UNIVERSIDAD DE MURCIA
Departamento de Informática y Sistemas

TRABAJO FIN DE GRADO

QuizGenerate

Murcia, a 2 de julio de 2025

Pablo Peris Solano
23329212J
pablo.periss@um.es

Francisco García Sánchez

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

I. Resumen

La creación de cuestionarios educativos se ha convertido en una actividad cada vez más digital, impulsada por herramientas online que permiten crear, realizar y compartir exámenes. Sin embargo, muchas de estas soluciones no ofrecen una experiencia adaptada a las necesidades educativas actuales. Ante esta situación, se ha desarrollado **QuizGenerate**, una aplicación web que permite a los usuarios crear, editar, realizar e incluso generar mediante inteligencia artificial (IA) cuestionarios tipo test. Esta plataforma no solo busca facilitar la creación de cuestionarios online por parte de los docentes, sino que también fomenta la autoevaluación de los alumnos que se preparan para un examen.

A partir del análisis de otras aplicaciones existentes, como Kahoot!, Revisely y Algor Education, se ha identificado una serie de limitaciones que QuizGenerate pretende superar. Kahoot! permite explorar cuestionarios creados por otros usuarios, pero no ofrece gratuitamente la creación de estos mediante IA. Revisely sí permite gratuitamente la generación de cuestionarios mediante IA, pero de forma muy limitada, además de que no ofrece la posibilidad de explorar cuestionarios de otras personas. Algor Education también utiliza IA, ya que permite generar contenido a partir de un documento, sin embargo, no solo genera cuestionarios, sino también otros materiales como mapas conceptuales y preguntas de verdadero o falso, lo que se traduce en un desperdicio de recursos. QuizGenerate propone una solución equilibrada, que integra las mejores características de estas plataformas, optimizando el uso de recursos y simplificando la interfaz de usuario.

La aplicación sigue una arquitectura cliente-servidor, compuesta por un *front-end* y un *back-end* que se comunican a través de una API RESTful. El *back-end* se ha desarrollado con código Python en **Django**, utilizando Django Rest Framework, mientras que la persistencia de datos se gestiona mediante PostgreSQL, con la librería psycopg2. Por su parte, el *front-end* está construido con **React** y JavaScript, utilizando Tailwind CSS y la librería Shadcn/ui para garantizar una interfaz adaptativa, moderna y coherente en distintos dispositivos.

El trabajo se ha llevado a cabo siguiendo una metodología estructurada en cinco fases: identificación de tecnologías apropiadas, análisis de soluciones actuales, desarrollo y validación del sistema, despliegue de la aplicación y documentación del trabajo. En QuizGenerate, los usuarios no registrados solo tendrán acceso a explorar y realizar cuestionarios públicos, y los usuarios registrados podrán crear, editar, realizar y generar cuestionarios tipo test mediante IA.

La arquitectura del sistema sigue un enfoque de múltiples capas: la capa de presentación para mostrar los datos en una interfaz, la de aplicación para procesarlos y la capa de persistencia para almacenarlos en una base de datos. El sistema de autenticación usa JSON Web Tokens (JWT) almacenados en cookies *HttpOnly* para una mayor seguridad.

QuizGenerate ha sido desplegado usando Docker en un servidor de **Hetzner Cloud**. Además, durante el desarrollo se realizaron pruebas continuas para verificar el correcto funcionamiento del sistema. Como posibles mejoras futuras se plantea el acceso por roles a la aplicación (administrador, profesores y alumnos), un sistema de notificaciones, la posibilidad de crear cuestionarios de otros tipos, el soporte multilingüe, la financiación mediante publicidad, la capacidad de analizar documentos en otros formatos, y la posibilidad de consultar un historial de puntuaciones de los cuestionarios que se han realizado.

II. Extended abstract

The creation of educational quizzes has become an increasingly digital activity, driven by online tools that allow users to create, take, and share exams. However, many of these solutions fail to provide an experience tailored to today's educational needs. In response to this situation, QuizGenerate has been developed: a web application that enables users to create, edit, take, and even generate multiple-choice quizzes using artificial intelligence (AI). This platform not only aims to simplify online quiz creation for educators but also encourages self-assessment among students preparing for exams.

By analyzing existing applications like Kahoot!, Revisely, and Algor Education, a series of limitations were identified, which QuizGenerate seeks to overcome. While Kahoot! allows users to explore quizzes created by others, its AI-powered question generator is restricted to paid expensive plans. This pricing model creates a significant barrier for students, especially those from low-income backgrounds or regions with limited educational budgets. Revisely does enable AI-generated quizzes for free, but in a very limited way, and it lacks the option to browse quizzes made by other users. Algor Education also uses AI to generate content from documents, but it produces not only quizzes but also other materials like concept maps and true/false questions, resulting in resource inefficiency. QuizGenerate is designed to let users not only create manual multiple-choice quizzes but also generate them automatically using AI by processing uploaded text or PDF documents, similar to features in Revisely or Algor Education. Additionally, the platform enables users to explore and filter public quizzes created by others, mirroring functionality found in Kahoot!

To achieve the proposed idea, a client-server architecture was developed, made up of two main components: a front-end and a back-end, which communicate through HTTP requests. Each component was built using a specific set of programming languages, frameworks and libraries suited to its purpose, ensuring seamless integration and optimal performance.

On the one hand, the back-end was developed using Python as the programming language, combined with Django and the extension Django Rest Framework (DRF), a decision aligned with industry standards for AI-powered web applications. This stack was chosen for its scalability, built-in security features, and seamless integration with machine learning models. For data persistence, the project uses two relational databases. During development, the system uses SQLite as Django's default database for its fast transaction processing capabilities. For production, the system utilizes PostgreSQL with the psycopg2 library for efficient Python integration. In addition, PDFs are processed using PyMuPDF for high-fidelity text extraction, followed by OpenAI API calls to transform the cleaned text into quiz questions, optimizing both accuracy and operational costs.

On the other hand, the front-end was developed using React as the framework and JavaScript as the programming language. React enables the creation of reusable components and promotes a modular structure, while JavaScript enables dynamic interactivity, asynchronous operations, and DOM manipulation in web browsers. HTML is used to define the structure of the interface, and both Tailwind CSS and the Shadcn/ui library were included to provide a clean, responsive, and consistent design.

Additionally, libraries such as Sonner for toast notifications and Motion for animated quiz interactions were integrated, significantly enhancing the user experience. This combination ensures a smooth and intuitive user experience across different devices and screen sizes.

It could also be considered to mention another layer: the gateway layer. The gateway provides a simple way to connect browser clients to the app securely via HTTPS, using Nginx as the underlying technology.

The methodology applied in the project is structured into five main phases: (i) identification of appropriate technologies, which involves researching existing solutions and the required technologies for front-end development (React, JavaScript, Tailwind CSS, Shadcn/ui UI, Sonner, Motion, etc.) and back-end development (Python, Django, Django Rest Framework, RESTful services, PostgreSQL, JWT, PyMuPDF, OpenAI, etc.), among other relevant tools and libraries; (ii) analysis of existing solutions, involving comprehensive research into recent educational platforms and identifying opportunities for QuizGenerate to enhance their features; (iii) system development, involving building the database schema, implementing back-end business logic, developing the front-end user interface, and integrating both parts; (iv) application deployment, the process of making the web application publicly accessible on the internet; (v) and project documentation, which involves documenting software components, preparing the thesis, and preparing the final presentation for the thesis defence.

After defining the methodology, the system scope and requirements were established through natural language use cases. The application distinguishes between unregistered and registered users. While unregistered users can browse and take public quizzes, registered users gain access to additional features including quiz creation, editing, and AI-powered generation.

The system architecture follows a multi-layered approach: the presentation layer for displaying data in the interface, the application layer for processing data, and the persistence layer for storing it in a database. Additionally, it is possible to consider a fourth layer, which is the gate-away layer, that allows the user access to the API or to the front-end, depending on the URL path.

The application's domain model centers on the `CustomUser` entity, which extends Django's default `AbstractUser` model to support additional fields (e.g., profile photos) while preserving core authentication functionality. Users can create quizzes, where each `Quiz` contains multiple Questions, and each `Question` comprises multiple `Answer` entities.

To enable communication between the front-end and back-end, a RESTful API has been designed with clearly defined endpoints. Each endpoint corresponds to one of the system entities and uses standard HTTP methods to read, create, update, and delete resources. This fully decoupled architecture allows the system to support multiple user interfaces and third-party clients across different platforms.

A navigation diagram was created to map navigation flows between pages, ensuring a smooth experience. This early design process helped establish a consistent and user-friendly interface adaptable to different screen sizes.

Following the design phase, a modular back-end structure was implemented with these core directories: `.venv` for Python dependency isolation, `apps/` (containing `quizzes/` and `users/` subdirectories) for application logic, `media/` for user-uploaded assets like profile photos, and `projectTFG/` housing the Django project configuration (settings, URLs, and WSGI setup).

The data processing logic for user-uploaded PDF documents is handled within the `quizzes` app, where PDFs are first recognized and converted to plain text using PyMuPDF, specifically processing only the user-selected pages. This extracted text then serves as input for the OpenAI API (configured via the .env file, which also specifies the AI engine), where the request must clearly define the response format rules and include all user parameters, including the desired number of questions, answer options per question, quiz language, and the foundational prompt (which can be either the processed PDF text or direct user input). The system then uses this structured request to generate the quiz content according to the specified requirements, completing the transformation from raw document to customized assessment.

Data persistence is handled by PostgreSQL, containerized via Docker and accessed through the psycopg2 adapter. Django's ORM (Object-Relational Mapper) provides an additional abstraction layer, simplifying database operations while maintaining query efficiency.

The control flow in the back-end starts when an HTTP request is received from the client. The appropriate controller extracts the necessary parameters (from the URL, request body, or query), validates them against predefined schemas, and checks user authorization when applicable. Once validated, it calls the corresponding model method to execute the required database operations. After these operations complete, the controller builds and sends an HTTP response with the appropriate status code and JSON data. Error handling is also managed by Django Rest Framework, which responds with clear error messages and suitable HTTP status codes such as `400`, `404`, or `500`.

The authentication system is implemented using JWTs (JSON Web Tokens), stored as HttpOnly cookies, providing secure, stateless user sessions. Users authenticate through a login endpoint which issues both an access token and a refresh token. When the access token expires, the refresh token allows the client to request a new one, enhancing both security and usability without compromising user experience. Additionally, a middleware is used to validate tokens and restrict access to protected routes, ensuring that users can only perform actions allowed by their authentication status.

The front-end of QuizGenerate is structured as a SPA (Single Page Application) using React. Each part of the interface is divided into reusable components located in structured folders such as `components`, `constants`, `context`, `hooks`, `lib` and `pages`, ensuring maintainability and scalability. JavaScript enables dynamic interactivity in web browsers, while client-side navigation is managed through React Router. Tailwind CSS and Shadcn/ui provide a responsive and visually consistent design across different devices and screen sizes. Communication with the back-end is managed through services that use the Fetch API to encapsulate HTTP requests.

In the mobile version, the application's responsive design causes the header navigation bar to collapse into a hamburger menu, which expands when clicking the three-bar icon. The layout of all other pages also adapts to the device's screen width by reorganizing the content.

The interface is organized around a `Layout` component, which ensures consistent structure across all pages. This component wraps the main routes and provides persistent elements, such as a header at the top, a footer at the bottom, and the main content in the middle. The application also features several other noteworthy pages, including the `Home` page with its carousel section that provides a brief usage tutorial. The `App` component handles routing, connecting all pages with their respective URLs. For protected routes requiring authentication, a `PrivateRoute` component prevents unauthorized access.

Another fundamental component is `main`, which serves as the application's entry point where the `App` component is mounted to the DOM. Other notable components include `AuthProvider`, which supplies authentication context to the entire app (current user, login, logout, etc.); `Toaster`, required for displaying floating notifications from the Sonner library; and essential React components like `StrictMode` and `BrowserRouter` that are necessary for any React project.

Since this application is fundamentally quiz-based, user experience is particularly important for this purpose. To enhance interactivity, the Motion library (a React animation library) is implemented. This enables smooth transitions between questions during quiz sessions and provides dynamic feedback messages for user responses. It is worth noting that each quiz has a time limit. When time expires, the quiz is automatically submitted and the application provides feedback.

Furthermore, it has implemented a gateway service using Nginx, a technology that allows users to access the app securely via HTTPS, enabled by a Let’s Encrypt certificate. The gateway also provides API access while redirecting HTTP requests to HTTPS, ensuring all connections remain secure.

During the development, the system was regularly tested to ensure that each component worked correctly in different scenarios. For the back-end, endpoints were tested using Postman to verify that each route returned the expected results, including handling invalid inputs and authentication errors. For the front-end, manual testing was performed on both desktop and mobile views, simulating user interactions and verifying interface responsiveness. The browser's developer tools were used to emulate different screen sizes and inspect real- time WebSocket communication.

The application is deployed using Docker, leveraging its potential and ease of use, with each component running in isolated containers containing all necessary dependencies. The production environment utilizes these Docker images: postgres:14.11 for PostgreSQL, python:3.12-slim for the back-end API, node:22-alpine for front-end building, and nginx:alpine as a gateway. Nginx functions both as a front-end static file server and as a reverse proxy to route requests to either the API or front-end based on URL patterns. A CAX11 Hetzner Cloud server (2 CPUs, 4GB RAM, 40GB SSD) is provisioned for continuous service at €3.79/month. For

public accessibility, the quizgenerate.xyz domain (a €1 .xyz domain registered through Namecheap) is linked to the Hetzner server.

In conclusion, the development of QuizGenerate, a responsive web app for creating and taking quizzes, has successfully achieved its core objectives, delivering an intuitive and visually engaging user experience. While the current version focuses on essential functionality, it establishes a strong foundation for future enhancements, including role-based access (administrator, teachers and students), a notification system, support for additional quiz formats, multilingual features, funding through advertising, expanded document analysis capabilities, and the option to view a history of scores from completed quizzes. With further refinement, QuizGenerate has the potential to grow into a more robust and competitive player in the online educational quiz landscape.

1 Introducción

En el panorama educativo actual, la creación de cuestionarios digitales se ha convertido en una herramienta indispensable para docentes e instituciones [1]. Los cuestionarios no solo permiten evaluar el progreso de los estudiantes, sino que también sirven como instrumentos de aprendizaje activo y retroalimentación inmediata. Sin embargo, el proceso de diseño manual de evaluaciones sigue siendo un desafío debido a que consume una gran cantidad de tiempo.

Un estudio publicado en mayo de 2024 señala que la falta de capacitación adecuada y la resistencia al cambio limitan la capacidad de los docentes para integrar tecnologías de manera efectiva en su práctica educativa [2]. Es aquí donde la inteligencia artificial (IA) emerge como un aliado transformador, capaz de generar cuestionarios fácilmente a partir de contenido educativo en cuestión de segundos.

En este contexto, se presenta **QuizGenerate**, una plataforma online que combina la facilidad de uso con el poder de la IA para revolucionar la creación de cuestionarios. A diferencia de otras herramientas, QuizGenerate permite su creación tanto manual como automática, integrando soluciones de IA para generar las preguntas a partir de documentos PDF.

La plataforma está diseñada para ser intuitiva y accesible, permitiendo a los usuarios registrarse, personalizar su perfil, explorar y realizar cuestionarios publicados por otras personas, crear sus propios cuestionarios o, incluso, generarlos mediante IA. Estas funcionalidades consolidan a QuizGenerate como una solución integral, precisa y accesible, allanando el camino hacia un futuro donde la tecnología y la pedagogía avanzan de la mano.

Es interesante señalar el origen del nombre de la aplicación, QuizGenerate, que es una combinación de las palabras en inglés "*quiz*" (cuestionario) y "*generate*" (generar).

Este documento ha sido estructurado de la siguiente forma:

- **Capítulo 2 (Estado del arte):** se analiza el contexto previo al desarrollo de este proyecto, detallando las soluciones existentes en el mercado, así como sus principales fortalezas y debilidades. Además, se describen las tecnologías empleadas para el desarrollo de la plataforma.
- **Capítulo 3 (Análisis de objetivos y metodología):** se describen los objetivos que se persiguen con el desarrollo de este proyecto, así como la metodología empleada para alcanzar estos objetivos.
- **Capítulo 4 (Diseño y resolución del trabajo realizado):** se presenta el análisis de los casos de uso a desarrollar, el diseño y la arquitectura de la aplicación, la implementación detallada tanto del *back-end* como del *front-end*, las pruebas realizadas al sistema y el proceso de despliegue.
- **Capítulo 5 (Conclusiones y vías futuras):** se exponen las conclusiones del desarrollo de este proyecto, evaluando si se han alcanzado los objetivos propuestos. Además, se discuten posibles mejoras o funcionalidades futuras que podrían incrementar el valor de la plataforma.

