Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Aplicación multiplataforma para la gestión de un juego de cartas

Autor:
Óscar Sáez González

Tutor:
Mario Corrales Astorgano

A mi familia y a las personas que han estado cerca de mi durante estos meses.
Muchas gracias por el apoyo.

Agradecimientos

Me gustaría agradecer a mi tutor Mario Corrales Astorgano por facilitarme el desarrollo de este proyecto y adaptarse a mis necesidades durante todo este tiempo. También me gustaría agradecer a mi familia y amigos por la confianza y el apoyo mostrado en todo momento.

Resumen

La inexistencia de aplicaciones enfocadas en la gestión de los escenarios 1.3 del juego de cartas *Marvel Champions* y la existencia de comunidades de jugadores en redes sociales, crea la oportunidad de desarrollar una aplicación para gestionar los escenarios del juego basada en las necesidades de los jugadores.

Este proyecto propone la iniciativa de realizar una aplicación implementando el diseño centrado en el usuario para obtener así las necesidades de los jugadores relacionadas con la gestión de escenarios, y para evaluar la aplicación a través de un test de usabilidad. Además, se propone desarrollar una aplicación multiplataforma desde cero permitiendo así la formación en todas las partes del desarrollo de un proyecto.

Abstract

The lack of applications whose purpose is the management of the scenarios of the Marvel Champions card game and the existence of communities of players on social networks, creates the opportunity of developing an application to manage the game's scenarios based on players' need.

This project suggests the idea of developing an application using user-centered design to meet the players' needs related to the management of scenarios, and to assess implementation through a usability test. Furthermore, it is proposed to develop a multi-platform application from scratch allowing training in all parts of project development.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

Marvel Champions [1] es un juego de cartas cooperativo que permite jugar con los diferentes superhéroes del universo *Marvel* para enfrentarse a diferentes villanos. Para ello, los jugadores deben crearse mazos de cartas personalizados. Los villanos tienen sus propios mazos, que también son personalizables.

Además, este juego es un *Living Card Game* (LGC), que significa que periódicamente se sacan al mercado nuevas expansiones con nuevas cartas, tanto para los superhéroes como para los villanos. Por lo tanto, las posibilidades de personalización de los mazos se van incrementando con el tiempo.

Por otro lado, al igual que pasa con muchos juegos, existen varias comunidades de jugadores en redes socilaes, que permiten intercambiar opiniones, estrategias o cualquier aspecto del juego con otras personas afines. De estas comunidades ha surgido multitud de contenido, como canales de youtube, podcast o herramientas informáticas, centradas en diferentes aspectos del juego.

Teniendo todo esto en cuenta, y analizando las aplicaciones existentes relacionadas con el juego, se observa que existen multitud de aplicaciones para la gestión de mazos de los superhéroes, pero ninguna para la gestión de los escenarios 1.3 del juego, incluyendo los mazos de los villanos, de lo cual surge la idea de este proyecto.

1.2. Motivación

Del contexto previo surge la idea de realizar una aplicación para gestionar los escenarios del juego. El acceso a la comunidad del juego permite aplicar el diseño centrado en el usuario para extraer las necesidades de los jugadores relacionadas con la gestión de escenarios, y para evaluar una versión inicial de la aplicación a través de test de usabilidad.

El desarrollo de una aplicación multiplataforma utilizando tecnologías actuales que permita al estudiante formarse en todas las partes del desarrollo de un proyecto, incluyendo interfaz de usuario (UI), desarrollo de API y gestión de bases de datos.

1.3. Terminología

En esta sección se explican las diferentes terminologías utilizadas en el juego para aclarar posibles confusiones.

- Escenario: Configuración específica y temática en la que los jugadores enfrentan desafíos. Cada escenario tiene sus propias reglas, condiciones de victoria y desafíos únicos que los jugadores deben superar trabajando como superhéroes. Un escenario incluirá un villano principal que los jugadores deben derrotar, así como una serie de esbirros, eventos y condiciones especiales que complican el desafío.
- Conjunto modular: Componente adicional que se puede agregar al escenario base para aumentar su dificultad o variabilidad. Estos módulos pueden incluir cartas de villanos adicionales, esbirros, eventos especiales o cualquier otro elemento que agregue complejidad al juego.
- Rasgo: Característica específica o atributo que una carta posee y que puede influir en cómo se juega esa carta o cómo interactúa con otras cartas en el juego.
- Campaña: forma de jugar una serie de escenarios interconectados que cuentan una historia continua. El/los jugador/es enfrenta/n varios escenarios que pueden incluir diferentes villanos, secuencias de eventos y desafíos. Cada escenario puede tener sus propias condiciones de victoria y derrota, y el éxito o fracaso en un escenario puede tener consecuencias en los siguientes escenarios de la campaña.
- Aumento: en el caso de los escenarios o conjuntos modulares, son cartas que tienen efectos negativos que se aplican a los héroes que se revelan durante la fase de encuentro.
- Esbirro: carta que representa un secuaz o una criatura que el villano puede desplegar para ayudarlo en su lucha contra los héroes. Estos esbirros suelen ser cartas de juego con una habilidad y estadísticas propias que los hacen representar una amenaza adicional para los héroes.
- Plan secundario: mecánica que representan objetivos alternativos o eventos secundarios que los héroes deben enfrentar además del objetivo principal de derrotar al villano.

1.4. Estado de la cuestión

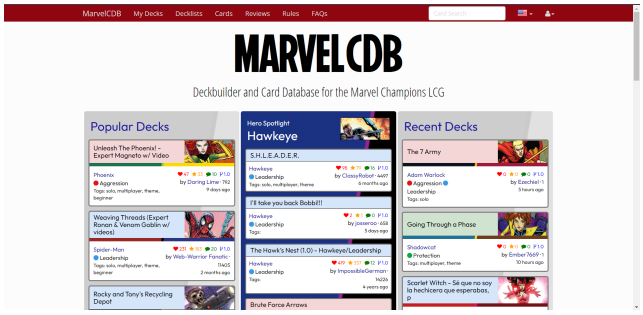
Actualmente, existen diferentes aplicaciones para crear, guardar y compartir mazos de héroes, además de buscar cartas por nombre o por diferentes campos de esta. Sin embargo, no existe aún ninguna

aplicación destinada a la gestión de escenarios, incluyendo creación, guardado y distribución de estos, así como el historial de partidas del usuario y de otros usuarios.

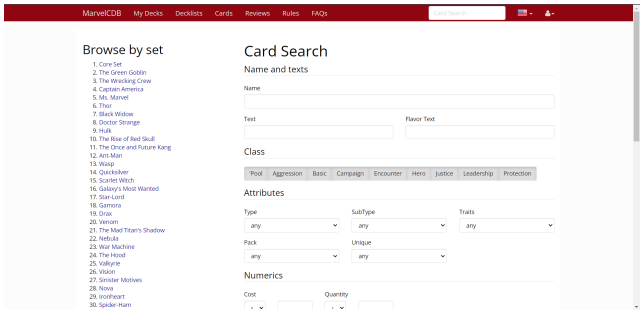
A continuación, se muestran una serie de aplicaciones relacionadas de las cuales se ha realizado una búsqueda.

1.4.1. MarvelCDB

La aplicación web MarvelCDB [2] está destinada a la gestión y construcción de mazos de héroes. Los usuarios pueden compartir sus mazos y buscar mazos creados por otros usuarios, además de dejar comentarios y reseñas sobre las cartas y los mazos, compartiendo así opiniones con la comunidad. También ofrece herramientas para analizar las estadísticas de los mazos, como el costo promedio de las cartas, la distribución de las cartas por tipo, etc.



(a) Pantalla principal de MarvelCDB



(b) Pantalla de búsqueda de cartas de MarvelCDB

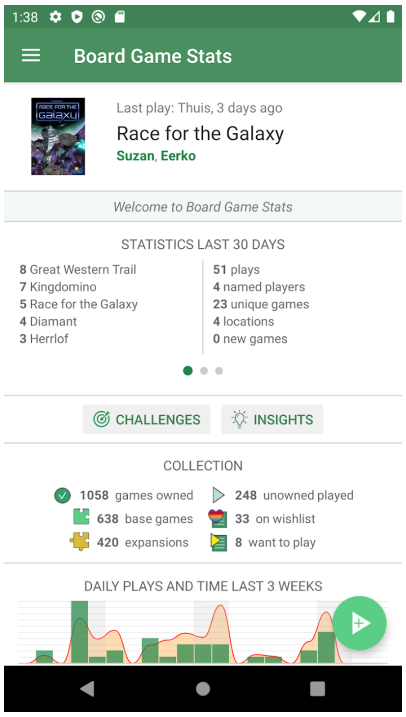
Figura 1.1: MarvelCDB

1.4.2. Board Game Stats

La aplicación móvil Board Game Stats (BG Stats) [3] es una herramienta diseñada para aquellos usuarios de juegos de mesa que desean realizar un seguimiento detallado de sus sesiones de juego, estadísiticas y colecciones. Esta incluye registro de sesiones de juego, ofrecer estadísticas detalladas sobre los juegos del usuario como número de partidas jugadas, duración promedio de los juegos, etc. Así como generar gráficos y tablas que visualizan tus hábitos de juego.

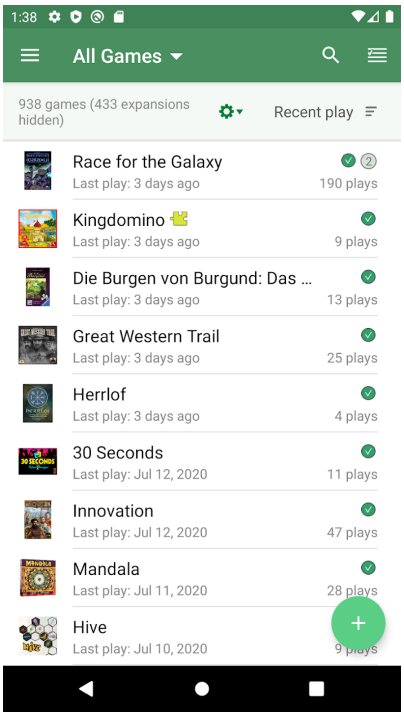
1.4.3. Marvel Champions Tracker

Marvel Champions Tracker [4] es una aplicación web diseñada para ayudar a los usuarios del juego *Marvel Champions* a gestionar y llevar un seguimiento de sus partidas, registrando los puntos de vida y estado de los héroes y villanos durante una partida. También facilita el seguimiento de las cartas de encuentro, amenazas en los planes secundarios y otros elementos del juego. Y, al igual que la aplicación anterior, proporciona estadísticas sobre las partidas jugadas, pero esta vez centrándose en estadísticas concretas de *Marvel Champions*, como héroes más utilizados, villanos más desafiantes, etc.



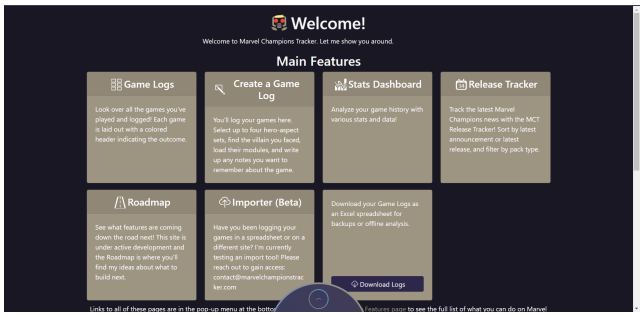
(a) Pantalla principal de BG Stats

(b) Pantalla de estadísticas de juego de BG Stats

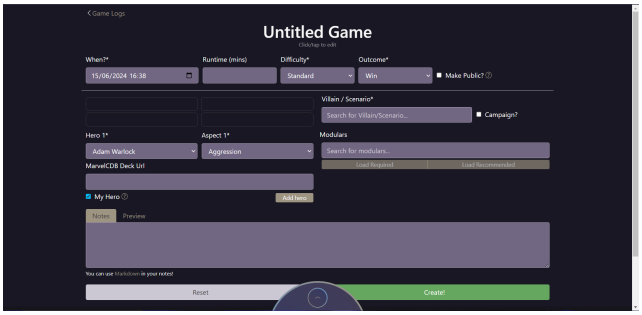


(c) Pantalla de juegos de BG Stats

Figura 1.2: Board Game Stats



(a) Pantalla principal de Marvel Champions Tracker



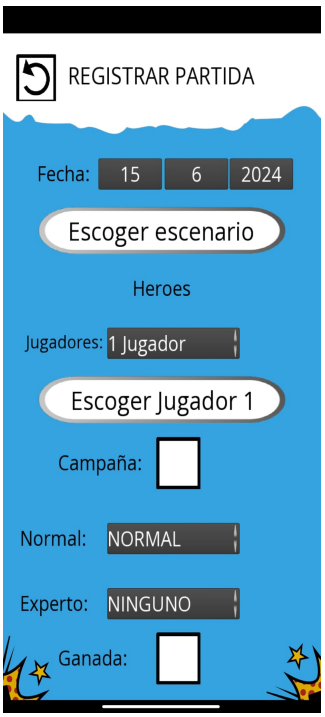
(b) Pantalla de registro de partidas de Marvel Champions Tracker

Figura 1.3: Marvel Champions Tracker

1.4.4. M Champions LCG Statistics

Esta aplicación móvil [5] está diseñada para los usuarios del juego *Marvel Champions* y su propósito principal es ayudar a estos a registrar, analizar y gestionar sus partidas y estadísticas del juego. Permite a los usuarios registrar detalles de cada partida, incluyendo el héroe utilizado, el villano enfrentado, la configuración del escenario y los resultados. Al igual que las aplicaciones anteriores, ofrece estadísticas de las partidas registradas, mostrando héroes más utilizados, tasas de victoria, etc.

(a) Pantalla principal de M Champions LCG Statistics



(b) Pantalla de registro de partida de M Champions LCG Statistics

Figura 1.4: M Champions LCG Statistics

1.5. Tecnologías empleadas

El desarrollo incluye front-end y back-end, por lo que en esta sección se describen las diferentes tecnologías utilizadas, así como la justificación para el uso de las mismas.

1.5.1. Flutter

Dado que se trata de una aplicación móvil/web se ha optado por utilizar una tecnología híbrida para el front-end que es Flutter [6]. Tras varias comparativas con otras tecnologías híbridas como son React Native [7] o Kotlin Multiplatform [8], se ha decidido utilizar Flutter debido a que el tiempo de desarrollo con esta tecnología es más rápido que con ambas anteriores gracias al *"hot reload"*, que permite hacer cambios en el código y ver los resultados instantaneamente. Sin embargo, React Native necesita recargar la app completa cada vez que se quiera probar algo nuevo, lo que ralentiza el desarrollo. Al igual que Kotlin Multiplatform que necesita compilar la app para observar un nuevo cambio.

Flutter ofrece una extensa documentación y herramientas de apoyo, además de una amplia comunidad, haciendo que sea fácil de entender para el desarrollador. Mientras que la comunidad de Kotlin Multiplatform no es muy extensa debido a que es una tecnología muy reciente, y las herramientas de soporte que ofrece React Native son limitadas en comparación con Flutter.

Figura 1.5: Logo de Flutter

1.5.2. Node.js

Para el back-end se ha decidido usar Node.js [9] debido a su facilidad de aprendizaje, ya que JavaScript [10] es un lenguaje que ya se ha utilizado en algunas asignaturas y se está familiarizado con él. Además, como Node.js es muy popular, tiene una extensa comunidad, lo que hace más fácil resolver errores.

También aporta una alta velocidad a la aplicación gracias al motor V8 de Google que utiliza para compilar y ejecutar código JavaScript. Al ser JavaScript un lenguaje de programación de un solo hilo, Node.js utiliza un modelo de eventos que permite gestionar múltiples conexiones simultáneas sin bloquear el proceso principal. Esto aporta un mayor rendimiento y eficiencia en la gestión de conexiones, además de una mayor escalabilidad para la aplicación [11].

Figura 1.6: Logo de Node.js

1.5.3. MySQL

La razón principal por la que se ha elegido MySQL [12] como gestor de bases de datos es porque ofrece un alto grado de control dobre la estructura de datos y las consultas. Además, ya se tiene experiencia utilizando este gestor y SQL en general, por lo que es lo más cómodo.

MySQL es uno de los gestores de bases de datos relacionales más populares y utilizados. Esto implica que hay una gran cantidad de recursos, documentación y comunidad de usuarios dispuestos a ayudar. También dispone de un alto rendimiento para una gran variedad de cargas de trabajo.

Figura 1.7: Logo de MySQL

1.5.4. Docker

Para desplegar ambos proyectos, front-end y back-end se ha utilizado Docker [13] ya que aporta rapidez en el despliegue y consistencia, porque un contenedor Docker contiene todo lo necesario para ejecutar una aplicación. Además, permite aislar aplicaciones en contenedores independientes, es decir, cada contenedor tiene su propio entorno. Esto reduce los conflictos entre aplicaciones que dependen de diferentes versiones de las mismas bibliotecas o herramientas. También aporta mayor eficiencia en cuanto a recursos ya que los contenedores son más ligeros en términos de recursos que las máquinas virtuales tradicionales.

Figura 1.8: Logo de Docker

1.6. Objetivos

- Construir una aplicación multiplataforma desde cero que permita gestionar los escenarios del juego *Marvel Champions*.
- Conocer y emplear las metodologías centradas en el usuario, incluyendo el análisis de usuarios y sus necesidades y la evaluación con usuarios reales.
- Aprender las particularidades del desarrollo de un proyecto *full-stack*, utilizando tecnologías emergentes como *Flutter y Node.js*.

1.7. Metodología

El diseño centrado en el usuario es un proceo iterativo que se enfoca en comprender a los usuarios y sus necesidades en todas las fases de diseño y desarrollo [14]. En este proceso existen cuatro fases clave [15]:

1. **Comprender a los usuarios**: fase en la que se investiga y comprende el entorno y las circunstancias en las que los usuarios interactúan con un sistema.
2. **Especificar las necesidades del usuario**: se definen las necesidades de los usuarios, estableciendo los objetivos de diseño.
3. **Diseñar las soluciones**: se desarrollan propuestas de diseño que atiendan las necesidades identificadas, teniendo en cuenta la usabilidad y la accesibilidad.
4. **Evaluar los resultados**: se prueban los diseños realizados con usuarios reales para obtener retroalimentación y mejorar continuamente el producto.

Para el desarrollo del proyecto, se ha optado por una metodología iterativa e incremental en consecuencia de la naturaleza y desarrollo del proyecto, que es realizado por un único programador.

Esta metodología nos permite tener una aplicación con funcionalidad tras cada iteración, es ideal para el tipo de proyecto planteado. Esta metodología permite la posibilidad de revisar el cumplimiento de las especificaciones con el cliente (en este caso los usuarios) y de añadir nuevos requerimientos si fueran necesarios. Evitando así realizar cosas que no habría que hacer.

En la Figura 1.9 se muestran los pasos que se siguen durante un desarrollo iterativo e incremental. En el Capítulo 3 se abordará en detalle el trabajo llevado a cabo en cada iteración del proyecto.

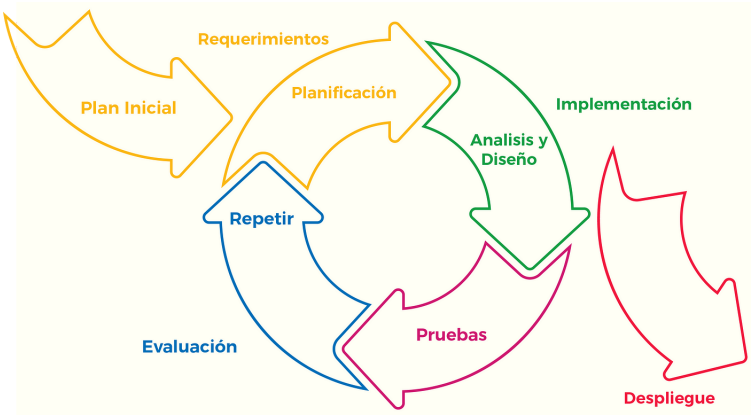


Figura 1.9: Metodología iterativa e incremental [16]

1.8. Estructura de la memoria

- **Introducción**: Se presenta globalmente el proyecto, explicando el contexto del proyecto, las motivaciones para realizarlo, el estado de la cuestión, los objetivos a lograr durante la realización del proyecto y la metodología usada para desarrollarlo.
- **Planificación**: Se presenta la organización temporal con la estimación del tiempo para cada iteración del proyecto, sus riesgos, costes y el entorno tecnológico utilizado para su desarrollo. Por último, se incluye como ha sido finalmente la planificación del proyecto y una comparación con la planificación estimada.

- **Descripción de las iteraciones**: Explicación del desarrollo del proyecto en cada una de las iteraciones.
- **Estado final de la aplicación**: Descripción del análisis y el diseño del proyecto, además de su estructuración. Incluye sus correspondientes diagramas.
- **Pruebas**: Contiene la descripción y resultado de las pruebas realizadas a la aplicación.
- **Diseño centrado en el usuario**: Descripción de todo el proceso aplicado de diseño centrado en el usuario desde el análisis de las necesidades de los usuarios hasta la realización de las pruebas de usabilidad.
- **Conclusiones**: Contiene las conclusiones obtenidas tras la realización del proyecto.
- **Apéndices**: Documentación adicional al proyecto en la que se encuentran los acrónimos, el cuestionario inicial, el cuestionario de evaluación, y los correspondientes manuales de despliegue y de uso de la aplicación.
- **Bibliografía**: Referencias bibliográficas utilizadas en el proyecto.

Capítulo 2

Planificación

2.1. Planificación inicial

En esta sección se detalla la planificación tanto temporal como de trabajo estimado según las iteraciones descritas en la sección de Metodología 1.7.

Para cumplir con las 300 horas que se atribuyen al Trabajo de Fin de Grado en la correspondiente guía docente, se ha estimado un trabajo de 5 días semanales donde se dediquen 4 horas diarias. Además, en caso de necesitarse, se podría ampliar este trabajo a los fines de semana.

Las fechas de comienzo y finalización del proyecto son respectivamente el 6 de febrero de 2024 y el 3 de junio de 2024, que corresponde con un periodo de aproximadamente 4 meses. Teniendo en cuenta que se trabaja en el proyecto 4 horas al día de lunes a viernes durante este periodo de tiempo se obtiene un total de 340 horas de trabajo.

Para el desarrollo del proyecto se utilizará una metodología iterativa e incremental, en la que se realizarán las siguientes iteraciones:

- Primera iteración: Fase de inicio del proyecto, que comienza el 6 de febrero de 2024 y se estima una duración de 4 semanas. Se tratará de realizar la planificación del proyecto, de analizar la API a utilizar, de realizar la encuesta para los usuarios y obtener sus correspondientes respuestas. Tras haber obtenido las respuestas, analizarlas, crear en base a estas respuestas el usuario ideal, comenzar el modelo de dominio, y por último, empezar con la creación del proyecto.

- Segunda iteración: Comenzará el 4 de marzo de 2024 y tendrá una duración estimada de 5 semanas. Se llevará a cabo parte de la funcionalidad, en concreto el registro e inicio de sesion de usuario, y la posibilidad de añadir, actualizar, eliminar y obtener diferentes configuraciones de partidas.

- Tercera iteración: Comenzará el 8 de abril de 2024 y tendrá una duración estimada de 5 semanas. Se desarrollarán las funcionalidades de dar *like* a una partida, guardar una partida en la lista de

favoritos, valorar la dificultad/diversión de una partida, y por último, buscar partidas y filtrarlas por campos. En caso de tener tiempo suficiente, se desarrollará la gestión de campañas 1.3.

- Cuarta iteración: En esta última fase, se realizarán las pruebas finales, entre ellas el test de usabilidad con usuarios auténticos, y se completará el trabajo con el fin de preparar la entrega del proyecto. Comenzará el 13 de mayo de 2024 y su duración estimada es de 3 semanas. Esta fase empleará menos tiempo, ya que esta es una fase de revisión y ya se habrá dedicado tiempo en las fases anteriores.

En la Figura 2.1 se muestra la gráfica de la distribución de semanas por iteración, incluyendo las semanas acumuladas en cada una de las iteraciones. Podemos apreciar que la duración de las iteraciones es similar, ya que se han estimado con una variación de una sola semana, y en el caso de la iteración 2 y 3 no hay variación.

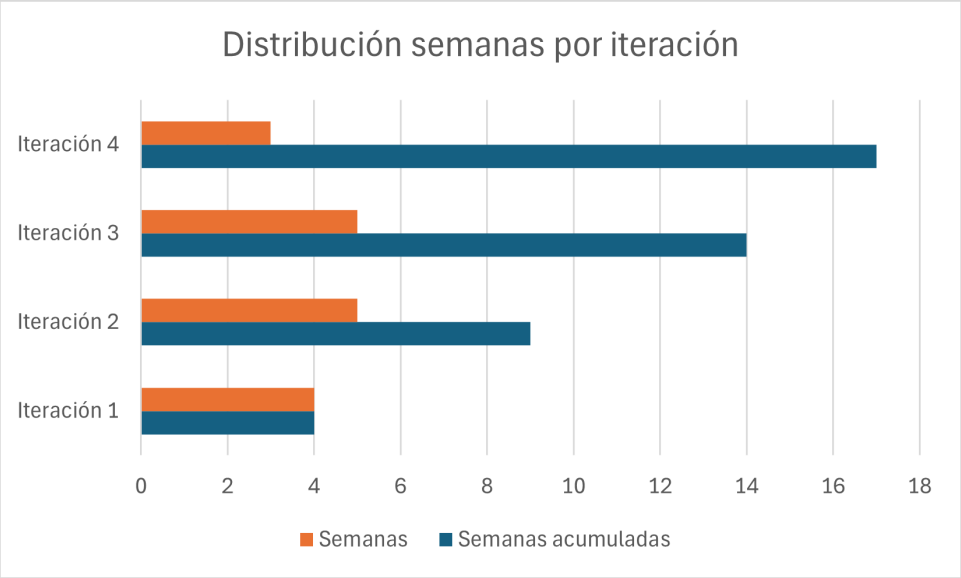


Figura 2.1: Distribución semanas por iteración

En la gráfica de la Figura 2.2 se muestra también la distribución por iteración, pero en este caso de las horas, incluyendo las horas acumuladas en cada iteración.

2.2. Entorno de trabajo

En esta sección se describen las herramientas del entorno de trabajo, además de su preparación y configuración.

Para el desarrollo del proyecto se ha utilizado un portátil Asus TUF Gaming FX505DT-BQ051 [17] (ver Tabla 2.1) con sistema operativo Windows 10 y un móvil Xiaomi Redmi Note 11 Pro [18] (ver Tabla 2.2) con Android 13 (API 33 [19]).

Para la preparación del entorno de desarrollo se ha seguido la documentación oficial de Android

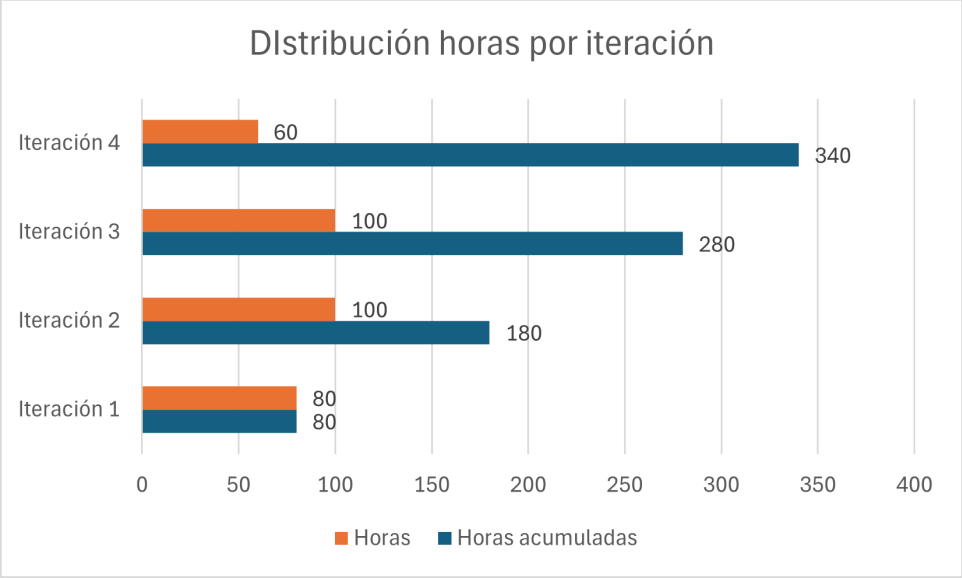


Figura 2.2: Distribución horas por iteración

Asus TUF Gaming FX505DT-BQ051	
Procesador	AMD Ryzen 5 3550H
Memoria	8 GB
Disco SSD	512 GB
Tarjeta gráfica	Nvidia Geforce GTX 1650
Resolución	1920 x 1080
Pulgadas	15.6
Sistema operativo	Windows 10
Arquitectura del sistema	64 bits

Tabla 2.1: Especificaciones del portátil de trabajo

Studio [20] y se han descargado los instaladores tanto de Visual Studio Code [21] como de MySQL [22] para su posterior instalación en el sistema.

El software utilizado para el desarrollo del proyecto se define a continuación:

- **Android Studio**[23]: Entorno de desarrollo integrado oficial para la plataforma Android desarrollado por Google. Utilizado para el desarrollo del front-end en Flutter.
- **Astah Professional**[24]: Herramienta de modelado UML utilizada para la realización del Modelo de Dominio y diagrama Entidad-Relación. Utilizado bajo la licencia de la UVa.
- **Chrome**[25]: Navegador web de código cerrado desarrollado por Google. Utilizado para diversas funciones.
- **GitLab**[26]: Forja para alojar proyectos utilizando el sistema de control de versiones Git [27]. Utilizado para la gestión de control de versiones del proyecto.

Xiaomi Redmi Note 11 Pro	
Procesador	Helio G96 Octa-core Max 2.05GHz
Memoria	8 GB
Almacenamiento	128 GB
Resolución	1080 x 2400 (Full HD+)
Pulgadas	6.67
Sistema operativo	Android 13 (Tiramisú)
Batería	5000 mAh

Tabla 2.2: Especificaciones del smartphone de trabajo

- **Microsoft Excel**: Programa que permite editar hojas de cálculo desarrollada por Microsoft para Windows, macOS, Android y iOS. Utilizado para la distribución de horas y semanas del proyecto y para el análisis de las respuestas de los usuarios al cuestionario de evaluación.
- **Microsoft Teams**: Plataforma unificada de comunicación y colaboración que combina chat, videollamadas, almacenamiento de archivos e integración de aplicaciones. Utilizada para la comunicación con el tutor.
- **MySQL**[12]: Sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation. Utilizado para la gestión de la base de datos de la aplicación.
- **Overleaf**[28]: Editor colaborativo de LaTeX basado en la nube que se utiliza para escribir, editar y publicar documentos científicos. Utilizado para escribir la memoria.
- **Postman**[29]: Aplicación americana que permite probar API web. Utilizada para probar la API de la aplicación.
- **Visual Studio Code**[30]: Editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Utilizado para el desarrollo del back-end en Node.js.

2.3. Análisis de riesgos

Todo proyecto tiene un grado de incertidumbre sobre el éxito o fracaso del mismo. En esta sección se presenta el análisis de los posibles riesgos del proyecto (ver Tabla 2.3). Esta tabla contiene el nombre del riesgo, una breve descripción, la amenaza que tiene sobre el proyecto, la probabilidad de que ocurra y el impacto que tendría en caso de materializarse.

ID	Nombre	Descripción	Amenaza	Probabilidad	Impacto
1	Equipo pequeño	Solo una persona realizando el Trabajo de Fin de Grado	Retraso en la fecha de finalización del proyecto	Baja	Medio
2	Enfermo	Al ser trabajo de una única persona, en caso de estar enfermo el proyecto se queda pausado	Retraso en los tiempos de desarrollo del proyecto	Media	Medio
3	Falta de experiencia	La falta de experiencia con alguna tecnología nueva	Emplear más tiempo en el desarrollo del proyecto	Media	Bajo
4	Tutor ocupado	El tutor tiene más cosas que hacer y no tiene el tiempo adecuado para atender el Trabajo de Fin de Grado. O se encuentra enfermo/-de vacaciones	No acabar el proyecto a tiempo	Baja	Alto
5	Cambio de los requisitos	Los requisitos cambian después de la implementación	Tiempo perdido y recursos en trabajo sin utilidad	Media	Alto
6	Problemas o incapacidad para implementar	Si hubiera alguna funcionalidad difícil o inviable	Pérdida de tiempo y retraso del proyecto	Media	Medio
7	Asignaturas pendientes	Asignaturas pendientes además del Trabajo de Fin de Grado	En caso de no aprobarlas, no poder presentar el trabajo el mismo año	Baja	Alto

Tabla 2.3: Análisis de riesgos

A continuación se muestra la tabla en la que aparece el plan de acción de los riesgos descritos en el análisis (ver Tabla 2.4). Esta tabla contiene el nombre del riesgo, el plan de reducción (prevención del riesgo), el plan de miitgación (en caso de materializarse dicho riesgo) y el apetito/tolerancia en caso de ocurrir (asimilar el riesgo).

ID	Nombre	Plan de reducción	Plan de mitigación	Apetito
1	Equipo pequeño	Empezar el Trabajo de Fin de Grado lo antes posible	Acelerar/cambiar la metodología del desarrollo del proyecto	Alto
2	Enfermo	Tener cuidado con los contagios y con los resfriados	No trabajar mientras estoy enfermo y descansar al máximo. Al recuperarse, recuperar las horas perdidas	Alto
3	Falta de experiencia	Invertir tiempo libre en familiarizarse con las tecnologías en lugar de hacerlo durante el desarrollo	Dedicar horas extra para cubrir la falta de experiencia	Alto
4	Tutor ocupado	Escoger un tutor responsable y dejar hablado este posible tema	Avanzar solo el trabajo hasta que el tutor pueda atender	Medio
5	Cambio de los requisitos	Realizar el trabajo según la planificación y por orden de importancia, así como tener buena comunicación con los tutores y los usuarios para asegurarse	Invertir horas extra	Medio
6	Problemas o incapacidad para implementar	Informarse cuanto antes de las funcionalidades a desarrollar y su dificultad	Realizar horas extra	Medio
7	Asignaturas pendientes	Realizar las demás asignaturas antes de comenzar el Trabajo de Fin de Grado	Presentar el trabajo otro año	Bajo

Tabla 2.4: Plan de acción

2.4. Estimación de costes

En esta sección se muestra la estimación de costes del proyecto. Los costes se podrían dividir en 4 tipos diferentes: costes de hardware, coste de recursos humanos, costes del espacio de trabajo y costes de licencias. Los costes de hardware se consideran como coste amortizado, ya que se disponía de ellos de antemano. El coste amortizado se calcula dividiendo las horas de uso del material por sus horas de vida total y multiplicando por su coste original, como se muestra en la siguiente fórmula:

$$coste\ amortizado = \frac{horas\ de\ uso}{horas\ de\ vida\ totales} \times coste\ original \tag{2.1}$$

Se sabe que el proyecto debe durar 300 horas, pero podemos prever hasta un 13 % más de este tiempo como margen, es decir, 340 horas. El portátil costó 800 € y fue comprado hace 4 años aproximadamente. Por lo tanto, si pasamos esos 4 años a horas, el coste amortizado sería el siguiente:

$$\left(\frac{340}{4 \times 8760}\right) \times 800\ € = 7.77\ € \tag{2.2}$$

El cálculo del coste amortizado del móvil sabiendo que tiene aproximadamente 2 años de antigüedad, un coste original de 220€, y suponiendo un 80 % de uso en el proyecto se muestra a continuación:

$$\left(\frac{272}{2 \times 8760}\right) \times 220\ € = 3.41\ € \tag{2.3}$$

En la Tabla 2.5 se muestran los dispositivos hardware utilizados para la realización del proyecto, y su correspondiente coste original y coste amortizado.

