario.
- Identificador de la asignatura a la que pertenece el cuestionario.
- Indicador de si las preguntas del cuestionario se tendrán que mostrar de forma secuencial o no. En caso de no mostrarse de forma secuencial las preguntas deben mostrarse de forma aleatoria.
- Fecha de apertura del cuestionario.
- Fecha de cierre del cuestionario.
- Lista de preguntas pertenecientes al cuestionario.

4.3.16. get-quiz-grades

Este método es usado para obtener las notas de todos los alumnos matriculados una asignatura para un cuestionario determinado. Este método hace la consulta a través de SQL y no mediante el sistema de modelos que ofrece Django debido a que, mediante una consulta SQL, podíamos obtener los datos que necesitábamos con mayor facilidad que si lo hiciéramos con el sistema de modelos anteriormente mencionado. Ver Figura 4.3.

Un lado positivo de Django es que protege de ataques del tipo *SQL injection*. El método finalmente devuelve una lista con los siguientes datos por cada elemento de la lista:

- Identificador del alumno.
- Nombre del alumno.
- Apellidos del alumno.
- Nota del alumno (devuelve no presentado en caso de que no haya ninguna nota para ese alumno).

```
1     alumnos = User.objects.raw("SELECT u.id, u.first_name, u.last_name, n.
    nota AS nota " +
2         "FROM api_user AS u JOIN notas AS n ON u.id = n.idAlumno_id WHERE
    n.idCuestionario_id = %s AND n.idAlumno_id = %s UNION " +
3         "SELECT u.id, u.first_name, u.last_name, n.nota as nota FROM " +
4         "api_user AS u " +
5         "JOIN es_alumno AS alumn " +
6         "ON u.id = alumn.idAlumno_id " +
7         "left JOIN (SELECT * from notas WHERE idCuestionario_id = %s) AS n
     ON " +
8         "u.id = n.idAlumno_id "+
9         "WHERE alumn.idAsignatura_id = %s ", [str(request.data["
    idCuestionario"]), request.user.id, str(request.data["idCuestionario"
    ]), str(cuestionario.idAsignatura.id)], translations=name_map)
10
```

Figura 4.3: *Código usado para realizar la consulta con la que se obtienen las notas de todos los alumnos de una asignatura para un cuestionario determinado.*

4.3.17. delete-question

Método usado para eliminar una pregunta de la base de datos. Este método es únicamente accesible si el usuario es profesor o administrador. Como parámetro recibe el *identificador de la pregunta* que se desea eliminar, y devuelve un mensaje confirmando si se ha eliminado o no la pregunta.

4.3.18. update-question

Método usado para modificar una pregunta de la base de datos. Este método es únicamente accesible si el usuario es profesor o administrador. Como parámetro recibe el identificador de la pregunta y los campos que la componen. Tras la modificación se devuelve un mensaje confirmando si se ha actualizado o no la pregunta.

4.3.19. get-students

Método usado para obtener todos los alumnos que están matriculados en una asignatura. Este método es únicamente accesible si el usuario es profesor o administrador.
El método devuelve una lista con los alumnos, y por cada alumno se envía la siguiente

información:

- Identificador del alumno.
- Nombre del alumno.
- Apellidos del alumno.

4.3.20. enroll-students

Método usado para matricular a un número de alumnos (puede ser más de uno) en una asignatura. Este método es únicamente accesible si el usuario es profesor o administrador. Como parámetros de entrada el método recibe una lista con varios alumnos y el identificador de la asignatura en la que se desea matricular a los alumnos.
Como respuesta al *front-end* se envía la siguiente información:

- Una variable indicando si se han matriculado a todos los alumnos correctamente.
- Una lista con los alumnos que no se hayan matriculado correctamente en la asignatura (esta lista puede estar vacía).

4.3.21. insert-qr

Método usado para insertar el *hash* de las respuestas de un alumno generado por la aplicación en caso de que este tenga problemas de conexión. Este método es únicamente accesible si el usuario es profesor o administrador, ya que un alumno nunca debe poder guardar un *hash* de sus respuestas en la base de datos. Como parámetros el método recibe el identificador del alumno para el que se va a guardar el *hash*, el identificador del cuestionario al que pertenecen las respuestas del *hash*, y el propio *hash*.
Tras la ejecución del método se enviará un mensaje confirmando si se ha introducido el *hash* correctamente o si se ha producido un error en el proceso.

4.3.22. get-hashes

Método usado para obtener el *hash* generado mediante el código QR y el generado a la hora de subir las respuestas al servidor para un alumno en concreto. Este método es únicamente accesible si el usuario es profesor o administrador. Como parámetros recibe el identificador del cuestionario y el identificador del alumno para el que se quiere obtener los *hashes* de su respuesta. Finalmente el método devuelve ambos *hashes*.

4.4. Aspectos específicos de Django que se han utilizado

A lo largo de esta sección hablaremos de aquellos aspectos de Django que hayamos usado para desarrollar la aplicación.

4.4.1. Panel de administración de Django

Django ofrece un panel de administración desde el cual se pueden insertar elementos en la base de datos. Hemos decidido usar este panel de administración para que aquellos usuarios que tengan el rol de *"administrador"* puedan manejar la base de datos a través de una interfaz gráfica, ya que en caso de no haber usado este panel de control habría dos opciones:

- Que los administradores manejasen la base de datos mediante consultas SQL o algún programa que permitiese la inserción/modificación de filas en las distintas tablas de la base de datos.
- Desarrollar un apartado de la aplicación únicamente visible para los administradores que les permitiese hacer lo mismo que pueden hacer desde el propio panel de Django.

Decidimos descartar la primera opción debido a que nos parecía algo engorrosa desde el punto de vista del usuario, ya que no podemos asumir que todos los administradores tengan conocimientos sobre bases de datos. También decidimos descartar la segunda opción debido a

que supondría el gasto de un tiempo que decidimos invertir en otros aspectos de la aplicación. Ver Figura 4.4.

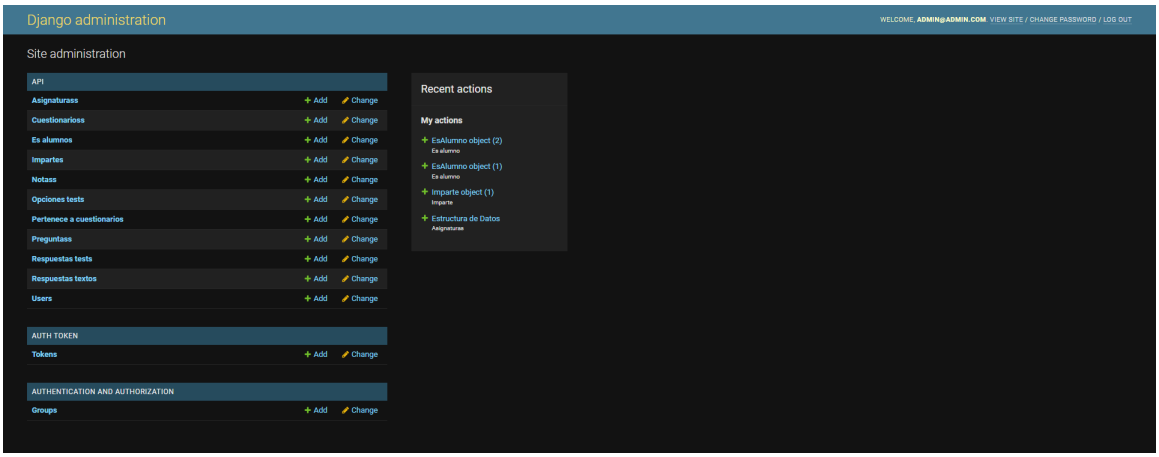

Figura 4.4: *Vista del panel de control de Django*

4.4.2. Librerías utilizadas

- **django-cors-headers**: Permite que se hagan peticiones de un servidor local a otro servidor en la máquina local. Lo que nos ha permitido hacer peticiones en la API para poder seguir probando y desarrollando la aplicación. Esta biblioteca ha sido fundamental para el desarrollo de la aplicación, ya que sin ella no habríamos tenido forma de comunicar React con Django.

- **djangorestframework**: Permite crear APIs de forma sencilla. Entre otras cosas, ofrece una interfaz gráfica para realizar las peticiones y también distintos métodos para autenticar a un usuario, como, por ejemplo, los *Tokens*, mecanismo que es usado por la aplicación. En un comienzo usamos el sistema de sesiones que ofrece Django, pero a la hora de implementarlo nos encontramos con una serie de dificultades e incompatibilidades con nuestra idea del proyecto. En primer lugar el sistema de sesiones de Django tenía una incompatibilidad con React, ya que React y Django funcionan en servidores independientes. Esto hacía que para que las sesiones funcionasen tuviéramos que compilar el proyecto cada vez que queríamos probar un cambio. Esta compilación

llevaba de media unos 30 segundos.

El segundo problema que encontramos es que las sesiones caducaban, y esto presentaría un problema, ya que el objetivo del proyecto es que un usuario pueda acceder a los cuestionarios que se encuentren en su *local storage* cuando no tenga conexión. Si la sesión caducaba a la hora de enviar las respuestas tras recuperar la conexión a internet estas serían rechazadas por el servidor, puesto que, la sesión ya habría expirado.

La solución a estos dos problemas fue hacer uso del sistema de autenticación mediante tokens de Django. Estos tokens son generados a la hora de hacer *login*, y son almacenados en el *local storage* del usuario. Cada vez que el usuario hace una petición al servidor se envía el token junto a ella, y de esta forma se autentica al usuario. Estos tokens caducan a los 30 días, tiempo más que suficiente para que un usuario lo renueve.

- PyCryptodome: Es una biblioteca que contiene distintos métodos para encriptar distinto contenido en Python. Esta biblioteca ha sido usada para encriptar los cuestionarios, añadiendo una capa de seguridad a la aplicación, ya que sin ella los alumnos podrían acceder a los cuestionarios antes de que comenzaran.
- MySqlClient: Es una biblioteca que permite conectar Django con una base de datos de este tipo.

Capítulo 5

Descripción del front-end

A lo largo de este capítulo se explicará en más detalle el funcionamiento de la aplicación según el rol que tenga el usuario. También se mencionarán la librerías que se han utilizado, así como la razón por la que se han usado. Finalmente se explicará el funcionamientos de los distintos componentes de *React* que se han creado para el funcionamiento de la aplicación.

5.1. Funcionalidad básica desde el punto de vista del usuario

Se distinguen tres grupos de usuarios que pueden usar la aplicación web.

5.1.1. Estudiantes

Cuando el estudiante inicie sesión, podrá ver el listado de asignaturas en las que está matriculado. Al seleccionar una asignatura, se mostrará un listado de cuestionarios. En este listado de cuestionarios, el estudiante podrá ver tanto los cuestionarios que haya realizado como los que no. Los cuestionarios tienen un plazo a partir del cual se pueden realizar. Una vez superado ese plazo el estudiante aparecerá como no presentado. En el caso de los cuestionarios realizados y corregidos, el estudiante podrá ver la nota de los mismos así como revisarlos. Al revisar un cuestionario, el estudiante podrá revisar las respuestas que haya dado para cada pregunta y ver las soluciones correctas de las preguntas.

En el caso de que haya un cuestionario disponible, el estudiante deberá descargárselo. Este se guardará encriptado en el dispositivo del estudiante. Para poder desencriptalo y proceder a realizarlo, el estudiante deberá introducir la contraseña que le otorgará el profesor al comienzo del examen. Al terminar de realizarlo el estudiante podrá enviarlo. Si el tiempo disponible para realizar el cuestionario se acaba, el cuestionario se cierra y se envía automáticamente. Al realizar el envío, si el estudiante se encuentra sin conexión, se generará un código QR que se deberá mostrar al profesor. Cuando el profesor lo escanee, lo redirigirá a la aplicación web donde tendrá que iniciar sesión. Después de iniciar sesión se insertará automáticamente en la base de datos el *hash* de la respuestas del estudiante. Cuando el estudiante vuelva a tener conexión, se enviarán las respuestas del cuestionario, se generará un *hash* de dichas respuestas que se compararán con el *hash* que se insertó con el código QR. Si estos dos *hashes* coinciden, el profesor podrá asegurarse de que el estudiante no ha modificado las respuestas después de la realización del examen.