2 Estado del arte

El objetivo en este capítulo es analizar las prestaciones ofrecidas por las aplicaciones actuales dedicadas a la realización y creación de cuestionarios online usando IA. El capítulo también incluye una completa descripción de las tecnologías empleadas en el desarrollo de este proyecto.

2.1 Análisis de la situación de partida

En la actualidad, existen numerosas plataformas online que permiten a los usuarios crear y realizar cuestionarios tipo test, algunas incluso integrando soluciones de IA para generar las preguntas. Para el análisis comparativo de alternativas de este tipo de aplicaciones, se han seleccionado tres plataformas: Kahoot!, Revisely y Algor Education. La elección de estas herramientas se ha basado en una combinación de criterios relevantes en el contexto educativo y tecnológico actual: corrección automatizada de los cuestionarios, accesibilidad multiplataforma, popularidad en entornos educativos, integración con la IA y visibilidad (publicidad) reciente a través de redes sociales.

En la evaluación, se han considerado las siguientes características clave:

- Funcionalidad: explorar la funcionalidad que ofrece la herramienta.
- Multiplataforma: disponibilidad en móvil (iOS y Android) y en su versión web.
- Experiencia de usuario: facilidad de uso, diseño visual y organización de la información.
- Precio: alcance del plan gratuito y disponibilidad de otros planes mensuales.

Una de las plataformas más conocidas en este ámbito es **Kahoot!** [3], cuya popularidad se ve potenciada gracias a la facilidad de uso y adopción como método de aprendizaje en diversos centros educativos. Precisamente por esto, el diseño de la aplicación puede resultar algo infantil (ver Figura 1). Kahoot! es una aplicación web muy asentada, que además está disponible para descargar en iOS y Android. Esta plataforma es perfecta si lo que se pretende es explorar cuestionarios de todo tipo, ya que dispone de infinidad de cuestionarios y canales enteros enfocados en distintas áreas del conocimiento. Sin embargo, para crear un cuestionario, el usuario debe ir escribiendo manualmente todas las preguntas, lo que resulta algo tedioso. Gracias al avance de la IA, hoy en día es posible la generación automática de preguntas y respuestas a partir de un texto.

Kahoot! no ofrece una solución gratuita que integre IA para generar preguntas a partir de un archivo PDF. Sin embargo, el plan *Gold* para estudiantes posibilita esta función mediante el acceso a *AI Kahoot! Generator*, por un precio de 30.49€ al mes, lo que resulta un presupuesto descartable para la mayoría de estudiantes.

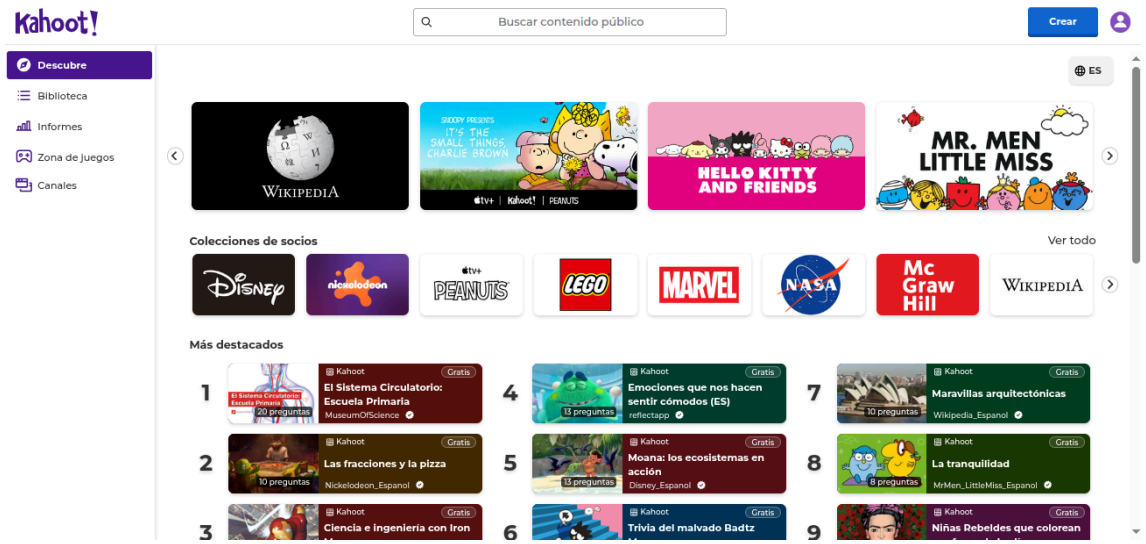

Figura 1. Aplicación web de Kahoot!

Otra alternativa cada vez más popular es **Revisely** [4], la cual está más enfocada en integrar la potencia de la IA generativa para la generación de cuestionarios. Esta plataforma es bastante más interesante, ya que desde la sección *AI Quiz Generator* se pueden generar cuestionarios mediante IA a partir de textos, documentos, imágenes y vídeos. Esta aplicación tiene un diseño web minimalista, intuitivo y profesional (ver Figura 2), pero no está disponible para descargar ni en iOS ni en Android. Además, la aplicación responde de forma eficaz cuando se pone a prueba su función de generar cuestionarios a partir de un archivo PDF, ya que es posible seleccionar las páginas que se desean analizar, el número de preguntas y opciones de respuesta, y el idioma en el que se generará el cuestionario.

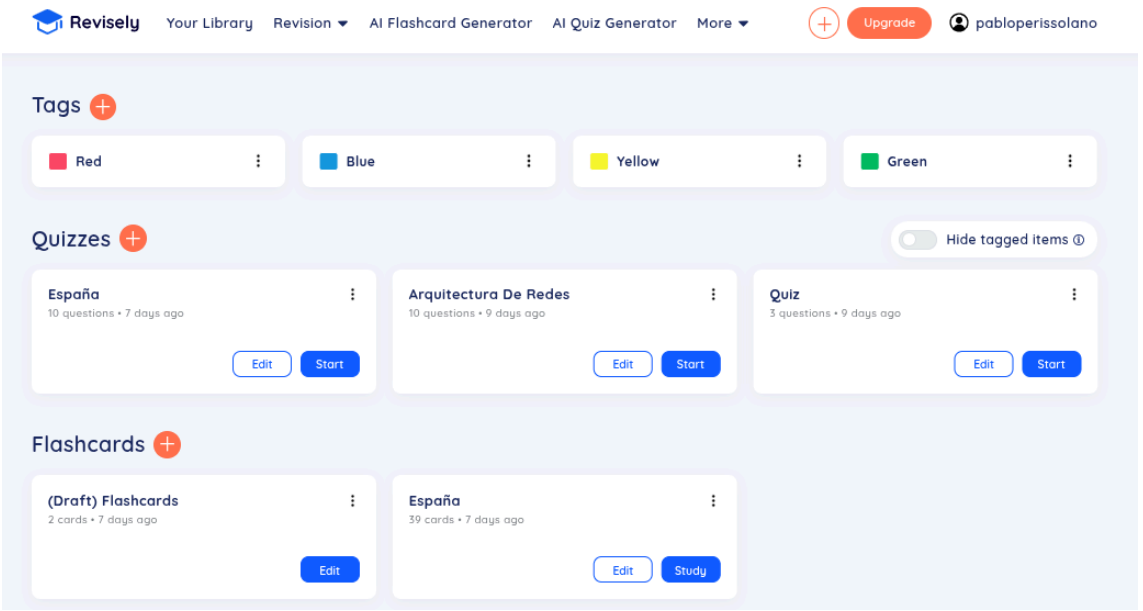

Figura 2. Aplicación web de Revisely

No obstante, aunque es una aplicación que funciona bastante bien, el plan gratuito está demasiado limitado, ya que únicamente se pueden analizar 5 páginas de un PDF. Por ello, si se necesita acceder frecuentemente a las funciones de IA, habrá que pagar 11€ al mes para mejorar el plan, ya que así se podrá analizar como máximo 200 páginas por documento. Además, otro punto desfavorable a tener en cuenta es que Revisely no permite explorar cuestionarios públicos, ya que ni siquiera tiene buscador para encontrar los cuestionarios creados por otras personas. La única forma de realizar un cuestionario creado por otro usuario es a través de su enlace, lo que resulta poco funcional si lo que se pretende con la aplicación es explorar otras creaciones.

Por último, otra plataforma que se está haciendo muy popular en redes sociales durante estos últimos años es **Algor Education** [5], que desde finales de 2021 es capaz de generar cuestionarios, *flashcards* y mapas conceptuales a partir del material de estudio adjuntado. El diseño de Algor Education es robusto y profesional (ver Figura 3), y aunque tenga muchas funcionalidades, el usuario es capaz de intuir dónde acceder para realizar la tarea deseada. Al igual que Revisely, esta plataforma funciona en versión web, pero la aplicación no es descargable en iOS ni en Android. En cuanto a su funcionalidad, utiliza un sistema de créditos de IA, los cuales se van consumiendo a medida que se usan estas funciones.

Un punto débil de Algor Education es que no es posible generar únicamente un cuestionario a partir de un archivo PDF, sino que cuando este se adjunta, solo da la opción de generar todo un bloque de contenidos relacionados, tales como: mapa conceptual, bloque de texto, *flashcard*, preguntas y respuestas y, por último, el cuestionario. Esto provoca que la IA de la aplicación esté muy poco optimizada, ya que al generar se centra en tareas que posiblemente el usuario ni siquiera necesite, provocando un desperdicio de créditos IA. Otro punto desfavorable a tener en cuenta es la gran limitación del plan gratuito, el cual solo ofrece 30 créditos de prueba y capacidad de analizar 15.000 caracteres, que equivalen aproximadamente a 4-5 páginas de un PDF. Por ello, si se desea obtener más créditos, se deberá mejorar el plan pagando 6,99€ al mes.

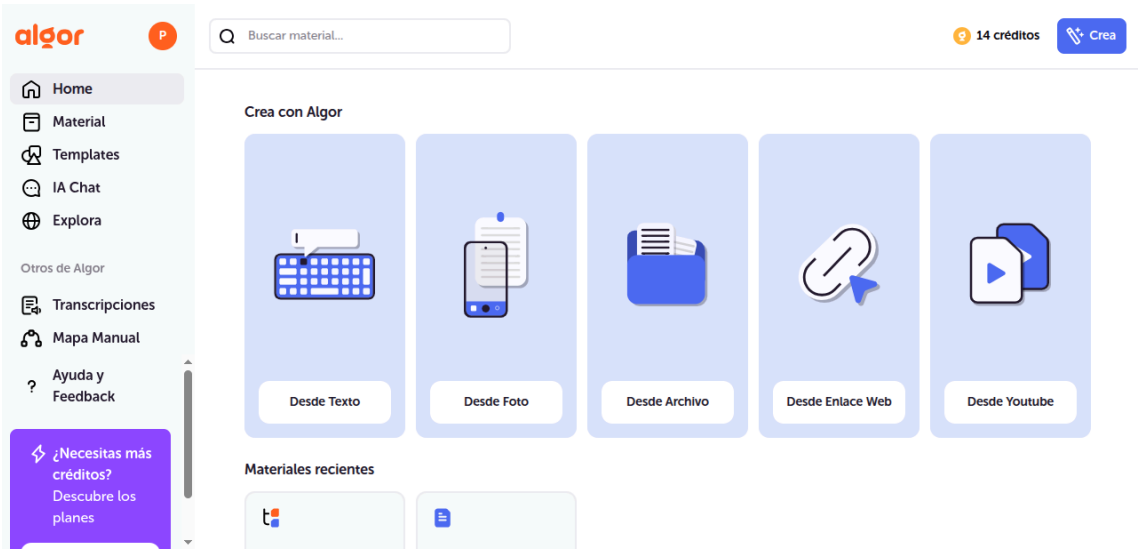

Figura 3. Aplicación web de Algor Education

En este trabajo se describe el desarrollo de **QuizGenerate**, una aplicación web que pretende superar las limitaciones de las alternativas mencionadas combinando lo mejor de cada una. Para empezar, se podrá acceder al portal de la aplicación desde ordenador (pantalla grande) y dispositivos móviles, gracias a un diseño web adaptable. Además, se pretende que este diseño sea minimalista y que el usuario tenga una experiencia sencilla e intuitiva, evitando que se sienta abrumado por un exceso de información. Por otro lado, QuizGenerate pretende que los usuarios, además de poder crear cuestionarios tipo test manualmente, también puedan generarlos mediante IA, a partir de un texto o un documento PDF adjunto, tal como se puede hacer en Revisely o en Algor Education. Al mismo tiempo, con esta aplicación también se pretende que los usuarios puedan explorar y filtrar cuestionarios públicos creados por otras personas, como también se puede hacer en la web de Kahoot!. Otro punto diferencial frente a las alternativas comentadas es que se pretende crear una plataforma de código abierto y gratuita, es decir, que no requiera pagar un plan mensual a la hora de analizar una gran cantidad de páginas de un documento, además de que la API (Interfaz de Programación de Aplicaciones) esté disponible y abierta al público. Esta afirmación puede resultar un tanto ambiciosa, sin embargo, si la aplicación escala comercialmente se podría financiar mediante publicidad. En la Tabla 1 se muestra un resumen comparativo de las soluciones analizadas.

Tabla 1. Resumen comparativo de las soluciones analizadas

Característica	Kahoot!	Revisely	Algor Education	QuizGenerate
Funcionalidad	Insuficiente (no permite funciones IA gratuitamente)	Muy eficaz pero limitada (solo permite 5 páginas por documento)	Ineficiente y limitada (desperdicia créditos IA y solo permite 15.000 caracteres)	Limitado a la capacidad máxima del modelo IA utilizado (suele ser suficiente)
Multiplataforma	Totalmente	Solo web	Solo web	Solo web
Experiencia de usuario	Muy completa	Simple pero efectiva	Muy completa	Simple pero efectiva
Precio	30,49€	11€	6,99€	Gratis

2.2 Tecnologías empleadas

En este apartado se describen los lenguajes de programación, entornos de desarrollo, librerías, *frameworks* y herramientas que se han decidido emplear para desarrollar tanto el *back-end* como el *front-end* de la aplicación.

2.2.1 Back-end

Para el desarrollo de la aplicación del lado del servidor, se ha optado por el lenguaje de programación **Python** [6]. Este lenguaje es utilizado para desarrollar todo tipo de aplicaciones, debido a que es un lenguaje muy versátil y fácil de aprender. Con el avance de la IA todavía se

ha vuelto más popular, llegando a convertirse en el lenguaje principal para desarrollar aplicaciones que integran IA [7], como en este caso.

El principal *framework* para este lenguaje es **Django** [8], aunque, para desarrollar esta aplicación se utilizará una extensión llamada **Django Rest Framework (DRF)** [9], que está enfocada en el desarrollo de APIs RESTful. Esta extensión también incorpora un enrutador que se utiliza para registrar las URL que se manejan en la API implementada. Además, la librería **Simple JWT** [10] de Django Rest Framework facilita mucho la gestión del acceso al *back-end* por parte de los usuarios, ya que hace uso de un sistema de autenticación basado en tokens que será gestionado mediante las *cookies* del navegador. Para la gestión de dependencias del *back-end*, lo que se debe hacer en proyectos Python es crear un entorno virtual para instalar las librerías que sean necesarias mediante la herramienta **pip** [11].

En cuanto a la base de datos, realmente se han utilizado dos: **SQLite** [12] para el desarrollo y **PostgreSQL** [13] para producción y el despliegue de la aplicación. Ambos son populares sistemas de gestión de bases de datos relacionales. La principal razón de esta elección es el soporte nativo que proporciona Django con las bases de datos relacionales, ya que se integra directamente con su ORM (Mapeo Objeto-Relacional) y sus métodos CRUD (Crear, Leer, Actualizar y Eliminar). En cambio, si con Django se quisiera usar una base de datos no relacional, sería necesario instalar librerías adicionales. La razón de usar dos bases de datos diferentes para desarrollo y producción es que SQLite es muy ligera, pero de escaso rendimiento para conexiones simultáneas, lo que puede reflejarse en lentitud cuando hay varios usuarios conectados a la vez. En cambio, PostgreSQL es una base de datos asentada, robusta y más escalable, por lo que está más preparada y es más recomendable en entornos de producción.