Dispositivo	**Coste amortizado**	**Coste**
Asus TUF Gaming FX505DT-BQ051	7.77 €	800 €
Xiaomi Redmi Note 11 Pro	3.41 €	220 €
Coste total	11.18 €	1020 €

Tabla 2.5: Coste de hardware

Respecto a los costes de recursos humanos, el sueldo medio de un desarrollador software junior en España es de 12.09 € la hora [31], si el tiempo estimado del proyecto son 340 horas, el coste de recursos humanos es de 4110.6€.

En cuanto al coste del espacio de trabajo, incluye luz e internet, ya que se trabaja desde el domicilio personal. Estos gastos son de media al mes 40€, teniendo en cuenta que son aproximadamente 4 meses de proyecto, el coste de espacio de trabajo sale a 160€.

Por último, el coste de licencias sería de 40€, ya que la única licencia relevante es la de Astah Professional que cuesta 10 € al mes [32]. Dado que el proyecto es de 4 meses el precio total es de 40 €, pero esta licencia la paga la UVa.

En la Tabla 2.6 se muestra el coste total del proyecto y el coste real en caso de no haber dispuesto de ningún equipo de antemano.

Tipo de gasto	Coste	Coste desde cero
Hardware	11.18 €	1020 €
Recursos humanos	4110.6 €	4110.6 €
Espacio de trabajo	160 €	160 €
Licencias	0 €	40 €
Coste total	4281.78 €	5330.6 €

Tabla 2.6: Costes del proyecto

2.5. Planificación final

En esta sección se describe como ha resultado la planificación tras la realización del proyecto y como ha variado de la planificación inicial de la Sección 2.1, comentando las diversas modificaciones ocurridas en las iteraciones y la comparación de las horas y semanas estimadas y las realmente empleadas.

La iteraciones han sufrido diferentes cambios debido a que no se anticipó el tiempo necesario para implementar ciertas funcionalidades de la aplicación, así como algunos diseños de pantallas. Esto, también ha causado la variación del tiempo realmente empleado para cada iteración respecto del estimado. A continuación, se explican los diferentes cambios en la estructura de las iteraciones así como las horas y semanas realmente empleadas para su desarrollo.

Las iteraciones que más cambios han sufrido, en cuanto a tareas realizadas, han sido las iteraciones 2, 3 y 4. Inicialmente, se planificó que la iteración 2 correspondería con la realización de las funcionalidades de registro e inicio de sesión, y creación, actualización, eliminación y obtención de una partida. Sin embargo, en esta iteración, solo se realizó la del inicio de sesión, la del registro de usuario, y la de creación de una partida, ya que, de esta última, tanto el diseño de la interfaz de usuario como el desarrollo de la funcionalidad llevó un tiempo considerable. También, algunos contratiempos durante esta iteración ocasionaron un retraso en el cronograma previsto, entre ellos, la necesidad de hacer un *script* para volcar la información necesaria de la API de MarvelCDB y el cambio realizado en el proyecto front-end de utilizar el patrón BLoC a emplear el paquete GetX (estos contratiempos se explicarán más detalladamente en la descripción de esta iteración 3.2).

En la iteración 3 también hubo cambios considerables, se esperaba desarrollar las funcionalidades de dar *like* a una partida, guardar una partida en favoritos, valorar la dificultad/diversión de una partida, y buscar partidas y filtrarlas por campos. Sin embargo, dado que no se habían completado las tareas propuestas para la segunda iteración, no todas las funcionalidades propuestas para esta iteración fueron realizadas. También, hubo numerosos contratiempos debido a la falta de claridad sobre los detalles que debían mostrarse de las partidas, lo cual se fue ampliando en cada reunión con el tutor.

Dado que toda la funcionalidad de la aplicación aún no se había desarrollado al acabar la tercera iteración, parte de esta se llevó a cabo durante la cuarta y última iteración, además de realizar las debidas pruebas descritas en el Capítulo 5 y en la Sección 6.2. La funcionalidad realizada en esta

iteración es aquella relacionada con la pantalla de favoritos, con la del perfil del usuario y con la de búsqueda de partidas. Todo esto hizo que la iteración se alargara más de lo esperado, como se puede ver más adelante en las gráficas que muestran las horas y semanas empleadas en las iteraciones.

A lo largo de estas iteraciones se han ido teniendo reuniones cada tres semanas, con algunas excepciones, como en Semana Santa, que el tiempo entre reuniones se expandió, y durante la tercera y cuarta iteración las reuniones se realizaron con mayor frecuencia. En total han sido nueve reuniones. No son muchas reuniones, pero no se han necesitado más, ya que además de estas reuniones, ha habido comunicación a través de Microsoft Teams.

Estas modificaciones en las iteraciones han implicado un cambio en el número de horas que se han dedicado a estas, y en el caso de la cuarta iteración, también en el número de semanas. La primera iteración no ha sufrido ninguna modificación respecto a tareas realizadas en ella, pero si respecto al tiempo empleado en esta ya que en vez de haber sido de 80 horas ha sido de 60.

Como se puede contemplar, en las dos primeras iteraciones no se dedicó el tiempo necesario para realizar lo que se tenía planeado inicialmente, por lo que, en las dos iteraciones posteriores se tuvieron que realizar más horas de las planificadas y realizar tareas planificadas para las dos primeras iteraciones. Todo esto se podría considerar un riesgo en el plazo de entrega del proyecto. Sin embargo, como se planificó al inicio que la cuarta y última iteración finalizara la primera semana de junio, se dejaron dos semanas de holgura al final del proyecto en caso de ir escaso de tiempo debido a las otras asignaturas que se realizan durante el segundo cuatrimestre de cuarto. Gracias a ello, estas dos semanas se pudieron utilizar para poder acabar el proyecto a tiempo.

Todo esto se puede observar en las Figuras 2.3 y 2.4 que muestran la comparación de semanas y horas empleadas realmente con las estimadas en la planificación inicial respectivamente. A pesar de no haber realizado las horas planificadas en las dos primeras iteraciones, estas horas han sido compensadas durante las iteraciones tres y cuatro, y por ello, tras la finalización de las cuatro iteraciones el número de horas acumuladas empleadas realmente y de horas acumuladas que se estimaron son muy similares.

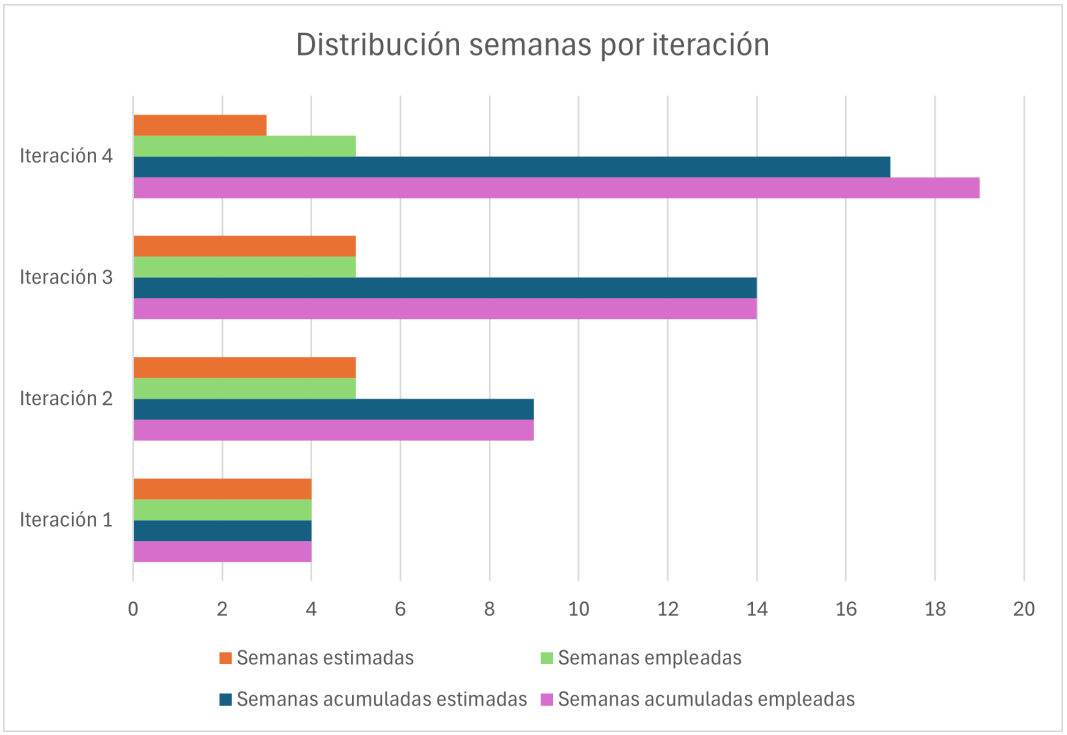


Figura 2.3: Comparación del número de semanas estimadas y reales por cada iteración

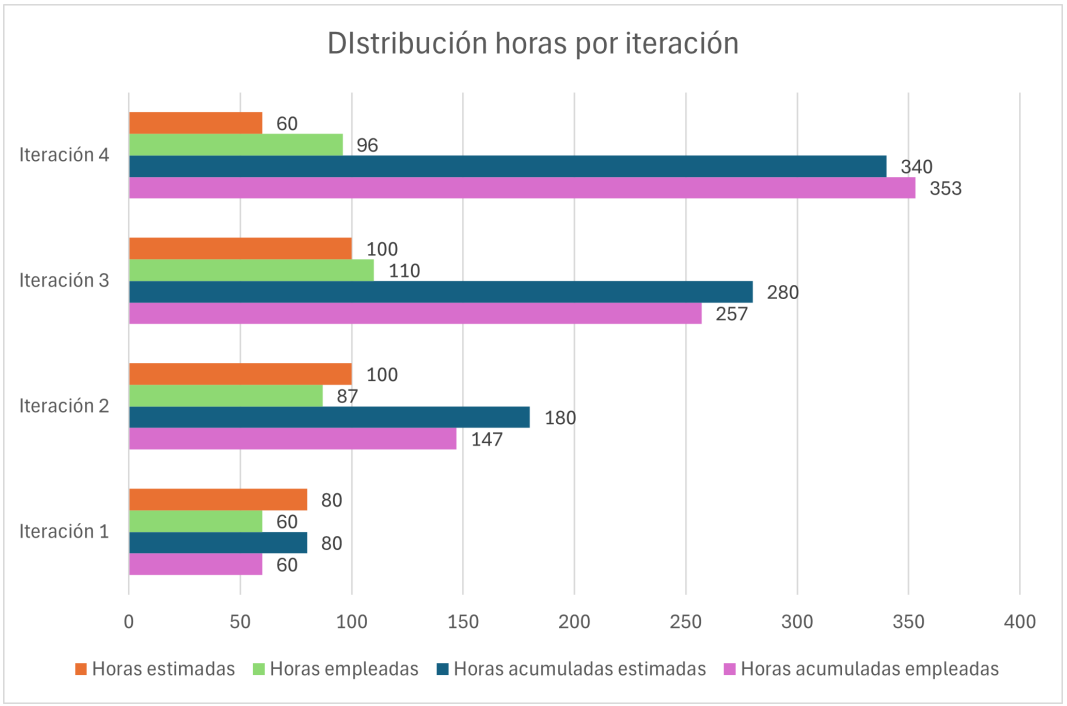


Figura 2.4: Comparación del número de horas estimadas y reales por cada iteración

Capítulo 3

Descripción de las iteraciones

En este capítulo se desarrollará la explicación de cada una de las iteraciones llevadas a cabo en este proyecto. Se comentarán desde la parte de análisis hasta la finalización del proyecto, comentando tanto los avances como los contratiempos.

3.1. Iteración 1

Esta primera iteración, se corresponde con el estudio de las distintas herramientas a utilizar en el desarrollo (ver Secciones 1.5 y 2.2), el análisis de las necesidades del usuario 6.1, el análisis del proyecto 4.1, y la selección de la arquitectura de los proyectos 4.2.3.

Debido a la experiencia previa se eligen como IDEs Android Studio, ya que cuenta con númerosas facilidades para el desarrollo en flutter, y Visual Studio Code.

Antes de comenzar con el análisis del proyecto, es necesario realizar un análisis de las necesidades del usuario para establecer los requisitos. Para obtener las necesidades de los usuarios del juego Marvel Champions, se crea el cuestionario (ver Apéndice B) que se comparte por diferentes foros por los que usuarios del juego hablan acerca de este. Se esperaron unas semanas hasta que el número de personas que realizó el cuestionario dejó de subir. Tras obtener la respuesta de 101 personas se procedió a realizar un análisis de estas respuestas, obteniendo así los diferentes contenidos de importancia que debía tener la aplicación.

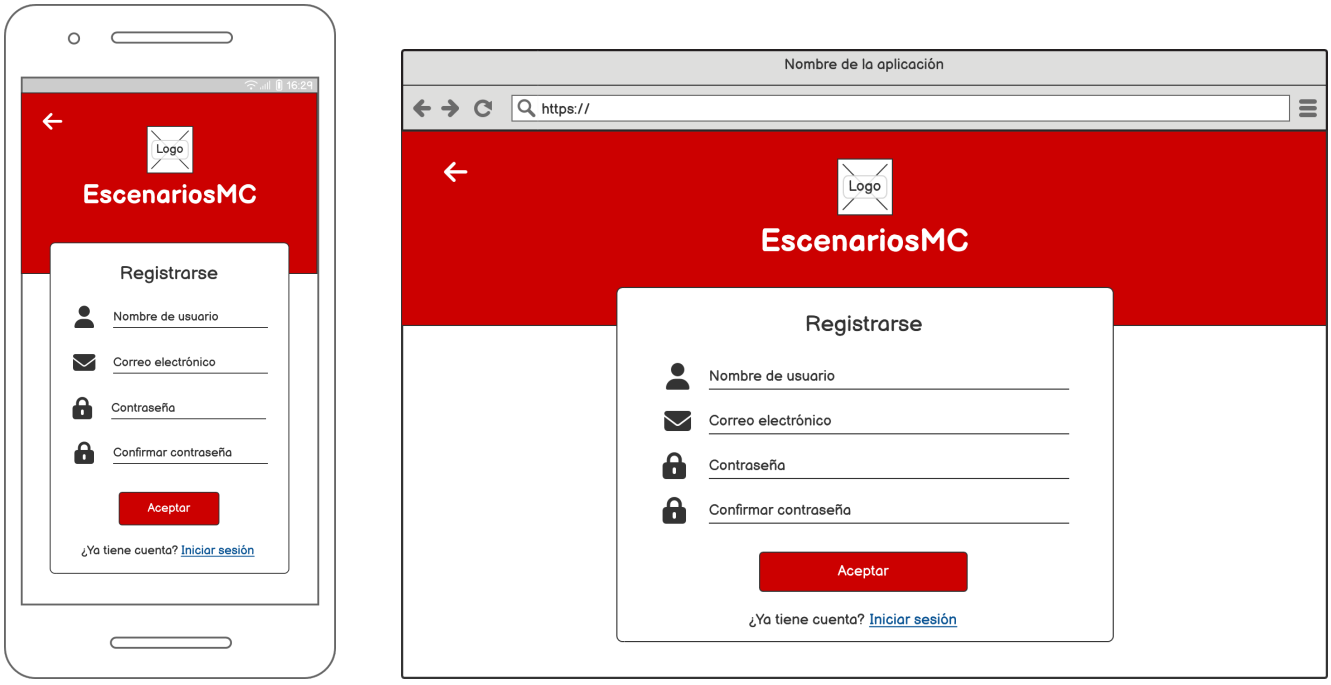
Posteriormente, se realizó la fase de análisis en la que se detallaron todos los requisitos, los casos de uso y el modelo de dominio mostrados en el siguiente capítulo.

Antes de comenzar a desarrollar la aplicación es necesario seleccionar una arquitectura adecuada para los dos proyectos. La arquitectura elegida para ambos proyectos es la Arquitectura Limpia, ya que es una buena forma de organizar el código de forma modular, de manera que sea fácilmente mantenible y testeable. Además de que aporta escalabilidad y claridad en la organización del código.

3.2. Iteración 2

En esta iteración se propone crear los dos proyectos, de back-end y front-end, y realizar la funcionalidad correspondiente al registro de usuario, inicio de sesión de usuario, menú, y publicación de partidas.

Lo primero es crear la base de datos con la tabla de usuarios para poder crear los usuarios, lo que equivale al registro de usuarios en el proyecto back-end. Tras crear un *endpoint* con esta funcionalidad, lo siguiente que se realiza es el diseño de la pantalla de registro de usuario en Balsamiq [33] (ver Figura 3.1), que posteriormente se desarrolla en Flutter. Tras realizar el diseño, se añade la funcionalidad en la parte del front-end.



(a) Diseño en móvil

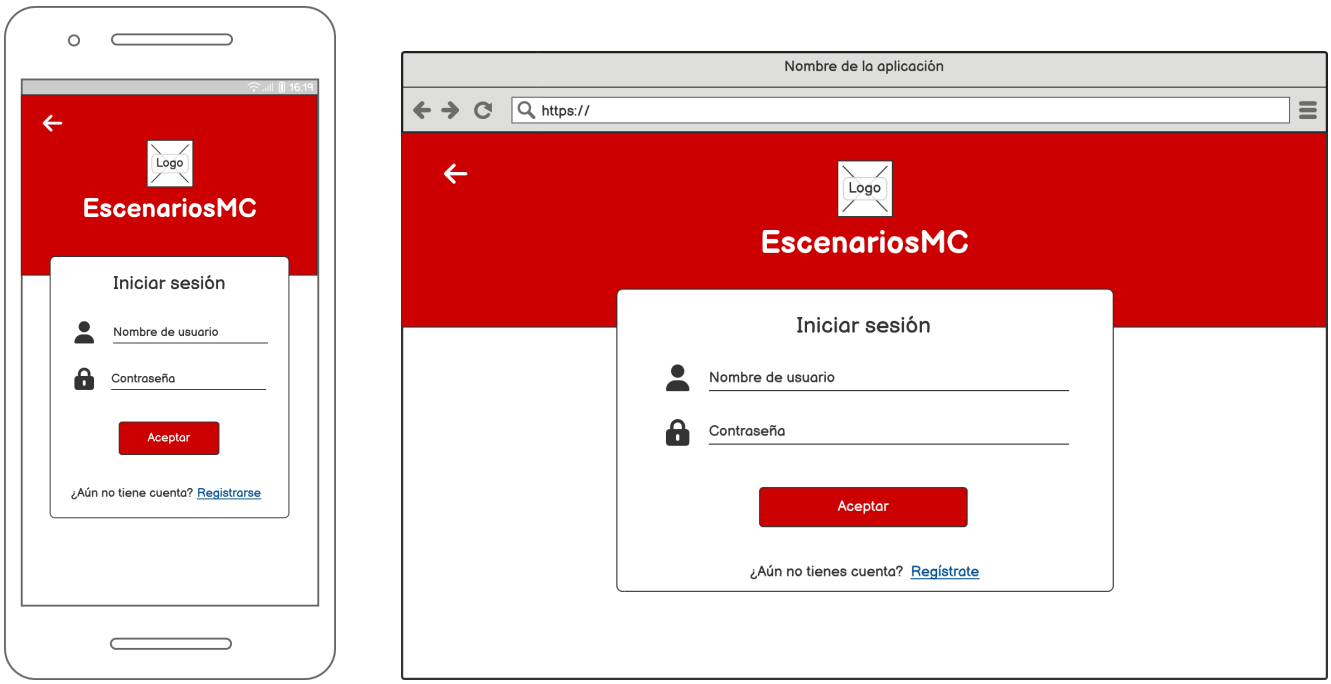
(b) Diseño en web

Figura 3.1: Diseño de la pantalla de registro de usuario

Después, se lleva a cabo la funcionalidad del inicio de sesión de la misma manera que el registro de usuario, y se añade autenticación JWT para que posteriormente, solo aquellos usuarios que hayan iniciado sesión (tengan un token JWT) puedan acceder a ciertos recursos específicos. El diseño de esta pantalla se muestra en la Figura 3.2.

Posteriormente, se lleva a cabo el desarrollo del menú de la aplicación, tanto para web como para móvil. Ya que, dependiendo de la plataforma los menús son diferentes.

Durante el desarrollo de la funcionalidad de la publicación de partidas ocurre un contratiempo. No es posible acceder a los escenarios y conjuntos modulares que ofrece la API de MarvelCDB de una manera sencilla, ya que esta API está pensada para la creación de mazos y no la gestión de escenarios. Por lo que, la solución a la que se llega para acceder a estos recursos es realizar un *script* que obtenga tanto escenarios como conjuntos modulares y los vuelque en la base de datos. Por lo tanto, es necesario la



(a) Diseño en móvil

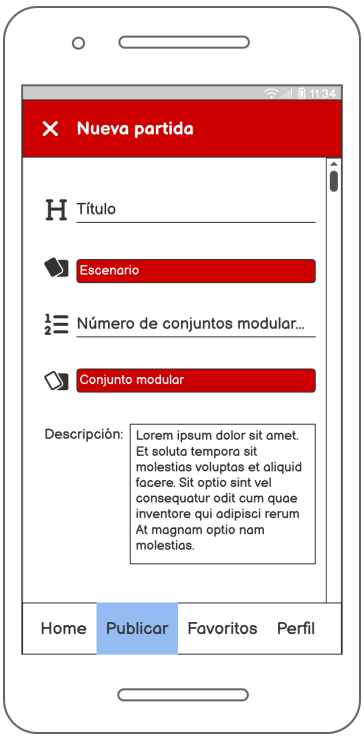
(b) Diseño en web

Figura 3.2: Diseño de la pantalla de inicio de sesión

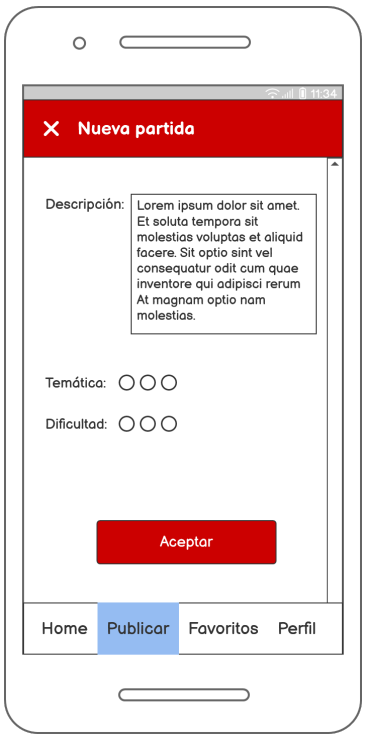
ejecución de este *script* antes de la puesta en marcha del proyecto back-end. El diseño de esta pantalla se muestra en las Figuras 3.3 y 3.4.

Al mismo tiempo ocurre otro contratiempo. En un principio se estaba utilizando el patrón BLoC (Business Logic Component) para separar la lógica de negocio de la interfaz de usuario, pero se decide cambiar este patrón y utilizar GetX, ya que el rendimiento de la aplicación, la optimización de recursos y la agilidad de desarrollo utilizando BLoC es menor que utilizando GetX.

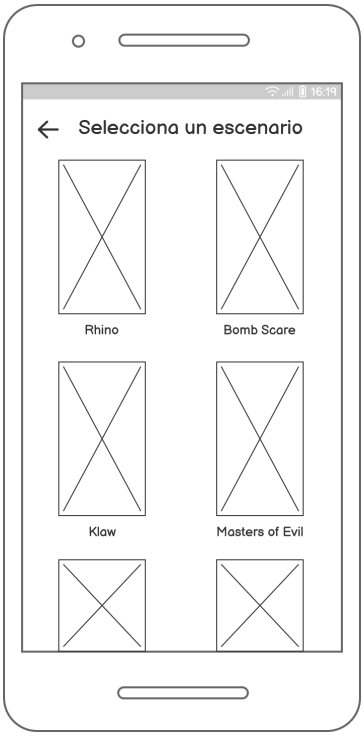
GetX [34] es un paquete de gestión de estado y de navegación. Se decide utilizar ya que simplifica la gestión de estados al proporcionar un mecanismo reactivo y eficiente para actualizar los componentes de la interfaz de usuario. Este utiliza el patrón observador para notificar a estos componentes cuando el estado cambia. Además de proporcionar su propio sistema de inyección de dependencias que facilita la gestión de las dependencias de los componentes de la aplicación. Está enfocado en el rendimiento y en el consumo mínimo de recursos, lo cual es beneficioso entre otras cosas para la mejora de la experiencia del usuario, ya que será una aplicación más rápida y fluida debido a que realiza las actualizaciones de estado con más eficiencia y rapidez [35].



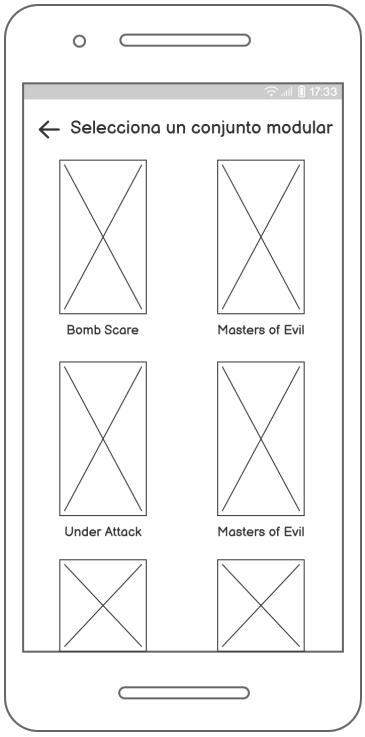
(a) Pantalla publicar partida 1



(b) Pantalla publicar partida 2

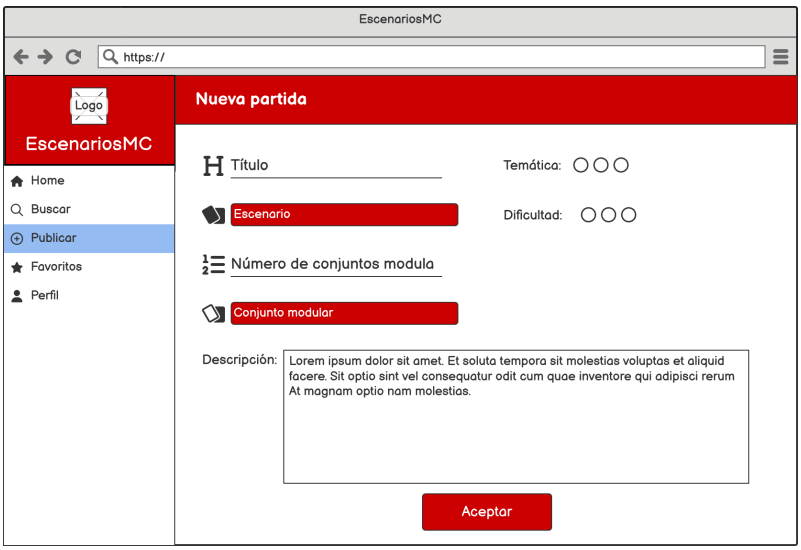


(c) Pantalla publicar; selección de escenario

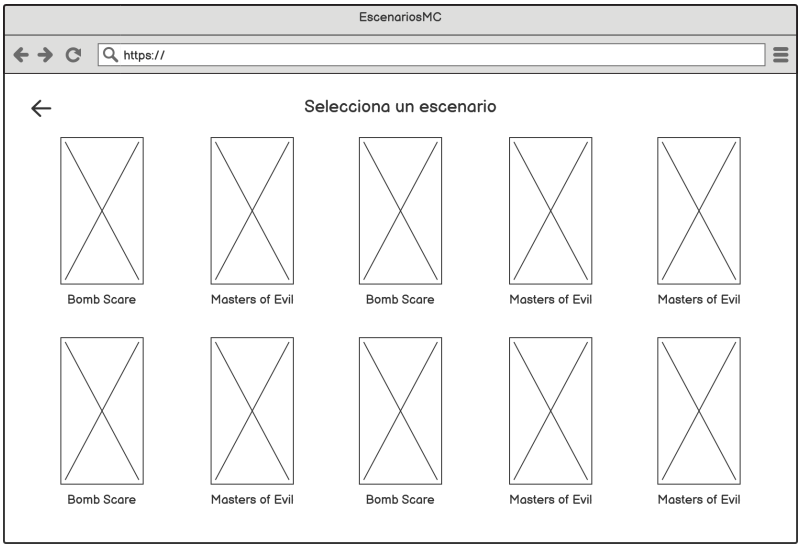


(d) Pantalla publicar; selección de conjunto modular

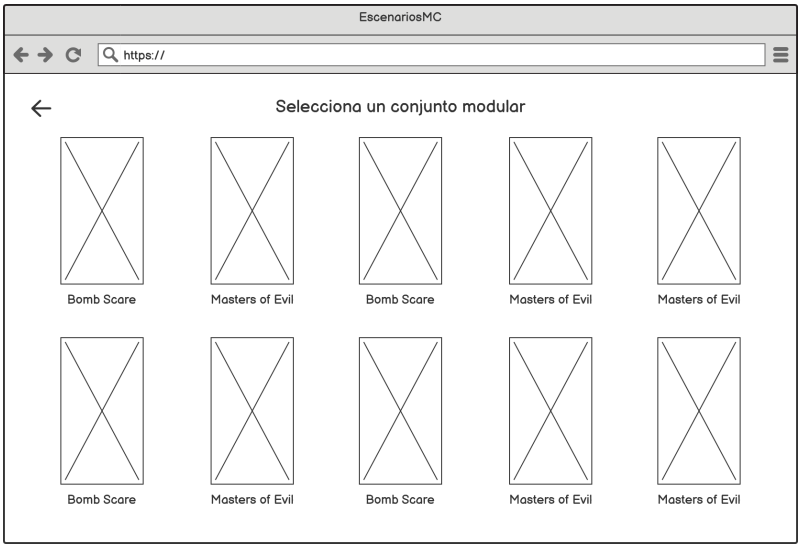
Figura 3.3: Diseño de la pantalla para publicar una partida en móvil



(a) Pantalla publicar partida 1



(b) Pantalla publicar; selección de escenario



(c) Pantalla publicar; selección de conjunto modular

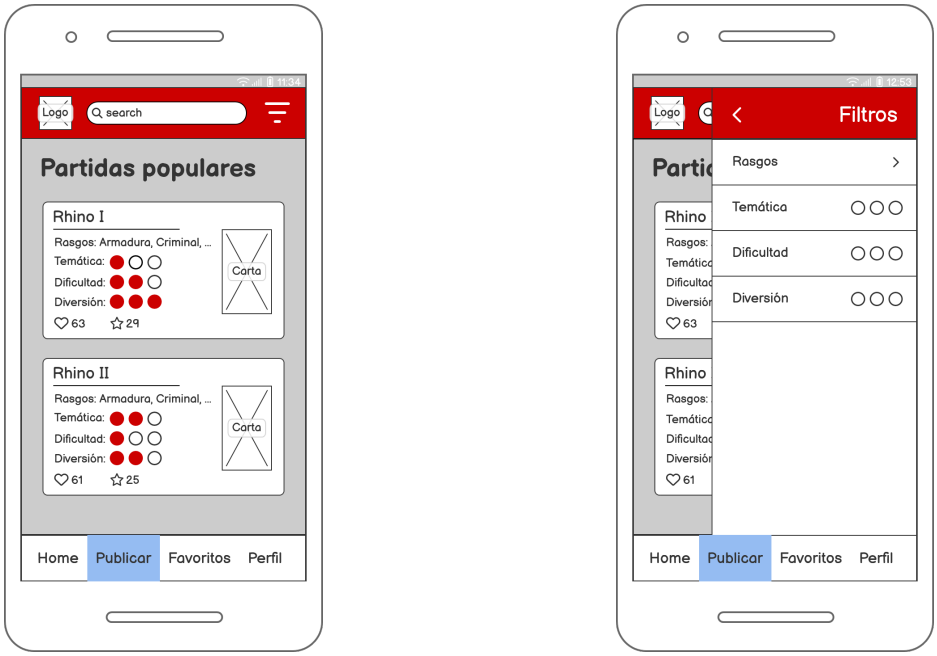
Figura 3.4: Diseño de la pantalla para publicar una partida en web

3.3. Iteración 3

En esta tercera iteración se propone realizar la funcionalidad correspondiente a la obtención de las partidas en la pantalla principal de la aplicación, poder mostrar con detalle cada una de las partidas con toda su información, realizar la funcionalidad correspondiente a dar *like* a una partida, a guardarla en favoritos, y a valorar tanto su grado de dificultad como de diversión.

Lo primero a realizar en esta iteración es la obtención de las partidas para mostrarlas en la pantalla principal de la aplicación. Para ello en el proyecto Node.js se crea un endpoint para obtener las distintas partidas publicadas por los diferentes usuarios. Para no obtener todas abruptamente y poder causar lentitud o un tiempo de respuesta largo al obtener las partidas desde el proyecto Flutter, se decide utilizar paginación en la llamada a los endpoints. De esta manera, las partidas se van cargando dinámicamente, y no causa el colapso o un rendimiento lento de la aplicación.

El diseño de la pantalla principal, en la que se pueden ver las partidas publicadas por diferentes usuarios, pudiendo así ordenar estas partidas por los diferentes campos de estas (por ejemplo, por grado de dificultad, de diversión o por mayor número de *likes*) se puede ver en las Figuras 3.5 y 3.6. Esta pantalla llevó algo más de tiempo del esperado ya que hubo algunos problemas con la acción de refrescar la pantalla.