5.1.2. Profesores

Cuando el profesor inicie sesión, podrá ver el listado de asignaturas que imparte. Al seleccionar una asignatura, se mostrará un listado de todos cuestionarios pertenecientes a dicha asignatura. Al igual que el estudiante, también tiene la posibilidad de realizarlos. Al revisarlos se mostrará una tabla con todos los estudiantes a los que se les haya asignado el cuestionario. En esta tabla podrá ver la nota de los estudiantes y en el caso de los estudiantes que le hubiesen enseñado el código QR, podrá ver si los *hash* coinciden. También puede revisar las respuestas de cada estudiante en detalle. El profesor también puede matricular estudiantes en las asignaturas que imparte, subir cuestionarios a las distintas asignaturas que imparte y descargar preguntas del banco de preguntas.

5.1.3. Administrador

El administrador es el único que puede acceder a través del panel de administración de *Django*. Este panel de administración proporciona una interfaz para gestionar la base de

datos de forma sencilla. De esta manera el administrador puede crear distintas asignaturas, profesores, estudiantes, etc.

5.2. Librerías utilizadas en el front-end

Las librerías que se han utilizado son las siguientes:

- jquery: Es una librería que permite utilizar *JQuery* en *React*.
- crypto-js: Es una librería que permite el cifrado y descifrado de, por ejemplo, cadenas de caracteres. Se ha utilizado para descifrar el test del *Local Storage* a la hora de realizarlo.
- js-yaml: Es una librería que permite convertir datos de formato JSON a YAML. Se ha utilizado para convertir las preguntas seleccionadas del banco de preguntas en formato YAML para su posterior descarga.
- workbox: Es una librería que se encarga de gestionar el cacheo de recursos y simplifica bastante la programación e implementación de los *Service Workers*. Está compuesta por distintos módulos que se pueden combinar para conseguir el comportamiento que se desea del *Service Worker*. Los módulos que se han utilizado son los siguientes:
  - workbox-core: Esta librería contiene el código base del que dependen los distintos módulos.
  - workbox-precaching: Esta librería proporciona la funcionalidad de pre-cacheo de los archivos estáticos de la aplicación web.
  - workbox-routing: Esta librería permite elegir qué peticiones interceptar y tratarlas usando las estrategias de cacheo.
  - workbox-strategies: Esta librería proporciona las distintas estrategias de cacheo, *Cache First, Network First, etc*.

- workbox-background-sync: Esta librería proporciona la funcionalidad que permite que el *Service Worker* reintente las peticiones fallidas.

- react-qr-code: Esta librería ofrece un componente de *React* que genera el código QR a partir del valor que se le proporciona. Se ha utilizado para generar el código QR que el estudiante deberá mostrar al profesor cuando este no tenga conexión.

- react-data-table-component: Esta librería ofrece un componente de *React* que es una tabla con funcionalidades avanzadas, como la ordenación de filas y paginación.

- react-countdown: Esta librería ofrece un componente de React que permite crear un contador. Se ha utilizado para mostrar al usuario el tiempo que le queda para finalizar el cuestionario.

5.3. Descripción de los componentes de React más importantes

5.3.1. NavBar

Este componente es un menú de navegación que aparece durante toda la navegación de la aplicación, ya que es esencial para navegar a ciertos apartados de la página y también para cerrar sesión. El menú de navegación del profesor es igual al del estudiante, con la excepción de que el profesor puede acceder a los apartados "Añadir alumno" y "Crear cuestionarios" como se puede ver en las Figuras 5.1 y 5.2.

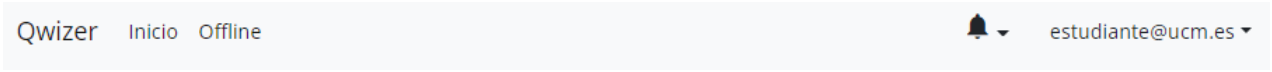

Figura 5.1: *Menú de navegación del estudiante*

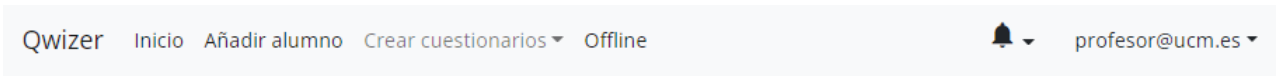

Figura 5.2: *Menú de navegación del profesor*

5.3.2. TarjetaAsignatura

Este componente representa una asignatura, ya sea esta una en la que esté matriculado el estudiante o que imparte el profesor. Contiene la siguiente información sobre la asignatura: el número de tests que tiene dicha asignatura, el número de tests realizados que se han corregido y finalmente el número de tests pendientes por hacer. Ver Figura 5.3.

Figura 5.3: *Componente* `TarjetaAsignatura`

5.3.3. TarjetaCuestionario

Este componente representa un cuestionario de una asignatura. Su funcionamiento es el siguiente:

- **En el caso de ser un estudiante:** Cuando haya cuestionario disponible, el usuario podrá descargárselo haciendo click en el botón "Descargar test". Una vez finalizada la descarga aparecerá el botón de "Realizar" y se podrá proceder a realizar el test, tal y como se puede observar en las Figuras 5.4 y 5.5.

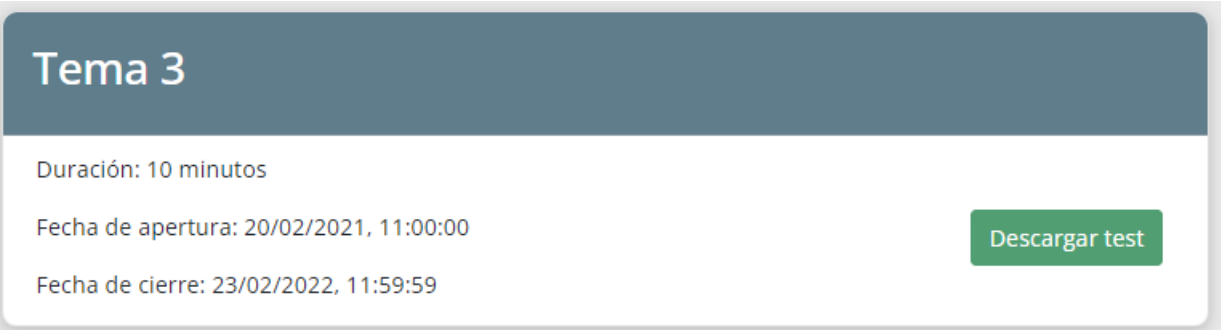

Figura 5.4: *Antes de descargar el test*

Figura 5.5: *Después de descargar el test*

Una vez que el cuestionario se haya corregido, el estudiante podrá ver su nota y le aparecerá un botón para revisarlo. Ver Figura 5.6

Figura 5.6: *Cuestionario corregido*

- En el caso de ser un profesor: Tendrá también tendrá la posibilidad de descargárselo para poder realizarlo. A diferencia del estudiante, el botón de revisar le aparecerá

sin necesidad de que haya realizado anteriormente el cuestionario. Al revisar el cuestionario, no solo podrá revisar sus respuestas en caso de que lo haya realizado, sino también la de sus estudiantes.

5.3.4. QuestionContainer

Este componente es utilizado tanto para la realización de cuestionarios como para su revisión. Ver Figuras 5.7 y 5.8.

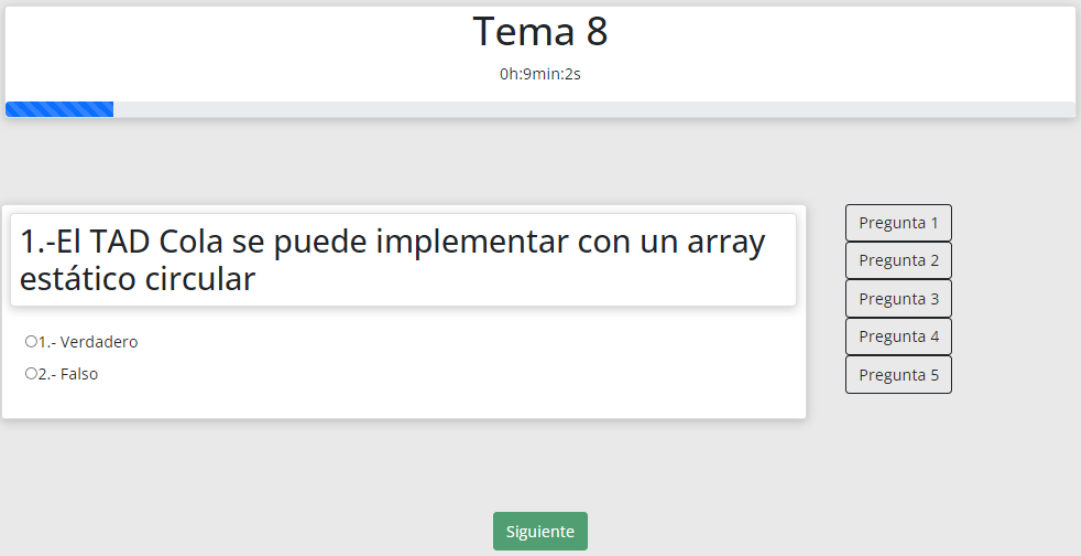

Figura 5.7: *Realización de test*

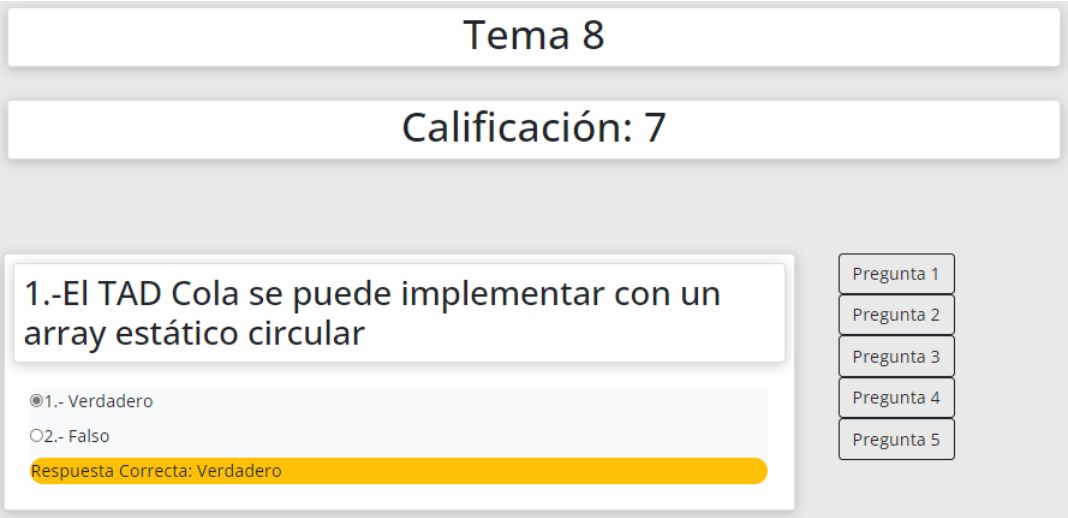

Figura 5.8: *Revisión de test*

Este componente está compuesto, a su vez, por los siguientes componentes:

5.3.4.1. Contador

Tal y como se ha mencionado anteriormente, este componente es proporcionado por la librería *react-countdown*. Este componente simplemente ofrece los números de la cuenta atrás. La barra azul que progresa según avanza el tiempo no forma parte de él.

5.3.4.2. QuestionNav

El componente `QuestionContainer` permite navegar de forma secuencial hacia adelante y hacia atrás durante la realización del test. Si el cuestionario tiene muchas preguntas, para navegar desde la primera hasta la última habría que pasar por todas las preguntas, lo cual puede llegar a ser tedioso para el usuario. Por ello se ha creado el componente `QuestionNav`, que permite al usuario navegar directamente a la pregunta que desee sin necesidad de pasar por preguntas intermedias. Ver Figura 5.9

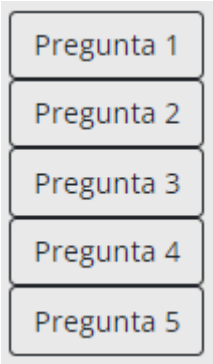

Figura 5.9: *Componente* `QuestionNav`

5.3.4.3. TextQuestion y TestQuestion

Estos componentes muestran las preguntas del cuestionario. Las preguntas de un cuestionario pueden ser de tipo *"text"* y tipo *"test"*. Las preguntas tipo *"text"* son aquellas en las que el usuario tiene que escribir la respuesta a la pregunta. En las del tipo *"test"* el usuario no debe escribir la respuesta, sino simplemente debe seleccionar una de las opciones de la pregunta. Además estos componentes son reutilizados en diversos apartados de la aplicación.