Para la funcionalidad de generación de cuestionarios a partir de un documento PDF, se ha hecho uso de la herramienta **PyMuPDF** [14], la cual es capaz de procesar un PDF y extraer el texto que interese del documento. Una vez extraído el texto del documento, la aplicación debe generar las preguntas tipo test automáticamente. Para ello, se utiliza la librería **OpenAI** [15] con el fin de realizar fácilmente las peticiones a la API del modelo de IA configurado, enviándole también los parámetros que introduzca el usuario. En el caso de QuizGenerate, se utiliza la API de **Deep Seek** [16] para generar las preguntas, ya que su coste es muy económico en comparación con el resto de los modelos de IA (GPT-4, Claude, etc.), además de que los resultados que se obtienen son formidables.

2.2.2 Front-end

Para la realización del *front-end* se ha usado el lenguaje **JavaScript** [17], con el fin de crear una página web dinámica e interactiva. Este se ha combinado con la popular librería **React** [18], que facilita mucho la creación de aplicaciones web gracias a un diseño basado en componentes reutilizables. Además, gracias al **Virtual DOM** [19] de React, se optimiza mucho el renderizado de la página web, ya que cuando en esta surgen cambios, no se refresca la página completa, sino únicamente el componente que ha cambiado. Por otro lado, se ha utilizado **Vite** [20] como herramienta de construcción, con el fin de orquestar el desarrollo del proyecto.

Por supuesto, para que todas estas tecnologías funcionen correctamente, el proyecto *front-end* ha sido desarrollado en el entorno de ejecución **Node.js** [21], un entorno de JavaScript que es

el encargado de gestionar las dependencias y de arrancar el proyecto. Por otro lado, se ha utilizado **HTML** [22] para definir la estructura de la página web. Para aplicar los estilos se ha utilizado **CSS** [23] y **Tailwind CSS** [24], que han sido de gran utilidad para mejorar el aspecto visual de la aplicación y hacer un diseño adaptable (*responsive*).

Como librería de componentes se ha utilizado **Shadcn/ui** [25], una popular librería creada para React que también utiliza Tailwind CSS para hacer los componentes adaptables, lo que ha facilitado en gran medida el desarrollo del proyecto. Otras librerías de React que se han utilizado para mejorar la experiencia de usuario de la aplicación son: **Lucide React** [26] para los iconos, **Sonner** [27] para las notificaciones flotantes, y **Motion** [28] para generar movimiento en las preguntas a la hora de resolver los *quizzes*.

2.2.3 Miscelánea

En este apartado se hará mención a las tecnologías secundarias que también han servido de ayuda para desarrollar el proyecto. Comenzaremos por el sistema operativo, el cual ha sido **Ubuntu 24.04.1 LTS** [29], ya que para desarrollar aplicaciones es muy eficiente y viene con Python preinstalado. El entorno de desarrollo que se ha utilizado en todo momento ha sido **Visual Studio Code** [30] que, junto con extensiones y **GitHub Copilot** [31], ha permitido acelerar en gran medida el proceso de desarrollo de la aplicación. De igual forma, se ha utilizado **GitHub** [32] para controlar las versiones del código. Con el fin de acelerar el desarrollo del documento también se ha hecho uso de herramientas de IA generativa, tales como **Chat GPT** [33].

Por otro lado, para probar el funcionamiento de las peticiones HTTP (Protocolo de Transferencia de Hipertexto) a la API *(back-end)*, se ha hecho uso de **Postman** [34], una popular herramienta utilizada para comprobar que las respuestas del servidor eran correctas en todo momento. Además, se ha utilizado la herramienta **Draw** [35] para representar los diagramas de casos de uso, el diagrama de la arquitectura de la aplicación, el del modelo de dominio y el de navegación. Por otra parte, se ha utilizado la herramienta **Graphviz** [36] para generar el diagrama de la base de datos.

Para el despliegue de la aplicación en producción, se ha utilizado **Nginx** [37] para definir el punto de entrada (a modo de pasarela), y **Docker** [38] para orquestar su ejecución mediante contenedores independientes.

3 Análisis de objetivos y metodología

En esta sección se describen los objetivos que se persiguen con el desarrollo de este proyecto, así como la metodología empleada (tareas y temporalización) para alcanzar estos objetivos.

3.1 Objetivos

Se parte de la premisa de que existe una necesidad creciente de modernizar el proceso de aprendizaje para hacerlo más eficiente y optimizar los tiempos de estudio. Por ello, el objetivo general que se plantea en este Trabajo Fin de Grado (TFG) es el desarrollo de una aplicación Web donde los usuarios puedan crear y realizar exámenes tipo test en línea. Este objetivo se puede dividir en los siguientes subobjetivos específicos:

(OB1) Identificar las tecnologías y herramientas más apropiadas para acometer el trabajo
(OB2) Determinar las principales propiedades y carencias de las soluciones actuales en el dominio de la aplicación
(OB3) Desarrollar una aplicación que subsane las carencias identificadas satisfaciendo los principales estándares de calidad en el desarrollo de software
(OB4) Desplegar la aplicación en un escenario de producción
(OB5) Documentar el trabajo realizado

Es importante señalar que este trabajo persigue elaborar un prototipo de aplicación que sea útil a la hora de estudiar. Por ello, esta puede ser mejorada en el futuro con más funcionalidades. Por otro lado, también se establece como objetivo de este TFG la realización de una memoria en la que se describen las principales características del trabajo realizado.

3.2 Metodología

A continuación, se enumeran las etapas en las que se ha dividido la realización del TFG. Para cada tarea se establecen las subtareas específicas y los objetivos abordados.

Tarea 1. Análisis preliminar
Objetivo(s): OB1, OB2

1.1. Estudiar el estado del arte sobre herramientas para la creación y realización de cuestionarios tipo test en línea
1.2. Estudiar las tecnologías para el desarrollo de un *front-end* adaptable (*responsive*): HTML5, CSS, TailWind CSS, JavaScript, Vite, React, Shadcn/ui, etc.
1.3. Estudiar las tecnologías para el desarrollo del *back-end*: servicios REST, Django Rest Framework (DRF), Python, PostgreSQL (base de datos).
1.4. Estudiar librerías de IA y de procesamiento PDF en Python: PyMuPDF, OpenAI, etc.

Tarea 2. Análisis y diseño del sistema
Objetivo(s): OB2, OB3

2.1. Definir el alcance del sistema
2.2. Establecer los requisitos funcionales y no funcionales del sistema
2.3. Diseñar la base de datos del sistema
2.4. Diseñar el componente *back-end* del sistema a partir de servicios REST
2.5. Diseñar el componente *front-end* del sistema siguiendo un modelo responsivo.

Tarea 3. Desarrollo del sistema
Objetivo(s): OB3

3.1. Crear el esquema de la base de datos

3.2. Desarrollar los servicios que constituyen la lógica de negocio (*back-end*)
3.3. Desarrollar los componentes que constituyen la interfaz de usuario (*front-end*)
3.4. Conectar el *front-end* y el *back-end*

Tarea 4. Verificación y validación
Objetivo(s): OB3, OB4
4.1. Verificar el sistema frente a los requisitos especificados
4.2. Definir los casos de prueba
4.3. Poblar la base de datos con conjunto de datos necesarios para las pruebas
4.4. Validar el sistema a partir de los casos de prueba definidos
4.5. Desplegar el sistema en un entorno de producción

Tarea 5. Documentación del sistema
Objetivo(s): OB5
5.1. Revisiones periódicas con el tutor para revisar el estado del Trabajo Fin de Grado
5.2. Documentar internamente cada uno de los componentes *software* desarrollados
5.3. Elaborar la memoria del Trabajo Fin de Grado
5.4. Elaborar la presentación del Trabajo Fin de Grado

En la Tabla 2 se muestra una estimación de la planificación mensual de cada tarea, de forma que estén repartidas en el tiempo. Además, como podemos observar, la tarea del desarrollo del sistema (T3) es una de las que requiere mayor cantidad de tiempo debido a su dificultad.

En la Tabla 3 se puede ver la estimación de horas dedicadas a cada tarea junto con sus horas reales. El propósito de esta tabla es gestionar adecuadamente el tiempo que se debe dedicar a cada tarea.

Tabla 2. Plan de trabajo: planificación mensual de las tareas

Tarea	01/25	02/25	03/25	04/25	05/25	06/25	07/25
T1							
1.1	X						
1.2	X						
1.3	X						
1.4	X						
T2							
2.1	X						
2.2	X						
2.3	X						
2.4	X	X					
2.5		X	X				
T3							
3.1	X						
3.2	X	X	X	X			
3.3		X	X	X	X		
3.4						X	
T4							
4.1						X	
4.2						X	
4.3						X	

4.4						X	
4.5						X	
T5							
5.1	X	X	X	X	X	X	X
5.2	X	X	X	X	X	X	
5.3	X	X	X	X	X	X	
5.4							X

Tabla 3. Plan de trabajo: estimación de horas por tarea

Tarea	Descripción	Horas estimadas	Horas reales
T1	**Análisis preliminar**	**20**	**20**
1.1	Estudiar estado del arte	2	1
1.2	Estudio tecnologías *front-end*	8	10
1.3	Estudio tecnologías *back-end*	8	8
1.4	Estudio procesamiento PDF con IA	2	1
T2	**Análisis/diseño sistema**	**50**	**48**
2.1	Definir alcance del sistema	2	1
2.2	Establecer requisitos	2	1
2.3	Diseñar la BBDD	2	2
2.4	Diseñar *back-end*	22	20
2.5	Diseñar *front-end*	22	24
T3	**Desarrollo de sistema**	**180**	**200**
3.1	Crear esquema de la BBDD	5	2
3.2	Desarrollar el *back-end*	85	77
3.3	Desarrollar el *front-end*	85	120
3.4	Conectar *back-end* y *front-end*	5	1
T4	**Verificación/Validación**	**20**	**12**
4.1	Verificar los requisitos	2	1
4.2	Definir los casos de prueba	2	1
4.3	Poblar la base de datos	2	1
4.4	Validar el sistema	2	2
4.5	Desplegar el sistema	12	7
T5	**Documentación**	**30**	**45**
5.1	Revisiones con tutor	5	2
5.2	Documentar el desarrollo	10	20
5.3	Elaborar la memoria	10	20
5.4	Elaborar la presentación	5	3
Total		**300**	**325**

4 Diseño y resolución del trabajo realizado

En este capítulo se describe el proceso de análisis y diseño del prototipo de la aplicación, así como su desarrollo e implementación. También se incluyen las pruebas realizadas y el despliegue.

4.1 Análisis

En este apartado se analiza el alcance del sistema a través de la definición de sus requisitos, representados mediante casos de uso en lenguaje natural. La aplicación distingue entre dos categorías principales de usuarios: **usuarios no registrados** y **usuarios registrados**. Los usuarios no registrados únicamente pueden acceder al portal de la aplicación para explorar los cuestionarios publicados por los usuarios registrados, además de poder realizarlos. En cambio, los usuarios registrados, a parte de estas funciones, también tienen acceso a una experiencia más completa, que incluye la creación y gestión de sus propios cuestionarios (*quizzes*), personalización de su perfil y la capacidad de generar cuestionarios mediante IA a partir de un texto o archivo PDF. Además, no es problema si los usuarios registrados no se acuerdan de su contraseña, ya que podrán solicitar cambiarla a través de un correo electrónico. Por otro lado, estos también podrán eliminar su cuenta, eliminando de esta forma todos los cuestionarios asociados a su perfil.

En los siguientes subapartados se detallan los casos de uso que reflejan las acciones disponibles para cada tipo de usuario en el sistema: usuarios no registrados (ver apartado 4.1.1) y usuarios registrados (ver apartado 4.1.2).

4.1.1 Casos de uso para un usuario no registrado

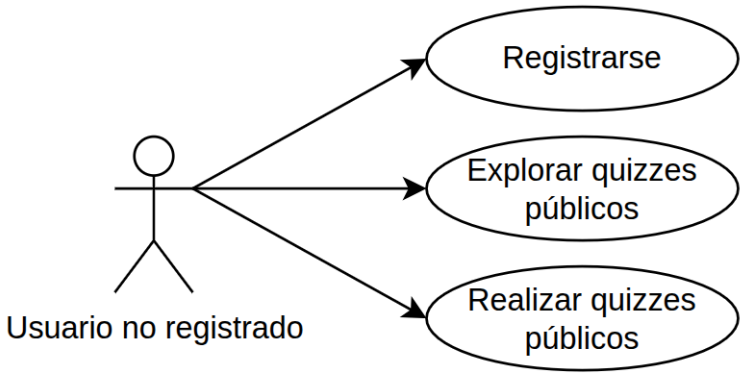

Figura 4. Diagrama de casos de uso para un usuario no registrado

En la Figura 4 se presentan gráficamente los casos de uso asociados a un usuario no registrado, enumerados a continuación:

- **CdU 1 - Registrarse:** El usuario puede registrarse en la aplicación proporcionando el nombre de usuario, un correo electrónico, una contraseña y, opcionalmente, su nombre y apellidos. Adicionalmente puede registrarse usando su cuenta de Google.

- **CdU 2 - Explorar quizzes públicos:** El usuario puede explorar los quizzes públicos de otros usuarios. Este podrá filtrarlos por título y categoría, además de poder ordenarlos por fecha de creación, número de preguntas, título o categoría.
- **CdU 3 - Realizar quizzes públicos:** El usuario podrá realizar los quizzes públicos para poner a prueba sus conocimientos.

4.1.2 Casos de uso para un usuario registrado

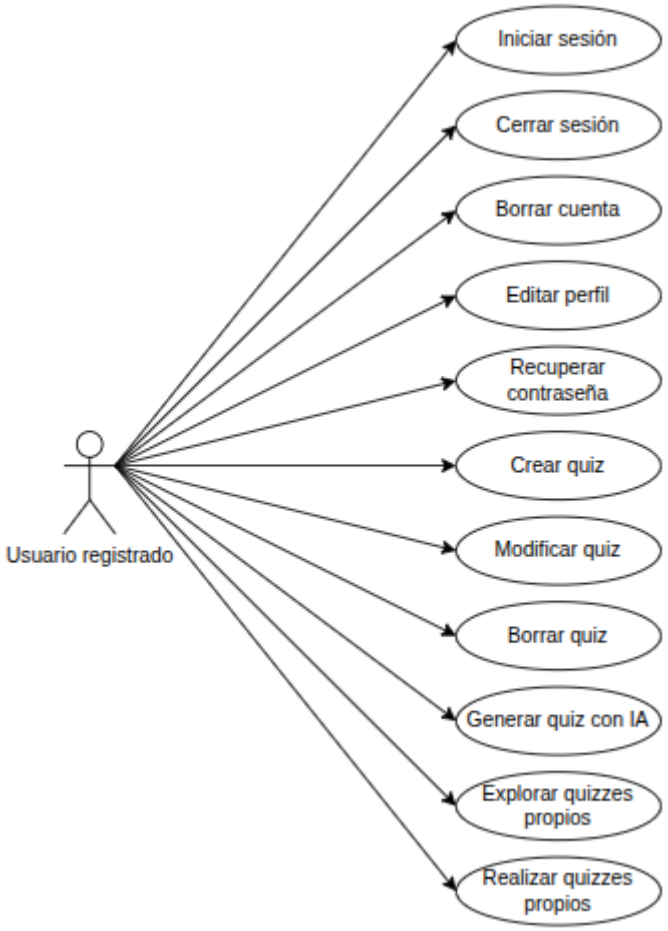

Figura 5. Diagrama de casos de uso para un usuario registrado

En la Figura 5 se presentan gráficamente los casos de uso asociados a un usuario registrado, enumerados a continuación:

- **CdU 4 - Iniciar sesión:** El usuario podrá iniciar sesión mediante su correo o nombre de usuario y la contraseña, o también mediante su cuenta de Google.
- **CdU 5 - Cerrar sesión:** El usuario podrá cerrar sesión.
- **CdU 6 - Borrar cuenta:** El usuario podrá eliminar todos los datos asociados a su cuenta, incluyendo todos sus cuestionarios.
- **CdU 7 - Editar perfil:** El usuario podrá editar la información de su perfil, incluyendo su nombre de usuario, correo, contraseña, nombre de pila, apellidos y foto de perfil.