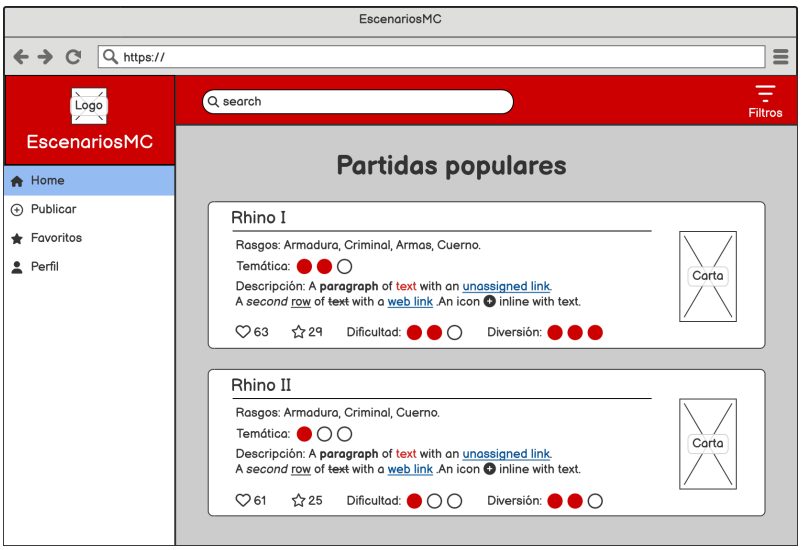


(a) Pantalla principal

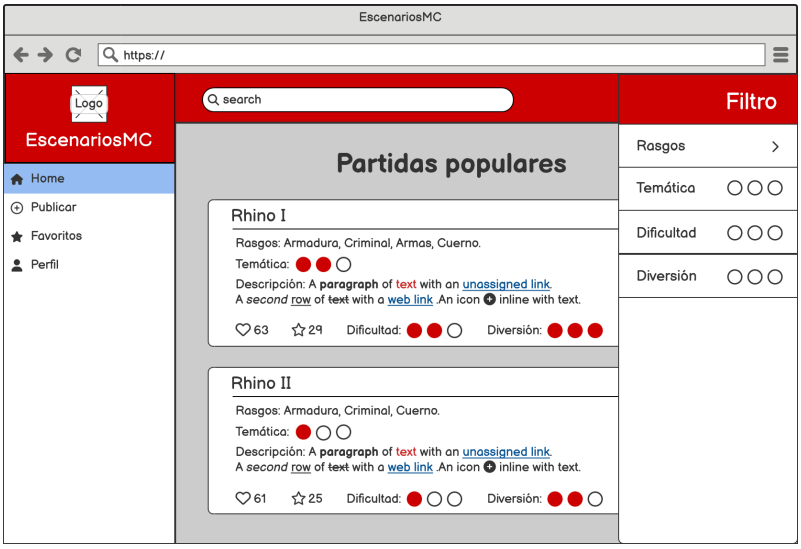
(b) Filtros pantalla principal

Figura 3.5: Diseño de la pantalla principal en móvil

Lo siguiente es realizar la pantalla para mostrar las partidas con toda la información necesaria (el diseño de la pantalla correspondiente se muestra en la Figura 3.7). Realizar esta pantalla llevó mucho tiempo ya que hubo numerosos contratiempos con la información necesaria a mostrar de la partida, las acciones a realizar por el usuario dentro de esta partida, como guardar la partida en favoritos o dar *like*. Para esto último fue necesario crear nuevos endpoints en el proyecto Node.js para poder realizar estas



(a) Pantalla principal



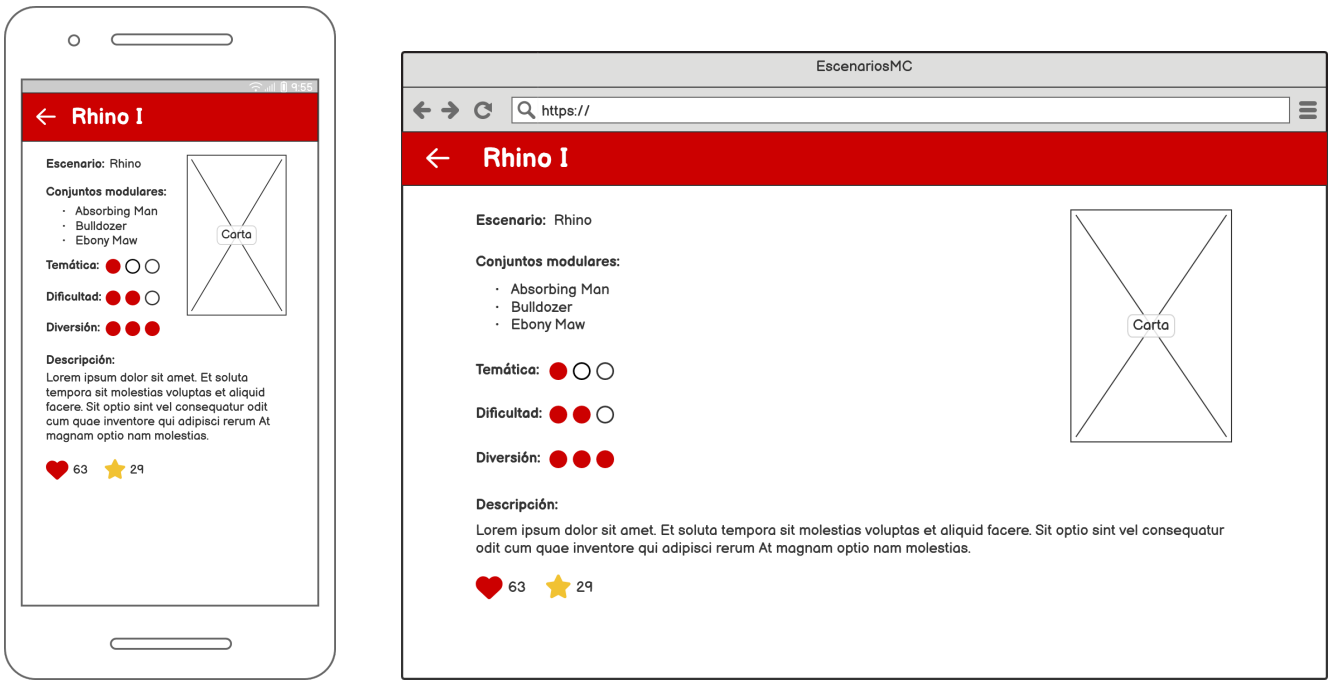
(b) Filtros pantalla principal

Figura 3.6: Diseño de la pantalla principal en web

tareas.

Respecto a la información que se debía mostrar de las partidas, se añadió información tanto del escenario que tenía la partida como de sus conjuntos modulares: número de aumentos 1.3, tipos de aumento, número de esbirros 1.3, número de planes secundarios 1.3 y número de cartas totales. Estos aumentos se debían de obtener al volcar la información de la API de MarvelCDB en el script (se hablará de él en el siguiente capitulo), por lo que, el desarrollo de esta pantalla supuso bastante tiempo en esta iteración.

En esta pantalla, también se agregó la funcionalidad para poder valorar tanto el grado de dificultad de una partida como el de diversión. Para ello fue necesario crear nuevos *endpoints* al igual que con las acciones de dar *like* o guardar una partida en favoritos. Por todo ello, el desarrollo de esta pantalla llevó más tiempo del esperado.



(a) Diseño en móvil

(b) Diseño en web

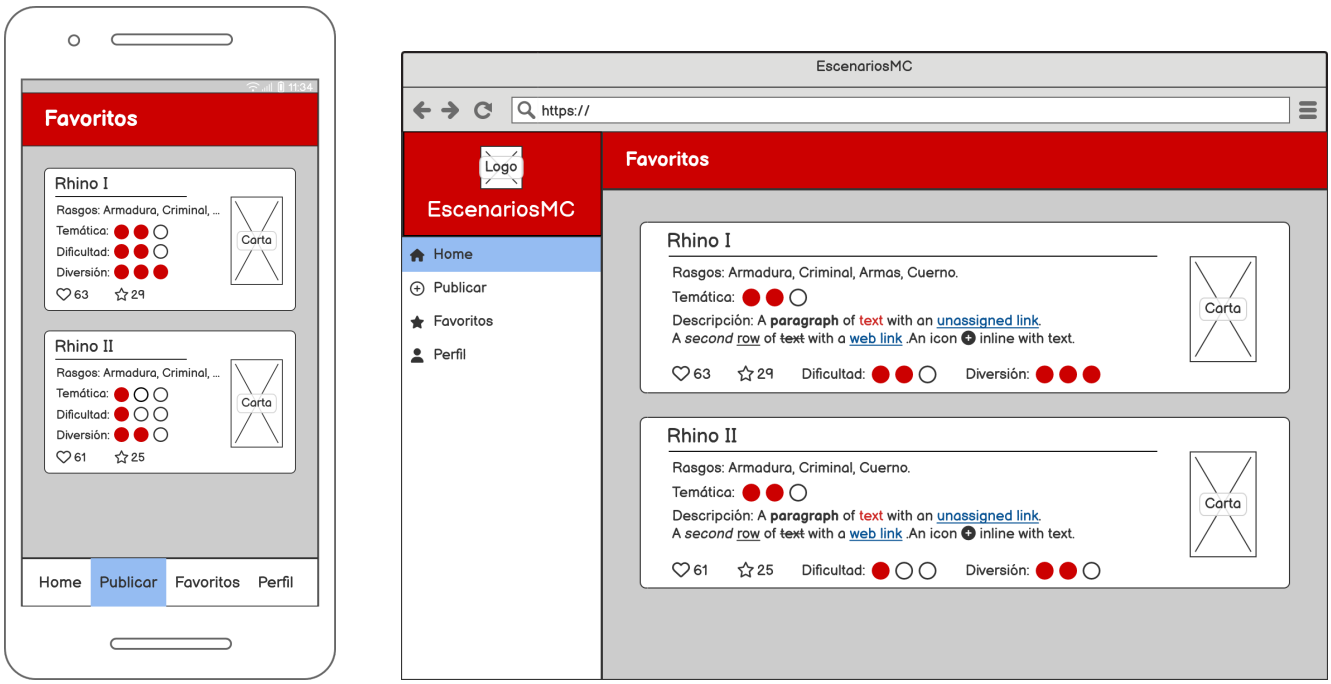
Figura 3.7: Diseño de la pantalla de detalles de una partida

3.4. Iteración 4

Esta iteración se corresponde con el desarrollo de las pantallas de favoritos, del perfil del usuario y de búsqueda de partidas[1], junto con sus correspondientes *endpoints* en la API del proyecto back-end para que la obtención de la información correspondiente se pudiera realizar. El diseño de la pantalla de favoritos se puede observar en la Figura 3.8, y el de la pantalla del perfil del usuario en la 3.9. También se desarrollan las funcionalidades para eliminar una partida propia de un usuario y editarla. Además de desarrollar todo esto, se realizaron pruebas manuales con la interfaz de usuario de la aplicación para comprobar que no hubiera errores ni *bugs*, ya que en esta iteración también se realiza el test de usabilidad con usuarios reales. Tras realizar estas pruebas se realizaron varios cambios en el código, como arreglar algunos *bugs* y cambiar la forma de representar los datos o acciones en algunas pantallas. Por último, se llevó a cabo el despliegue de la aplicación con Docker para poder realizar el test de usabilidad con los usuarios.

Primero se llevó a cabo el *endpoint* en el proyecto back-end que permite obtener las partidas guardadas en favoritos por un usuario. Tras haber realizado esto, se desarrolla la pantalla en la que se muestran estas partidas en el proyecto Flutter. Después se desarrolló el *endpoint* para obtener las partidas creadas por un determinado usuario. Este *endpoint* se utiliza en la pantalla del perfil del usuario que se desarrolla justo después. Y por último, se llevaron a cabo dos *endpoints* necesarios para las tareas a realizar en la pantalla de búsqueda de partidas, desarrollada tras conseguir la funcionalidad de estos dos *endpoints*. Estos son, uno para poder buscar las partidas por título o por el nombre de

[1] No se realizó diseño de la pantalla de búsqueda de partidas ya que en un principio se pensó que esa funcionalidad podría ir en la pantalla principal como se ve en el diseño de esta. Sin embargo, dado que esa funcionalidad es muy importante para los usuarios del juego, durante el desarrollo se decidió realizar una pantalla únicamente pensada en la búsqueda de partidas.



(a) Diseño en móvil

(b) Diseño en web

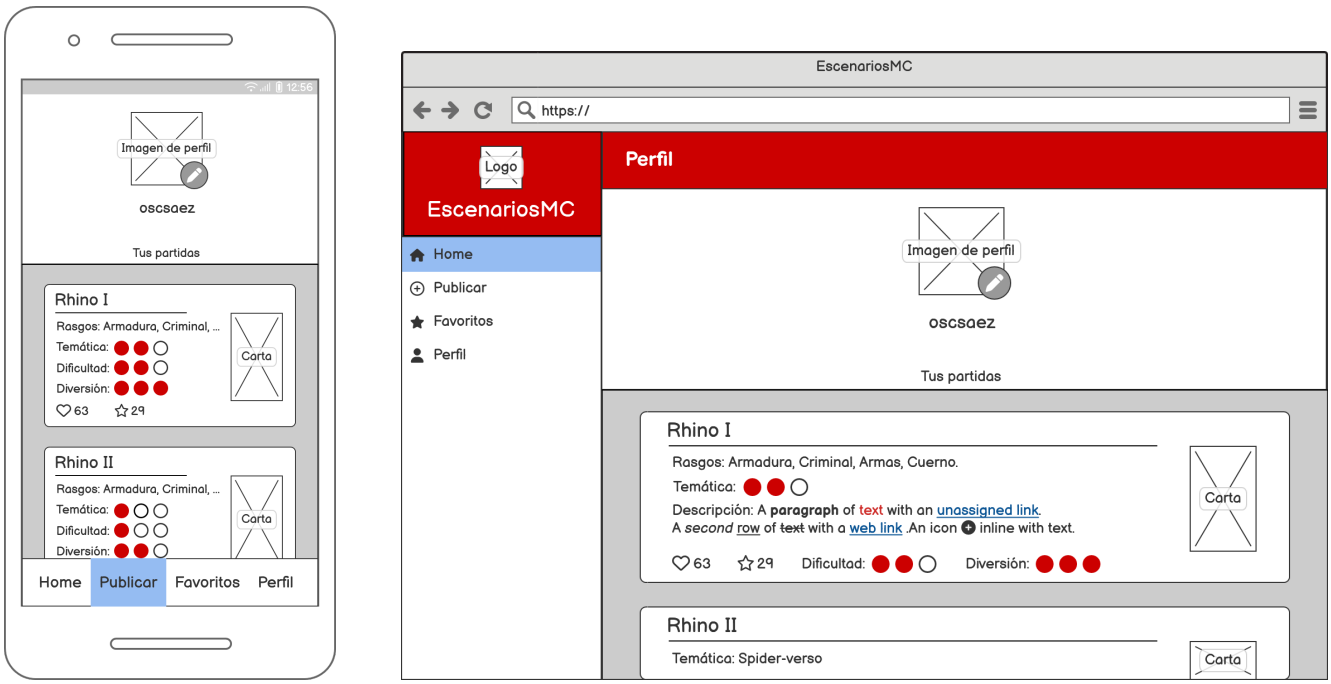
Figura 3.8: Diseño de la pantalla de favoritos

su autor, y otro para buscar partidas por los diferentes campos de las partidas, por ejemplo buscar una partida con grado de dificultad alto y poco temática.

Posteriormente, se realizaron los *endpoints* para poder eliminar una partida y poder editar una partida, y las respectivas acciones en el front-end. Para indicar que el usuario puede eliminar una partida, al seleccionar una partida, en los detalles de esta, aparece un botón para poder eliminarla. Al clicarlo, aparece un cuadro de diálogo que pide al usuario confirmación de la acción. En los detalles de la partida, también se encuentra un botón para la acción de editar la partida, al presionarlo, se navega a otra pantalla igual que la de crear una nueva publicación (ver Figura 3.3 o 3.4) en la que el usuario puede cambiar la información que desee y aceptar estos cambios, o descartarlos en caso de arrepentirse.

Como se ha mencionado al comienzo de esta sección, se realizaron pruebas manuales de la aplicación, realizando todo tipo de tareas que pueden realizarse en ella. Se encontraron varios *bugs* y formas de representar la información mejorables, por lo que, todos estos cambios se llevaron a cabo antes de realziar el test de usabilidad con usuarios auténticos.

Tras haber finalizado con el desarrollo de la aplicación, se lleva a cabo el despliegue de esta utilizando Docker en una de las máquinas virtuales de la escuela. Debido a que realizar un test de usabilidad presencial con tantos usuarios es inviable, se vio necesario el despliegue de la aplicación y la comunicación con estos usuarios a través de un grupo de Telegram para proporcionarles el cuestionario, y para que, en caso de caida del servidor o error en la aplicación, se comentara a través de este grupo y así poder solucionar estos problemas. Toda la información acerca de este test de usabilidad se encuentra en la Sección 6.2 del Capítulo 6.



(a) Diseño en móvil

(b) Diseño en web

Figura 3.9: Diseño de la pantalla del perfil del usuario

Capítulo 4

Estado final de la aplicación

En este capítulo se procede a desarrollar el análisis, diseño y estructura final de la aplicación. Incluyendo el cuestionario inicial y el análisis de sus respuestas, desarrollando los distintos casos de uso, patrones utilizados e incluyendo los correspondientes diagramas.

4.1. Análisis

4.1.1. Análisis de requisitos

En esta sección se desarrollan los distintos requisitos extraidos de las respuestas del cuestionario inicial (ver Apéndice B) proprocionadas por los usuarios del juego presentes en diferentes foros. Todo este proceso de análisis de las necesidades de los usuarios se encuentra en la Sección 6.1 del Capítulo 6.

Un requisito es una necesidad sobre el sistema que se documenta y existen 3 tipos: Funcionales, No funcionales y de información.

Requisitos funcionales

En la Tabla 4.1 se exponen los distintos requisitos funcionales del proyecto. Estos requisitos son aquellos que declaran cómo debe comportarse el sistema. El formato que se expone en la tabla consta del identificador, el nombre del requisito y una breve descripción del mismo.

ID	Nombre	Descripción
RF01	Registrarse	El sistema será capaz de registrar al usuario en el sistema.
RF02	Iniciar sesión	El sistema será capaz de autenticar al usuario ya registrado.
RF03	Añadir partida	El sistema será capaz de crear y almacenar una partida configurada por el usuario.
RF04	Actualizar partida	El sistema será capaz de actualizar y almacenar esa actualización de la partida.
RF05	Obtener partida/s	El sistema será capaz de obtener la/s partida/s publicada/s por otros usuarios y por el mismo.
RF06	Eliminar partida	El sistema será capaz de eliminar una partida creada por el usuario.
RF07	Dar like	El sistema será capaz de almacenar el número de likes de una partida y mostrarlo.
RF08	Guardar partida en favoritos	El sistema permitirá al usuario almacenar partidas publicadas por otros usuarios como favoritas.
RF09	Valorar la dificultad de una partida	El sistema permitirá al usuario valorar el grado de dificultad de diferentes partidas.
RF010	Valorar la diversión de una partida	El sistema permitirá al usuario valorar el grado de diversión de diferentes partidas.
RF11	Buscar partidas	El sistema permitirá al usuario realizar una búsqueda por nombre de la partida.
RF12	Filtrar por campo	El sistema permitirá al usuario filtrar las partidas por cualquiera de los campos que contiene la partida.
RF13	Autenticación del usuario	El sistema realizará la correspondiente autenticación de los usuarios en la aplicación.

Tabla 4.1: Requisitos funcionales

Requisitos no funcionales

En la Tabla 4.2 se describen los requisitos no funcionales del proyecto. Estos requisitos son características o restricciones que debe cumplir el sistema sistema o software más allá de su funcionalidad básica. El formato en el que se exponen en la tabla consta de las mismas columnas que en la 4.1.

ID	Nombre	Descripción
RNF01	Versión de Flutter	El sistema necesitará la versión de Flutter 3.13.8
RNF02	Versión de Node.js	El sistema necesitará la versión de Node.js 9.8.1
RNF03	Versión de MySQL	El sistema necesitará la versión de MySQL 8.0.34
RNF04	Versión de Docker	El sistema necesitará la versión de Docker 24.0.6
RNF05	Almacenamiento de datos	El sistema almacenará infromación que el usuario ha creado a través de este en una base de datos relacional SQL.
RNF06	Obtención de información de la API proporcionada por MarvelCDB	El sistema será capaz de recuperar información de la API externa de MarvelCDB.
RNF07	Protección de datos	El sistema almacenará las contraseñas de los usuarios cifradas.
RNF08	Integración de la API de MarvelCDB	El sistema mostrará información de la API externa de MarvelCDB.

Tabla 4.2: Requisitos no funcionales

Requisitos de información

En la Tabla 4.3 se describen los requisitos de información del proyecto. Estos requisitos son especificaciones detalladas o condiciones que describen la información que debe manejar el sistema informático o software. El formato en el que se exponen en la tabla consta de las mismas columnas que en los casos anteriores (Tablas 4.1 y 4.2).

ID	Nombre	Descripción
RI01	Gestión de datos del usuario	El sistema almacenará nombre de usuario, correo electrónico y contraseña de cada usuario.
RI02	Gestión de datos de las partidas	El sistema almacenará toda la información introducida por el usuario acerca de la partida que procede a publicar.
RI03	Gestión de datos de los escenarios	El sistema almacenará toda la información necesaria acerca de los escenarios obtenidos de la API de MarvelCDB.
RI04	Gestión de datos de los conjuntos modulares	El sistema almacenará toda la información necesaria acerca de los conjuntos modulares obtenidos de la API de MarvelCDB.
RI05	Gestión de *likes*, favoritos, dificultad y diversión	El sistema almacenará todas las relaciones vinculadas a los *likes*, la lista de favoritos, la dificultad y la diversión de una partida.

Tabla 4.3: Requisitos de información

4.1.2. Casos de uso

En el diagrama de la Figura 4.1 se muestran los casos de uso correspondientes a la aplicación, y a continuación sus explicaciones detalladas.

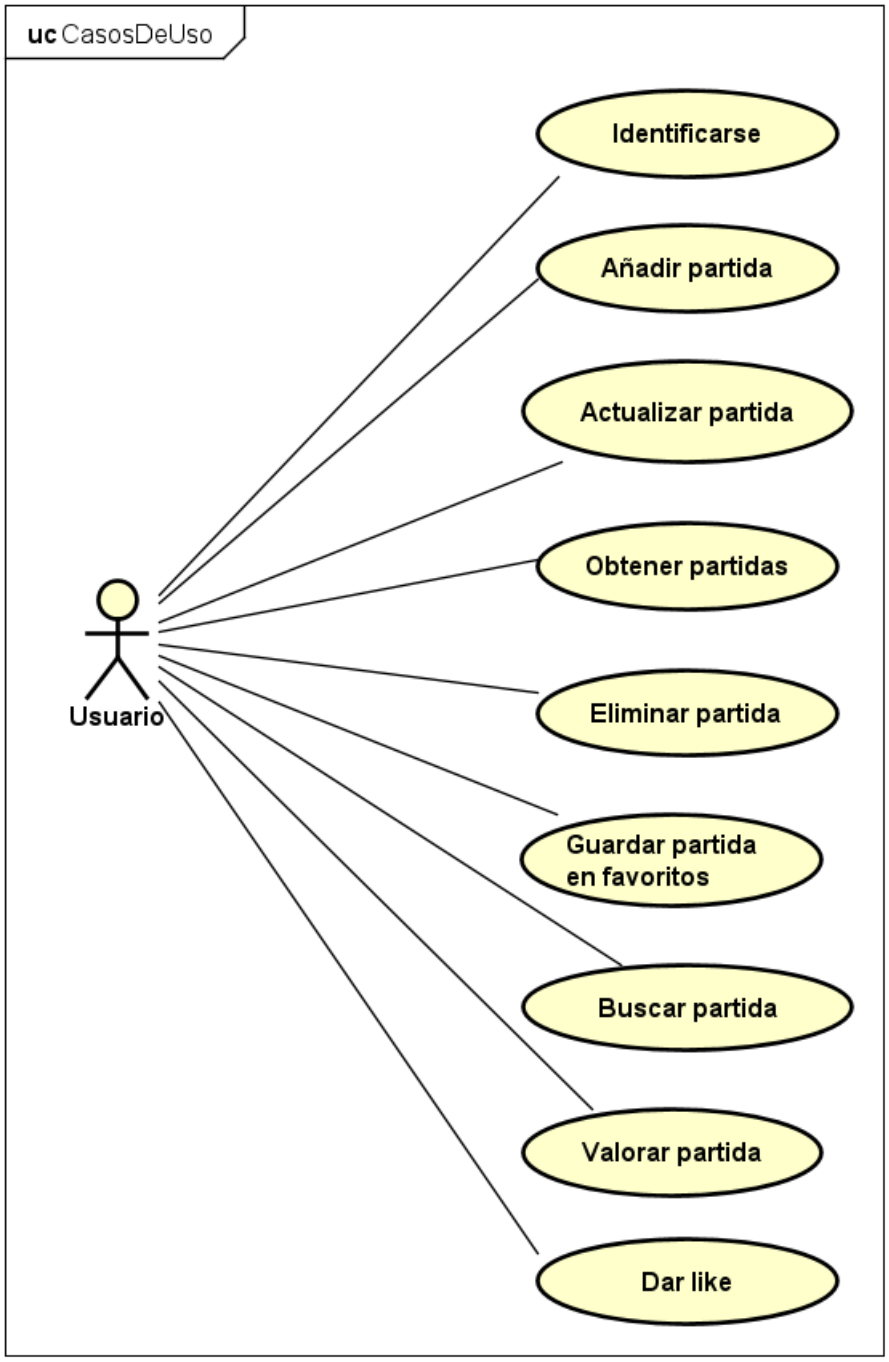


Figura 4.1: Diagrama casos de uso

ID	CU01
Nombre	Identificarse
Descripción	El usuario deberá poder identificarse en el sistema
Precondición	Ninguna
Secuencia normal	1. El usuario introduce nombre de usuario y contraseña. 2. El sistema comprueba que el nombre de usuario y la contraseña son correctos. 3. El sistema muestra un mensaje indicando que el usuario se ha identificado correctamente.
Secuencia alternativa	3.a Si el sistema comprueba que el nombre de usuario y la contraseña no son correctos muestra un mensaje indicando que no son correctos, y el caso de uso queda sin efecto.
Postcondición	Ninguna
Frecuencia	Media

Tabla 4.4: Caso de uso CU01

ID	CU02
Nombre	Añadir partida
Descripción	El usuario deberá poder crear una partida nueva
Precondición	El usuario debe estar identificado
Secuencia normal	1. El usuario introduce el título y descripción de la partida, selecciona el escenario, los conjuntos modulares, el nivel de dificultad y el nivel de temática. 2. El sistema comprueba que la información introducida por el usuario es la mínima necesaria para crear la partida. 3. El sistema crea la partida y la almacena. 4. El sistema muestra un mensaje indicando que la partida se ha creado correctamente.
Secuencia alternativa	2.a Si el sistema comprueba que el título de la partida está vacío, muestra un mensaje indicando que ese campo debe rellenarse, y el caso de uso queda sin efecto. 2.b Si el sistema comprueba que el escenario no ha sido seleccionado, muestra un mensaje indicando que no se ha seleccionado ningún escenario, y el caso de uso queda sin efecto. 2.c Si el sistema comprueba que no se ha seleccionado ningún nivel de dificultad, muestra un mensaje indicando que debe seleccionarse, y el caso de uso queda sin efecto. 2.d Si el sistema comprueba que no se ha seleccionado ningún nivel de temática, muestra un mensaje indicando que debe seleccionarse, y el caso de uso queda sin efecto.
Postcondición	Ninguna
Frecuencia	Media

Tabla 4.5: Caso de uso CU02

ID	CU03
Nombre	Actualizar partida
Descripción	El usuario deberá poder actualizar una partida ya creada
Precondición	El usuario debe de estar identificado
Secuencia normal	1. El usuario selecciona la partida que desea actualizar. 2. El sistema muestra la partida seleccionada. 3. El usuario introduce la información que quiere editar de la partida. 4. El sistema actualiza la partida. 5. El sistema muestra un mensaje indicando que la partida ha sido actualizada correctamente.
Secuencia alternativa	3.a Si el usuario cancela la acción, el caso de uso queda sin efecto.
Postcondición	Ninguna
Frecuencia	Baja

Tabla 4.6: Caso de uso CU03

ID	CU04
Nombre	Obtener partidas
Descripción	El usuario deberá poder obtener tanto sus partidas como las de otros usuarios
Precondición	Ninguna
Secuencia normal	1. El usuario accede a la página en la que se muestran las partidas. 2. El sistema muestra las partidas.
Secuencia alternativa	Ninguna
Postcondición	Ninguna
Frecuencia	Alta

Tabla 4.7: Caso de uso CU04

ID	CU05
Nombre	Eliminar partida
Descripción	El usuario deberá poder eliminar una partida creada por él
Precondición	El usuario debe de estar identificado
Secuencia normal	1. El usuario selecciona la partida que desea eliminar. 2. El sistema pide confirmación sobre la eliminación de la partida. 3. El usuario confirma. 4. El sistema muestra un mensaje indicando que la partida se eliminó correctamente.
Secuencia alternativa	3.a Si el usuario cancela la acción, el caso de uso queda sin efecto.
Postcondición	Ninguna
Frecuencia	Baja

Tabla 4.8: Caso de uso CU05

ID	CU06
Nombre	Guardar partida en favoritos
Descripción	El usuario deberá poder guardar una partida en la lista de favoritos
Precondición	El usuario debe estar identificado
Secuencia normal	1. El usuario selecciona la partida que desea guardar en favoritos. 2. El sistema muestra la partida seleccionada. 3. El usuario selecciona la opción para guardar la partida. 4. El sistema almacena la partida en favoritos para ese usuario.
Secuencia alternativa	Ninguna
Postcondición	Ninguna
Frecuencia	Alta

Tabla 4.9: Caso de uso CU06

ID	CU07
Nombre	Buscar partida
Descripción	El usuario deberá poder buscar una partida filtrando por diferentes campos.
Precondición	Ninguna
Secuencia normal	1. El usuario introduce los campos de la partida que desea buscar. 2. El sistema comprueba que existan partidas con esos campos 3. El sistema muestra las partidas con los campos correspondientes.
Secuencia alternativa	2.a Si ninguna partida corresponde con los campos introducidos por el usuario, el sistema muestra un mensaje indicando que no hay partidas con esos campos.
Postcondición	Ninguna
Frecuencia	Alta

Tabla 4.10: Caso de uso CU07

ID	CU08
Nombre	Valorar partida
Descripción	El usuario deberá poder valorar tanto el grado de dificultad como el grado de diversión de una partida.
Precondición	El usuario debe estar identificado
Secuencia normal	1. El usuario selecciona la partida que desea valorar. 2. El sistema muestra la partida seleccionada. 3. El usuario selecciona la valoración de la partida. 4. El sistema almacena la valoración seleccionada.
Secuencia alternativa	Ninguna
Postcondición	Ninguna
Frecuencia	Alta

Tabla 4.11: Caso de uso CU08

ID	CU09
Nombre	Dar *like*
Descripción	El usuario deberá poder dar *like* a una partida
Precondición	El usuario debe estar identificado
Secuencia normal	1. El usuario selecciona la partida a la que desea dar *like*. 2. El sistema muestra la partida seleccionada. 3. El usuario selecciona la opción para dar *like*. 4. El sistema almacena el *like* correspondiente.
Secuencia alternativa	Ninguna
Postcondición	Ninguna
Frecuencia	Alta

Tabla 4.12: Caso de uso CU09

4.1.3. Modelo de dominio

Teniendo en cuenta los requisitos y casos de uso definidos y descritos anteriormente, se elabora el diagrama de clases representado en la Figura 4.2. Podemos interpretar este modelo de dominio de la siguiente manera:

La clase *User* representa al propio usuario de la aplicación. Este usuario se puede relacionar con las partidas de diferentes formas por lo que tendrá una relación directa con la clase *Game*. Un usuario puede tener publicadas varias partidas, o no tener ninguna publicada. Además, tiene más relaciones con *Game*, ya que los usuarios pueden valorar el grado de dificultad y diversión de las partidas, darlas like o incluso guardarlas en favoritos. La relaciones que representan el grado de dificultad y el de diversión asignado por un usuario a una partida, están representadas en el diagrama a través de clases asociativas, ya que además de la multiplicidad, es necesario indicar que existe un valor entero que representa la valoración.

La clase *Game* se relaciona con la clase *Scenario*, ya que una partida siempre contiene un único escenario. Pero, las partidas también pueden tener relación con los conjuntos modulares (clase *ModularSet*), ya que una partida puede tener de 0 a 3 conjuntos modulares.

La clase *Card* representa las cartas que contiene un paquete (clase *Pack*). Algunas de estas cartas pueden forman parte de un escenario o de un conjunto modular. Cabe destacar que las cartas pueden ser de diferentes tipos, representado a través del enum *CardType*.

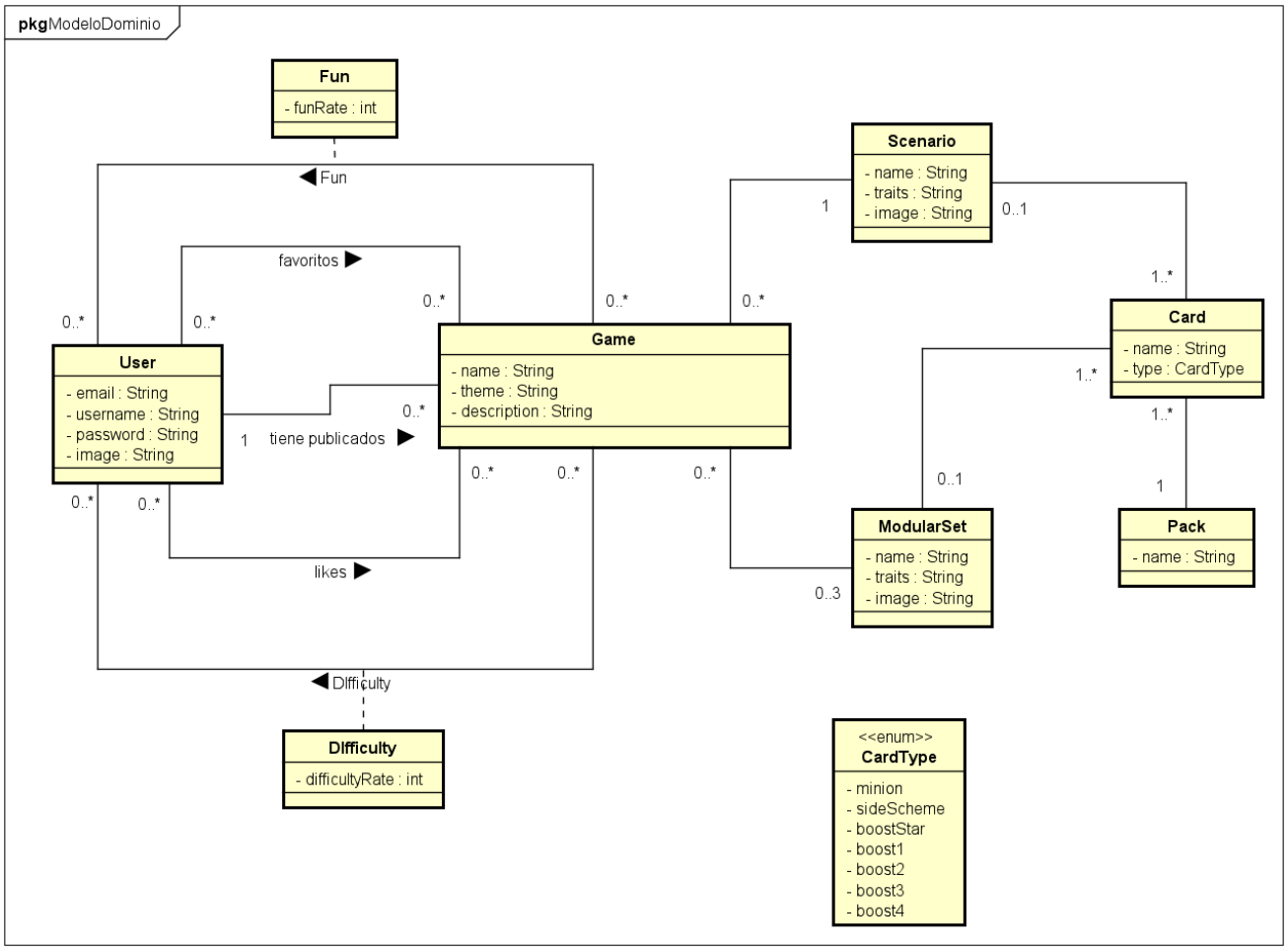


Figura 4.2: Modelo de dominio de la aplicación

4.1.4. Diagramas de actividad

En esta sección se incluyen los diagramas de actividad correspondientes a la ejecución de los diferentes casos de uso descritos en el apartado anterior.