Estos componentes tienen 3 modos:

- **Modo test**: Este es modo principal que aparece cuando un usuario realiza un cuestionario. En este modo el usuario puede se seleccionar una opción en caso de que sea una pregunta tipo *test* o escribir la respuesta en caso de que sea una pregunta tipo *text*. Ver Figuras 5.10 y 5.11.

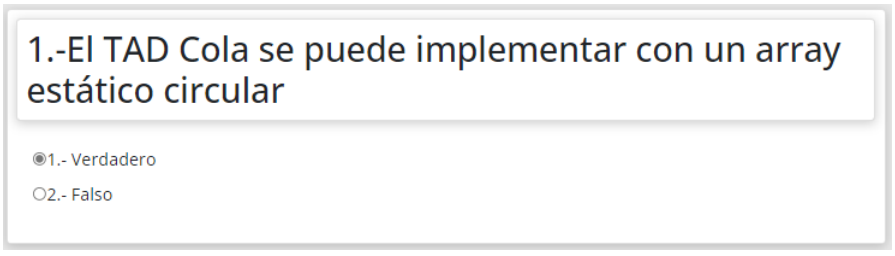

Figura 5.10: *Componente* `TestQuestion`

- **Modo revisión**: Este modo aparece cuando el usuario revisa un cuestionario. En

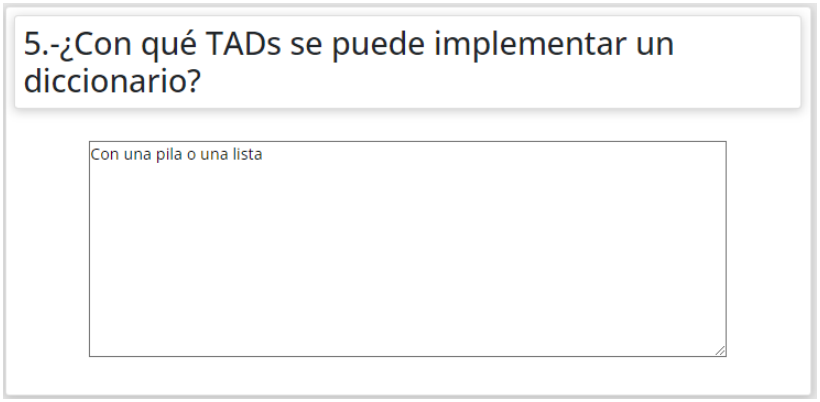

Figura 5.11: *Componente* `TextQuestion`

este modo el usuario no puede modificar las respuestas. Simplemente puede ver las respuestas que envió al terminar de realizar el cuestionario y las respuestas correctas de dichas preguntas. Ver Figuras 5.12 y 5.13.

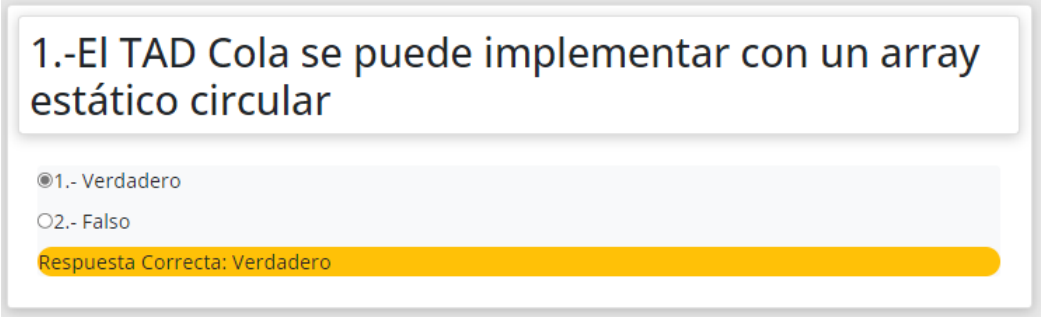

Figura 5.12: *Componente* `TestQuestion`

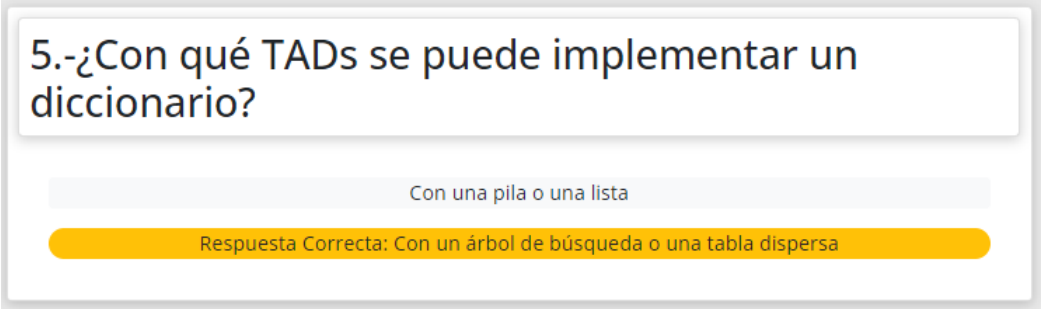

Figura 5.13: *Componente* `TextQuestion`

- **Modo visualización**: Este modo aparece cuando el profesor quiere visualizar una pregunta del banco de preguntas. En este modo el profesor puede ver la pregunta de forma mas detallada, es decir, con el titulo de la pregunta, la pregunta en sí y las opciones que tiene, así como la opción correcta en caso de que sea de tipo *test* o la respuesta correcta en caso de que sea tipo *text*. Ver Figuras 5.14 y 5.15.

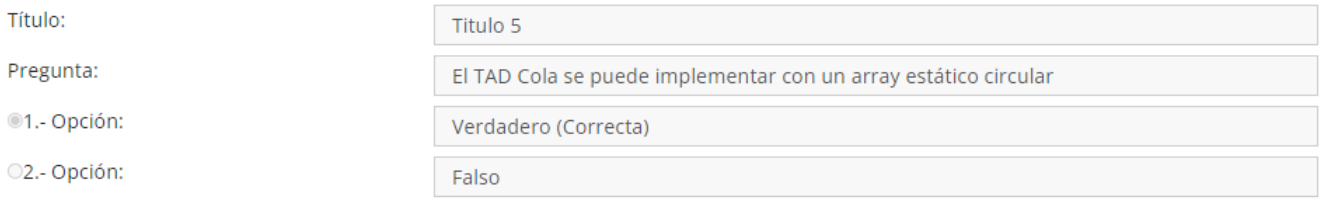

Figura 5.14: *Componente* `TestQuestion`

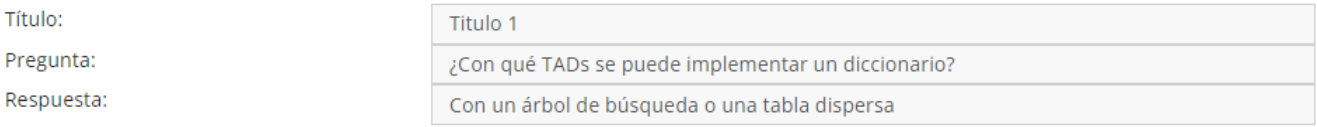

Figura 5.15: *Componente* `TextQuestion`

5.3.5. DataTable

Este componente es proporcionado por la librería *react-data-table-component*. Una razón por la que decidimos usar este componente es debido a la posibilidad de dibujar tablas visualmente atractivas de forma sencilla. Además, permite ordenar las columnas de menor a mayor y también ofrece paginación. Este componente es utilizado en los siguientes componentes:

5.3.5.1. RegisterContainer

Este componente muestra el listado de alumnos del centro con sus nombres y apellidos, y, permite al profesor seleccionarlos para matricularlos en las asignaturas que el profesor imparte. Ver Figura 5.16.

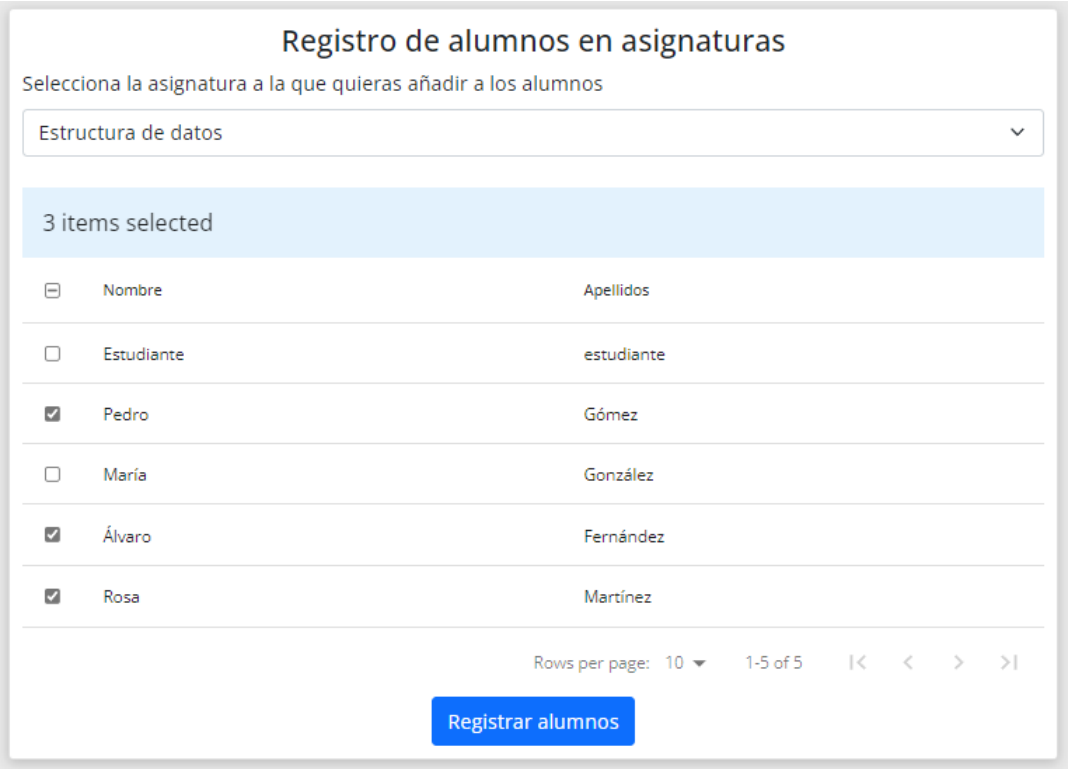

Figura 5.16: *Componente* `RegisterContainer`

5.3.5.2. RevisionNotasContainer

Este componente es el que se muestra cuando un profesor revisa un cuestionario. En él aparece el listado de alumnos que están matriculados en la asignatura a la que está asignado el cuestionario. Este listado también contiene las notas de cada alumno que haya realizado el cuestionario así como un botón para revisar las respuestas. Además, en caso de que un alumno haya realizado el cuestionario de manera *o ine*, el profesor podrá ver si los dos *hashes* generados son iguales o no. Ver Figuras 5.17, 5.18 y 5.19.

Figura 5.17: *Componente* `RevisionNotasContainer`

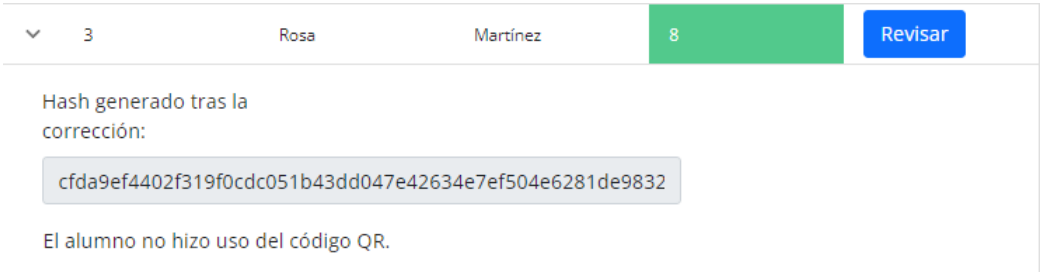

Figura 5.18: *En el caso de que se haya realizado el cuestionario de forma online*

Figura 5.19: *En el caso de que se haya realizado el cuestionario de forma o ine*

5.3.5.3. BancoPreguntas

Este componente muestra todas las preguntas de todas las asignaturas. Este banco de preguntas permite visualizar las preguntas, eliminarlas, descargarlas en formato YAML y editarlas. Ver Figuras 5.20 y 5.21.