- **CdU 8 - Recuperar contraseña:** Si el usuario ha olvidado la contraseña de su cuenta, no es un problema, ya que en el panel de login podrá pinchar en "¿Has olvidado la contraseña?" para que se le envíe un correo con un enlace para cambiar la contraseña.
- **CdU 9 - Crear quiz:** El usuario podrá crear un cuestionario, escribiendo manualmente la información del cuestionario, las preguntas y sus posibles opciones de respuesta.
- **CdU 10 - Modificar quiz:** El usuario podrá modificar la información asociada a un cuestionario creado por él, tales como el nombre, la descripción (opcional), la categoría y el tiempo límite, incluso pudiendo modificar las preguntas y respuestas de este. Además, el usuario también podrá cambiar el estado del cuestionario a público o privado.
- **CdU 11 - Eliminar quiz:** El usuario podrá eliminar un cuestionario creado por él.
- **CdU 12 - Generar quiz con IA:** El usuario podrá generar un cuestionario con IA. Simplemente tendrá que indicar los datos esenciales como el nombre, la descripción (opcional), la categoría, la visibilidad (público o privado) y el tiempo límite del cuestionario. Para generar las preguntas, el usuario deberá indicar el número de preguntas y respuestas que desea, además del idioma. Las preguntas se generan a partir de, o bien un texto, o bien un documento PDF indicando qué páginas se desean analizar.
- **CdU 13 - Explorar quizzes propios:** El usuario puede explorar los quizzes que haya creado. Este podrá filtrarlos por título y categoría, además de poder ordenarlos por fecha de creación, número de preguntas, título o categoría.
- **CdU 14 - Realizar quizzes propios:** El usuario podrá realizar los quizzes que haya creado para poner a prueba sus conocimientos

4.2 Diseño

El diseño de la aplicación sigue una arquitectura de múltiples capas (ver Figura 6): presentación, aplicación, persistencia y, siguiendo un enfoque moderno, la pasarela también se puede considerar como otra capa. En la **capa de presentación** se renderiza la interfaz de usuario y se consume la API. En la **capa de aplicación** se gestiona la lógica de negocio procesando los datos y se expone la API. La **capa de persistencia** se encarga del almacenamiento de los datos en la base de datos. Por último, la **capa de la pasarela** es la puerta de entrada a la aplicación, permitiendo el acceso seguro tanto a la capa de presentación (*front-end*), como a la capa de aplicación (API del *back-end*).

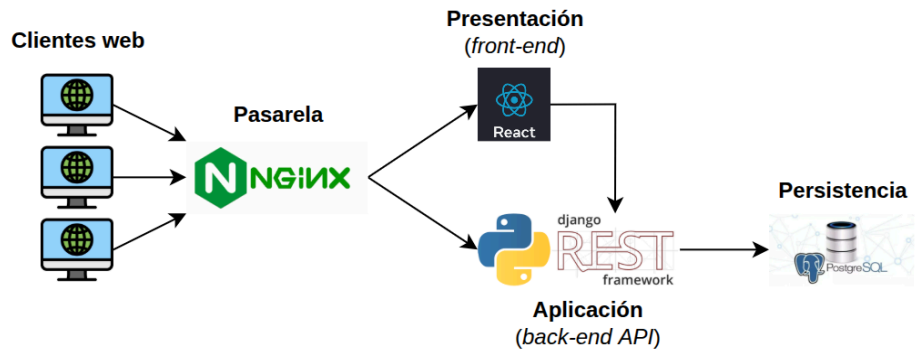

Figura 6. Arquitectura de QuizGenerate

A continuación, se estudiará en profundidad el diseño de la arquitectura, dividiendo el análisis en tres subsecciones: modelo de datos, diseño del *back-end* y diseño del *front-end*, con el objetivo de ofrecer una visión estructurada y detallada de cada componente clave del sistema.

4.2.1 Modelo de datos

En la Figura 7 se puede observar un diagrama simplificado del modelo de dominio de la aplicación, en el que solo se muestran las entidades y sus relaciones, ya que los atributos se mostrarán más adelante en el diagrama de la base de datos.

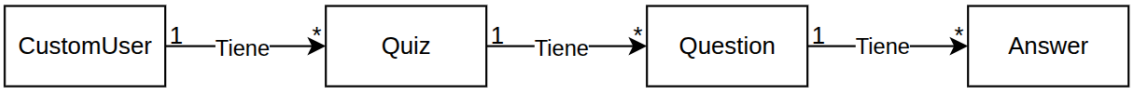

Figura 7. Modelo de dominio de QuizGenerate

La entidad principal del dominio de la aplicación es el usuario, representada en el modelo como ***CustomUser***. Esto se debe a que extiende el modelo de usuario predeterminado de Django (*AbstractUser*), permitiendo añadir nuevos campos como la foto de perfil sin sacrificar las funciones del usuario predeterminado, como la autenticación manejada por Django. Esta jerarquía se puede observar mejor en la Figura 8. En cuanto al resto del dominio, un usuario puede tener varios cuestionarios (*quizzes*) donde, a su vez, cada ***Quiz*** puede tener varias preguntas (*questions*) donde, de nuevo, cada ***Question*** puede tener varias entidades ***Answer*** asociadas.

En la Figura 8 se muestra el esquema completo de la base de datos de QuizGenerate, generado automáticamente empleando la herramienta Graphviz. Este esquema presenta en detalle las entidades, sus atributos y las relaciones establecidas entre ellas. Puede parecer un esquema abrumador lleno de tablas innecesarias, sin embargo, la razón de esto es que la mayoría son tablas generadas automáticamente por Django, debido a que son necesarias para cuestiones internas del framework, como lo son el sistema de usuarios predeterminado, sus permisos, la autenticación basada en tokens y basada en sesiones. Realmente, las únicas tablas relevantes son las siguientes: ***CustomUser***, ***Quiz***, ***Question*** y ***Answer***.

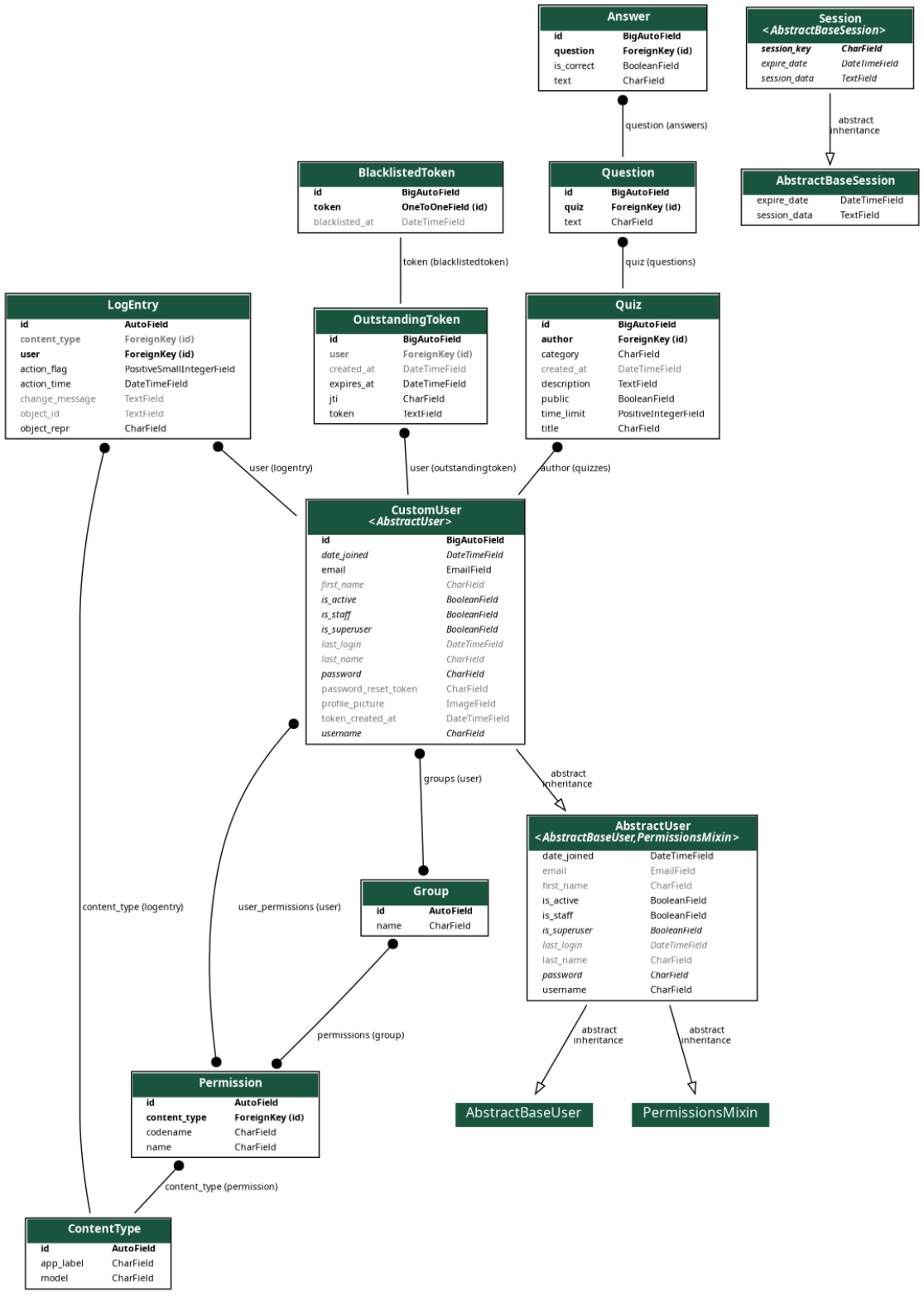

Figura 8. Diagrama de la base de datos

4.2.2 Diseño del back-end

El diseño del *back-end* está completamente desacoplado del *front-end*. El servidor es el único responsable de acceder a los datos de la base de datos, procesarlos y responder a las peticiones a través de su API. Esto posibilita la creación de distintas interfaces de usuario que

acceden al mismo *back-end*, donde cada una es configurada en función de las necesidades de cada proyecto.

El servidor *back-end* responderá las peticiones HTTP que lleguen a su **API REST** utilizando el formato JSON (JavaScript *Object Notation*). Cada recurso se identifica de forma única a través de su URL (Localizador Uniforme de Recursos). Los métodos HTTP permiten interactuar con estos recursos realizando diferentes acciones: **GET** para recuperar datos, **POST** para crear nuevos recursos, **PUT** para actualizar completamente recursos existentes, **PATCH** para realizar modificaciones parciales y **DELETE** para eliminar recursos. En la Tabla 4, se describen los *endpoints* disponibles y las operaciones que se pueden realizar en la API. Además, en color verde se muestran los *endpoints* públicos, y en color rojo aquellos que requieren autenticación para acceder. Por razones de simplicidad, todas las URL de la tabla van precedidas implícitamente por **/api/v1/**.

Tabla 4. Endpoints de la API

Endpoint	GET	POST	PUT	PATCH	DELETE
auth/register/		✔			
auth/login/		✔			
auth/google/		✔			
auth/logout/		✔			
auth/refresh/		✔			
auth/password-reset-request/		✔			
auth/password-reset-confirm/		✔			
users/me/	✔		✔	✔	✔
users/me/change-password/		✔			
quizzes/take/{id}/	✔				
quizzes/public/	✔				
quizzes/public/{id}/	✔				
quizzes/me/	✔	✔			
quizzes/me/{id}/	✔		✔	✔	✔
quizzes/me/{id}/questions/	✔	✔			
quizzes/me/{id}/questions/{id}/	✔		✔	✔	✔
quizzes/me/{id}/questions/{id}/answers/	✔	✔			
quizzes/me/{id}/questions/{id}/answers/{id}/	✔		✔	✔	✔
quizzes/me/generator/		✔			

A modo de ejemplo, se explicarán brevemente algunos *endpoints* importantes. El primero de ellos es `quizzes/me/`, el cual soporta los métodos GET y POST. Este *endpoint* es utilizado

para obtener (GET) la lista de los cuestionarios asociados al usuario autenticado, pero sin contener las preguntas anidadas de cada cuestionario (únicamente los metadatos), con el fin de no sobrecargar la respuesta del servidor. Por otro lado, este *endpoint* también es necesario para que el usuario pueda crear (POST) un nuevo *quiz* propio.

Otro *endpoint* a destacar es `quizzes/me/{id}/`, que acepta los métodos de obtención (GET), actualización (PUT y PATCH) y borrado (DELETE) del cuestionario con ese id. La obtención del cuestionario incluye los metadatos y el array con las preguntas, sin embargo, la actualización solo se permite para los metadatos, ya que sus preguntas y respuestas podrán ser actualizadas mediante sus correspondientes *endpoints* (accediendo con sus respectivos ids).

4.2.3 Diseño del front-end

Para definir el diseño de la interfaz de usuario de QuizGenerate, se decidió no utilizar ninguna herramienta de construcción de *mockups*, debido a que la representación de los datos en la interfaz de usuario iba mentalmente cambiando conforme avanzaba el proyecto, lo que resultaba contraproducente al ir actualizando cada vez el *mockup* en una herramienta.

En la Figura 9 se puede observar el resultado de la versión de escritorio de la página web principal cuando un usuario inicia sesión: “Mis *Quizzes*”. En primer lugar, la pantalla presenta un menú de navegación en su cabecera con el logo de la aplicación a la izquierda y un icono de perfil a la derecha, el cual despliega un menú con botones para personalizar el perfil del usuario, cerrar sesión y eliminar la cuenta. En medio de la cabecera, aparece la barra de navegación con los enlaces correspondientes para acceder a otras páginas: “*Quizzes* Públicos”, “Mis *Quizzes*”, “Crear *Quiz*” y “Generar *Quiz*”. En cuanto a la página “Mis *Quizzes*”, en la parte superior derecha se muestra un selector de resultados para que el usuario pueda indicar el número de *quizzes* que desea visionar. Esto es lo que se conoce como paginación, y se hizo precisamente para evitar la sobrecarga del servidor haciendo peticiones de menor tamaño, en lugar de cargar todos los *quizzes* simultáneamente. Además, aunque la parte inferior de la Figura 9 aparece cortada, si el número de elementos a mostrar (selector) es menor al número de *quizzes* totales que realmente tiene el usuario, aparecerán unos botones para navegar en la paginación y poder acceder a los demás recursos. Por otro lado, en la parte superior aparece un panel de búsqueda, donde se podrán filtrar y ordenar los *quizzes* a partir de los parámetros que introduzca el usuario. Por último, en la parte central se mostrarán los resultados de búsqueda de los *quizzes* con sus respectivas características, y además unos botones con los que se podrá publicar, editar, eliminar o evaluar (realizar) el cuestionario. Aunque no se aprecie la parte inferior de la página, siempre aparecerá un *footer* con diversos enlaces de interés y las credenciales del autor.

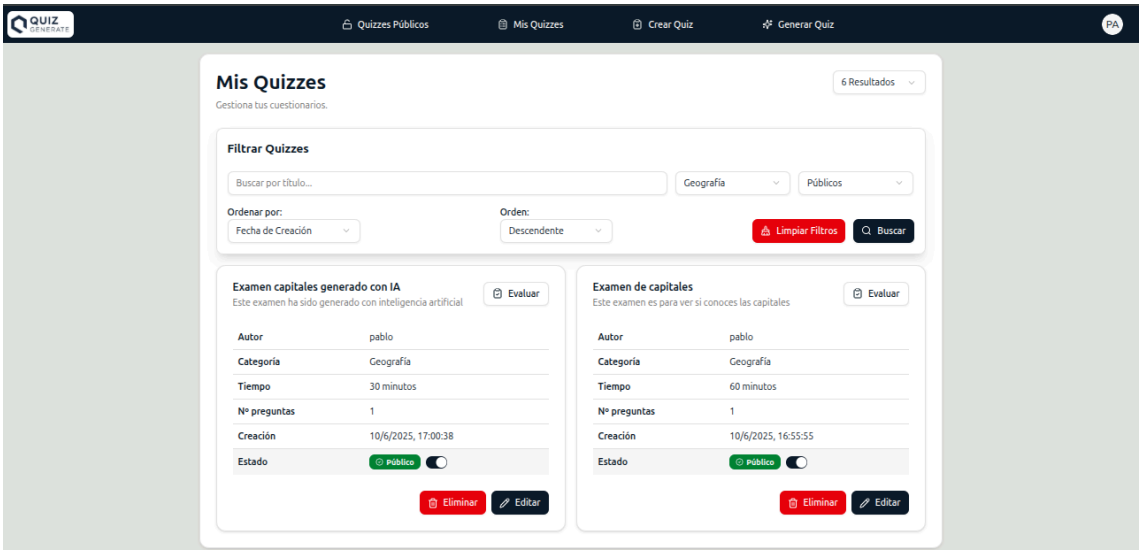

Figura 9. Página "Mis Quizzes" (escritorio)

En la versión móvil, mostrada en la Figura 10, el diseño *responsive* de la aplicación provoca que la barra de navegación de la cabecera se oculte bajo un menú hamburguesa, el cual se despliega al pinchar el icono de tres barras. El diseño del resto de la página también se adapta al ancho del dispositivo, ya que ahora se reorganizará el panel de búsqueda y se mostrará un *quiz* por fila en lugar de dos.

