UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Máster Universitario en Ingeniería Web

Trabajo Fin de Máster

Red social destinada al estudio mediante repetición espaciada

Autores

Jorge Da Silva Del Palacio

Igor López Salazar

Tutor

Santiago Alonso Villaverde

Julio de 2024

Agradecimientos

“Quiero agradecer a mis padres y a mi hermano por apoyarme en mi decisión de hacer este máster, así como a mis compañeros de piso por todo lo que hemos vivido durante este año.”

“También querría agradecer a nuestro tutor, Santiago Alonso Villaverde, porque gracias a su asistencia hemos conseguido un mejor proyecto.”

“Y, por último, gracias a mi compañero Igor, sin el cual lógicamente este proyecto no habría podido salir adelante, y con el que ha sido un placer compartir proyecto.”

Jorge Da Silva Del Palacio

“Agradecer a familia y amigos por apoyarme y ayudarme en mis decisiones. Así como a mis compañeros de piso por las experiencias vividas durante este año.”

“Dar gracias a Santiago, por haber sido nuestro tutor durante este periodo y habernos ayudado a mejorar el proyecto.”

“Además, agradecer también a Jorge, mi compañero, por su apoyo y esfuerzo para sacar adelante este proyecto, aun con los quebraderos de cabeza encontrados a lo largo del camino, gracias.”

Igor López Salazar

Resumen

La memorización de conceptos es una necesidad común en muchos ámbitos, desde los estudios hasta el ámbito laboral. De entre todos los métodos que es posible utilizar, uno de los que aporta mejores resultados es la repetición espaciada. Aunque se puede aplicar de diversas formas, la más extendida es mediante el sistema Leitner, donde el conocimiento a adquirir se divide en cartas de un mazo. Cada una de esas cartas debe ser estudiada periódicamente, y conforme más se asienten los conocimientos, más tiempo transcurrirá antes de volver a repasarlo; la representación usual de esta técnica es mediante cajas, donde las cajas más tempranas implican sesiones más recurrentes, y las cajas más lejanas sesiones más distantes.

Aunque actualmente existen herramientas que permiten el estudio mediante este método, no existe ninguna que facilite el intercambio de información a la vez que realiza una gestión correcta de los estudios del sistema Leitner. Es por ello por lo que se ha decidido crear una aplicación web, orientada como una red social, que permita realizar el proceso de estudio mediante la repetición espaciada, pero además permitiendo compartir los mazos con otros usuarios.

La aplicación, llamada IntelliDeck, ayudará en el estudio, permitiendo la creación de mazos acorde al sistema Leitner, y luego gestionando las cartas a mostrar cada día y los resultados obtenidos de los estudios. Si el usuario no tiene conocimiento suficiente como para crear su propio mazo, puede recurrir a aquellos creados por otras personas, y luego valorarlos en base a su calidad.

Este Trabajo Fin de Máster recoge una posible solución para ayudar a la gente en la memorización de información, e integra diferentes conceptos aprendidos durante el Máster en Ingeniería Web a través de algunas de las asignaturas. Para llevar a cabo este proyecto, se utilizará Angular como tecnología para el front-end, NodeJS para el back-end y MongoDB para la base de datos, gestionando además el control de versiones mediante Git. En concreto, se utilizará GitHub para un acceso más sencillo al código y una mayor facilidad de gestión, ya que también facilita la integración y el despliegue continuos mediante GitHub Actions. Por último, el proyecto entero se llevará a cabo utilizando la metodología Rational Unified Process (RUP), realizando los diagramas necesarios para facilitar el entendimiento del proyecto.

Palabras clave

Repetición espaciada, sistema Leitner, red social, aplicación web, Angular, NodeJS, MongoDB

Abstract

Concept memorization is a common necessity in various areas, from studies to the workspace. Among the different existing methods, the one which provides the best results is spaced repetition. Although it can be applied in diverse ways, the most widespread one is using the Leitner system, where the knowledge to be acquired is divided in cards from a deck. Each of those cards must be studied periodically, and the more the information is consolidated, the more time will elapse before reviewing it again; the most common representation of this technique is by means of boxes, where the earlier boxes imply more recurrent sessions, and the later boxes more distant sessions.

Event though different tools exist that allow studying by using this method, none of them facilitates the exchange of information while performing a correct management of the Leitner system studies. Because of that, it has been decided to create a web application, oriented as a social network, that allows the study process to be conducted using spaced repetition, also making possible to share decks with other users.

The application, named IntelliDeck, is aimed to help studying, allowing the creation of decks according to the Leitner system, and then managing the cards to be shown each day and the results obtained from the sessions. If the knowledge of the user is not sufficient, it is possible to resort to those created by other people, and then rate them based on their quality.

This master's thesis explains a possible solution to help people in the memorization of information, and integrates different concepts learned throughout some subjects of the Master in Web Engineering. To carry out this project, Angular will be used as the front-end technology, NodeJS for the back-end and MongoDB for the database, while also managing version control through Git. Specifically, GitHub will be used for easier access to code and easier management, as it also facilitates continuous integration and deployment via GitHub Actions. Finally, the entire project will be carried out using the Rational Unified Process (RUP) methodology, making the necessary diagrams to facilitate the understanding of the project.

Keywords

Spaced repetition, Leitner system, social network, web application, Angular, NodeJS, MongoDB

Índice general

Índice de figuras

Índice de tablas

Símbolos y abreviaturas

API Application Programming Interface.

AWS Amazon Web Services.

BD Base de Datos.

CI/CD Continuous Integration / Continuous Deployment.

CRUD Create, Read, Update, Delete.

CSS Cascading Style Sheets.

DAO Data Access Object.

DoS Denial of Service.

HTML HyperText Markup Language.

HTTP HyperText Transfer Protocol.

ID IDentificador.

JS JavaScript.

JSON JavaScript Object Notation.

JWT Json Web Token.

MB MegaByte.

MVC Modelo-Vista-Controlador.

MVVM Model-View-ViewModel.

NoSQL No Structured Query Language.

ODM Object-Document Mapper.

RUP Rational Unified Process.

SCSS Sassy Cascading Style Sheets.

SDLC Software Development Life Cycle.

SPA Single-Page Application.

SQL Structured Query Language.

TFM Trabajo Fin de Máster.

TLS Trasport Layer Security.

TS TypeScript.

URL Uniform Resource Locator.

UX User eXperience.

Capítulo 1. Introducción

A lo largo de esta memoria se recoge todo el proceso llevado a cabo para la realización de un Trabajo Fin de Máster (TFM) a presentar en el Máster en Ingeniería Web de la Universidad Politécnica de Madrid, integrando gran parte de los conocimientos adquiridos en el mismo.

El TFM propuesto está centrado en facilitar la memorización de conceptos mediante la repetición espaciada, uno de los métodos más eficaces en este ámbito. Además, se orientará como una red social, para simplificar el intercambio de información entre usuarios.

El primer punto de esta memoria será realizar una pequeña introducción del proyecto, de nombre IntelliDeck. Para ello, se analizará la situación actual, explicando en profundidad los conceptos sobre los que se fundamenta este TFM y analizando el mercado en busca de aplicaciones similares. Posteriormente, se listarán los objetivos generales y específicos de la aplicación.

Tras la introducción, se procederá a explicar el marco teórico del TFM, donde se proporcionará el conocimiento necesario para comprender la base metodológica y tecnológica del proyecto.

Seguidamente, en base a lo propuesto por la metodología de desarrollo seleccionada, se procederá a la obtención de requisitos del proyecto, para efectuar a continuación un análisis del sistema a implementar. El siguiente paso será detallar cómo se ha realizado la implementación del sistema respetando la metodología elegida, para luego extraer una serie de conclusiones acerca de todo el proceso llevado a cabo.

Para finalizar, se propondrán posibles líneas futuras para el proyecto, y se listarán todas las referencias bibliográficas utilizadas durante la realización de esta memoria.

1.1 Situación Actual

La necesidad en el estudio de interiorizar conceptos mediante la memorización es un requerimiento para gran parte de la población, desde estudiantes hasta trabajadores. Para ello, existen diferentes formas de realizar una memorización efectiva. Uno de los más predominantes es el conocido como repetición espaciada.

Esta metodología surge de la mano de Hermann Ebbinghaus en el año 1885, mientras realizaba experimentos en relación a la memoria. Esta investigación estaba orientada a la psicología y fue originalmente publicada en alemán, siendo en 1913 traducida al inglés [1]. Pese a que el origen de la investigación se halla en la psicología, los conceptos explicados han sido desde entonces aplicados en diferentes ámbitos.

Uno de los conceptos que Ebbinghaus definió tras sus investigaciones fue la curva del olvido, aunque no fue él quien acuñó este termino. Durante sus experimentos, Ebbinghaus realizaba una serie de memorizaciones, y después comprobaba en que medida se había deteriorado ese conocimiento. En base a esto, propuso un gráfico que representaba la perdida de la información en la memoria conforme pasa el tiempo: esto es lo que se conoce como curva del olvido.

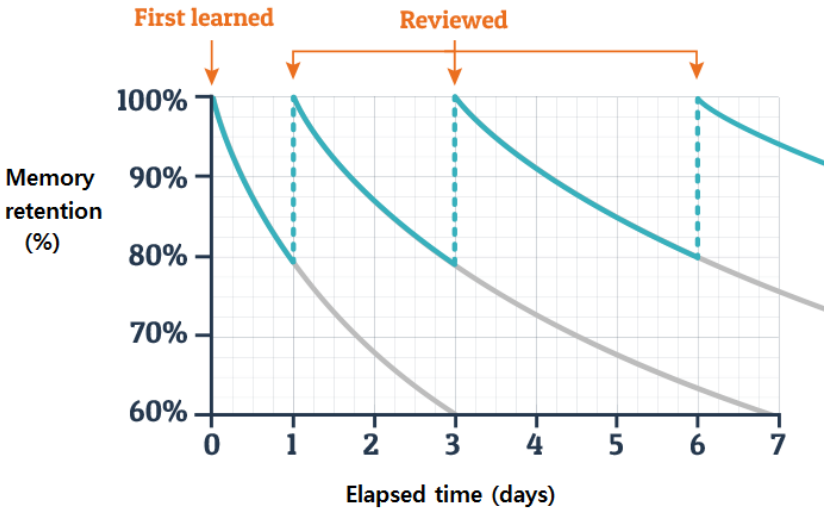


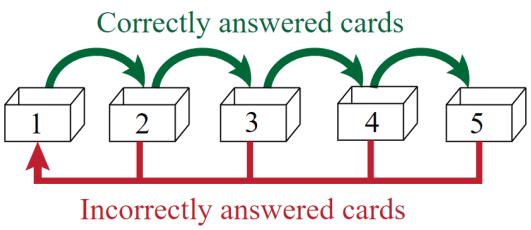
Figura 1.1: *Representación de la curva del olvido [2]*

Como se puede apreciar en la imagen, tras haber estudiado un concepto, este se deteriora en la mente y se va olvidando paulatinamente. Sin embargo, revisándolo periódicamente, es posible aumentar el tiempo que este es retenido en la memoria. Esta revisión periódica es lo que se conoce como repetición espaciada.

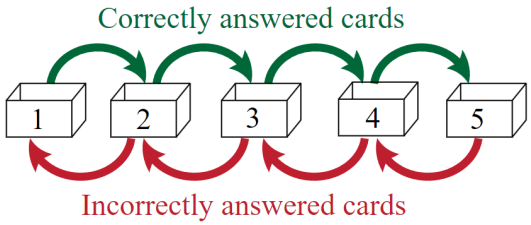
La repetición espaciada es un método de revisión de información en intervalos sistemáticos. Al inicio del proceso de aprendizaje, estos intervalos son cercanos entre sí, para posteriormente estar más distanciados. Por ejemplo, Ebbinghaus utilizó intervalos de 19 minutos, 65 minutos, 8 horas, 1 día, 2 días, 6 días y 30 días [3]. La duración de estos intervalos se debe a que durante los primeros 20 minutos después del aprendizaje, los conceptos se olvidan muy rápidamente. La pendiente en la curva, sin embargo, se vuelve mucho menos abrupta conforme se avanza en el espaciado, debido a que los conceptos comienzan a interiorizarse.

La capacidad de afincar los conocimientos mediante la repetición espaciada ha sido analizada en numerosos estudios, y se han propuesto diferentes métodos para aplicarla. De entre todos ellos, este proyecto se ha centrado en uno de los más conocidos, el cual fue propuesto por Sebastian Leitner y al que da nombre: el sistema Leitner.

El planteamiento de este sistema es el siguiente: se establecen unas cajas, cada una de las cajas representando un intervalo (en su libro, Leitner propone la seguida de Fibonacci en días como los intervalos a usar). Por otro lado, se crean cartas con una pregunta por un lado y su respuesta en el otro, y se ponen todas en la caja del intervalo menor. Si, al realizar un estudio, la carta es respondida correctamente, esta avanzará a la siguiente caja. En caso de que una carta se responda erróneamente, esta volverá a la primera caja [4]. Existe también otra variante en la que las cartas no retroceden hasta la primera caja, sino que solamente a la caja anterior.



(a) *Representación del sistema Leitner con retroceso a la primera caja [5]*



(b) *Representación del sistema Leitner con retroceso a la caja anterior [6]*

Figura 1.2: *Representaciones de los métodos de aplicación del sistema Leitner*

En caso de querer profundizar en el entendimiento de la repetición espaciada y el sistema Leitner, se recomienda la lectura de [1, 4, 7, 8].

Al plantear este proyecto, además, se le dio gran importancia a la posibilidad de compartir información entre los usuarios, para así poder aprender de gente que pueda saber más sobre un tema concreto; es decir, se decidió aplicar el estudio colaborativo.

El estudio colaborativo es una metodología educacional de enseñanza y aprendizaje que aboga por que un grupo de estudiantes traten conjuntamente de resolver un problema [9], y que implica tanto el trabajo en equipo tradicional como el proceso de compartir información en línea mediante chats y foros.

Actualmente, la forma más popular de estudiar es mediante Internet, ya que, conforme pasa el tiempo, es posible encontrar información más veraz y completa, además de que cada vez forma más y más parte del ámbito educacional, permitiendo el acceso a enciclopedias, diccionarios, traductores y demás fuentes fiables de forma cómoda y centralizada.

Una buena forma de asegurar la aplicación del estudio colaborativo a la vez que se atiende al gran uso de Internet para el estudio es mediante la utilización de una red social. En la actualidad, las redes sociales se usan de media 143 minutos al día por persona, siendo los jóvenes entre 16 y 24 los que más tiempo las utilizan. Debido a esto, gran parte de la población está ya familiarizada con este tipo de herramientas, por lo que no es necesaria una etapa de adaptación, haciendo más fácil su uso.

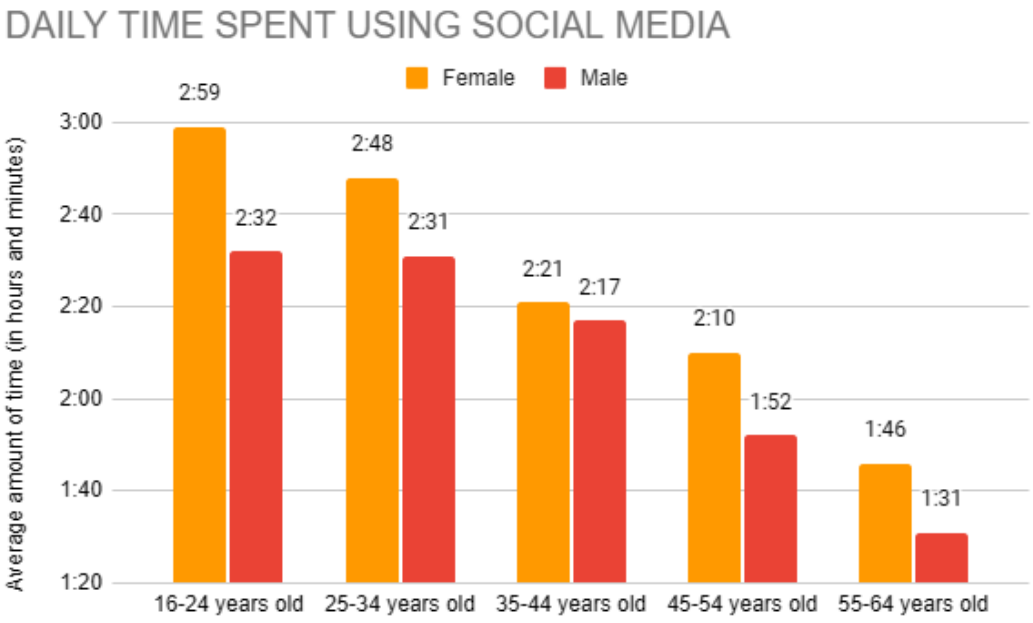


Figura 1.3: *Media de horas diarias por género y grupos de edad hasta enero de 2024 [10]*

A día de hoy, existen otras herramientas que proporcionan un servicio similar, las cuales son listadas a continuación:

- **AnkiApp:** Esta herramienta permite la creación de cartas para su estudio mediante repetición espaciada, permitiendo además compartir con el resto de usuarios las cartas creadas. Sin embargo, aplica una variante del sistema Leitner, ya que el usuario, en caso de acertar, puede determinar la dificultad que le ha supuesto dicha carta, para así situarla en una caja u otra. Además, las sesiones de estudio son cortas, de no más de un minuto, por lo que no todas las cartas que deben ser estudiadas se revisan.

- **Cram:** Mediante esta herramienta es posible realizar estudios de repetición espaciada siguiendo el sistema Leitner, o bien creando mazos, o bien usando algunos ya existentes. Además, proporciona mini-juegos (como una versión del *Asteroids* de Atari, por ejemplo) para poder memorizar los conceptos. Aún así, la interfaz es obtusa, por lo que puede causar rechazo o una adaptación sustancial hasta que un usuario pueda utilizar la herramienta correctamente.

- **StudyStack:** Esta aplicación posibilita la creación de mazos con cartas, así como compartir estos mazos con otros usuarios. Además, proporciona la capacidad de usar gamificación para facilitar el asentamiento de los conceptos, con crucigramas o el ahorcado, por ejemplo. Pese a todo, no realiza la gestión de los futuros estudios, por lo que queda a cuenta del usuario realizar la repetición espaciada.

Partiendo de las diferentes aplicaciones que se han encontrado, se ha decidido buscar nuestra propia solución para satisfacer a las necesidades ya mencionadas, para así conseguir una herramienta que integre gran parte de las bondades de las aplicaciones recientemente analizadas.

1.2 Objetivos

Habiendo establecido la problemática a la que hacer frente, el siguiente punto a tratar fue la definición de objetivos. En concreto, se han recogido objetivos de dos tipos: generales (en relación a la funcionalidad que proporcionará la aplicación) y específicos (aquellos destinados a demostrar la correcta obtención de nociones y conocimientos a lo largo del máster cursado). Ambos tipos de objetivos son recogidos en la siguientes subsecciones.

1.2.1 Objetivo general

El objetivo general de este proyecto es proporcionar una herramienta que ayude a las personas a realizar una memorización de conceptos efectiva mediante la repetición espaciada. Además, deberá facilitar el intercambio de conocimientos con otros usuarios de la aplicación, asegurando por el camino la veracidad y calidad de la información.

1.2.2 Objetivos específicos

Además del objetivo general, también se han establecido ciertos objetivos orientados a la adquisición de nuevos conocimientos, así como la cimentación de aquellos obtenidos durante los estudios:

- Desarrollar un proyecto utilizando la metodología de desarrollo Rational Unified Process (RUP).
- Asegurar un control de versiones adecuado mediante GitHub, así como la integración de mecanismos de CI/CD para un despliegue optimizado del sistema.
- El uso de Angular para la realización de las interfaces de usuario y las interacciones con estas.
- La utilización de NodeJS para la realización de un sistema *RESTful* para la gestión de la información del sistema.
- Aprender a realizar tests unitarios y de integración en entornos NodeJS, utilizando para ello la librería Jest.
- Asegurar la total fiabilidad del sistema en lo que respecta a la seguridad de los datos y de la aplicación en sí, evitando todas aquellas vulnerabilidades a las que pueda ser susceptible la aplicación.

Capítulo 2. Marco Teórico

En esta sección, se realizará una explicación de todos los conceptos implicados en el desarrollo del proyecto, los cuales se clasificarán en diferentes categorías: los conceptos relacionados con la metodología de desarrollo del proyecto y los vinculados a las tecnologías usadas durante la implementación.

2.1 Metodología de desarrollo

A la hora de desarrollar un proyecto software, es importante atender al Software Development Life Cycle (SDLC), el ciclo de vida del desarrollo software, que cubre todas las disciplinas del software desde su concepción en base a los requerimientos de un cliente hasta su puesta en marcha y mantenimiento [11]. Las disciplinas de dicho ciclo de vida son listadas a continuación:

- Obtención de requisitos
- Análisis
- Diseño
- Implementación
- Pruebas
- Despliegue

Debido a la complejidad que supone tratar el software en diferentes disciplinas, históricamente se han desarrollado distintas metodologías para aplicar el SDLC.

La primera propuesta fue realizada por Winston W. Royce en el año 1970, y es conocida como la metodología en cascada. Esta metodología establece que cada una de las etapas del ciclo de vida no puede empezar hasta que la disciplina previa ha concluido, por lo cual las etapas se realizan de forma secuencial [12]. Debido a su naturaleza, requiere que todos los requisitos del sistema a desarrollar sean conocidos desde el principio.

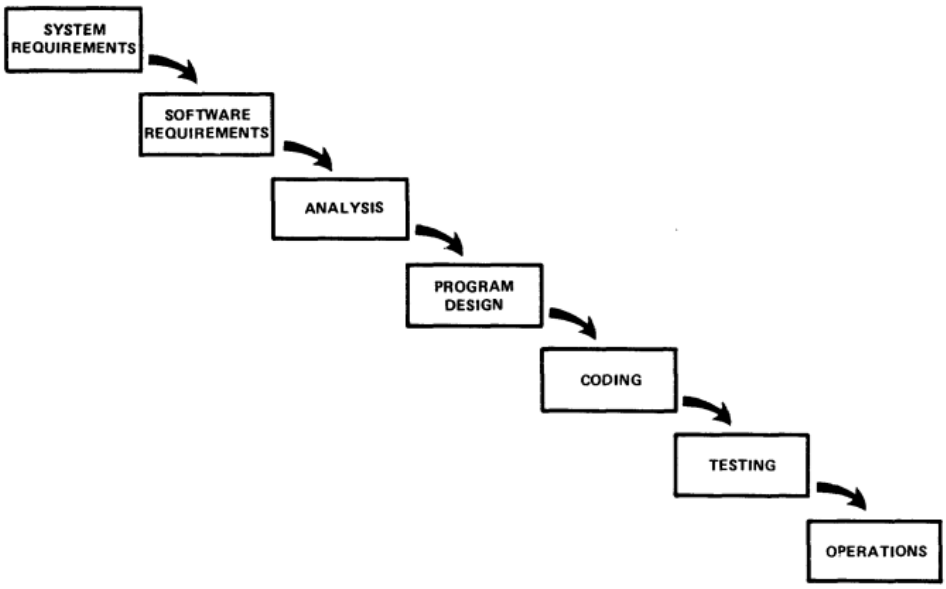


Figura 2.1: *Planteamiento de la metodología en cascada realizado por Winston W. Royce en 1970 [12]*

Sin embargo, tal y como el propio Royce determinaba en su artículo, la metodología mostrada en la imagen previa es arriesgada e invita al fallo. De hecho, sucesos como el cambio de requerimientos por parte del cliente son acciones que suponen un contratiempo significativo al utilizar esta metodología [13], intensificándose los problemas conforme mayor sea el proyecto, debido a que no es sencillo retroceder entre etapas.

Es a causa de esto por lo que, paulatinamente, fueron surgiendo distintas metodologías que abrazaban un nuevo concepto: iteraciones. Es en torno a este concepto que se definen las metodologías incrementales. En un modelo incremental, los requisitos no tienen por qué ser conocidos desde el principio, y cada iteración que se realiza del sistema produce un incremento en las funcionalidades del producto, hasta que este esté finalizado [14]. Cada una de estas iteraciones, por su parte, se lleva a cabo realizando todas las etapas del SDLC definidas por el modelo en cascada [15]. De entre todas las metodologías incrementales existentes, en este proyecto se ha utilizado Rational Unified Process (RUP).

RUP es una metodología concebida por Grady Booch, Ivar Jacobson y James Rumbaugh en 1999, la cual se puede definir en base a tres conceptos [16]:

- Está dirigida por casos de uso.
- Está centrada en la arquitectura.
- Es iterativa e incremental.

El concepto “caso de uso” hace referencia a una descripción de una secuencia de eventos que, en conjunto, hacen que un sistema realice algo útil para un actor del sistema [17], refiriéndose el término actor a un usuario que vaya a utilizar la aplicación.

Al ser RUP una metodología de desarrollo incremental, las diferentes etapas del SDLC se realizan a lo largo de las iteraciones. Sin embargo, a diferencia de otras metodologías incrementales, donde en cada incremento se trabajan todas las etapas, en RUP es posible realizar un

trabajo menor o mayor de cada una de estas disciplinas, como se puede apreciar el la imagen inferior.

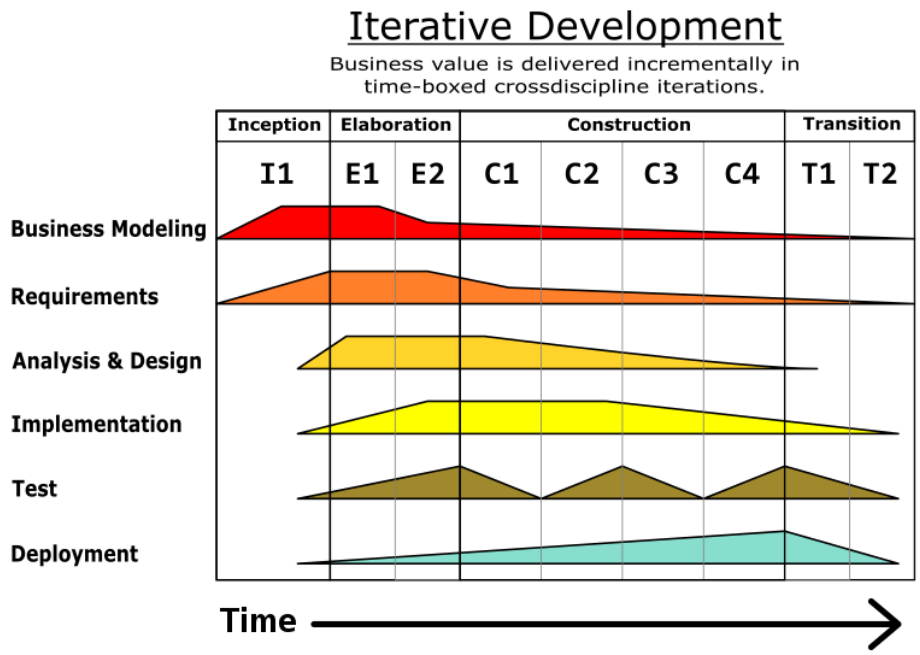


Figura 2.2: *Disciplinas de RUP a lo largo del proyecto [18]*

Como es posible apreciar, tareas como la obtención de requisitos se realizan de forma más exhaustiva al inicio del proyecto, y la cantidad de tiempo dedicada a esta disciplina conforme se avanza en el tiempo se va reduciendo. De igual forma, algunas etapas van adquiriendo carga de trabajo conforme se avanza en el proyecto. La distribución del esfuerzo de cada disciplina en la anterior imagen está condicionada por las fases del proyecto.

Un proyecto de RUP se divide en cuatro fases diferentes, y cada una de estas obtiene un resultado de interés. A continuación, se realiza una breve descripción de cada una de ellas, así como del objetivo a lograr:

- **Inicio:** Se centra en hacer avanzar la idea inicial hasta un punto en el que esté lo suficientemente definida como para comprender el alcance que se trata de obtener. También es importante durante esta fase delimitar el ámbito del sistema propuesto, plantear un esbozo de la arquitectura que se pretende lograr, y encontrar los riesgos más acuciantes del proyecto.
- **Elaboración:** Es la fase donde la arquitectura es mayoritariamente definida, determinando las funcionalidades y características más significativas del sistema.
- **Construcción:** Durante esta fase se completa la elaboración del software a realizar, finalizando las tareas de análisis y diseño en el proceso y manteniendo una monitorización de los riesgos críticos para que puedan ser mitigados en caso de que se materialicen.
- **Transición:** Esta fase está centrada en poner el producto software en manos de los usuarios, así como de cerrar todas las necesidades propias del proyecto: ejemplo de esto podría ser la realización de un manual de usuario. Además, se extraen todos los conocimientos que se consideren significativos para futuros proyectos a realizar.

Para una más fácil gestión, cada una de las fases se divide en iteraciones. Cada una de estas iteraciones tiene una longitud variable dependiendo del tipo de proyecto, y sirven para facilitar la gestión. En proyectos grandes, una iteración puede durar hasta 12 semanas, mientras que en un proyecto reducido, como es el caso de este TFM, una o dos. Además, como es posible apreciar en la imagen 2.2, cada fase posee una cantidad diferente de iteraciones, dependiendo de la cantidad de trabajo que requieran dichas fases.

Para cada una de las disciplinas del SDLC, existen diferentes actividades definidas, cada una con el objetivo de transmitir cierta información significativa al equipo desarrollador. Esta información puede ser transmitida mediante documentos de texto, así como recurriendo a diagramas. A la hora de desarrollar este proyecto, se ha hecho especial énfasis en las actividades definidas para las disciplinas de requisitos y análisis, que pueden ser encontradas en los capítulos 3 y 4, respectivamente.

De forma previa al trabajo a realizar en estas dos disciplinas, también es importante definir el modelo del dominio; es decir, una red de objetos interconectados, donde cada uno de estos objetos representa un individuo significativo del sistema, que puede ser tan grande como una corporativa o tan pequeño como una línea de un pedido [19]. El modelo del dominio está compuesto por clases y sus diferentes tipos de relaciones, siendo las clases la descripción de uno de los objetos del dominio [20]. Es importante subrayar que la funcionalidad de las clases es meramente facilitar el entendimiento del sistema, por lo que es posible que, al momento de codificarse, algunas de esas clases se integren dentro de otras.

En lo que respecta a la disciplina de obtención de requisitos, las actividades a realizar son las siguientes:

- **Encontrar actores y casos de uso:** Partiendo del modelo del dominio previamente explicado, se trata de localizar todos los actores que se prevé interactúen con el sistema, así como los casos de uso que realizarán en el mismo. Esta actividad es una primera aproximación a los casos de uso, por lo que es posible que algunos de ellos no se mantengan o que surjan nuevos en etapas posteriores de la obtención de requisitos. Esta actividad se representa mediante un diagrama de casos de uso, en el cual se muestran las relaciones actor - caso de uso, actor - actor y caso de uso - caso de uso. En el apartado 3.2 es posible contemplar en diferentes diagramas todos los casos de uso y actores encontrados para el proyecto. Además, en esa misma sección es posible observar el diagrama de contexto, un diagrama de estados que determina cómo es posible moverse por los diferentes estados de la aplicación, estableciendo los casos de uso como las transiciones entre estados.

- **Priorizar casos de uso:** En esta actividad, se seleccionan los casos de uso de mayor riesgo y se priorizan para un más pronto desarrollo, implicando un análisis, diseño, desarrollo y probado más temprano.

- **Detallar casos de uso:** Tras realizar la priorización, los casos de uso seleccionados son detallados en profundidad, describiendo la secuencia de acciones que se tomará en el mismo, con todas sus posibles variaciones. Estas secuencias se describen utilizando para ello diagramas de estados. Se realiza una definición de caja negra; es decir, solo se mencionarán qué acciones se realizarán en base a las entradas y salidas de la aplicación, sin entrar a explicar el funcionamiento interno. No es necesario que, para concluir la fase de obtención de requisitos, todos los casos de uso sean detallados: basta con explicar

todos aquellos que posean un alto riesgo o que puedan tener complicaciones en ser entendidos. Es posible encontrar los diagramas de algunos casos de uso detallados en la sección 3.3.

- **Estructurar casos de uso:** Una vez detallados los casos de uso de mayor riesgo, y con el objetivo de reducir la redundancia, es posible extraer partes comunes de los casos de uso, para así facilitar entender estos, permitiendo modularizar mejor el trabajo para cuando se requiera realizar la implementación. Al estructurar los casos de uso, surgen dos tipos de relaciones entre estos: la inclusión y la extensión.

  - Un caso de uso incluirá a otro cuando uno de los casos de uso sea una parte de otro caso de uso, que es el todo, y no pueda ser considerado un caso de uso por sí mismo.
  - Por otro lado, una extensión representa que un caso de uso añade nuevas acciones a otro caso de uso. La diferencia con la inclusión radica en que el caso de uso extendido es también un caso de uso realizable por un actor.

  Es posible apreciar estas relaciones en los diagramas de especificación de la sección 3.3.

- **Prototipar la interfaz de usuario:** Por último, es adecuado también realizar bocetos de las diferentes pantallas que se encontrarán los actores al interactuar con el sistema, y que les permitirán realizar los casos de uso especificados. Los prototipos de las pantallas más significativas se hallan en la sección 3.4.

Por otro lado, también se han llevado a cabo actividades de la disciplina de análisis. Las actividades realizadas son enumeradas a continuación:

- **Análisis de la arquitectura:** En esta actividad, el objetivo principal es localizar todas las clases requeridas para llevar a cabo el Modelo-Vista-Controlador (MVC) de la arquitectura (este concepto se puede hallar explicado en el apartado 2.1.1). Cada uno de los tipos de clases proviene principalmente de un punto concreto de la obtención de requisitos: los modelos del modelo del dominio, las vistas de las interfaces de usuario y los controladores de los casos de uso. Las clases obtenidas han sido, posteriormente, agrupadas en paquetes para una más fácil gestión del propio software. Es posible encontrar el diagrama que representa todas las clases extraídas y clasificadas en paquetes en la sección 4.1.

- **Análisis de los casos de uso:** Tras haber localizado las diferentes clases del proyecto, se realiza un análisis de casos de uso, donde se determina cómo se comunican entre sí las diferentes clases necesarias para que dicho caso de uso sea llevado a cabo. Esto se realiza en un diagrama de colaboración, donde la importancia radica en la comunicación en sí, y no en la secuencia a seguir. El análisis de los casos de uso se realiza de igual forma que se realizó la especificación durante la obtención de requisitos: priorizando los casos de uso que han sido definidos de mayor riesgo, e igual que durante esa actividad, no es necesario realizar este diagrama por cada caso de uso, solo para aquellos que posean una mayor dificultad para ser entendidos. En la sección 4.2 es posible hallar los diagramas más significativos de esta actividad.

La razón de no haber elaborado ningún diagrama en las fases posteriores al análisis se debe a que, debido a que los integrantes de este proyecto habíamos adquirido el suficiente conocimiento en torno a los casos de uso y la arquitectura del sistema, no era necesario realizar más diagramas para una mayor documentación del sistema a obtener.

En caso de desear realizar una lectura en profundidad sobre RUP, se recomienda la lectura de [16, 20, 21, 22, 23].

2.1.1 Modelo-Vista-Controlador

Además de la metodología seleccionada, también es importante entender el patrón de diseño de software que se aplicará en este proyecto, el cual es conocido por el acrónimo MVC. Este es un paradigma para factorizar el código en segmentos funcionales para conseguir una mayor reusabilidad, manteniendo separados los modelos de datos, las vistas que posibilitan la entrada y salida de estos datos, y los controladores que comunican ambos entre sí [24].