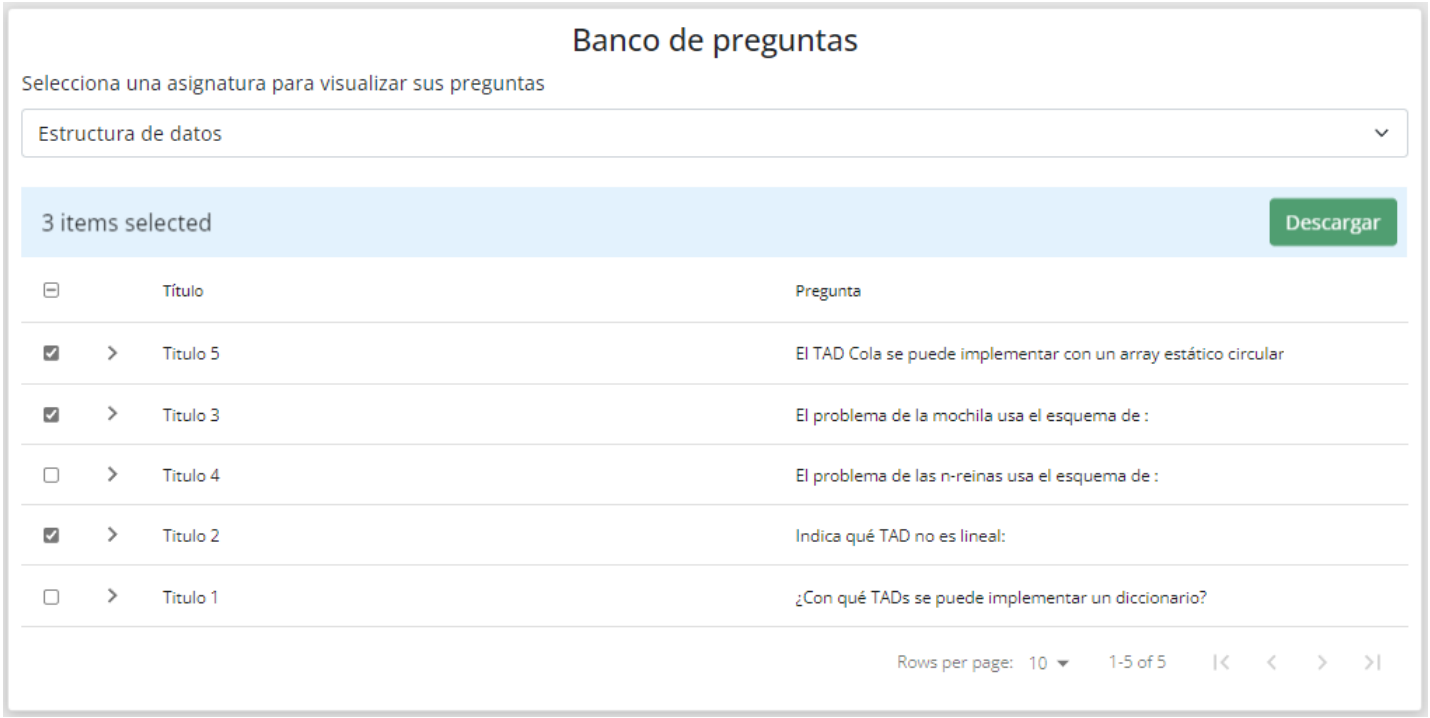

Figura 5.20: *Descargar preguntas en formato YAML*

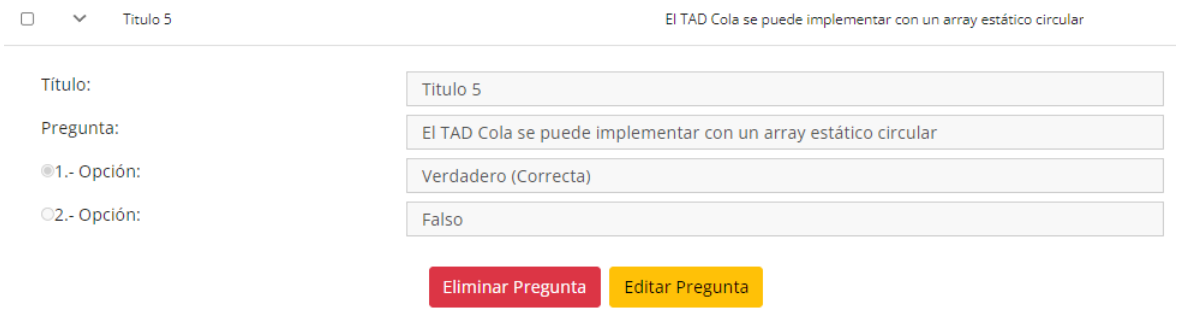

Figura 5.21: *Visualizar, eliminar y editar preguntas*

Al editar las preguntas se usan los componentes `EditTestQuestion` y `EditTextQuestion`. La razón por la que se han creado estos componentes en vez de reutilizar los componentes `TestQuestion` y `TestQuestion`, es porque al editar preguntas se editan varios campos: título de la pregunta, la pregunta en sí, las opciones en caso de que sea tipo *test* o la respuesta en caso de ser tipo *text*. La lógica de edición requeriría mucho código y haría que el componente sea más difícil de mantener. Ver Figuras 5.22 y 5.23.

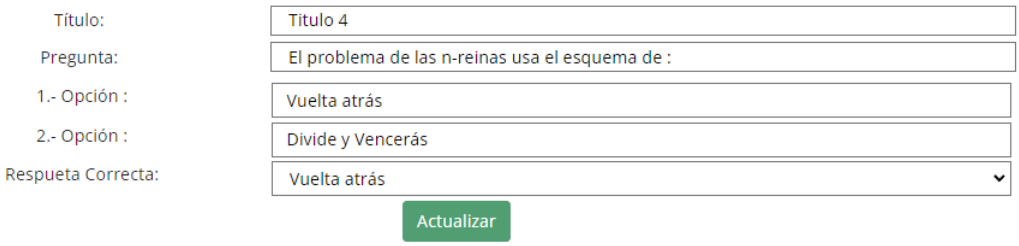

Figura 5.22: *Componente* `EditTestQuestion`

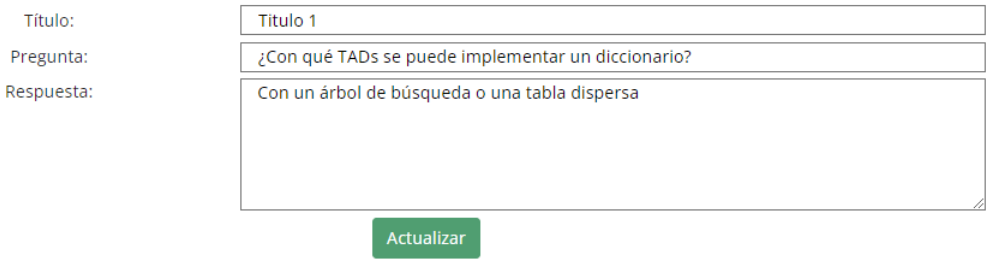

Figura 5.23: *Componente* `EditTextQuestion`

5.3.5.4. CrearCuestionario

Este componente permite crear un cuestionario desde la aplicación web. Está compuesto por tres secciones:

- **Información del cuestionario:** En esta primera sección se rellena gran parte de la información del cuestionario. Esta información es la siguiente: el nombre del cuestionario, la contraseña para poder realizarlo, la asignatura a la que pertenece, la duración, la fecha de apertura y cierre, y finalmente si la navegación es secuencial o no. Si la navegación es secuencial, no se permitirá navegar entre preguntas de forma salteada, además de que una vez que el usuario haya pasado de pregunta, no podrá volver a ella. Ver Figura 5.24.

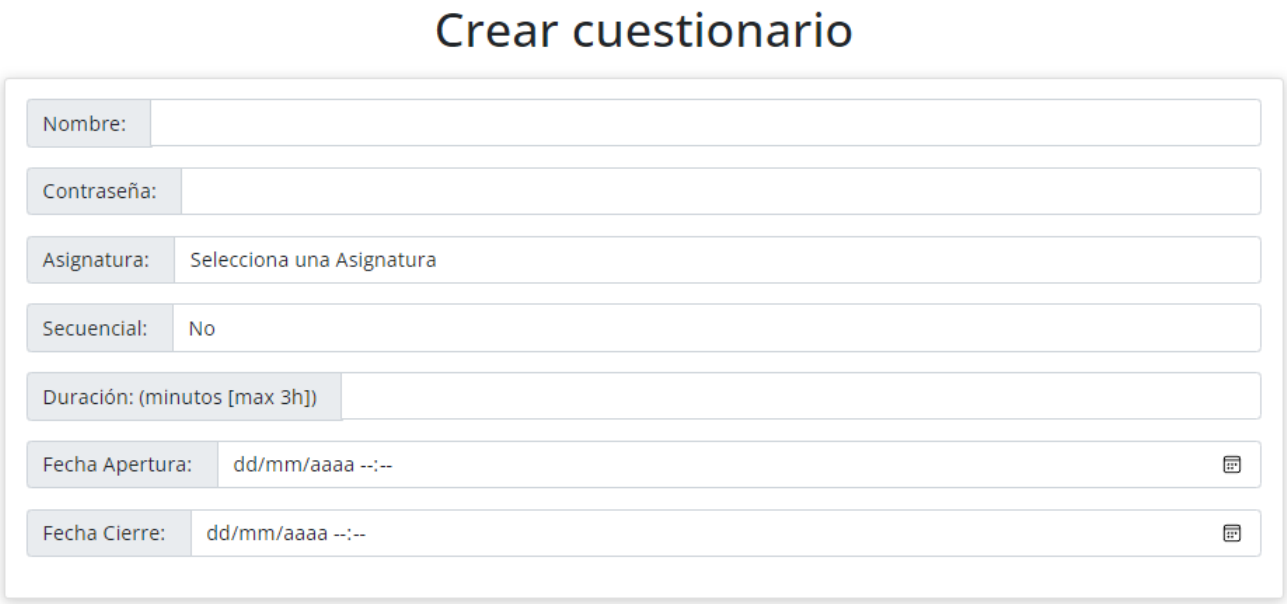

Figura 5.24: *Información del cuestionario*

- **Banco de preguntas:** En esta segunda sección el profesor puede visualizar las preguntas de cada asignatura y añadir aquellas que desee al cuestionario. Ver Figura 5.25.