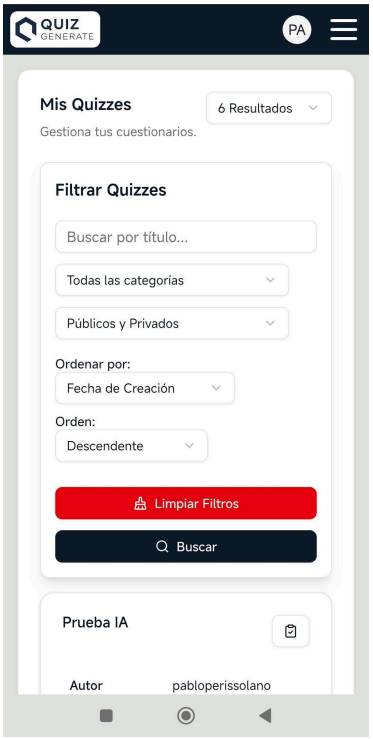

Figura 10. Página "Mis Quizzes" (móvil)

Otra página que se considera relevante es la de realización de un cuestionario, mostrada en la Figura 11. En la parte superior izquierda de esta página se observa el título y la descripción del *quiz*, y en la parte superior derecha, se observa el tiempo que queda para que este finalice. Debajo de la descripción se encuentra la barra de progreso, que indica el número de la pregunta actual del cuestionario, mostrando también el número de preguntas restantes. En el centro de la pantalla aparece la pregunta actual y sus posibles respuestas, de tal forma que cuando se pincha sobre alguna, se muestra un mensaje de acierto o error según sea la opción correcta. Por último, en la parte inferior de la pantalla aparecen dos botones: el botón de anterior para navegar a la pregunta anterior y el botón de siguiente para navegar a la pregunta posterior. Para obtener los resultados y la puntuación del cuestionario, aparecerá en la última pregunta un botón de enviar el cuestionario en lugar del botón "Siguiente". Los resultados también aparecerán cuando finalice el tiempo del *quiz*, ya que se mandarán automáticamente las respuestas que haya dado tiempo a enviar.

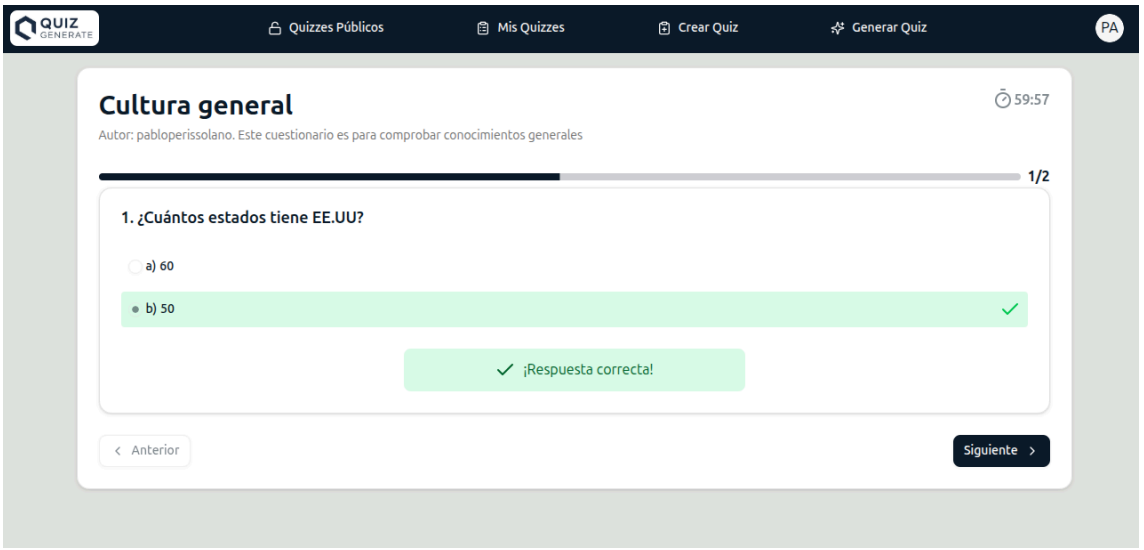

Figura 11. Página "Mis Quizzes"

En la Figura 12 se presentan los diagramas de navegación de QuizGenerate, que ilustran la interconexión entre las distintas pantallas de la aplicación según el estado de autenticación del usuario. La división en dos diagramas busca clarificar las rutas accesibles para cada tipo de usuario (autenticado -abajo en la Figura 12- o no autenticado -arriba en la Figura 12-). Cabe destacar que la cabecera, presente en todas las páginas, adapta su contenido según este estado, permitiendo un acceso rápido a casi cualquier pantalla desde cualquier parte de la aplicación, siempre que el rol del usuario lo permita.

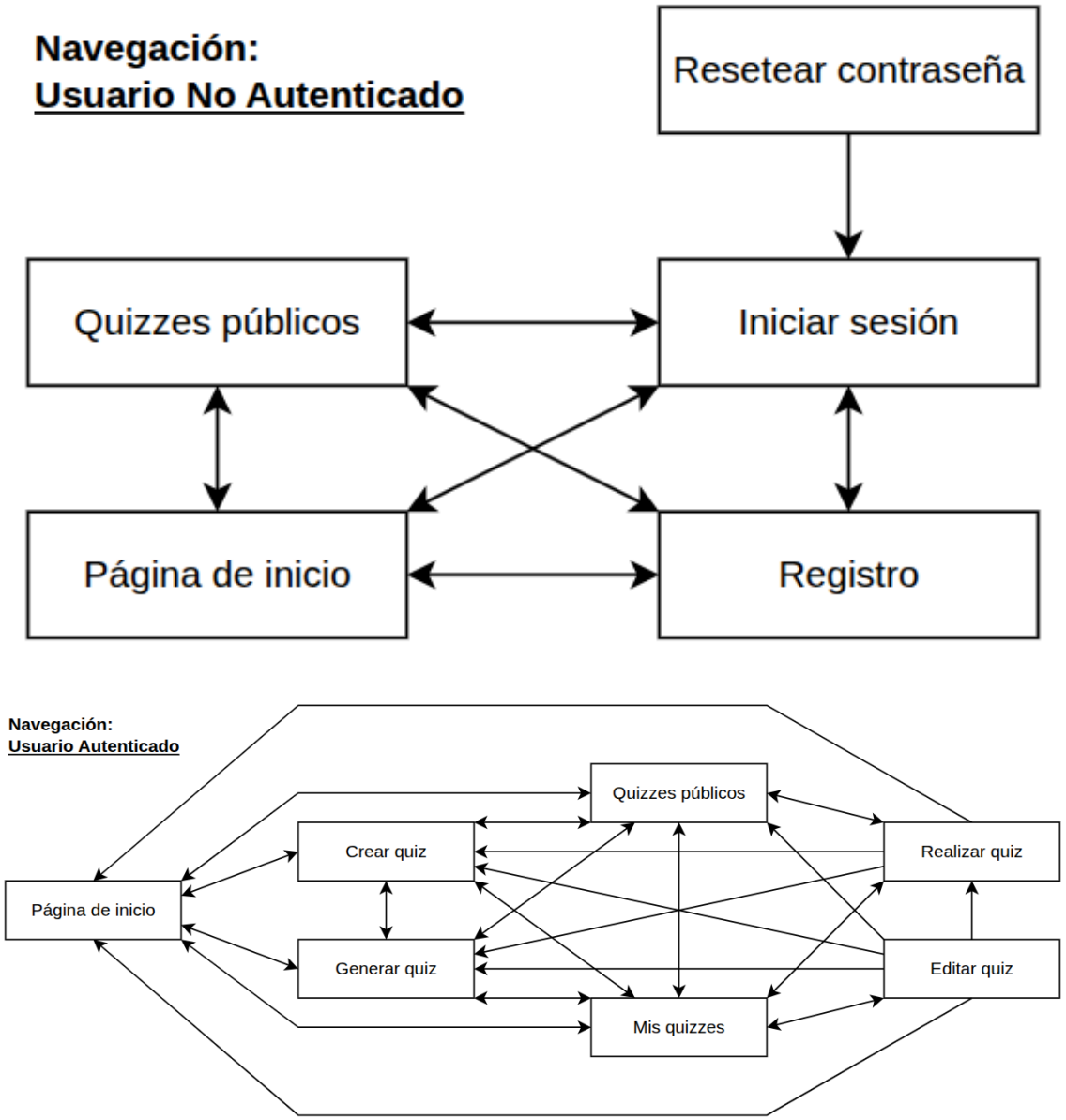

Figura 12. Diagramas de navegación de QuizGenerate

4.3 Implementación

En este apartado se detallan los aspectos más relevantes de la implementación del *back-end* y del *front-end* de la aplicación QuizGenerate. Por motivos de extensión, no es posible detallar la documentación completa del código, por ello la atención se centrará principalmente en la estructura de los proyectos, en cómo se coordinan los diferentes ficheros del código fuente y las funciones destacadas de la aplicación. Se recomienda al lector revisar el código fuente disponible en el repositorio público de GitHub: https://github.com/PabloPerisSolano/TFG

4.3.1 Implementación del back-end

El *back-end* de la aplicación se ha creado mediante un proyecto Django desde cero. Es por ello que, tal como se puede observar en la Figura 13, se sigue una estructura de directorios típica del *framework*. Lo primero que se puede destacar es el directorio `.venv/`, que corresponde con el entorno virtual. Esta carpeta es necesaria en todos los proyectos de Python, ya que es la

encargada de almacenar y gestionar las dependencias del proyecto. Si se sigue avanzando en la estructura de directorios, se encuentra la carpeta `apps/`, la cual almacena los directorios que contienen el código fuente de las aplicaciones del servidor. En este caso, el *back-end* está dividido en dos aplicaciones: por un lado, en la carpeta `users/` se gestionan las peticiones relacionadas con la autenticación e información de usuarios y, por otro lado, en la carpeta `quizzes/` se gestionan las peticiones relacionadas con los cuestionarios de los usuarios. Por su parte, los archivos `__init_.py` simplemente sirven para indicar que el directorio donde están es un paquete Python, los directorios `__pycache__/` sirven para guardar los archivos compilados de los módulos; ambos son irrelevantes. Posteriormente se encuentra la carpeta `media/`, que simplemente almacena los archivos multimedia como las fotos de perfil de los usuarios. Por último, la carpeta `projectTFG/` almacena los ficheros relacionados con la configuración global del proyecto Django. De esta carpeta se puede destacar el fichero `urls.py`, que contiene los *endpoints* de acceso a las aplicaciones de `apps/`; y el fichero `settings.py`, que contiene la configuración general del proyecto Django.

Los archivos de la parte inferior de la Figura 13 también son muy interesantes. Primero se puede destacar el fichero `.env`, el cual no se debe subir a Github, debido a que contiene información sensible como la clave de la base de datos, la del modelo de IA y otras variables de entorno. El siguiente fichero es el `db.sqlite3`, que actúa durante el desarrollo como base de datos predeterminada en proyectos Django. Más adelante se observa el fichero `Dockerfile`, el cual es fundamental para construir la imagen del *back-end* durante el levantamiento de los contenedores en producción. Ya casi al final, aparece el fichero `manage.py`, que también es fundamental para levantar el servidor en fase de desarrollo y realizar migraciones de la base de datos. Por último, los ficheros `requirements-dev.txt` y `requirements.txt`, son útiles para que un usuario que clone el repositorio pueda descargar todas las dependencias utilizadas en fase de desarrollo y producción, respectivamente.

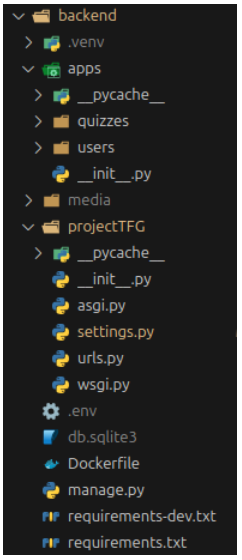

Figura 13. Estructura de directorios del *back-end*

A continuación, se mostrará como ejemplo la entrada de la petición GET `api/v1/quizzes/me/` para explicar el flujo de información a través del servidor. Este *endpoint* permite consultar los *quizzes* propios de un usuario y crear uno nuevo. El primer paso es redirigir la petición al controlador correspondiente dependiendo del comienzo de la URL. En la Figura 14 aparece la configuración del fichero `urls.py` de la carpeta principal `projectTFG`, donde se puede comprobar que las URLs `api/v1/quizzes/` serán redirigidas al fichero `urls.py` de la carpeta `apps/quizzes/`.

Figura 14. Punto de entrada de la API

Continuando con el flujo, en la Figura 15 aparece el fichero `urls.py` de la carpeta `apps/quizzes/`, donde se puede apreciar que la petición `api/v1/quizzes/me/` será recepcionada por la vista `UserQuizListCreateView` del fichero `views.py`.

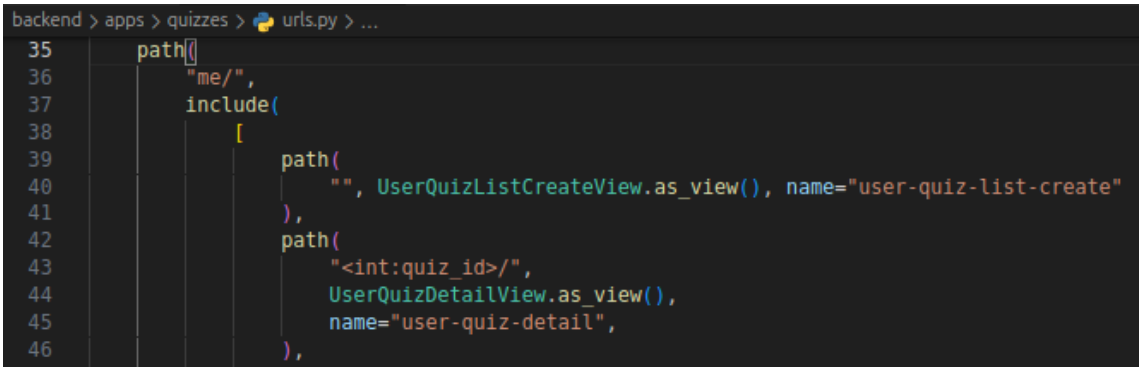

Figura 15. Peticiones dirigidas a quizzes/me/

Tal como se puede apreciar en la Figura 16, la vista `UserQuizListCreateView` hereda de `generics.ListCreateAPIView`, por lo que permite recibir peticiones GET y POST. Las primeras líneas de código de la vista indican que se requiere autenticación del usuario para acceder y que utiliza paginación para devolver la lista de *quizzes*. Además, se utilizará un serializador diferente dependiendo del método HTTP de la petición (`QuizListSerializer` para GET y `QuizCreateSerializer` para POST). Los

serializadores sirven para transformar objetos del modelo (en este caso *Quiz*) a JSON, y están implementados en el fichero `serializers.py`. Otro punto a destacar de la vista es que para devolver los *quizzes* del usuario se debe filtrar por autor, teniendo en cuenta el usuario autenticado en la petición (`author=self.request.user`). De igual forma, el usuario autenticado será el autor cuando se crea un Quiz (método `perform_create`).

backend > apps > quizzes > views.py > UserQuizListCreateView

```python
class UserQuizListCreateView(generics.ListCreateAPIView):
    permission_classes = [IsAuthenticated]
    pagination_class = CustomPageNumberPagination
    serializer_class = QuizListSerializer

    def get_queryset(self):
        queryset = Quiz.objects.filter(author=self.request.user)
        return filter_and_order_quizzes(queryset, self.request.query_params)

    def get_serializer_class(self):
        if self.request.method == "POST":
            return QuizCreateSerializer
        return self.serializer_class

    def perform_create(self, serializer):
        serializer.save(author=self.request.user)
```