El origen del MVC se remonta a 1979, cuando Trygve Reenskaug definió los conceptos de modelo, vista, controlador y editor. Su propuesta estaba orientada a evitar que los usuarios tuviesen que trabajar con conjuntos de datos grandes y complejos. Posteriormente, Adele Goldberg y el equipo de Smalltalk lo implementó para la librería Smalltalk-80 [25], y desde entonces el paradigma ha ido evolucionando conforme han ido cambiando las necesidades del software, pero siempre manteniéndose relevante.

Para conocer más de como surgió este paradigma y los diferentes planteamientos que se realizaron al ser concebido, se recomienda la lectura de [26]. En caso de querer aprender más sobre el patrón MVC, léase [27].

2.2 Tecnologías y Herramientas

En esta sección, se realizará una breve explicación de todas las tecnologías seleccionadas para el desarrollo del proyecto, y se mencionará la razón que las ha llevado a ser escogidas.

- **HTML5:** HyperText Markup Language (HTML) es el lenguaje más utilizado para representar el contenido de una página web. Este lenguaje es interpretado por el navegador del usuario para mostrar el contenido, definido mediante etiquetas. Su última versión, HTML5, es la que será utilizada en este proyecto.
- **SCSS:** Sassy Cascading Style Sheets (SCSS) es un superconjunto de Cascading Style Sheets (CSS). Este último es un lenguaje que permite alterar el aspecto que tendrán las etiquetas de HTML al ser mostradas. Normalmente, las hojas de estilo se definen en ficheros aparte para separar el contenido de la página web de su visualización. SCSS, por otro lado, añade más funcionalidades que CSS no posee de por sí, como la anidación del estilo de las etiquetas, entre otros [28]. Se ha decidido introducir SCSS en lugar de CSS para aprender a utilizar otro tipo de tecnología para el estilizado de las páginas web.
- **TS:** TypeScript (TS) es un lenguaje interpretado de programación utilizable tanto en el lado del cliente como en el del servidor de la aplicación. Cuando se utiliza en el lado cliente, permite integrar una capa de dinamismo en las páginas web. TS es un superconjunto de JavaScript (JS), ya que ambos cumplen la misma funcionalidad en un sistema, solo que el primero añade características que el segundo no tiene, como el tipado fuerte [29].

- **Angular:** Es un marco de trabajo, o *framework*, orientado a facilitar la creación de aplicaciones Single-Page Application (SPA), donde se reescribe el contenido de una página web al momento de cambiar de página, en lugar de cargar páginas enteras completas, con el objetivo de conseguir transiciones más veloces. Angular va a ser utilizada en el proyecto mediante la programación reactiva: en lugar de seguir el flujo síncrono usual de la programación funcional, donde se espera a obtener un resultado, en la programación reactiva el flujo es asíncrono. Esto significa que el resultado se obtendrá cuando finalice el flujo correspondiente, sin mantenerse la aplicación a la espera del resultado [30]. Además, este *framework* no aplica el patrón MVC explicado en el apartado 2.1.1. En su lugar, usa el patrón Model-View-ViewModel (MVVM). En este patrón, el controlador de MVC es sustituido por la vista del modelo, siendo este un estado de la información del modelo. Se ha decidido utilizar este *framework* ya que actualmente es uno de los más utilizados del mercado para el desarrollo de aplicaciones web.

- **Angular Material:** Esta es una librería destinada a facilitar el desarrollo de la aplicación. A diferencia de otras librerías similares, Angular Material no es una librería de terceros (siendo parte de Angular) y, además, está destinada a proporcionar componentes completos para la web; es decir, el componente añadido contiene el HTML, CSS y TS necesarios. Un ejemplo de un componente de Angular Material es un calendario de selección de fecha, pero existen muchos más [31].

- **NodeJS:** NodeJS es un entorno de JS multiplataforma, orientado a eventos y asíncrono, destinado a la implementación de servidores web. NodeJS es muy utilizado actualmente, y al ser posible trabajar también con TS, simplifica la implementación de la aplicación [32].

- **Express:** Este *framework* es el más utilizado para la creación de servidores en NodeJS, permitiendo la creación de APIs *RESTful* de manera sencilla. Además, simplifica las tareas de validación y autenticación gracias a su capacidad de establecer diferentes *middlewares* a las rutas de la API [33].

- **npm:** Este gestor de paquetes habilita la instalación y uso de cualquier librería de terceros al trabajar con NodeJS. La utilización de npm facilita integrar nuevas librerías al proyecto, simplificando su desarrollo y expansión [34].

- **Jest:** La funcionalidad de este *framework* está enfocada a facilitar la realización de diferentes test para entornos JS, entre los que se encuentra NodeJS [35]. Se utilizará para llevar a cabo los test necesarios (como test unitarios o de integración) en back-end, para asegurar una amplia cobertura de código. Esto permitirá, además, aprender una forma de realizar tests en entornos JS.

- **Jasmine:** Esta librería es la utilizada por defecto para realizar test de Angular, y es altamente similar a Jest en lo que a sintaxis de los test se refiere, lo que facilita el desarrollo de los test a realizar en front-end [36]. Se ha optado por la utilización de esta librería ya que permitirá conocer los pormenores de la realización de test en front-end.

- **Stryker Mutator:** La utilidad de esta librería es facilitar la realización de test que modifiquen el funcionamiento del sistema, conocidos como test de mutación, que sirven para comprobar la validez de los test creados [37]. Se optado por introducir esta librería con el fin de asegurar la alta calidad de los test.

- **GitHub:** Es una plataforma, orientada a desarrolladores, que permite almacenar y gestionar proyectos software, facilitando además el control de versiones para una implementación más segura [38]. El proyecto se alojará dentro de un repositorio de GitHub, y los desarrolladores irán agregando cambios mediante *commits*: cada uno de estos *commits* será una versión del código de la aplicación.

- **GitHub Actions:** Es una herramienta que forma parte de GitHub y tiene como objetivo facilitar el uso de Continuous Integration / Continuous Deployment (CI/CD) en un proyecto. Mediante GitHub Actions, es posible validar y desplegar la aplicación de forma automatizada tras cada *commit*, haciendo que el desarrollo sea mucho más eficiente.

- **SonarCloud:** SonarCloud es una herramienta que permite comprobar la calidad del código desarrollado. Gracias a esto, es posible conocer vulnerabilidades del sistema o *code smells*, entre otros [39]. Este último concepto se refiere a código implementado que, sin ser un error, puede delatar un problema del propio código, como puede ser el código muerto o código duplicado, por ejemplo.

- **MongoDB:** MongoDB es un sistema de gestión de BDs No Structured Query Language (NoSQL) documental, donde la información, en lugar de ser almacenada en tablas relacionales, se conserva en ficheros JSON estructurados [40]. La utilización de este programa es consecuente al deseo de aprender a utilizar un paradigma distinto a SQL para la creación de BDs.

- **Mongoose:** Mongoose es un Object-Document Mapper (ODM) basado en esquemas que permite modelar la información de una aplicación, permitiendo además la creación y validación de consultas a una BD basada en MongoDB [41]. En este proyecto, la utilización de Mongoose será troncal, ya que además de utilizarse para definir la BD del proyecto, su uso también incluirá la gestión de los datos al atender a consultas de la API.

Capítulo 3. Obtención de Requisitos

Tras conocer la situación actual en relación a la repetición espaciada, habiendo establecido los objetivos del proyecto y teniendo especificado el marco teórico de la aplicación, es posible comenzar a aplicar RUP, ejecutando la obtención de requisitos del sistema a realizar.

3.1 Modelo del Dominio

Tal y como se ha mencionado en el marco teórico, la primera tarea durante la obtención de requisitos del sistema es la obtención del modelo del dominio, donde se pone en conjunto todo el vocabulario específico del software a implementar para facilitar el entendimiento entre el equipo de desarrollo. En la siguiente imagen, es posible apreciar el modelo del dominio planteado para este TFM:

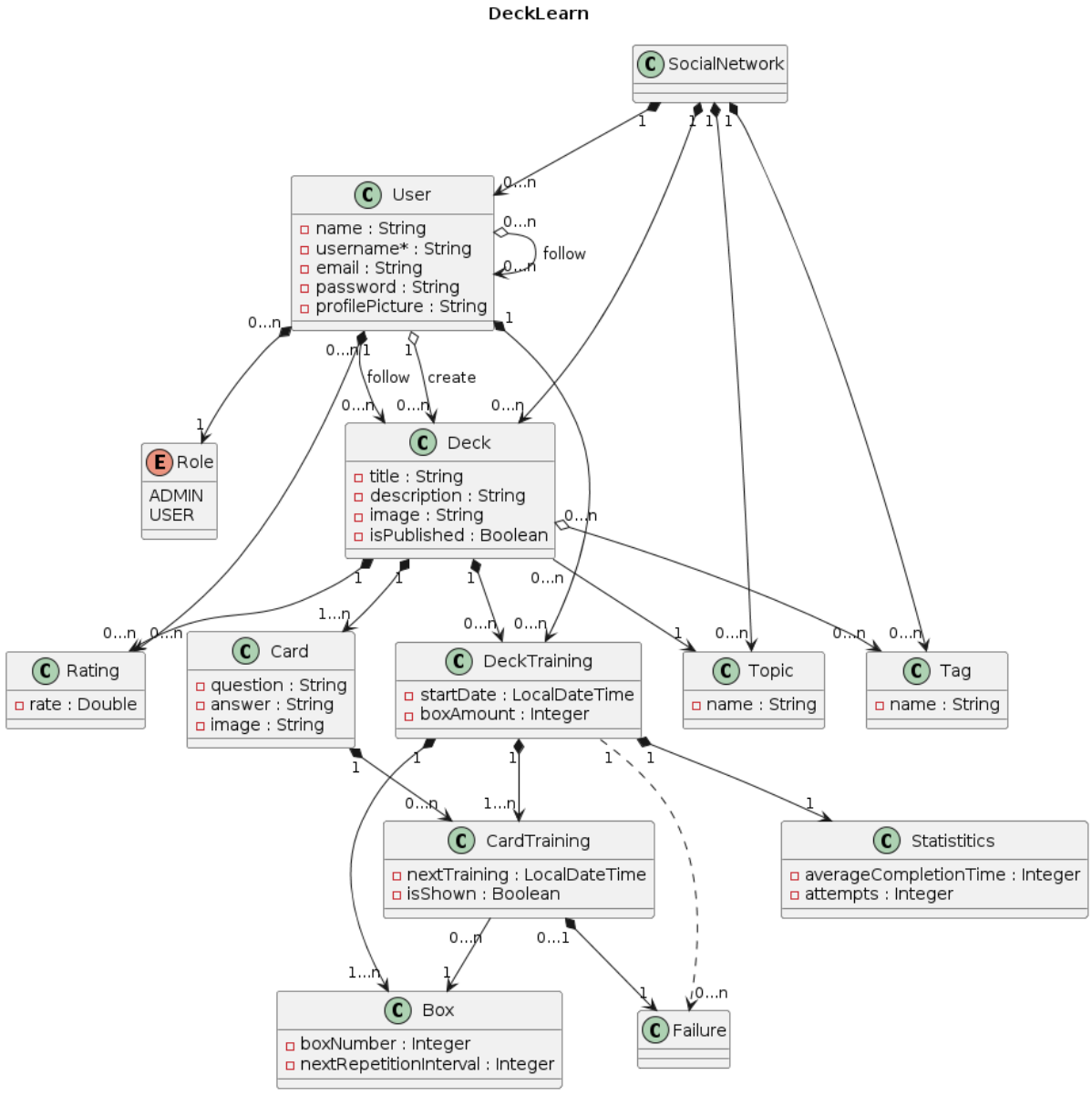


Figura 3.1: *Modelo del dominio*

Este diagrama representa todas las clases del sistema y sus relaciones entre ellas. A continuación, se ha realizado una explicación de cada una de estas clases, así como de las relaciones:

- La clase *SocialNetwork* representa al sistema en su totalidad, y alberga dentro de sí todas las entidades necesarias para conocer la información de la aplicación: usuarios, mazos, temas y etiquetas.
- La clase *User* representa a todas las personas que vayan a interactuar con el sistema, y almacena toda la información necesaria para su identificación y gestión.
- La enumeración *Role* establece los diferentes roles de los usuarios de la aplicación, lo cual condicionará las diferentes funcionalidades a las que podrá acceder cada usuario.
- La clase *Deck* representa un mazo del sistema Leitner de repetición espaciada.
- Los usuarios pueden crear diversos mazos.
- Los usuarios pueden seguir distintos mazos.
- Los usuarios pueden seguir a otros usuarios.
- La clase *Card* representa una pregunta y su consiguiente respuesta en lo respectivo a la repetición espaciada.
- Un mazo está compuesto de distintas cartas.
- La clase *Topic*, o tema, representa distintos temas de estudio, definidos por un administrador, para ayudar a otros usuarios a conocer el contenido del mazo.
- A un mazo se le puede asociar un tema.
- La clase *Tag*, o etiqueta, son similares a los temas, en tanto que sirven para dar a entender el contenido del mazo, pero estas etiquetas pueden ser definidas por los usuarios para un entendimiento más amplio del mazo, sin estar restringidas a una selección limitada proporcionada por el administrador. Las etiquetas permiten especificar mejor el tema del mazo, o bien expresar la dificultad, idioma, o lo que considere importante el creador del mazo.
- Un mazo puede tener asociadas distintas etiquetas.
- La clase *Rating* es la valoración que un usuario ha realizado sobre un mazo concreto.
- Un usuario puede valorar los mazos que siga.
- La clase *DeckTraining* representa el estudio de un mazo, y la clase *CardTraining* representa el de una carta.
- Un usuario puede realizar un estudio de mazo usando un mazo concreto.
- Un estudio de mazo está compuesto por los estudios de cartas de cada una de las cartas que componen dicho mazo.

- La clase *Box* representa una caja del sistema Leitner de un estudio y su correspondiente intervalo. En esta caja se colocarán las cartas estudiadas que correspondan. La cantidad de cajas es definida por el estudio de mazo.
- La clase *Failure* representa el fallo de una carta a la hora de ser estudiada, lo cual repercute en el estudio del mazo.
- La clase *Statistics* representa las estadísticas de un usuario en el estudio de un mazo.

Aunque no es necesario realizarlo, se han determinado algunos de los atributos de estas clases. Esto se debe a que se consideraba importante definir algunos de estos datos para un mayor entendimiento. En otros casos, sin embargo, se debe a que estos atributos han surgido durante el desarrollo de otras actividades de RUP. Es importante matizar que no todos los atributos han sido identificados: es posible que alguno de los atributos finales de alguna clase no aparezcan en el diagrama.

3.2 Encontrar Actores y Casos de Uso

Tras haber definido el modelo del dominio y comprender el vocabulario básico dentro del sistema, el siguiente punto es encontrar los casos de uso del mismo. A continuación, se irán mostrando todos los diagramas donde se han extraído los actores y casos de uso del sistema, y se realizará una explicación de los mismos para comprender lo que representan.

Figura 3.2: *Relación entre los actores de los casos de uso*

En esta imagen, se puede apreciar que se han definido dos actores para el sistema; por un lado, están los usuarios, que son la mayoría de los que interactuarán con la aplicación, y por el otro lado, los administradores, que podrán realizar los mismos casos de uso que los usuarios, pero tendrán algunos propios.

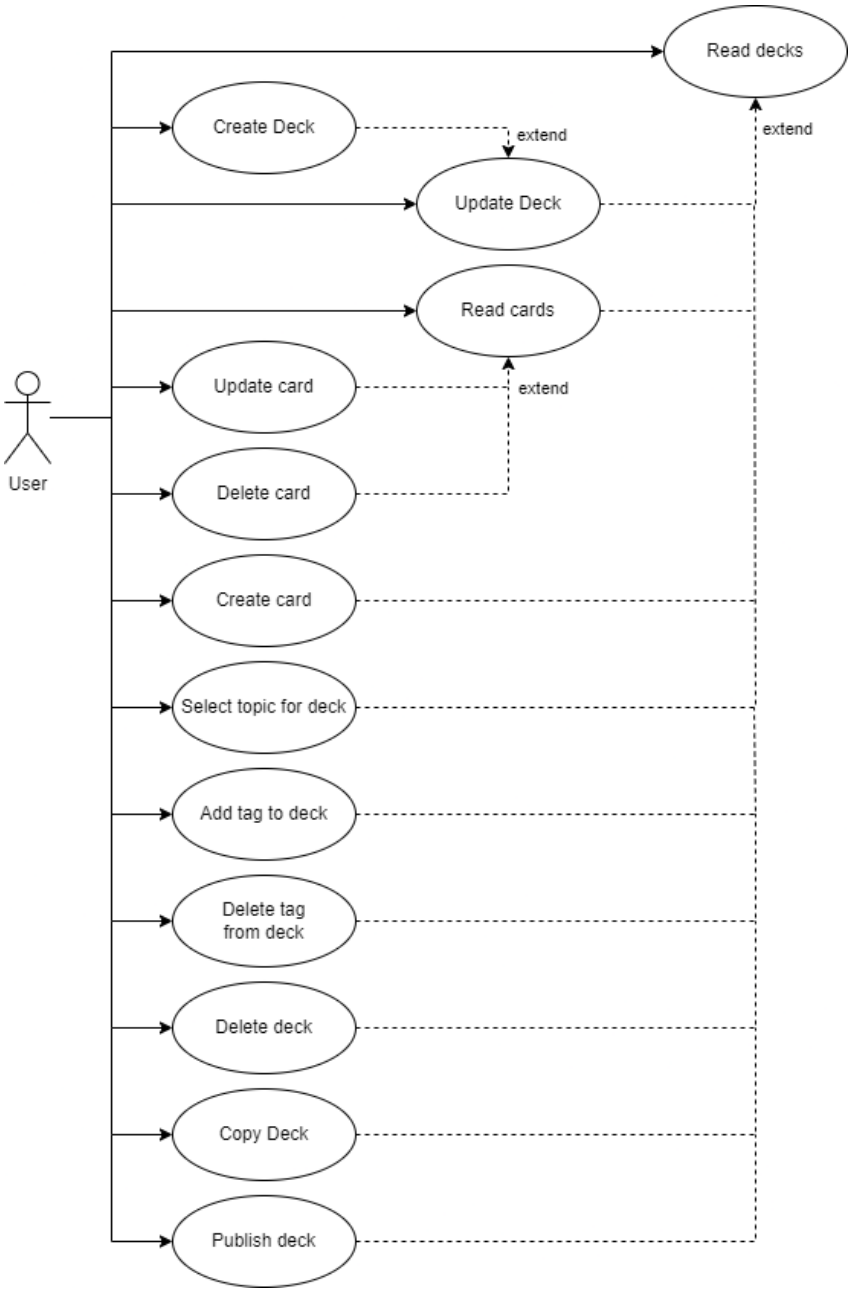


Figura 3.3: *Casos de uso centrados en torno a la creación y edición de mazos*

Los casos de uso del diagrama previo son todos aquellos que tienen relación con el proceso de creación y edición de los mazos de estudio, entre los que se incluyen:

- La creación, actualización y borrado de mazos, así como de las cartas que los componen.
- Añadir o eliminar temas o etiquetas a los mazos. Al añadir etiquetas, si estas no existen en el sistema, se crearán automáticamente.
- Publicar un mazo, tras lo cual no podrá ser editado.
- Copiar un mazo, para permitir realizar modificaciones a mazos ya publicados.

Como se puede apreciar, algunos casos de uso extienden a otros caso de uso: esto se debe a que, al realizar el primer caso de uso, se realizan las actividades del caso de uso extendido, junto con algunas acciones propias.

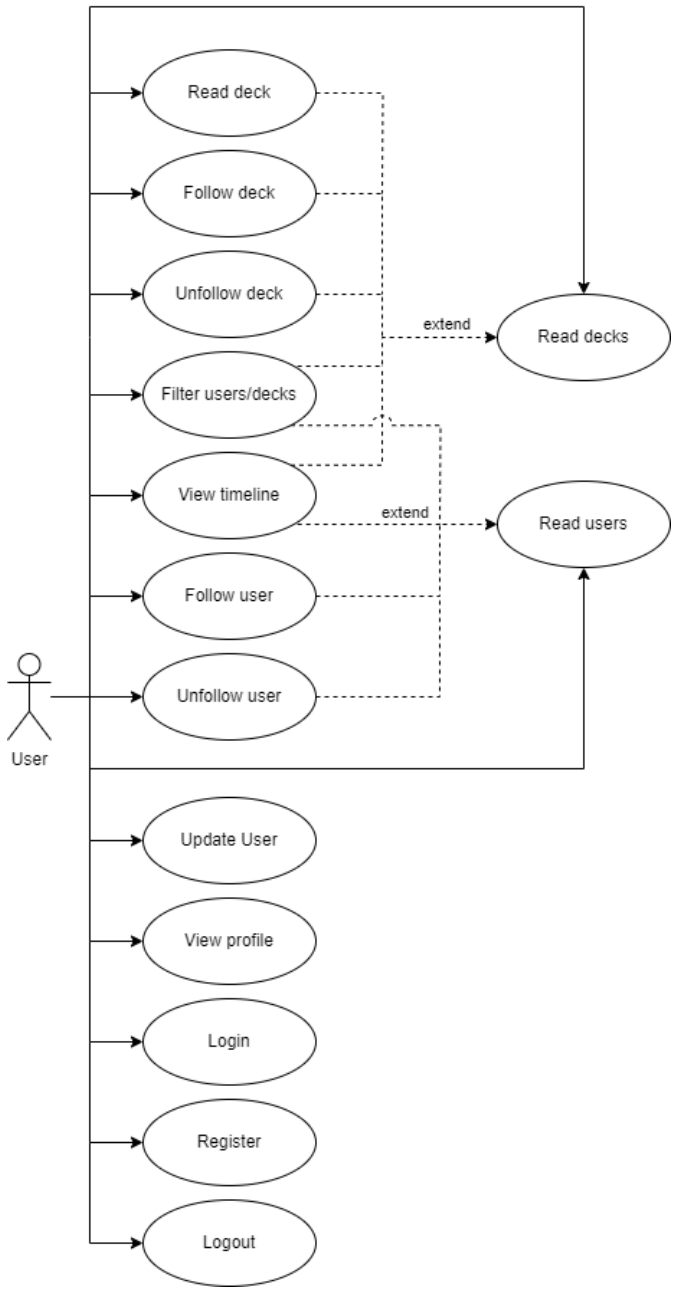


Figura 3.4: *Casos de uso centrados en el aspecto de red social de la aplicación*

En lo que concierne a la red social, existen varios casos de uso que describen funcionalidades a poder realizarse:

- Permitir ver la información de un mazo.
- La posibilidad de seguir o dejar de seguir el mazo de otro usuario.
- Realizar una búsqueda de usuarios y/o mazos en base a filtros.

- Ver la timeline propia.
- Seguir o dejar de seguir usuarios.
- Actualizar un usuario. Este caso de uso hace referencia a la acción de alterar los datos personales.
- Ver el perfil propio.
- Registrarse en la aplicación, así como iniciar o cerrar sesión.

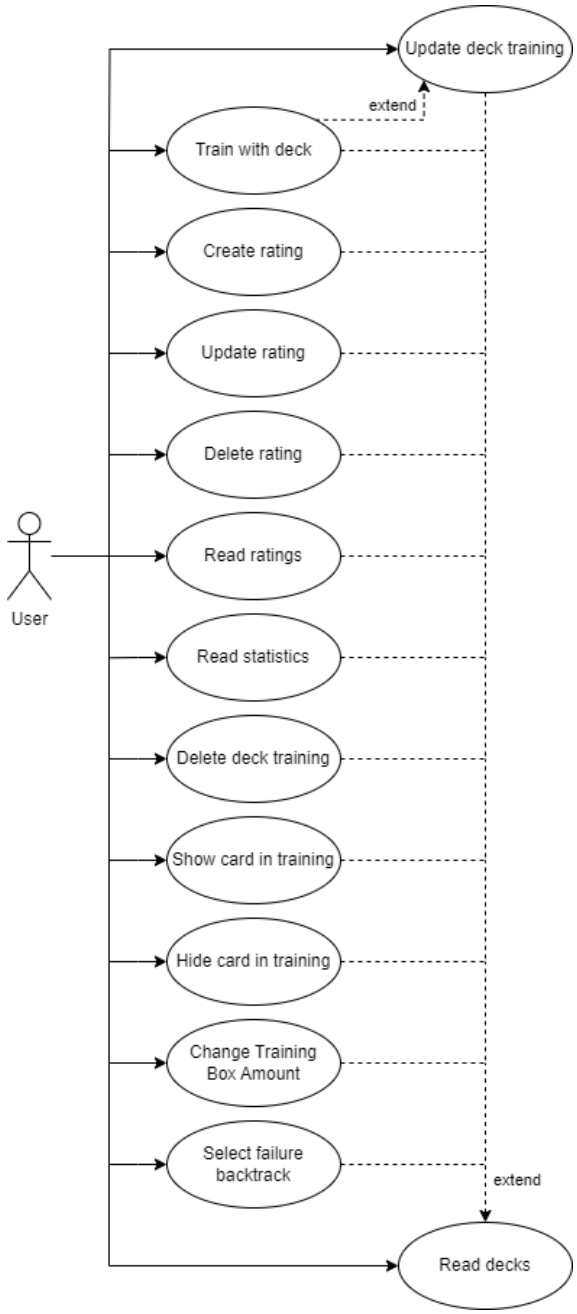


Figura 3.5: *Casos de uso centrados en el entrenamiento de mazos*

Por otro lado, tenemos los casos de uso relacionados con el entrenamiento de mazos, tanto propios como de usuarios ajenos, que han sido planteados para la aplicación:

- Realizar un estudio de mazo. Al finalizar el aprendizaje, se actualizará el estudio del mazo.
- Crear, actualizar o borrar la valoración realizada por un usuario a un mazo, así como ver las valoraciones ajenas.
- Ver las estadísticas del estudio del mazo.
- Eliminar un estudio de mazo.
- También es posible actualizar el estudio del mazo directamente, para así poder reiniciarlo.
- La posibilidad de mostrar o esconder una carta durante el entrenamiento, para así no estudiarla.
- Seleccionar la cantidad de cajas del estudio del mazo.
- Seleccionar la cantidad de retroceso con respecto a cajas a la hora de cometer un fallo.

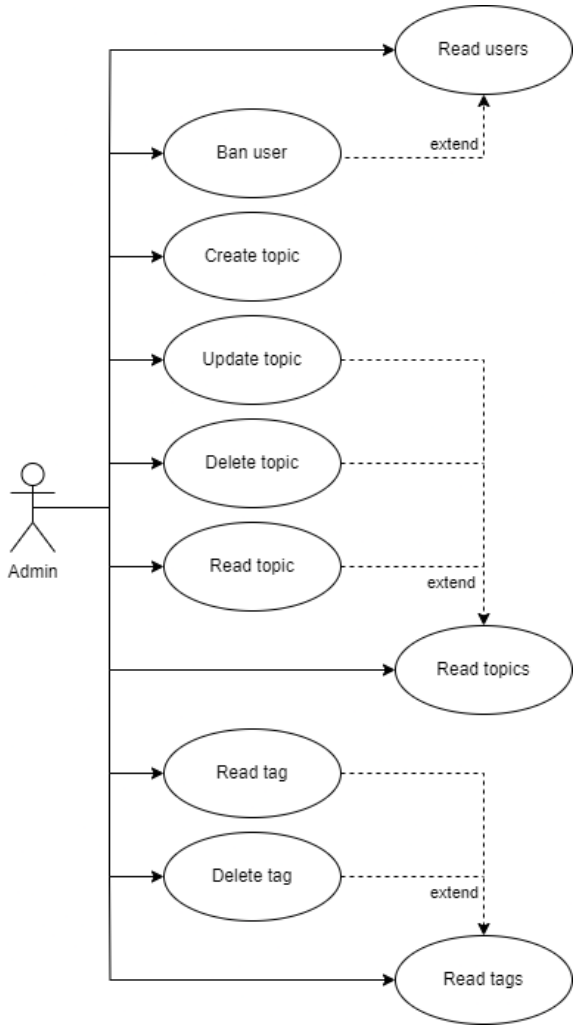


Figura 3.6: *Casos de uso reservados para usuarios administradores*

Por último, existen los casos de uso que tan solo pueden realizar los administradores, recogidos en el diagrama previo, y explicados a continuación:

- Prohibir el acceso a usuarios que realicen acciones deplorables.
- Definir, actualizar y borrar un tema.
- Borrar etiquetas creadas por usuarios.

Con todos los casos de uso definidos, se ha llevado a cabo un diagrama de contexto, donde se recogen todos los estados del sistema y como moverse a través de ellos mediante los casos de uso. A continuación es posible apreciar este diagrama:

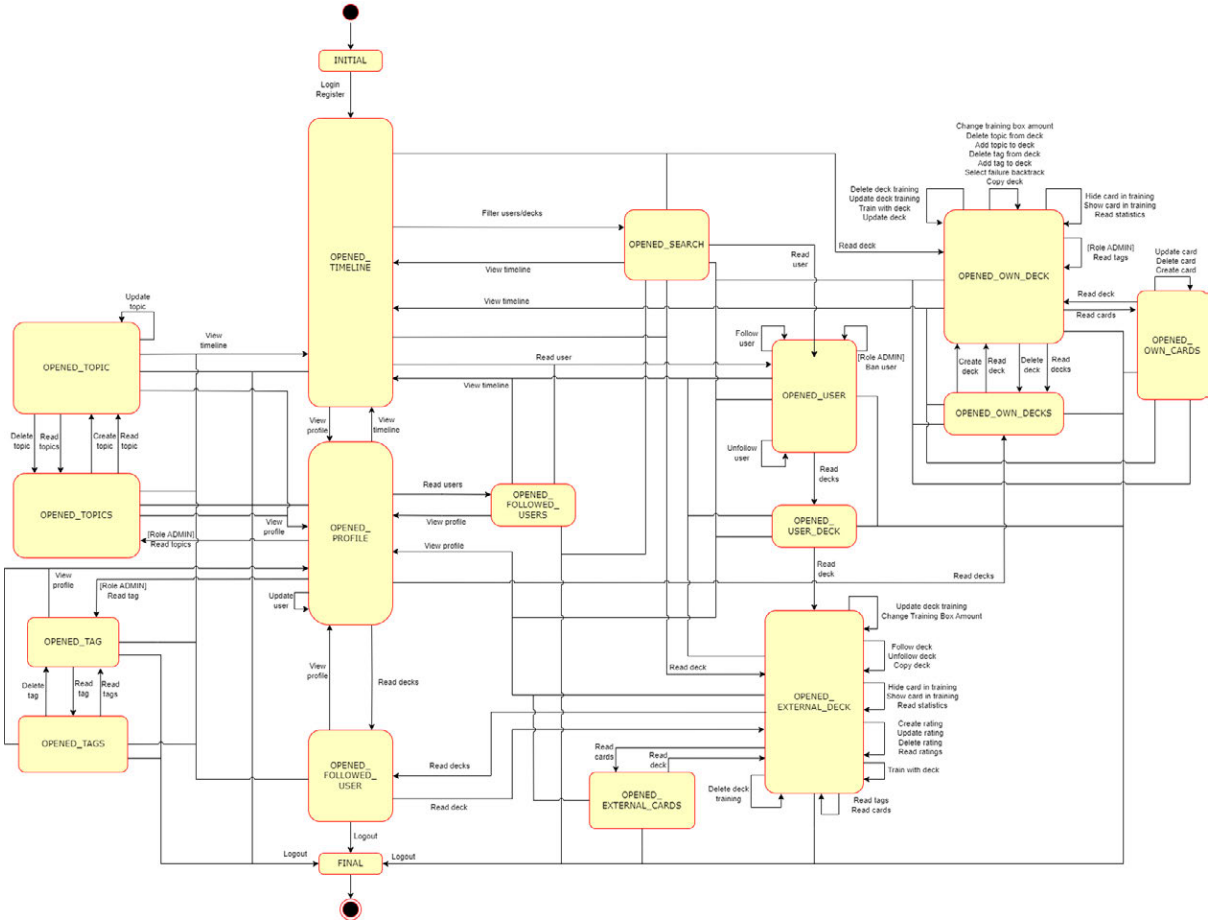


Figura 3.7: *Diagrama de contexto de la aplicación*

3.3 Especificación de Casos de Uso

Tras encontrar todos los actores y casos de uso, el siguiente paso es priorizar algunos de estos para realizar una especificación de los mismos, y así comprender correctamente las implicaciones y el alcance de dichos casos de uso.

A continuación, se presentan los diagramas de secuencia que especifican el funcionamiento de los casos de uso priorizados más significativos.

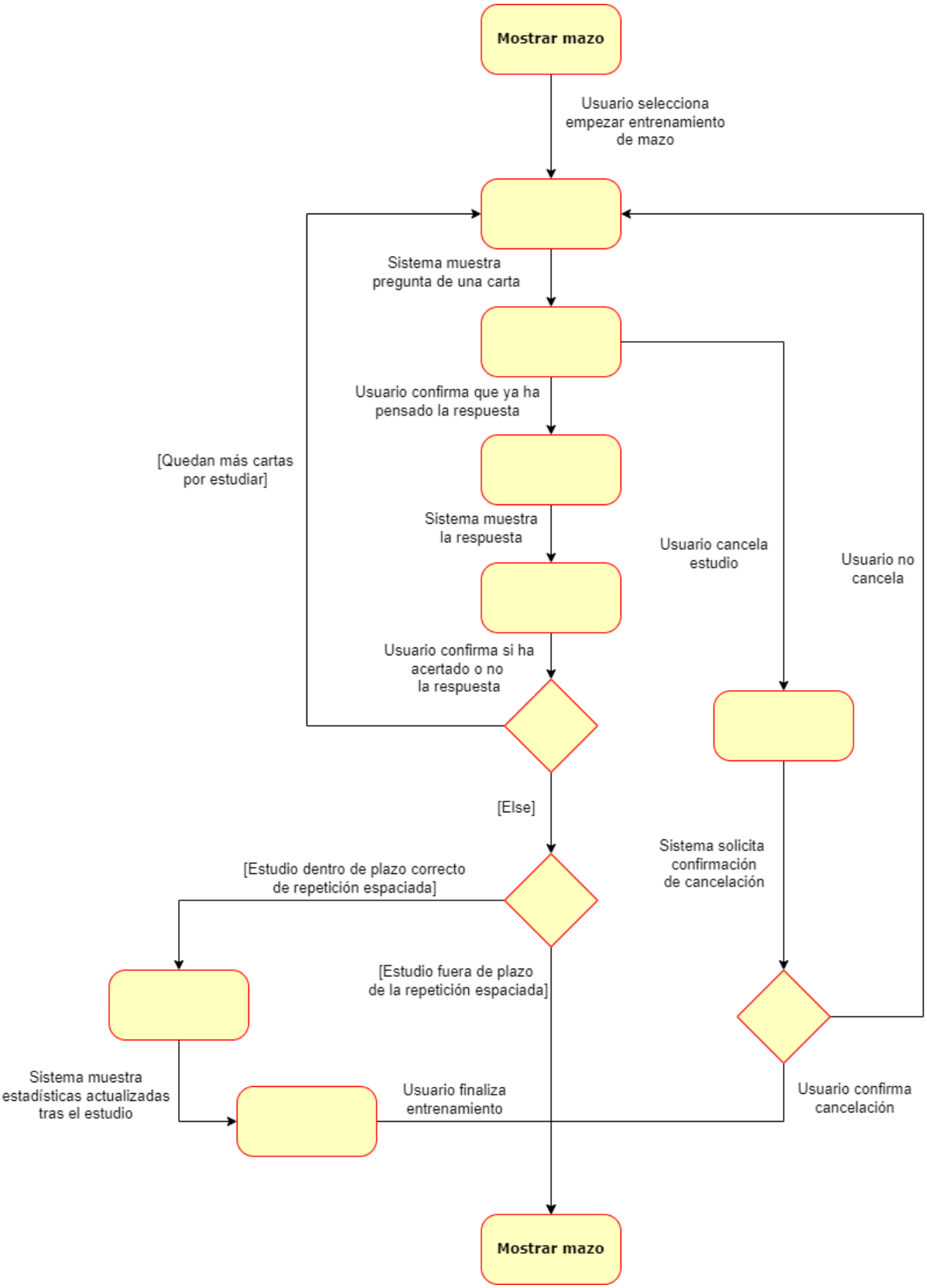


Figura 3.8: *Especificación del caso de uso entrenar con mazo*

El primer caso de uso que se decidió priorizar fue entrenar con un mazo. Este caso de uso se priorizó para que cualquiera que trate de entender los entrenamientos de repetición espaciada

basados en el sistema Leitner pueda comprender de qué se está hablando. Además, debido a todos los caminos posibles a seguir y todas las acciones que deben realizarse durante este caso de uso, su especificación permite realizar una explicación completa del mismo.