CU01. Identificarse

Cuando el usuario desea identificarse, este introduce nombre de usuario y contraseña. Tras esto, el sistema comprueba que tanto el nombre de usuario como la contraseña son correctos. Si los datos son correctos, el sistema mostrará la correspondiente pantalla al usuario. En caso de ser datos erróneos, el sistema mostrará un mensaje indicando que el nombre de usuario o la contraseña introducidos son incorrectos.

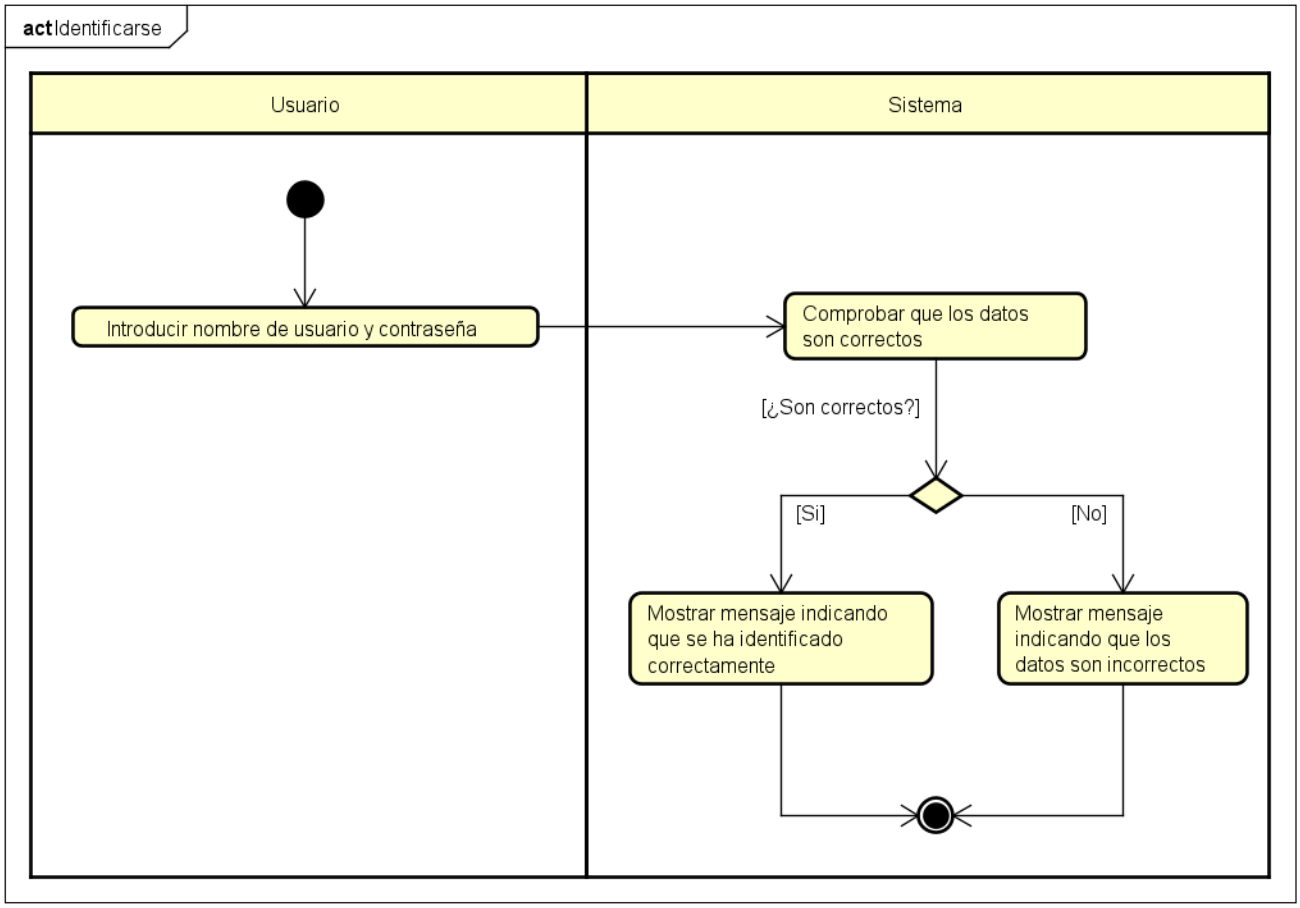


Figura 4.3: Diagrama de actividad correspondiente al CU01

CU02. Añadir partida

Cuando el usuario desea publicar una nueva partida, este debe introducir el título y descripción de la partida, y seleccionar el escenario, los conjuntos modulares deseados, el grado de dificultad y el grado de temática. Entonces, el sistema comprueba que todos estos datos han sido introducidos (los únicos opcionales son los conjuntos modulares y la descripción de la partida). Si los datos han sido introducidos, el sistema creará la partida y mostrará a través de un mensaje que la partida ha sido publicada correctamente. En caso de no haber introducido todos estos datos, el sistema mostrará un mensaje indicando que esos campos son necesarios para la creación de la partida.

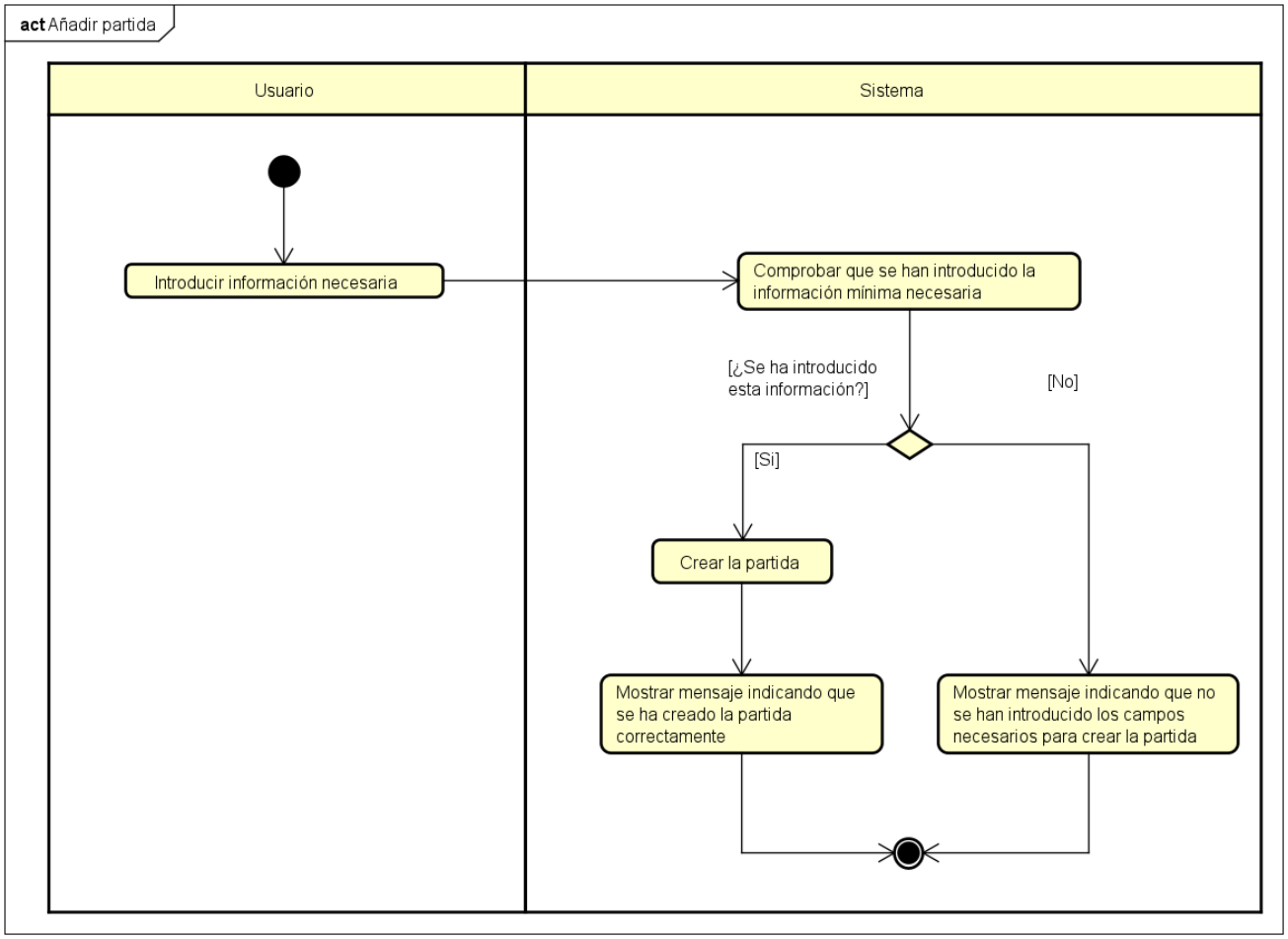


Figura 4.4: Diagrama de actividad correspondiente al CU02

CU03. Actualizar partida

Cuando el usuario desea actualizar una partida, este la selecciona, el sistema muestra la partida seleccionada y el usuario introduce la información que desea editar de esta.

Tras realizar todos los cambios deseados sobre la partida seleccionada, el usuario confirma, el sistema actualiza la partida con los cambios realizados y muestra un mensaje indicando al usuario que la partida ha sido actualizada correctamente.

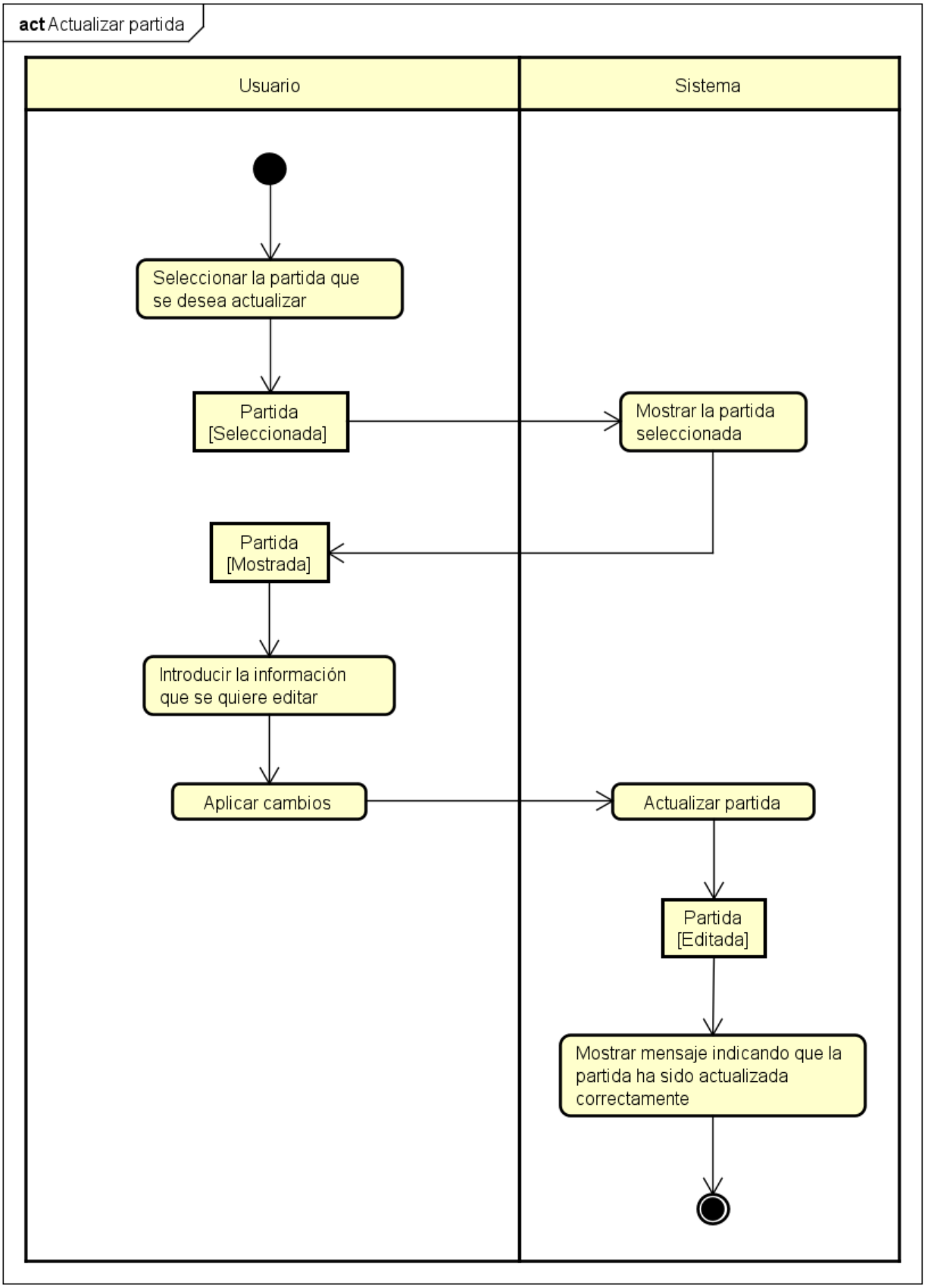


Figura 4.5: Diagrama de actividad correspondiente al CU03

CU04. Obtener partidas

Cuando el usuario desea obtener ya sea sus partidas o las partidas publicadas por otros usuarios, accede a la pantalla en la que estas se muestran, y el sistema muestra las correspondientes partidas.

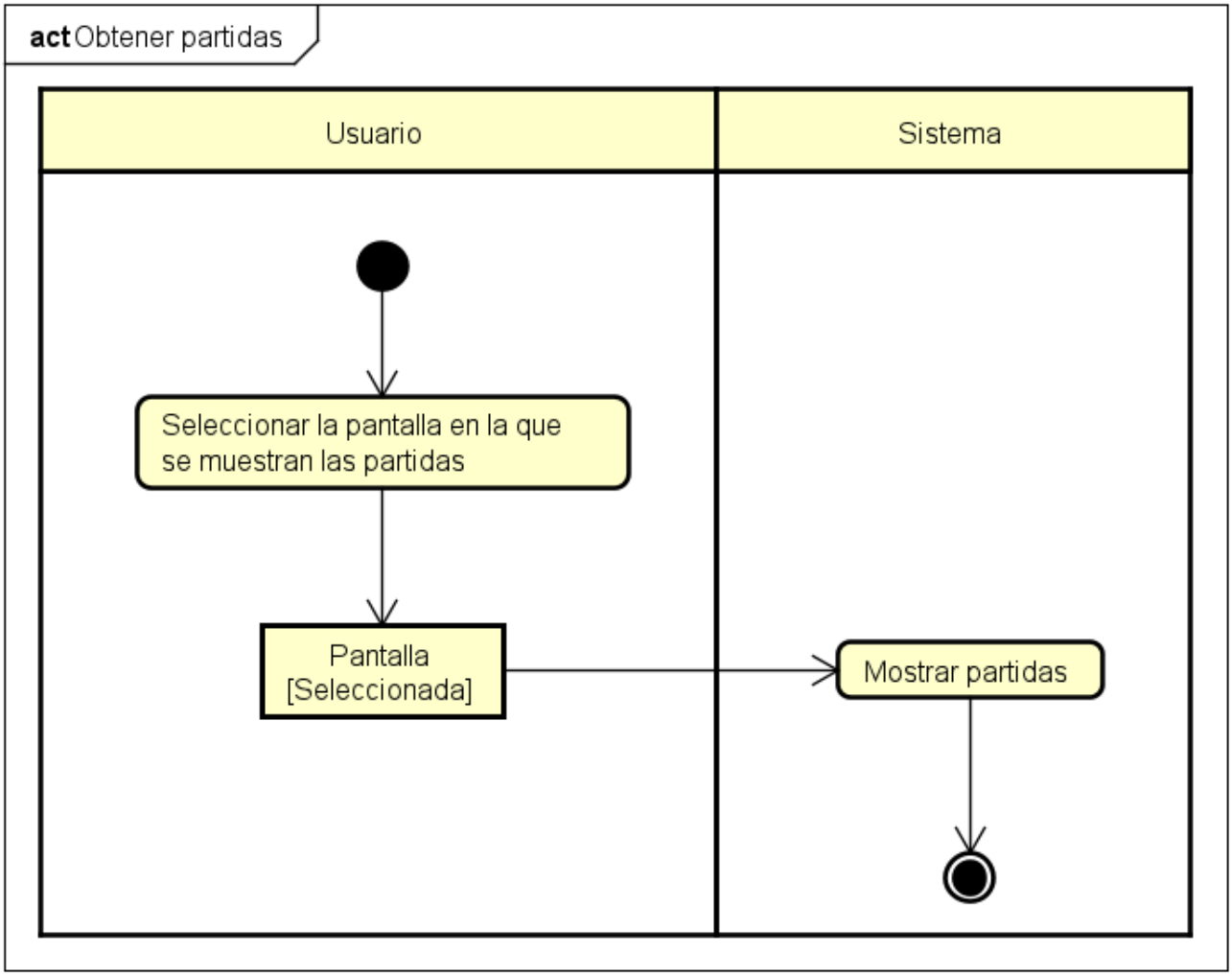


Figura 4.6: Diagrama de actividad correspondiente a CU04

CU05. Eliminar partida

Cuando el usuario desea eliminar una partida, este selecciona la partida, el sistema la muestra, y el usuario realiza la acción necesaria para eliminarla. Entonces, el sistema solicita confirmación para eliminar la partida. Si el usuario confirma, el sistema eliminará la partida e indicará a través de un mensaje que la partida fue eliminada. Si el usuario cancela la eliminación de la partida, el caso de uso queda sin efecto.

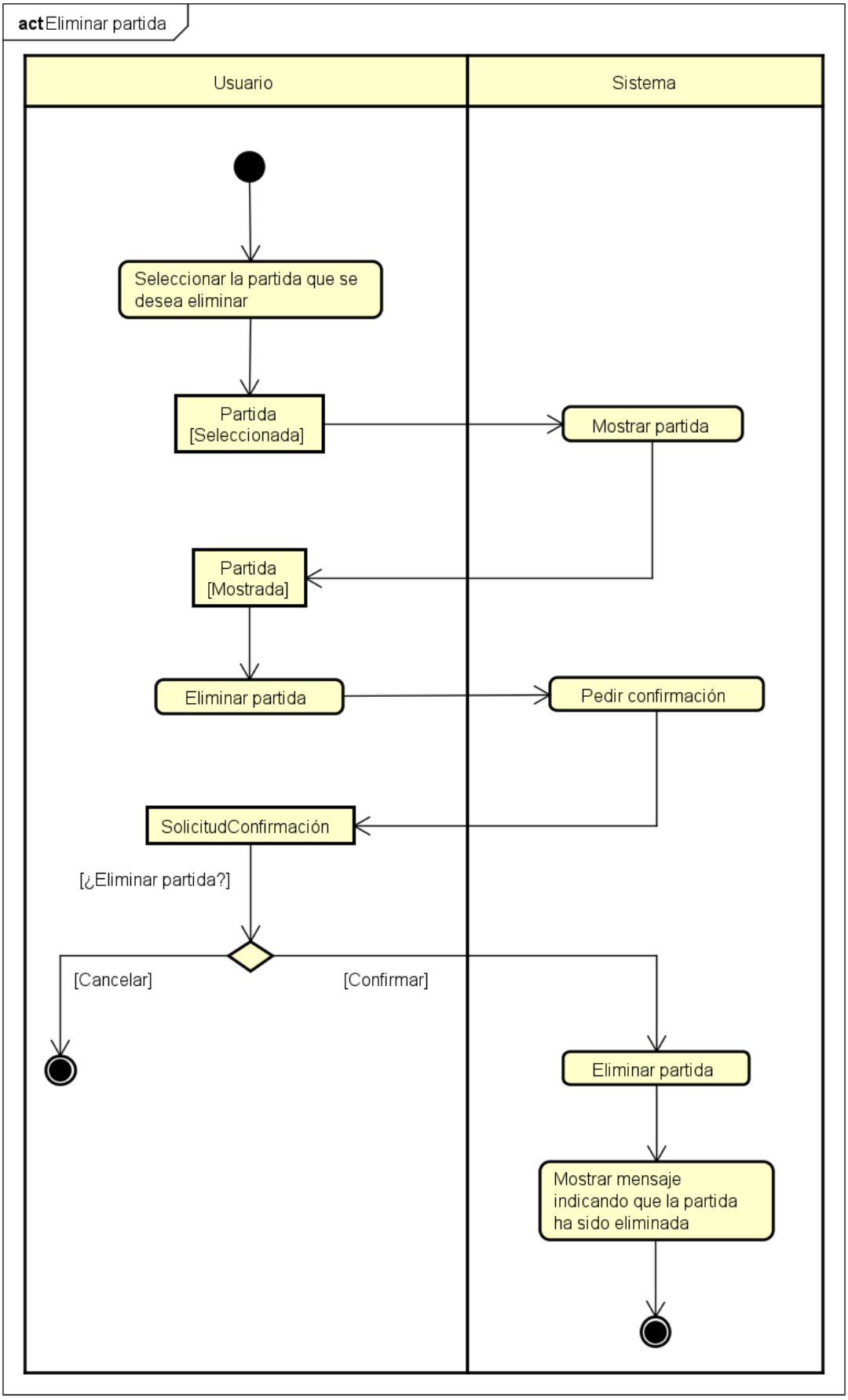


Figura 4.7: Diagrama de actividad correspondiente al CU05

CU06. Guardar partida en favoritos

La secuencia a realizar mostrada en el diagrama de secuencia de la Figura 4.8 comienza igual que en los diagramas de actividad anteriores. El usuario selecciona la partida que desea guardar en favoritos y el sistema la muestra. Tras esto, el usuario realiza la acción necesaria para añadir la partida en favoritos y el sistema almacena la partida en la lista de favoritos del usuario.

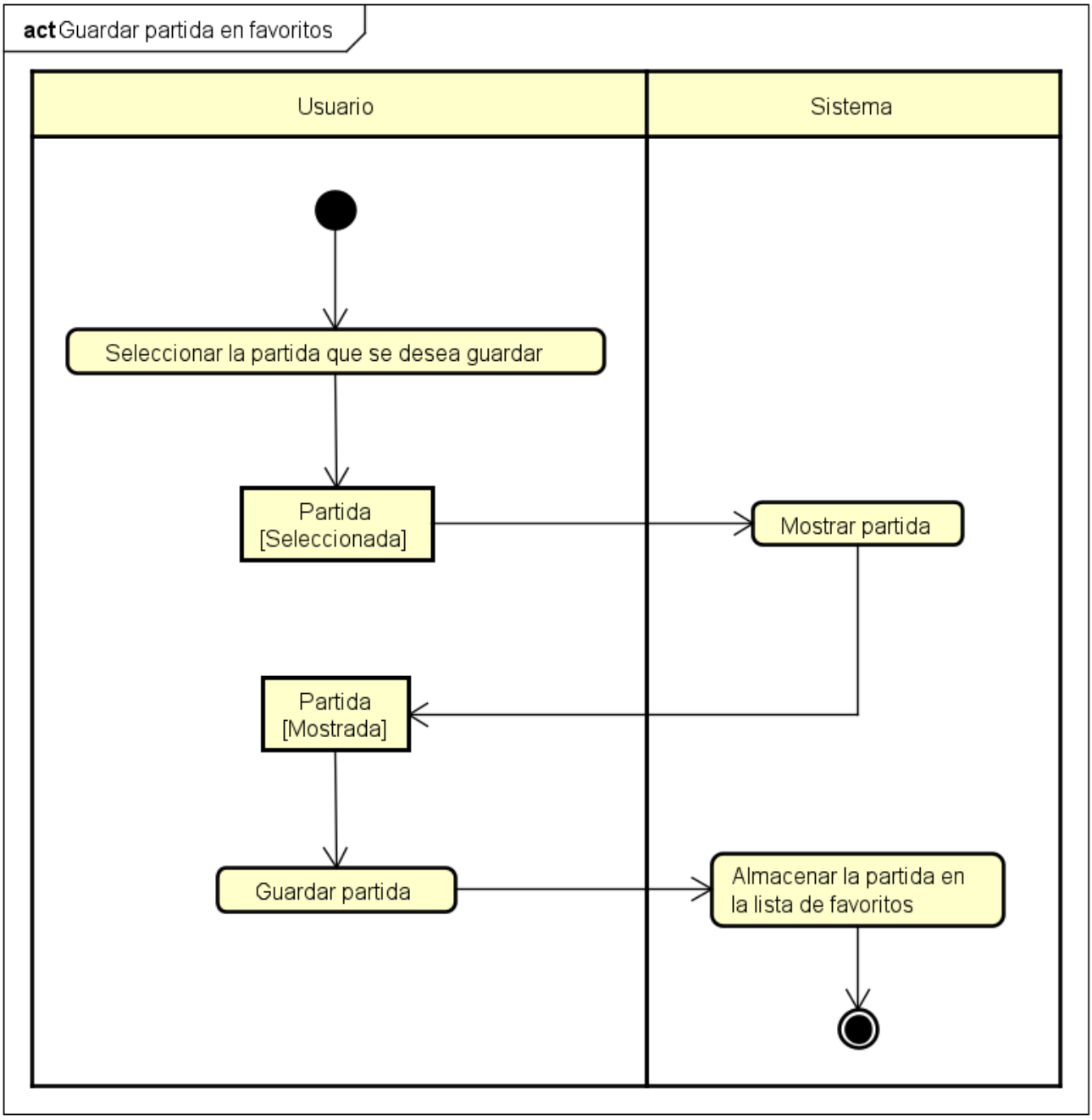


Figura 4.8: Diagrama de actividad correspondiente al CU06

CU07. Buscar partida

El caso de uso comienza con el usuario introduciendo uno o diferentes campos que desee que la partida a buscar tenga, y realiza la acción necesaria para buscar partidas. Entonces el sistema obtiene las partidas que corresponden con los filtros aplicados por el usuario y, en caso de existir partidas que contengan esos campos el sistema las muestra, en caso contrario muestra un mensaje indicando que no hay partidas con los filtros aplicados.

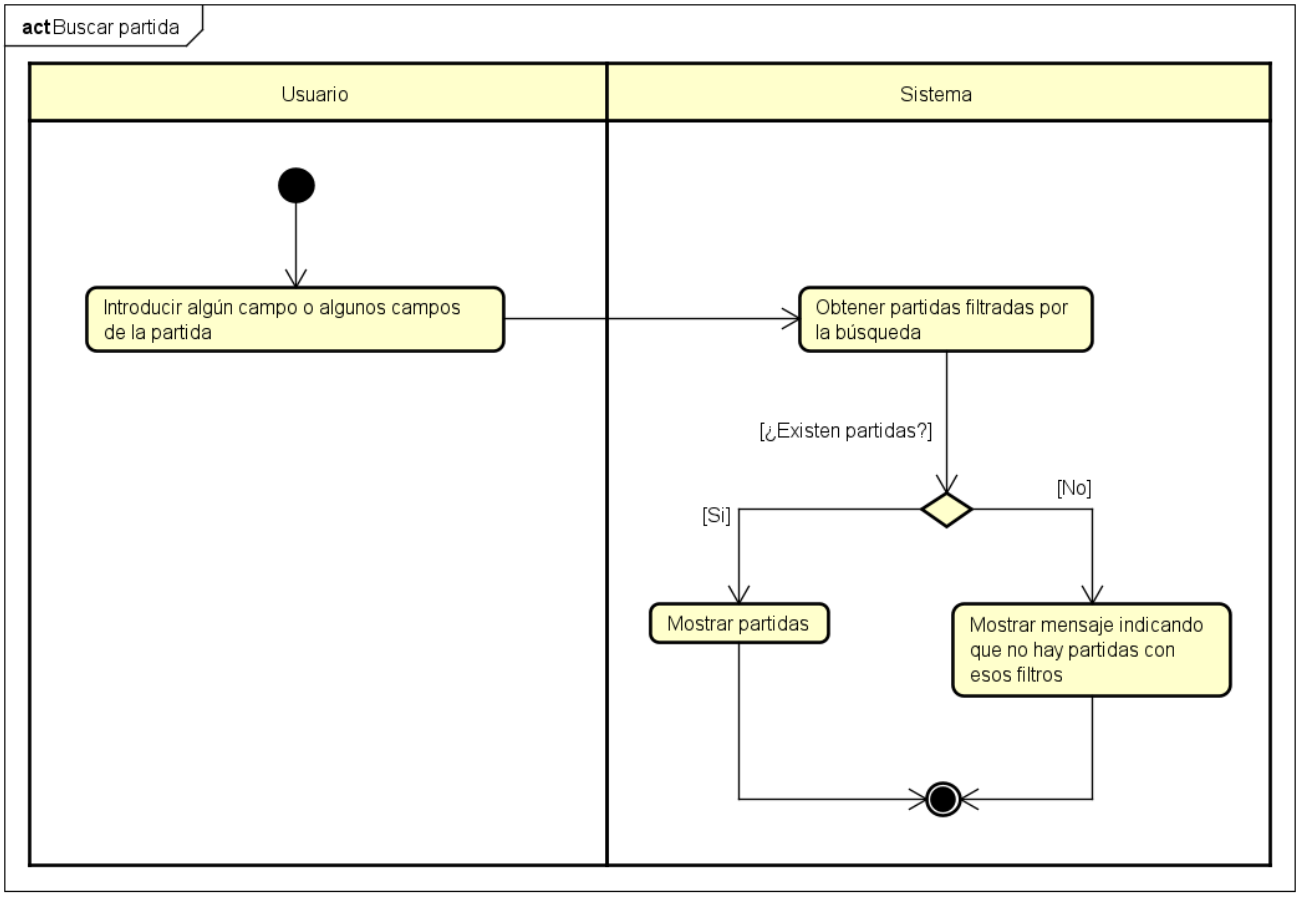


Figura 4.9: Diagrama de actividad correspondiente al CU07

CU08. Valorar partida

En este diagrama de actividad se muestra la secuencia de valorar tanto el grado de dificultad de una partida como el de diversión, ya que la tarea a realizar es muy similar.

Cuando el usuario desea valorar una partida, este selecciona la partida deseada, el sistema muestra la partida, y el usuario selecciona la valoración deseada de la partida. Por último, el sistema almacena el valor introducido por el usuario.

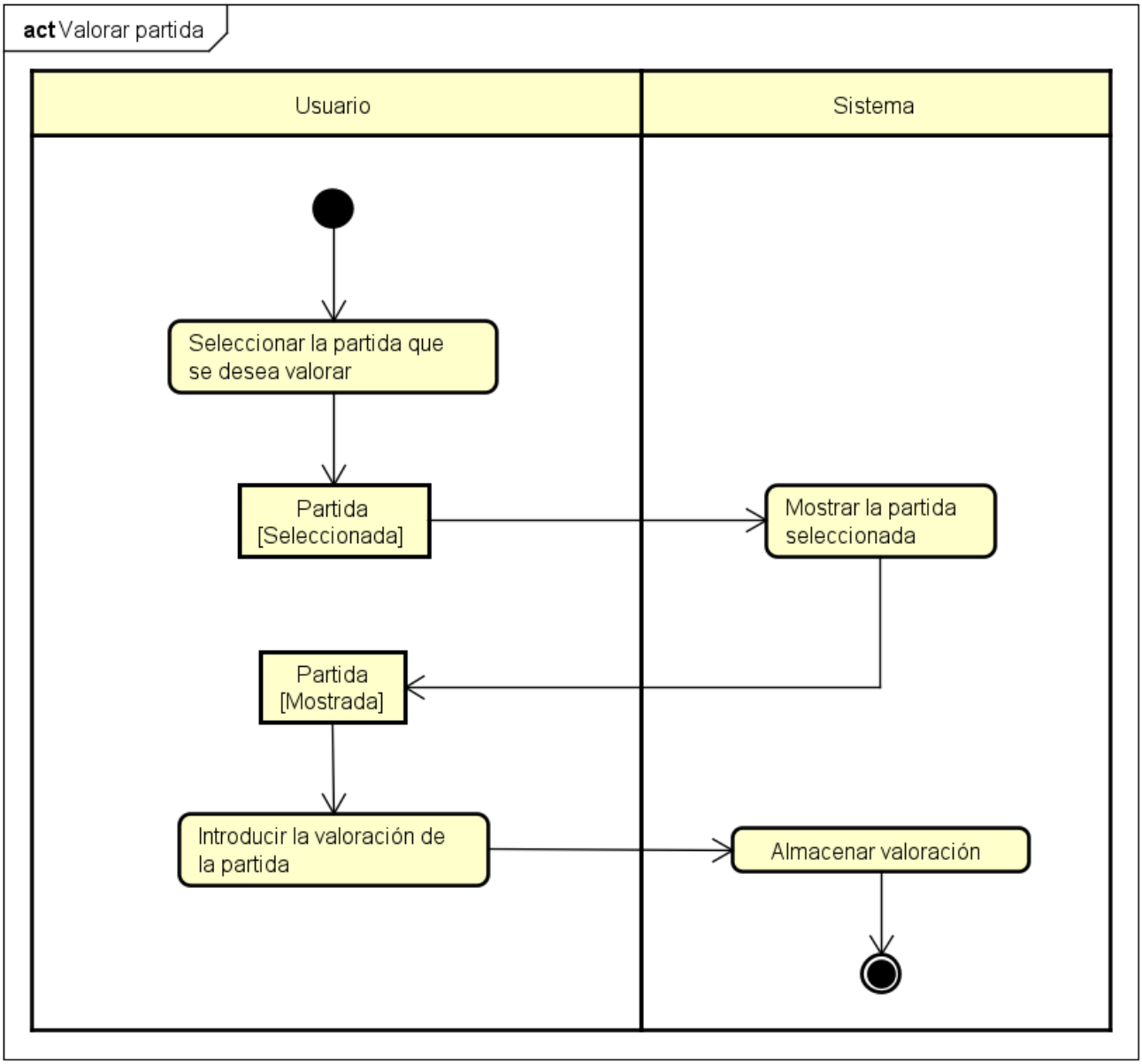


Figura 4.10: Diagrama de actividad correspondiente al CU08

CU09. Dar *like*

El usuario selecciona la partida a la que desea dar *like*, el sistema muestra esta partida y el usuario realiza la acción necesaria para dar *like*. Finalmente, el sistema almacena el *like* del usuario.