Figura 16. Vista UserQuizListCreateView

En la Figura 17 se muestra el serializador encargado de responder a la petición GET `api/v1/quizzes/me/`. Como puede observarse, los datos que se incluyen son los metadatos del *Quiz* (id, autor, nombre, etc.), pero no las preguntas asociadas a cada uno, ya que esto sobrecargaría el JSON de respuesta.

backend > apps > quizzes > serializers.py > QuizListSerializer

```python
class QuizListSerializer(serializers.ModelSerializer):
    author = serializers.StringRelatedField()
    category_display = serializers.CharField(
        source="get_category_display", read_only=True
    )

    class Meta:
        model = Quiz
        fields = [
            "id",
            "author",
            "title",
            "description",
            "public",
            "time_limit",
            "created_at",
            "category_display",
            "num_questions",
        ]
```

Figura 17. Serializador QuizListSerializer

El último paso de la cadena para satisfacer la petición es la interacción del modelo con la base de datos. En la Figura 18 se muestran los campos que conforman la entidad *Quiz* del fichero `models.py`, destacando algunos como la categoría (enumeración) y el autor, donde puede observarse que todos los *quizzes* de un usuario se eliminan cuando este elimina su cuenta (borrado en cascada, CASCADE).

backend > apps > quizzes > models.py > Quiz

```python
class Quiz(models.Model):
    CATEGORY_CHOICES = [
        ("SCIENCE", "Ciencia"),
        ("GEOGRAPHY", "Geografía"),
        ("HISTORY", "Historia"),
        ("SPORTS", "Deportes"),
        ("MUSIC", "Música"),
        ("ART", "Arte"),
        ("TECH", "Tecnología"),
        ("LANGUAGE", "Idiomas"),
        ("LITERATURE", "Literatura"),
        ("MATH", "Matemáticas"),
        ("CINEMA", "Cine/TV"),
        ("HEALTH", "Salud/Bienestar"),
        ("OTHER", "Otros"),
    ]

    author = models.ForeignKey(
        settings.AUTH_USER_MODEL, on_delete=models.CASCADE, related_name="quizzes"
    )
    title = models.CharField(max_length=255)
    description = models.TextField(default="")
    public = models.BooleanField(default=False)
    time_limit = models.PositiveIntegerField(default=3600)  # Tiempo límite en segundos
    created_at = models.DateTimeField(auto_now_add=True)
    category = models.CharField(
        max_length=50,
        choices=CATEGORY_CHOICES,
        default="OTHER",
    )

    @property
    def num_questions(self):
        return self.questions.count()  # Propiedad calculada

    def __str__(self):
        return self.title
```

Figura 18. Entidad Quiz del modelo

Una vez se ha explicado el flujo de información, se detallará la implementación de una de las funcionalidades más relevantes de la aplicación QuizGenerate: la generación de cuestionarios mediante IA. En la Figura 19 se puede observar la petición que se envía a través de la librería OpenAI, una biblioteca que simplifica la petición a la API de IA configurada en el fichero `.env`. En este fichero, también se configura el modelo de IA que se desea utilizar y la clave `AI_API_KEY` que se necesita enviar. Se puede observar que la petición está dividida en dos partes: *system* y *user*. Esto ayuda a que el modelo entienda mejor lo que debe hacer y cómo debe responder. En el bloque *system* se le debe indicar al modelo el contexto, las reglas que debe seguir y el formato de salida deseado. En *user* se define la petición concreta del usuario, teniendo en cuenta los parámetros que se introducen: número de preguntas, número de

opciones por cada pregunta, idioma y *prompt* (texto a analizar). Como se verá más adelante, este *prompt* puede ser el texto plano directamente introducido por el usuario o el texto extraído de las páginas seleccionadas de un documento PDF.

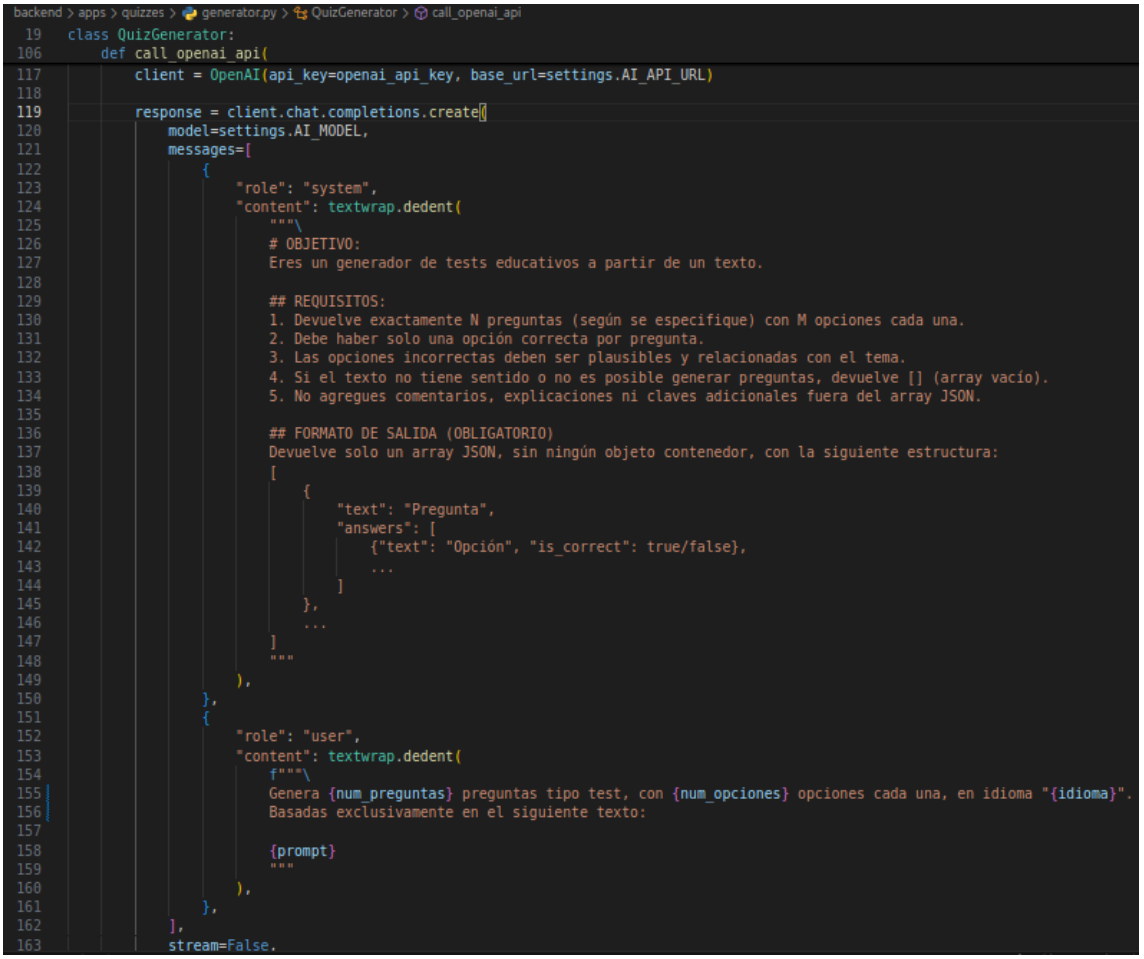

Figura 19. Petición OpenAI para generar preguntas

Por otro lado, una funcionalidad muy importante de la aplicación es el procesamiento de un PDF. Cuando un usuario adjunta el documento, el servidor lo detecta y llama a la función que aparece en la Figura 20. Esta función recibe el fichero PDF y el rango de páginas que se desea analizar. Gracias al módulo **fitz** de la librería PyMuPDF, es posible acceder al contenido del documento. El bucle recorre las páginas seleccionadas por el usuario y va almacenando en una variable el texto extraído con la función `get_text()` para, finalmente, retornarlo.

```python
backend > apps > quizzes > generator.py > QuizGenerator > extract_text_from_pdf
class QuizGenerator:

    @staticmethod
    def extract_text_from_pdf(pdf_file, selected_pages=None):
        if hasattr(pdf_file, "read"):
            doc = fitz.open(stream=pdf_file.read(), filetype="pdf")
        else:
            doc = fitz.open(pdf_file)

        extracted_text = ""

        if selected_pages is None:
            selected_pages = range(1, len(doc) + 1)

        for page_num in selected_pages:
            page_index = page_num - 1
            if 0 <= page_index < len(doc):
                page = doc[page_index]
                page_text = page.get_text()
                extracted_text += f"\n--- Página {page_num} ---\n{page_text.strip()}\n"

        return extracted_text
```

Figura 20. Procesamiento de PDF

4.3.2 Implementación del front-end

El *front-end* de la aplicación se ha creado en un entorno Node.js mediante un proyecto Vite + React desde cero. Es por ello que, tal como se puede observar en la Figura 21, se sigue una estructura de directorios típica del *framework*. Lo primero a destacar es el directorio `node_modules/`, que se encarga de almacenar y gestionar las dependencias del proyecto Node. El segundo directorio es `public/`, y simplemente almacena los archivos multimedia que sirve el *front-end*. Antes de conocer el contenido de la carpeta `src/`, se describirán brevemente los ficheros de la carpeta principal. Al igual que en el *back-end*, el fichero `.env` almacena las variables de entorno, y el `Dockerfile` sirve para construir y servir el *front-end* de React usando **Nginx** en producción. Por otro lado, el fichero `index.html` es la plantilla HTML principal en la que se monta toda la aplicación. Además, en los ficheros `package.json` y `package-lock.json` están definidas las dependencias del proyecto. Por último, los ficheros `components.json`, `eslint.config.js`, `jsconfig.json` y `vite.config.js` son ficheros de configuración general del proyecto.

La carpeta `src/` contiene todo el código fuente de la aplicación React. En esta carpeta se pueden observar los siguientes subdirectorios:

- `assets/`: Contiene archivos estáticos que se importan desde el código (iconos, etc.).
- `components/`: Contiene los componentes reutilizables de la interfaz (botones, cabecera, *footer*, tarjetas, cuadros de diálogo, selectores, interruptores, etc.).
- `constants/`: Contiene variables y objetos constantes de forma centralizada (rutas, *endpoints*, variables, etc.).
- `context/`: Contiene la definición de contextos de React para compartir el estado global (usuario autenticado, etc.).
- `hooks/`: Contiene los *hooks* personalizados de React para lógica reutilizable (`useAuth`, `useAuthFetch`, etc.).

- `lib/`: Contiene las funciones utilitarias o librerías propias que no son componentes ni *hooks*.
- `pages/`: Contiene los componentes de página, donde cada uno representa una ruta principal de la app (*Home*, *Login*, Perfil, etc.).

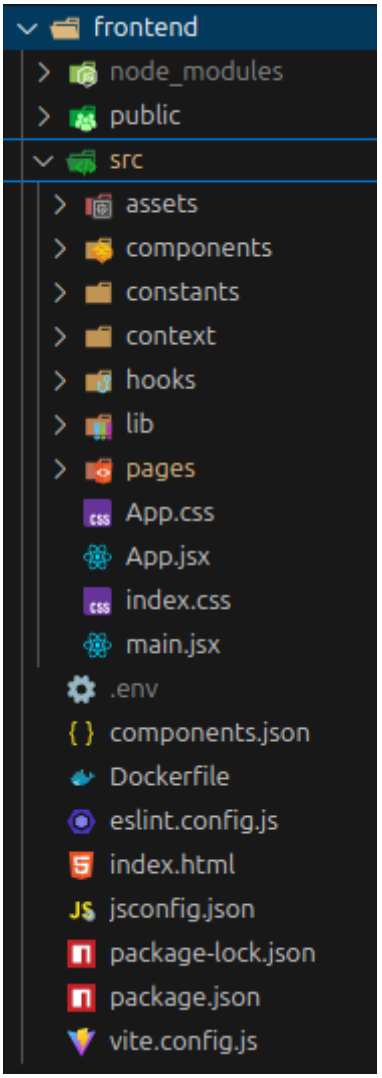

Figura 21. Estructura de directorios del *front-end*

Los otros ficheros que aparecen en `src/` también son fundamentales, ya que tanto `App.css` como `index.css` contienen los estilos CSS globales de la aplicación. Por otro lado, los ficheros `App.jsx` y `main.jsx` también son muy importantes. En la Figura 22 se puede ver el fichero `main.jsx`, que es el punto de entrada de la aplicación, donde se monta el componente `App` (`App.jsx`) en el DOM. También se observan componentes como el `AuthProvider`, que provee de contexto de autenticación a toda la aplicación (usuario actual, *login*, *logout*, etc.); y el `Toaster`, que es necesario para mostrar las notificaciones flotantes de la librería Sonner. Por otro lado, los componentes `StrictMode` y `BrowserRouter` son necesarios para cualquier proyecto de React.

frontend > src > main.jsx

```jsx
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import "./index.css";
import { BrowserRouter } from "react-router-dom";
import App from "./App.jsx";
import { Toaster } from "@/components/ui";
import { AuthProvider } from "@/context";

createRoot(document.getElementById("root")).render(
  <StrictMode>
    <BrowserRouter>
      <AuthProvider>
        <App />
        <Toaster richColors theme="light" closeButton />
      </AuthProvider>
    </BrowserRouter>
  </StrictMode>
);
```

Figura 22. Fichero main.jsx

En la Figura 23 se muestra el fichero `App.jsx`, el cual define las rutas de la aplicación, relacionando cada página (componente) con una URL usando **React Router** [39]. Como se puede observar, en todas las rutas el componente que se renderiza está contenido dentro del `Layout`, con el fin de mostrar la misma estructura en todas las páginas de la web: cabecera en la parte superior, contenido principal en la parte central y *footer* en la parte inferior. Además, se puede observar que algunas rutas están protegidas con el componente `PrivateRoute`, ya que estas únicamente podrán ser accesibles si el usuario está autenticado.

```
frontend > src > App.jsx > App
function App() {
  return (
    <Routes>
      <Route element={<Layout />}>
        <Route path={ROUTES.HOME} element={<Home />} />
        <Route path={ROUTES.LOGIN} element={<Login />} />
        <Route path={ROUTES.REGISTER} element={<Register />} />
        <Route path={ROUTES.RESET_PASSWORD} element={<ResetPassword />} />
        <Route path={ROUTES.PUBLIC_QUIZZES} element={<PublicQuizzes />} />
        <Route path={ROUTES.QUIZ_TAKE} element={<QuizTake />} />

        <Route
          path={ROUTES.MY_QUIZZES}
          element={
            <PrivateRoute>
              <MisQuizzes />
            </PrivateRoute>
          }
        />

        <Route
          path={ROUTES.MY_QUIZ_DETAIL_PATH}
          element={
            <PrivateRoute>
              <QuizDetail />
            </PrivateRoute>
          }
        />

        <Route
          path={ROUTES.CREATE_QUIZ}
          element={
            <PrivateRoute>
              <QuizCreator />
            </PrivateRoute>
          }
        />

        <Route
          path={ROUTES.GENERATE_QUIZ}
          element={
            <PrivateRoute>
              <QuizGenerator />
            </PrivateRoute>
          }
        />
      </Route>
    </Routes>
```

Figura 23. Fichero app.jsx

Una vez conocida la estructura del proyecto, a continuación se mostrará un ejemplo que abarca desde el *login* exitoso de un usuario hasta el acceso a la página principal de “Mis Quizzes”, con el fin de explicar el flujo de información a través del *front-end*, la autenticación, y la comunicación con el *back-end*. Al acceder a la aplicación, el usuario se encuentra con la página de inicio, mostrada en la Figura 24. Esta página resulta especialmente útil para quienes desean aprender a utilizar la plataforma, ya que incluye un carrusel con ejemplos de uso a modo de tutorial.