Gracias a este diagrama, es posible comprender que se han concebido los entrenamientos dentro del plazo de la repetición espaciada, y aquellos que el usuario realiza voluntariamente fuera de plazo. Además, estos últimos no muestran estadísticas, debido a que no cuentan a la hora de la computación de estas.

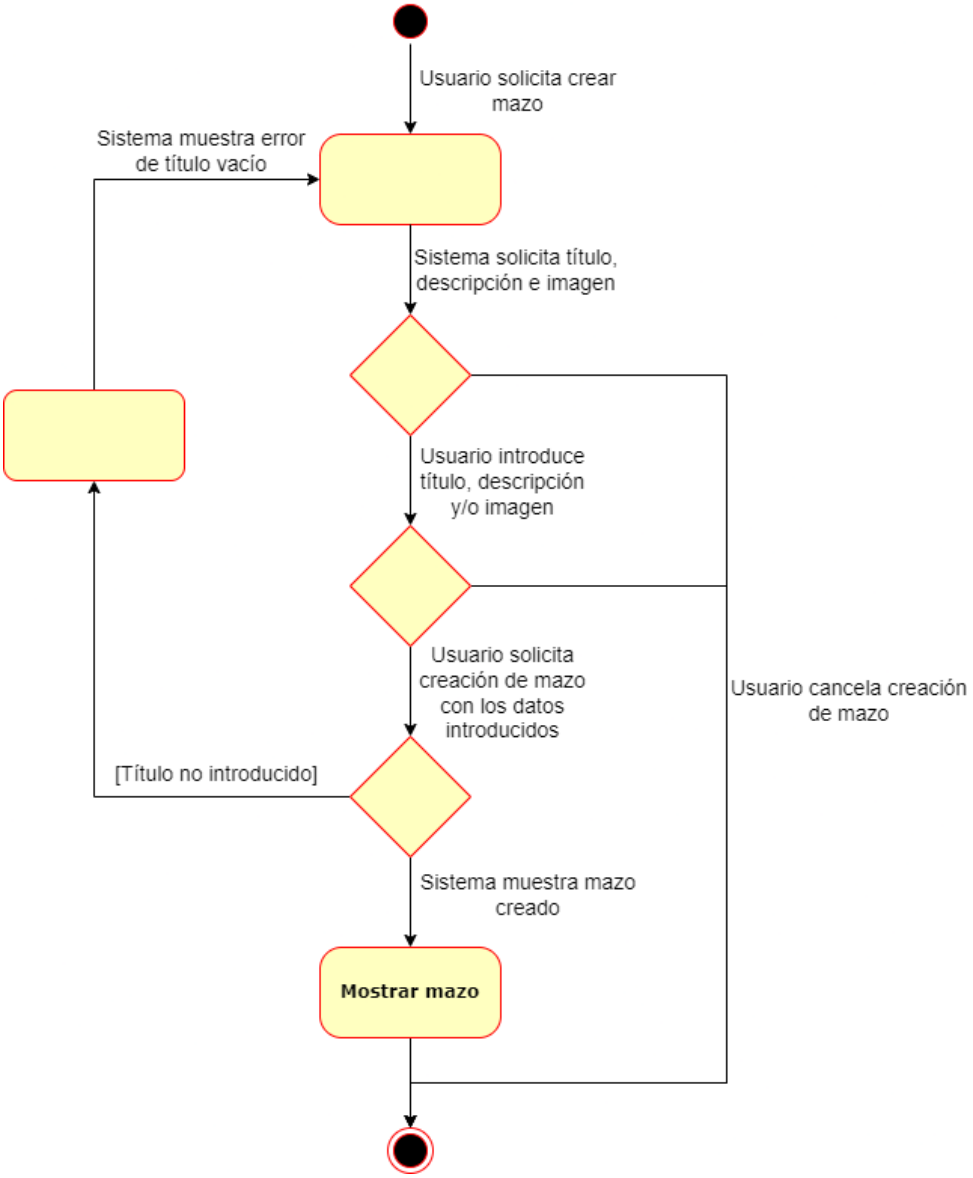


Figura 3.9: *Especificación del caso de crear mazo*

Se le dio importancia al caso de uso de crear mazo debido a que, en un inicio, este caso de uso poseía una gran cantidad de funcionalidad, y se consideró importante describirlo para poder entenderlo. Sin embargo, posteriormente el caso de uso se dividió en varios, debido a que era demasiado complejo, para que así fuese más fácil gestionarlo. Lo que se puede apreciar en la imagen es lo referente únicamente a la creación de los datos propios del mazo. Las especificaciones del resto de casos de uso que surgieron tras dividir la funcionalidad de los entrenamientos se encuentran en el Anexo B.

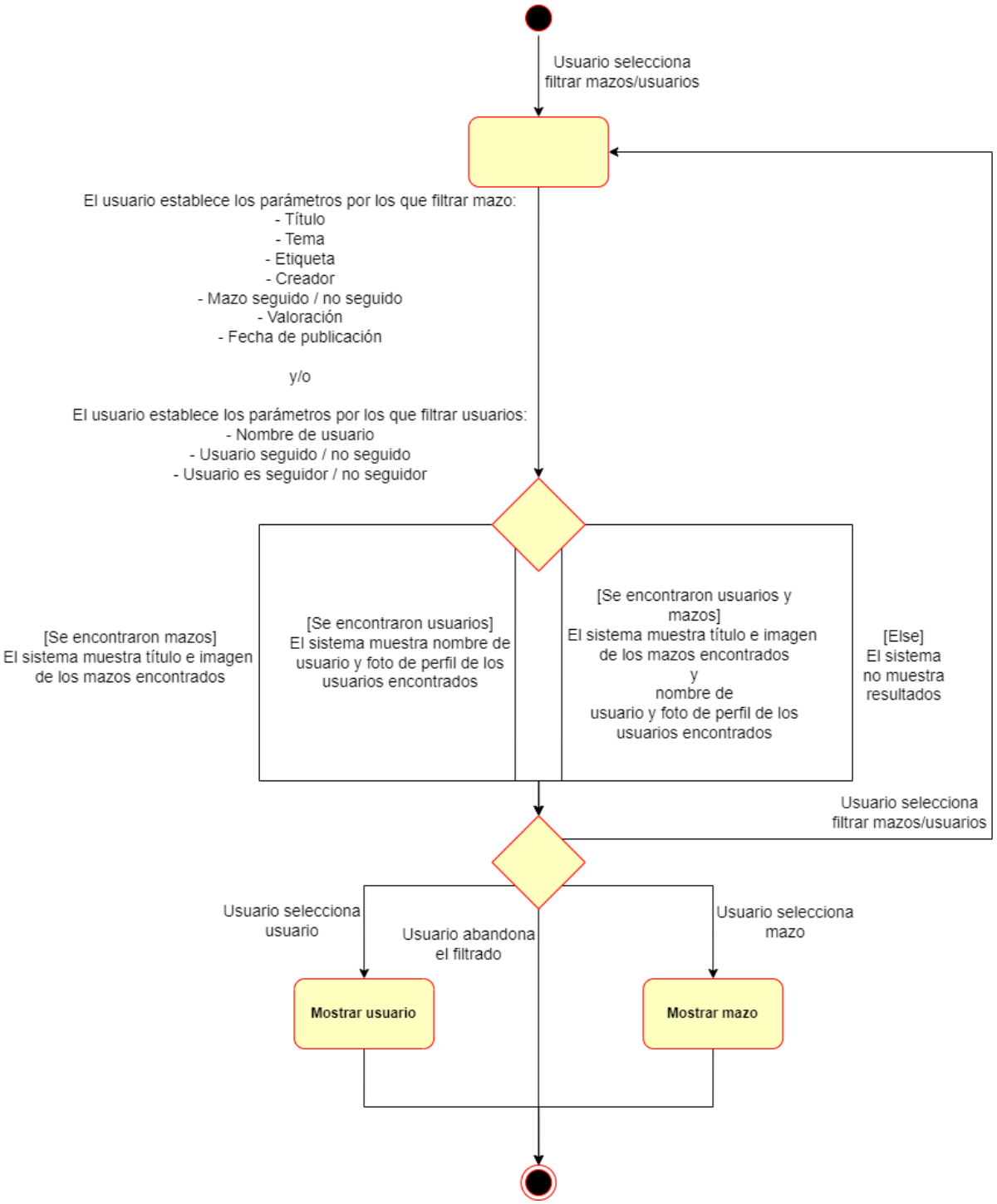


Figura 3.10: *Especificación del caso filtrar usuarios y mazos*

En lo que respecta al caso de uso de filtrar usuarios y mazos, la razón principal de su priorización se debe a que se consideró importante definir todos los posibles filtros que se podrán aplicar tanto a los usuarios como a los mazos.

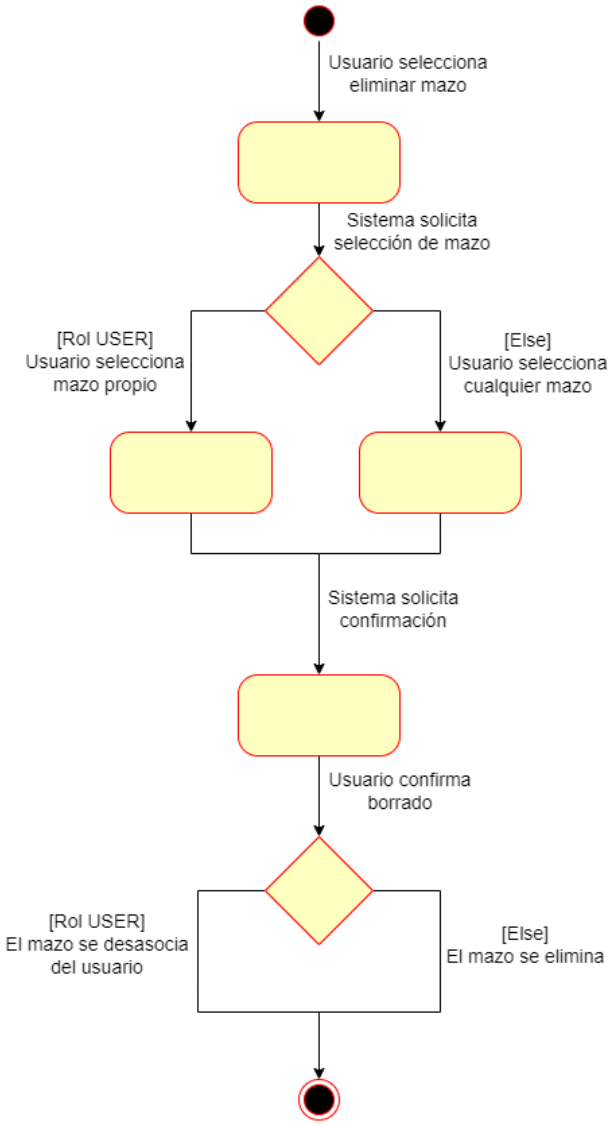


Figura 3.11: *Especificación del caso de uso borrar mazo*

Por último, la priorización del caso de uso relacionado al borrado de mazos se debe no a la complejidad del propio caso, ya que de por sí es considerablemente simple, sino que la razón es que dependiendo del rol que posea quién realice el borrado, la acción tendrá consecuencias totalmente distintas en el sistema, por lo que es necesario especificar estas diferencias para entender en su totalidad el caso de uso.

El planteamiento de borrado de mazos que se ha establecido define que, si un usuario borra un mazo, el cual será propio, este no se borrará del sistema. Simplemente dejará de estar ligado a él, pero el resto de usuarios podrán seguir usándolo en sus entrenamientos. Sin embargo, si el borrado lo realiza un administrador, implica que el borrado se debe a cuestiones más serias (un mazo con contenido inadecuado para la aplicación, por ejemplo), por lo que el mazo se elimina de forma total para todos los usuarios.

3.4 Interfaces de usuario

Para concluir con la fase de obtención de requisitos, se han creado las interfaces de usuario para diferentes casos de uso. Las interfaces creadas, sin embargo, no han sido exclusivamente de los casos de uso priorizados, ya que algunos de estos (borrar mazos, por ejemplo), tan solo serán un botón en una interfaz, por lo que no albergan complicación. Por tanto, las interfaces de usuario definidas son de aquellos casos de uso que deban mostrar una mayor cantidad de información o que sean más complejos. Una vez las interfaces más complejas estén definidas, el resto de pantallas simplemente deberán basarse en estas, de tal manera que se obtenga un aspecto uniforme de la aplicación. Todas las interfaces que se muestran a continuación han sido creadas utilizando la herramienta Figma. Es importante subrayar que no todas las pantallas prototipadas se muestran a continuación: por cuestiones de legibilidad, se han seleccionado las más relevantes. El resto de pantallas se pueden encontrar en el fichero Anexo A enviado junto con este documento.

Aunque estos diseños son tan solo prototipos, por lo que podrán acabar variando en su versión final, la intencionalidad detrás de definir el aspecto de las interfaces se halla en que se desea comprobar que se proporciona una User eXperience (UX) cómoda al usuario. Para ello, además de asegurar que cada pantalla permite la realización de uno o varios casos de uso de forma sencilla, se le ha prestado especial importa a labores tales como:

- La creación de una paleta de colores.
- Facilitar la navegación entre las diferentes interfaces.
- Introducción de ayudas al usuario en las pantallas que lo requieran.
- La legibilidad del texto.

Gracias a estas y otras acciones, se asegura que las pantallas podrán ser utilizables para los usuarios. Por ejemplo, como es posible observar en las capturas que se proporcionarán a continuación, moverse a través de las interfaces se ha posibilitado mediante el uso de una barra de navegación en la parte superior de las pantallas, que permite el acceso a las interfaces más significativas para el usuario.

En lo que respecta a la paleta de colores, se han establecido los cinco colores principales que utilizará la aplicación, y que son mostrados a continuación:

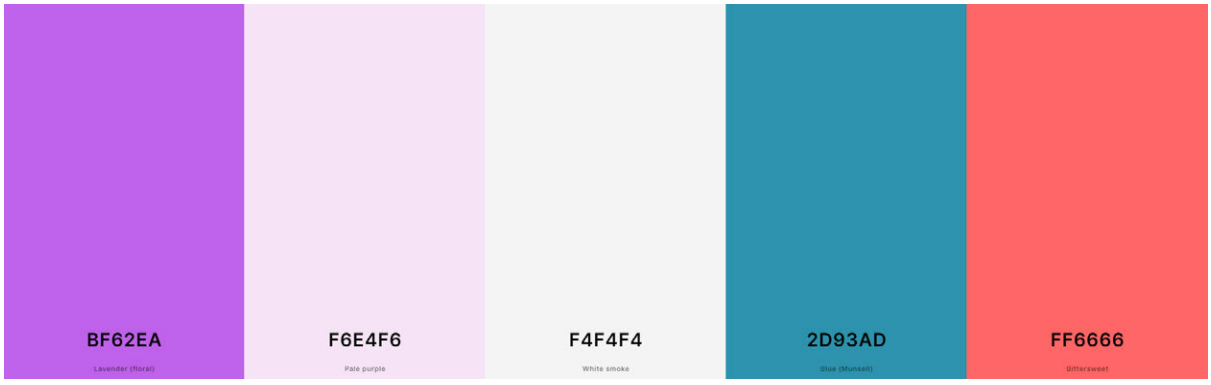


Figura 3.12: *Paleta de colores de IntelliDeck*

Cada uno de estos colores está pensado para ser utilizado en diferentes situaciones, listadas a continuación. Sin embargo, es importante matizar que, aunque los cinco colores antes mostrados son los principales, pequeñas variantes de estos son utilizados para detalles o resaltar contenido en lugares concretos.

- **#BF62EA:** Este color es el principal de IntelliDeck, y se utiliza para resaltar contenido, los detalles, el logo, etcétera.
- **#F6E4F6:** Este es el color secundario de la aplicación, que permite añadir más tonalidades de color a las pantallas, y es utilizado principalmente cuando se quiere resaltar algo de una importancia inferior.
- **#F4F4F4:** Es el tono de blanco seleccionado para la aplicación.
- **#2D93AD:** El color utilizado para representar que ciertas acciones se han realizado correctamente.
- **#FF6666:** Este color está destinado mayoritariamente a los errores y botones con acciones destructivas.

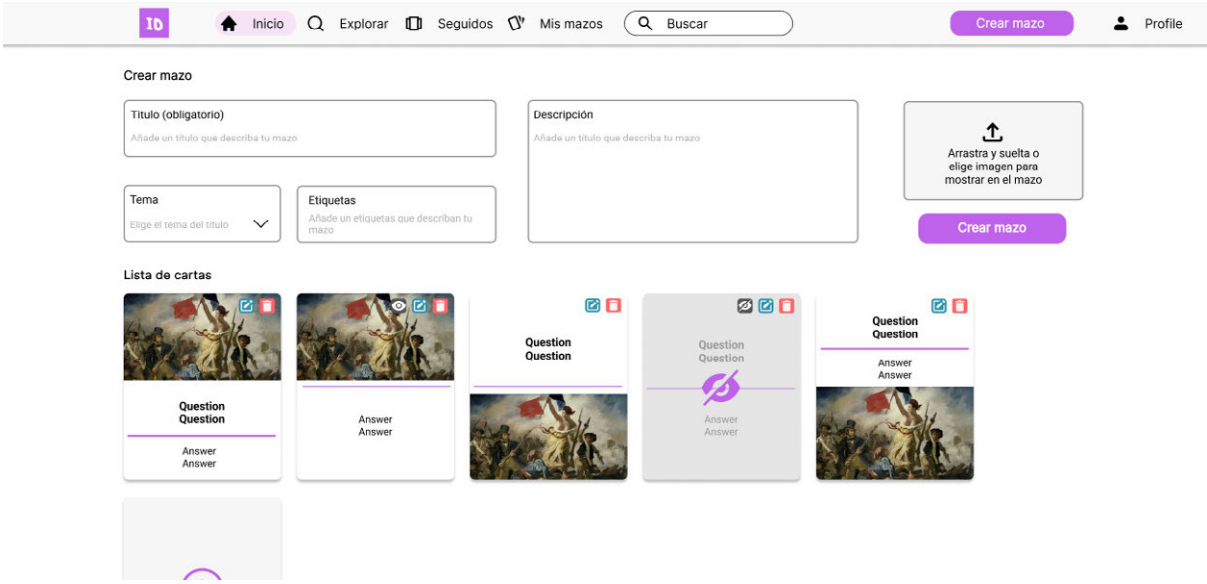


Figura 3.13: *Prototipo de la interfaz de usuario para la pantalla de crear mazo*

La primera pantalla en ser prototipada fue la que posibilita la creación de los mazos. Sin embargo, como se puede apreciar al contemplar la interfaz, esta engloba varios casos de uso dentro de sí, no centrándose solo en la creación de mazo. Así, esta pantalla permite la realización de los siguiente casos de uso:

- Crear mazo
- Actualizar mazo
- Crear carta

- Actualizar carta
- Borrar carta
- Seleccionar tema del mazo
- Añadir etiquetas al mazo
- Eliminar etiquetas del mazo
- Mostrar carta
- Ocultar carta

De esta forma, se asegura que la interfaz contiene suficiente funcionalidad, asegurando un buen UX.

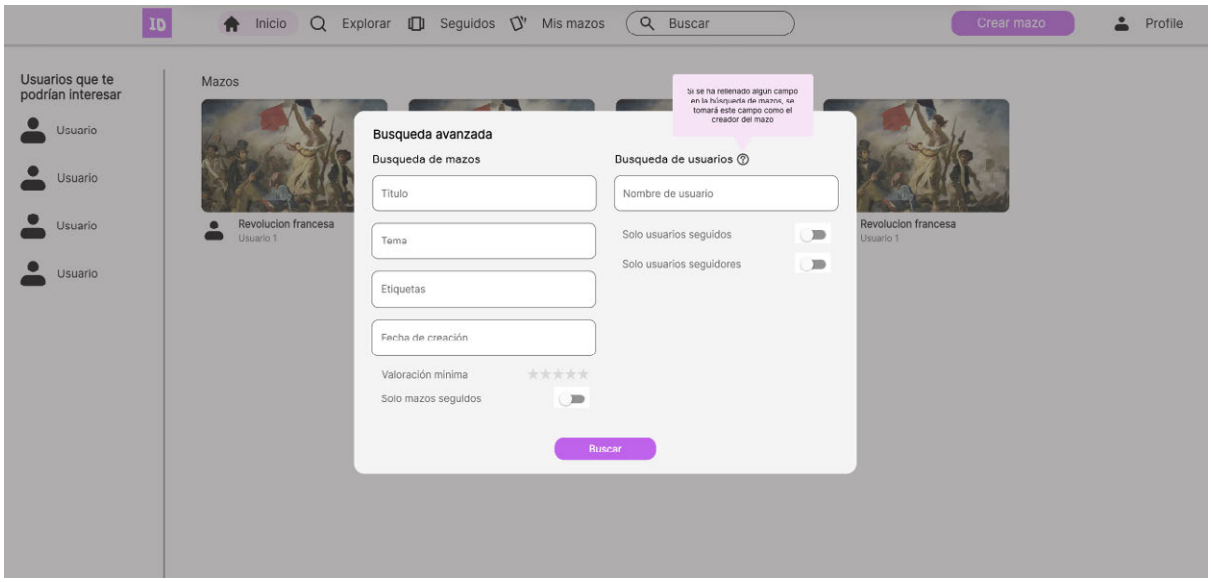


Figura 3.14: *Prototipo de la interfaz de usuario para la pantalla de filtrado de usuarios y mazos*

En esta pantalla, por ejemplo, es posible apreciar las ayudas mencionadas al hablar de UX: gracias a estas, el usuario podrá entender como se tratará el campo de “nombre de usuario” cuando este sea introducido. Además, también es posible verificar que se han tenido en cuenta todos los filtros establecidos al realizar la especificación del caso de uso de filtrado, visibles en la imagen 3.10.

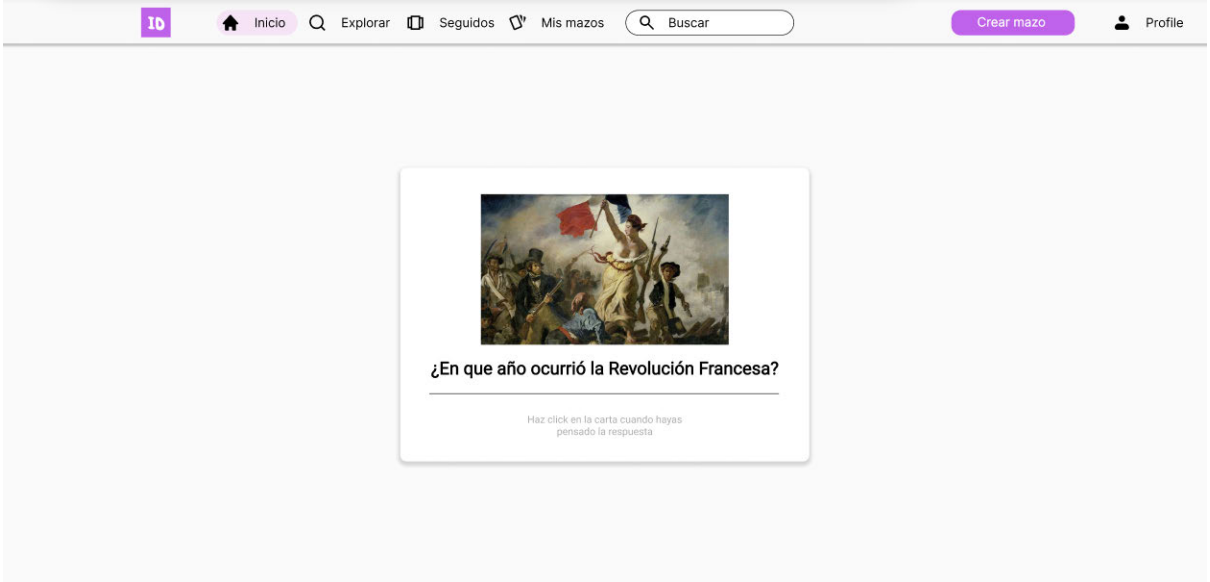


Figura 3.15: *Prototipo de la interfaz de usuario para la pantalla de estudio de mazo (mostrando pregunta y foto de una carta)*

La interfaz recientemente mostrada es una de las diferentes variantes que se han diseñado para el entrenamiento con un mazo, siendo este el prototipo para mostrar la pregunta de una carta y la imagen asociada. Otros prototipos existen para las diferentes alternativas contempladas, y todos ellos en su conjunto posibilitan la realización del caso de uso “entrenar con mazo”, para el cual han sido diseñados.

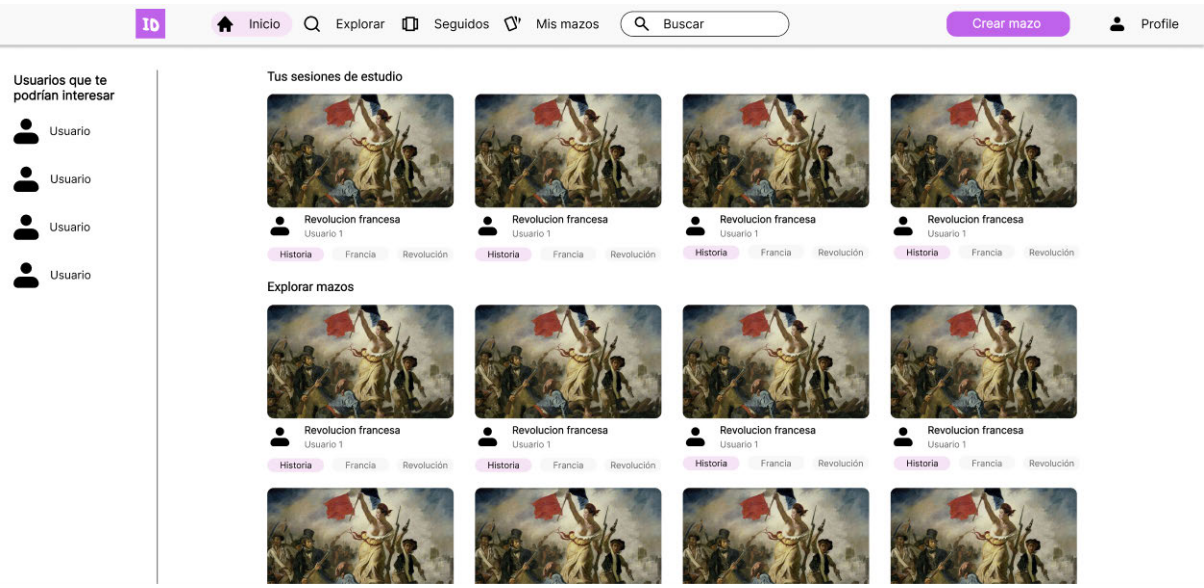


Figura 3.16: *Prototipo de la interfaz de usuario para la pantalla de timeline*

Por último, también se ha definido la pantalla que muestra el timeline de un usuario, con los mazos pendientes de estudiar ese día, así como usuarios y mazos recomendados. La importancia de esta interfaz radica en definir cómo se mostrarán las listas de mazos, ya que es parte importante de diferentes interfaces del proyecto.

Capítulo 4. Análisis del sistema

Tras realizar las actividades de obtención de requisitos, es posible comenzar el análisis del sistema. En las siguientes secciones, se presentan los diagramas de las dos actividades de análisis que han sido realizadas.

4.1 Análisis de la arquitectura

Como ya se ha explicado en el marco teórico, el objetivo de esta actividad es extraer todas las clases de la arquitectura en base al patrón de diseño de software seleccionado, que en este caso es MVC.

El criterio para extraer cada uno de los tipos de clases ha sido el siguiente:

- **Modelos:** Se han extraído del modelo del dominio. Solo se han creado clases de aquellas que se considera tienen suficiente funcionalidad como para ser una clase de modelo. Aquellas que no se han incluido han pasado a formar parte de otras clases.
- **Vistas:** Basadas en las interfaces que se han diseñado en el apartado 3.4. Todas las demás pantallas se han determinado basándose en las ya definidas.
- **Controladores:** Cada controlador hace referencia a un caso de uso del sistema. En esta aproximación inicial, se ha mantenido un controlador por cada caso de uso para ayudar a delimitar correctamente las interacciones y las funcionalidades de cada clase en actividades posteriores del análisis.

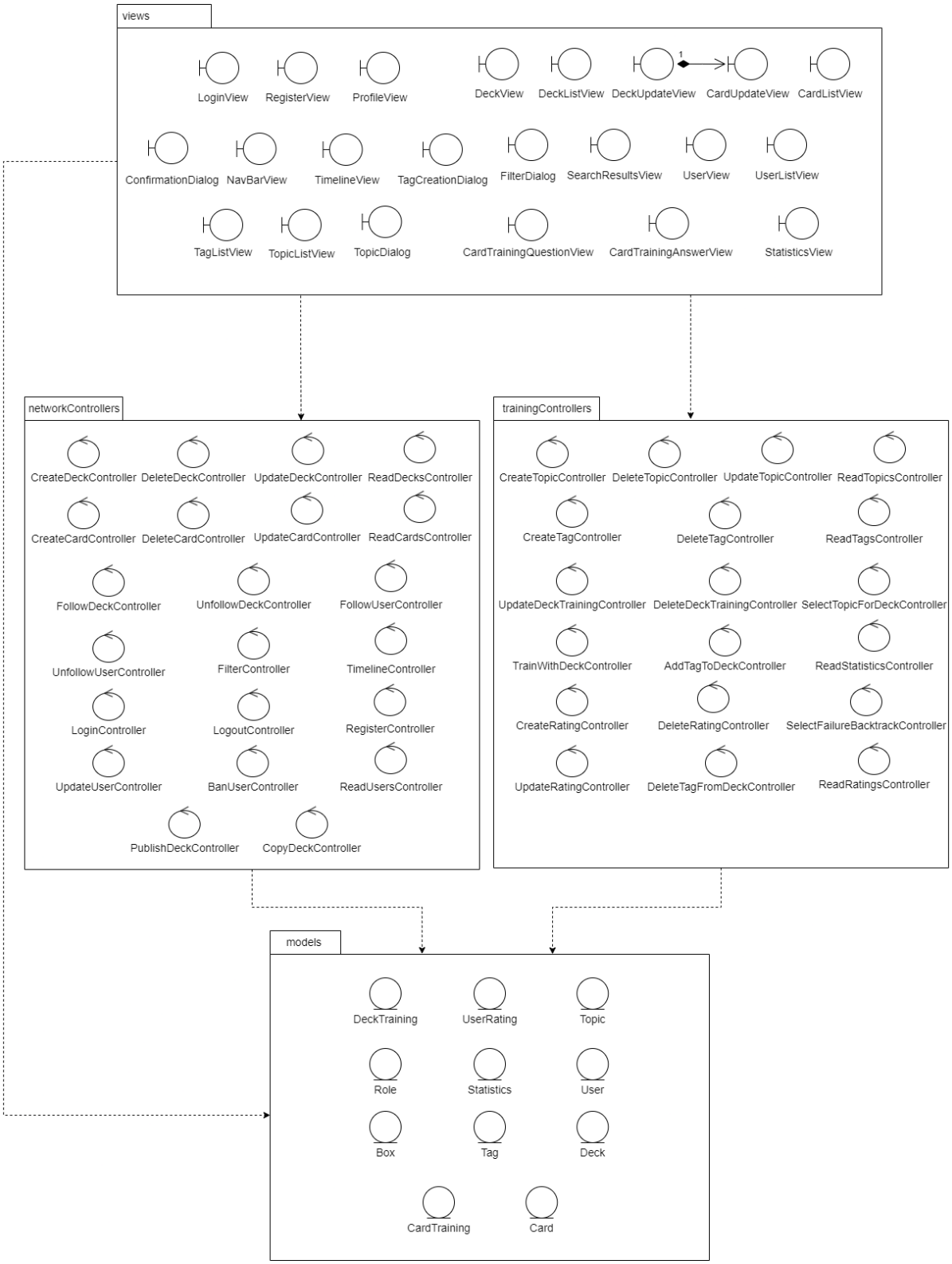


Figura 4.1: *Clases extraídas del análisis de la arquitectura y clasificadas por paquetes MVC*

4.2 Análisis de los casos de uso

Tras haber definido las clases que se prevé formarán la arquitectura, es posible comenzar a realizar un análisis de los casos de uso priorizados en actividades de requisitos. Para ello, se han utilizado diagramas de colaboración. A continuación, se presentarán algunos de estos diagramas, junto con una descripción breve de lo que representan.

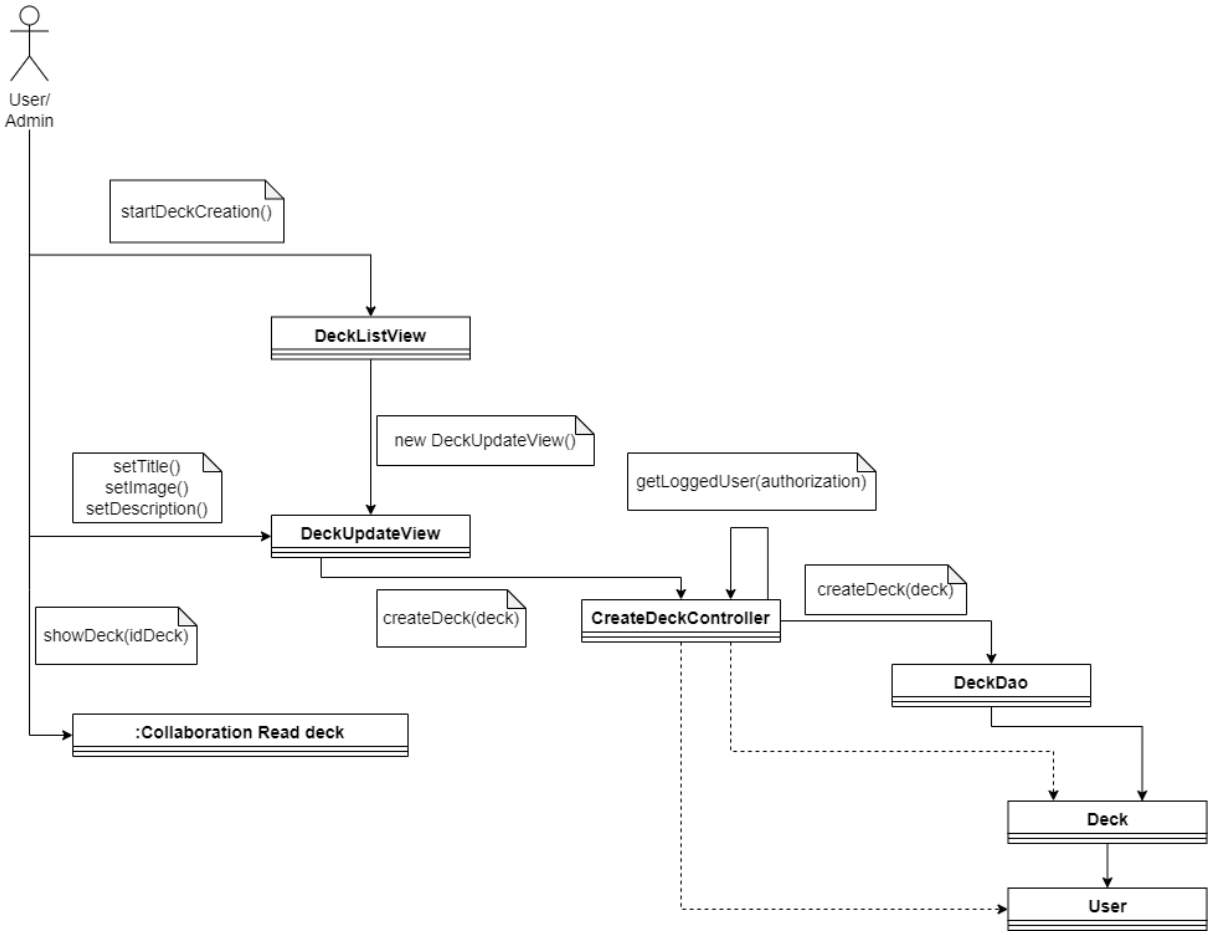


Figura 4.2: *Diagrama de colaboración para el caso de uso crear mazo*

Como se puede ver en el diagrama, el actor se comunicará con las vistas utilizando las funciones especificadas: en este caso, desde la lista de mazos podrá solicitar la creación de uno nuevo, lo que lo llevará a otra pantalla desde la que introducir la información. Dicha pantalla es la utilizada para actualizar mazos; la razón de esto es que la pantalla servirá tanto para el caso de uso de creación como de modificación. Una vez toda la información ha sido otorgada y el usuario decida crear el mazo, se llamará al controlador correspondiente, que tras obtener la información del usuario para complementar la información obtenida de la vista, la enviará a la BD para que sea almacenado mediante el DAO pertinente.