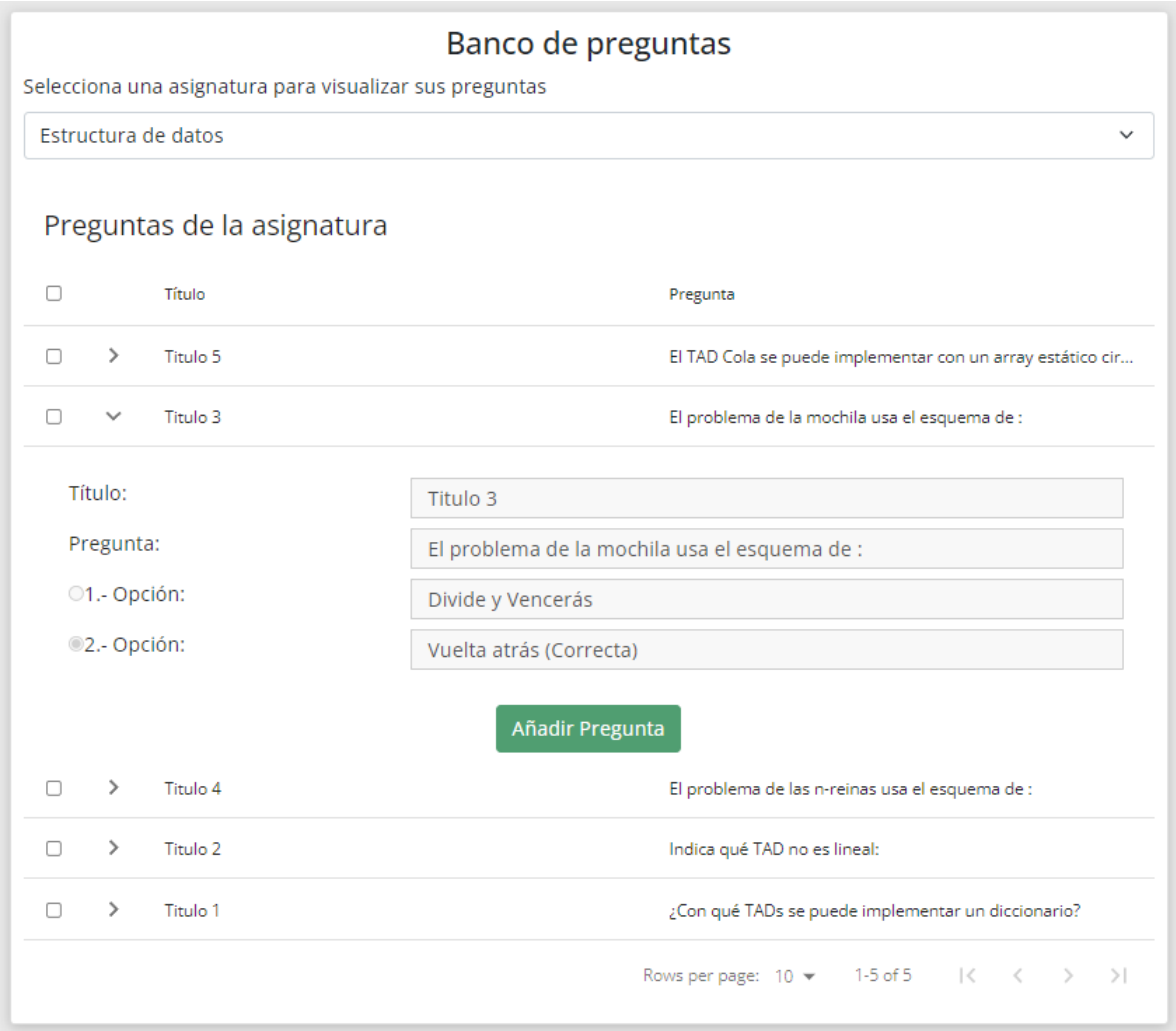

Figura 5.25: *Banco de preguntas*

- **Preguntas añadidas al cuestionario:** En esta tercera y última sección se mostrarán las preguntas que se hayan añadido al cuestionario y se podrá establecer la puntuación que se suma al acertar la pregunta así como la puntuación que se restará si se falla. También es posible eliminar las preguntas añadidas al cuestionario. Una vez que se hayan añadido las preguntas, el profesor puede proceder a guardar el cuestionario. Ver Figura 5.26.

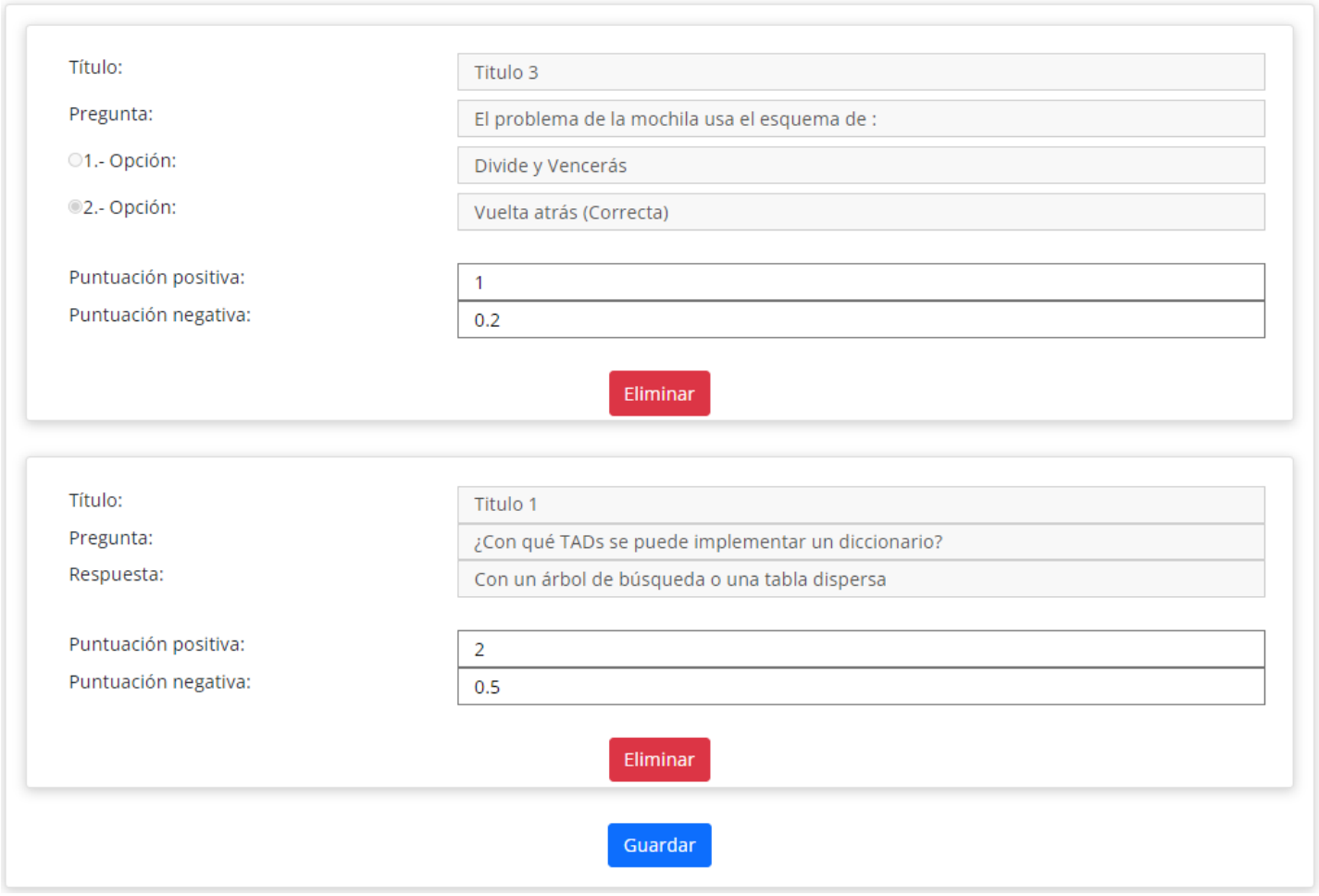

Figura 5.26: *Preguntas añadidas al cuestionario*

Capítulo 6

Otras funcionalidades

En este capítulo se explicarán otras funcionalidades que tiene la aplicación y también el funcionamiento del modo *o ine*.

6.1. Subir preguntas

Esta funcionalidad permite al profesor subir preguntas al banco de preguntas. Para ello se debe seleccionar la asignatura a la que se desee asociar dichas preguntas y luego seleccionar el fichero en formato YAML con las preguntas. Ver Figura 6.1.

Figura 6.1: *Interfaz para subir preguntas en formato YAML*

6.1.1. Descripción del formato YAML

Primero se empieza declarando el campo *preguntas* que es el que contendrá la lista de preguntas. Cada pregunta esta compuesta por los siguientes campos:

- tipo: Este campo indica el tipo de pregunta, que puede ser de tipo *"test"* o *"text"*.
- pregunta: Este campo contiene la pregunta en sí, que debe escribirse entre comillas.
- titulo: Este campo contiene el titulo de la pregunta.
- opciones: Este campo debe tomar valores distintos en función del tipo de pregunta. Si la pregunta es de tipo:
  - test: El campo deberá tomar un array de valores entre comillas que representarán las opciones de la pregunta.
  - text: El campo deberá tomar la respuesta a la pregunta entre comillas.
- opciones: Este campo solo debe estar para preguntas tipo *"test"* y toma como valor el índice de la opción correcta del array del campo *opciones*.

Ver Figura 6.2.

```
preguntas:
    - tipo: "text"
      pregunta: " ¿Con qué TADs se puede implementar un diccionario?"
      titulo: "Titulo 1"
      opciones: "Con un árbol de búsqueda o una tabla dispersa"
      punt_positiva: 3
      punt_negativa: 0.3
```

Figura 6.2: *Ejemplo: Formato YAML de una pregunta*

6.2. Subir cuestionarios

Esta funcionalidad permite al profesor subir cuestionarios mediante un fichero YAML. A diferencia de subir una pregunta, al subir un cuestionario no es necesario seleccionar la asignatura a la que pertenece, ya que ya está especificado en el fichero YAML. Por lo tanto simplemente se debe seleccionar el fichero en formato YAML con el cuestionario y hacer clic en el botón de "Subir cuestionario". Ver Figura 6.3.

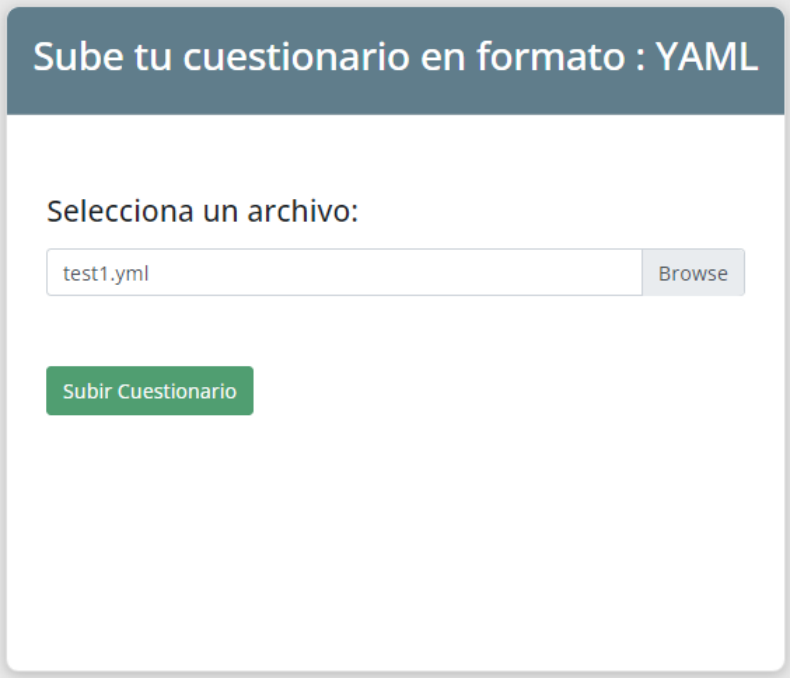

Figura 6.3: *Interfaz para subir un cuestionario en formato YAML*

6.2.1. Descripción del formato YAML

Primero se empieza declarando el campo *cuestionario*, que es el que contendrá toda la información del cuestionario. Los campos del cuestionario son los siguientes:

- titulo: Este campo contiene el titulo del cuestionario.
- password: Este campo contiene la contraseña del cuestionario.
- asignatura: Este campo contiene la asignatura a la que pertenece el cuestionario.
- secuencial: Este indica si el cuestionario permite navegación secuencial, tomando el valor 1, o no, tomando el valor 0.
- duracion: Este campo indica la duración del cuestionario en minutos.
- fecha apertura: Este campo indica la fecha de apertura del cuestionario, y tiene el siguiente formato: Año/Mes/Día Horas:Minutos:Segundos. Cada uno de estos valores deben tener dos dígitos.
- fecha cierre: Este campo indica la fecha de cierre del cuestionario, y el formato es igual al de fecha de apertura.
- preguntas: Este campo contiene un array de preguntas. Toma el mismo formato al subir preguntas explicado en la sección anterior, pero con una ligera diferencia: las preguntas ahora tienen dos campos adicionales que son los siguientes:
  - punt positiva: Este campo toma la puntuación positiva que se suma a la calificación al acertar la pregunta.
  - punt negativa: Este campo toma la puntuación que se resta a la calificación al fallar la pregunta.

Ver Figura 6.4.

```yaml
cuestionario:
    titulo: "Tema 8"
    password: "1234"
    asignatura: "Estructura de datos"
    secuencial: 0
    duracion: 10
    fecha_apertura: '21/02/20 11:00:00'
    fecha_cierre: '23/02/23 11:59:59'
    preguntas:

        - tipo: "text"
          pregunta: " ¿Con qué TADs se puede implementar un diccionario?"
          titulo: "Titulo 1"
          opciones: "Con un árbol de búsqueda o una tabla dispersa"
          punt_positiva: 3
          punt_negativa: 0.3

        - tipo: "test"
          pregunta: "El problema de la mochila usa el esquema de:"
          titulo: "Titulo 2"
          opciones: ["Divide y Vencerás","Vuelta atrás"]
          op_correcta: 1
          punt_positiva: 1.0
          punt_negativa: 0.3
```

Figura 6.4: *Ejemplo: Formato YAML de un cuestionario*

6.3. Explicación del modo o□ine

En esta sección se explicará de forma más detallada el funcionamiento del modo *o ine* y la generación del código QR.