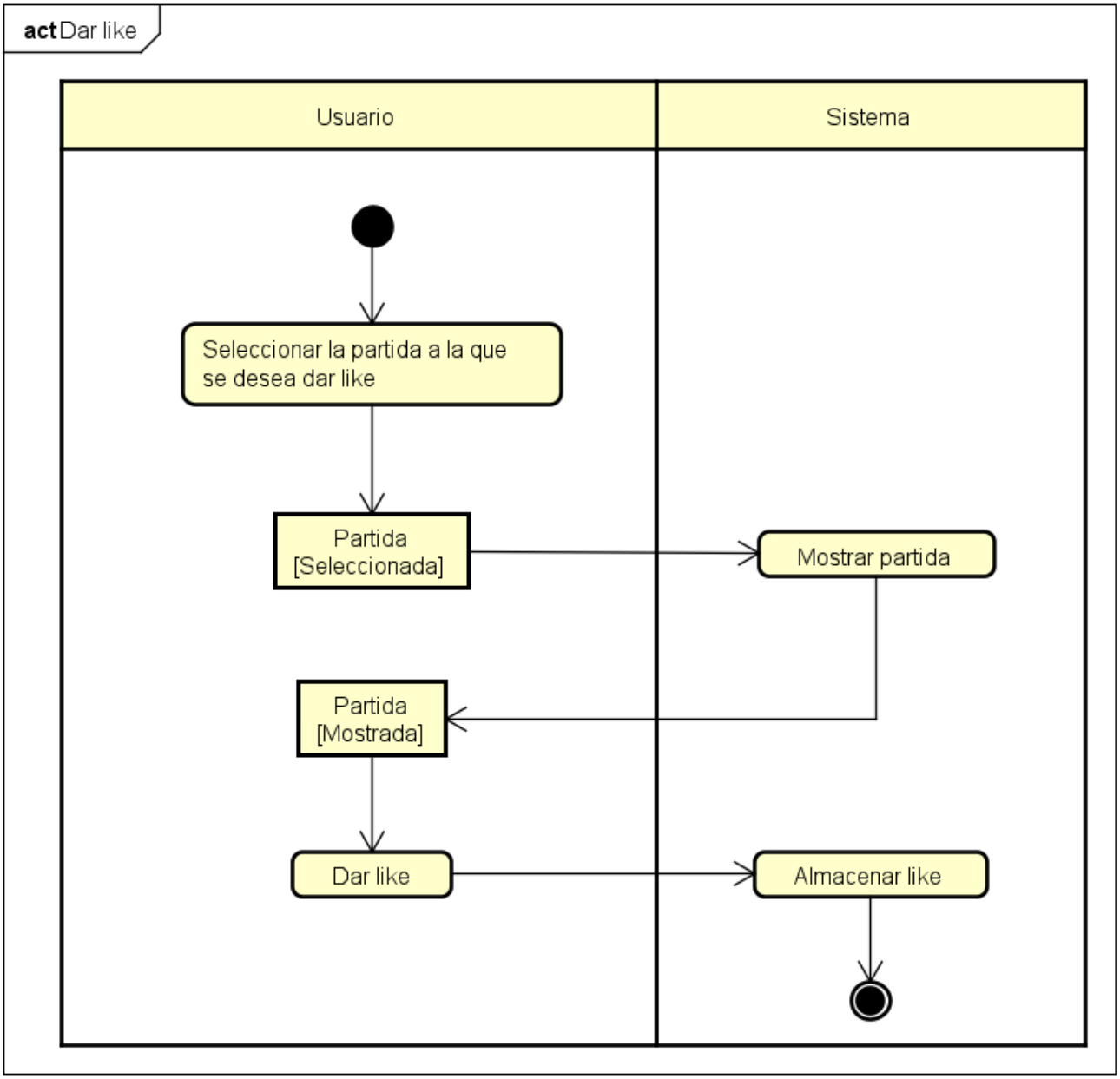


Figura 4.11: Diagrama de actividad correspondiente al CU09

4.2. Estructura de los proyectos

En esta sección se muestra tanto la estructura final del proyecto Flutter, como la de Node.js. La Figura 4.12 muestra la vista de proyecto Flutter disponible en el IDE Android Studio. Esta es una vista general de todas las capas del proyecto. De la misma forma se muestra la estructura general del proyecto Node.js que proporciona Visual Studio Code. A continuación se desarrolla más detalladamente le contenido de cada una de ellas.

4.2.1. Estructura del proyecto front-end

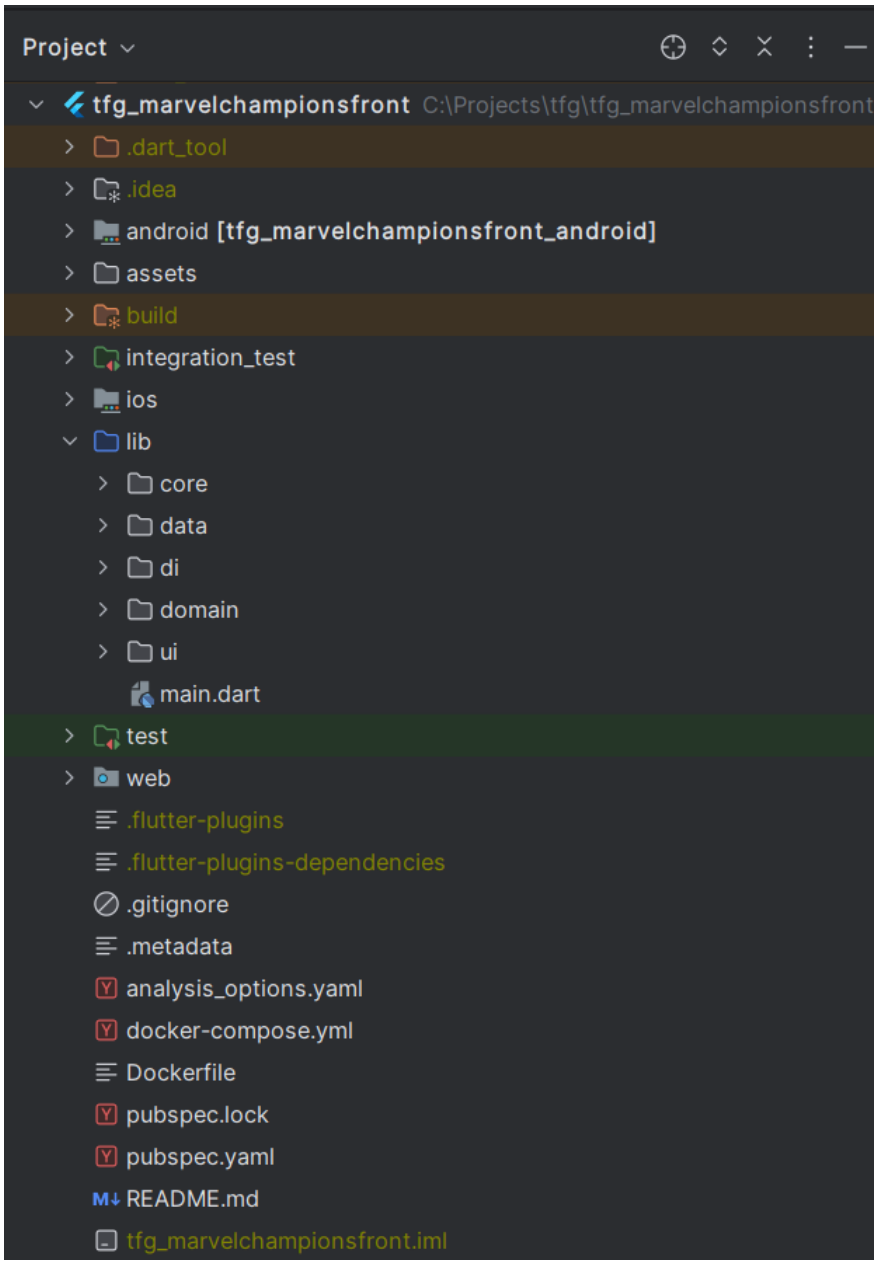


Figura 4.12: Estructura general del proyecto Flutter desde Android Studio

En el proyecto Flutter todo el código esta contenido en la carpeta *lib*. Se trata de código compartido tanto para Android, web e iOS incluyendo las vistas. En su interior se encuentran cuatro directorios: *core*, *di data*, *domain*, y *ui*, además del fichero *main.dart* que es el punto de entrada de la aplicación.

También se pueden observar diferentes directorios para cada una de las plataformas en las que puede ejecutarse la aplicación, *android*, *ios*, y *web*. Estos directorios contienen archivos y configuraciones relacionados con la implementación y ejecución de la aplicación en las diferentes plataformas. Estos directorios son importantes porque permiten que una aplicación Flutter se implemente y ejecute en múltiples plataformas sin tener que escribir código específico para cada una de ellas.

El directorio *android* contiene ficheros como el *AndroidManifest.xml* que contiene la configuración de la aplicación en Android, así como archivos relacionados con la configuración de Gradle, que se utiliza para compilar y construir la aplicación Android.

El directorio *ios* contiene los ficheros necesarios para la implementación de la aplicación en dispositivos iOS, incluyendo el archivo *Runner.xcodeproj*. Se trata de un proyecto Xcode que se utiliza para compilar y ejecutar la aplicación en dispositivos iOS.

El directorio *web* contiene los archivos relacionados a la implementación de la aplicación en la web utilizando Flutter. Contiene archivos de configuración específicos de la web, como *index.html*, así como ficheros JavaScript y CSS necesarios para ejecutar la aplicación en un navegador web.

También podemos observar el directorios *assets* que contiene las imágenes estáticas utilizadas en la aplicación, como puede ser el logo, imágenes predeterminadas, etc. Y también contiene las diferentes tipografías utilizadas en la aplicación.

Los directorios *test* y *integration_test* contienen respectivamente las pruebas unitarias y las pruebas *end-to-end* de la aplicación. En caso de haber pruebas de integración deberían estar en el directorio *integration_test*, pero como ya se explica más adelante en el Capítulo 5, solo se han realizado una prueba unitaria y una prueba *end-to-end*.

Se pueden observar las carpetas *.dart-tools*, *.idea* y *build* que son generados automáticamente por el propio entorno de desarrollo. La carpeta *.dart-tools* es utilizado por Dart [36] y contiene archivos utilizados por varias herramientas de Dart. La carpeta *.idea* contiene varios archivos xml con ciertas configuraciones del proyecto. Y el directorio *build* es donde se generan o se crean todos los archivos compilados necesarios para ejecutarse correctamente en las diferentes plataformas [37].

También, se encuentran los ficheros *pubspec.yaml* y *pubspeck.lock* que contienen las configuraciones del proyecto, así como las versiones de las diferentes dependencias externas, la ruta a la carpeta o carpetas que contienen las imagenes o las fuentes utilizadas en la aplicación [38]. Y *analysis_options.yaml*, este es un archivo que determina cuán estricto debe ser Flutter a la hora de analizar el código [39].

Finalmente, podemos observar los ficheros *docker-compose.yml* y *Dockerfile*. Estos ficheros se utilizan para construir, configurar y ejecutar aplicaciones en contenedores Docker. Concretamente, el fichero *Dockerfile* contiene instrucciones para construir una imagen de contenedor. Estas instrucciones incluyen la ejecución de comandos que deben realizarse dentro del contenedor, la exposición de puertos, la copia de archivos, entre otras cosas. El fichero *docker-compose.yml* sirve para definir y coordinar varios servicios o contenedores. De esta manera, un *docker-compose* permite orquestar múltiples contenedores a través de un solo fichero. En este proyecto front-end, el fichero *docker-compose.yml* únicamente está presente para facilitar el despliegue de esta parte de la aplicación, ya que con este fichero el despliegue del front-end se reduce a la ejecución de un único comando, como se describe en el Apéndice D.

Respecto a las cuatro capas contenidas dentro del directorio *lib*, su contenido será explicado más detalladamente en las siguientes secciones.

4.2.2. Estructura del proyecto back-end

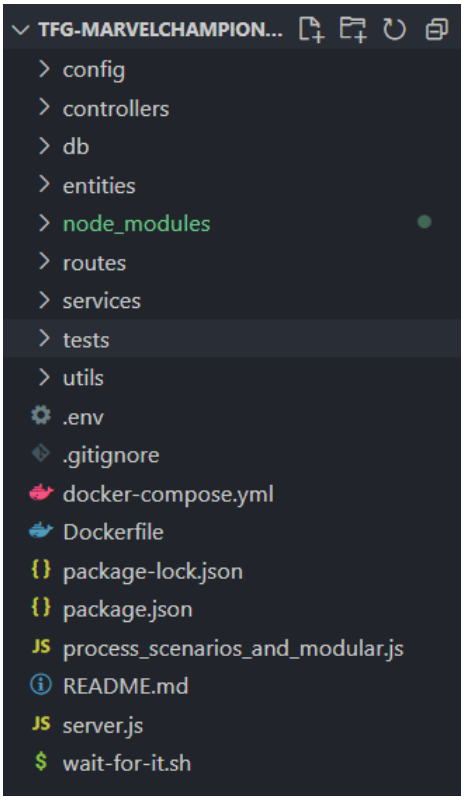


Figura 4.13: Estructura general del proyecto Node.js desde Visual Studio Code

En la estructura del proyecto Node.js mostrado en la Figura 4.13 se pueden observar nueve directorios y nueve ficheros a nivel de proyecto. Si observamos los ficheros, están el fichero *server.js* que es el punto de entrada principal de la aplicación. Incluye lógica para inicializar y configurar la aplicación: configuración de enrutamiento, importación de módulos externos utilizados durante la ejecución de la aplicación, configuración de la especificación swagger [40] del proyecto y configuración de *middleware* de autenticación.

Los ficheros *package.json* y *package-lock.json* contienen las dependencias externas utilizadas en el proyecto, sus *scripts* de ejecución, la versión y el autor [41].

El fichero *process_scenarios_and_modular.js* se trata de un *script* cuya función es obtener todos los escenarios y conjuntos modulares (e información relevante de estos) de la API de MarvelCDB y volcarlos en la base de datos local. Este *script* es necesario ya que la forma en que la API de MarvelCDB proporciona los datos necesarios para este proyecto no son accesibles de una forma sencillas como endpoints o similar. De esta forma, los escenarios y conjuntos modulares son almacenados en la base de datos local y más accesibles para el proyecto front-end.

También encontramos el fichero *.env* que contiene las variables de entorno utilizadas en el fichero *docker-compose.yml* explicado más adelante. Y, el fichero *.gitignore* que contiene los ficheros y directorios que deben ser ignorados por el sistema de control de versiones Git [27].

Y, por último, los ficheros *docker-compose.yml* y *Dockerfile*. Estos ficheros han sido explicados en

la anterior Sección 4.2.1. En este caso, el fichero *docker-compose.yml* contiene la coordinación del servicio de la base de datos con el servicio de la API, y el fichero *Dockerfile* contiene la creación de la imagen de contenedor de la API. Además, se ha añadido el fichero *wait-for-it.sh*, se trata de un ejecutable que sirve para sincronizar servicios. Específicamente, hace que un servicio espere a que otro esté disponible antes de proceder con la ejecución de alguna instrucción. En este caso se utiliza para que el servicio de la API espere a que el servicio de la base de datos esté disponible.

Respecto a los diferentes directorios, tenemos:

- *config*, que contene ficheros para la conexión a la base de datos local, variables de entorno, claves para encriptar información sensible almacenada en la base de datos, y la configuración de la autenticación JWT.
- *db*, que únicamente contiene el *script* SQL de la base de datos.
- *utils*, que contiene operaciones que son compartidas entre diferentes partes del proyecto.
- *tests* contiene las pruebas de integración realizadas de la API.
- *node_modules* contiene los módulos necesarios para el proyecto.

El resto de directorios serán explicados más detalladamente en la siguiente sección, al igual que las capas del proyecto Flutter.

4.2.3. Arquitectura del sistema al completo

La arquitectura utilizada para la organización del sistema en general, teniendo en cuenta los dos proyectos juntos es la **Arquitectura Limpia** o ***Clean Architecture***.

La arquitectura limpia [42] es un conjunto de principios y patrones de diseño que deben facilitar el proceso de construcción software, así como su mantenimiento y escalado. La idea es que cada nivel tiene su responsabilidad y se comunica únicamente con sus niveles inmediatamente contiguos.

Las capas en esta arquitectura se organizan de forma jerárquica unas encima de otras y las dependencias siempre van hacia abajo. Es decir, que una capa dependerá solamente de las capas inferiores, pero nunca de las superiores como se muestra en la Figura 4.14.

Las diferentes capas utilizadas en esta arquitectura [43] son:

- **Entidades**: son el conjunto de reglas comerciales relacionadas que son críticas para el funcionamiento de una aplicación. Así como objetos de dominio o modelos de datos fundamentales. Son completamente independientes de cualquier otro elemento dentro de la arquitectura.
- **Casos de uso**: encargada de representar las acciones que un usuario puede realizar en la aplicación. Los casos de uso interactúan con las entidades y dependen de ellas, pero no saben nada sobre las capas externas.

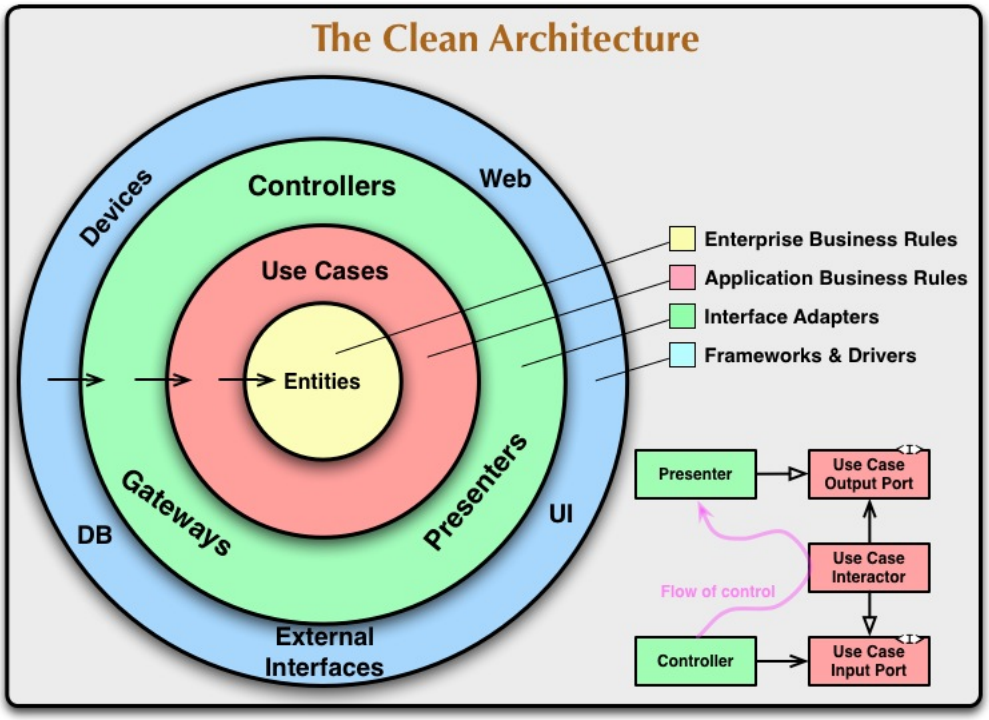


Figura 4.14: Diagrama Arquitectura Limpia

- **Adaptadores de interfaz**: proporcionan un puente entre el mundo exterior y los casos de uso y las entidades.
- **Infraestructura**: representa a los agentes externos. En esta capa es donde van los componentes de Entrada/Salida: la interfaz de usuario, la base de datos, los marcos y los dispositivos.

Las capas de entidades, de casos de uso y de adaptadores de interfaz se encuentran en el proyecto back-end: la capa de entidades se corresponde con el directorio *entities* mostrado en la Figura 4.13, ya que contiene el modelo de datos de la aplicación. La capa de casos de uso se corresponde con el directorio *services* ya que aquí se representan las diferentes tareas o acciones que un usuario puede realizar en la aplicación. Y la capa de adaptadores de interfaz se corresponde con el directorio *controllers* porque proporciona un puente entre la capa de infraestructura que mencionaremos más adelante, y los casos de uso y las entidades.

La capa de infraestructura se corresponde con el proyecto front-end al completo, ya que este es el que muestra toda la información proporcionada por las entidades a través de las diferentes acciones que puede realizar el usuario (casos de uso).

Arquitectura del proyecto Flutter

Se ha utilizado también la **Arquitectura Limpia** para este proyecto, ya que aporta separación de responsabilidades, facilidad a la hora de realizar pruebas, escalabilidad y mantenibilidad.

Dentro de este proyecto, cuya estructura se muestra en la Figura 4.12, el directorio *data* se corresponde con la capa de adaptadores de interfaz, ya que se encarga de interactuar con fuentes de datos externas, en este caso con la API proporcionada por el proyecto Node.js.

El directorio *domain* contiene tanto la capa de entidades como la capa de casos de uso dentro de este, ya que es este directorio el que contiene el modelo de datos y los casos de uso que se pueden realizar en la aplicación.

El directorio *ui* contiene las vistas de la aplicación y los controladores de estas vistas. Se corresponde con la capa de infraestructura, ya que las vistas son las interfaces de usuario de la aplicación.

La capa *data* se encarga de obtener los datos comunicándose con la API proporcionada por el proyecto Node.js. La capa *domain* engloba toda la lógica de negocio de la aplicación Flutter, y es donde se encuentra el código que debe ser agnóstico de cualquier otra parte de la app. Esta accede a la capa *data* y transforma esa información en el modelo de datos de la aplicación. Y por último, los controladores dentro de la capa *ui* se encargan de llamar a los casos de uso cuando es necesario y de proporcionar la información que deban mostrar las vistas.

Arquitectura del proyecto Node.js

La arquitectura utilizada para este proyecto ha sido la arquitectura ***Controller-Service-Repository***.

El objetivo de la arquitectura *controller-service-repository* [44] es la separación de preocupaciones. Esta es bastante simple, ya que únicamente hay 3 capas: la capa *controller*, es la única responsable de exponer la funcionalidad para que pueda ser consumida por entidades externas. La capa *service* es la que contiene toda la lógica de negocio, accediendo así al repositorio adecuado cuando necesite recuperar o almacenar información. Y por último, la capa *repository*, que se encarga de almacenar y recuperar conjuntos de datos de una fuente de datos existente.

Dentro del proyecto Node.js mostrado en la Figura 4.13, el directorio *controllers* se corresponde con la capa *controller* de esta arquitectura explicada anteriormente, ya que es la encargada de exponer la funcionalidad para que pueda ser utilizada por otras entidades. En este caso, esta capa es utilizada por el directorio *routes*. Esta es una capa adicional que define como responderá el servidor a las diferentes solicitudes HTTP que reciba, y responde llamando a la funcionalidad proporcionada por la capa *controllers*.

El directorio *services* se corresponde con la capa *service* de la arquitectura y contiene la implementación de la lógica de negocio de la aplicación.

Y finalmente, el directorio *entities* se corresponde con la capa *repository*, ya que es el responsable de almacenar y recuperar información, en este caso, de la base de datos MySQL. Sin embargo, esta es una capa DAO ya que esta es la que realiza las operaciones necesarias de acceso a la base de datos. Esto último se explica al completo en la siguiente Sección 4.3.2.

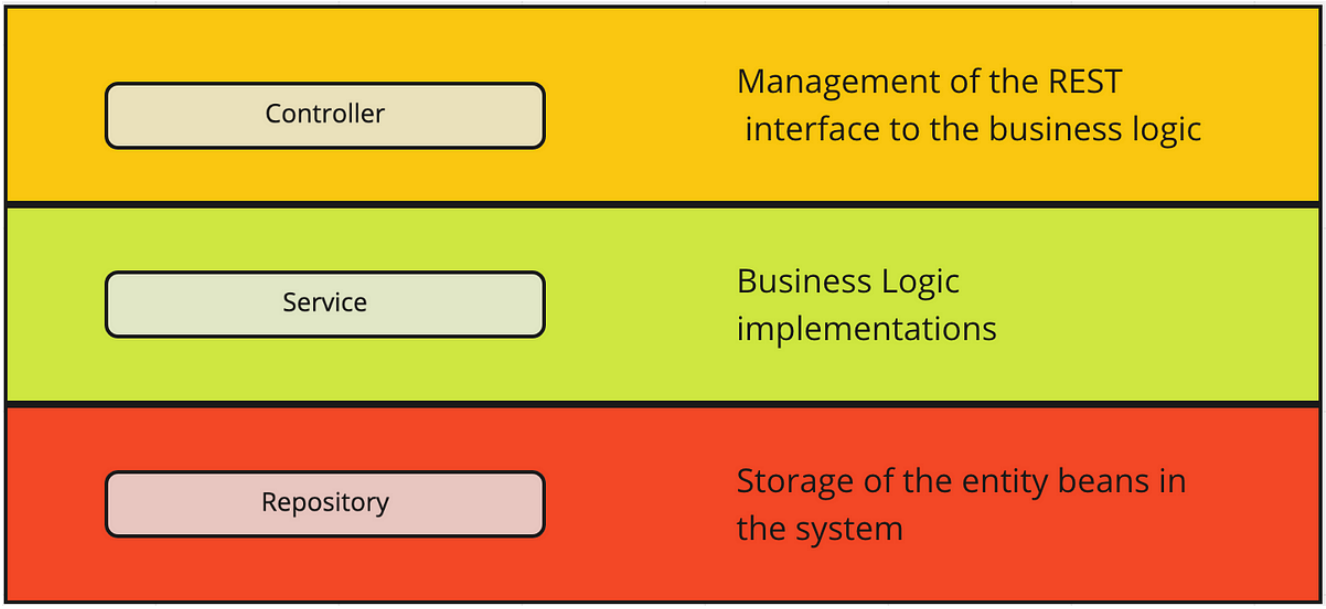


Figura 4.15: Diagrama Arquitectura *Controller-Service-Repository*

4.3. Diseño

Esta sección describe las técnicas y patrones de diseño aplicados en el proyecto. Se añaden también los distintos diagramas de paquetes y de despliegue.

4.3.1. Diseño de datos

En la Figura 4.16 se muestra el diagrama entidad-relación de la base de datos generado por la herramienta MySQL Workbench. Los tipos de líneas que representan las diferentes relaciones son:

- Línea sólida: representa una relación uno a muchos (1:n) entre dos entidades. Esto significa que una entidad en el extremo uno de la línea puede estar relacionada con cero o más entidades en el extremo muchos de la relación.
- Línea punteada: representa una relación opcional entre dos entidades. Esto significa que la existencia de una entidad en el extremo opcional de la línea no es obligatoria.
- Línea doble: representa una relación de muchos a muchos (n:m) entre dos entidades. Esto indica que múltiples entidades en ambos extremos de la relación pueden estar relacionados entre sí. Este tipo de relación no está presente en el diagrama.

Las claves primarias de las entidades se identifican en el diagrama a través de una llave de color amarilla. Y en caso de claves primarias compuestas, se identifican por una llave de color naranja (en cada campo que compone la clave se muestra este icono).

Y por último, las claves foráneas se representan en el diagrama de dos formas: a través de un rombo naranja, y a través de la línea que representa la relación. En el extremo de la relación cuya forma es

similar a un triángulo, se trata de la entidad que contiene la clave foránea de un campo de la entidad en el otro extremo de la relación (similar a dos líneas paralelas perpendiculares a la línea que indica la relación).

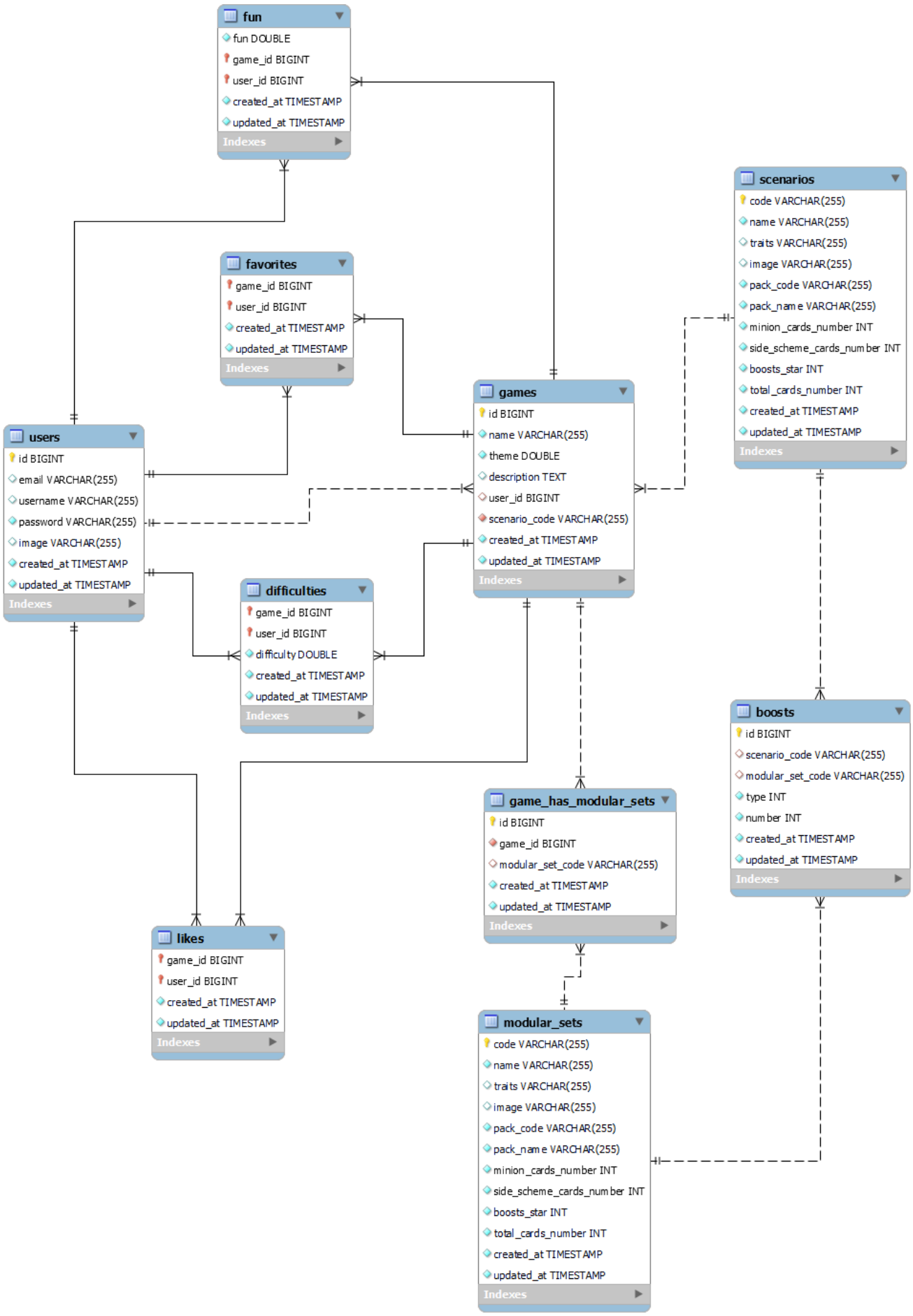


Figura 4.16: Diagrama Entidad-Relación

4.3.2. Patrones de diseño

Los patrones de diseño [45] son soluciones habituales a problemas que ocurren con frecuencia en el diseño de sofware. Son como planos prefabricados que se pueden personalizar para resolver un problema de diseño recurrente en el código.

Los patrones de diseño varían en su complejidad, nivel de detalle y escala de aplicabilidad al sistema completo que se diseña [46]. Existen patrones básicos y de bajo nivel que normalmente se llaman *idioms* que se suelen aplicar a un único lenguaje de programación. Y otros más universales y de más alto nivel llamados Patrones de Arquitectura. Estos últimos serán los expuestos en esta sección.

Además, los patrones pueden clasificarse por su propósito, habiendo así tres clasificaciones:

- Patrones creacionales: se centran en la creación de objetos e incrementan la flexibilidad y la reutilización de código existente. Dentro de esta categoría existen patrones como Singleton o Factory.
- Patrones estructurales: tratan la manera en que los objetos se conectan con otros objetos, para asegurar que los cambios del sistema no requieren cambiar esas conexiones. Patrones como Adaptador, Compuesto o Fachada se incluyen en esta categoría.
- Patrones de comportamiento: se centran en cómo los objetos interactúan y comunican entre sí, ayudando a definir las relaciones y responsabilidades entre ellos. Dentro de esta categoría se encuentran patrones como Observador, Estrategia o Comando.

A continuación se describen los patrones utilizados en este proyecto.

Patrón Singleton

El patrón Singleton [47] resuelve el problema de la instanciación excesiva de objetos de una misma clase, idénticos entre sí, en distintos puntos de nuestro software. La idea de este patrón es que exista un único objeto de la clase que estamos utilizando y que tiene este problema. En la Figura 4.17 se muestra como funciona este patrón.

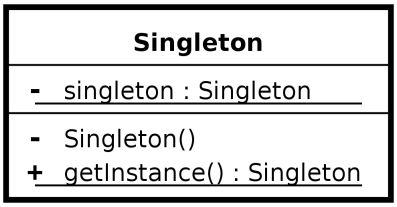


Figura 4.17: Esquema Patrón Singleton [48]

Este patrón ha sido utilizado para diversas cosas de la aplicación. Todos los repositorios, casos de uso y controladores de las vistas implementan este patrón, ya que solo hay una única instancia de cada uno de ellos durante la ejecución de la aplicación. Esto es así dado que las clases repositorio o las de casos de uso únicamente tienen funciones por lo que crear varias objetos de estas clases crearía instancias idénticas, y esto no es necesario.

Que los controladores de las vistas utilicen este patrón es beneficioso ya que de esta forma pueden ser accedidos de manera global. Así, tras realizar una acción en una pantalla que implique un cambio en la información mostrada por otra pantalla, esta se podrá actualizar gracias a la implementación de este patrón.

Por último, existe una única instancia de la clase Usuario que representa el usuario actual en la aplicación si se ha iniciado sesión en ella.

Patrón Observador

El patrón observador [49] permite establecer una relación uno a muchos entre objetos, de modo que cuando un objeto cambia su estado, todos los objetos dependientes son notificados y actualizados automáticamente. Este patrón es útil cuando se necesita notificar automáticamente a los observadores cuando ocurre un cambio de estado.