El resto de los diagramas de colaboración creados atienden a la necesidad de describir otros casos de uso, pero todos siguen la misma estructura: el usuario se comunica con una o más vistas, donde una de ellas envía información al controlador del caso de uso que se esté realizando. Este controlador podrá realizar una o más funciones en base a las necesidades del susodicho caso de uso, tras lo cual enviará la información al DAO del modelo con el que se está trabajando para reflejar los cambios en la BD. Por último, una vez realizada la acción

solicitada, la vista en la que se encuentra el actor colaborará con otro caso de uso, lo cual está representado mediante el prefijo *:Collaboration*. Es posible apreciar que en los diagramas seguidamente presentados, esta forma de operar se mantiene: esto es debido a que se está aplicando el patrón MVC.

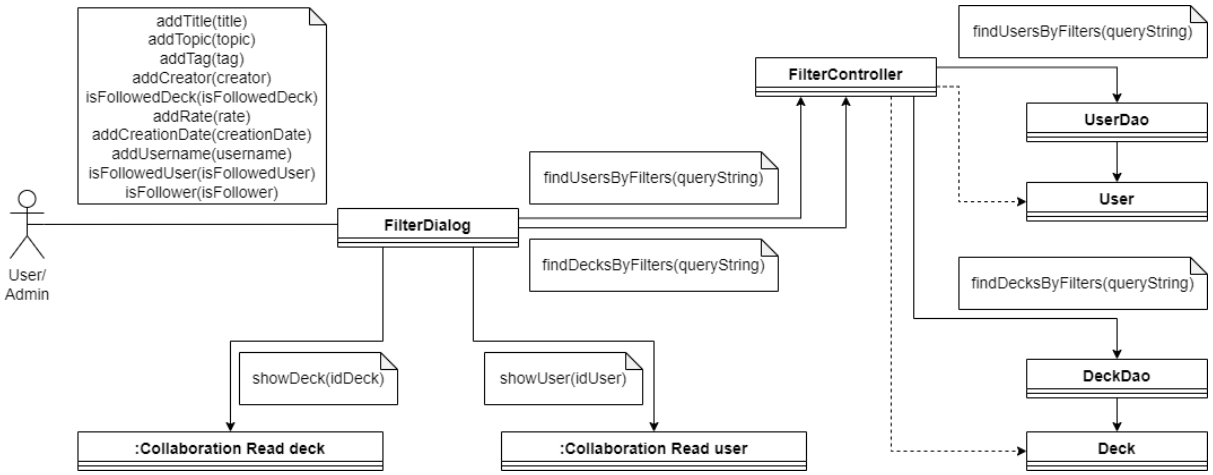


Figura 4.3: *Diagrama de colaboración para el caso de uso filtrar usuarios y mazos*

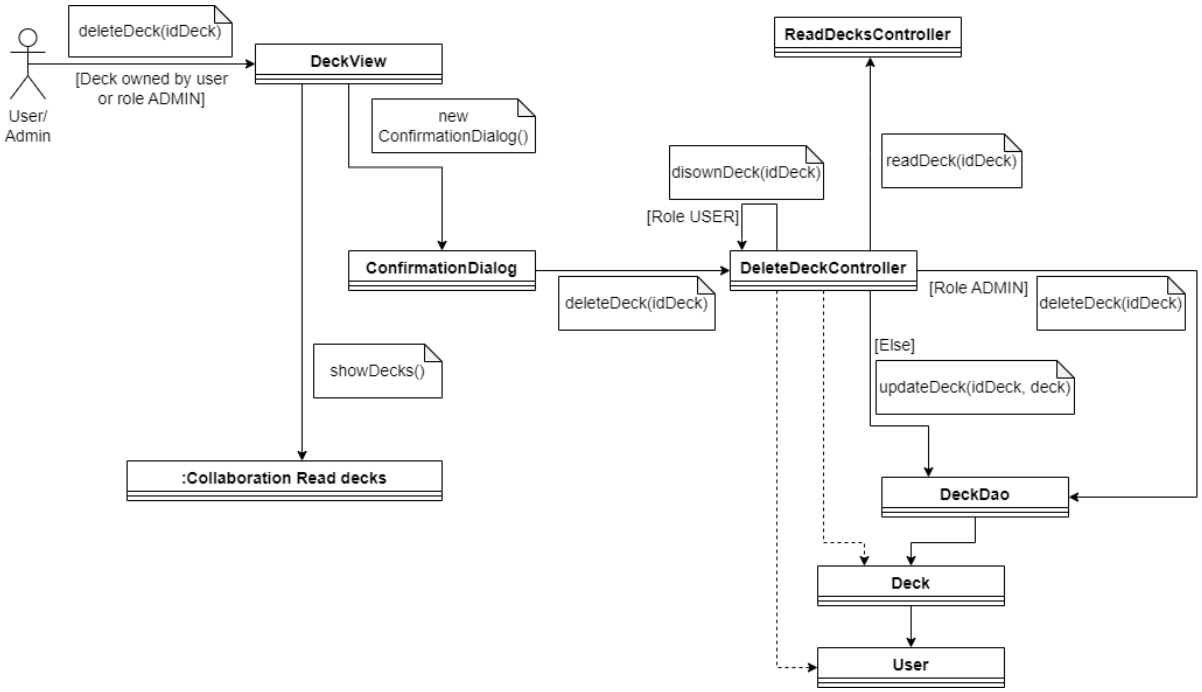


Figura 4.4: *Diagrama de colaboración para el caso de uso borrar mazo*

En este diagrama en concreto, se puede apreciar que además, dependiendo del rol que posea el actor, se realizarán las llamadas a unas funciones u otras, ya que la funcionalidad de borrado posee dos lógicas.

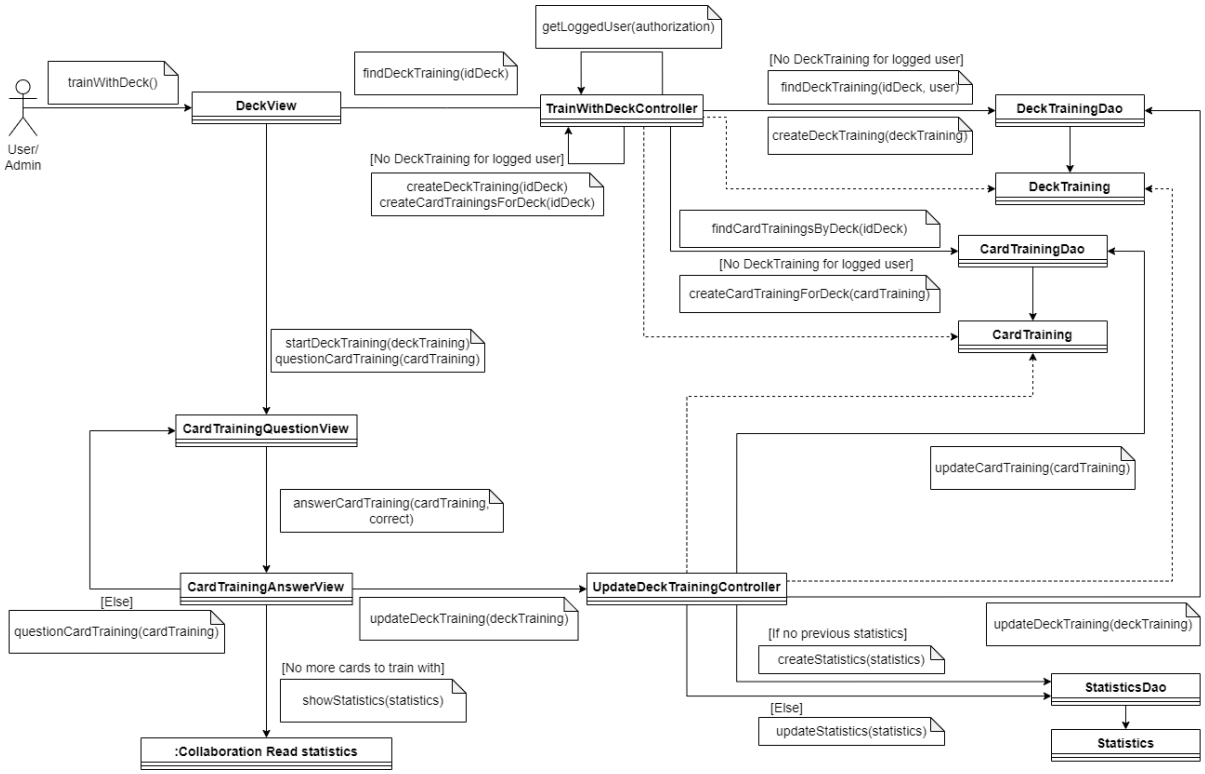


Figura 4.5: *Diagrama de colaboración para el caso de uso entrenar con mazo*

Por último, es posible comprobar que el diagrama del caso de uso de un entrenamiento es más complejo, ya que requiere que se realicen llamadas a dos controladores distintos, siendo dependiente también de los datos previos para realizar la secuencia del caso de uso correspondiente.

Capítulo 5. Implementación

Tras haber completado las actividades de la disciplina de análisis, se comenzó a trabajar en la implementación del sistema. Sin embargo, a diferencia de las anteriores disciplinas, donde todo el trabajo se realizó en una sola iteración, en esta el trabajo ha sido dividido en diferentes iteraciones, de aproximadamente dos semanas. Es por ello que todo el proceso de implementación que se detallará a continuación ha sido dividido teniendo en cuenta qué se ha realizado en cada iteración, demostrando así una aplicación correcta de RUP.

5.1 Iteración 1

En esta primera iteración, se ha decidido comenzar el desarrollo principalmente de los casos de uso que fueron priorizados durante la obtención de requisitos. Sin embargo, debido a que se ha tomado la decisión de que el desarrollo de las interfaces de usuario se realice íntegramente, también se les ha dado prioridad en la iteración a otros casos de uso, los cuales no habían sido priorizados previamente. En concreto, se han decidido implementar los siguientes casos de uso en esta primera iteración:

- Crear mazo
- Crear carta
- Actualizar carta
- Borrar carta
- Obtener temas
- Seleccionar tema del mazo
- Obtener etiquetas
- Añadir etiqueta al mazo
- Borrar etiqueta del mazo
- Login
- Registro de usuario

Además, en esta iteración también se ha trabajado en la creación de los repositorios de GitHub de front-end y back-end, así como en la integración continua de ambos, la cual también será explicada durante esta sección.

La duración de esta iteración se estimó que serían dos semanas, en base al tiempo de dedicación disponible de los miembros del equipo. Sin embargo, la estimación no fue del todo acertada, y el tiempo necesario para la finalización de esta iteración fue de dos semanas y media.

5.1.1 Creación de los repositorios de GitHub e Integración Continua de ambos

Una de las primeras tareas que se realizó al comenzar el desarrollo de esta iteración fue la creación de ambos repositorios de código, tanto de front-end como de back-end, así como su correspondiente validación mediante la integración continua. Durante esta iteración, el despliegue continuo no se realizó, quedándose postergado a una iteración posterior, ya que no era prioritario para permitir comenzar la implementación del sistema.

El primer paso fue definir la estructura de directorios que seguirían ambos proyectos, utilizando como punto de partida el análisis de la arquitectura realizado previamente, representado en la imagen 4.1. En este análisis, se establecieron las necesidades para cada una de las partes del MVC, pero las diferentes clases extraídas no pertenecen tanto front-end como a back-end; en cambio, las vistas son exclusivas de front-end, habiendo que definir aquellas que se utilizarán en back-end para la obtención de información. Los controladores se ubican mayoritariamente en back-end, ya que es en esta capa de la aplicación donde se desea mantener la lógica del sistema. Sin embargo, algunos controladores permanecen en front-end (por ejemplo, entrenar con un mazo), ya que esta lógica está centrada en las interfaces de usuario. Por último, los modelos forman parte de ambas capas.

A continuación, se presenta la estructura de carpetas que se a utilizado para el desarrollo del back-end:

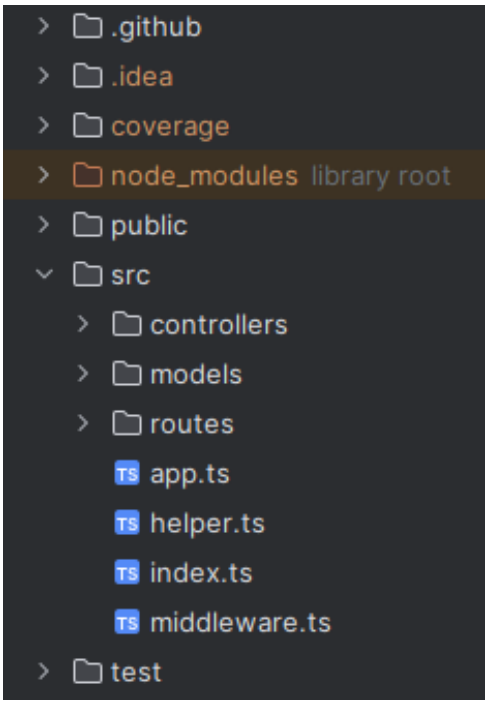


Figura 5.1: *Carpetas para la estructuración de las clases implementadas en back-end*

Además de la estructura de carpetas presentada en la imagen, también existen algunos ficheros de configuración en la raíz del proyecto, como los ficheros *.gitignore* (para la exclusión de ficheros en el repositorio de GitHub) o *package.json* (donde, principalmente, se encuentran las dependencias incluidas mediante npm). En lo que respecta al propio código del back-end, podemos comprobar que en la carpeta *src* se han incluido tres directorios: cada uno de ellos

para cada uno de los tipos de clases de MVC. De entre estos directorios, el único que no respeta el nombre de MVC es el directorio *routes*, que en realidad son las vistas de back-end, ya que proporcionan una forma de que otros servicios obtengan los datos deseados en base a cierta información proporcionada, respetando la definición de vista.

En lo referido al contenido de las tres carpetas dentro de *src*, a continuación se detalla cómo están comprendidos cada uno de los tipos de clases dentro de estas:

- En la carpeta *src/models*, se hallan todos los modelos de los distintos tipos de datos del sistema. Estos modelos son definidos como esquemas de Mongoose, de modo que esta librería sea capaz de crear la base de datos sin la necesidad de que el programador haga nada. Además, realiza validaciones de los datos en base al tipo y otros requerimientos que se establezcan (como validar si un número es superior al mínimo establecido, por ejemplo).
- El directorio *src/routes* se encarga de aglutinar todas las rutas que permiten la realización de peticiones a la API para la obtención de información. Cada una de estas rutas define el flujo que tienen que seguir las llamadas a los controladores (en caso de que sea necesaria más de una llamada).
- Por último, en la carpeta *src/controllers* se encuentran las clases con más lógica de todo el sistema: los controladores. Estas clases se encargan de hacer peticiones NoSQL mediante Mongoose a la BD, y luego, si fuese necesario, tratar los datos para las acciones que hayan solicitado los usuarios.

Otro directorio en el que merece la pena hacer hincapié es el que posee el nombre *public*, ya que está destinado a almacenar todos los recursos del servidor que deban ser públicos para los usuarios. En esta iteración, los únicos recursos de este tipo son las imágenes de las cartas y los mazos. El uso de esta carpeta evitará el envío constante de las imágenes en las peticiones a back-end, ya que mediante una URL será posible acceder a estas sin complicaciones.

Por último, es imperante hacer mención del directorio *.github*, donde se encuentra el fichero que permite realizar la integración continua mediante GitHub Actions. En el fichero que define el flujo de trabajo, el primer paso es definir al subir código a qué rama se realizará la integración continua: en el caso de este proyecto, se ha optado por las ramas *develop* y *master*.

Una vez decididas las ramas que verificar, la siguiente tarea a realizar es establecer los pasos que se tendrán que seguir para asegurar la completa validez del código. A continuación, se detallan todos estos pasos:

- Instalar todas las dependencias del proyecto.
- Crear el fichero con las variables de entorno que requiere el sistema para su correcto funcionamiento. Para evitar que estos datos los puedan ver atacantes o usuarios maliciosos, se ha utilizado el sistema de secretos de GitHub, que permite el guardado de cierta información sensible para tareas similares.
- Crear los ficheros que contienen las claves publicas y privadas para la firma y verificación de los Json Web Token (JWT), recurriendo para ello también a los secretos de GitHub. Los token JWT sirven para autenticar usuarios y constan de tres partes, las cuales transmiten diferente información acerca del usuario autenticado o el propio token en sí:

  - La cabecera del JWT incluye los metadatos acerca del tipo de token y el algoritmo de encriptación, entre otros.
  - El cuerpo es donde se envía la información deseada acerca del usuario: nombre, rol…
  - Por último, la firma permite validar que el token recibido es de fiar.
- Ejecutar los test.
- En caso de que no haya habido fallo, SonarCloud deberá realizar un análisis estático del código para evaluar su calidad.

A continuación, es posible apreciar un ejemplo de todos estos pasos funcionando correctamente tras una subida de código:

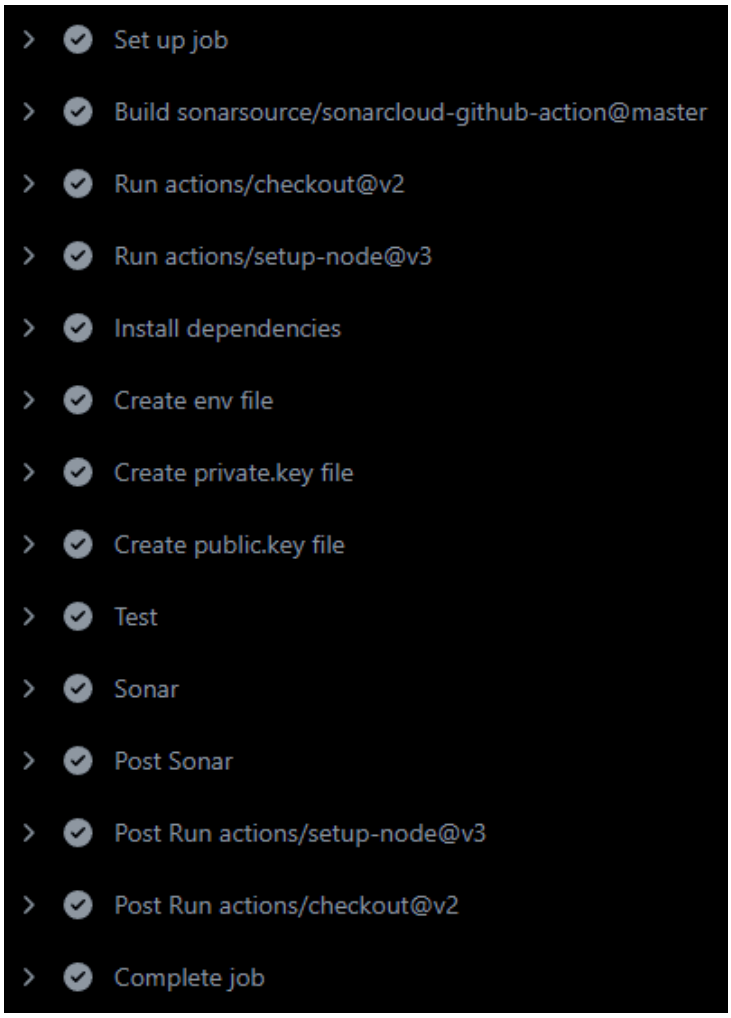


Figura 5.2: *Pasos realizados para la integración continua de back-end mediante GitHub Actions*

Además, para poder verificar con un simple vistazo que tanto los test como el análisis estático de SonarCloud funcionan, se han añadido dos insignias al fichero README.md:

Figura 5.3: *Insignias de GitHub Actions y SonarCloud del proyecto*

En lo que respecta a la capa de front-end, la estructura de directorios que se ha utilizado ha sido la siguiente:

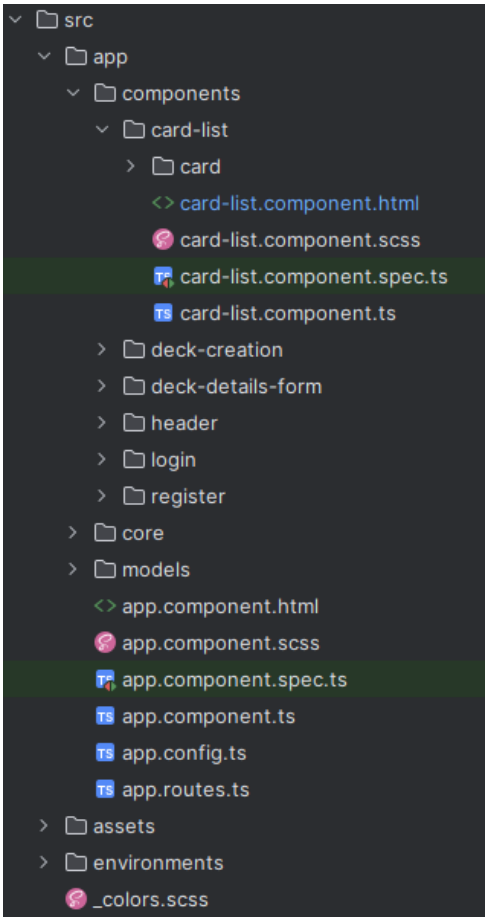


Figura 5.4: *Carpetas para la estructuración de las clases implementadas en front-end*

En esta estructura, podemos encontrar que el directorio más importante,y en el que la gran mayoría del esfuerzo de desarrollo en front-end se centrará es *src/app/components*. Esto se debe a que en esta carpeta se almacenarán la gran mayoría de los componentes de la aplicación.

Un componente es un bloque de contenido que facilita la reutilización de código, y que son las piezas fundamentales de la creación de las interfaces de usuario de una aplicación Angular. Estos componentes son clases TS con propiedades que pueden ser utilizadas en un fichero HTML gracias a un decorador: funciones que modifican clases, métodos o propiedades de forma declarativa.

Como se puede apreciar en la imagen 5.4, dentro del directorio *src/app/components* se almacenan todos los componentes, cada uno con su propio subdirectorio, y contienen todos los ficheros requeridos por cada uno: el fichero TS, la plantilla HTML, una clase SCSS con los

estilos propios de ese componente que no se vayan a reutilizar en el resto de la aplicación, y un fichero de test para el componente. Además, el resto de carpetas almacenan clases para otro tipo de labores importantes:

- El directorio *src/app/core* contiene todos los servicios requeridos para la comunicación con back-end.
- En *src/app/models* se almacenan todos los modelos del front-end.
- La carpeta *src/assets* aglomera los recursos del sistema, como la imagen del logo de IntelliDeck, por ejemplo.
- Por último, *src/environments* guarda las variables para los diferentes entornos del proyecto, siendo en este caso para desarrollo y producción.

Es significativo subrayar que los servicios que se hallan dentro de la carpeta *src/app/core* se encargan mayoritariamente de hacer peticiones a back-end para la obtención de los datos, comunicando así ambos repositorios, aunque también existen servicios que se encargan de facilitar el traspaso de información entre componentes. Sin embargo, como ya se ha explicado en el apartado 2.2, la comunicación con back-end en Angular no se va a realizar de forma síncrona, sino que se hará por medio de la programación reactiva. Este paradigma de programación se orienta en reaccionar a los eventos para alterar los datos datos de la aplicación (bien puede ser solo parte de la información que se esté desplegando en una interfaz como cambiar de pantalla completamente). Por tanto, funciona de forma asíncrona.

Los servicios utilizan la programación reactiva para hacer peticiones a la API de back-end, generando un *Observable* que se utilizará para tratar los datos una vez lleguen. Sin embargo, por motivos de reutilización, no son los servicios los que esperarán la respuesta de la API: tras hacer la petición, los servicios devolverán el *Observable* inmediatamente a los componentes que requiriesen los datos, y serán estos los que esperarán a que la respuesta llegue al *Observable*, para así reaccionar a esta como se considere necesario.

Volviendo a los directorios, existen también otros que no aparecen en la imagen 5.4 que son igualmente importantes, como la carpeta *.github*, conteniendo, tal y como ocurría en back-end, el proceso de integración continua de GitHub Actions. Aunque los pasos a realizar no son exactamente los mismos, en esencia es el mismo proceso. A continuación se muestra la integración continua del repositorio front-end tras la subida de código:

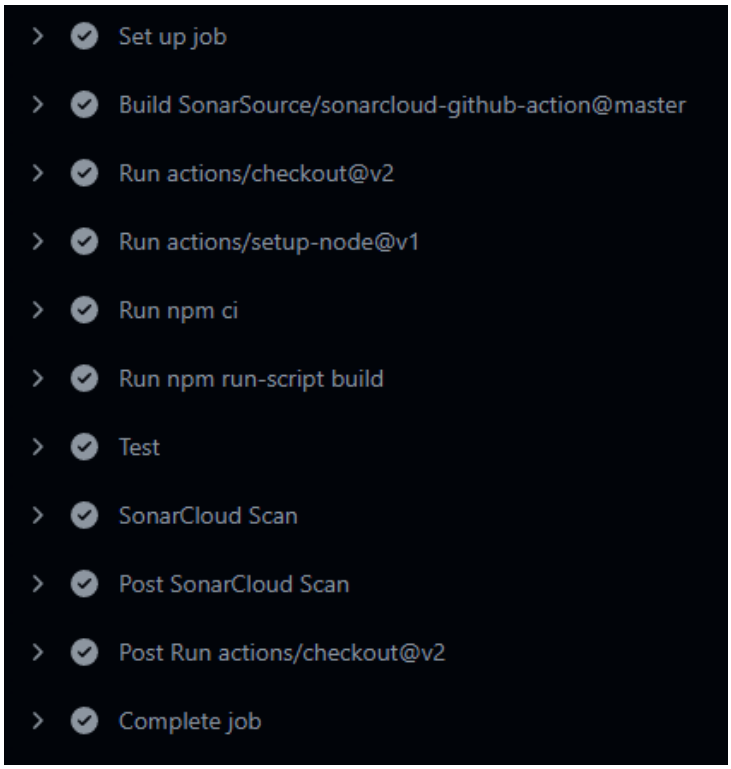


Figura 5.5: *Pasos realizados para la integración continua de front-end mediante GitHub Actions*

5.1.2 Interfaces de usuario implementadas

En esta iteración, se han creado tres pantallas diferentes y un diálogo con el que interactuarán los usuarios. Estas interfaces han sido creadas no como parte de algún módulo, tal y como se hacía en versiones previas de Angular, sino como *standalone*; es decir, como un componente independiente, proporcionando diferentes ventajas.

El uso de *standalone* a la hora de crear los componentes es preferido debido a que proporciona mayor flexibilidad que la que otorgaba el uso de módulos, mucho más rígidos. Además, tiene en cuenta el principio de única responsabilidad esencial a la hora de diseñar e implementar código: un componente (también es aplicable a clases y funciones) solo debe estar destinado a cumplir un rol dentro de la aplicación. El uso de los módulos entorpecía la correcta aplicación de dicho principio, ya que los componentes poseían dependencias a código que quizá no necesitasen. Además, los componentes *standalone* facilitan la organización óptima del código y evitar el código muerto o no utilizado.

El primer componente creado es el destinado a permitir el acceso de los usuarios a la aplicación: el componente de login. En la siguiente imagen se muestra el aspecto de dicha interfaz:

Figura 5.6: *Interfaz de usuario: Login*

Además, se ha realizado tratamiento de errores en este componente: por ejemplo, si un usuario tratase de acceder sin haber rellenado alguno de los dos campos, la pantalla mostraría el siguiente error:

Figura 5.7: *Interfaz de usuario: Login con error*

Por otro lado, también se ha creado el componente del registro de usuarios, que permite que aquel que desee crearse una cuenta para utilizar la aplicación aporte todos los datos requeridos:

Figura 5.8: *Interfaz de usuario: Registrar usuario*

Y, al igual que la pantalla anterior, en esta también es posible apreciar que se ha realizado validación en los campos introducidos para el tratamiento de errores:

Figura 5.9: *Interfaz de usuario: Registrar usuario con error*

La última pantalla implementada en está iteración es la que permite la creación de mazos. A diferencia de las anteriores dos interfaces, donde son tan solo un único componente, en esta pantalla se han utilizado varios en sincronía para asegurar la reutilización:

- La barra de navegación, a reutilizar en el resto de pantallas de la aplicación.
- El resto de la interfaz, compuesta, a su vez de los siguientes componentes:
    - El componente titulado en la pantalla “crear mazo”, que permite agregar temas, etiquetas, el título y demás campos.
    - El listado de cartas, conformado de igual forma por cada una de las cartas, siendo estas componentes también.

Aunque el diseño es muy acorde a lo que se presentó en el prototipo de la imagen 3.13, la funcionalidad para mostrar u ocultar cartas se ha desplazado a otra pantalla, a implementar a futuro, debido al planteamiento realizado en el modelo del dominio.

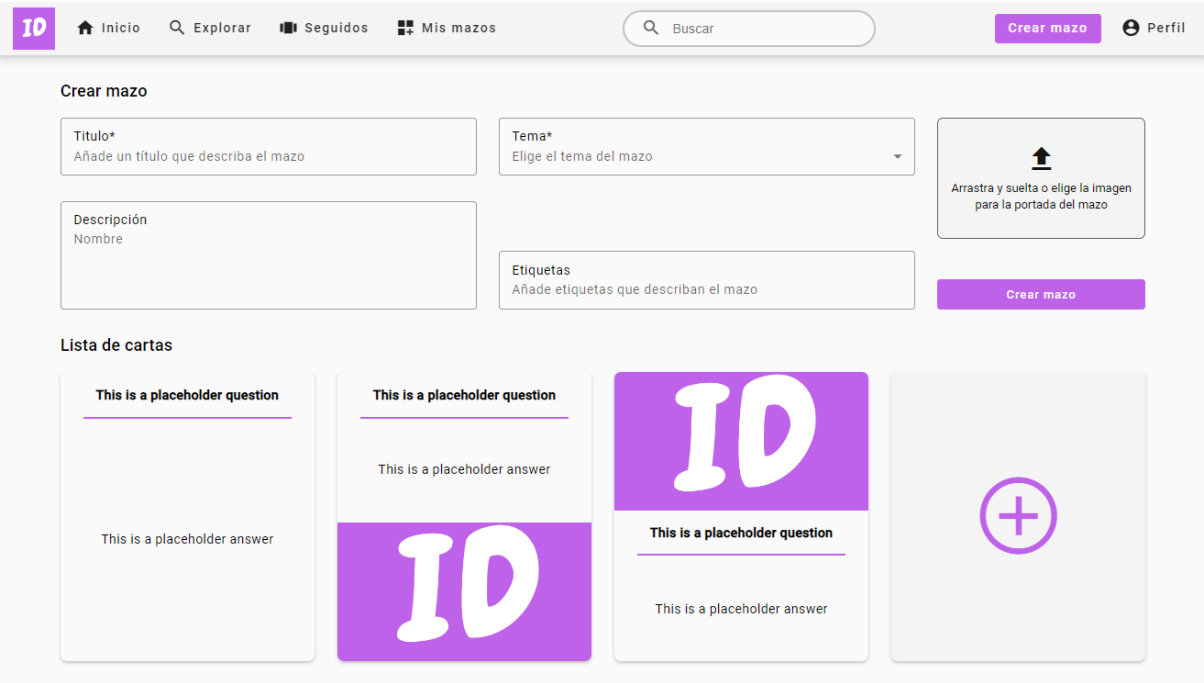


Figura 5.10: *Interfaz de usuario: Crear mazo*

Para mantener la interfaz clara y concisa, la creación de las cartas del mazo se ha externalizado a un diálogo. Asimismo, este diálogo cambia en función de si el usuario desea introducir una imagen en la carta o no:

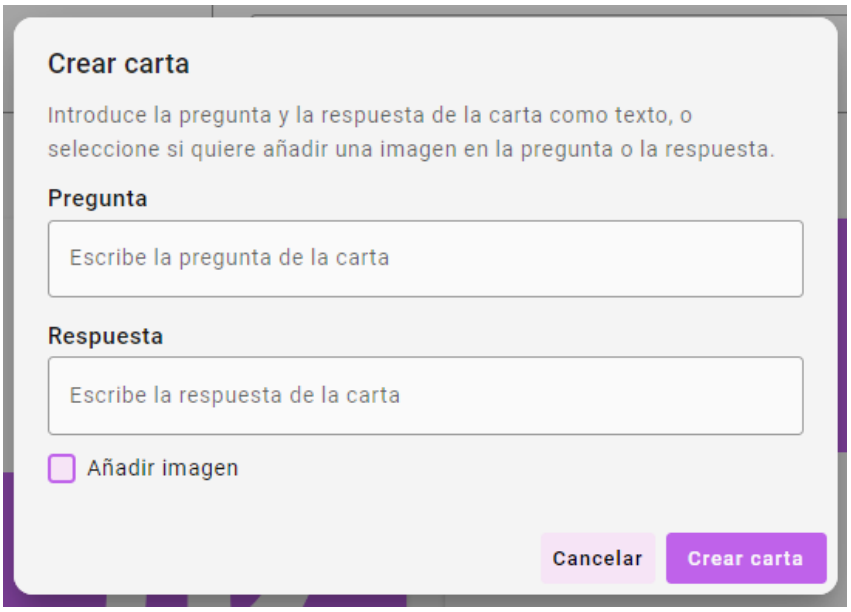


Figura 5.11: *Interfaz de usuario: Diálogo para crear carta*

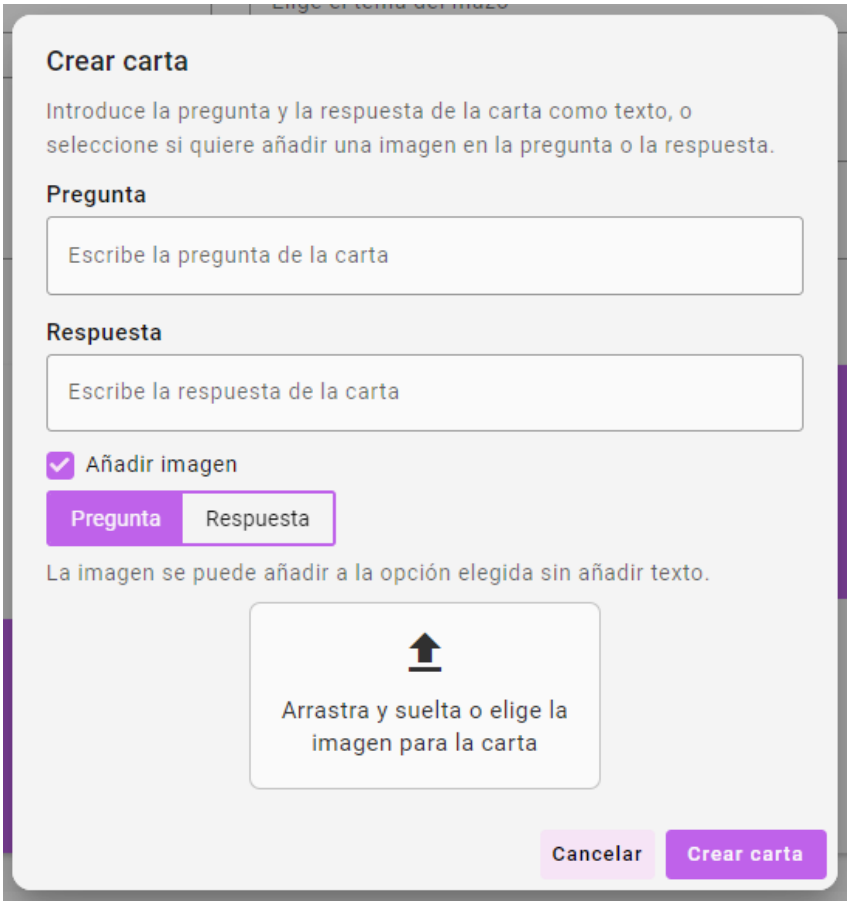


Figura 5.12: *Interfaz de usuario: Diálogo para crear carta extendido para introducir imagen*

Además de la interfaz de usuario per se, también existen otras funcionalidades al pasar el ratón por encima de una carta, o las ayudas para escribir los temas, por ejemplo, que se van a mostrar a continuación:

Además de la interfaz de usuario per se, también existen otras funcionalidades, como al pasar el ratón por encima de una carta, por ejemplo, donde aparecen las opciones para borrar o editar una carta.