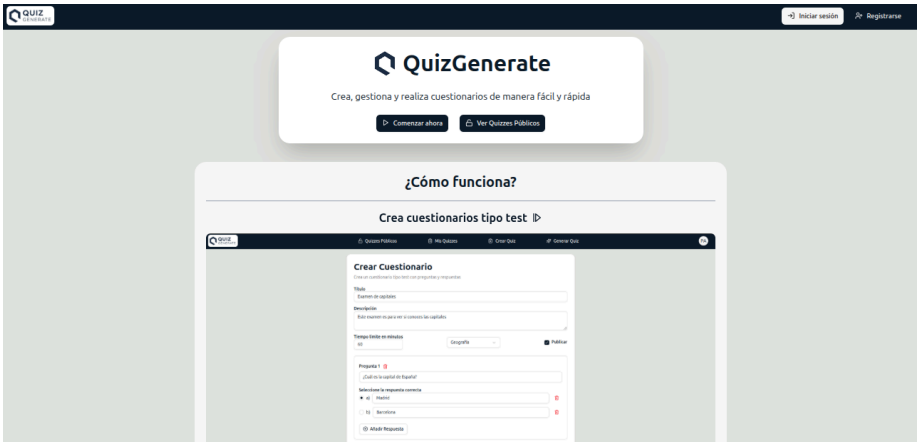

Figura 24. Página de inicio

El primer paso para acceder a todas las funcionalidades es iniciar sesión. Para ello, el usuario debe navegar a la página de *login* pinchando en el botón "Iniciar sesión" de la cabecera, o en "Comenzar ahora" de la página de inicio. En ese momento, el usuario debe completar el formulario con sus credenciales y pulsar el botón para enviar una petición al *back-end*, donde se verificará si el usuario existe. En la Figura 25 se puede apreciar el manejador que ejecuta la petición POST al *endpoint* de *login*, enviando el valor de las variables `username` y `password` en formato JSON. También se puede observar que en caso de error en el *login* aparecerá un mensaje flotante (*toast* de la librería Sonner) con el texto "Credenciales incorrectas". Por otro lado, se utilizan funciones importantes como `fetchWithAuth` para centralizar la lógica de las peticiones y `handleLogin`, que en caso de *login* exitoso registra el usuario actual en el contexto de autenticación. Estas funciones serán explicadas más adelante.

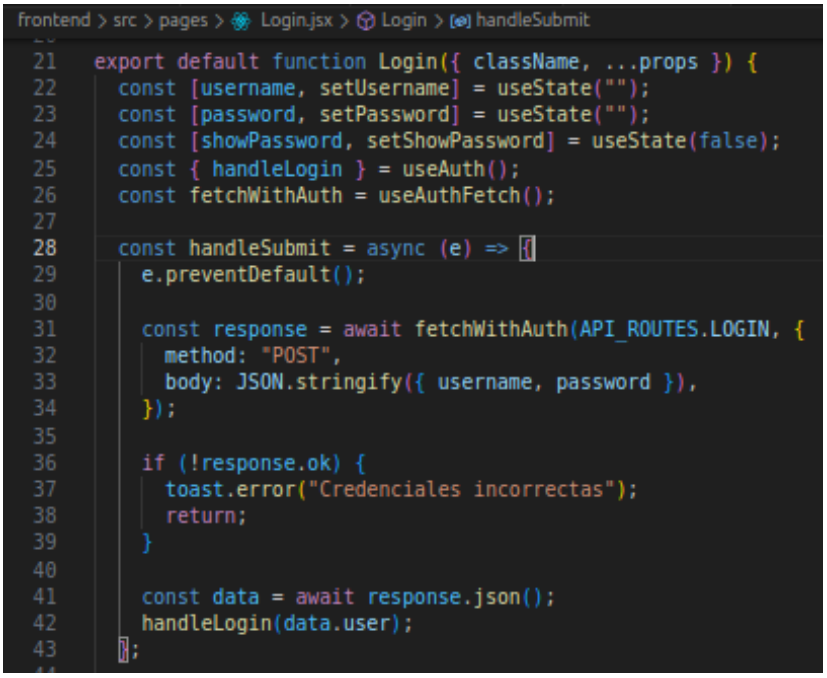

Figura 25. Petición del *login*

La lógica de las peticiones realizadas al *back-end* se maneja de forma centralizada en `useAuthFetch()`, un *hook* personalizado creado con la intención de seguir buenas prácticas de React. La implementación puede apreciarse en la Figura 26. Como se puede observar, se devuelve una petición asíncrona con la cabecera `Content-Type: application/json` por defecto, a menos que se indique una diferente. También es muy importante la opción de `credentials: "include"`, la cual es necesaria para que el navegador pueda enviar y recibir *cookies* del *back-end*, que en este caso corresponden con los *token* JWT de autenticación. De nuevo, en caso de error de autenticación (`401`) se mostrará un mensaje de advertencia y se ejecutará la función `closeSession()`, la cual cerrará la sesión y redirigirá al usuario a la página de *login*. Por último, en caso de éxito de la petición el método termina devolviendo la respuesta del servidor, pero en caso de error inesperado (bloque `catch`) se mostrará un mensaje de error de servidor.

Figura 26. Lógica de peticiones centralizada

La otra función necesaria para completar el *login* es `handleLogin()`. En la Figura 27 se puede apreciar que esta recibe como parámetro la información del usuario logueado exitosamente, estableciéndolo en el contexto de autenticación y navegando directamente a la página principal de “Mis Quizzes”.

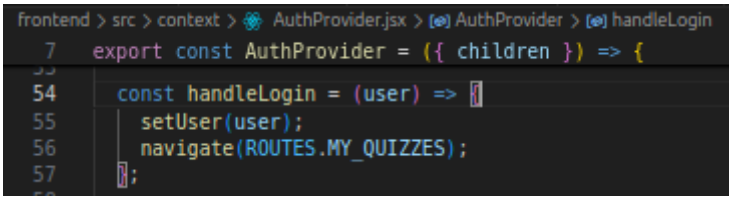

Figura 27. Manejador de login exitoso

Una vez que el usuario se ha autenticado correctamente, es redirigido a la página principal, donde se ejecuta una petición al servidor para poblar la página de datos y mostrar los cuestionarios personales. En la Figura 28 se muestra dicha petición, ejecutada con el método `fetchWithAuth` explicado anteriormente. Como el componente `CardPagedQuizzes` se reutiliza tanto en la página "Mis Quizzes" como en "Quizzes Públicos", la petición puede resultar un tanto abrumadora, sin embargo, la lógica de ambas es prácticamente la misma, diferenciándose en el nombre del *endpoint* solicitado. Además, esta petición puede recibir muchos parámetros en la URL, con el fin de que el usuario pueda establecer el grado de la paginación de los cuestionarios (`page` y `page_size`), el orden en que estos se mostrarán (`sort_by` y `sort_order`) y los filtros para consultar únicamente aquellos de interés (`title`, `category` y `public`). Finalmente, se cargan los *quizzes* a partir de la respuesta de la API y se muestran en la interfaz, o en caso de error en la petición, se muestra el siguiente mensaje de error: "Error al obtener los cuestionarios". En las dependencias del `useEffect()` se encuentran las variables `currentPage`, `itemsPerPage` y `filters`, lo que provocará que el método se vuelva a ejecutar cada vez que el usuario modifique estos valores, refrescando de esta forma la página con los nuevos *quizzes* solicitados.

Figura 28. Lógica de petición de quizzes

Una vez se ha detallado el funcionamiento de la autenticación, la conexión con el *back-end*, y el acceso a la aplicación, se explicará el funcionamiento de otros componentes relevantes. El componente **`DropdownAvatar`** es el menú de configuración del perfil que se activa al pulsar el avatar del usuario. En este menú se gestiona toda la lógica relacionada con la información del usuario autenticado. En la Figura 29 se puede observar que este tiene varios botones, los cuales despliegan diversos cuadros de diálogo con formularios para que el usuario pueda modificar la información de su perfil. En la Figura 30 se muestran los distintos cuadros de diálogo que se abren al pulsar cada botón del menú de configuración. Como se puede observar, para controlar su apertura y cierre se utiliza una variable booleana por cada cuadro de diálogo, la cual se activa al pulsar el botón correspondiente en el menú. Con esta implementación se

pretende que cada componente sea responsable de gestionar sus variables y de enviar el formulario mediante una petición.

Figura 29. Menú de configuración del perfil

```
frontend > src > components > profile > DropdownAvatar.jsx > ...
24  export const DropdownAvatar = ({ children }) => {
87
88        <DialogEditPhoto
89          open={isPhotoDialogOpen}
90          onOpenChange={setIsPhotoDialogOpen}
91        />
92
93        <DialogEditProfile
94          open={isProfileDialogOpen}
95          onOpenChange={setIsProfileDialogOpen}
96        />
97
98        <DialogChangePassword
99          open={isPasswordDialogOpen}
100         onOpenChange={setIsPasswordDialogOpen}
101       />
102
103       <DialogDeleteAccount
104         open={isDeleteDialogOpen}
105         onOpenChange={setIsDeleteDialogOpen}
106       />
```

Figura 30. Cuadros de diálogo del perfil

Una de las vistas más relevantes de la aplicación es "**Generar Quiz**" (ver Figura 31), la cual puede dividirse en dos secciones principales. La primera corresponde al componente `BaseQuizCreate`, que contiene el formulario para definir los metadatos del cuestionario: título, descripción, tiempo límite, categoría y estado. Este componente, al ser reutilizable, también es empleado en la vista "Crear Quiz". La segunda parte se centra en la generación automática de preguntas mediante IA. En esta sección, el usuario puede especificar el número de preguntas deseadas, el número de opciones por pregunta y el idioma del cuestionario. Finalmente, en la parte inferior de la vista, se presenta un formulario para adjuntar el documento PDF del cual se extraerán las preguntas. Es posible indicar un rango de páginas a analizar escribiendo una lista o conjunto de rangos. Por ejemplo: 1-3, 5, 7-9 procesará las páginas 1, 2, 3, 5, 7, 8 y 9. Si el campo se deja vacío, se analizarán todas las páginas del documento. Si el usuario prefiere generar las preguntas a partir de un texto plano, deberá cambiar a la pestaña "Texto", donde podrá introducirlo en el *input*. Una vez que se han introducido todos los datos se pulsará el botón de "Generar Cuestionario", que ejecutará la petición al *back-end* enviando todos los parámetros. Dependiendo de la cantidad de texto y preguntas deseadas, este proceso de generación puede tardar varios segundos, debido a que el servidor tiene que procesar el texto y la petición a la API externa de IA.

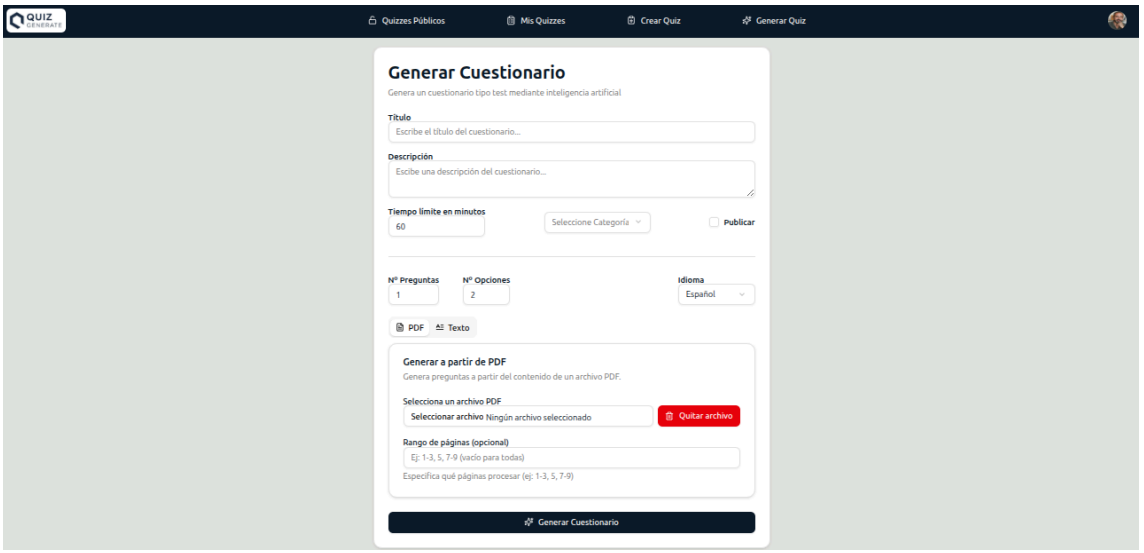

Figura 31. Página de generar cuestionario

Uno de los aspectos más relevantes de la implementación de dicha página es la petición que se envía al *back-end*. En la Figura 32 se observa que existen dos formas de realizar este envío. La primera consiste en enviar el documento PDF mediante un objeto `FormData`, ya que es la forma adecuada de transmitir formularios que contienen archivos y datos en una petición HTTP. La segunda opción es enviar el *prompt* como texto plano, para lo cual basta con convertir el formulario a formato JSON.

```
frontend > src > pages > QuizGenerator.jsx > QuizGenerator > handleCreateQuiz
 26  export default function QuizGenerator() {
 50    const handleCreateQuiz = async (e) => {
129      let body;
130
131      if (isPdf) {
132        body = new FormData();
133        body.append("title", title);
134        body.append("description", description);
135        body.append("time_limit", tiempo * 60);
136        body.append("public", publicar);
137        body.append("category", category);
138        body.append("num_preguntas", num_preguntas);
139        body.append("num_opciones", num_opciones);
140        body.append("idioma", idioma);
141        body.append("pdf", pdfFile);
142        if (pageRange.trim()) {
143          body.append("page_range", pageRange.trim());
144        }
145      } else if (isText) {
146        body = JSON.stringify({
147          title,
148          description,
149          time_limit: tiempo * 60,
150          public: publicar,
151          category,
152          num_preguntas,
153          num_opciones,
154          idioma,
155          prompt,
156        });
157      }
158
159      setLoading(true);
160
161      const res = await fetchWithAuth(API_ROUTES.QUIZ_GENERATOR, {
162        method: "POST",
163        body,
164      });
```

Figura 32. Petición para generar un cuestionario

Una implementación enfocada a mejorar la experiencia de usuario es el contenedor animado de la librería Motion. Este se utiliza en la página de realización de un *quiz* para mostrar transiciones entre cada pregunta. Como se puede observar en la Figura 33, cada vez que se cambia de pregunta (`currentQuestion.id`) se ejecuta una animación que dura 0,3 segundos, en la cual el contenedor que envuelve la pregunta empieza invisible y desplazado 40 píxeles a la derecha, y termina visible y en el centro. El resultado es la sustitución de la pregunta anterior por la pregunta actual.

```
frontend > src > pages > QuizTake.jsx > TakeQuizPage
 34   rt default function TakeQuizPage() {
242         <AnimatePresence mode="wait">
243           <motion.div
244             key={currentQuestion.id}
245             initial={{ opacity: 0, x: 40 }}
246             animate={{ opacity: 1, x: 0 }}
247             exit={{ opacity: 0, x: -40 }}
248             transition={{ duration: 0.3 }}
249           >
```

Figura 33. Contenedor de preguntas animadas

Por último, en la Figura 34 se puede apreciar la reutilización del componente `CardPagedQuizzes`, que muestra las vistas "Mis *Quizzes*" y "*Quizzes* Públicos". El factor diferencial radica en el valor del parámetro `isPublicVariant`, ya que el contenido será diferente en función de este.

```
frontend > src > pages > MisQuizzes.jsx > ...
1  import { CardPagedQuizzes } from "@/components/cards";
2
3  export default function MisQuizzes() {
4    return <CardPagedQuizzes isPublicVariant={false} />;
5  }
```

```
frontend > src > pages > PublicQuizzes.jsx > ...
1  import { CardPagedQuizzes } from "@/components/cards";
2
3  export default function PublicQuizzes() {
4    return <CardPagedQuizzes isPublicVariant={true} />;
5  }
```

Figura 34. Componente MisQuizzes y PublicQuizzes

4.3.3 Implementación de la pasarela

La Figura 35 hace referencia al fichero de configuración de la pasarela con Nginx, donde se pueden observar tres bloques *server*. El primer bloque *server* es utilizado para redirigir las peticiones del dominio `www.quizgenerate.xyz` a `quizgenerate.xyz`. En el segundo bloque *server* se puede apreciar que el servidor Nginx acepta peticiones HTTPS (puerto 443) para el dominio `quizgenerate.xyz`. Como se trata de HTTPS, se utilizan certificados SSL de **Let's Encrypt** [40] para cifrar las conexiones. Además, esta configuración de la pasarela permite acceder tanto al *front-end* como a la API y sus archivos multimedia, dependiendo de si la URL entrante empieza con `/`, `/api/v1/` o `/api/v1/media/` respectivamente. En caso de que la petición se dirija al *front-end*, se servirán los archivos estáticos de las páginas almacenados en `/usr/share/nginx/html`. En caso de que la petición se dirija al *back-end*, se añade la cabecera `X-Forwarded-Proto` para que Django sepa el *host*, la IP real del cliente y el protocolo original HTTPS. En caso de que la petición se dirija a los archivos multimedia, se buscará el fichero solicitado en la carpeta `/app/media/` para devolverlo y guardarlo en la caché del navegador durante 30 días. Por último, el tercer bloque *server* está configurado para redirigir todo el tráfico HTTP (puerto 80) a HTTPS.