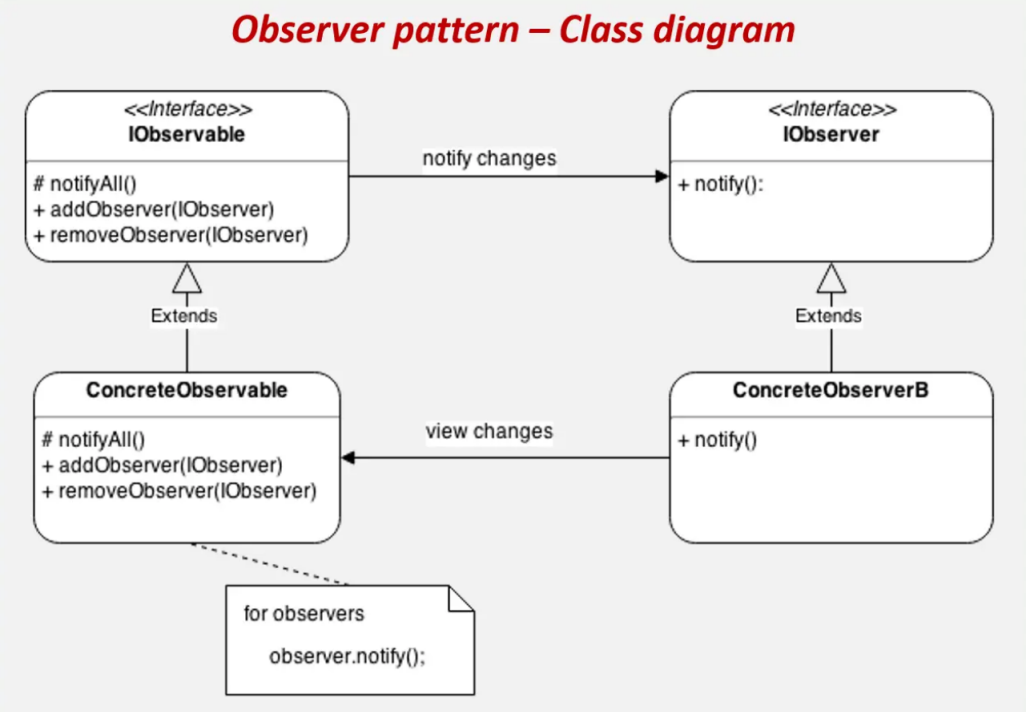


Figura 4.18: Esquema Patrón Observador [50]

Este patrón es implementado a través de los objetos ”Rx<T>” y los widgets ”Obx” de GetX.

- Los objetos ”Rx<T>” [51] son objetos reactivos, es decir, encapsulan un valor que puede ser

observado por otros componentes. De esta manera, cada vez que el valor de un objeto de este tipo cambia, notifica a todos sus observadores sobre el cambio.

- Los widgets ”Obx” [52] permiten a los widgets dentro de este reaccionar automáticamente a los cambios en los valores de los objetos reactivos mencionados anteriormente. Dentro de un ”Obx”, cualquier objeto ”Rx<T>” será automáticamente registrado como observador. Esto significa que el widget se reconstruirá cada vez que el valor de ”Rx<T>” cambie. A continuación se muestra un ejemplo de la utilización de estos widgets junto a los objetos reactivos.

```
@override
  Widget build(BuildContext context) {
    return SafeArea(
      child: Scaffold(
        body: Obx(() => userSession.value.id != null
          ? Stack(
              children: [
                _userInfo(),
                _buttonSignOut()
              ],
            )
          : const AppLoginProcedure(text: AppConstants.LOGIN_PROCEDURE_PROFILE_PAGE)
        )
      ),
    );
  }
```

Figura 4.19: Ejemplo Patrón Observador con widget Obx y objeto Rx<T>

En este ejemplo el objeto reactivo al que el widget ”Obx” está suscrito es *userSession*. En caso de existir el usuario con sesión iniciada en la aplicación se construye el widget *Stack*, en caso contrario se construye el widget *AppLoginProcedure* para indicar al usuario que debe iniciar sesión para usar esa pantalla.

En resumen, los objetos ”Rx<T>” actúan como sujetos u observables en el patrón observador. Son los que mantienen el estado y notifican a los observadores cuando hay un cambio. Los widgets ”Obx” son los observadores. Estos están ”suscritos” a los objetos ”Rx<T>” y se reconstruyen automáticamente cuando los valores de estos objetos cambian.

Patrón Estrategia

El propósito del patrón [53] estrategia es separar algoritmos diferentes en clases diferentes para tener la posibilidad de seleccionar la variante de algoritmo en tiempo de ejecución.

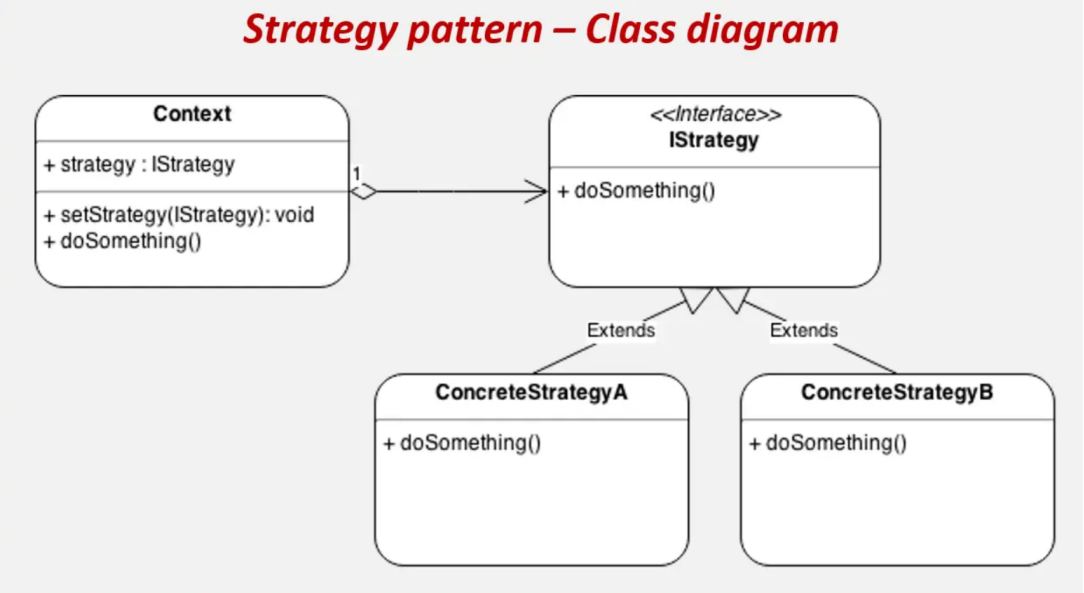


Figura 4.20: Esquema Patrón Estrategia [54]

Este patrón ha sido implementado en la funcionalidad de los filtros de la pantalla principal en el proyecto Flutter. Estos filtros tratan de obtener las partidas ordenadas de diferentes formas, por ello existe una clase abstracta llamada *GetGamesUseCase* y diferentes clases hijas de estas cuyo proposito es obtener las partidas ordenadas de diferentes formas. Por ejemplo, *GetMostDifficultGamesUseCase*, que obtiene las partidas por orden descendiente de dificultad.

El contexto sería el controlador correspondiente a la pantalla de filtros, entonces el usuario al seleccionar una de las maneras de ordenar las partidas se seleccionará la variante del algoritmo correspondiente en tiempo de ejecución, obteniendo así las partidas correspondientes.

Patrón MVVM

La función del patrón MVVM (*Model-View-ViewModel*) [55] es estructurar y separar las funciones de una aplicación, en concreto en el contexto de las interfaces de usuario.

- *Model*: representa la lógica de negocio de la aplicación, encapsulando la funcionalidad de esta. Incluye estructuras de datos, peticiones a través de la red y lógica necesaria para manipular datos. El modelo no tiene conociemiento alguno acerca de la vista, es el *ViewModel* quien hace de intermediario entre ambos.
- *View*: representa la interfaz de usuario de la aplicación. Es responsable de msotrar la información del *ViewModel* u obteniendo las entradas de usuario. Simplemente se limitan a observar los

cambios del *ViewModel* y actualizar los datos mostrados en la interfaz.

- *ViewModel*: es responsable de preparar la información del modelo para ser representada por la vista (*View*), reciben información del usuario desde la vista, la validan y actualizan el modelo en consecuencia.

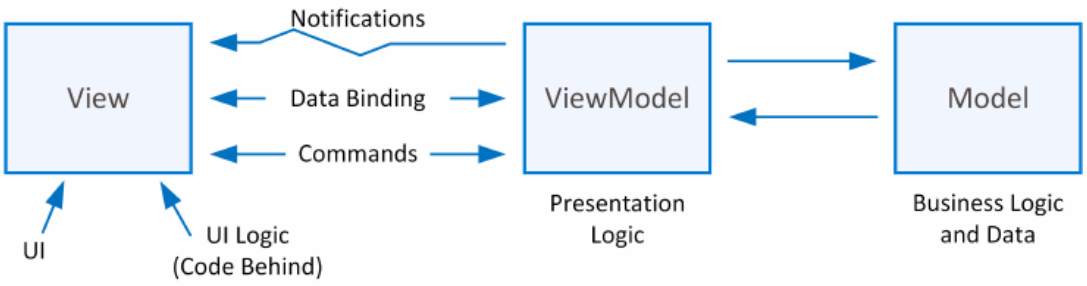


Figura 4.21: Esquema Patrón MVVM [56]

Este patrón se ha implementado en el proyecto Flutter. Dentro de la capa *ui*, los *StatelessWidgets* del directorio *pages* representan las vistas, conteniendo así las interfaces de usuario y la lógica correspondiente de estas. Las clases que heredan de *GetxController* dentro del directorio *controllers* equivalen a los *ViewModel* que reciben la información introducida por el usuario en la vista y actualizan o muestran los datos del modelo. Y por último, la capa *domain* conteniendo los casos de uso y el modelo de dominio de la aplicación, se corresponde con el modelo.

Patrón DAO/DTO

El patrón DAO/DTO [57] propone separar la lógica de negocio de la lógica de acceso a los datos. De esta manera, el DAO proporcionará los métodos de inserción, actualización, obtención y eliminación de datos. Así, para acceder a la fuente de datos, la capa de lógica de negocio solo debe utilizar el DAO. Los DAOs transforman la información obtenida de la fuente de datos en DTOs, estos son objetos que sirven para transmitir la información entre la capa de lógica de negocio y los DAOs. En la Figura 4.22 se muestra un esquema de este patrón.

Este patrón se ha implementado en el proyecto Node.js en la capa *entities*. Esta es una capa DAO, ya que hace de intermediaria entre la lógica de negocio (capa *services*) y la base de datos MySQL. Esta capa devuelve DTOs que son utilizados por la capa *services* y transformados en el modelo de datos de la capa de negocio.

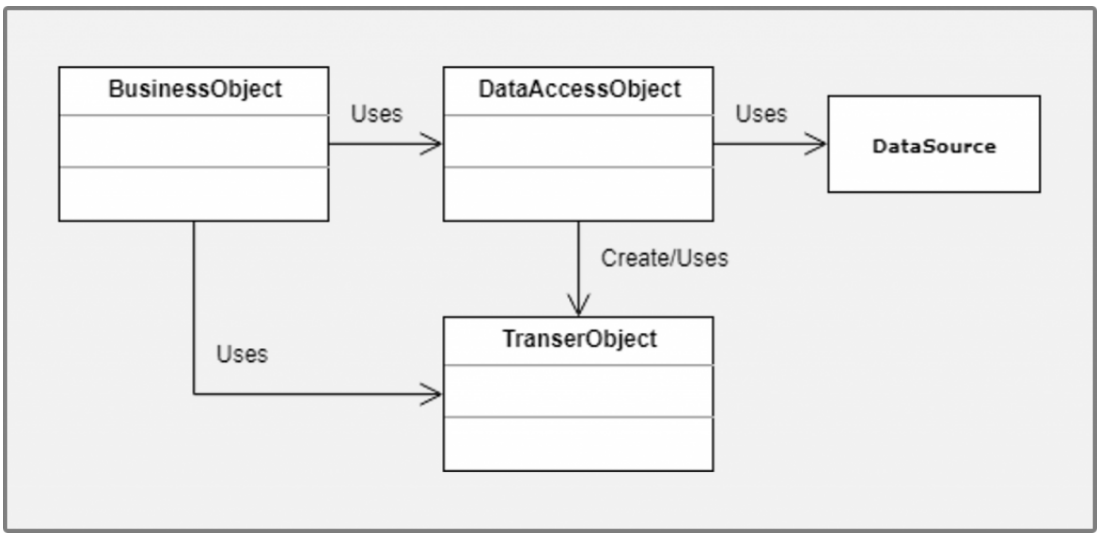


Figura 4.22: Esquema Patrón DAO/DTO [57]

4.3.3. Diagrama de paquetes

Un diagrama de paquetes son diagramas estructurales que representan la organización y disposición de los diferentes elementos de un modelo en forma de paquetes. Proporciona una organización jerárquica de la aplicación. En la Figura 4.23 se muestra el diagrama de paquetes del proyecto Flutter, y en la Figura 4.27 el correspondiente al proyecto Node.js.

Se puede observar la división en capas de la arquitectura limpia explicada anteriormente. Además de las capas *ui*, *domain*, *data*, *core* y *di* explicadas en la Sección 4.2.3, se observan varios paquetes de color blanco. Estos son paquetes externos a la aplicación.

El paquete *ui*, ver Figura 4.24, incluye cuatro paquetes:

- *pages*: incluye las vistas de la aplicación.
- *controllers*: contiene los controladores de las vistas.
- *widgets*: contiene los *widgets* utilizados por varias vistas.
- *utils*: incluye operaciones y clases de utilidad para los diferentes paquetes internos de la capa *ui*.

El paquete *domain*, ver Figura 4.25, contiene tres paquetes:

- *use_cases*: incluye las diferentes tareas que el usuario puede realizar en la aplicación.
- *model*: contiene las clases que representan el modelo de dominio de la aplicación.
- *repositories*: contiene los repositorios utilizados por los casos de uso para realizar las correspondientes tareas. Se trata de interfaces (en dart clases abstractas ya que no dispone de interfaces, si no que se utiliza la palabra reservada *implements* para que una clase sobreescriba la definición de otra) que son implementadas en la capa *data*.

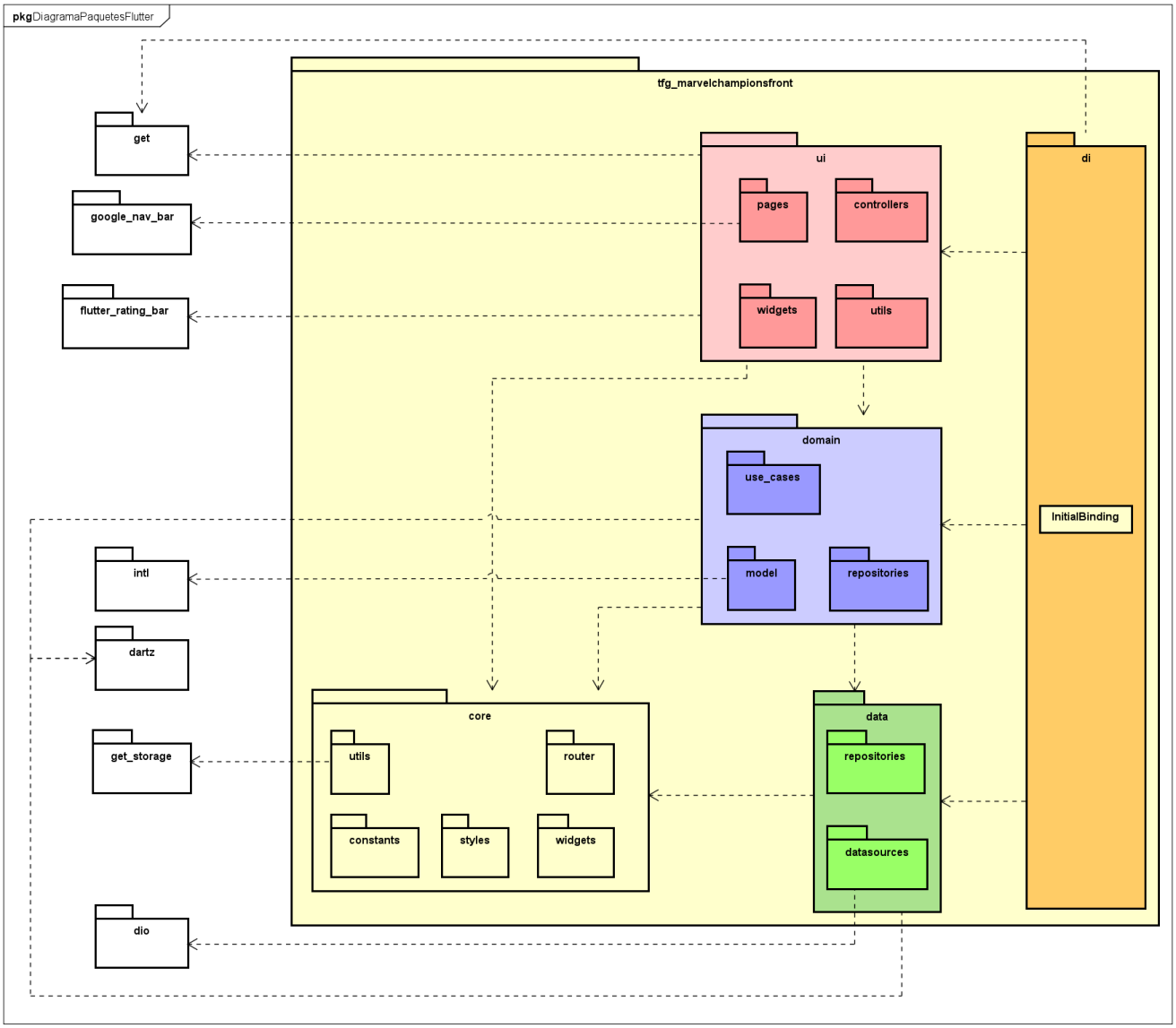


Figura 4.23: Arquitectura de la aplicación front-end

El paquete *data*, ver Figura 4.26, es la capa que accede al proyecto back-end, e incluye dos paquetes:

- *repositories*: contiene las implementaciones de los repositorios de la capa *domain*.
- *datasources*: contiene las clases que acceden al proyecto back-end a través de peticiones HTTP.

El paquete *core* incluye código de utilidad para las anteriores capas, incluyendo así cinco paquetes:

- *utils*: incluye clases utilizadas por las capas del proyecto mencionadas anteriormente y el usuario guardado en sesión.
- *constants*: contiene las constantes tanto numéricas como de cadenas utilizadas en la aplicación.
- *styles*: contiene los diferentes estilos de texto utilizados por la capa *ui* para mostrarlos a través de las vistas y *widgets*.

- *router*: contiene el enrutamiento de las diferentes vistas de la aplicación.
- *widgets*: contiene *widgets* más generales que los de la capa de *ui* utilizados en esta misma capa. Por ejemplo, campos de texto, botón para volver hacia atrás, botón para aceptar, etc.

Y, por último, el paquete *di* que contiene la clase *InitialBinding*. Esta se utiliza para la inyección de dependencias y la inicialización de controladores al inicio de la aplicación. Permite registrar todas las dependencias cuando la aplicación se inicia, asegurando que estén disponibles en toda la aplicación desde el primer momento.

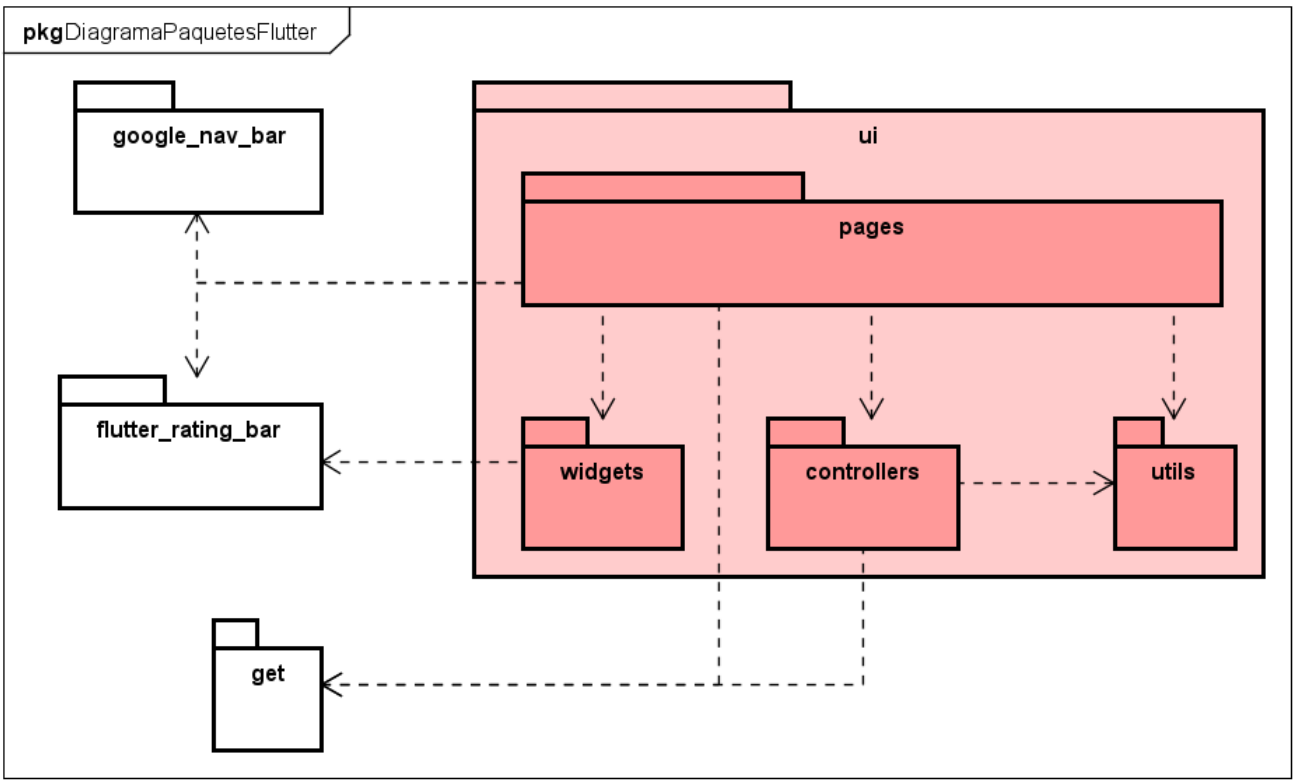


Figura 4.24: Diagrama de paquetes de la capa *ui*

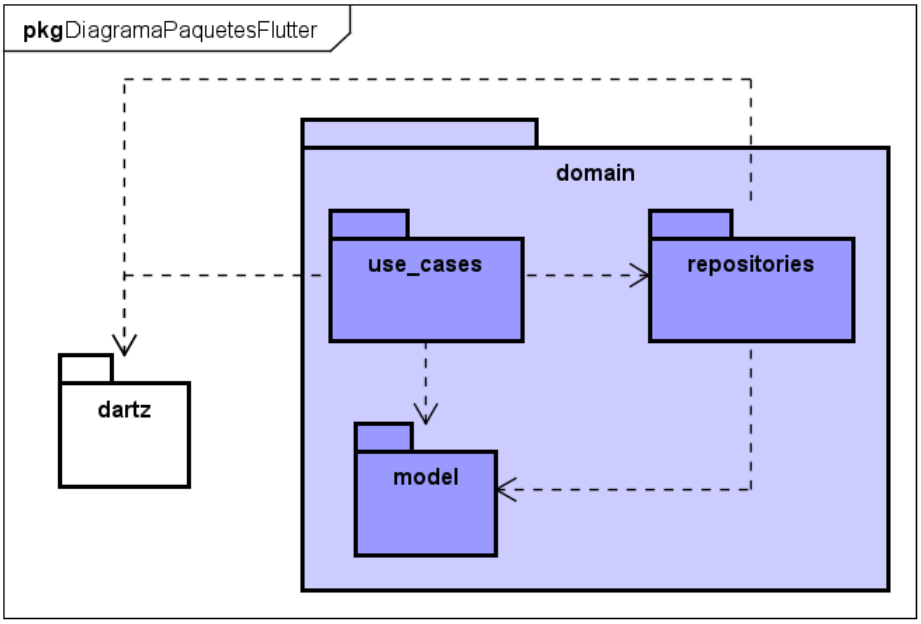


Figura 4.25: Diagrama de paquetes de la capa *domain*

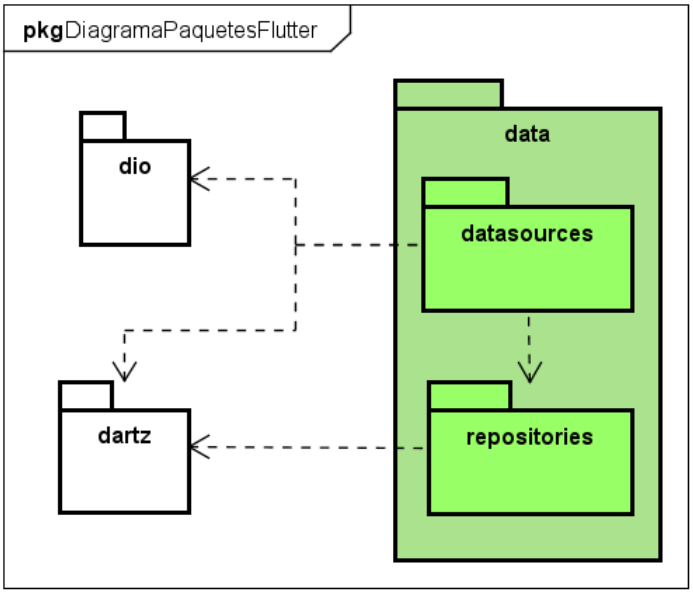


Figura 4.26: Diagrama de paquetes de la capa *data*

En la Figura 4.27 se pueden ver los paquetes más importantes del proyecto Node.js. Como ya se ha mencionado anteriormente, esta sigue el patrón de arquitectura *Controller-Service-Repository*, que se puede ver perfectamente en este diagrama.

El paquete *routes* contiene varias clases que gestionan las rutas para la API REST.

El paquete *controllers* contiene los controladores, que son llamados por las clases que gestionan las rutas cuando se realiza una petición a estas. Obtienen la información necesaria de la petición (si es que la hay), llaman al servicio que corresponda y devuelven la información que debe devolver la ruta en

formato JSON.

El paquete *services* contiene la implementación de la capa de negocio. Las clases de este paquete llaman a las clases del paquete *entities* para realizar diferentes acciones con la fuente de datos. También tiene una dependencia con el paquete *config*, ya que utiliza el fichero de configuración de JWT para generar un *token* JWT en caso de inicio de sesión y registro.

El paquete *entities* incluye clases que almacenan, actualizan, eliminan y obtienen información de la fuente de datos. Utiliza el objeto *db* del paquete *config* que representa la base de datos MySQL.

El paquete *config* contiene ficheros de configuración de JWT, ficheros para la conexión con la base de datos, y ficheros con variables usadas globalmente en el proyecto.

El paquete *utils* contiene operaciones reutilizables de utilidad general en el proyecto. Proporciona funcionalidad común y evita bloques de código repetitivos.

Y por último, el paquete *db*. Este es un paquete completamente independiente del resto, ya que únicamente contiene un script SQL para la creación de la base de datos MySQL.

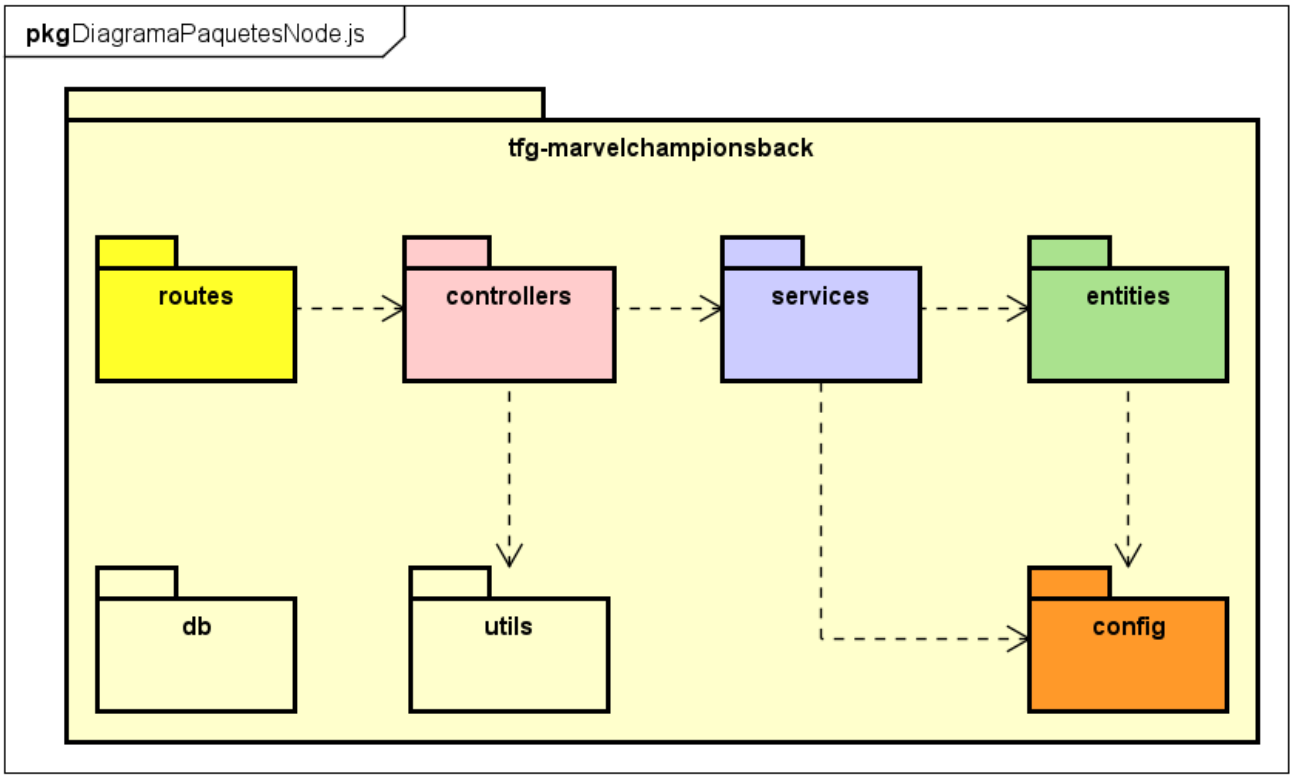


Figura 4.27: Arquitectura de la aplicación back-end

4.3.4. Diagrama de despliegue

Un diagrama de despliegue [58] es un tipo de diagrama que se utiliza para modelar la disposición física de los artefactos software en los componentes de hardware, llamados nodos. Además, represen-

ta las configuraciones físicas de software y hardware esenciales para la ejecución, funcionamiento y escalabilidad del sistema.

En la Figura 4.28 se puede observar el diagrama de despliegue del sistema al completo. En este caso se utiliza la máquina virtual proporcionada por la universidad como servidor que aloja el sistema back-end, y el front-end únicamente para la aplicación web, ya que para los dispositivos móviles solo es necesario generar la aplicación APK e IPA respectivamente. En ella se ha instalado Docker de la misma forma que se indica en el Apéndice D, que es la única herramienta necesaria para desplegar el sistema al completo.

El proyecto Node.js gestiona la conexión con la base de datos de forma automática y expone a la red la API, que puede ser accedida a través de la URL que se indica en el Apéndice D.

El proyecto front-end puede ser ejecutado en dispositivos Android e iOS como aplicaciones APK e IPA respectivamente, y en cualquier dispositivo con cualquier tipo de navegador en el que acceder a la aplicación a través de la correspondiente URL. Esto es posible gracias a que Flutter ofrece la posibilidad de desarrollar la aplicación para múltiples plataformas incluyendo estas tres. Tanto en dispositivos Android como iOS, la aplicación establecerá conexiones HTTP con la API que ofrece el proyecto back-end a través de las cuales realizará el almacenamiento, obtención, actualización y borrado de datos. Respecto a la aplicación web, en la Figura 4.28 se puede observar que se despliega en la misma máquina y es accedida a través de conexiones HTTP a la URL que se indica en el mismo apéndice que se ha mencionado anteriormente.

Se ha desplegado tanto la aplicación Android como la aplicación web, pero no ha sido posible desplegarla para iOS dado que para ello es necesario compilar la aplicación IPA en un ordenador Mac y no se dispone de este.

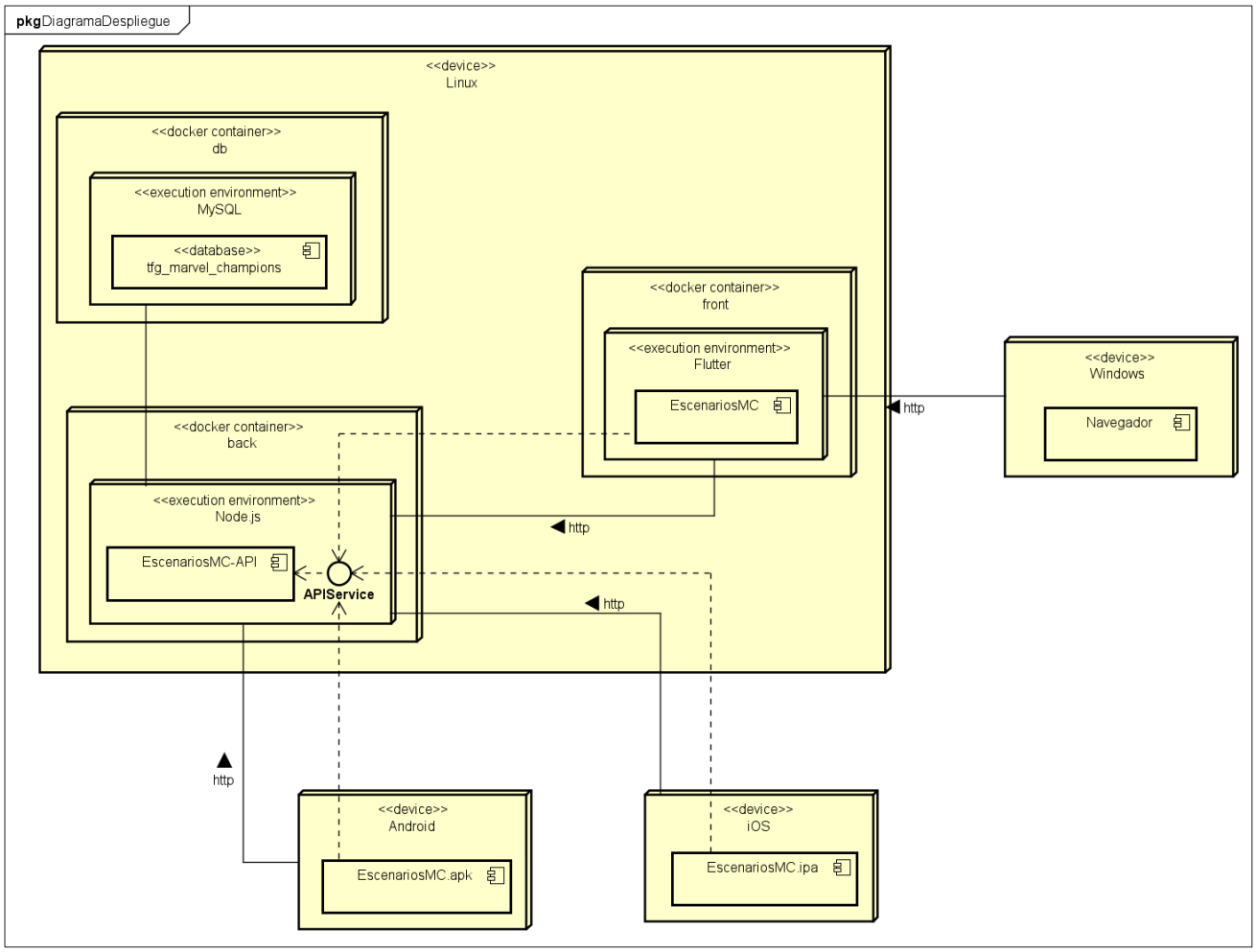


Figura 4.28: Diagrama de despliegue del sistema

Capítulo 5

Pruebas

En este capítulo se comentará la manera de probar ambas partes del proyecto. Debido a la falta de tiempo, se ha optado por hacer un estudio acerca de como se harían los diferentes tipos de pruebas junto a una implementación de cada una de ellas. Probar un sistema completo es una tarea muy extensa en tiempo y recursos. Podría dedicarse casi tanto tiempo como al desarrollo del propio sistema en sí. La idea es realizar pruebas automáticas en las que se utilizan tecnologías que realizan los tests por ti. Estas herramientas pueden ser *flutter_test* [59] para el proyecto Flutter y *jest* [60] para el proyecto Node.js.