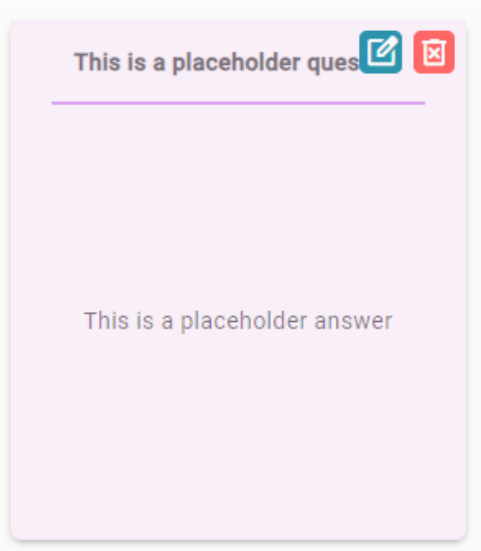


Figura 5.13: *Interfaz de usuario: Crear mazo, al pasar el ratón sobre una carta*

Asimismo, también se han proporcionado ayudas para seleccionar el tema y las etiquetas. En el caso de las etiquetas, las opciones mostradas cambiarán a aquellas que concuerden con la búsqueda que está realizando, aunque de igual forma podrá añadir etiquetas no existentes. Cuando una etiqueta es añadida al mazo, está se muestra con una *chip*, que de igual forma puede ser eliminada.

Figura 5.14: *Interfaz de usuario: Crear mazo, con etiquetas seleccionadas y propuestas de compleción*

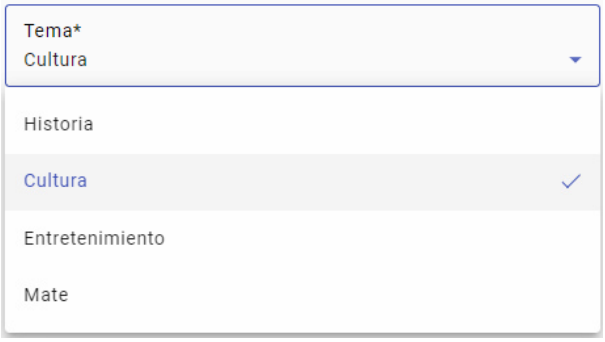


Figura 5.15: *Interfaz de usuario: Crear mazo, con las opciones de temas desplegadas*

Además, igual que con el resto de interfaces, se ha tenido en cuenta el tratamiento de errores, mostrándose a continuación los casos más significativos:

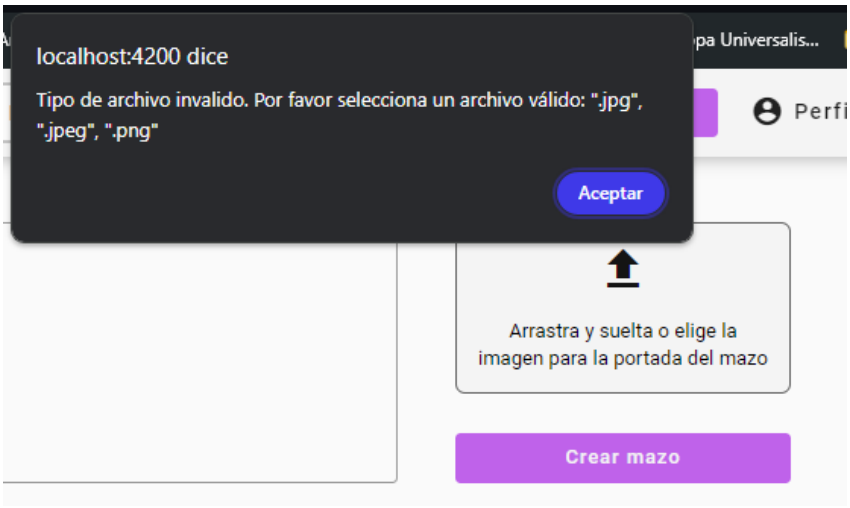


Figura 5.16: *Interfaz de usuario: Crear mazo con error debido a fichero no válido*

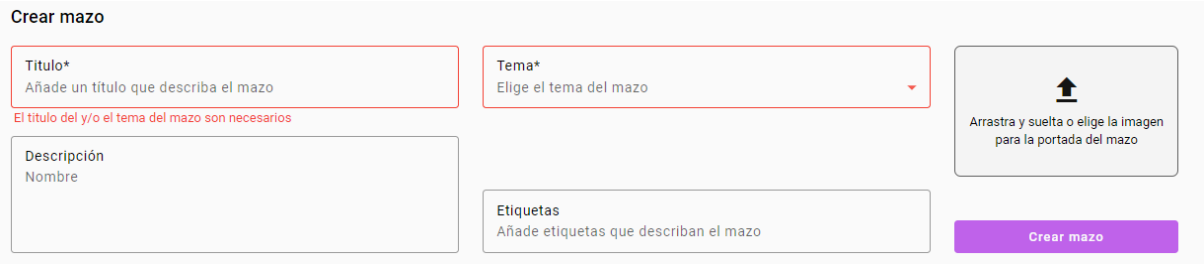


Figura 5.17: *Interfaz de usuario: Crear mazo con error debido a campos requeridos no completados*

Es importante matizar que, debido a la complejidad de esta pantalla, al realizar el envío de información a back-end mediante los endpoints mostrados en la siguiente sección, se realiza una gestión síncrona de los datos: ya que algunas peticiones dependen del resultado de otras, el proceso no puede ser completamente asíncrono: existe una dependencia entre peticiones. Por poner un ejemplo, cada una de las cartas, para crearse, antes debe haber subido su foto (si la tuviese) a back-end, por lo que para crear la carta antes hay que esperar la respuesta de subir la foto.

5.1.3 Endpoints implementados

Para que las interfaces de usuario puedan trabajar en sincronía con la BD, es necesario que se definan ciertos endpoints en back-end, los cuales permitirán realizar las funciones CRUD (Create, Read, Update, Delete) con cada uno de los modelos de datos del sistema.

Para conseguir una mayor seguridad del sistema, se han introducido ciertas medidas que pretenden mitigar ciertos ataques o vulnerabilidades del mismo. Lo primero que se ha realizado es un saneamiento de los datos introducidos por el usuario, con el objetivo de evitar ataques NoSQL Injection, capaces de introducir o extraer información de la Base de Datos. Además, se han encriptado las contraseñas para su guardado en BD mediante lo que se denominan sales criptográficas.

Las sales criptográficas son datos aleatorios que se agregan a las contraseñas a *hashear*, refiriéndose este último termino al proceso de convertir una cadena de caracteres en otra

completamente distinta de forma unidireccional. El problema de este tratamiento es que dos contraseñas idénticas producirán *hashes* idénticos, por lo que los atacantes pueden recurrir a las conocidas como tablas arcoíris o *rainbow tables* para ver si es posible conocer el valor previo al *hasheo*. Sin embargo, al añadir esos datos aleatorios que son las sales, dos contraseñas distintas producirán dos *hashes* distintos, dificultando descubrir la contraseña real. Estas sales son frecuentemente almacenadas junto con la contraseña, para así poder realizar validaciones de igualdad para permitir los login, por ejemplo [42]. En este proyecto, se ha introducido la librería de bcrypt, que permite realizar precisamente todo lo descrito en este párrafo de forma sencilla.

Además, también se ha agregado una función global gestora de errores, de modo que la gran mayoría de los errores que pudieran surgir al utilizar las rutas del sistema sean tratados de forma centralizada. En caso de que los errores no puedan ser atribuidos a falta de datos, conflictos o demás situaciones esperadas, se ha establecido un error global 500 INTERNAL SERVER ERROR, ya que de lo contrario un error podría causar la desconexión de la capa de back-end.

A continuación, se presentan todos los endpoints creados para esta iteración, junto con una breve explicación de los datos que pueden recibir y las respuestas que pueden devolver (se ha excluido la respuesta 500 INTERNAL SERVER ERROR debido a que es común para todos los endpoints del sistema):

Tabla 5.1: *Detalles del endpoint: Crear mazo*

Tipo	**Ruta**	**Descripción**
POST	/decks	Posibilita la creación de un mazo
Parámetros	**Tipo**	**Descripción**
title	String	Título del mazo
topic	ObjectID	Identificador del tema a añadir
tags	ObjectID[]	Identificadores de las etiquetas a agregar
description	String	Descripción proporcionada por el usuario con información acerca del contenido del mazo
image	String	Ruta de la imagen del mazo
Código de respuesta	**Nombre**	**Descripción**
201	CREATED	Se envía al crearse el mazo correctamente
400	BAD REQUEST	Se envía en caso de que los campos requeridos no se envíen
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si el tema no se ha podido encontrar
404	NOT FOUND	Si alguna etiqueta no se ha encontrado

Tabla 5.2: *Detalles del endpoint: Crear carta*

Tipo	**Ruta**	**Descripción**
POST	/decks/:id/cards	Posibilita la creación de una carta dentro de un mazo
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo al que añadir la carta
question	String	Pregunta de la carta
answer	String	Respuesta a la pregunta propuesta
image	String	Ruta de la imagen de la carta
whereImage	String	Lugar de la carta donde se mostrará la imagen, enumerado (QUESTION, ANSWER, NONE)
Código de respuesta	**Nombre**	**Descripción**
201	CREATED	Se envía al crearse la carta correctamente
400	BAD REQUEST	Se envía en caso de que los campos requeridos no se envíen
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
400	BAD REQUEST	Si el usuario no es el propietario del mazo
400	BAD REQUEST	Si el mazo ya está publicado
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.3: *Detalles del endpoint: Registrar usuario*

Tipo	**Ruta**	**Descripción**
POST	/users	Posibilita el registro de un usuario
Parámetros	**Tipo**	**Descripción**
name	String	Nombre real de usuario
surname	String	Apellido/s del usuario
username	String	Nombre de usuario
email	String	Dirección de correo electrónico
password	String	Contraseña del usuario
profilePicture	String	Ruta de la imagen de perfil
role	String	Rol del usuario dentro de la aplicación, enumerado (USER, ADMIN)
Código de respuesta	**Nombre**	**Descripción**
201	CREATED	Se envía al crearse el usuario correctamente
400	BAD REQUEST	Se envía en caso de que los campos requeridos no se envíen

Tabla 5.4: *Detalles del endpoint: Login*

Tipo	**Ruta**	**Descripción**
POST	/login	Permite el acceso del usuario a la aplicación
Parámetros	**Tipo**	**Descripción**
username	String	Nombre de usuario
password	String	Contraseña del usuario
Código de respuesta	**Nombre**	**Descripción**
200	OK	Se envía al comprobar que el usuario está registrado, junto con un token JWT que permita el acceso del usuario a la aplicación
400	UNAUTHORIZED	Se envía en caso de que el usuario no tenga permitido el acceso, o si los datos no son correctos

Tabla 5.5: *Detalles del endpoint: Obtener temas*

Tipo	**Ruta**	**Descripción**
GET	/topics	Devuelve todos los temas definidos por los administradores
Código de respuesta	**Nombre**	**Descripción**
200	OK	Al devolver los temas que existan
204	NO CONTENT	Si no existiesen temas
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.6: *Detalles del endpoint: Obtener etiquetas*

Tipo	**Ruta**	**Descripción**
GET	/tags	Devuelve todas las etiquetas definidas en el sistema
Código de respuesta	**Nombre**	**Descripción**
200	OK	Al devolver las etiquetas existentes
204	NO CONTENT	Si no existiesen etiquetas
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.7: *Detalles del endpoint: Obtener etiqueta por nombre*

Tipo	**Ruta**	**Descripción**
GET	/tags/:name	Devuelve la etiqueta que coincida con el nombre especificado. En caso de que no exista, la crea
Parámetros	**Tipo**	**Descripción**
name	String	Nombre de la etiqueta
Código de respuesta	**Nombre**	**Descripción**
200	OK	Al devolver la etiqueta (si está existía ya)
201	CREATED	Al devolver la etiqueta (si está no existía)
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.8: *Detalles del endpoint: Subir foto*

Tipo	**Ruta**	**Descripción**
POST	/images	Posibilita la subida de una foto al servidor
Parámetros	**Tipo**	**Descripción**
file	File	El fichero a subir al servidor
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras subir correctamente la imagen, junto con la URL donde ver la imagen
400	BAD REQUEST	Si no se envía imagen
400	BAD REQUEST	Si la imagen ocupa un espacio superior a 8 MB
401	UNAUTHORIZED	El usuario no ha iniciado sesión

5.2 Iteración 2

Tras terminar la primera iteración, se dio comienzo a la segunda, donde el desarrollo se ha centrado en seguir añadiendo funcionalidad al sistema. Es en esta iteración donde se ha implementado la gran mayoría de los casos de uso de este proyecto. A continuación, se presentan todos aquellos que se han decidido implementar en esta iteración:

- Leer mazos
- Leer mazo
- Actualizar mazo
- Borrar mazo
- Leer cartas
- Copiar mazo
- Publicar mazo
- Seguir mazo
- Dejar de seguir mazo
- Leer usuarios
- Seguir usuario
- Dejar de seguir usuario
- Filtrar usuarios/mazos
- Crear valoración
- Actualizar valoración
- Borrar valoración
- Leer valoraciones
- Ver timeline

La cantidad de casos de uso ha aumentado con respecto a la iteración anterior debido, principalmente, a dos factores: por un lado, el proyecto ya se encuentra iniciado, por lo que no es necesario preparar todo desde cero, sino iterar sobre lo desarrollado anteriormente. Por el otro lado, esta es la única iteración en la que no se ha realizado ninguna tarea adicional, como lo fue la integración continua en la anterior iteración, por lo que se ha podido dedicar más tiempo en la implementación.

En lo que a la elección de casos de uso se refiere, se ha decidido concluir en su mayoría con los casos de uso relacionados a las capacidades de red social del sistema, dejando para la última iteración aquellas funcionalidades relacionadas con los entrenamientos. La razón detrás

de esta decisión es asegurar una coherencia alta en las funcionalidades implementadas, para así agilizar el trabajo.

En lo que respecta a la estimación para esta iteración, se volvió a calcular que el desarrollo llevaría dos semanas, esta vez cumpliendo el plazo y terminando la iteración en dos semanas.

5.2.1 Interfaces de usuario implementadas

En esta iteración, se han creado la mayoría de las interfaces del sistema, para permitir a los usuarios realizar los casos de uso previamente mencionados. A continuación, se presentan todas las pantallas realizadas, junto con una breve explicación:

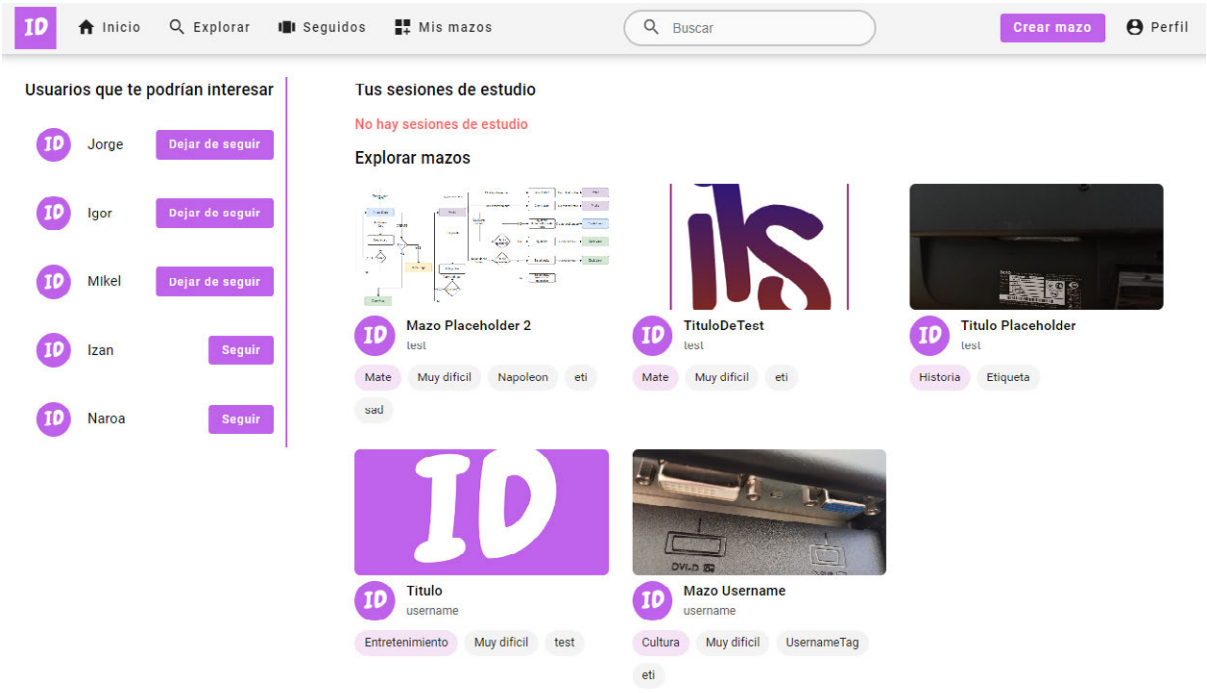


Figura 5.18: *Interfaz de usuario: Timeline sin estudios pendientes*

La primera pantalla realizada es la timeline, la cual será la interfaz principal a la que los usuarios accederán tras haber iniciado sesión. Como se puede apreciar, esta pantalla contiene tres tipos de datos distintos:

- Propuestas de usuarios que podrían interesar a quien esté usando la aplicación.
- Los entrenamiento pendientes del usuario. En la anterior captura, el usuario no tiene ningún estudio pendiente, por lo que no aparece ningún mazo.
- Propuestas de mazos que se considera pueden resultar interesantes al usuario.

En la anterior captura también se puede apreciar que desde la timeline es posible realizar los casos de uso de seguir y dejar de seguir usuarios. Asimismo, a continuación se puede apreciar cómo se presentan los mazos a estudiar si el usuario tiene estudios pendientes de realizar:

Figura 5.19: *Interfaz de usuario: Estudio pendiente en timeline*

Además, también se ha creado un diálogo para el filtrado de usuarios y mazos, el cual está accesible desde cualquier parte de la aplicación, gracias al menú de navegación. A continuación se presenta el propio diálogo, así como la presentación de los mazos y usuarios encontrados.

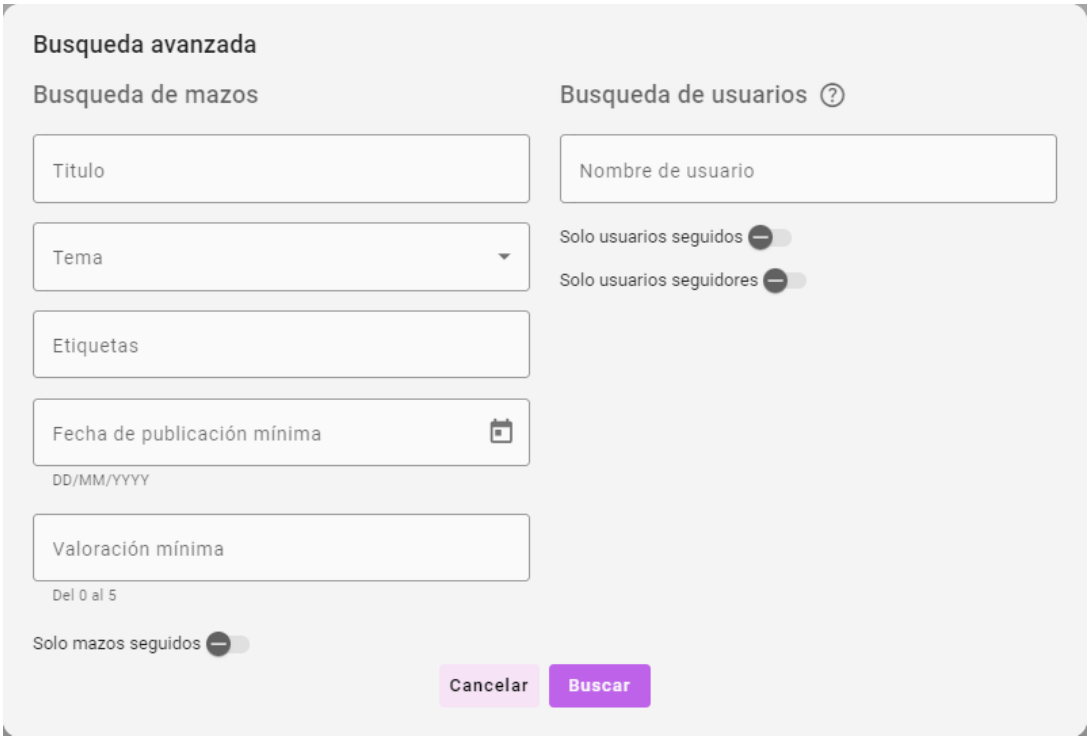


Figura 5.20: *Interfaz de usuario: Diálogo para filtrado de usuarios y mazos*

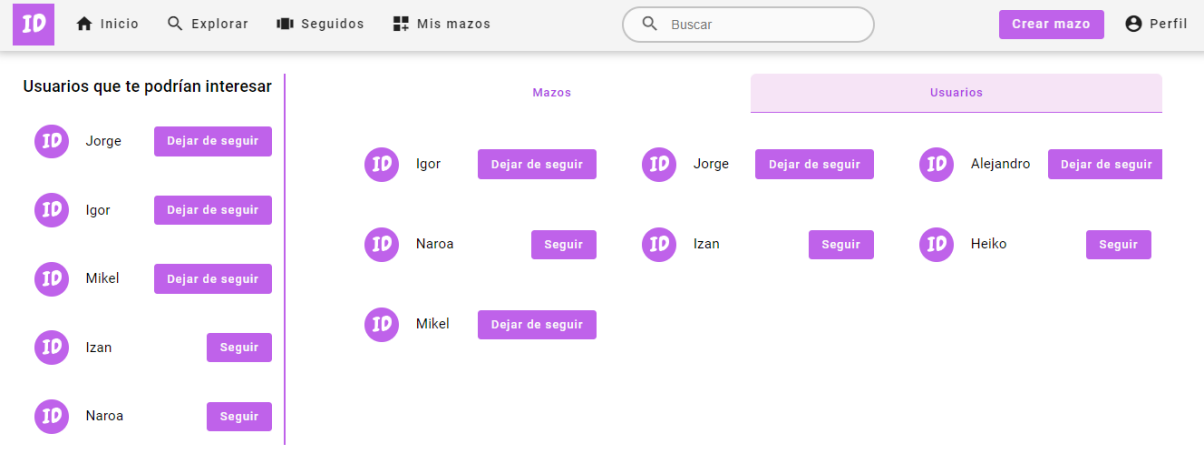


Figura 5.21: *Interfaz de usuario: Resultados de filtrado de usuarios*

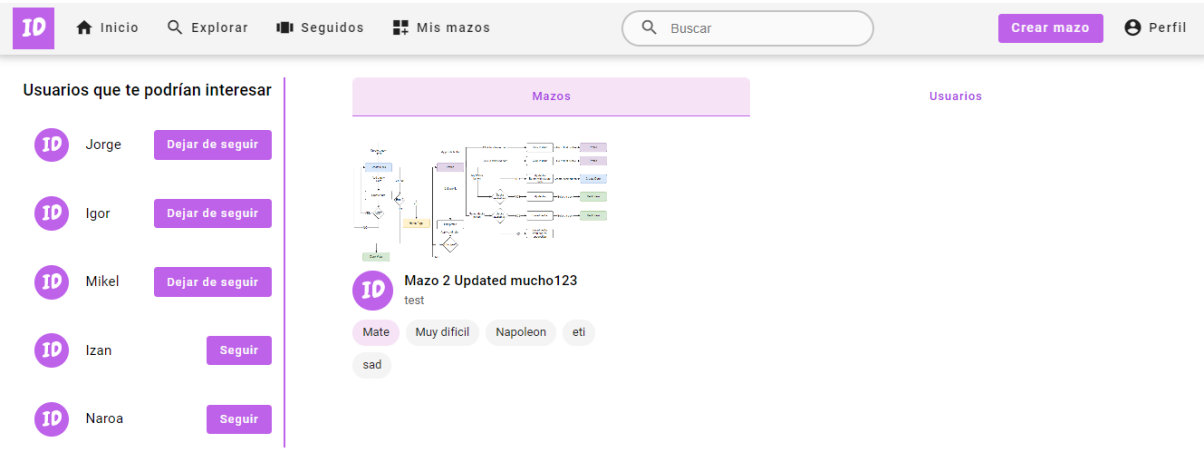


Figura 5.22: *Interfaz de usuario: Resultados de filtrado de mazos*

Como se puede apreciar en la imagen 5.20, en el diálogo de filtrado existe una pequeña ayuda que sirve para que el usuario sepa que establecer el nombre de usuario en la búsqueda condiciona a que solo aparezcan los mazos creados por dicho usuario. Dicha ayuda se muestra al pasar el ratón por encima del icono.

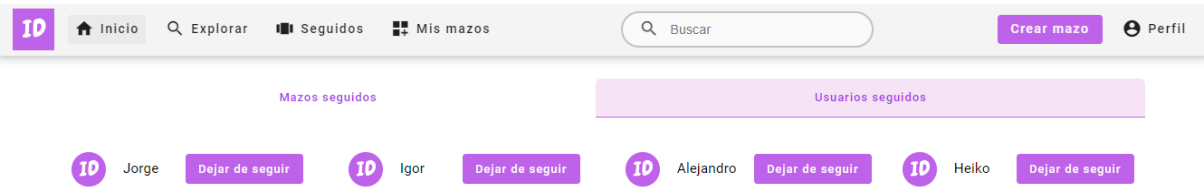


Figura 5.23: *Interfaz de usuario: Listado de usuarios seguidos*

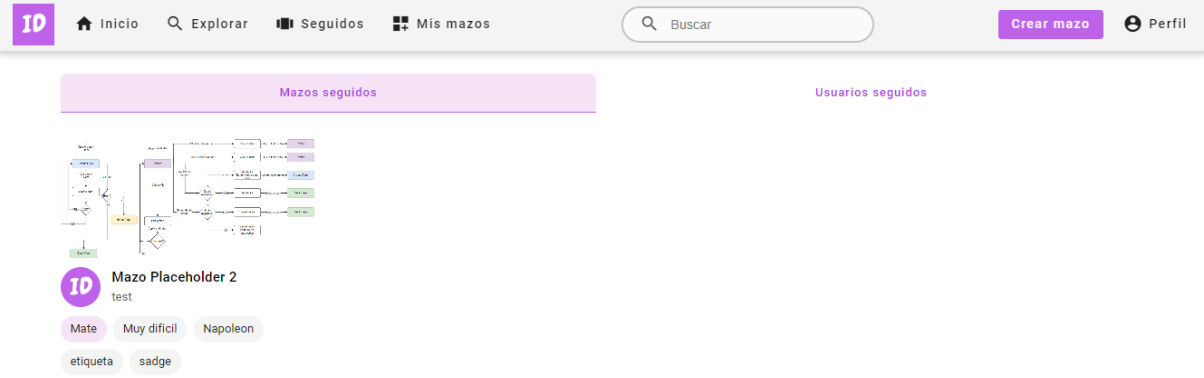


Figura 5.24: *Interfaz de usuario: Listado de mazos seguidos*

Otra funcionalidad que ha sido igualmente agregada es la de listar los mazos y los usuarios seguidos. Esta funcionalidad, al igual que el filtrado, es accesible mediante la barra de navegación de la aplicación y, como es posible apreciar, utiliza el mismo componente para mostrar los datos, demostrando que se ha conseguido realizar una reutilización satisfactoria de los componentes para diferentes casos de uso.

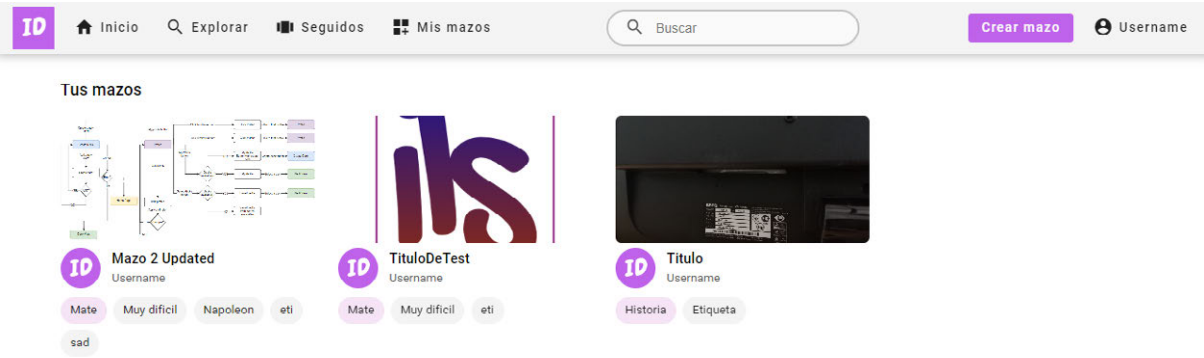


Figura 5.25: *Interfaz de usuario: Listado de mazos propios*

Basado en el mismo componente, y accesible también desde la barra de navegación, se ha permitido a los usuarios listar los mazos propios, tanto los publicados como los que no.

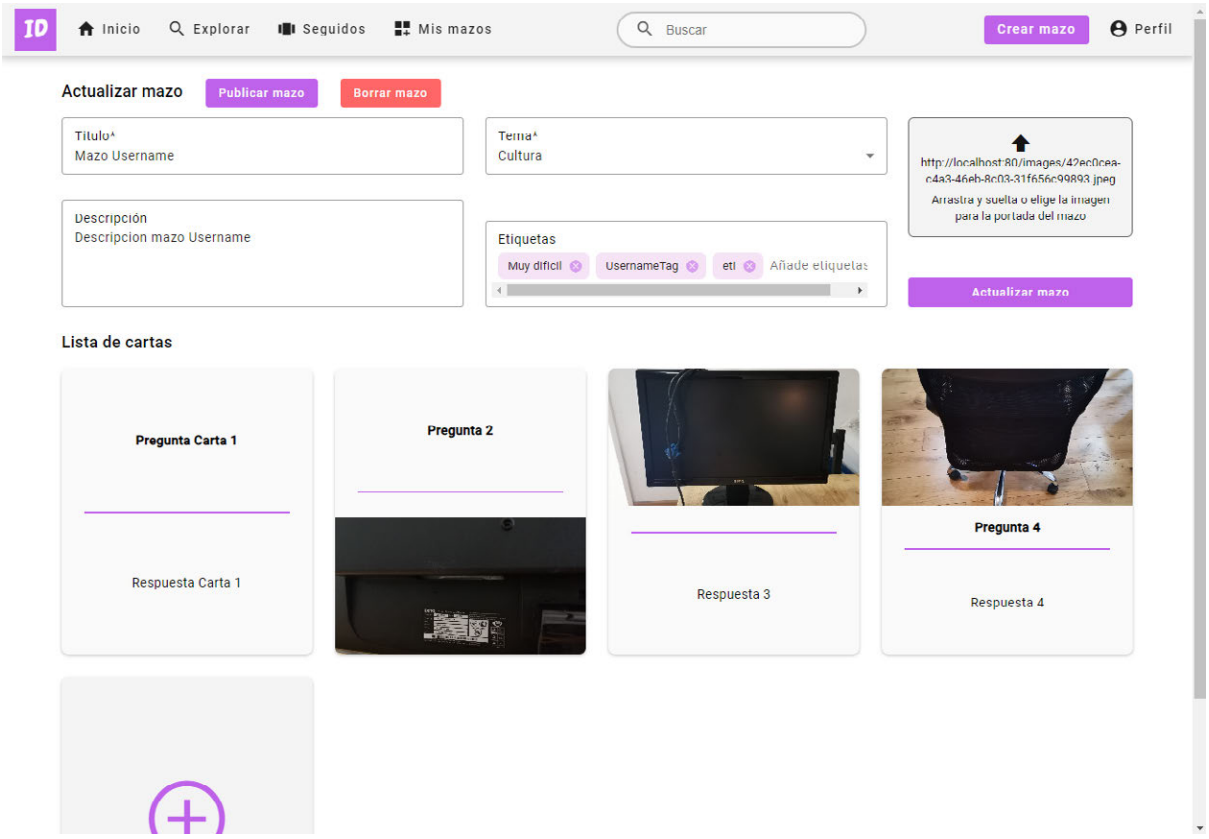


Figura 5.26: *Interfaz de usuario: Mazo propio sin publicar*

Al clicar en un mazo en cualquiera de las anteriores pantallas, se abrirá el mazo seleccionado. Si el mazo es propio y, además, está pendiente de publicar, la interfaz mostrada será muy similar a la utilizada para crear mazos (véase imagen 5.10), solo que existirán algunas opciones más, como la publicación de un mazo o el borrado de mazo. En este punto, borrar un mazo ocasiona un borrado completo del mazo. Además, los datos podrán ser modificados y se podrán gestionar las cartas, tal y como ocurría al crear el mazo.