6.3.1. Caso de uso

Supongamos que un estudiante se descarga un examen horas antes de su fecha de inicio, y no cierra sesión. El estudiante cierra el navegador. Si, a la hora de realizar el cuestionario, no tiene conexión, debe poder acceder a la página web de Qwizer. Una vez en la página, deberá pulsar el apartado "*O ine*"del menú de navegación donde aparecerán todos los cuestionarios descargados, y podrá realizarlos. Al terminar, se le generará un QR que deberá mostrárselo al profesor. El profesor escaneará el QR, el cual le redirigirá a la aplicación web de Qwizer y le pedirá que inicie sesión. Una vez que inicie sesión, se añadirá el *hash* de las respuestas

a la base de datos. Cuando el estudiante vuelva a tener conexión, las respuestas se enviarán automáticamente al servidor. El profesor al revisar el cuestionario, podrá ver si el *hash* que se insertó con el código QR coincide con el que se generó cuando las respuestas enviadas. Si coinciden, el profesor puede estar seguro de que las respuestas enviadas por el estudiante no han sido modificadas después del examen.

6.3.2. Service Worker

Como se ha explicado en el capítulo 3, el *Service Worker* es esencial para que la aplicación web funcione de manera *o ine*. Se ha programado para que ofrezca las dos siguientes funcionalidades:

6.3.2.1. Acceso o□ine a la página web

Esta es la primera funcionalidad básica del *Service Worker* y permite que se pueda acceder a la página aunque no se disponga de conexión internet. Para conseguirla simplemente hay que pre-cachear los archivos estáticos de la aplicación. Esto se consigue gracias a la función `precacheAndRoute()` que proporciona la librería *workbox-precaching* de *Workbox*.

```
1
2           precacheAndRoute(self.__WB_MANIFEST);
3
4           precacheAndRoute([
5               {url: '/service-worker.js', revision: null},
6               {url:'/manifest.json', revision:null}
7           ]);
8
9
```

Figura 6.5: *Código para el pre-cacheo de los archivos estáticos*

La primera llamada a la función `precacheAndRoute()` cachea los archivos estáticos de la aplicación, pero en esa primera llamada no cachea los archivos *"service-worker.js"* y *"manifest.json"* por ello se vuelve a hacer otra llamada a la función y se añaden estos archivos.

6.3.2.2. Envío de respuestas al recuperar la conexión a internet

Esta funcionalidad conocida como *Background-Sync*, permite reintentar las peticiones fallidas. Al enviar las respuestas del cuestionario cuando no hay conexión, estas fallan y se cachean en la base datos del navegador. Esta base de datos es conocida como *IndexedDB*, y es es una base de datos NoSQL. Cuando el *Service Worker* detecta que el cliente vuelve a tener conexión, reintenta las peticiones fallidas que se cachearon.

```
1
2     const bgSyncPlugin = new BackgroundSyncPlugin('test-post-requests', {
3                 maxRetentionTime: 24 * 60,
4             });
5
6     const sentTestUrl = "http://127.0.0.1:8000/api/response";
7
8     registerRoute(
9         sentTestUrl,
10        new NetworkOnly({
11            plugins: [bgSyncPlugin],
12        }),
13        'POST'
14    );
15
16
```

Figura 6.6: *Código para el reintento de peticiones*

Lo primero que se hace es crear un objeto de la clase *BackgroundSyncPlugin* que proporciona la librería *workbox-background-sync*. Este objeto se encarga de cachear las peticiones fallidas en la base de datos *IndexedDB* y de reintentarlas cuando haya conexión. Al crearlo, hay que especificar el nombre de la entrada del *IndexedDB* donde se almacenarán las peticiones fallidas y también el tiempo máximo de reintentos, que en este caso es de 24 horas. Lo segundo que hay que hacer es interceptar la petición que envía las respuestas del cuestionario. Esto se hace con la función `registerRoute()` que proporciona la librería *workbox-routing*, donde primero se especifica la URL de la petición, después la estrategia de cacheo pasandole el objeto de la clase *BackgroundSyncPlugin* y finalmente el tipo de petición. En este caso se ha usado la estrategia de cacheo *NetworkOnly* proporcionado por

workbox-strategies. Esta estrategia realmente no cachea; simplemente especifica que la petición solo puede ser resuelta a través de internet y no a través de la caché.

6.3.3. Código QR

La generación del código QR es esencial para poder comprobar la integridad de las respuestas. Es decir, con el código QR nos podemos asegurar de que el estudiante no haya modificado las respuestas del cuestionario después del examen. El código QR contiene una URL con la siguiente información: el identificador del usuario y del cuestionario, y un *hash* de las respuestas. El profesor, al revisar el cuestionario, podrá ver si los hashes coinciden o no. Ver Figuras 6.7 y 6.8.

Figura 6.7: *Hashes coinciden*

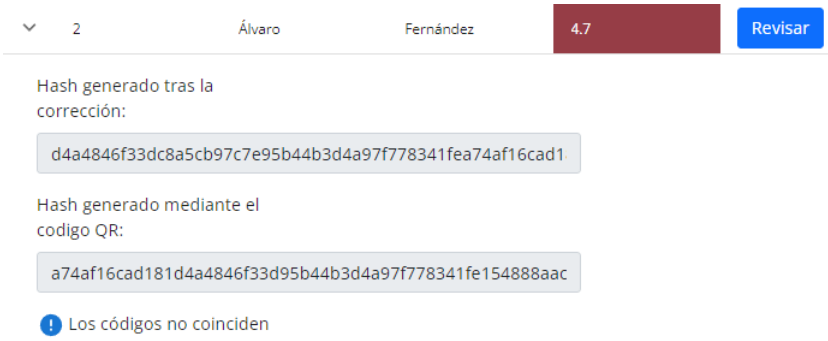

Figura 6.8: *Hashes no coinciden*

Capítulo 7

Conclusiones y trabajo futuro

En este capítulo expondremos las conclusiones a las que hemos llegado gracias al desarrollo del proyecto. También comentaremos aquellas funcionalidades que, debido a la falta de tiempo, no pudimos incluir en la aplicación.

7.1. Objetivos conseguidos

Los objetivos que se plantearon en un principio y que se comentaron durante la sección 1.2 se completaron en su totalidad:

- Los estudiantes pueden descargarse los cuestionarios para su posterior realización. Además, estos cuestionarios están cifrados de manera de que el estudiante no pueda ver las preguntas antes del comienzo del examen.
- Durante el plazo en el que esta abierto el cuestionario, los estudiantes pueden proceder a realizarlo con la clave que proporciona el profesor al inicio del examen. Las respuestas del estudiante se almacenan en el navegador.
- Las respuestas del cuestionario se envían si el alumno decide acabar antes de tiempo o si se acaba el tiempo disponible para su realización. Las respuestas enviadas se corrigen automáticamente. En los casos en los que no hay conexión, se genera el QR con el *hash* de las respuestas que escaneará el profesor.

Además de estos objetivos principales completamos otros, los cuales fueron planteándose a lo largo del desarrollo del proyecto. Estos objetivos son los siguientes:

- Revisión de cuestionarios. Los estudiantes, después de realizar los cuestionarios, pueden revisarlos. Los profesores también pueden revisar los cuestionarios de sus alumnos.
- Creación de cuestionarios desde la aplicación web. Los profesores pueden crear un cuestionario de dos formas. La primera hace uso de la interfaz web de la propia aplicación. La segunda opción consiste en subir a través de la aplicación un fichero YAML, el cual contiene toda la información del cuestionario.
- Subir preguntas al banco de preguntas mediante un fichero YAML. Los profesores pueden subir varias preguntas simultáneamente mediante un fichero YAML. Este fichero se puede subir mediante un apartado de la aplicación.
- Permitir que un profesor matricule a un alumno en su asignatura. Los profesores pueden matricular a alumnos en las asignaturas que ellos imparten. Esto lo pueden hacer a través de un apartado de la aplicación.

7.2. Dificultades durante el desarrollo

Durante el desarrollo nos hemos topado con diversas dificultades, debido a que las tecnologías que hemos usado presentaban una novedad para nosotros. Consultando varias fuentes, como por ejemplo, documentación oficial de las tecnologías, libros, blogs, y mediante tutoriales en formato audiovisual hemos conseguido superar estas dificultades y hemos aprendido muchas cosas interesantes del desarrollo web.

7.2.1. React

Como era la primera vez que hacíamos uso de React, en muchas ocasiones aparecían errores y no entendíamos el porqué. La razón de estos errores era el desconocimiento que

teníamos sobre el ciclo de vida de los componentes de React, lo cual era algo novedoso para nosotros. Sin embargo, gracias a la práctica y la lectura de la documentación obtuvimos los conocimientos necesarios para identificar los errores rápidamente.

7.2.2. Modificación del modelo *user* de Django

Uno de los problemas a los que nos enfrentamos para crear el *login* de nuestra aplicación fue el modelo *user*. Este modelo es proporcionado por Django, y por defecto es usado a la hora de autenticar un usuario, haciendo uso de el nombre de usuario y la contraseña. A nosotros no nos interesaba que se autenticase mediante el nombre de usuario, pero nos interesaba que se hiciera uso del correo electrónico, ya que nuestra aplicación no cuenta con nombres de usuario. Este modelo, a diferencia de otros que hemos creado, es el más complejo ya que cuenta con una gran cantidad de atributos, los cuales son usados por Django para llevar a cabo el proceso de autenticación. Después de leer la documentación oficial y blogs, finalmente encontramos la solución.

7.2.3. Integración de React con Django

Otro de los problemas que nos encontramos a la hora de iniciar sesión, era el manejo de las propias sesiones. Django por defecto usa *sesiones* para autenticar las peticiones. Para usar las sesiones desde el *front-end* teníamos que compilar el proyecto de React e integrarlo con Django cada vez que queríamos probar una nueva funcionalidad. Esto dificultaba el desarrollo ya que después de compilar el proyecto teníamos que moverlo a la carpeta correspondiente del proyecto de Django. Todo este proceso duraba aproximadamente un minuto. Por ello decidimos cambiar la *autenticación por sesiones* a *la autenticación por tokens*.

7.2.4. Cifrado de los cuestionarios

Este problema se dio a la hora de cifrar los cuestionarios en el *back-end* para posteriormente ser enviados al *front-end*, y una vez el alumno tuviera la contraseña, descifrarlos. El problema se puede dividir a su vez en dos problemas:

- El primer problema radicó en la biblioteca usada para cifrar contenido en el *back-end*, la cual es *pycryptodome*. A la hora de cifrar una cadena de texto haciendo uso de esta biblioteca, el texto cifrado es devuelto en formato ASCII en hexadecimal, formato que no es aceptado por el protocolo HTTP. Además, los errores que nos aparecían no eran nada esclarecedores. Tras bastante investigación, y con ayuda de nuestro tutor, finalmente lo solucionamos codificando esta cadena de texto en base 64.

- El segundo problema apareció debido a las incompatibilidades que había entre la biblioteca usada para cifrar en el *back-end* y la usada en el *front-end*. Para poder hacer que las dos bibliotecas funcionasen correctamente tuvimos que probar varios *modos de bloque* a la hora de cifrar.

7.2.5. PWA y Service Worker

Una de las mayores dificultades con las que nos encontramos fue hacer que la aplicación web fuese progresiva. Esta es la funcionalidad a la que más tiempo le dedicamos. La razón de su dificultad era el uso del *Service Worker*. Aunque entendíamos su funcionamiento, no teníamos muy claro como integrarlo con React. Buscando información al respecto, logramos encontrar la forma de integrarlo, ya que en la documentación oficial de React ofrecían un plantilla con dos ficheros. El primero se encargaba de integrar el *Service Worker* y el segundo era el que había que modificar para programar el comportamiento del mismo. Otra de las dificultades era programarlo para que gestionase las diferentes versiones de la caché y hacer que reintentase la peticiones fallidas, lo cual no era una tarea sencilla. Investigando, encontramos *Workbox*, una librería muy utilizada para la creación de los *Service Workers*, ya que simplificaba su programación. Una vez que supimos como integrarlo y que para programarlo teníamos que usar *Workbox*, invertimos mucho tiempo leyendo la documentación de *Workbox*. Finalmente conseguimos programar el *Service Worker* con el comportamiento que buscábamos. Una de las cosas que había que tener en cuenta a la hora de probar esta nueva funcionalidad, era que había que compilar el proyecto, ya que al no hacerlo no solo no

funcionaba sino que también aparecían errores en funcionalidades anteriores que ya estaban funcionando correctamente.