Una forma de hacer que una aplicación sea más fácil de probar es escogiendo una arquitectura adecuada para el proyecto. En este caso la arquitectura limpia elegida, como ya se comentó anteriormente, facilita las pruebas, ya que se separan las responsabilidades.

5.1. Pruebas front-end

Para probar una aplicación Flutter, que involucra diferentes plataformas (como ya dijimos antes, en este caso, Android, iOS y web) es necesario utilizar diferentes tipos de prueba para poder probar cada parte de la aplicación. Estas se muestran en la Figura 5.1 y son: pruebas unitarias, pruebas de integración y pruebas *end-to-end* [61].

- Las pruebas unitarias son las encargadas de verificar que componentes o funciones aisladas funcionan como se espera, y se suelen utilizar para probar clases o funciones. Este tipo de pruebas abarcan el 70 % de las pruebas de la aplicación.
- En las pruebas de integración los módulos o componentes software se combinan y prueban en grupo. En estas pruebas cualquier servicio externo se simula (se utilizan *Mocks* [62]) y se necesita un emulador de dispositivo para ejecutar estos tests. Este tipo de pruebas comprenden el 20 % de las pruebas de la aplicación.
- Las pruebas *end-to-end* simulan al usuario real y prueban el flujo de trabajo de una aplicación

de principio a fin. Obviamente, se necesita un emulador para ejecurar estas pruebas. Y nada se simula, se utilizan los servicios y funcionalidades reales, por lo que son muy lentas de ejecutar y a veces ”intermitentes” (algunas veces pasan y otras no). Este tipo de pruebas cubren el 10 % de las pruebas de la aplicación.

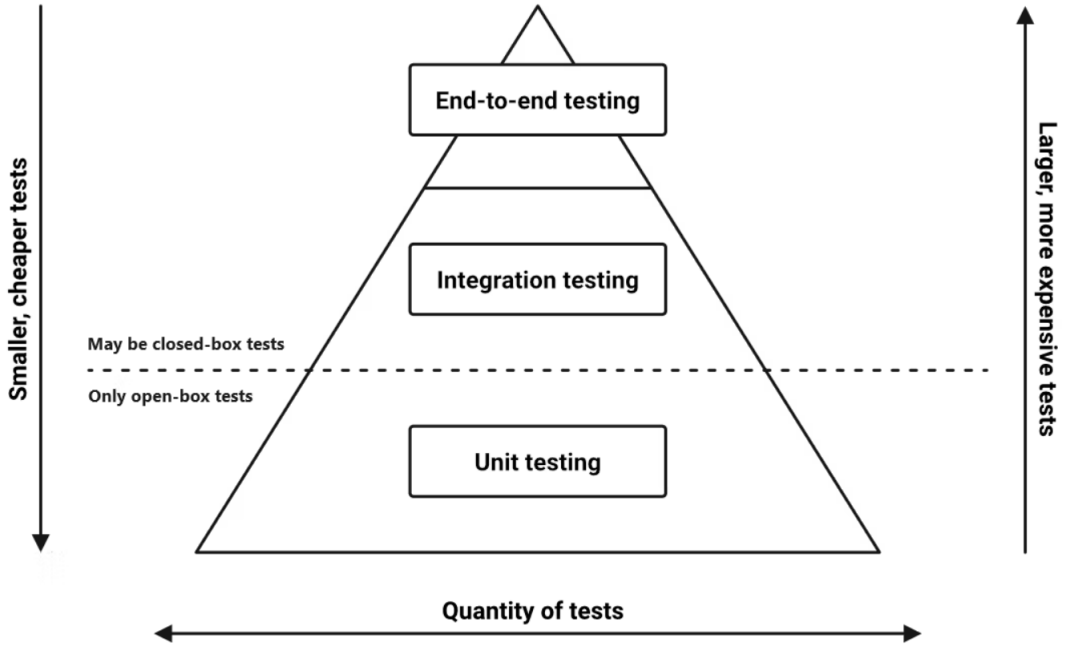


Figura 5.1: Pirámide de pruebas

Como se ha comentado anteriormente, no se ha tenido tiempo para realizar todas las pruebas que necesitaría el proyecto, pero sí se han realizado dos tests para probar las herramientas de prueba de Flutter. Se ha realizado un test unitario y un test *end-to-end*.

La prueba que se muestra en la Figura 5.2 es una prueba unitaria del caso de uso para obtener una partida por su identificador. Se ha empleado un *Mock*, con el paquete externo *mocktail* [63], para simular el comportamiento del repositorio que utiliza el caso de uso. Cabe destacar que tanto la función del repositorio como la proporcionada por el caso de uso son funciones asíncronas, por lo que ha sido necesario utilizar la función ***thenAnswer*** para especificar el valor que devuelve la función del repositorio. También ha sido necesario utilizar la función ***completion*** para especificar que la verificación de si los objetos son iguales se realice tras la espera de la función asíncrona.

La siguiente prueba (ver Figura 5.3) es una prueba *end-to-end* en la que se prueba la tarea de inicio de sesión, involucrando al completo el sistema, incluyendo los dos proyectos realizados. En esta prueba se crea un objeto usuario que posteriormente será comparado con el objeto que devuelve el inicio de sesión de la API para verificar que la tarea se realiza correctamente. Además, esta prueba incluye pasos intermedios, entre estos, la navegación desde la pantalla principal hasta la pantalla de inicio se sesión. Posteriormente, se rellenan los campos necesarios para iniciar sesión (nombre de usuario y contraseña) y se presiona el botón para realizar la acción. Se trata de un test *end-to-end* ya que se prueba el flujo de trabajo del sistema de principio a fin.

```dart
class MockGameRepository extends Mock implements GameRepository {}

void main() {

  late GetGameByIdUseCase getGameByIdUseCase;
  late GameRepository gameRepository;

  late int id;
  late Game game;

  setUp(() {
    gameRepository = MockGameRepository();
    getGameByIdUseCase = GetGameByIdUseCase(gameRepository: gameRepository);

    id = 1;
    game = Game(id: 1);
  });

  test('Get games by id correctly', () {
    when(() => gameRepository.getGameById(id)).thenAnswer((_) async => Right(game));
    expect(getGameByIdUseCase.execute(id), completion(equals(Right(game))));
    verify(() => gameRepository.getGameById(id)).called(1);
  });
}
```

Figura 5.2: Código del test unitario para obtener una partida por identificador

5.2. Pruebas back-end

En la parte del back-end se ha utilizado *Jest* [60] y *Supertest* [64]. *Jest* es un marco de pruebas de Javascript que se utiliza para probar aplicaciones y bibliotecas. *Supertest* es una biblioteca de Node.js que permite realizar solicitudes HTTP a un servidor web y realizar aserciones sobre las respuestas recibidas. Es especialmente útil para realizar pruebas de integración de los endpoints proporcionados por la API. En este caso se ha decidido hacer una prueba de integración que pruebe el funcionamiento de la llamada a la API para obtener una partida por su correspondiente identificador.

En la Figura 5.4 se muestra la prueba de integración. Primero se crea el objeto JSON que debe devolver el endpoint. Y después, dentro de la función ***it***, que representa un test, se hace la correspondiente llamada a la API para obtener la partida por su respectivo identificador. Este debe devolver un código de estado HTTP 201 y debe ser igual al objeto JSON creado anteriormente. Tras la ejecución de los tests dentro de la función ***afterAll*** se cierra el servidor y la base de datos para que *jest* no se quede esperando indefinidamente a que se cierren ambos.

```dart
void main() {
  IntegrationTestWidgetsFlutterBinding.ensureInitialized();

  late User user;

  setUp(() {
    app.main();

    user = User(
      id: 4,
      username: 'oscar',
      email: 'oscar@gmail.com'
    );
  });

  testWidgets('Login user correctly', (widgetTester) async {
    // Wait until all animations are completed
    await widgetTester.pumpAndSettle();

    // Find a widget with this text
    await widgetTester.tap(find.text("Publicar"));
    await widgetTester.pumpAndSettle();

    await widgetTester.tap(find.text("Iniciar sesión"));
    await widgetTester.pumpAndSettle();

    // Find the username text field and add a username
    final usernameTextFieldFinder = find.byWidgetPredicate((widget) =>
      widget is AppTextField && widget.hintText == 'Nombre de usuario'
    );
    await widgetTester.enterText(usernameTextFieldFinder, 'oscar');
    await widgetTester.pumpAndSettle();

    expect(find.text('oscar'), findsOneWidget);

    // Find the password text field and add a password
    final passwordTextFieldFinder = find.byWidgetPredicate((widget) =>
      widget is AppTextField && widget.hintText == 'Contraseña'
    );
    await widgetTester.enterText(passwordTextFieldFinder, 'oscaroscar');
    await widgetTester.pumpAndSettle();

    expect(find.text('oscaroscar'), findsOneWidget);

    // After pressing the button the user that has logged in should be the correct one
    await widgetTester.tap(find.text("Aceptar"));
    await widgetTester.pumpAndSettle();

    expect(userSession.value.id, user.id);
    expect(userSession.value.username, user.username);
    expect(userSession.value.email, user.email);
  });
}
```

Figura 5.3: Código del test *end-to-end* para la tarea de iniciar sesión

```
const request = require('supertest');
const server = require('../../server');
const db = require('../../config/config');

describe('Get games by identifier', () => {
    const game = {...};

    afterAll((done) => {
        server.serv.close(() => {
            db.end(done);
        });
    });

    it('should obtain a game by its identifier', async () => {
        const res = await
request(server.app).get('/api/games/11');
        expect(res.status).toBe(201)
        expect(res.body).toEqual(game);
    });
});
```

Figura 5.4: Código del test de integración del endpoint para obtener una partida por identificador

Capítulo 6

Diseño centrado en el usuario

6.1. Análisis de las necesidades de usuario

Antes de comenzar a describir el proceso de realización de los test de usabilidad, se comentará en detalle el análisis de necesidades de los usuarios. Se desea que el proyecto se aproxime a lo que necesitan los usuarios, por ello se necesita saber lo que creen ellos necesario que deba contener la aplicación.

Se realizó un cuestionario inicial (ver Apéndice B) para dar forma a la aplicación con las ideas sugeridas por los usuarios en este, obteniendo así los requisitos funcionales mostrados en la Tabla 4.1.

6.1.1. Análisis de las respuestas de los usuarios

Tras esperar unas semanas a que el número de respuestas al cuestionario se estabilizara y dejara de subir, se analizaron los resultados obtenidos. A primera vista, se pueden observar varias cosas interesantes. Respecto a las primeras preguntas como la edad y el género, estas no son relevantes para el desarrollo del proyecto.

Los usuarios suelen ser activos jugando a este juego (ver Figura 6.1), lo que indica que la aplicación será bastante utilizada en un futuro. En cuanto a la frecuencia de dificultad jugada (ver Figura 6.2), la mayoría de usuarios se han decantado por la normal, pero si que hay cierta variedad en las respuestas. Por lo que, sería interesante añadir en la aplicación una medida del grado de dificultad de la partida, para que le resto de usuarios pueda verla.

Las respuestas a la razón por la cuál los usuarios eligen al villano cuando van a jugar una partida (ver Figura 6.3) están muy igualadas como se puede contemplar. Esto nos indica que añadir etiquetas o una descripción en las partidas indicando si la partida es temática o si se ha tenido en cuenta la dificultad del villano, etc, creará una mejor experiencia de usuario.

Otra cosa que se puede observar a través de las respuestas de los usuarios (ver Figura 6.4), es en

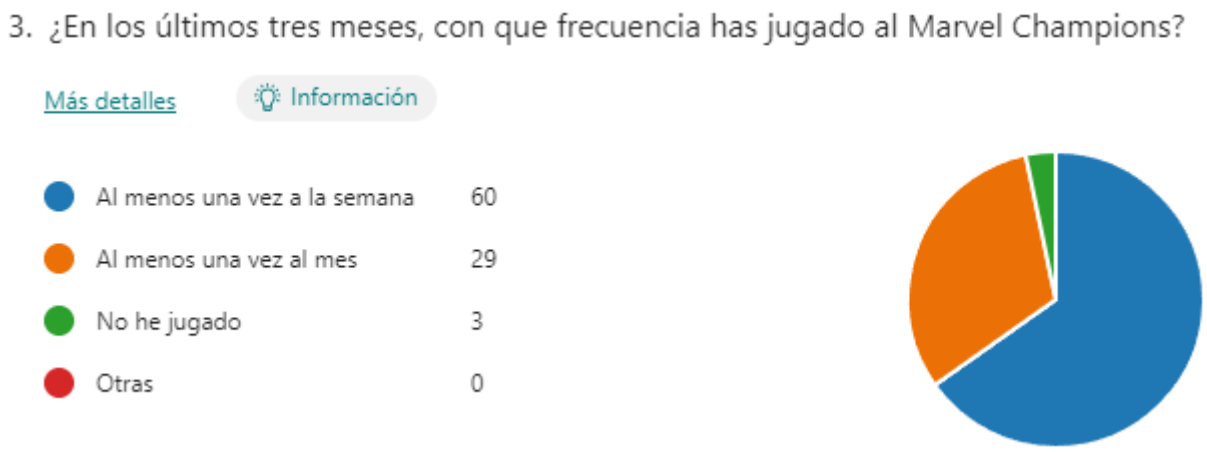


Figura 6.1: Resumen de respuestas a la pegunta 3

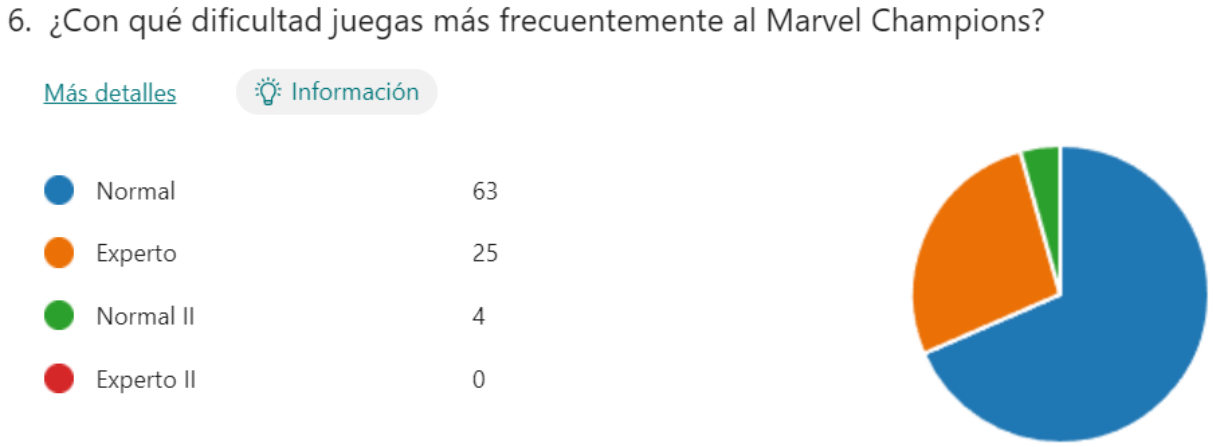


Figura 6.2: Resumen de respuestas a la pregunta 6

que se basa principalmente su elección de los conjuntos modulares. La mayoría de gente opta por elegir los conjuntos modulares recomendados. Aunque, también tienen en cuenta que el villano/a elegido/a sea temático, las mecánicas del villano/a y la dificultad de este. Esto apoya la misma idea anterior de necesitar una descripción en la que el usuario pueda expresar la razón por la que elige los conjuntos modulares en cada partida.

Respecto al tipo de aplicaciones que más utilizan los usuarios, se puede observar que la mayoría de ellos han seleccionado aplicaciones móviles. Por lo tanto, la plataforma objetivo del proyecto sería móvil. Sin embargo, en las respuestas a esta pregunta (ver Figura 6.5), se puede observar que si que hay varias personas que han seleccionado aplicaciones web, por lo que, al ser Flutter una tecnología de desarrollo de interfaces híbrida, las plataformas objetivo serán tanto móvil como web.

La gran mayoría de usuarios utilizan la aplicación *MarvelCDB* para buscar mazos de otros usuarios, cartas, y subir los mazos utilizados (ver Figura 6.6). Por lo tanto, realizar una interfaz de usuario similar es una buena opción. Podemos concluir que los usuarios quieren una interfaz sencilla, sin muchas complicaciones.

Las respuestas más relevantes para este análisis son las de la pregunta 13, en la que los usuarios

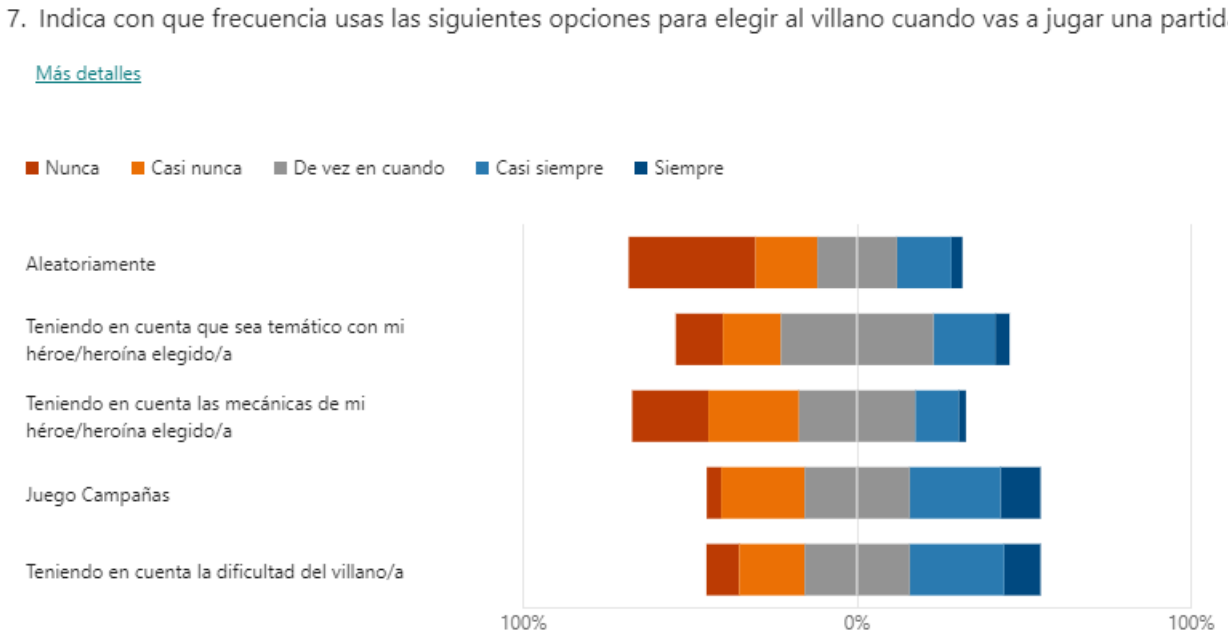


Figura 6.3: Resumen de respuestas a la pregunta 7

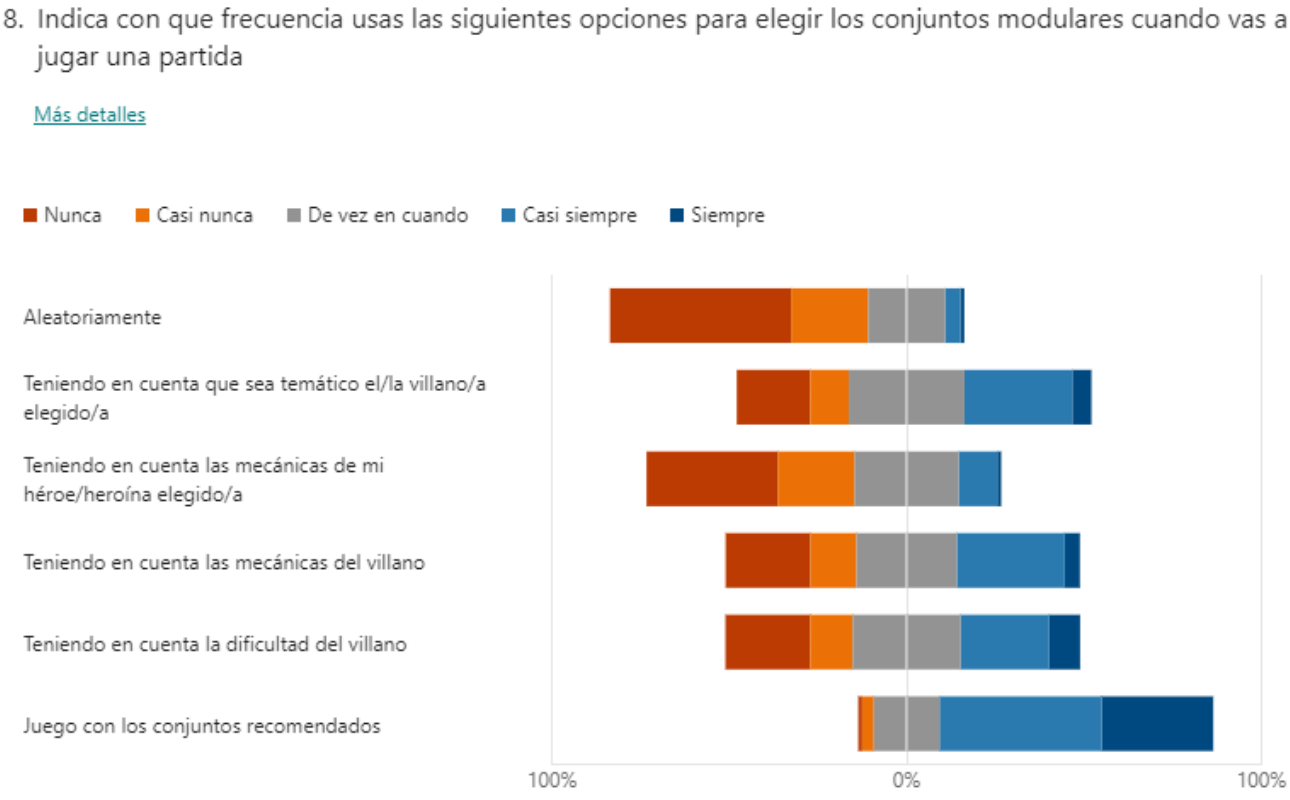


Figura 6.4: Resumen de respuestas a la pregunta 8

contestan lo que creen imprescindible para la aplicación.

Ven necesario establecer un valor de dificultad a las partidas. Como ya se ha dicho antes, añadir una medida del grado de dificultad de la partida sería interesante. También, se ha mencionado en esta

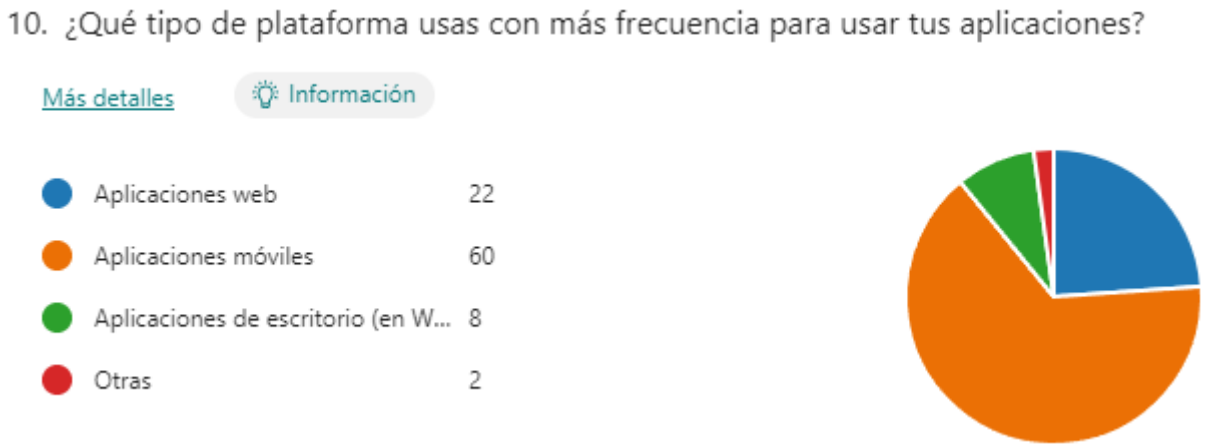


Figura 6.5: Resumen de respuestas a la pregunta 10

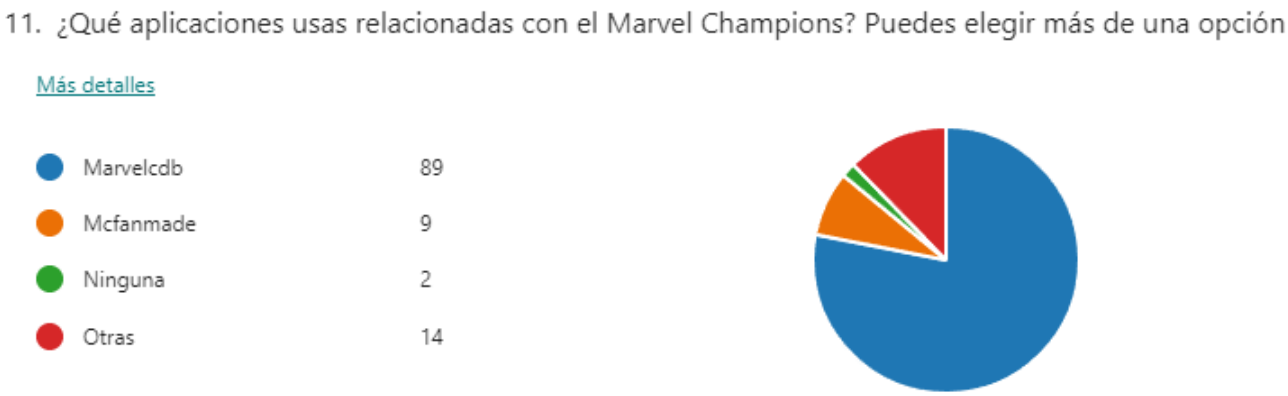


Figura 6.6: Resumen de respuestas a la pregunta 11

pregunta que la comunidad pueda valorar el grado de dificultad y el grado de diversión de la partida, para así obtener una experiencia satisfactoria.

Una de las respuestas más contestadas en esta pregunta es la búsqueda de escenarios, filtrado, ya sea por temática, dificultad, rasgos, etc. Por lo que, tanto la búsqueda de escenarios, como el filtrado de todo tipo de campos será introducido en la aplicación.

También se ha mencionado la posibilidad de que el usuario pueda guardar otras partidas (quizá marcadas como favoritas), y la facilidad de ver la imágen y texto de las cartas.

Por último, se ha mencionado la gestión de campañas. Esta última necesidad no es prioritaria, si no que lo es la gestión de escenarios. Sin embargo, en caso de realizar el objetivo principal de la aplicación antes de tiempo, esta necesidad podría ser desarrollada.

6.1.2. Persona primaria

En esta sección se expone la información obtenida a través del cuestionario en forma de usuario ideal de la aplicación.

Se continúa empleando técnicas de desarrollo centrado en el usuario, por lo que se procede a crear una persona primaria con el fin de reducir toda la información obtenida en el análisis de las respuestas del cuestionario realizado.

Esta persona primaria, será un resumen de los datos recogidos, cuyos objetivos nos ofrecerán obtener una visión más clara de las necesidades que plantean los usuarios.

Edad: 36
Género: Masculino
Frecuencia de juego: Al menos una vez por semana
Elección de villano: Teniendo en cuenta la dificultad del villano/a
Elección de conjuntos modulares: Juega con los conjuntos recomendados
Plataforma preferente para usar aplicaciones: Aplicaciones móviles
Aplicación utilizada referente a Marvel Champions: MarvelCDB
Necesidades:

- Compartir escenarios
- Búsqueda de escenarios
- Filtrado por temáticas/rasgos/dificultad
- Gestión de campañas

Esta persona ficticia está basada en las respuestas con un mayor número de selección. Recoger los datos de esta forma nos permite observar más claramente las necesidades que debe cubrir la aplicación.

6.2. Test de usabilidad

Los test de usabilidad [65] son un tipo de investigación de usuario que evalúa la experiencia del usuario al interactuar con una aplicación. Esto ayuda a los diseñadores y equipos de producto a determinar lo intuitiva y fácil de utilizar que es una aplicación. En esta sección se describe todo el proceso y los resultados de los test de usabilidad realizados durante el proyecto.

Se ha decidido realizar dos test de usabilidad, uno remoto, con los mismos usuarios que rellenaron el cuestionario al inicio del proyecto, estos son jugadores habituales de *Marvel Champions* que pertenecen

a algunos foros y grupos de Telegram. Se les pide realizar algunas tareas, y tras finalizarlas deben contestar un cuestionario con varias preguntas acerca de las tareas realizadas. Y el segundo test se realiza de manera presencial, observando cómo los usuarios interactúan con la aplicación y anotando los resultados.

En este estudio, se han utilizado diversas métricas para evaluar la usabilidad de la aplicación. Estas métricas proporcionan una visión integral del desempeño de los usuarios y de la facilidad de uso del sistema. A continuación, se detallan las métricas empleadas:

- Número de errores encontrados durante el proceso
- Tiempo empleado para completar la tarea
- Si la tarea fue completada o no
- Comentarios realizados por los usuarios

Dado que se ha proporcionado un cuestionario de evaluación (ver Apéndice C), tanto los usuarios que realicen el test de usabilidad remoto como los que lo realicen presencialmente podrán indicar si finalizaron la tarea correctamente y podrán aportar comentarios acerca de la interfaz y del funcionamiento del sistema.

A partir de estas métricas se medirán los siguientes atributos de usabilidad:

- **Facilidad de aprendizaje**: es la facilidad para aprender a usar un sistema, desde lo más básico a todas las funcionalidades que incluye el sistema.
- **Eficiencia**: es el esfuerzo que le supone al usuario realizar lo que quiere en un determinado sistema.
- **Satisfacción**: es el criterio que mide respuestas subjetivas de una persona al interactuar con un sistema, y se mide con cuestionarios.

En ambas pruebas, las tareas a realizar en la aplicación son las siguientes:

- Registrarse/Identificarse
- Ver los escenarios publicados ordenados por diferentes criterios
- Crear/Eliminar un escenario
- Buscar escenarios aplicando varios filtros
- Añadir un escenario a mi lista de favoritos
- Ver la lista de favoritos
- Dar *like* a un escenario
- Valorar la dificultad y la diversión de un escenario

6.2.1. Test de usabilidad remoto

Para la realización de este test, previamente se ha desplegado el sistema completo en una máquina virtual prestada por la escuela utilizando Docker. De esta manera los usuarios del juego pueden interactuar con la aplicación desplegada en web sin ningún tipo de problema. También se les proporciona un cuestionario que deben rellenar tras realizar las tareas que se les pide (ver Apéndice C), este contiene un SUS [66] para poder medir la usabilidad del sistema.

Se ha dejado a los usuarios ir respondiendo a este cuestionario durante una semana. De esta forma, se les concede un tiempo considerable para que la mayoría de usuarios interesados en contestarlo lo hagan. Este cuestionario fue contestado por siete usuarios remotos y por tres presenciales, cuyos resultados están expuestos en la siguiente Sección (6.2.2).

Resultados del test de usabilidad remoto

A continuación, se muestran los diferentes resultados obtenidos de las respuestas de los diez usuarios totales, ya que como el cuestionario es anónimo, no se puede distinguir cuales son los siete usuarios remotos y cuales los tres presenciales. Debido a las características del test, no se ha podido obtener la misma información que en el presencial. No se ha podido medir el tiempo empleado para completar cada una de las tareas a realizar, pero el resto de métricas si se pueden obtener a través del cuestionario realizado.

En la primera pregunta del cuestionario, se indica que se responda a 10 afirmaciones sobre la usabilidad del sistema. En la Figura 6.7 se muestra una gráfica de barras apiladas en la que se observa el grado de aceurdo y desacuerdo de los usuarios con estas afirmaciones. Las conclusiones que podemos sacar de este gráfico son que se trata de un sistema fácil de aprender, ya que la mayoría de usuarios indicaron que no necesitaron aprender muchas cosas antes de utilizar el sistema. Además, la gran mayoría de usuarios indicaron que se sintieron seguros usando el sistema. También, podemos ver que ningún usuario piensa que el sistema es innecesariamente complejo, por lo que, invierten poco esfuerzo en realizar las tareas. Esto indica que es un sistema eficiente, minimiza la inversión de esfuerzo por parte del usuario. Por último, se puede observar que hay una buena experiencia general, ya que nadie ha indicado la necesidad de apoyo técnico para poder utilizar el sistema, nadie ha indicado que sea mínimamente complejo, y el 90 % de los usuarios han indicado que les gustaría utilizar el sistema con frecuencia. Por todo esto, se puede observar que el grado de satisfacción obtenido es muy elevado.

Realizando el cálculo exacto del SUS se obtienen 86.0475 puntos sobre 100. Teniendo en cuenta que el puntaje promedio de entre más de 500 webs y aplicaciones es de 68 puntos [67][68], podemos afirmar que el puntaje obtenido es alto. Teniendo en cuenta que las respuestas que se pueden indicar para estos enunciados siguen la Escala de Likert [69] y equivalen a 1, 2, 3, 4 o 5 en función de la respuesta (siendo 1 ”Totalmente en desacuerdo” y 5 ”Totalmente de acuerdo”), el resultado del SUS se obtiene de la siguiente forma:

1. Calcular la media aritmética de las diferentes respuestas obtenidas en cada uno de los enuncia-

dos.

2. Sumar las respuestas de los enunciados impares y restarle 5.
3. Sumar las respuestas de los enunciados pares y restar 25 a ese total.
4. Sumar ambos resultados y multiplicar el resultado de esta suma por 2.5.