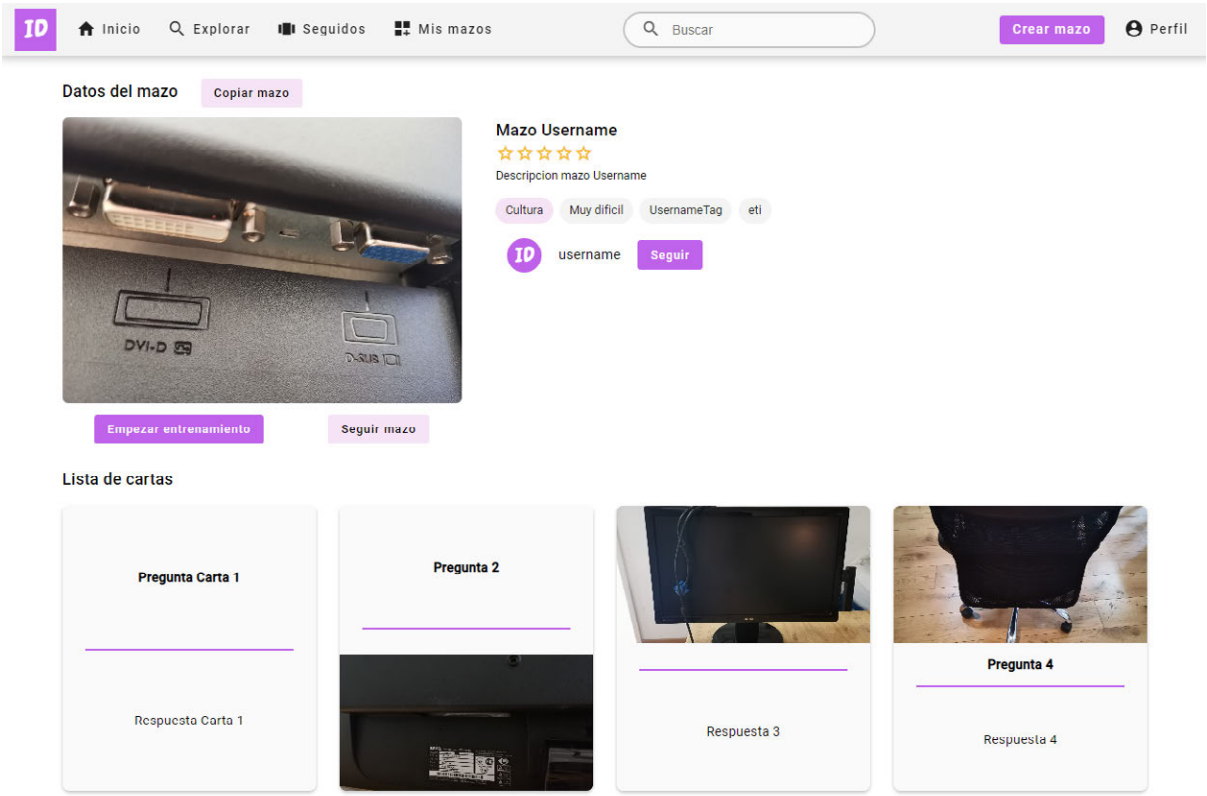


Figura 5.27: *Interfaz de usuario: Mazo externo*

Por otro lado, si el mazo es un mazo de otro usuario, este tendrá otra presentación, principalmente aportando información y utilidades para permitir los entrenamientos del mazo. En esta pantalla se muestran, entre otros datos, las cartas del mazo, y se permite realizar valoraciones, seguir o dejar de seguir un mazo, o entrenar con el mazo (sin embargo, en esta iteración todavía no se proporciona la funcionalidad).

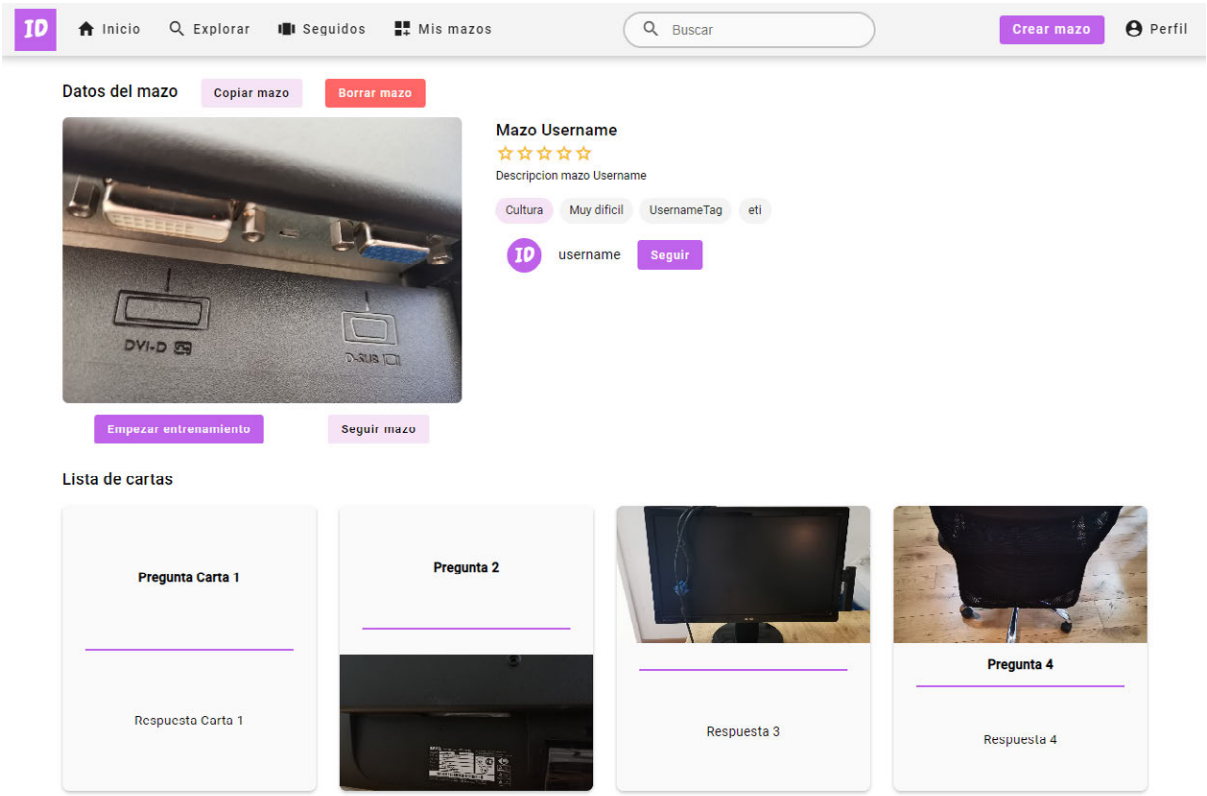


Figura 5.28: *Interfaz de usuario: Mazo propio ya publicado*

Además, si el mazo es propio y está ya publicado, se permitirá realizar su borrado. Sin embargo, esto solo ocasionará, como ya se explicó al especificar el caso de uso, que el mazo se desasocie del usuario, pero el resto de usuarios podrán seguir viéndolo y usándolo para sus estudios.

Por último, también se ha permitido añadir la valoración de los mazos, mediante un diálogo dedicado a ello. Así, los usuarios podrán añadir su propia valoración del contenido del mazo. Si ya tienen una valoración, por otro lado, podrán modificarla o incluso borrarla utilizando el mismo diálogo.

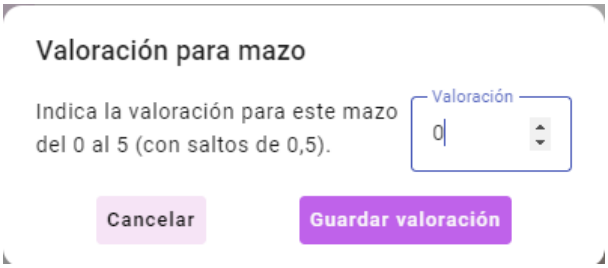


Figura 5.29: *Interfaz de usuario: Diálogo de valoración de mazo sin valoración previa*

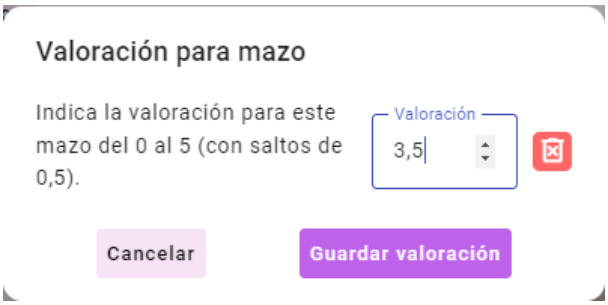


Figura 5.30: *Interfaz de usuario: Diálogo de valoración de mazo existiendo valoración del usuario*

5.2.2 Endpoints implementados

Esta es la iteración donde se encuentran la gran mayoría de los endpoints implementados, así como los más complejos, los cuales serán explicados de forma más extensa. A continuación, se encuentran los detalles de cada uno de estos endpoints:

Tabla 5.9: *Detalles del endpoint: Leer usuarios seguidos por usuario*

Tipo	**Ruta**	**Descripción**
GET	/users/followed/:id	Devuelve los usuarios seguidos por el usuario especificado
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del usuario
Código de respuesta	**Nombre**	**Descripción**
200	OK	Se envían todos los seguidores
204	NO CONTENT	Si el usuario no sigue a otros usuarios
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.10: *Detalles del endpoint: Leer mazo*

Tipo	**Ruta**	**Descripción**
GET	/decks/:id	Leer un mazo en base a su ID
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo a publicar
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras localizar el mazo, junto con el mazo solicitado
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo

Tabla 5.11: *Detalles del endpoint: Actualizar mazo*

Tipo	**Ruta**	**Descripción**
PUT	/decks/:id	Posibilita la actualización de un mazo propio
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo a actualizar
topic	ObjectID	Identificador del tema a añadir
tags	ObjectID[]	Identificadores de las etiquetas a agregar
title	String	Título del mazo
description	String	Descripción proporcionada por el usuario con información acerca del contenido del mazo
image	String	Ruta de la imagen
Código de respuesta	**Nombre**	**Descripción**
200	OK	Se envía al actualizar el mazo satisfactoriamente
400	BAD REQUEST	Si el usuario no es el propietario del mazo
400	BAD REQUEST	Si el mazo ya está publicado
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si el tema no se ha podido encontrar
404	NOT FOUND	Si alguna etiqueta no se ha encontrado

Tabla 5.12: *Detalles del endpoint: Borrar mazo*

Tipo	**Ruta**	**Descripción**
DELETE	/decks/:id	Posibilita la eliminación de un mazo propio
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo a borrar
Código de respuesta	**Nombre**	**Descripción**
200	OK	Se envía si el mazo ya está publicado y el usuario no tiene permisos de administrador. En este caso, el mazo se le desasocia al usuario, pero no es borrado.
204	NO CONTENT	Si el mazo está sin publicar o la solicitud la realiza un usuario con rol ADMIN, sí elimina el mazo completamente, así como sus cartas y entrenamientos en curso.
400	BAD REQUEST	Si el usuario no es el propietario del mazo
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.13: *Detalles del endpoint: Leer cartas*

Tipo	**Ruta**	**Descripción**
GET	/decks/:id/cards	Devuelve todas las cartas de un mazo
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo del que recolectar las cartas
Código de respuesta	**Nombre**	**Descripción**
200	OK	Junto con las cartas solicitadas
204	NO CONTENT	En caso de que no existan cartas en dicho mazo
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.14: *Detalles del endpoint: Actualizar carta*

Tipo	**Ruta**	**Descripción**
PUT	/decks/:id/cards/:cardId	Posibilita la actualización de los datos de una carta
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo en el que actualizar la carta
cardId	ObjectID	Identificador de la carta a actualizar
question	String	Pregunta de la carta
answer	String	Respuesta a la pregunta propuesta
image	String	Ruta de la imagen de la carta
whereImage	String	Lugar de la carta donde se mostrará la imagen, enumerado (QUESTION, ANSWER, NONE)
Código de respuesta	**Nombre**	**Descripción**
200	OK	Al actualizar correctamente los datos
400	BAD REQUEST	Si el usuario no es el propietario del mazo
400	BAD REQUEST	Si el mazo ya está publicado
400	BAD REQUEST	Si los IDs enviados no pueden convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra la carta a actualizar

Tabla 5.15: *Detalles del endpoint: Borrar carta*

Tipo	**Ruta**	**Descripción**
DELETE	/decks/:id/cards/:cardId	Posibilita el borrado de una carta
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo en el que borrar la carta
cardId	ObjectID	Identificador de la carta a borrar
Código de respuesta	**Nombre**	**Descripción**
204	NO CONTENT	Tras borrar correctamente la carta
400	BAD REQUEST	Si el usuario no es el propietario del mazo
400	BAD REQUEST	Si el mazo ya está publicado
400	BAD REQUEST	Si los IDs enviados no pueden convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Es importante matizar que, aunque los casos de uso “actualizar carta” y “borrar carta” ya se habían implementado en la iteración anterior para su uso en la interfaz de usuario de crear mazo (como demuestra la imagen 5.13), en ese momento no era necesario hacer llamadas a back-end para el borrado o actualización: toda esa gestión se realizaba en front-end, y todas las cartas se creaban a la vez al guardar el mazo. Sin embargo, en esta iteración sí que son necesarios los dos endpoints previos, ya que aquí las cartas ya estarán en BD, por lo que al actualizar mazo, puede ser necesario actualizar o borrar ciertas cartas.

Tabla 5.16: *Detalles del endpoint: Copiar mazo*

Tipo	**Ruta**	**Descripción**
POST	/decks/:id/copy	Posibilita realizar una copia de un mazo publicado
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo a copiar
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras copiar el mazo, junto con la copia creada
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	No se encuentra el mazo a copiar

Tabla 5.17: *Detalles del endpoint: Publicar mazo*

Tipo	**Ruta**	**Descripción**
PUT	/decks/:id/publish	Publica un mazo, lo que imposibilita seguir editándolo
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo a publicar
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras publicar el mazo, junto con el mazo actualizado
400	BAD REQUEST	Si el usuario no es el propietario del mazo
400	BAD REQUEST	Si el mazo ya está publicado
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
400	BAD REQUEST	Si el mazo no posee cartas
400	BAD REQUEST	Si el mazo no tiene tema establecido
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.18: *Detalles del endpoint: Seguir mazo*

Tipo	**Ruta**	**Descripción**
PUT	/decks/:id/follow	Posibilita seguir un mazo
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo a seguir
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras comenzar a seguir el mazo
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
400	BAD REQUEST	Si el usuario ya sigue al mazo
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo

Tabla 5.19: *Detalles del endpoint: Dejar de seguir mazo*

Tipo	**Ruta**	**Descripción**
PUT	/decks/:id/unfollow	Posibilita dejar de seguir un mazo
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras dejar de seguir el mazo
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
400	BAD REQUEST	Si el usuario no seguía el mazo
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo

Tabla 5.20: *Detalles del endpoint: Seguir usuario*

Tipo	**Ruta**	**Descripción**
PUT	/users/:id/follow	Posibilita seguir un usuario
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del usuario a seguir
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras comenzar a seguir el usuario
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
400	BAD REQUEST	Si el usuario que ha hecho la petición ya sigue al otro usuario
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.21: *Detalles del endpoint: Dejar de seguir usuario*

Tipo	**Ruta**	**Descripción**
PUT	/users/:id/unfollow	Posibilita dejar de seguir un usuario
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del usuario
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras dejar de seguir el usuario
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
400	BAD REQUEST	Si el usuario que ha hecho la petición no sigue al otro usuario
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.22: *Detalles del endpoint: Leer mazos seguidos por usuario*

Tipo	**Ruta**	**Descripción**
GET	/decks/followed/:id	Devuelve los mazos seguidos de un usuario
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del usuario
Código de respuesta	**Nombre**	**Descripción**
200	OK	Junto con los mazos seguidos por el usuario asociado a la ID
204	NO CONTENT	Si el usuario no sigue ningún mazo
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.23: *Detalles del endpoint: Leer mazos de usuario*

Tipo	**Ruta**	**Descripción**
GET	/decks/user/:id	Devuelve los mazos creados por un usuario
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del usuario
Código de respuesta	**Nombre**	**Descripción**
200	OK	Junto con los mazos del usuario asociado a la ID
204	NO CONTENT	Si el usuario no ha creado ningún mazo
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.24: *Detalles del endpoint: Leer mazos propios*

Tipo	**Ruta**	**Descripción**
GET	/decks/own	Devuelve los mazos del usuario registrado
Código de respuesta	**Nombre**	**Descripción**
200	OK	Junto con los mazos del usuario registrado
204	NO CONTENT	Si el usuario no ha creado ningún mazo
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.25: *Detalles del endpoint: Leer mazos de timeline*

Tipo	**Ruta**	**Descripción**
GET	/decks/timeline	Devuelve los mazos de la timeline
Parámetros	**Tipo**	**Descripción**
page	Integer	Página de los resultados a devolver
Código de respuesta	**Nombre**	**Descripción**
200	OK	Se envían, de forma paginada, los mazos para la timeline
204	NO CONTENT	Si no se encuentra ningún mazo. Puede deberse a que no existan mazos, o que la página definida sea demasiado avanzada y no contenga mazos
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Se considera que es importante hacer hincapié en la paginación realizada en el anterior endpoint, con el objetivo principal de explicar el por qué se ha realizado. Debido a que la timeline debe mostrar una gran cantidad de mazos, para que así el usuario pueda descubrir alguno que le resulte interesante, si la petición de obtener los mazos de la timeline devolviese todos los mazos, se ralentizaría mucho la gestión de los datos desde front-end. Es por ello que el resultado se devuelve paginado: al hacer la petición, se debe solicitar la página deseada, y desde front-end, cuando sea necesario, se solicitará la siguiente página (o la anterior), simulando que todo el contenido está ya cargado cuando en realidad no es así.

Tabla 5.26: *Detalles del endpoint: Leer mazos a estudiar el día actual, para timeline*

Tipo	**Ruta**	**Descripción**
GET	/decks/today	Devuelve los mazos a estudiar ese día
Código de respuesta	**Nombre**	**Descripción**
200	OK	Se envían los mazos a estudiar
204	NO CONTENT	Si no hay ningún mazo que estudiar ese día
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.27: *Detalles del endpoint: Leer usuarios de timeline*

Tipo	**Ruta**	**Descripción**
GET	/users/timeline	Devuelve usuarios de la timeline
Código de respuesta	**Nombre**	**Descripción**
200	OK	Se envían los usuarios
204	NO CONTENT	Si hay ningún usuario a enviar
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.28: *Detalles del endpoint: Filtrar usuarios*

Tipo	**Ruta**	**Descripción**
GET	/users/filter	Devuelve usuarios en base a los filtros aplicados
Parámetros	**Tipo**	**Descripción**
username	String	Cadena de caracteres parcial del nombre de usuario (sensible a capitalización)
follower	Boolean	Si el usuario es un seguidor
followed	Boolean	Si el usuario es seguido
Código de respuesta	**Nombre**	**Descripción**
200	OK	Se envían los usuarios que pasen el filtro
204	NO CONTENT	Si no se encuentra ningún usuario válido
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.29: *Detalles del endpoint: Filtrar mazos*

Tipo	**Ruta**	**Descripción**
GET	/decks/filter	Devuelve mazos en base a los filtros aplicados
Parámetros	**Tipo**	**Descripción**
date	String	Fecha en formato aaaa-mm-dd. Esta fecha establece la fecha de publicación mínima del mazo
title	String	El título del mazo
avgDeckRating	Integer	Valoración mínima del mazo
topic	String	Cadena de caracteres parcial del nombre de del tema (sensible a capitalización)
tag	String	Cadena de caracteres parcial del nombre de etiqueta (sensible a capitalización)
creator	String	Cadena de caracteres parcial del nombre de usuario creador (sensible a capitalización)
followed	Boolean	Si el mazo es seguido
Código de respuesta	**Nombre**	**Descripción**
200	OK	Se envían los mazos que pasen el filtro
204	NO CONTENT	Si no se encuentra ningún mazo válido
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Los dos endpoints anteriores son los más complejos que se han desarrollado en back-end, por lo que es menester explicarlos con un poco de detalle. A diferencia de todo el resto de consultas, que utilizan funciones *find* de Mongoose, haciendo más sencillas las consultas ya que no es necesario realizar peticiones complejas, en estos dos endpoints no ha sido así. Debido a que estas peticiones requieren de comprobar los datos de varios modelos, y que las relaciones que poseen entre sí se han realizado utilizando el ObjectID del objeto a referenciar, no es posible hacer el filtrado en una sola petición *find*, y se ha tenido que recurrir a lo que

en BDs SQL se conoce como *join*, y que en MongoDB se realiza mediante la función *aggregate*. Mediante esta función, se posibilita realizar comprobaciones en los datos de varios modelos de forma conjunta (por ejemplo, en el endpoint anterior se comprueban en conjunto los datos de la colección de mazos, usuarios, temas y etiquetas).

Tabla 5.30: *Detalles del endpoint: Valorar mazo*

Tipo	**Ruta**	**Descripción**
POST	/decks/:id/ratings	Posibilita la valoración de un mazo
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo
rate	Integer	Valoración realizada por el usuario
Código de respuesta	**Nombre**	**Descripción**
201	CREATED	Tras crear la valoración
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
400	BAD REQUEST	Si la valoración es inferior a 0, superior a 10 (las valoraciones se almacenan como enteros porque el incremento permitido es solo de 0.5) o no es un número entero
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo
409	CONFLICT	Si el usuario ya tiene una valoración del mazo

Tabla 5.31: *Detalles del endpoint: Actualizar valoración de mazo*

Tipo	**Ruta**	**Descripción**
PUT	/decks/:id/ratings	Posibilita la actualización de una valoración de un mazo
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo
rate	Integer	Nueva valoración realizada por el usuario
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras actualizar la valoración
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
400	BAD REQUEST	Si la valoración es inferior a 0, superior a 10 o no es un número entero
404	BAD REQUEST	Si el usuario no tiene valoración previa
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo

Tabla 5.32: *Detalles del endpoint: Borrar valoración de mazo*

Tipo	**Ruta**	**Descripción**
DELETE	/decks/:id/ratings	Posibilita la eliminación de una valoración de un mazo
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo
Código de respuesta	**Nombre**	**Descripción**
204	NO CONTENT	Tras eliminar la valoración
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo

Tabla 5.33: *Detalles del endpoint: Obtener valoración de mazo del usuario registrado*

Tipo	**Ruta**	**Descripción**
GET	/decks/:id/ratings	Devuelve la valoración del mazo realizada por el usuario registrado
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo
Código de respuesta	**Nombre**	**Descripción**
200	OK	Junto con la valoración del usuario
204	NO CONTENT	Si el usuario no ha valorado el mazo previamente
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo

5.3 Iteración 3

Tras haber concluido la anterior iteración, donde se introdujeron gran cantidad de casos de uso al sistema, se comenzó el desarrollo de la que sería la última iteración de la fase de implementación, donde se han codificado los últimos casos de uso que se han considerado desarrollar. En esta iteración, además, se ha tratado de realizar el despliegue de tanto el repositorio de front-end como el de back-end, con mayor o menor éxito, como se explicará en la sección 5.3.1. En lo que respecta a los casos de uso implementados, estos se pueden encontrar listados a continuación:

- Entrenar con mazo
- Actualizar entrenamiento de mazo
- Borrar entrenamiento de mazo
- Ver estadísticas
- Mostrar carta en entrenamiento
- Ocultar carta en entrenamiento
- Cambiar la cantidad de cajas
- Cambiar el retroceso tras fallo
- Ver perfil
- Logout

También se quiere mencionar que, tras la finalización de esta iteración, algunos casos de uso que se plantearon no han sido desarrollados, principalmente debido a falta de tiempo. Estos son los destinados a la gestión del administrador, ya que durante el desarrollo esta gestión se ha realizado directamente sobre la BD, y posteriormente se han postergado debido a que son los casos de uso de menor importancia. Asimismo tampoco es posible actualizar los datos de un usuario. A continuación se listan todos los casos de uso no implementados:

- Actualizar usuario
- Inhabilitación de usuarios
- Crear temas
- Actualizar temas
- Borrar temas
- Leer tema
- Leer etiqueta
- Borrar etiqueta

Al igual que en la iteración anterior, la estimación realizada ha sido correcta: en este caso, solo se dispuso de semana y media para esta iteración, pero se consiguió cumplir con el plazo de forma satisfactoria.

5.3.1 Despliegue Continuo de ambos repositorios

Antes de comenzar a detallar las interfaces y los endpoints de esta iteración, es menester mencionar otra tarea realizada: el despliegue continuo de ambos repositorios. A continuación, se detalla este proceso.

Para el despliegue continuo, se barajaron diferentes proveedores que permitiesen realizar esta labor de forma gratuita. Aunque las opciones más populares (AWS y Google Cloud) ofrecen normalmente crédito inicial para probar su servicio, ya no se disponía de dicho crédito, por lo que hubo que barajar otras opciones. Entre las diferentes alternativas encontradas, se optó por utilizar Render, ya que su tarifa gratis ofrecía proteger las comunicaciones HTTP mediante TLS, además de otras medidas de seguridad que no ofrecían otros servicios.

El primer paso es introducir las directivas a realizar en el fichero que se encargaba de la integración continua. Además, el despliegue será dependiente de la integración, por lo que si los test fallasen, no se realizaría el despliegue pertinente. A continuación se muestran un par de capturas del repositorio de back-end donde es posible apreciar el flujo de trabajo desplegando tras una integración continua exitosa, y como se prescinde del susodicho despliegue en caso de error:

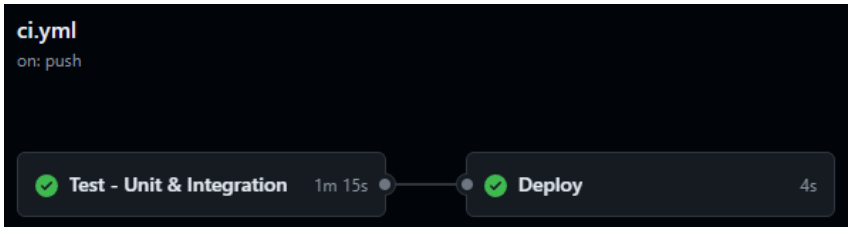


Figura 5.31: *Despliegue Continuo en GitHub Actions funcionando correctamente*

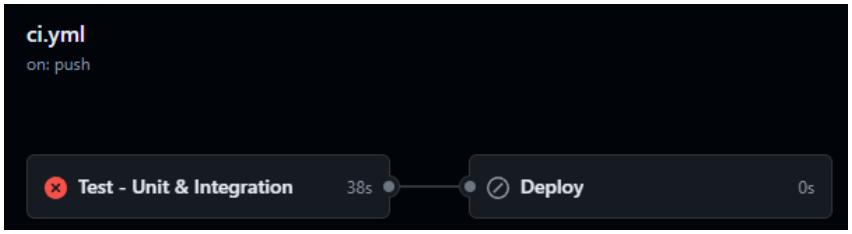


Figura 5.32: *Despliegue Continuo en GitHub Actions sin realizarse tras error de integración*

Para que este despliegue se realice de forma correcta, también es necesario aplicar cierta configuración en Render. Principalmente, se ha requerido aplicar la misma configuración que al realizar la integración continua: en el caso del repositorio de back-end, creando los ficheros *.env* para las variables de entorno, así como las claves para la firma de los token JWT, además de definir los comandos a usar para lanzar la aplicación.

Sin embargo, existe una necesidad más para poder realizar este despliegue: la BD del sistema. A causa de utilizar la versión gratuita de Render, la BD debía ser externalizada para poder ser parte del despliegue, por lo que se optó por crear una base de datos en MongoDB Atlas, accesible para el repositorio de back-end de forma remota. Una vez hecho eso, y con la modificación previamente mencionada al flujo de trabajo de los GitHub Actions, el despliegue continuo se realiza correctamente.

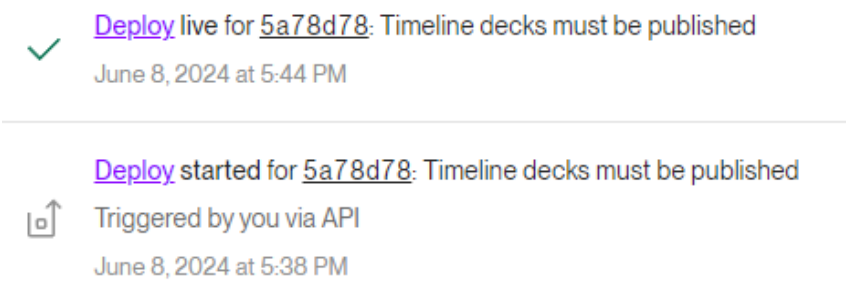


Figura 5.33: *Back-end de IntelliDeck desplegado correctamente en Render*

A pesar del éxito al realizar el despliegue, se ha descubierto que Render no proporciona un servicio de calidad suficiente en su plan gratuito como para que la aplicación desplegada sea usable, ya que en caso de que no se reciban peticiones en 15 minutos, el servicio sufrirá, como se le llama en Render, un *spin down*: se reducirá la velocidad a la que el servicio atiende las peticiones, haciendo que pueda llegar a tardar 50 segundos o más en responder a una sola petición. Esto es, lógicamente, inviable para el despliegue de una aplicación en producción; sin embargo, a causa de que la principal intención de integrar este servicio es aprender a realizar el despliegue continuo de una página web, se ha decidido tolerar la mencionada limitación.

Además, es importante subrayar que solo ha sido posible realizar el despliegue del repositorio de back-end: aunque se ha realizado de igual manera la configuración para front-end, debido a limitaciones de memoria no se ha conseguido realizar el despliegue. La causa de este problema es que, en Render, el plan gratuito solo permite la utilización de 512 MB para memoria. Sin realizar retoques en el consumo de memoria, el despliegue de la aplicación de Angular excedía el límite establecido, por lo cual no era posible el despliegue. Por ello, se optó por reducir la cantidad de memoria que requería el repositorio para poder ser desplegado. Sin embargo, no se consiguió encontrar la cantidad correcta de memoria que posibilitase el despliegue: o la memoria determinada era demasiado poca, o suponía que se excediese el límite de Render. A continuación, se muestran los errores obtenidos en los intentos de despliegue:

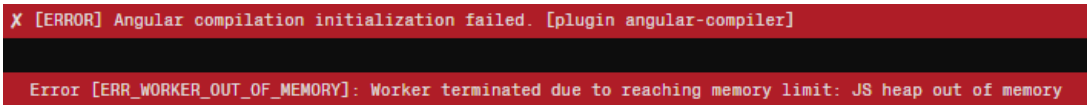


Figura 5.34: *Error en la consola de Render al asignar memoria insuficiente para el despliegue*

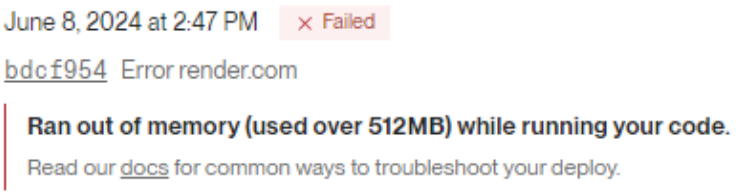


Figura 5.35: *Error tras intento de despliegue en Render superando el límite de memoria del plan gratuito*

A pesar de los problemas de Render al desplegar este repositorio, se optó por no cambiar de proveedor de alojamiento, debido a que con el resto de alternativas del mercado iba a suceder el mismo problema, al menos en lo que respecta a los planes gratuitos.

5.3.2 Interfaces de usuario implementadas

En esta iteración, se han terminado de implementar las interfaces que llegarán a la versión final de este proyecto, las cuales se centran principalmente en permitir los entrenamientos con los mazos. Este proceso sigue el flujo que se estableció para el caso de uso de entrenar con mazo y, a continuación, se presentarán las interfaces de usuario creadas para ello.

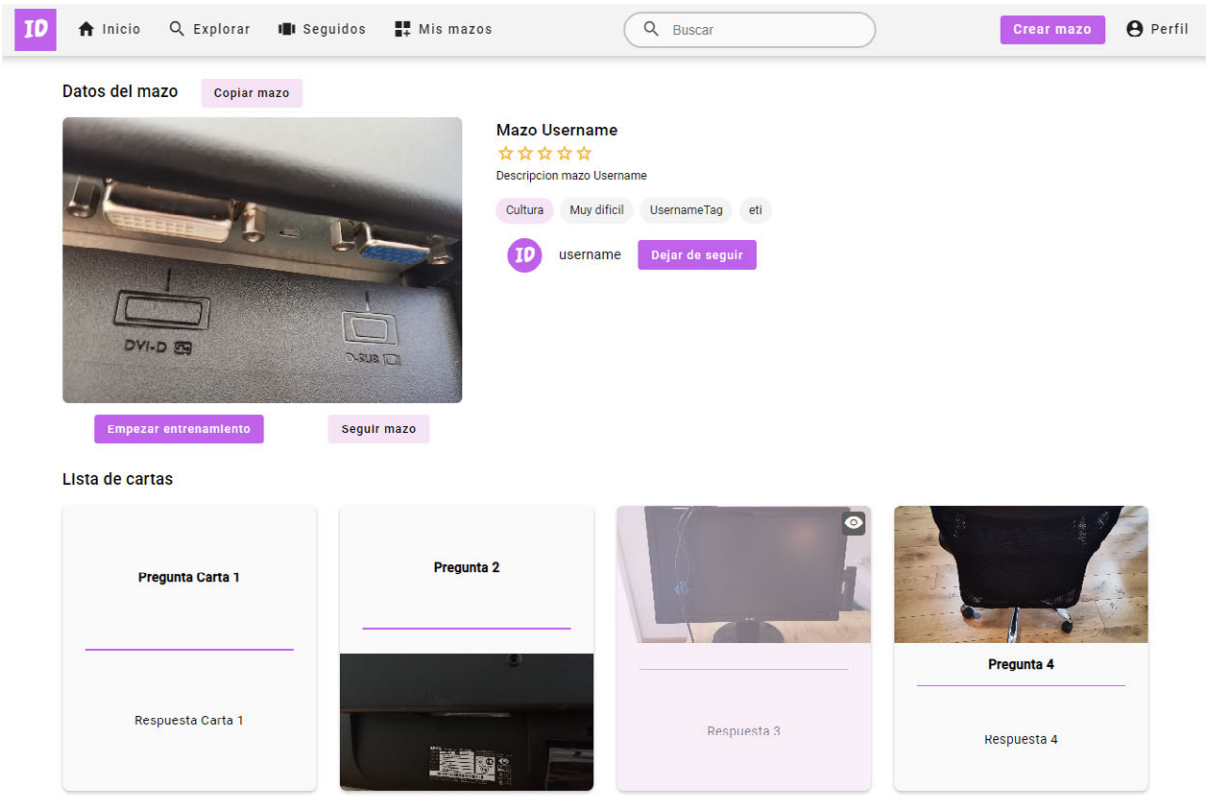


Figura 5.36: *Interfaz de usuario: Mazo externo con funcionalidades de ocultar cartas*

Para permitir hacer entrenamientos, es importante también proporcionar las utilidades para aplicar los casos de uso de ocultar y mostrar cartas, de modo que el estudio se pueda hacer de forma satisfactoria. En la anterior imagen, se puede apreciar que, al pasar por una carta, esta se podrá ocultar (o mostrar, dependiendo de su estado). A continuación se muestra cómo se presentan las cartas que están ocultas:

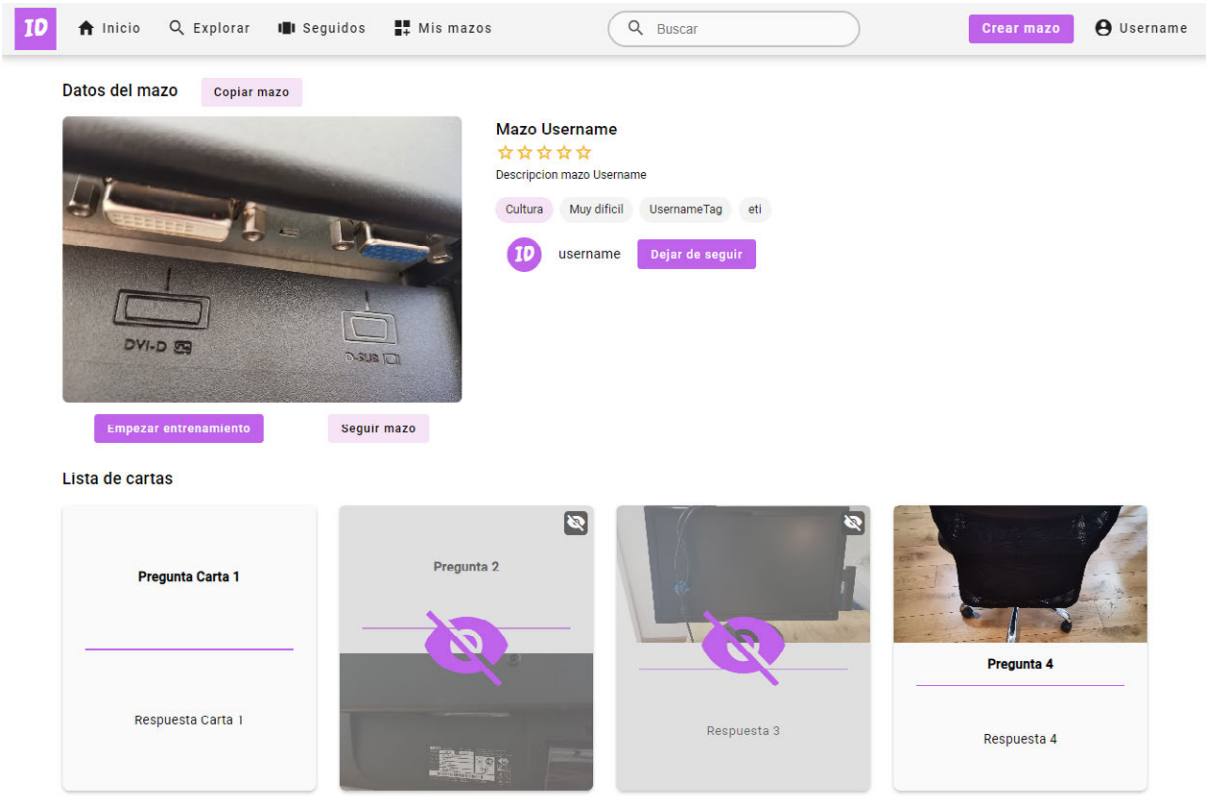


Figura 5.37: *Interfaz de usuario: Mazo externo con cartas ocultas*

Para iniciar un entrenamiento, bastará con seleccionar el botón “Empezar entrenamiento”. Tras clicar este botón, se mostrará un diálogo que servirá para definir la caja máxima del estudio (siendo el máximo de la aplicación la caja 13: es decir, con un espaciado de 377 días del estudio anterior) y el retroceso en las cajas: siguiendo lo establecido por las diferentes alternativas de aplicar el sistema Leitner mostradas en la imagen 1.2, se permitirá retroceder a la caja anterior o a la primera caja.