7.3. Trabajo futuro

Aunque la aplicación web ya dispone de muchas funcionalidades, aún falta pulir algunos aspectos para que esté más completa. Algunos de estos aspectos son los siguientes:

- Crear una interfaz para añadir nuevos usuarios: Actualmente para poder crear usuarios debemos hacer uso de un comando en la consola de Django (*python manage.py create superuser*), lo cual no es cómodo para un administrador. En un comienzo nos planteamos hacer uso del panel de administración de Django para llevar a cabo esta tarea, pero a la hora de añadir usuarios este panel no guarda un *hash* de las contraseñas, sino que las almacena en texto plano. Esto es un problema, ya que cualquiera con acceso a la base de datos podría obtener la contraseña de cualquier usuario. Además el comando sí que genera los usuarios haciendo uso de una *función resumen* para guardar la contraseña en la base de datos, y a la hora de hacer *login*, a la contraseña que introduce el usuario también se le aplica esta función, por lo que si hicieramos uso del panel de Django las contraseñas no coincidirían, ya que una se encontraría en texto plano y la otra sería un *hash* de la contraseña que ha introducido el usuario. La razón por la que queremos integrar la interfaz para crear usuarios con el panel de Django, es porque queremos que solo el administrador pueda crear las cuentas, ya que a cada cuenta hay que asignarle un rol (estudiante, profesor, administrador) y no queremos permitir que un alumno o profesor se registren como administradores. Para solucionar este problema creamos una función en la API de Django (4.3.6), pero no creamos ninguna interfaz en el *front-end* que hiciera uso de la misma.

- Añadir la posibilidad de restaurar las contraseñas de los usuarios: Actualmente, si a un usuario se le olvida la contraseña, no podrá acceder a la aplicación.

Por ello queremos que se implemente una forma que le permita al usuario cambiar de contraseña cuando se le olvide. Una posibilidad para llevar esto a cabo es hacer uso del campo de "último inicio de sesión" del modelo *users*. Para restaurar una contraseña, el administrador pondría ese campo a un valor inicial o nulo, y a la hora de iniciar sesión, se comprobaría el valor de ese campo. Si, por ejemplo, ese campo tuviese "null" como valor se detectaría que la contraseña ha de restablecerse, y le aparecería al usuario una pantalla en la que podría escribir su nueva contraseña.

- Posibilidad de hacer que las preguntas de un cuestionario se resuelvan únicamente de forma secuencial: Si bien se ha mencionado que a la hora de crear un cuestionario este podía ser secuencial o no, en la práctica no esta implementado. Es decir, aunque un cuestionario sea secuencial, el usuario puede seguir saltando de una pregunta a otra pregunta libremente. También puede volver a una pregunta anterior. Nos parece que la navegación secuencial es fácil de implementar y que puede ser de utilidad en ciertas situaciones, pero por falta de tiempo no se ha podido llevar a cabo.

- Posibilidad de hacer que las preguntas de un cuestionario se muestren de forma aleatoria: Otra funcionalidad que nos gustaría implementar es la aleatoriedad en las preguntas de un cuestionario. Esta funcionalidad nos parece importante ya que puede dificultar las copias.

- Posibilidad de resolver más de un cuestionario de forma o□ine: Tal y como está implementado actualmente, si el usuario realiza un cuestionario de manera *o ine* y quiere realizar otro, las respuestas del primer cuestionario se eliminarían. Por ello queremos que se puedan realizar varios cuestionarios sin que se eliminen las respuestas.

- Mejorar la interfaz de usuario para dispositivos móviles: Aunque actualmente la aplicación hace uso de una interfaz adaptable, es decir, que se adapta a distintos tamaños de pantallas, nos gustaría hacer que esta interfaz fuera visualmente más atractiva para el usuario en algunos apartados de la aplicación.

- Mejorar la interfaz de usuario en la sección de creación de un cuestionario: También pensamos que la interfaz para crear un cuestionario se podría mejorar para dar una mejor experiencia al profesorado.

- Añadir *Typescript* a la aplicación: *Typescript* es un lenguaje de programación con la misma sintaxis que *JavaScript*, pero fuertemente tipado. Es decir, las variables en *JavaScript* pueden tomar valores de cualquier tipo, pero con *Typescript* solo pueden tomar valores del tipo indicado en su declaración. Creemos que usar este lenguaje es una buena idea, porque muchas veces, al revisar los datos que se pasaban entre los componentes de React, teníamos que inspeccionar el componente al que se le pasaba los parámetros y ver si lo que recibía era una función, una cadena, un número, etc, pero sobre todo si era un objeto, ya que este contenía varias variables de distintos tipos.

Capítulo 8

Conclusions and future work

In this chapter we will expose about the conclusions we have reached thanks to the development of the project. We will also comment on those functionalities that, due to lack of time, we could not include in the application.

8.1. Achieved goals

The objectives that were initially set out and discussed during section 1.2 were fully completed:

- Students can download the questionnaires for later completion. In addition, these questionnaires are encrypted so that the student cannot see the questions before the start of the exam.
- During the period in which the questionnaire is open, students can proceed to take it by introducing the password provided by the teacher at the beginning of the exam. Student responses are stored in the browser.
- The answers of the questionnaire are sent if the students decide to finish early or if the time available for its completion runs out. The submitted answers are automatically corrected. In the cases where there is no internet connection, the QR is generated with the *hash* of the answers that the teacher will scan.

In addition to these main objectives we have completed others that were raised throughout the development of the project. These objectives are as follows:

- **Review of questionnaires**. Students can review the questionnaires after taking them. Teachers can also review their students questionnaires.
- **Creation of questionnaires through the web application**. Teachers can create a questionnaire in two ways. The first way is making use of the web interface of the application itself. The second way is to upload through the application a YAML file, which contains all the information of the questionnaire.
- **Upload questions to the question bank using a YAML file**. Teachers can upload several questions simultaneously using a YAML file. This file can be uploaded through a section of the application.
- **Allow a teacher to enroll a student in their subject**. Teachers can enroll students in the subjects they teach. This can be done through a section of the application.

8.2. Difficulties encountered

During the development we have encountered various difficulties because the technologies we have used were new to us. Consulting several sources, such as official documentation of technologies, books, blogs, and through tutorials in audiovisual format we have managed to overcome these difficulties and we have learned many interesting things about web development.

8.2.1. React

As it was the first time we made use of React, on many occasions errors appeared and we did not understand why. The reason for these errors was the lack of knowledge we had about the life cycle of React components, which was something new for us. However, thanks

to practice and reading the documentation we obtained the necessary knowledge to identify errors quickly.

8.2.2. Modifying Django *user* model

One of the problems we faced in creating the *login* of our application was the *user* model. This model is provided by Django, and by default is used when authenticating a user, making use of the username and password. We were not interested in it being authenticated by the username, but we were interested in it making use of email, since our application does not have usernames. This model, unlike others we have created, is more complex and has a large number of attributes, which are used by Django to carry out the authentication process. After reading the official documentation and blogs, we finally found the solution.

8.2.3. React integration with Django

Another problem we encountered when logging in was the management of the sessions themselves. Django uses *sessions* by default to authenticate requests. To use the sessions from the *front-end* we had to compile the React project and integrate it with Django every time we wanted to try a new functionality. This made development difficult, after compiling the project we had to move it to the corresponding folder of the Django project. This whole process took approximately a minute. That's why we decided to change *session authentication* to *token authentication*.

8.2.4. Encryption of questionnaires

Este problema se dio a la hora de cifrar los cuestionarios en el *back-end* para posteriormente ser enviados al *front-end*, y una vez el alumno tuviera la contraseña, descifrarlos. El problema se puede dividir a su vez en dos problemas:

- The first problem lay in the library used to encrypt in the *back-end*, which is *pycryptodome*. When encrypting using this library, the ciphertext is returned in ASCII format

in hexadecimal, a format that is not accepted by the HTTP protocol. In addition, the errors that appeared to us were not at all enlightening. After a lot of research, and with the help of our tutor, we finally solved it by encoding in base 64.

- The second problem appeared due to incompatibilities between the library used to encrypt in the *back-end* and the one used in the *front-end*. In order to make the two libraries work properly we had to try several *block modes* when encrypting.

8.2.5. PWA and Service Worker

One of the biggest difficulties we encountered was making the web application progressive. It is the functionality to which we spent more time. The reason for its difficulty was the use of the *Service Worker*. Although we understood how it works, we were didn't know how to integrate it with React. Searching for information about it, we managed to find a way to integrate it, since in the official documentation of React they offered a template with two files. The first was responsible for integrating the *Service Worker* and the second was the one that had to be modified to program its behavior. Another difficulty was to program it to manage the different versions of the cache and make it retry the failed requests, which was not an easy task. Researching, we found *Workbox*, a library widely used for the creation of *Service Workers* since it simplified their programming. Once we knew how to integrate it and that to program it we had to use *Workbox*, we spent a lot of time reading the *Workbox* documentation. Finally we managed to program the *Service Worker* with the behavior we were looking for. One of the things that had to be taken into account when testing this new functionality, was that the project had to be compiled, since by not doing so not only did it not work but also errors appeared in previous functionalities that were already working correctly.

8.3. Future work

Although the web application already has many functionalities, some aspects still need to be polished to make it more complete. Some of these aspects are the following:

- **Create an interface to add new users:** Currently to be able to create users we must make use of a command in the Django console (*python manage.py create superuser*), which is not comfortable for an administrator. At first we considered making use of the Django administration panel to accomplish this task, but when adding users this panel does not save a *hash* of the passwords, but stores them in plain text. This is a problem, as anyone with access to the database could get any user's password. In addition, the command previously mentioned generates users using a textithash function to save the password in the database, and when logging in, this function is applied to the password entered by the user, so if we made use of the Django panel the passwords would not match, since one would be in plain text and the other would be a textithash of the password that the user has entered. The reason why we want to integrate the interface to create users with the Django panel, it is because we want only the administrator to be able to create the accounts, since each account must be assigned a role (student, teacher, administrador) and we don't want to allow a student or teacher to register as administrators. To solve this problem we create a function in the Django API (4.3.6), but we did not create any interface in the *front-end* that made use of it.

- **Add the possibility to reset user passwords:** Currently, if a user forgets their password, they would not be able to access the app. That is why we want to implement a way that allows the user to change their password when they forget it. One possibility to carry this out is to make use of the "last login" field of the model *users*. To restore a password, the administrator would put that field to an initial value or null, and when logging in, the value of that field would be checked. If for example, that field had "null" as a value, it would detect that the password needs to be reseted, and a page would

appear to the user sp they can type in the new password.

- Possibility to have questionnaire questions resolved only sequentially: Although it has been mentioned that when creating a questionnaire this could be sequential or not, it is not implemented. That is, even if a questionnaire is sequential, the user can continue to jump from one question to another question freely. They can also return to a previous question. We think that sequential navigation is easy to implement and it can be useful in certain situations, but due to lack of time it has not been possible to carry it out.

- Possibility to make a questionnaire questions randomly displayed: Another functionality that we would like to implement is randomness in the questions of a questionnaire. We think this functionality is important since it can make cheating difficult.

- Possibility to solve more than one questionnaire o□ine: Currently, if the user takes a questionnaire offline and wants to perform take another, the answers from the first questionnaire would be deleted. That is why we want several questionnaires to be able to be carried out without the answers being deleted.

- Improve the user interface for mobile devices: Although currently the application makes use of an adaptable interface, that is, it adapts to different sizes of screens, we would like to make this interface visually more attractive to the user in some sections of the application.

- Improve the user interface in the quiz creation section: We also think that the interface to create a questionnaire could be improved to give a better experience to teachers.

- Add *Typescript* to the application: *Typescript* is a programming language with the same syntax as *JavaScript*, but strongly typed. That is, variables in *JavaScript*

can take values of any type, but with *Typescript* they can only take values of the type indicated in the declaration. We believe that using this language is a good idea, because many times, when reviewing the data that is passed between React components, we had to inspect the component to which the parameters were passed and see if what it received was a function, a string, a number, etc, but especially if it was an object, since it contained several variables of different types.