nginx > default.conf

```
# Redirige www.quizgenerate.xyz a quizgenerate.xyz (HTTPS)
server {
    listen 443 ssl;
    server_name www.quizgenerate.xyz;
    client_max_body_size 20M;

    ssl_certificate /etc/letsencrypt/live/quizgenerate.xyz/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/quizgenerate.xyz/privkey.pem;

    return 301 https://quizgenerate.xyz$request_uri;
}

# Servidor principal quizgenerate.xyz (HTTPS)
server {
    listen 443 ssl;
    server_name quizgenerate.xyz;
    client_max_body_size 20M;

    ssl_certificate /etc/letsencrypt/live/quizgenerate.xyz/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/quizgenerate.xyz/privkey.pem;

    # Ruta para el frontend
    location / {
        root /usr/share/nginx/html;
        try_files $uri /index.html;
    }

    # Ruta para el backend
    location /api/v1/ {
        proxy_pass http://backend:8000/api/v1/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme; # Importante para Django
    }

    # Ruta para archivos multimedia
    location /api/v1/media/ {
        alias /app/media/;
        expires 30d;
        add_header Cache-Control "public";
    }
}

# Redirección HTTP → HTTPS para ambos dominios
server {
    listen 80;
    server_name quizgenerate.xyz www.quizgenerate.xyz;
    return 301 https://quizgenerate.xyz$request_uri;
}
```

Figura 35. Fichero de configuración de Nginx

4.4 Pruebas

Durante el desarrollo de la aplicación, se han llevado a cabo pruebas de cada nueva funcionalidad implementada, verificando que todas las partes del sistema respondan correctamente ante distintos escenarios de uso. Para validar las respuestas del *back-end*, se ha utilizado la aplicación Postman, que permite enviar peticiones HTTP a la API en un entorno controlado, con el fin de analizar la correcta respuesta de los diferentes *endpoints*. En la Figura 36 se muestra la colección de Postman creada para las pruebas, organizada en diferentes carpetas de contenido relacionado. Esta colección de Postman está disponible en la carpeta anexos del repositorio de Github: https://github.com/PabloPerisSolano/TFG/tree/main/anexos.

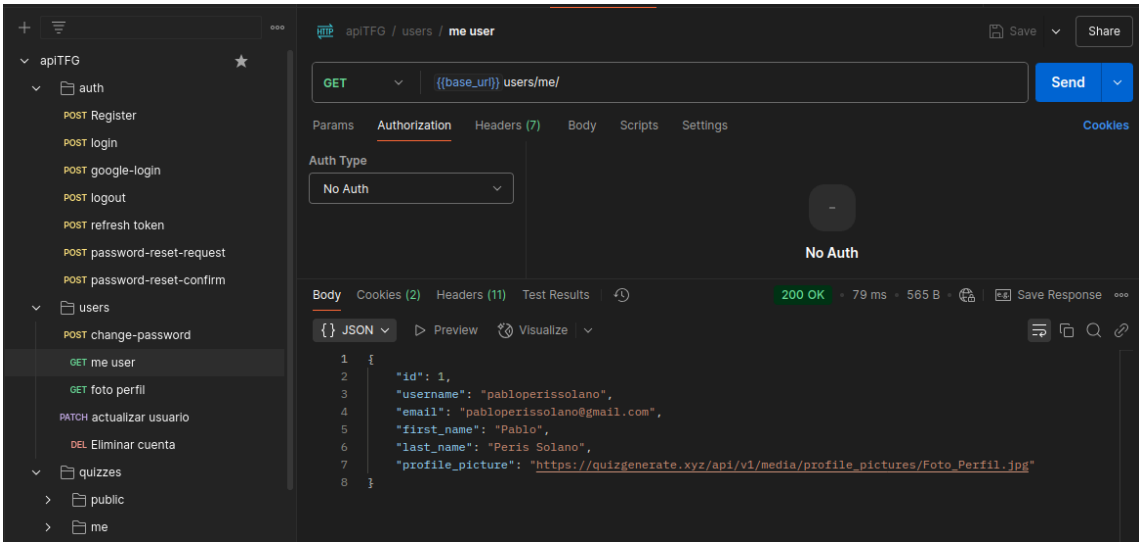

Figura 36. Pruebas con Postman

En cuanto al *front-end*, se realizaron comprobaciones manuales utilizando las herramientas de desarrollo del navegador. Esto permitió simular distintos tamaños de pantalla y dispositivos, asegurando que la interfaz de la aplicación mantuviera su diseño *responsive* y que la experiencia de usuario fuera consistente. Este enfoque ha permitido asegurar la eficacia del sistema sin necesidad de implementar pruebas unitarias.

4.5 Despliegue

La herramienta utilizada para desplegar la aplicación ha sido **Docker**, debido a su gran potencial y facilidad de uso. Esta herramienta permite ejecutar cada componente en un contenedor aislado que tiene todas las dependencias necesarias. En este caso, las imágenes de Docker que se han utilizado en producción han sido las siguientes: `postgres:14.11` para la base de datos PostgreSQL, `python:3.12-slim` para la API (*back-end)*, `node:22-alpine` para la construcción del *front-end*, y `nginx:alpine` como pasarela. En relación con esta última tecnología, la herramienta Nginx actúa en producción de dos formas: como servidor de archivos estáticos del *front-end* y como pasarela (*proxy* inverso) para poder redirigir las peticiones a la API *(back-end)* o al *front-end*, dependiendo del comienzo de la *URL*.

Para el despliegue y un servicio continuo de la aplicación, se decidió alquilar un servidor en **Hetzner Cloud** [41]. El servidor elegido fue el CAX11, el cual tiene las siguientes características: 2 CPUs, 4 GB de RAM, 40 GB de almacenamiento SSD y un precio de máximo 3,79 € al mes, dependiendo de su uso.

Además, para que todo el mundo pueda acceder fácilmente a la aplicación, se decidió alquilar un dominio `.xyz` en **Namecheap** [42] por 1€, con el fin de que la URL sea fácil de recordar. Una vez comprado el dominio `quizgenerate.xyz`, fue necesario asociar ese nombre de dominio con el servidor alquilado de Hetzner, de forma que ya estaría disponible para el resto de internet bajo el siguiente enlace: `https://quizgenerate.xyz/`. Por último, cabe destacar que el manual de usuario de la aplicación está disponible en la carpeta anexos del repositorio de Github del proyecto.

5 Conclusiones y vías futuras

5.1 Conclusiones

Sin lugar a dudas, la realización de este proyecto ha supuesto uno de los retos más ambiciosos como ingeniero de software hasta la fecha. Ha sido una formidable carrera de fondo donde se ha aprendido muchísimo a lo largo del desarrollo, aumentando los conocimientos en diversos ámbitos de la programación, tales como: buenas prácticas en el diseño de *APIs*, buenas prácticas en el diseño de interfaces de usuario, descubrimiento de nuevas tecnologías, fortalecimiento de los contenidos aprendidos durante el grado y, realmente lo más importante, el hecho de involucrarse en el desarrollo de un proyecto real, desde la fase de pensamiento inicial de la idea hasta la fase final del despliegue y puesta en producción de la aplicación.

QuizGenerate se presenta como una solución que permite a los usuarios crear, generar y realizar cuestionarios online. Además, la plataforma proporciona una interfaz intuitiva y accesible, lo que facilita su uso para aquellos usuarios que recurren a técnicas de estudio interactivas y automatizadas. Sin embargo, aunque el sistema actual es completamente funcional y útil para la gestión de cuestionarios, existen áreas de mejora que podrían potenciar aún más la plataforma. Características adicionales, como la incorporación de diferentes roles o de distintos tipos de cuestionario, podrían convertir esta herramienta en una solución aún más robusta y competitiva en el sector de la educación.

Por último, cabe destacar el uso de herramientas basadas en IA generativa, tales como Chat GPT, Github Copilot o Deep Seek, que a lo largo del desarrollo del proyecto han contribuido a agilizar el trabajo y mejorar la calidad, especialmente en el desarrollo y revisión de fragmentos de código, la comprensión de errores y la búsqueda de soluciones más eficientes. No obstante, su uso ha requerido un juicio crítico constante, ya que no siempre las respuestas han sido precisas o adecuadas al contexto, por lo que ha sido necesario contrastarlas con fuentes fiables y aplicar criterios propios. También es necesario recordar la gran labor que ha realizado el tutor, ya que durante toda la realización del documento ha sido un eslabón clave en cuanto a mejorar la redacción y el contenido de cada apartado.

5.2 Vías futuras

Como se mencionó previamente, la plataforma desarrollada ofrece una base sólida para la creación y gestión de cuestionarios. Sin embargo, durante el desarrollo del proyecto se plantearon diversas funcionalidades que, debido a limitaciones de tiempo y recursos, no pudieron ser implementadas.

Una de las mejoras más relevantes sería la incorporación de control de acceso basado en roles, ya que en el ámbito educativo es común que los profesores y alumnos tengan diferentes ambiciones en cuanto al uso de la aplicación. Por ejemplo, el rol de profesor tendría la posibilidad de crear una clase añadiendo a sus alumnos, para poder compartir los exámenes fácilmente. Por otro lado, el rol de alumno tendría la posibilidad de aceptar las invitaciones para unirse a las clases de los profesores. Esto requeriría forzosamente incorporar también un sistema de notificaciones. Por otro lado, sería también interesante la implementación del rol de administrador, que tendría la capacidad de revisar contenido inapropiado y eliminar usuarios con comportamientos indeseados.

Además, otra mejora que sería muy interesante de implementar es la posibilidad de crear cuestionarios de distinto tipo, ya que actualmente la aplicación solo permite cuestionarios tipo test de opción múltiple donde solo una respuesta es correcta. Sin embargo, esto podría ampliarse a cuestionarios de verdadero y falso, cuestionarios de varias respuestas correctas y cuestionarios para rellenar huecos.

Otro aspecto clave para enriquecer la plataforma sería la internacionalización de la aplicación. La posibilidad de cambiar el idioma de la interfaz facilitaría el acceso a un público más amplio, permitiendo que usuarios de distintas partes del mundo puedan disfrutar de la herramienta. Además, en caso de que la aplicación escale comercialmente se podría financiar mediante publicidad.

Una mejora que también se podría implementar es la capacidad de generar cuestionarios a partir de imágenes y vídeos mediante IA, ya que actualmente la aplicación solo es capaz de procesar documentos PDF y texto plano. Además, sería enriquecedor poder procesar también archivos con estructuras complejas como hojas de cálculo o esquemas resumen.

Por último, sería interesante implementar una interfaz donde el usuario pudiera consultar la lista de los *quizzes* y puntuaciones que ha realizado históricamente.

6 Bibliografía

[1] M. Gamarra, "Herramientas digitales educativas: importancia e influencia en el sistema actual". https://www.inesem.es/revistadigital/educacion-sociedad/herramientas-digitales-educativas/ (último acceso 24/06/2025)

[2] C.A. Hernández Suárez, J. D. Hernández Albarracín, J. Rodríguez Moreno, "Obstáculos y barreras de los docentes en la integración de TIC y sus repercusiones en el contexto postpandemia", Mundo Fesc, vol 14, no. 29, pp. 8-23 de 2024. https://doi.org/10.61799/2216-0388.1541 (último acceso 22/06/2025)

[3] Kahoot! | Learning games | Make learning awesome! https://kahoot.com/ (último acceso 22/06/2025)

[4] Revisely | AI-Powered Learning Resources. https://www.revisely.com/library (último acceso 22/06/2025)

[5] Algor Education: la plataforma más completa para aprender con la IA. https://www.algoreducation.com/es (último acceso 22/06/2025)

[6] Python. https://www.python.org/ (último acceso 22/06/2025)

[7] Los 9 mejores lenguajes de programación para la Inteligencia … https://www.wingsoft.com/blog/mejores-lenguajes-IA (último acceso 22/06/2025)

[8] Django: The web framework for perfectionists with deadlines. https://www.djangoproject.com/ (último acceso 22/06/2025)

[9] Django REST framework: Home. https://www.django-rest-framework.org/ (último acceso 22/06/2025)

[10] Simple JWT — Simple JWT 5.5.0.post18+g890e136 documentation. https://django-rest-framework-simplejwt.readthedocs.io/en/latest/ (último acceso 22/06/2025)

[11] Pip. https://pypi.org/project/pip/ (último acceso 22/06/2025)

[12] SQLite Home Page. https://sqlite.org/ (último acceso 22/06/2025)

[13] PostgreSQL: The world's most advanced open source database. https://www.postgresql.org/ (último acceso 22/06/2025)

[14] PyMuPDF 1.26.1 documentation. https://pymupdf.readthedocs.io/en/latest/ (último acceso 22/06/2025)

[15] OpenAI Platform. https://platform.openai.com/docs/overview (último acceso 22/06/2025)

[16] DeepSeek API Docs: Your First API Call. https://api-docs.deepseek.com/ (último acceso 22/06/2025)

[17] JavaScript - MDN Web Docs - Mozilla. https://developer.mozilla.org/es/docs/Web/JavaScript (último acceso 22/06/2025)

[18] React. https://es.react.dev/ (último acceso 22/06/2025)

[19] DOM virtual y detalles de implementación. https://es.legacy.reactjs.org/docs/faq-internals.html (último acceso 22/06/2025)

[20] Vite | Next Generation Frontend Tooling. https://vite.dev/ (último acceso 22/06/2025)

[21] Node.js — Run JavaScript Everywhere. https://nodejs.org/es (último acceso 22/06/2025)

[22] HTML: Lenguaje de etiquetas de hipertexto - MDN Web Docs. https://developer.mozilla.org/es/docs/Web/HTML (último acceso 22/06/2025)

[23] CSS - MDN Web Docs. https://developer.mozilla.org/es/docs/Web/CSS (último acceso 22/06/2025)

[24] Tailwind CSS - Rapidly build modern websites without ever leaving … https://tailwindcss.com/ (último acceso 22/06/2025)

[25] Build your Component Library - shadcn/ui. https://ui.shadcn.com/ (último acceso 22/06/2025)

[26] Lucide React. https://lucide.dev/icons/ (último acceso 22/06/2025)

[27] Sonner. https://sonner.emilkowal.ski/ (último acceso 22/06/2025)

[28] Motion. https://motion.dev/docs/react-quick-start (último acceso 22/06/2025)

[29] Ubuntu 22.04.5 LTS (Jammy Jellyfish). https://releases.ubuntu.com/jammy/ (último acceso 22/06/2025)

[30] Visual Studio Code - Code Editing. Redefined. https://code.visualstudio.com/ (último acceso 22/06/2025)

[31] GitHub Copilot · Your AI pair programmer. https://github.com/features/copilot (último acceso 22/06/2025)

[32] GitHub · Build and ship software on a single, collaborative platform … https://github.com/ (último acceso 25/06/2025)

[33] Chat GPT. https://chatgpt.com/ (último acceso 27/06/2025)

[34] Postman: The World's Leading API Platform | Sign Up for Free. https://www.postman.com/ (último acceso 22/06/2025)

[35] Draw. https://app.diagrams.net/ (último acceso 24/06/2025)

[36] Graphviz. https://graphviz.org/ (último acceso 22/06/2025)

[37] Nginx. https://nginx.org/ (último acceso 23/06/2025)

[38] Docker: Accelerated Container Application Development. https://www.docker.com/ (último acceso 23/06/2025)

[39] React Router Official Documentation. https://reactrouter.com/ (último acceso 23/06/2025)

[40] Certificados SSL/TLS Gratuitos. https://letsencrypt.org/es/ (último acceso 23/06/2025)

[41] Cheap hosted VPS by Hetzner: our cloud hosting services. https://www.hetzner.com/cloud (último acceso 24/06/2025)

[42] Namecheap — Domains & Hosting | Make More Online, for Less. https://www.namecheap.com/ (último acceso 24/06/2025)