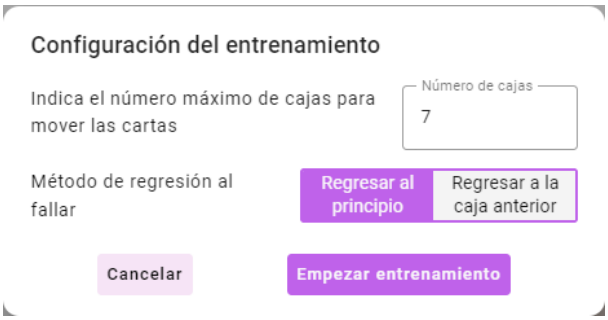


Figura 5.38: *Interfaz de usuario: Diálogo para configuración de entrenamiento*

Una vez se inicie el estudio, se irán mostrando las preguntas de las diferentes cartas. El usuario pensará la respuesta para sí, girará la carta y decidirá si su respuesta es la correcta o no. Existen diferentes diseños para presentar las preguntas y respuestas, dependiendo de los datos establecidos en la propia carta, mostrándose en las siguientes imágenes distintas representaciones:

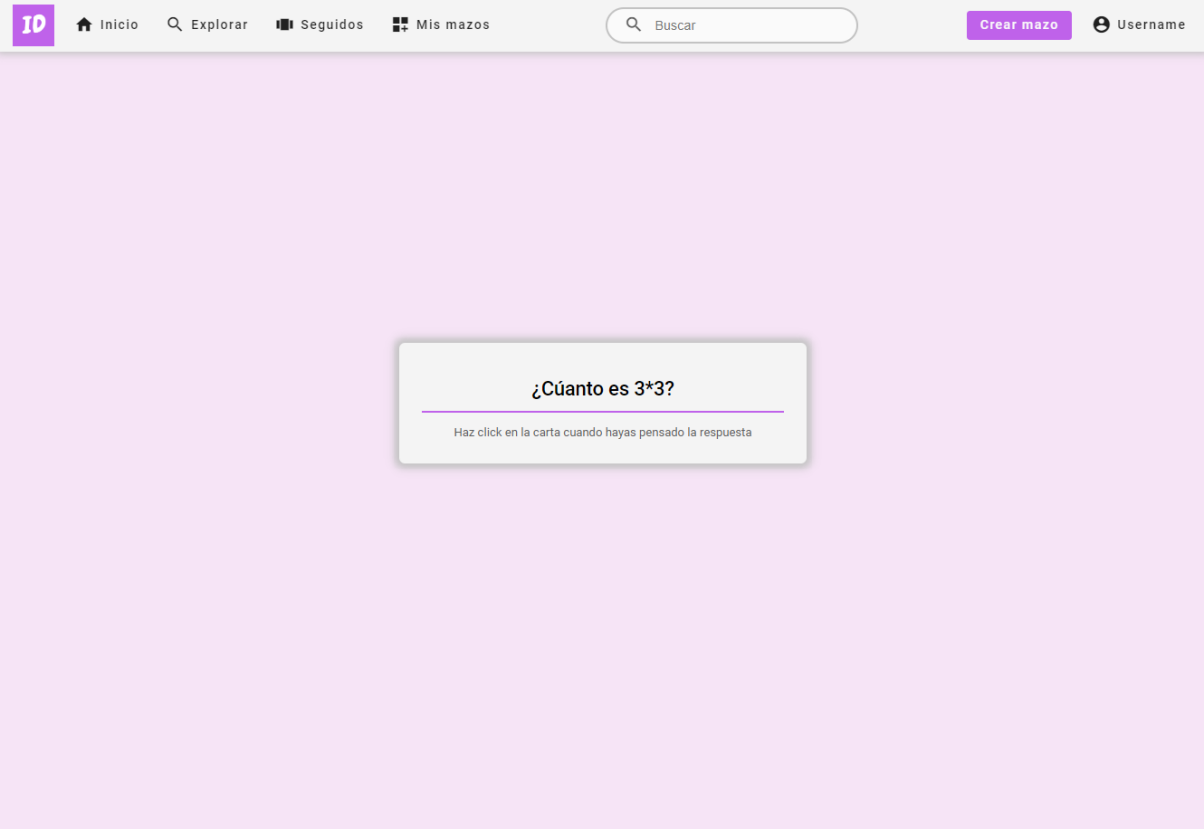


Figura 5.39: *Interfaz de usuario: Carta por girar con pregunta sin imagen*

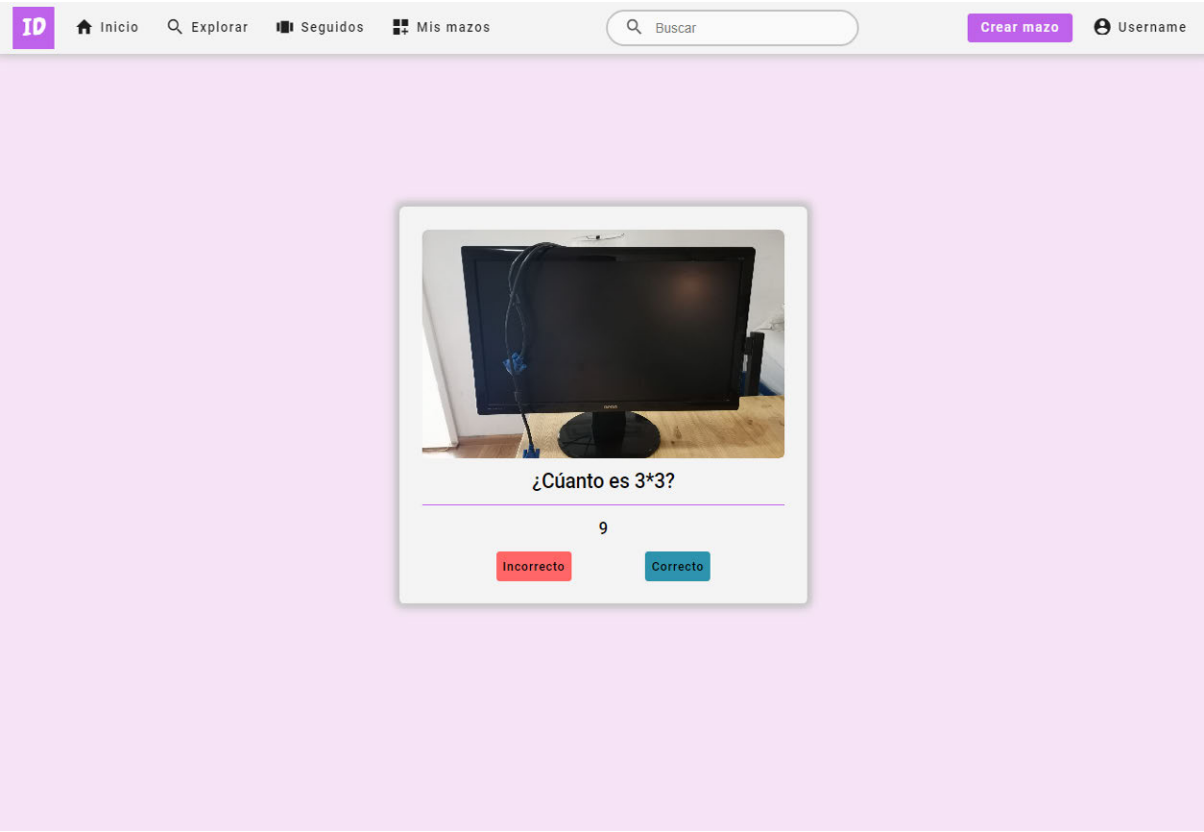


Figura 5.40: *Interfaz de usuario: Carta por responder con imagen en pregunta*

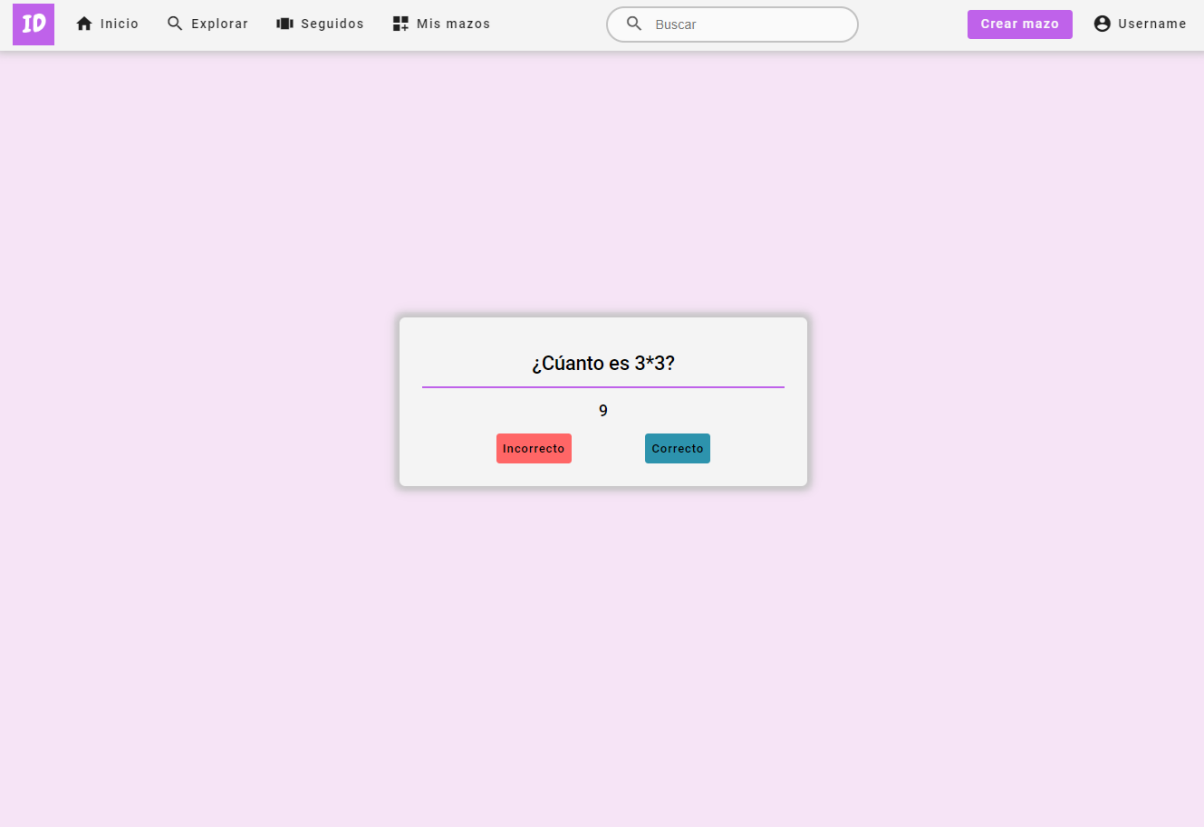


Figura 5.41: *Interfaz de usuario: Carta por responder sin imagen*

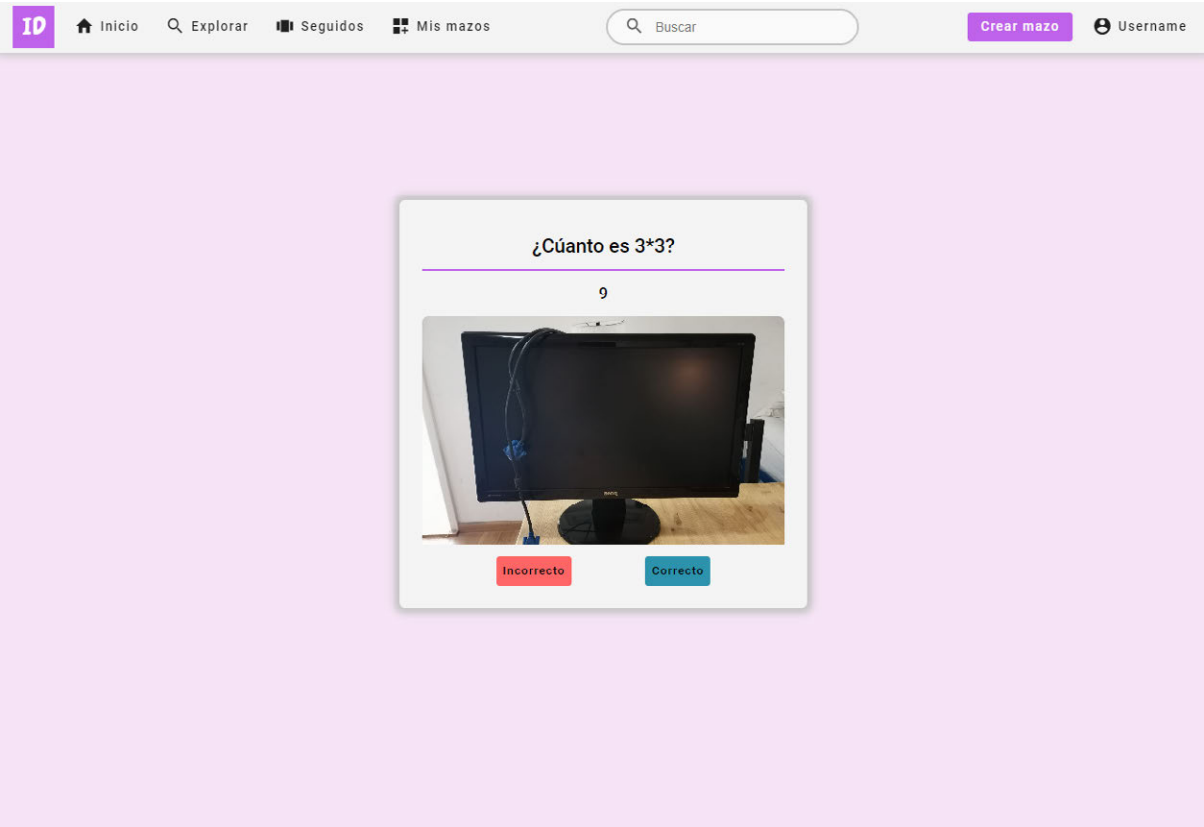


Figura 5.42: *Interfaz de usuario: Carta por responder con imagen en respuesta*

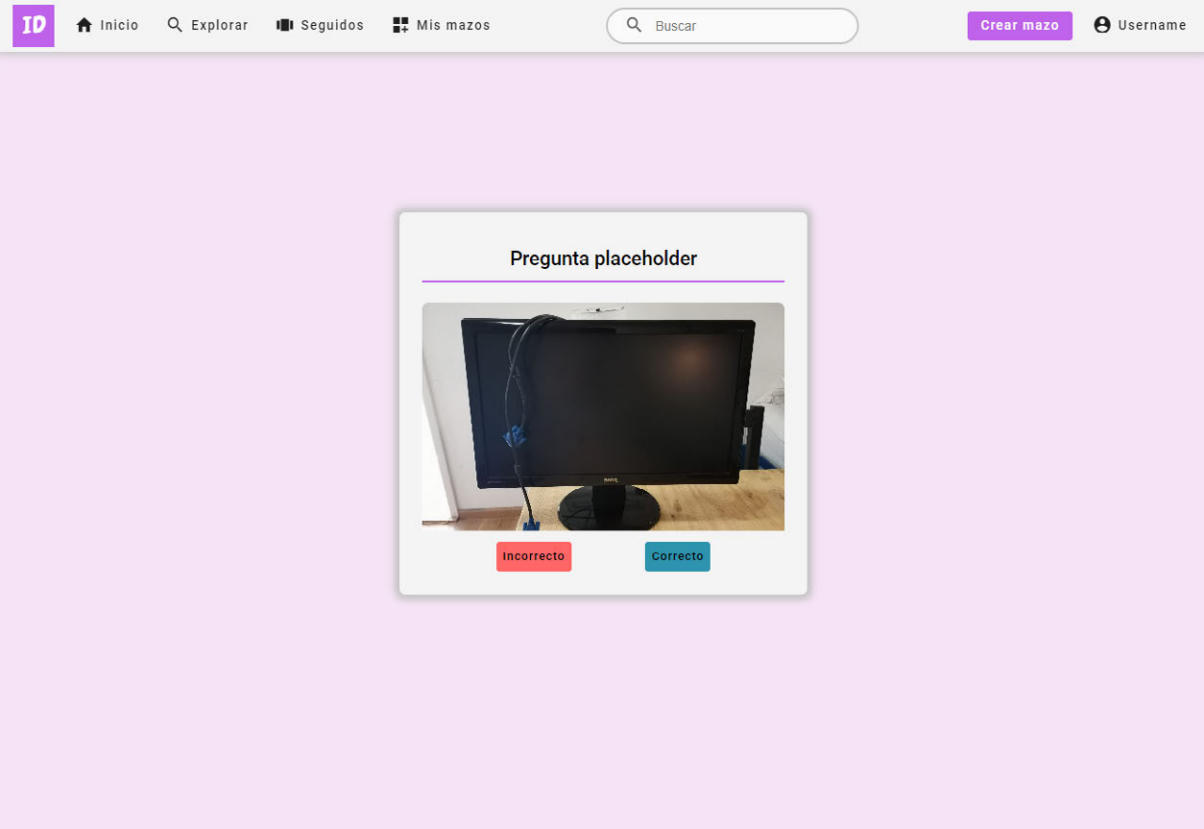


Figura 5.43: *Interfaz de usuario: Carta por responder con únicamente imagen en respuesta*

Una vez terminado el estudio de un mazo, se presentarán las estadísticas, mostrando el tiempo del estudio recién finalizado, la media de tiempo y la cantidad de intentos. Además, se presentará un gráfico representando cuántas cartas se hallan en cada una de las cajas.

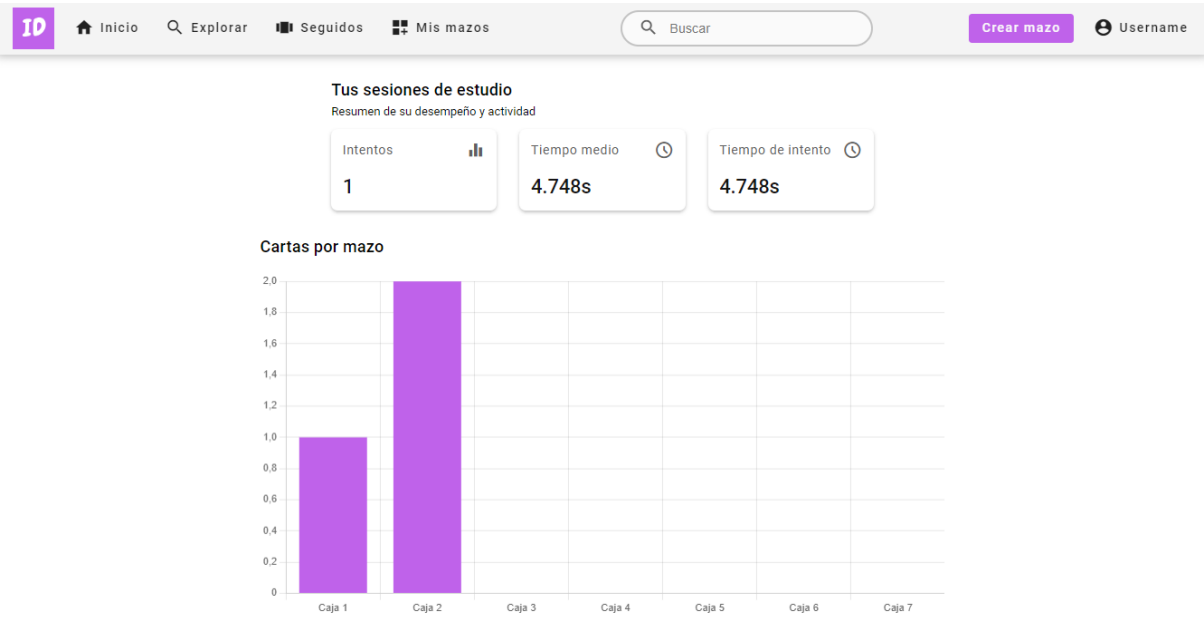


Figura 5.44: *Interfaz de usuario: Estadísticas del estudio del mazo*

Al regresar ahora a la pantalla del mazo estudiado, se presentarán nuevos botones: por un lado, el usuario tendrá la opción de eliminar su estudio, por las causas que el considere oportunas (como que ya no necesita seguir estudiando el mazo, por ejemplo) y, por el otro lado, podrá reiniciar el entrenamiento, haciendo que todas las cartas vuelvan a estar en la primera caja.

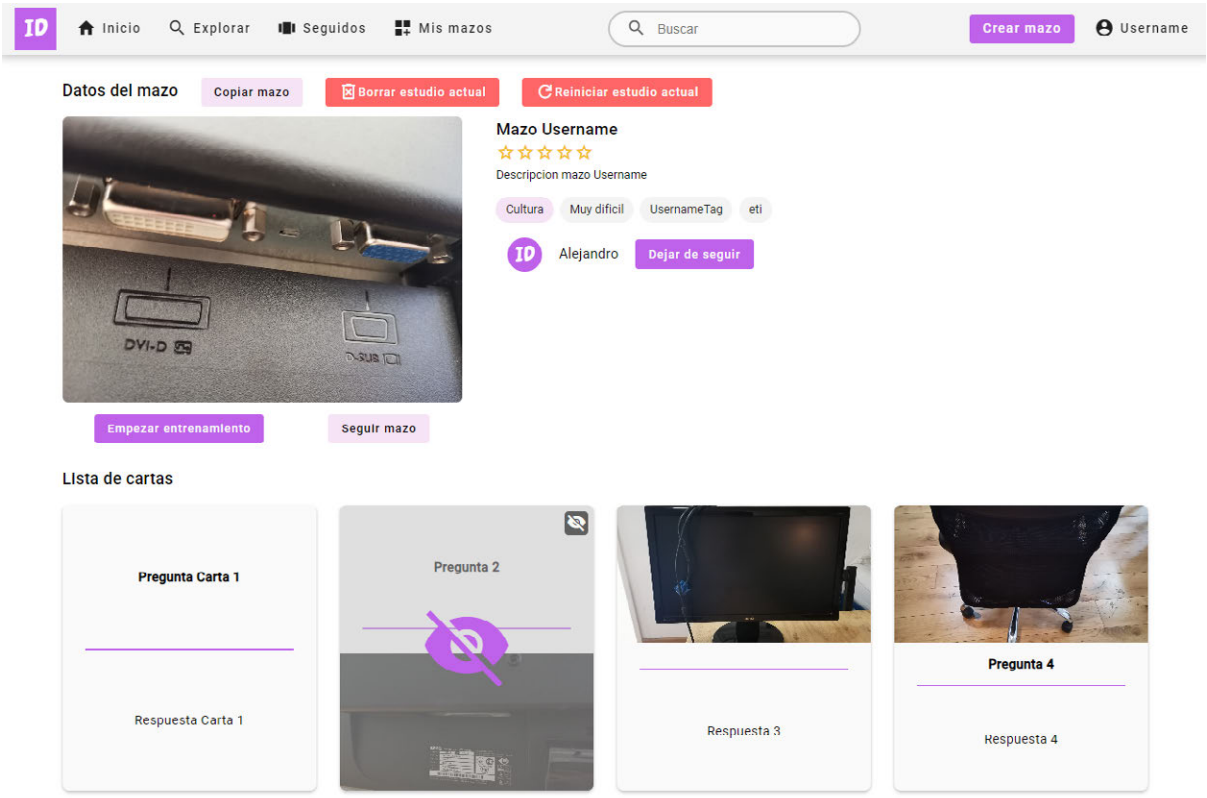


Figura 5.45: *Interfaz de usuario: Nuevos botones para el reinicio y borrado del entrenamiento*

Además, cuando un usuario trate de realizar un entrenamiento, y en dicho día no tenga que estudiar ninguna carta, se le mostrará un diálogo informando que el estudio no computará en las estadísticas, y tampoco repercutirá en la posición de las cartas en las cajas. Si el usuario hace unos de estos entrenamientos (denominados entrenamientos fuera de plazo), tampoco se mostrarán las estadísticas al finalizar de entrenar.

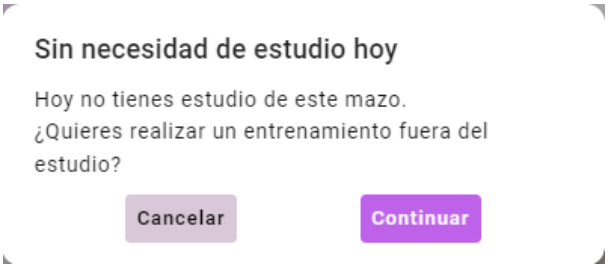


Figura 5.46: *Interfaz de usuario: Diálogo informando de que el estudio será fuera de plazo*

Como se puede apreciar en la barra de navegación de todas las pantallas mostradas en esta iteración, se ha comenzado a mostrar el nombre del usuario registrado. Esto se debe a que se

ha comenzado a ofrecer la funcionalidad de acceder al perfil de los usuarios, incluido el de uno mismo.

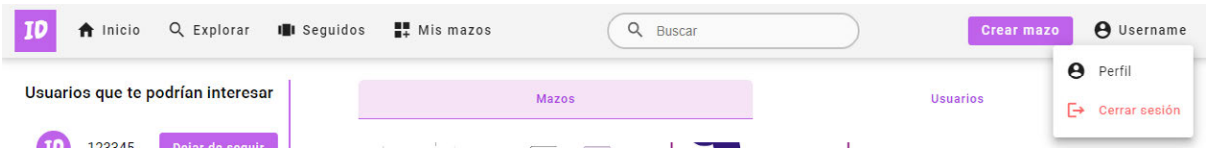


Figura 5.47: *Interfaz de usuario: Acceso a perfil y logout*

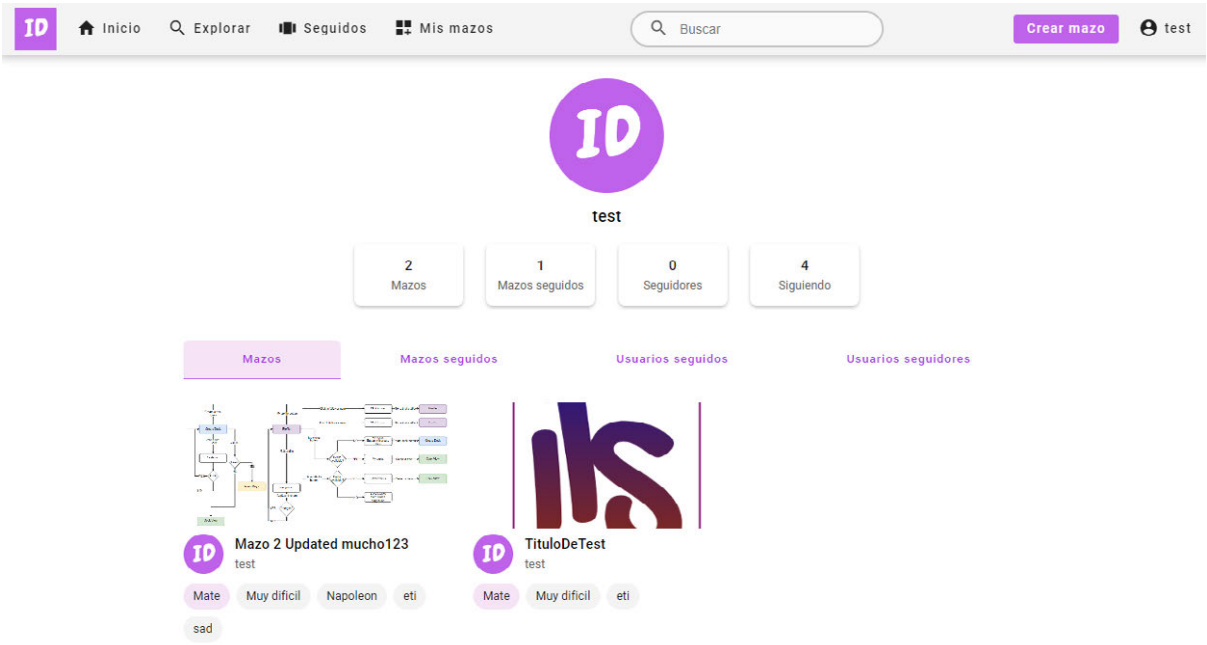


Figura 5.48: *Interfaz de usuario: Perfil de usuario*

En esta pantalla se muestra el nombre de usuario y la foto de perfil de cada usuario, así como la cantidad de mazos creados, de mazos seguidos, de seguidores, y de usuarios seguidos, así como un listado con cada uno de estos datos. También es posible acceder a los perfiles de otros usuarios al clicar en ellos en otras pantallas, como el timeline, por ejemplo.

5.3.3 Endpoints implementados

En esta última iteración, se han creado todos los endpoints necesarios para posibilitar los entrenamientos de repetición espaciada mediante los mazos ya publicados. Además, también se han introducido los endpoints para ciertos casos de uso relacionados con los usuarios, como los que sirven para ver el perfil, por ejemplo. A continuación, y como ya se ha realizado anteriormente en las iteraciones previas, se presentan las especificaciones de cada uno de los endpoints implementados:

Tabla 5.34: *Detalles del endpoint: Leer usuario*

Tipo	**Ruta**	**Descripción**
GET	/users/:id	Devuelve un usuario en base a un identificador
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del usuario
Código de respuesta	**Nombre**	**Descripción**
200	OK	Se envía el usuario asociado a la ID
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el usuario especificado

Tabla 5.35: *Detalles del endpoint: Leer información de usuario registrado*

Tipo	**Ruta**	**Descripción**
GET	/users/logged	Permite leer los datos del usuario registrado
Código de respuesta	**Nombre**	**Descripción**
200	OK	Junto con los datos del usuario
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.36: *Detalles del endpoint: Leer seguidores de usuario*

Tipo	**Ruta**	**Descripción**
GET	/users/followers/:id	Devuelve los seguidores del usuario especificado
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del usuario
Código de respuesta	**Nombre**	**Descripción**
200	OK	Se envían todos los seguidores
204	NO CONTENT	Si el usuario no es seguido por otros usuarios
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión

Tabla 5.37: *Detalles del endpoint: Crear entrenamiento de mazo*

Tipo	**Ruta**	**Descripción**
POST	/decks/:id/deckTraining	Crea un entrenamiento del mazo solicitado para el usuario registrado
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo
boxAmount	Integer	La cantidad de cajas del entrenamiento
backtrack	String	A qué caja tienen que retroceder las cartas falladas, enumerado (BACKTRACK_PRIOR, BACKTRACK_FIRST)
cards	Object[]	Lista de las cartas, compuestas cada una de los siguientes datos:
→ id	ObjectID	El identificador de la carta
→ isShown	Boolean	Si la carta debe mostrarse o no
Código de respuesta	**Nombre**	**Descripción**
201	CREATED	Tras crear el entrenamiento de mazo
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
400	BAD REQUEST	Si *boxAmount* es inferior a 1, superior a 13 (el límite de cajas de la aplicación) o no es un número entero
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo
409	CONFLICT	Si el usuario ya tiene un estudio de dicho mazo

Tabla 5.38: *Detalles del endpoint: Borrar entrenamiento de mazo*

Tipo	**Ruta**	**Descripción**
DELETE	/decks/:id/deckTraining	Elimina el entrenamiento del mazo solicitado para el usuario registrado
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo
Código de respuesta	**Nombre**	**Descripción**
204	NO CONTENT	Tras eliminar el entrenamiento de mazo
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo

Tabla 5.39: *Detalles del endpoint: Actualizar entrenamiento de mazo*

Tipo	**Ruta**	**Descripción**
PUT	/decks/:id/deckTraining	Actualiza el entrenamiento del mazo solicitado para el usuario registrado
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo
boxAmount	Integer	La cantidad de cajas del entrenamiento
backtrack	String	A qué caja tienen que retroceder las cartas falladas, enumerado (BACKTRACK_PRIOR, BACKTRACK_FIRST)
completionTimeSeconds	Integer	Segundos requeridos para realización del intento
resetDate	Boolean	En caso de que se quiera reiniciar la fecha de inicio del entrenamiento
cards	Object[]	Lista de las cartas, compuestas cada una de los siguientes datos:
→ id	ObjectID	El identificador de la carta
→ box	Integer	Nueva caja en la que situar la carta
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras actualizar el entrenamiento de mazo
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
400	BAD REQUEST	Si se trata de actualizar entrenamiento tras una sesión y no se manda el tiempo utilizado para el intento
400	BAD REQUEST	Si el tiempo del intento es inferior a 0
400	BAD REQUEST	Si *boxAmount* es inferior a 1, superior a 13 (el límite de cajas de la aplicación) o no es un número entero
400	BAD REQUEST	Si la caja de un carta es inferior a 0, superior a *boxAmount* o no es un número entero
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo
404	NOT FOUND	Si no se encuentra entrenamiento del mazo para el usuario

El endpoint previamente mostrado es capaz de cumplir dos funciones dentro de la aplicación: por un lado, actualiza el entrenamiento del mazo tras una sesión de estudio y, por el otro lado, se puede utilizar para reiniciar el estudio del mazo, en caso de que el usuario lo solicite. A causa de esta doble utilización, los campos *completionTimeSeconds* y *cards* son únicamente para actualizar tras un entrenamiento, mientras el resto sirven para el reinicio del estudio.