Capítulo 9

Contribución personal de cada autor

A lo largo de este capítulo hablaremos de nuestras contribuciones. Ambos hemos tocado aspectos tanto del *front-end* como del *back-end*. Aunque hubo un reparto de tareas, hay que cosas que hemos realizado juntos como por ejemplo:

- Diseño de la base de datos.
- Bocetos de la interfaz de la aplicación. Hemos creado los bocetos de la intefaz de aplicación y también como se navegaría entre sus distintos apartados.
- Correción de errores. Durante el desarrollo surgieron diversos errores tanto en el *front-end* como en el *front-end*.
- Testeo de la aplicación. Ambos, no solo testeábamos lo que habíamos desarrollado individualmente, sino también lo que el otro había desarrollado.

.

9.1. Zakaria El Fakhri Ouajih

Durante el desarrollo del Trabajo de Fin de Grado, he realizado las siguientes contribuciones:

- **Cuestionarios.** He creado la interfaz que permite realizar los cuestionarios y navegar por las distintas preguntas. Esto incluye el almacenamiento de las respuestas en el *LocalStorage* para su posterior envío. Al principio solo se podía navegar hacia adelante y atrás, pero con la sugerencia del director del proyecto, he implementado que se pueda navegar de forma salteada a través de las preguntas. Esta misma interfaz también es usada para la revisión de los cuestionarios, ya sea el usuario un estudiante que quiera revisar su cuestionario o un profesor que desee revisar los cuestionarios de sus alumnos. Para ello se ha creado en el *front-end* los siguientes componentes: `QuestionContainer, QuestionNav, TestQuestion y TextQuestion`. Y en el *back-end* la función que devuele el cuestionario corregido del usuario y la función que corrige el cuestionario y guarda la nota en la base de datos.

- **Modelo *user* y autenticación por *tokens*.** He modificado el modelo *user* que proporciona Django para poder autenticar por correo electrónico y contraseña. También he añadido algunos atributos más. He sustituido la autenticación por *sesiones* a autenticación por tokens, que permite la comunicación entre *front-end* y *back-end*.

- **Banco de preguntas.** Lo siguiente que he realizado es el banco de preguntas. Para ello, lo primero que hice fue crear en el *back-end* la función que devolvía todas las preguntas asociadas a una asignatura. Después creé la interfaz, donde decidí reutilizar los componentes `TestQuestion` y `TextQuestion`. Luego hice que se pudiera eliminar y editar una pregunta del banco de preguntas. Para ello creé las funciones correspondientes en el *back-end*. Para la edición de las preguntas había pensado en reutilizar los componentes `TestQuestion` y `TextQuestion`, pero descarté esa idea porque requeriría mucho código y haría el componente más difícil de mantener. Por ello decidí

crear los componentes `EditTestQuestion` y `EditTextQuestion`. También creé el componente `VisualizarPregunta`, que gestionaba los componentes anteriormente mencionados en función de si se visualizaba o editaba la pregunta. Al principio la tabla para mostrar el listado de preguntas la había hecho desde cero usando *bootstrap*, pero el director del proyecto sugirió que era mejor mantener uniformidad en la página y que usase el componente *DataTable* que usó Pedro para mostrar la tabla donde el profesor puede revisar los cuestionarios de los alumnos. Pedro encontró la librería que proporcionaba dicho componente y me explicó su funcionamiento para que pudiese usarlo.

- Crear cuestionarios. Después del banco de preguntas, realicé la creación de cuestionarios a través de la interfaz web. Para ello, primero cree el formulario y la validación de sus distintos campos. Inicialmente no se podían ver los detalles de las preguntas hasta que se añadía al cuestionario y el usuario podría introducir la puntuación positiva y negativa en caso de acertar o fallar la pregunta. El director del proyecto sugirió que se deberían poder pre-visualizar las preguntas antes de añadirlas, por lo que decidí integrar el componente `BancoPreguntas` en el apartado de crear cuestionario. Para ello modifiqué su comportamiento para que se pudiera añadir una pregunta del banco de preguntas al cuestionario que se este creando y también que el usuario no pudiera eliminar ni editar las preguntas del banco de preguntas, ya que, de esta manera, el usuario no eliminaría una pregunta por error cuando este quiera añadirla. Además de la interfaz, en el *back-end* he creado la función que permite guardar en la base de datos el cuestionario creado desde la interfaz web.

- Refactorización de código. Según avanzábamos con el desarrollo del proyecto, veía que algunos componentes empezaban a tener mucho código lo cual dificultaba su lectura. Por ello refactoricé y comenté el código del del *front-end*, en concreto los componentes `App, QuestionContainer, TestQuestion y TextQuestion`. También creé varios ficheros en la carpeta *utils* que contienen las funciones que realizan las

peticiones a las distintas rutas de la API.

- Apartado *offline* de la página. He creado en el *front-end* el componente `AvailableOffline`, que muestra los cuestionarios descargados que están almacenados en el *LocalStorage*, para su posterior realización por los usuarios que se encuentren sin conexión. Para ello he reutilizado y modificado el componente `TarjetaCuestionario` creado por Pedro.

- Service worker. El director del proyecto nos sugirió familiarizarnos con los *Service Workers* antes de hacer la aplicación progresiva. Lo primero que hice fue informarme sobre qué eran y cómo funcionaban, para ello leí el libro [3] que nos recomendó el profesor. Este libro no solo explicaba los los *Service Workers*. También proporcionaba ejemplos prácticos y sencillos para poder entender su comportamiento. Una vez que había entendido su funcionamiento, creé una pequeña página web de prueba para simplemente probar como se registra e instala el *Service Worker* en un navegador. Después de investigar y encontrar que *Workbox* es una librería muy utilizada que facilitaba la programación de los *Service Workers*, y la forma de integrar los *Service Workers* con React, empecé a leer la documentación de *Workbox*. Una vez leída la documentación, creé un proyecto nuevo de React para realizar pruebas con el *Service Worker*. Después de realzar las pruebas y conseguir programar el comportamiento del *Service Worker*, lo integré en el proyecto y realicé diversas pruebas para comprobar que funcionaba correctamente.

9.2. Pedro Martínez Gamero

Durante el desarrollo del Trabajo de Fin de Grado, he realizado las siguientes contribuciones:

- Cuestionarios: Con respecto a los cuestionarios he creado las funciones, tanto en el *front-end,* como en el *back-end*, encargadas de realizar los procesos de encriptado y desencriptado de los mismos. Esta tarea la llevé a cabo en las primeras semanas de desarrollo, ya que era una parte primordial del proyecto. También he creado los componentes encargados de mostrar las tarjetas con información sobre los cuestionarios (ver figura 5.6).

- Interfaz de usuario de la aplicación: He sido el encargado de implementar gran parte de la interfaz de usuario de la aplicación. Decidimos que me encargaría de este apartado ya que tengo un mayor conocimiento de tecnologías como, por ejemplo, *bootstrap*, gracias a la cual gran parte de la aplicación tiene una interfaz adaptable (*responsive*). Algunas de las secciones de la aplicación para las que implementé la interfaz son las siguientes:

  - Login.
  - Página de las asignaturas.
  - Páginas de los cuestionarios.
  - Página de subida de ficheros, tanto para subir cuestionarios como preguntas.
  - Página de revisión de las notas.
  - Modals encargados de mostrar los mensajes de error y de confirmación a la hora de realizar ciertas acciones dentro de la aplicación.
  - Menú de navegación.

  Además de desarrollar estas interfaces también ayudé, siempre que mi compañero me lo pedía, a desarrollar o adaptar sus interfaces.

- **Base de datos:** El modelo entidad relación en el que se basó la base de datos (ver figura 4.1) fue creado tanto por mi compañero como por mí. Sin embargo, me encargué de conectar la base de datos a *Django* y de generar los modelos para que se crearan las tablas en la misma. Además, también me encargué de modificar las tablas durante el avance del proyecto.

- **Revisión de las notas por parte del profesor:** Desarrollé el componente necesario en el *front-end* para esta sección. Este componente es `RevisionNotasContainer`. Para su creación hice uso de la biblioteca *react-data-table-component*, gracias a la cual se podían renderizar tablas visualmente atractivas y con varias funciones, como por ejemplo, poder ordenar las filas en función del valor de una columna. También creé la función correspondiente en el *back-end*, la cual hace uso de una consulta SQL (ver figura 4.3). Decidí realizar esta consulta haciendo uso de SQL ya que con el sistema de modelos de *Django* la consulta que teníamos que realizar se dividía en demasiadas consultas. Posteriormente también modificamos esta consulta para que un profesor también pueda ver la nota que ha obtenido a la hora de realizar el cuestionario que él mismo creó.

- **Cambios en la interfaz del banco de preguntas:** Junto con mi compañero Zakaria cambiamos la interfaz de usuario que originalmente tenía el componente `BancoPreguntas`. Para realizar este cambio hicimos uso de una librería llamada *react-data-table-component*, la cual ya había usado previamente en la sección *revisión de notas*. Esta librería nos permitió mejorar y simplificar la interfaz de usuario, además de hacerla adaptable (*responsive*).

- **Subida de cuestionarios mediante un fichero en formato YAML:** En el *front-end* creé el componente `UploadFile`, el cual permite subir un cuestionario mediante un fichero YAML. En el *back-end* también creé la función *upload*, la cual se encarga de almacenar los cuestionarios en la base de datos.

- Subir preguntas mediante un fichero en formato YAML. En el *front-end* creé el componente `UploadQuestions`, el cual se encarga de subir varias preguntas mediante un fichero YAML. En el *back-end* también creé la función *upload-questions* la cual se encarga de almacenar las preguntas en la base de datos.

- Matriculación de alumnos en asignaturas que imparte un profesor: Desarrollé el componente y las funciones en el *back-end* encargadas de gestionar esta funcionalidad. El componente en cuestión es `RegisterContainer`. Las funciones creadas son *enroll-students* y *get-students*. Esta sección hace uso de la librería *react-data-table-component*.

- Generación de códigos QR con los *hashes* de las respuestas: La última tarea que realicé fue la generación de los códigos QR, los cuales contienen un *hash* de las respuestas que ha dado un alumno para un cuestionario. Este código QR es creado haciendo uso de la librería *react-qr-code*, la cual permite generar un QR con una cadena de caracteres. Este código QR contiene una URL con la siguiente estructura:
  http://url de la aplicación/scanner/identificador del usuario/identificador de la asignatura/hash de las respuestas
  Una vez se escanea el código, en caso de que el profesor haya iniciado sesión, se le redireccionará a otra página en la cual se le indicará si el código se ha guardado o no en la base de datos. Además si un alumno intenta escanear este código se le mostrará un mensaje de error. Los componentes necesarios implementados en el *front-end* para desarrollar esta funcionalidad son `QrContainer` e `InsercionManual`. Por último la función que creé *back-end* es *insert-qr*.

- Service Workers: A pesar de que la implementación del *Service Worker* fue llevada a cabo por Zakaria, realicé una labor de investigación sobre ellos, permitiéndome conocer su funcionamiento.

Bibliografía

[1] Sean Amarasinghe. *Service Worker Development Cookbook*. Packt Publishing, 2016. isbn: 1786465299.

[2] Dean Alan Hume. *Progressive Web Apps in Action*. Manning, 2017. isbn: 978-1617294587.

[3] Dennis Sheppard. *Beginning progressive web app development: create a native app experience on the web*. Apress, 2017. isbn: 978-1-4842-3090-9.

[4] Nigel George. *Mastering Django: Core*. GNW Independent Publishing, 2020. isbn: 978-0648884415.

[5] Bootstrap. *Bootstrap Documentation*. url: https://getbootstrap.com/docs/5.2/getting-started/introduction/.

[6] Django Software Foundation. *Django*. url: https://www.djangoproject.com/.

[7] Google. *Workbox*. url: https://developers.google.com/web/tools/workbox/.

[8] JQuery. *JQuery Documentation*. url: https://api.jquery.com/.

[9] JSON. *JSON Documentation*. url: https://www.json.org/json-es.html.

[10] Latex. *Latex Documentation*. url: https://www.latex-project.org/help/documentation/.

[11] Meta. *React*. url: https://es.reactjs.org/.

[12] Mozilla. *JavaScript Documentation*. url: https://developer.mozilla.org/es/docs/Web/API/Document.

[13] MySql. *MySQL Documentation*. url: https://dev.mysql.com/doc/refman/8.0/en/select.html.

[14] Python. *Python Documentation*. url: https://docs.python.org/3/library/.

[15] YAML. *YAML Documentation*. url: https://yaml.org/.