Tabla 5.40: *Detalles del endpoint: Leer entrenamiento de mazo*

Tipo	**Ruta**	**Descripción**
GET	/decks/:id/deckTraining	Devuelve el entrenamiento del mazo solicitado del usuario registrado
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo
Código de respuesta	**Nombre**	**Descripción**
200	OK	Junto con el entrenamiento del mazo
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo
404	NOT FOUND	Si no se encuentra el entrenamiento de mazo del usuario registrado

Es importante matizar que, tras crear los modelos del sistema, las estadísticas especificadas en el modelo del dominio de la imagen 3.1 pasaron a ser un campo más de la clase *DeckTraining*, debido a que la relación era 1 a 1. Por ello, la utilización de este endpoint también es requerida para obtener las estadísticas tras una sesión de estudio.

Tabla 5.41: *Detalles del endpoint: Mostrar carta en entrenamiento*

Tipo	**Ruta**	**Descripción**
PUT	/decks/:id/deckTraining/cards/:cardId/show	Actualiza una carta para que sea mostrada en el entrenamiento del usuario
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo
cardId	ObjectID	Identificador de carta
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras actualizar la carta
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo
404	NOT FOUND	Si no se encuentra la carta especificada en el mazo
404	NOT FOUND	Si no se encuentra el entrenamiento de mazo del usuario registrado

Tabla 5.42: *Detalles del endpoint: Ocultar carta en entrenamiento*

Tipo	**Ruta**	**Descripción**
PUT	/decks/:id/deckTraining/cards/:cardId/hide	Actualiza una carta para que se oculte en el entrenamiento del usuario
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo
cardId	ObjectID	Identificador de carta
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras actualizar la carta
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo
404	NOT FOUND	Si no se encuentra la carta especificada en el mazo
404	NOT FOUND	Si no se encuentra el entrenamiento de mazo del usuario registrado

Tabla 5.43: *Detalles del endpoint: Obtener todas las cartas para realizar un entrenamiento*

Tipo	**Ruta**	**Descripción**
GET	/decks/:id/deckTraining/cards	Devuelve todas las cartas para un entrenamiento
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras devolver las cartas a estudiar
204	NO CONTENT	En el hipotético caso de que no existiesen cartas en el mazo
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo
404	NOT FOUND	Si no se encuentra el entrenamiento de mazo del usuario registrado

Tabla 5.44: *Detalles del endpoint: Obtener todas las cartas para entrenamiento el día actual*

Tipo	**Ruta**	**Descripción**
GET	/decks/:id/deckTraining/cards/today	Devuelve todas las cartas para un entrenamiento en el día actual
Parámetros	**Tipo**	**Descripción**
id	ObjectID	Identificador del mazo
Código de respuesta	**Nombre**	**Descripción**
200	OK	Tras devolver las cartas a estudiar
204	NO CONTENT	Si no se debe estudiar ninguna carta en el día actual
400	BAD REQUEST	Si el ID enviado no puede convertirse a ObjectID
401	UNAUTHORIZED	El usuario no ha iniciado sesión
404	NOT FOUND	Si no se encuentra el mazo
404	NOT FOUND	Si no se encuentra el entrenamiento de mazo del usuario registrado

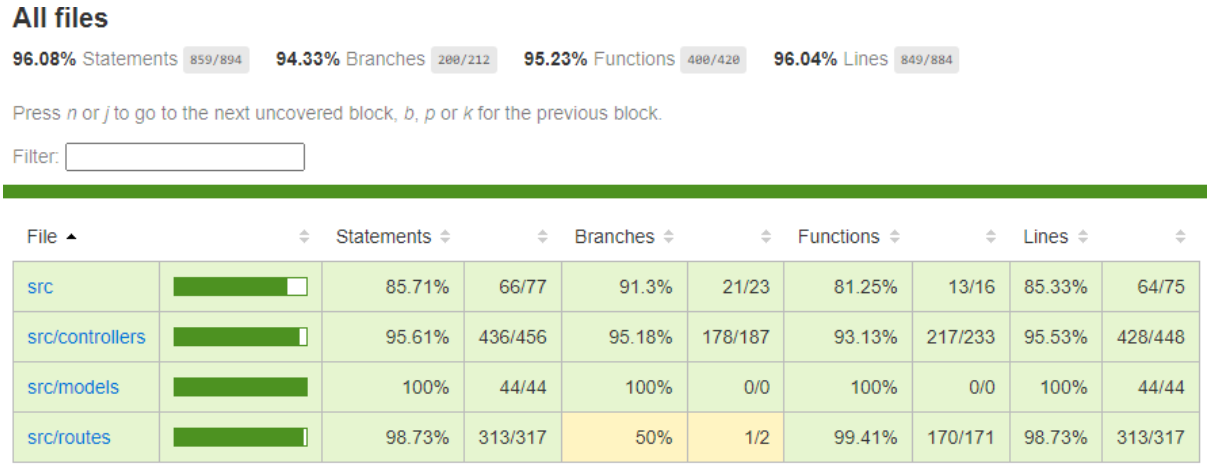
Los anteriores dos endpoints cumplen la misma función (que es devolver las cartas a estudiar), pero para situaciones distintas entre sí. Por un lado, el endpoint 5.44 sirve para realizar un entrenamiento normal: se solicitan las cartas a estudiar en el día concreto, y esas son las que se estudian. Sin embargo, es posible que el usuario quiera realizar un estudio fuera de plazo: en ese caso, se utiliza el endpoint 5.43, que devuelve todas las cartas para un estudio de este tipo.

5.4 Pruebas realizadas en el proyecto y resultados del análisis estático

A la par que se ha ido desarrollando la aplicación, también se han realizado test que validen la corrección del código, de modo que el sistema final sea de calidad. En concreto, se han realizado tres tipos de test: test de integración y test de mutación en back-end, y test unitarios en front-end.

Es importante matizar que la razón de no haber realizado test unitarios en back-end es que tanto las rutas como los modelos carecen casi por completo de lógica, por lo que realizar test unitarios para esos tipos de clases aportaría poco valor. Debido a esto, se ha decidido que era preferible unificar la validación en un solo tipo de test, que son los de integración. En contraposición con los test unitarios, que validan el funcionamiento de una sola función (o unidad de código, de ahí el nombre test unitario), sin atender a dependencias externas, los test de integración validan la correcta operatividad de ciertas unidades de código en sincronía, razón por la cual se consideran test de integración [43]. Sin embargo, estos test se han realizado sin la dependencia que genera la utilización de la BD, simplemente para así verificar que el código realizado es funcional siempre que la BD funcione correctamente.

En lo que respecta a las validaciones de back-end, tras realizar test para todos los endpoints presentados en las anteriores iteraciones, se ha obtenido un reporte de cobertura que indica cuánto del total del código es validado por los test. A continuación se presenta dicho reporte:



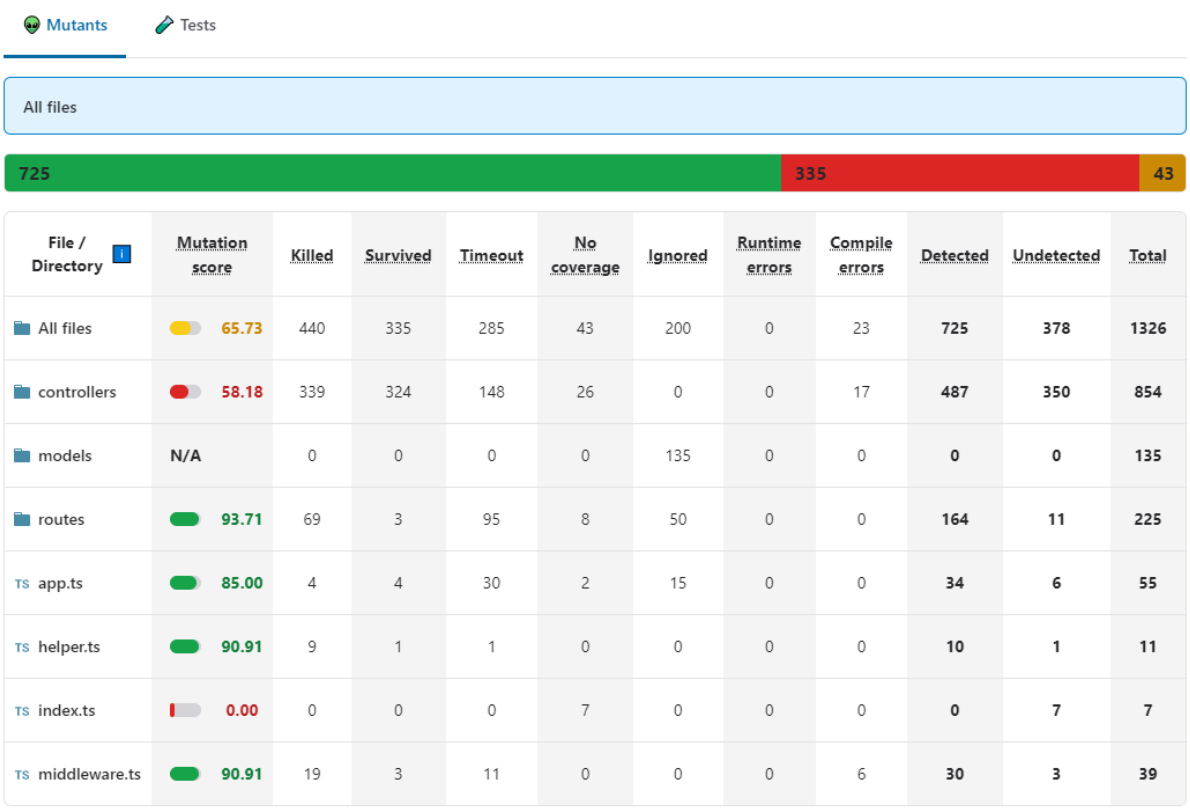
File	Statements		Branches		Functions		Lines	
src	85.71%	66/77	91.3%	21/23	81.25%	13/16	85.33%	64/75
src/controllers	95.61%	436/456	95.18%	178/187	93.13%	217/233	95.53%	428/448
src/models	100%	44/44	100%	0/0	100%	0/0	100%	44/44
src/routes	98.73%	313/317	50%	1/2	99.41%	170/171	98.73%	313/317

Figura 5.49: *Reporte final de la cobertura obtenida en back-end*

Como se puede apreciar, se ha obtenido una cobertura por encima del mínimo recomendado de 80 %, estando la cobertura total cerca del 96 %. Es importante, sin embargo, explicar por qué los ficheros ubicados en *src* tienen una cobertura inferior al resto de clases, y por qué el directorio *src/routes* aparece con el fondo amarillo en ciertas métricas, sinónimo de una validación subóptima: dos son los ficheros que ocasionan esto, y los motivos se detallan a continuación:

- Por un lado, el fichero *src/index.ts*, el cual no es validado en absoluto. Esto se debe a que dicho fichero contiene el código encargado de poner en marcha la API, establecer una conexión con la BD MongoDB y, en caso de que la conexión se efectúe, habilitar el puerto desde el que escuchar las peticiones a las que atender. Al realizar los test la configuración para escuchar las peticiones a las rutas de la API de forma distinta, no es necesario llamar a este fichero, y por tanto no tiene cobertura. Sin embargo, *src/index.ts* funciona adecuadamente fuera del entorno de testing, demostrando que opera de forma correcta.
- Por otro lado, el fichero *src/routes/images.routes.ts* también posee una cobertura inferior a la adecuada, aunque en este caso no es nula. El motivo es que este fichero se encarga de atender, principalmente, a las peticiones realizadas al endpoint descrito en la tabla 5.8. Aunque se ha conseguido dar cobertura a los casos en los que no se manda imagen de forma correcta, no se ha conseguido realizar test que envíen imágenes u otro tipo de archivos para realizar validaciones. Sin embargo, se han realizado pruebas manualmente para asegurar la correcta operatividad del código.

Además de los test de integración realizados, también se han realizado test de mutación. Este tipo de test fue inicialmente propuesto en 1978 [44], y consiste en realizar alteraciones al sistema, con el objetivo de comprobar que los test realizados son lo suficientemente adecuados como para detectar estas mutaciones. Debido a que los test de integración detectan el código muerto, pero no son especialmente útiles a la hora de validar que los test sean capaces de detectar los fallos de forma óptima, los test de mutación permiten cubrir dicho hueco.



Mutants Tests

All files

725 335 43

File / Directory	Mutation score	Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
All files	65.73	440	335	285	43	200	0	23	725	378	1326
controllers	58.18	339	324	148	26	0	0	17	487	350	854
models	N/A	0	0	0	0	135	0	0	0	0	135
routes	93.71	69	3	95	8	50	0	0	164	11	225
app.ts	85.00	4	4	30	2	15	0	0	34	6	55
helper.ts	90.91	9	1	1	0	0	0	0	10	1	11
index.ts	0.00	0	0	0	7	0	0	0	0	7	7
middleware.ts	90.91	19	3	11	0	0	0	6	30	3	39

Figura 5.50: *Reporte final de los test de mutación realizados en back-end*

Como se puede apreciar en la anterior imagen, prácticamente dos tercios de los mutantes generados son eliminados, por que los test están cumpliendo su labor acabando con las alteraciones. Sin embargo, que un tercio de los mutantes sean capaces de sobrevivir no suele ser un buen indicativo, por lo que conviene explicar la razón de tal cantidad de supervivientes: la mayoría de los mutantes que han sobrevivido son aquellos que modifican las peticiones a BD. Esto se debe a que, como se ha mencionado con anterioridad, no se han realizado test de integración con la BD, por lo que en su lugar, se generan *mocks* mediante Jest que simulan las respuestas de la BD, para que el test pueda funcionar con normalidad. A causa de que la petición definida no es la que se realiza en los test, todos los mutantes que modifican las peticiones sobreviven. Sin embargo, eliminar estos mutantes es sencillo, en tanto que solo es necesario la realización de los test que integren la conexión con la BD. Sin embargo, por falta de tiempo, no se han realizado este tipo de test.

Pese a que los test de integración realizados han sido incluidos en el proceso de CI/CD, no es el caso para los test de mutación. Esto se debe a que a diferencia de los primeros, que requieren menos de un minuto, el tiempo requerido para la ejecución de los test de mutación es aproximado a una hora, inviable para ser introducido en CI/CD, por lo que solamente se han realizado estas validaciones en local.

En lo que respecta a las validaciones de front-end, por falta de tiempo, no se han podido realizar todos los test que se desearían. Al igual que en back-end, en el repositorio de Angular se han creado test que validen el código para obtener un reporte de cobertura del mismo. Sin embargo, en este repositorio el porcentaje de cobertura no es tan elevado como lo era en back-end. A continuación se presenta parte del reporte de cobertura de front-end:

All files

50.03% Statements 759/1517 **28.97%** Branches 82/283 **43.24%** Functions 224/518 **49.57%** Lines 705/1422

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
environments		100%	1/1	100%	0/0	100%	0/0	100%	1/1
app/models		54.72%	110/201	30%	6/20	45.58%	62/136	57.59%	110/191
app/core/trainings		16.41%	11/67	0%	0/10	18.18%	2/11	13.84%	9/65
app/core/local		76%	19/25	100%	1/1	66.66%	12/18	75%	18/24
app/core		68.16%	152/223	68.57%	24/35	70.45%	31/44	67.28%	146/217
app/components/user-profile		33.33%	13/39	20%	1/5	13.33%	2/15	29.72%	11/37
app/components/user-list/user		40%	8/20	0%	0/1	37.5%	3/8	38.88%	7/18
app/components/user-list		100%	1/1	100%	0/0	100%	0/0	100%	1/1
app/components/user-decks		87.5%	7/8	100%	0/0	75%	3/4	87.5%	7/8
app/components/timeline		70%	21/30	50%	1/2	60%	9/15	67.85%	19/28

Figura 5.51: *Reporte final de la cobertura obtenida en front-end*

Además de los test realizados, también se ha recurrido al análisis de código mediante SonarCloud. Esta medida ha sido aplicada al proceso de integración continua de ambos repositorios, tanto front-end como back-end, ya que permite la localización de errores durante el desarrollo. A continuación, se muestra la *quality gate* de SonarCloud para back-end, la cual sirve para determinar si el estado del código de un repositorio es adecuado o no:

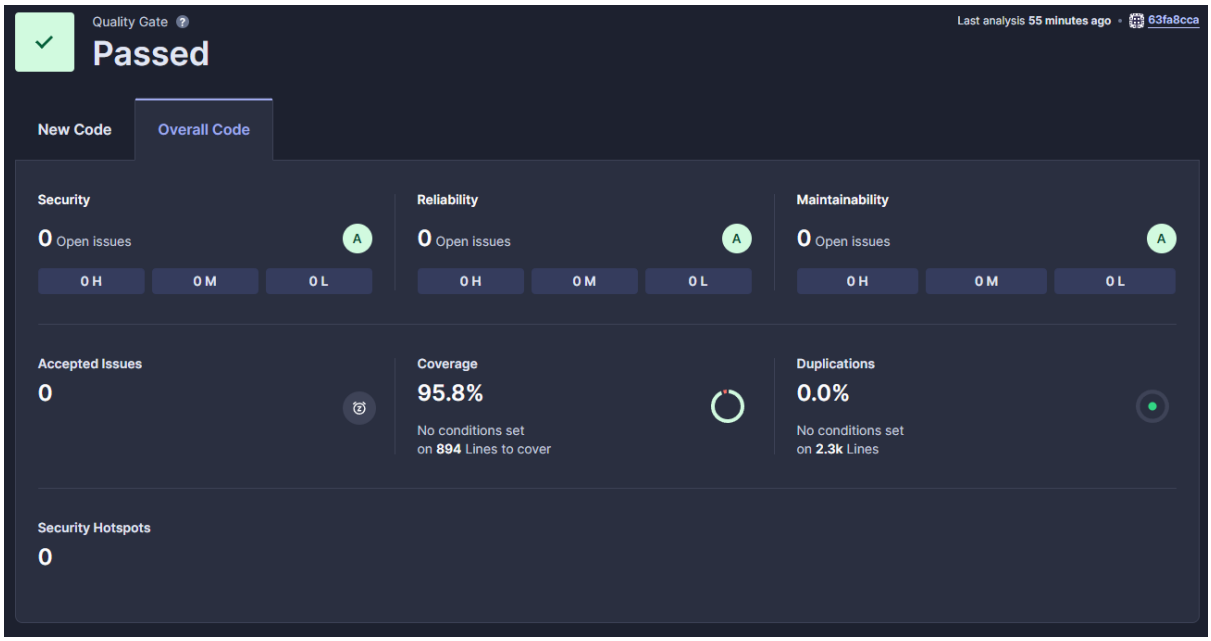


Figura 5.52: *Quality gate de SonarCloud del repositorio de back-end*

Como se puede apreciar, la *quality gate* no determina que exista ningún problema en la aplicación relacionado con seguridad, fiabilidad o mantenibilidad. Además, no es capaz de localizar código duplicado, lo cual es una buena señal en lo referente a la calidad del sistema desarrollado. Por último, tampoco es capaz de encontrar ningún punto caliente (o *hotspot* en inglés) de seguridad, confirmando la seguridad de la aplicación.

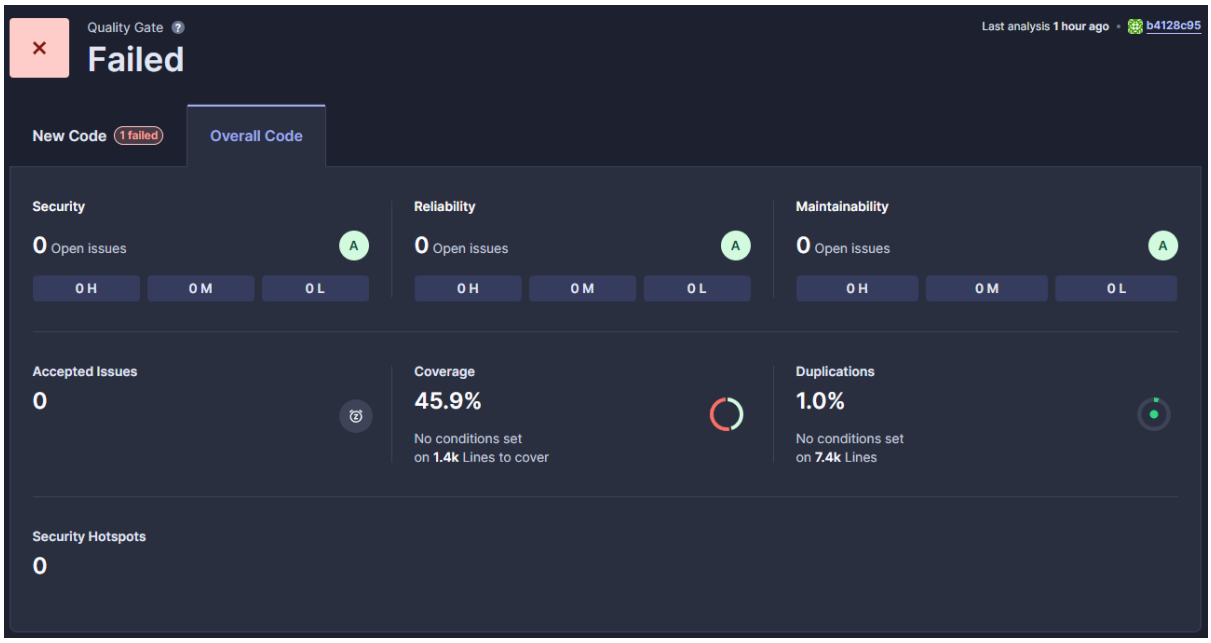


Figura 5.53: *Quality gate de SonarCloud del repositorio de front-end*

En lo que respecta a la *quality gate* del repositorio de front-end, los resultados también son satisfactorios: obviando que no se ha conseguido superar la *quality gate* por no llegar a la cobertura mínima del 80 %, no existe ningún problema en repositorio que deba ser tratado. La única medida que no es del todo satisfactoria es la duplicidad, estando por encima del 0 %. Sin embargo, no es una medida que posea una valor inadecuado, ya que no supera el máximo de 3 %.

Gracias al uso de SonarCloud durante el desarrollo, se han podido localizar errores, fragmentos de código duplicado o puntos calientes de seguridad que han sido resueltos al momento de surgir. Sin embargo, SonarCloud también ha detectado falsos positivos: errores que en realidad no lo eran. A continuación se presenta el caso más significativo, el cual ocurrió en el repositorio de back-end:

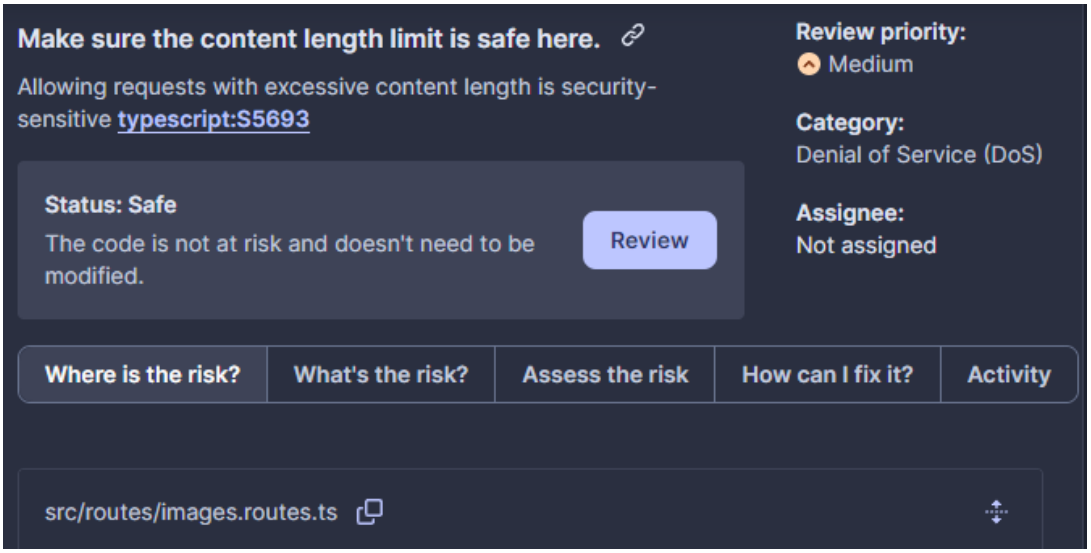


Figura 5.54: *Falso positivo reportado por SonarCloud a la hora de almacenar imágenes*

El código que produce el falso positivo se ha omitido con el objetivo de agilizar y facilitar la lectura, pero aún así es posible comprobar en la parte superior que el error detectado es un punto caliente; en concreto, uno que puede conllevar un ataque Denial of Service (DoS). Este ataque puede suceder, en el caso detectado por SonarCloud, si no se limita el tamaño máximo de los archivos. Aunque en un inicio este punto caliente no era tratado, tras solucionarlo el error se seguía reportando. Tras comprobar manualmente que las imágenes superiores al límite establecido de 8 MBs no se recibían, y que en su lugar ocasionaban un error 400 BAD REQUEST, este punto caliente se estableció como seguro, ya que era un falso positivo.

Capítulo 6. Conclusiones

Tras haber concluido el desarrollo del sistema, se comenzó con la que es la última fase de un proyecto RUP: la transición. Como ya se explicó en el apartado 2.1, esta fase, entre otras cosas, posibilita la extracción de conocimientos para futuros proyectos. Debido a que esta labor es vital para crecer como desarrollador y poder afrontar nuevos proyectos, se ha aprovechado esta fase para establecer las conclusiones obtenidas durante los tres meses de trabajo.

Atendiendo a los objetivos establecidos en el apartado 1.2, es posible apreciar que, en lo que respecta al objetivo general, se ha conseguido implementar una herramienta que ayuda a sus usuarios a estudiar mediante la repetición espaciada, y que además simplifica el intercambio de información mediante características de red social.

Por otro lado, en lo que respecta a los objetivos específicos, se pueden extraer una serie de conclusiones:

- **Punto de vista académico:** Se han conseguido aplicar los conocimientos obtenidos durante el máster para el desarrollo de una aplicación web desde cero, atendiendo a la importancia de una obtención de requisitos y análisis adecuados para conseguir una buena arquitectura.
- **Punto de vista técnico:** Se ha aprendido a utilizar diferentes tecnologías o librerías que no fueron instruidas durante el máster, además de afianzar y expandir los conocimientos de aquello que sí se había estudiado. Por ejemplo, se ha aprendido a utilizar NodeJS para la realización del back-end de una aplicación web, y a realizar pruebas mediante Jest, Jasmine y Stryker Mutator, a la vez que se han afianzado los conocimientos acerca de Angular y la programación reactiva.
- **Punto de vista personal:** Gracias a este TFM, se ha podido crecer profesionalmente, obteniendo conocimientos invaluables para el mercado laboral, especialmente valorando el haber aprendido a trabajar con una metodología de desarrollo como lo es RUP. Además, se ha podido comprobar la importancia del trabajo en equipo a la hora de llevar a cabo un proyecto de este calibre.

La realización de este proyecto ha sido una experiencia enriquecedora, la cual ha conseguido los resultados mostrados gracias también al buen hacer del tutor del proyecto, el cual ha ayudado aportando consejos y conocimientos que han posibilitado la finalización de este proyecto.

Capítulo 7. Líneas Futuras

A pesar de haber conseguido realizar un proyecto de elevada calidad y con una gran cantidad de funcionalidades, siempre existe margen de mejora, y este proyecto no es una excepción.

La primera y más clara línea futura a trabajar consiste en concluir los casos de uso que no han podido ser realizados durante este proyecto, y que engloban todas las interfaces de usuario para los administradores, así como el editar la información del usuario. Además de los casos de uso faltantes, se podrían introducir nuevas funcionalidades para proporcionar mayor utilidad. Por ejemplo, se podría introducir la gamificación que poseen algunas aplicaciones similares, o proporcionar la capacidad de añadir reseñas de mazos y chats entre usuarios, para así aumentar aún más la usabilidad del sistema.

Por otro lado, una línea futura de considerable interés en caso de que este proyecto continuase siendo desarrollado consistiría en encontrar otro método de despliegue continuo para IntelliDeck. Ya que la versión gratuita de Render no proporciona las capacidades necesarias para el despliegue, lo ideal sería buscar alternativas que provean soluciones de más alta calidad para conseguir llevar a cabo dicho despliegue.

Por último, si esta aplicación llegase a desplegarse y comenzase a utilizarse asiduamente, se trataría de externalizar el guardado de las fotos. Actualmente, y como ya se ha explicado durante el desarrollo, las imágenes son almacenadas en el servidor de back-end. La principal alternativa sería utilizar el servicio de almacenamiento en la nube Amazon S3, muy utilizado para guardar imágenes de aplicaciones web de forma segura.

Capítulo 8. Bibliografía

[1] Hermann Ebbinghaus. *Memory: A Contribution to Experimental Psychology*. Teachers College, Columbia University, 1913.

[2] Bo Ae Chun y Hae Ja Heo. «The effect of flipped learning on academic performance as an innovative method for overcoming Ebbinghaus' forgetting curve». En: *Proceedings of the 6th international conference on information and education technology*. 2018, págs. 56-60.

[3] Erwin Oliver Finkenbinder. «The curve of forgetting». En: *The American Journal of Psychology* 24.1 (1913), págs. 8-32.

[4] Sebastian Leitner. *Así se aprende: psicología aplicada del aprender (camino al éxito)*. Trad. por Ambrosio Berasain Villanueva. Herder, 1976. (Pub. orig. como *So lernt man lernen*).

[5] Zirguezi. *Leitner System Alternative*. This file is made available under the Creative Commons CC0 1.0 Universal Public Domain Dedication. To view a copy of this license, visit https://creativecommons.org/publicdomain/zero/1.0/deed.en. 19 de jul. de 2012. URL: https://commons.wikimedia.org/wiki/File:Leitner_system_alternative.svg (visitado 20-04-2024).

[6] Zirguezi. *Leitner System*. This file is made available under the Creative Commons CC0 1.0 Universal Public Domain Dedication. To view a copy of this license, visit https://creativecommons.org/publicdomain/zero/1.0/deed.en. 19 de jul. de 2012. URL: https://commons.wikimedia.org/wiki/File:Leitner_system.svg (visitado 20-04-2024).

[7] Peter C Brown, Henry L Roediger III y Mark A McDaniel. *Make it stick: The science of successful learning*. Harvard University Press, 2014.

[8] Ken Blanchard. *¡Saber y Hacer!* Grupo Editorial Norma, 2008.

[9] Marjan Laal y Seyed Mohammad Ghodsi. «Benefits of collaborative learning». En: *Procedia-social and behavioral sciences* 31 (2012), págs. 486-490.

[10] Simon Kemp. *The time we spend on social media*. 31 de ene. de 2024. URL: https://datareportal.com/reports/digital-2024-deep-dive-the-time-we-spend-on-social-media (visitado 21-04-2024).

[11] Nayan B Ruparelia. «Software development lifecycle models». En: *ACM SIGSOFT Software Engineering Notes* 35.3 (2010), págs. 8-13.

[12] Winston W Royce. «Managing the development of large software systems». En: (1970).

[13] Bill Curtis, Herb Krasner y Neil Iscoe. «A field study of the software design process for large systems». En: *Communications of the ACM* 31.11 (1988), págs. 1268-1287.

[14] S Shylesh. «A study of software development life cycle process models». En: *National Conference on Reinventing Opportunities in Management, IT, and Social Sciences*. 2017, págs. 534-541.

[15] Adel Alshamrani y Abdullah Bahattab. «A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model». En: *International Journal of Computer Science Issues (IJCSI)* 12.1 (2015), pág. 106.

[16] I. Jacobson, G. Booch y J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Object Technology Series. Addison Wesley Professional, 1999.

[17] Kurt Bittner y Ian Spence. *Use case modeling*. Addison-Wesley Professional, 2003.

[18] Jakob Farian Krarup. *Unified Process Model for Iterative Development*. This file is made available under the Creative Commons CC0 1.0 Universal Public Domain Dedication. To view a copy of this license, visit https://creativecommons.org/publicdomain/zero/1.0/deed.en. 21 de ago. de 2020. URL: https://commons.wikimedia.org/wiki/File:Unified_Process_Model_for_Iterative_Development.svg (visitado 14-04-2024).

[19] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley, 2012.

[20] J. Rumbaugh, I. Jacobson y G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley object technology series. Addison-Wesley, 2005.

[21] Craig Larman. *Applying UML and patterns: an introduction to object oriented analysis and design and interative development*. Pearson Education India, 2012.

[22] Grady Booch. *Object solutions: managing the object-oriented project*. Addison Wesley Longman Publishing Co., Inc., 1995.

[23] Grady Booch. *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Addison-Wesley Object Technology Series. CreateSpace Independent Publishing Platform, 2007.

[24] Eric Freeman et al. *Head First Design Patterns*. A brain-friendly guide. O'Reilly Media, Incorporated, 2004.

[25] Trygve Reenskaug, Per Wold, Odd Arild Lehne et al. *Working with objects: the OOram software engineering method*. Citeseer, 1996.

[26] Trygve Reenskaug. «The original MVC reports». En: Department of Informatics, University of Oslo, 10 de dic. de 1979.

[27] Steve McConnell. *Code Complete*. Developer Best Practices. Pearson Education, 2004.

[28] *Sass Basics*. 2024. URL: https://sass-lang.com/guide/ (visitado 27-05-2024).

[29] *What is TypeScript?* 2024. URL: https://www.typescriptlang.org/ (visitado 27-05-2024).

[30] *What is Angular?* 2024. URL: https://angular.dev/overview (visitado 27-05-2024).

[31] *Angular Material*. 2024. URL: https://material.angular.io/ (visitado 27-05-2024).

[32] *About Node.js®*. 2024. URL: https://nodejs.org/en/about (visitado 27-05-2024).

[33] *Express*. 2024. URL: https://expressjs.com/ (visitado 27-05-2024).

[34] *About npm*. 2024. URL: https://docs.npmjs.com/about-npm (visitado 27-05-2024).

[35] *Jest*. 2024. URL: https://jestjs.io/ (visitado 27-05-2024).

[36] *Jasmine*. 2024. URL: https://jasmine.github.io/api/5.1/global (visitado 12-06-2024).

[37] *Stryker Mutator*. 2024. URL: https://stryker-mutator.io/docs/ (visitado 04-06-2024).

[38] *Let's build from here*. 2024. URL: https://github.com/about (visitado 27-05-2024).

[39] *SonarCloud Documentation*. 2024. URL: https://docs.sonarsource.com/sonarcloud/ (visitado 27-05-2024).

[40] *MongoDB Documentation*. 2024. URL: https://www.mongodb.com/docs/ (visitado 27-05-2024).

[41] *Mongoose*. 2024. URL: https://mongoosejs.com/ (visitado 27-05-2024).

[42] Jonathan Knudsen. *Java cryptography*. O'Reilly Media, Inc., 1998.

[43] Marcio Eduardo Delamaro, JC Maidonado y Aditya P. Mathur. «Interface mutation: An approach for integration testing». En: *IEEE transactions on software engineering* 27.3 (2001), págs. 228-247.

[44] Richard A DeMillo, Richard J Lipton y Frederick G Sayward. «Hints on test data selection: Help for the practicing programmer». En: *Computer* 11.4 (1978), págs. 34-41.