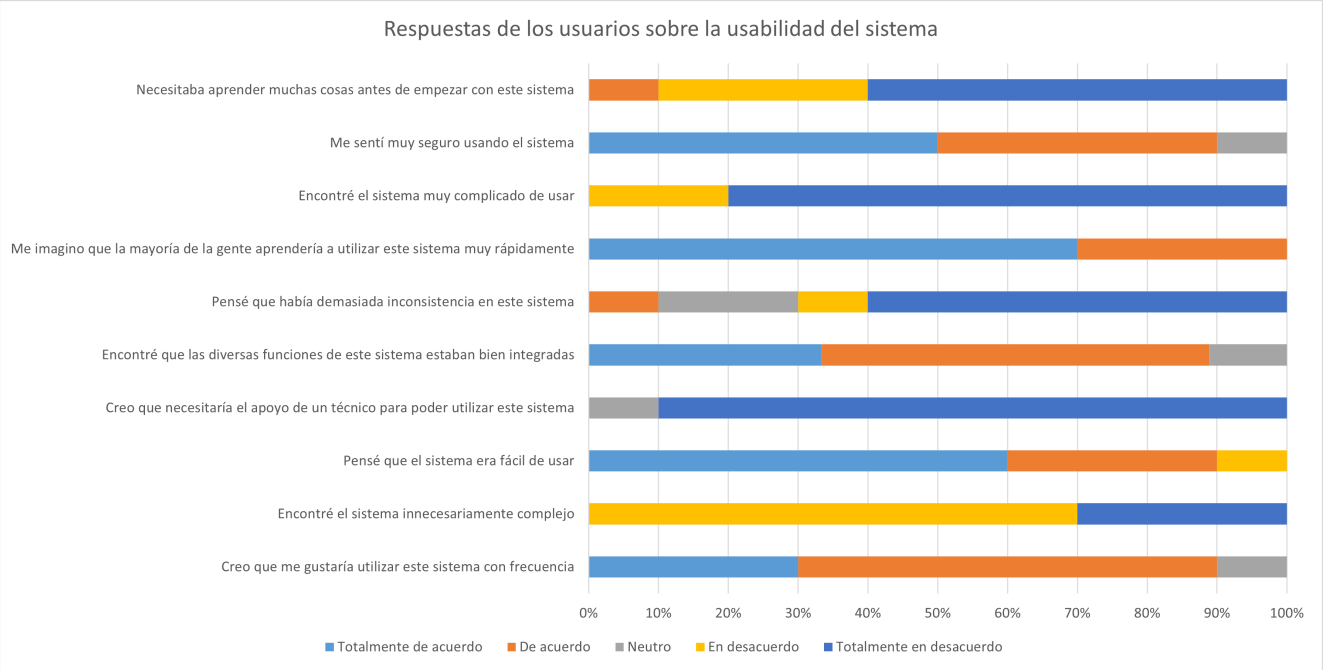


Figura 6.7: Respuestas de los usuarios sobre la usabilidad del sistema

En la Figura 6.8 se muestra la distribución de usuarios que han conseguido completar cada una de las tareas y de los que no (respuestas a la segunda pregunta del cuestionario). En este gráfico se puede observar que la mayoría de estas personas han conseguido realizar las tareas propuestas. Cabe destacar que uno de los usuarios que ha indicado que no ha realizado ninguna de las tareas excepto las únicas en la que no es necesario realizar el inicio de sesión o el registro, en la pregunta número siete del cuestionario de evaluación ha indicado que no ha realizado ninguna tarea en la que fuera necesario identificarse debido a que él en caso de utilizar la aplicación en un futuro, únicamente la utilizaría para ver las partidas publicadas por otros usuarios y para buscar partidas.

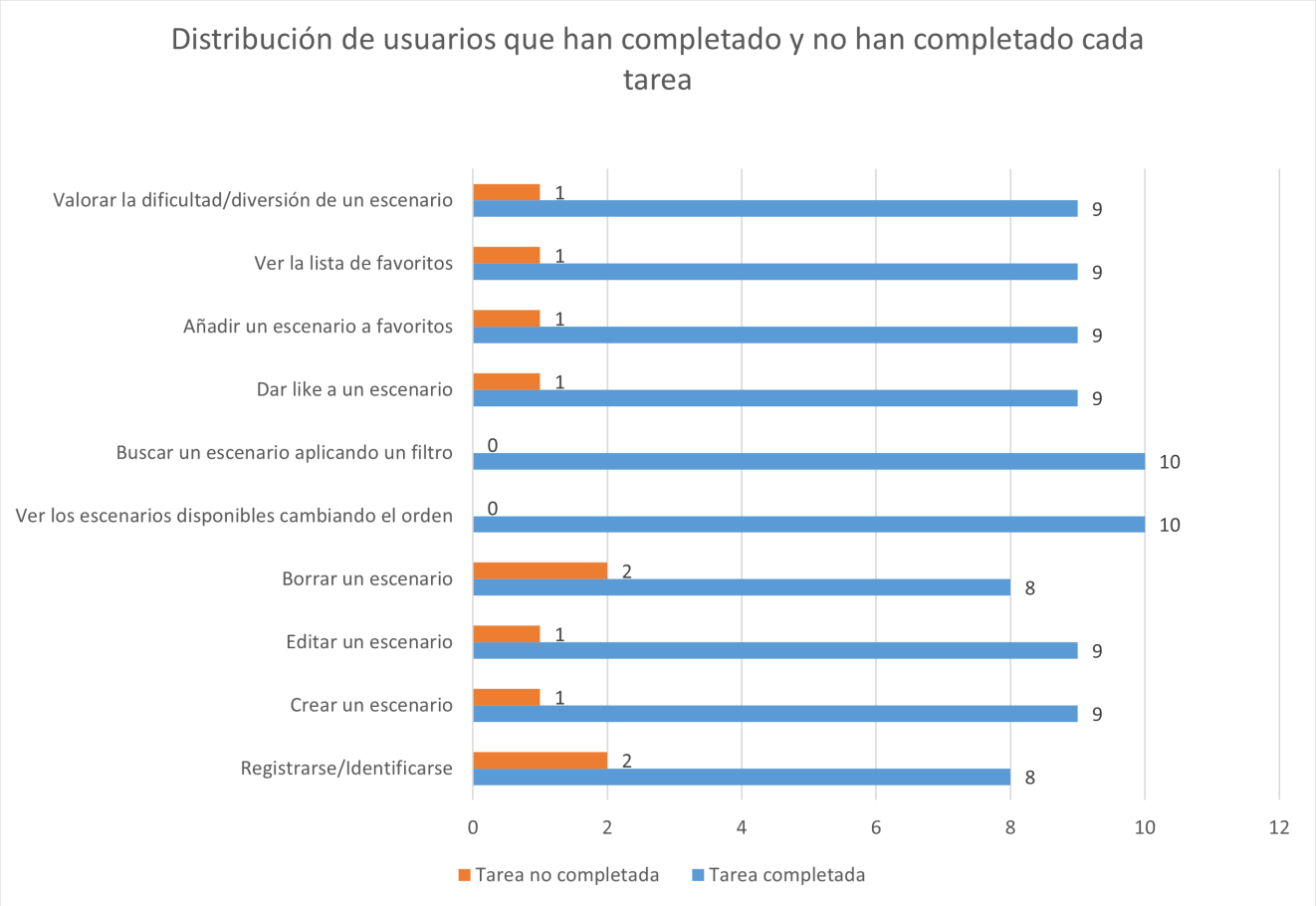


Figura 6.8: Distribución de usuarios que han completado y no han completado cada tarea

Las respuestas a la pregunta tres, en la que se pregunta acerca de la experiencia general con el sistema, por lo general indican un grado de satisfacción muy elevado, ya que todos los usuarios indican que han tenido una experiencia con el producto muy buena (ver Tabla 6.1). En algunas respuestas se señalan diferentes funcionalidades que no son del todo intuitivas, como valorar la dificultad/diversión de una partida, y diferentes mejoras.

En la pregunta cuatro, en la que se pregunta que es lo que más les gustó de la aplicación, varios usuarios indicaron que les gustó la rapidez y sencillez de crear las partidas, esto sugiere que la aplicación es eficiente en términos de tiempo que se tarda en completar las tareas. Generalmente, también les ha gustado los filtros de búsqueda, lo que indica que el sistema ofrece una experiencia agradable y fluida a la hora de realizar búsquedas de partidas. Todas las respuestas a esta pregunta se pueden ver en la Tabla 6.2.

La pregunta cinco trata de recopilar los aspectos de la aplicación que los usuarios cambiarían. En la Tabla 6.3 se muestran las respuestas realizadas por los usuarios a esta pregunta. Son respuestas muy variadas, pero lo que la mayoría de las personas ha señalado, ha sido la manera en la que se valora la dificultad/diversión de una partida, ya que indican que es poco intuitivo como hemos visto en las repuestas a la pregunta número tres.

En la pregunta seis, que pretende recopilar funcionalidades que los usuarios desean que contenga

Usuarios	Respuestas
1	Salvo ciertas funcionalidades concretas como valorar la dificultad/diversion que no son del todo intuitivas en primera instancia la experiencia fue buena
2	Buena. La mayoría de las tareas son fáciles de encontrar.
3	Me parece que se debería de poder dar a favoritos desde fuera,sin entrar al escenario
4	Es fácil de usar e intuitiva. Probablemente se podría mejorar visualmente.
5	Muy buena, la mayoría de cosas son intuitivas y fáciles
6	Satisfactoria
7	Maravillosa
8	Muy buena idea.
9	Tiene buena pinta de momento.
10	Bastante fácil de usar aunque como soy fan de los temas oscuras en las webs me resulta un poco dificl leer texto con el fondo blanco

Tabla 6.1: Respuestas a la pregunta tres del cuestionario de evaluación

Usuarios	Respuestas
1	La rapidez para poder crear partidas y agregarlas a mi lista de favoritos
2	Facilidad de la elección de Encuentro y Villanos. Se rellena muy rápidamente el formulario de creación. Buenas opciones de orden, filtrado y búsqueda. Me gusta poder valorar a otros Encuentros.
3	La interfaz y la comodidad a la hora de filtrar
4	Lo rápido y accesible que está todo.
5	Los filtros de búsqueda
6	Variedad y originalidad
7	La interfaz simple y atractiva
8	Sencillez.
9	Es muy ágil e intuitivo, y no lleva mucho tiempo hacer un escenario
10	La creación de escenarios es bastante sencilla

Tabla 6.2: Respuestas a la pregunta cuatro del cuestionario de evaluación

la aplicación. Se han obtenido diferentes funcionalidades, como poder realizar listas personalizadas, valorar la temática, un generador aleatorio de escenarios, etc. Además de estas nuevas funcionalidades indicadas, también se indican posibles mejoras como establecer la puntuación de las valoraciones en cinco en vez de en tres, indicar que no hay ninguna partida con los filtros de búsqueda aplicados, poder ver las cartas que lleva un conjunto de encuentros con algún enlace a la página de MarvelCDB, etc.

Por último, se pregunta acerca de algún comentario adicional que los usuarios deseen hacer. Entre las respuestas realizadas, se encuentran felicitaciones por la aplicación y, una respuesta interesante ya que en ella el usuario indica problemas que ha tenido con la aplicación y puntos a mejorar de esta. Este indica errores en algunos escenarios y conjuntos modulares obtenidos de la API de MarvelCDB, que la pantalla en la que se seleccionan escenarios y en la que se seleccionan conjuntos modulares, estos podrían estar ordenados alfabéticamente, ya que no lo están. También, señala, al igual que en la respuesta de la pregunta anterior, que le gustaría que al entrar en una partida, tanto el escenario como los conjuntos modulares estuvieran vinculados a las cartas que contienen en la página de MarvelCDB.

Además de todo esto, añade que le gustaría que se pudieran comentar las partidas creadas por otros

usuarios. Al principio del proyecto, esta funcionalidad se planteó, pero la implementación de esta medida requeriría la presencia de un administrador encargado de supervisar y moderar posibles comentarios ofensivos o discriminatorios.

Usuarios	Respuestas
1	La forma de votar la diversion o dificultad de las partidas
2	Me gustaría poder dar ”Me gusta” y ”Añadir a favoritos” desde la pestaña de Home. Creo que al modificar la temática y la dificultad de un Encuentro creado por mi, no se han aplicado los cambios. Para valorar la diversión y la dificultad de un encuentro me costó encontrar donde podía hacerlo.
3	No poder dar likes o añadir a favoritos directamente, junto con tener que scrollear para dar like o añadir a favoritos
4	Visualmente podría ser más atractiva.
5	Poder añadir a favoritos y dar like sin tener que entrar a la partida.
6	Todo correcto
7	Que no se puede instalar en el dispositivo movil
8	Cartas en inglés.
9	En el móvil puede ser un poco tosca la interfaz por movimientos rápidos al salir el teclado de pantalla.
10	que los escenarios y los conjuntos estén en orden aleatorio en vez de orden de publicación u orden alfabético

Tabla 6.3: Respuestas a la pregunta cinco del cuestionario de evaluación

Usuarios	Respuestas
1	Agregar listas personalizadas que no sean únicamente favoritos y likes
2	Me gustaría poder valorar la temática y que las valoraciones fueran de cinco ”puntos” en vez de tres.
3	No
4	Estaría muy bien la posibilidad de elegir un escenario random de los disponibles dentro de unos criterios.
5	Que cuando no encuentre ninguna partida con los filtros de búsqueda lo indique.
6	No
7	Generador aleatorio de escenario
8	Nada
9	Selección individual de cartas de módulos. Ej: Ya hay un mazo publicado del Proyecto Despertar que solo incluye a Nimrod del conjunto de Días del futuro pasado. Podría intentar implementarse algo así.
10	poder ver que cartas lleva ese conjunto de encuentros con algún enlace a marvelcdb o poder al menos ver la carta nº1 del conjunto en grande cuando entre al escenario para cuando la busque entre mis cartas

Tabla 6.4: Respuestas a la pregunta seis del cuestionario de evaluación

6.2.2. Test de usabilidad presencial

Dado que el test de usabilidad remoto se ha realizado en la aplicación web, este test presencial se ha realizado en la aplicación Android, para así probar la aplicación en las dos plataformas. La prueba se realizará por tres personas, que no han tenido contacto con la aplicación previamente, pero estas

son personas jóvenes (dos de ellas de 22 años y una de ellas de 21) y por lo tanto, habituadas a la tecnología.

Antes de comenzar el test se pondrá en contexto a los usuarios, se les explicará en que consiste la aplicación y las tareas que tienen que realizar. Durante el proceso se estará observando como estas personas interactúan con la aplicación y se anotarán los resultados de las métricas mencionadas anteriormente (ver Sección 6.2).

Resultados del test de usabilidad presencial

A continuación se muestran los resultados obtenidos de las tres pruebas recogidos en tablas en las que se indica si la tarea ha sido completada o no, el tiempo empleado en ello, si creen que la interfaz es intuitiva o no y el número de errores realizados durante el proceso.

En la tarea registrarse/identificarse los tres usuarios completaron la tarea sin ningún error y bastante rápido para ser la primera vez que utilizaban la aplicación (ver Figura 6.5).

Registrarse/Identificarse	Usuario 1	Usuario 2	Usuario 3
Tarea completada	Si	Si	Si
Tiempo empleado	30s	41s	32s
Interfaz intuitiva	Si	Si	Si
Número de errores	0	0	0

Tabla 6.5: Métricas test de usabilidad de la tarea registrarse/identificarse

En la Figura 6.6 se puede observar que la tarea se realizó sin ningún problema, y al igual que en la anterior, bastante rápido.

Al igual que en las anteriores tareas, la tarea de publicar un escenario se realizó sin ninguna complicación.

Ver escenarios ordenados diferentes	Usuario 1	Usuario 2	Usuario 3
Tarea completada	Si	Si	Si
Tiempo empleado	15s	10s	10s
Interfaz intuitiva	Si	Si	Si
Número de errores	0	0	0

Tabla 6.6: Métricas test de usabilidad de la tarea ver escenarios ordenados de distinta forma

Publicar un escenario	Usuario 1	Usuario 2	Usuario 3
Tarea completada	Si	Si	Si
Tiempo empleado	41s	1min	31s
Interfaz intuitiva	Si	Si	Si
Número de errores	0	0	0

Tabla 6.7: Métricas test de usabilidad de la tarea publicar un escenario

La tarea eliminar un escenario también se realizó sin ninguna complicación (ver Figura 6.8), pero

durante esta tarea el primer usuario comentó que el mensaje de confirmación de eliminación de una partida era algo pequeño, que quizá debería ser más grande.

Eliminar un escenario	Usuario 1	Usuario 2	Usuario 3
Tarea completada	Si	Si	Si
Tiempo empleado	10s	7s	7s
Interfaz intuitiva	Si	Si	Si
Número de errores	0	0	0

Tabla 6.8: Métricas test de usabilidad de la tarea eliminar un escenario

Respecto a la tarea de buscar escenarios aplicando varios filtros, el primer usuario indicó que la manera de filtrar por dificultad, por temática y por diversión es poco intuitiva, ya que no hay nada que indique que los iconos para buscar por grado de dificultad/temática/diversión son clicables. El tercer usuario comentó que en caso de filtrar por campos y que no exista ningún escenario con esos campos, le gustaría que hubiera algún mensaje que indicara al usuario que no hay ningún escenario existente con los filtros aplicados. A pesar de ello, ambos completaron la tarea con éxito y sin tardar demasiado como se puede ver en la Figura 6.9).

Buscar escenarios	Usuario 1	Usuario 2	Usuario 3
Tarea completada	Si	Si	Si
Tiempo empleado	50s	32s	43s
Interfaz intuitiva	No	Si	Si
Número de errores	0	0	0

Tabla 6.9: Métricas test de usabilidad de la tarea buscar escenarios aplicando varios filtros

Durante la realización de la tarea de añadir un escenario a la lista de favoritos, los usuarios dos y tres añadieron que se podrían guardar los escenarios en la lista de favoritos sin necesidad de tener que ver los detalles de este, desde fuera. Y que el icono de guardado en favoritos que hay fuera de los detalles del escenario da pie a clicarlo desde ahí. El segundo usuario añadió que el botón para añadir un escenario en favoritos debería estar lo primero en la pantalla de visualización de un escenario para así no tener que hacer *scroll* hasta abajo para poder realizar esta tarea. Aún así, ninguno de los usuarios tubo problema en finalizar la tarea sin errores (ver Figura 6.10).

Añadir a favoritos	Usuario 1	Usuario 2	Usuario 3
Tarea completada	Si	Si	Si
Tiempo empleado	17s	11s	11s
Interfaz intuitiva	Si	Si	Si
Número de errores	0	1	1

Tabla 6.10: Métricas test de usabilidad de la tarea añadir a favoritos

Los tres usuarios realizaron la tarea de ver la lista de favoritos sin ningún problema y añadieron que es muy intuitivo acceder a esta lista (ver Figura 6.11).

La tarea de dar *like* a un escenario se realizó sin ningún problema como se puede ver en la Figura

Ver lista de favoritos	Usuario 1	Usuario 2	Usuario 3
Tarea completada	Si	Si	Si
Tiempo empleado	6s	3s	4s
Interfaz intuitiva	Si	Si	Si
Número de errores	0	0	0

Tabla 6.11: Métricas test de usabilidad de la tarea ver la lista de favoritos

6.12, pero al igual que en la tarea de añadir un escenario a favoritos, los usuarios dos y tres comentaron que debería poder darse *like* desde fuera de los detalles de un escenario y el segundo indicó que el botón de dar *like* en la pantalla de visualización de un escenario debería ir lo primero, como se explicó anteriormente.

Dar *like* a un escenario	Usuario 1	Usuario 2	Usuario 3
Tarea completada	Si	Si	Si
Tiempo empleado	11s	5s	7s
Interfaz intuitiva	Si	Si	Si
Número de errores	0	0	0

Tabla 6.12: Métricas test de usabilidad de la tarea dar *like* a un escenario

En la Figura 6.13 se muestran los resultados de la tarea de valorar la dificultad y la diversión de un escenario. Durante esta tarea, el primer y el tercer usuario indicaron que la manera de valorar ambos aspectos no es del todo intuitivo, ya que más allá de los mensajes ”¡Valora aquí la dificultad!” y ”¡Valora aquí la diversión!”, la forma en la que se realiza esta valoración no se diferencia en nada de la manera en que se muestra la dificultad medai de la partida o la diversión media. Además, añadieron que podría haber dos botones para subir y bajar el grado de diversión/dificultad.

Valorar la dificultad y la diversión de un escenario	Usuario 1	Usuario 2	Usuario 3
Tarea completada	Si	Si	Si
Tiempo empleado	17s	12s	14s
Interfaz intuitiva	No	Si	No
Número de errores	0	0	0

Tabla 6.13: Métricas test de usabilidad de la tarea valorar la dificultad y la diversión de un escenario

6.3. Conclusiones test de usabilidad

En base a los resultados obtenidos de ambas pruebas podemos confirmar que el sistema desarrollado es muy **fácil de aprender** y de usar, ya que todos los usuarios han conseguido finalizar todas las tareas propuestas excepto dos (y uno de ellos comentó que no realizó esas tareas debido a que en el futuro esas tareas no las realizaría), y no solo eso, si no que los usuarios que realizaron la prueba de manera presencial, han conseguido realizarlas en un tiempo bastante corto y prácticamente sin errores. También podemos afirmar que la métrica de eficiencia se ha cumplido satisfactoriamente, debido a que

todos los usuarios fueron capaces de completar las tareas asignadas sin mayor esfuerzo, lo cual es un indicador positivo de la **eficiencia** de la aplicación. Como ya se ha observado antes en la gráfica 6.7, en general los usuarios han tenido una buena experiencia con el uso del sistema, lo que refuerza la idea de que se trata de una aplicación eficiente. Además, como ya se dijo anteriormente, el tiempo empleado en la realización de las tareas ha sido consistentemente bajo y el número de errores nulo en la mayoría de las tareas.

Gracias a los resultados obtenidos en la pregunta dos del cuestionario, podemos afirmar que el grado de **satisfacción** de los usuarios es elevado, ya que se obtiene un resultado del SUS de 86.0475 puntos. Este resultado indica una alta usabilidad del sistema, que la mayoría de usuarios lo encuentran fácil de usar y eficiente. A pesar de esto, se puede mejorar el sistema, ya que varios usuarios han indicado en las siguientes preguntas comentarios valiosos con los que poder mejorarlo.

Cabe destacar que todas las mejoras propuestas por los usuarios tanto de manera presencial como remota, serían consideradas en un futuro, ya que el tiempo disponible para la realización del proyecto no da más de si.

Capítulo 7

Conclusiones

En este capítulo se comentan las conclusiones que se han obtenido tras la realización del proyecto. También se habla sobre los trabajos futuros que se podrían realizar sobre la aplicación para ampliarla y/o mejorarla en términos de funcionalidad.

Lo primero a destacar es todo el proceso de comunicación con el usuario, desde el cuestionario inicial hasta las pruebas de usabilidad realizadas. Este es un proceso innovador en los trabajos de fin de grado de la universidad debido a todo este proceso tan cercano al usuario. Por ello, es de gran valor para el proyecto, ya que de esta forma se consigue crear un producto acorde a las necesidades de los usuarios reales, en este caso del juego de cartas *Marvel Champions*.

Otro rasgo del proyecto a destacar son las tecnologías elegidas. Se trata de tecnologías modernas y muy utilizadas en el entorno laboral como lo es Docker o Node.js. Que se trate de unas tecnologías tan modernas es un gran punto positivo, ya que se ha aprendido ampliamente de estas. También el uso de herramientas auxiliares como Postman para el desarrollo de la API es un punto positivo ya que se trata de una herramienta muy utilizada en el entorno laboral.

Algo importante a destacar sobre un proyecto es si se han cumplido o no los objetivos principales establecidos (ver Sección 1.6 del Capítulo 1). En este proyecto los objetivos han sido completados, ya que se ha conseguido construir una aplicación multiplataforma desde cero que permite gestionar los escenarios del juego de mesa *Marvel Champions* habiendo realizado tanto el front-end como el back-end de la aplicación, lo que ha permitido aprender las particularidades del desarrollo *full-stack* utilizando las tecnologías Flutter y Node.js. Además, se han empleado las metodologías centradas en el usuario, realizando un análisis de usuarios y sus necesidades con el cuestionario inicial distribuido por diferentes foros, y evaluando la aplicación con usuarios reales con el cuestionario de evaluación realizado. Este último cuestionario ha permitido, entre otras cosas, medir la usabilidad del sistema teniendo en cuenta la opinión de diversas personas pudiendo obtener así un grado de satsifacción promedio elevado, específicamente de 86.0475 puntos sobre 100.

Por último, resaltar que el punto negativo que se ha encontrado ha sido la falta de tiempo, ya que un proyecto como este en el que se desarrolla tanto front-end como back-end y que se centra en el usuario, necesita más tiempo que el que puede abarcar un trabajo de fin de grado. Hubiera sido

interesante poder realizar las mejoras y cambios que los usuarios han propuesto en ambas pruebas de usabilidad realizadas. A continuación, se muestran funcionalidades y mejoras que se podrían realizar en un futuro.

7.1. Trabajo futuro

Al principio del proyecto se consideraron tanto objetivos principales como secundarios. Estos objetivos secundarios se encuentran en esta sección, ya que además de serlo, durante las pruebas de usabilidad han sido propuestos por los usuarios. También se muestran los posibles cambios y nuevas funcionalidades mencionadas por los usuarios durante las pruebas de usabilidad. La siguiente lista muestra todas estas nuevas funcionalidades y mejoras:

- **Nuevas funcionalidades**: la gestión de campañas sería una de las nuevas funcionalidades que se propuso como objetivo secundario al inicio del proyecto, y también fue mencionado por los usuarios durante las pruebas de usabilidad. Los usuarios también mencionaron generar escenarios de forma aleatoria, valorar la temática de diferentes partidas, poder agregar listas personalizadas e integrar la aplicación tanto en inglés como en español.
- **Mejoras de funcionalidades**: estas mejoras fueron propuestas por los propios usuarios, entre ellas: mejorar la forma en la que se valora la diversión/dificultad de las partidas, poder dar *like* y añadir partidas a favoritos desde la pantalla principal sin necesidad de entrar en los detalles de la partida, indicar que no existe ninguna partida con los filtros de búsqueda aplicados en caso de que así sea y poder ver las cartas que lleva un conjunto de encuentros.
- **Pruebas**: para completar el proyecto de forma profesional sería necesario la realización de las debidas pruebas de la aplicación; pruebas unitarias, de integración y *end-to-end*.
- **Interfaz de usuario**: se podría embellecer la interfaz de usuario de la aplicación si se hubiera dispuesto de más tiempo, para así poder ofrecer una mejor experiencia de usuario.
- **Pruebas en iOS**: se podría realizar pruebas en un dispositivo iOS en caso de disponer de un Mac. Este problema se inclina más hacia la falta de disponibilidad de este dispositivo que a la falta de tiempo. Aún así, realizar esta prueba sería esencial en caso de desplegar una aplicación multiplataforma como esta.

Apéndices

Apéndice A

Acrónimos

- **API**: *Application Programming Interface*
- **APK**: *Android Package*
- **BLoC**: *Business Logic Component*
- **CRLF**: *Carriage Return Line Feed*
- **DAO**: *Data Access Object*
- **DTO**: *Data Transfer Object*
- **HTTP**: *Hypertext Transfer Protocol*
- **IDE**: *Integrated Development Environment*
- **IPA**: *iOS App Store Package*
- **JSON**: *JavaScript Object Notation*
- **JWT**: *JSON Web Token*
- **LF**: *Line Feed*
- **LGC**: *Living Card Game*
- **REST**: *Representational State Transfer*
- **SDK**: *Software Development Kit*
- **SUS**: *System Usability Scale*
- **UI**: *User Interface*
- **URL**: *Uniform Resource Locator*

Apéndice B

Cuestionario inicial

En este apéndice se encuentra el cuestionario realizado a algunos jugadores de *Marvel Champions* al inicio del proyecto para obtener los requisitos funcionales de la aplicación. El cuestionario es el siguiente:

1. Edad.
2. Género.
3. ¿En los últimos tres meses, con que frecuencia has jugado al Marvel Champions?
    a) Al menos una vez a la semana
    b) Al menos una vez al mes
    c) No he jugado
    d) Otras
4. ¿Cómo juegas más frecuentemente al Marvel Champions?
    a) True solo
    b) Dos manos
    c) Multijugador
    d) Otras
5. ¿De qué formas juegas más frecuentemente al Marvel Champions?
    a) Físico
    b) TTS
    c) drgoncards
    d) octgn
    e) Otras
6. ¿Con qué dificultad juegas más frecuentemente al Marvel Champions?

a) Normal

b) Experto

c) Normal II

d) Experto II

7. Indica con que frecuencia usas las siguientes opciones para elegir al villano cuando vas a jugar una partida.

	Nunca	Casi nunca	De vez en cuando	Casi siempre	Siempre
Aleatoriamente	○	○	○	○	○
Teniendo en cuenta que sea temático con mi héroe/heroína elegido/a	○	○	○	○	○
Teniendo en cuenta las mecánicas de mi héroe/heroína elegido/a	○	○	○	○	○
Juego Campañas	○	○	○	○	○
Teniendo en cuenta la dificultad del villano/a	○	○	○	○	○

Figura B.1: Pregunta 7

8. Indica con que frecuencia usas las siguientes opciones para elegir los conjuntos modulares cuando vas a jugar una partida (ver Figura B.2).

9. ¿En los últimos tres meses, con que frecuencia has jugado a otros juegos de mesa que no sean Marvel Champions?

   a) Al menos una vez a la semana

   b) Al menos una vez al mes

   c) No he jugado

   d) Otras

10. ¿Qué tipo de plataforma usas con más frecuencia para usar tus aplicaciones?

    a) Aplicaciones web

    b) Aplicaciones móviles

    c) Aplicaciones de escritorio (en Windows, Linux)

    d) Otras

	Nunca	Casi nunca	De vez en cuando	Casi siempre	Siempre
Aleatoriamente	○	○	○	○	○
Teniendo en cuenta que sea temático el/la villano/a elegido/a	○	○	○	○	○
Teniendo en cuenta las mecánicas de mi héroe/heroína elegido/a	○	○	○	○	○
Teniendo en cuenta las mecánicas del villano	○	○	○	○	○
Teniendo en cuenta la dificultad del villano	○	○	○	○	○
Juego con los conjuntos recomendados	○	○	○	○	○

Figura B.2: Pregunta 8

11. ¿Qué aplicaciones usas relacionadas con el Marvel Champions? Puedes elegir más de una opción

    a) Marvelcdb

    b) Mcfanmade

    c) Ninguna

    d) Otras

12. ¿Usas aplicaciones para gestionar tus partidas de Marvel Champions? Si es así, indica qué aplicaciones utilizas

13. ¿Hay alguna funcionalidad que crees imprescindible en una aplicación de gestión de escenarios? (Por ejemplo, búsqueda de escenarios por dificultad, por conjuntos de encuentro; escenarios favoritos, gestión de campañas)

14. Si hay información adicional que quieras añadir, escríbela en esta respuesta

Apéndice C

Cuestionario de evaluación

En este apéndice se encuentra el cuestionario de evaluación realizado para los usuarios del juego *Marvel Champions*. Además de varias preguntas acerca de posibles cambios, lo que mas les ha gustado, etc. Contiene un *System Usability Scale* (SUS) utilizado para medir la usabilidad de la aplicación. Un SUS es un cuestionario validado que permite medir la percepción de la usabilidad de un sistema [66].

1. Responde a las siguientes afirmaciones.

	Totalmente en desacuerdo	En desacuerdo	Neutro	De acuerdo	Totalmente de acuerdo
Creo que me gustaría utilizar este sistema con frecuencia	○	○	○	○	○
Encontré el sistema innecesariamente complejo	○	○	○	○	○
Pensé que el sistema era fácil de usar	○	○	○	○	○
Creo que necesitaría el apoyo de un técnico para poder utilizar este sistema	○	○	○	○	○
Encontré que las diversas funciones de este sistema estaban bien integradas	○	○	○	○	○
Pensé que había demasiada inconsistencia en este sistema	○	○	○	○	○
Me imagino que la mayoría de la gente aprendería a utilizar este sistema muy rápidamente	○	○	○	○	○
Encontré el sistema muy complicado de usar	○	○	○	○	○
Me sentí muy seguro usando el sistema	○	○	○	○	○
Necesitaba aprender muchas cosas antes de empezar con este sistema	○	○	○	○	○

Figura C.1: Pregunta 1

2. ¿Has conseguido completar las siguientes tareas?

	Si	No
Registrarse	○	○
Crear un escenario	○	○
Editar un escenario	○	○
Borrar un escenario	○	○
Ver los escenarios disponibles cambiando el orden	○	○
Buscar un escenario aplicando un filtro	○	○
Dar like a un escenario	○	○
Añadir un escenario a favoritos	○	○
Ver la lista de favoritos	○	○
Valorar la dificultad/diversión de un escenario	○	○

Figura C.2: Pregunta 2

3. ¿Cómo describirías tu experiencia general con el producto?
4. ¿Qué fue lo que más te gustó de la aplicación?
5. ¿Qué fue lo que menos te gustó de usar la aplicación o qué cosas cambiarías?
6. ¿Echas en falta alguna funcionalidad de la aplicación?
7. Añade cualquier comentario adicional que quieras hacer.

Apéndice D

Manual de despliegue

En este apéndice se explica el procedimiento para desplegar la aplicación para su desarrollo o instalación. El código de la aplicación se encuentra en dos repositorios del GitLab de la Escuela de Ingeniería Informática de la UVa, uno contiene el front-end de la aplicación y el otro el back-end. Los enlaces son los siguientes:

- Front-end: `https://gitlab.inf.uva.es/oscsaez/tfg-marvelchampionsfront`
- Back-end: `https://gitlab.inf.uva.es/oscsaez/tfg-marvelchampionsback`

A continuación, se muestran los pasos a seguir para desplegar el sistema al completo. Para desplegar la versión web, únicamente hace falta tener instalado Docker en la máquina donde se vaya a desplegar para poder llevarse a cabo. Sin embargo, para generar un APK para la versión Android es necesario instalar Flutter como se mostrará más adelante.

El contenido de los ficheros *Dockerfile* y *docker-compose.yml* tanto de la parte front-end del sistema como de la parte back-end serán explicados en la Sección D.4 de este apéndice.

D.1. Instalación Docker

1. Ejecutar *sudo apt update* para actualizar el índice de paquetes de la máquina.
2. Instalar Docker con el comando *sudo apt install docker-ce*.

D.2. Despliegue web

1. Clonar los repositorios de ambos proyectos en el directorio que se desee utilizando el comando *git clone <URL_del_repositorio>*.

2. Ejecutar *docker-compose up*[1] dentro del proyecto back-end para desplegarlo.

3. Ejecutar de nuevo *docker-compose up*1 dentro del proyecto front-end para desplegarlo.

D.3. Despliegue en Android

D.3.1. Instalación Flutter

Cabe destacar que la instalación de Flutter se ha realizado en un equipo con sistema operativo Windows 10, por lo que la instalación que se muestra a continuación es para este sistema.

1. Descargar el SDK de Flutter desde la página oficial [70].

2. Descomprimir el contenido del archivo en la ubicación que se prefiera, con la excepción de que la ruta a esa ubicación no contenga caracteres especiales ni espacios, ni requiera permisos de administrador.

3. Añadir Flutter al PATH:

   a) Abrir la ventana de variables de entorno.

   b) En la sección de **Variables del sistema**, buscar la variable "**Path**", seleccionarla y hacer clic en **Editar**.

   c) Introducir una nueva ruta al directorio "**bin**" dentro de la carpeta donde se descomprimió Flutter.

4. Ejecutar *flutter –version* para comprobar la configuración del PATH.

5. Ejecutar *flutter doctor –android-licenses* para aceptar las licencias de las herramientas de Android, si no, Flutter no podrá utilizar las herramientas necesarias del SDK de Android.

6. Ejecutar *flutter doctor* para verificar la instalación de Flutter. Este indicará si hay algún componente que necesite atención adicional.

D.3.2. Despliegue del back-end

En este caso solo será necesario desplegar el proyecto back-end con Docker:

1. Clonar el repositorio del proyecto back-end utilizando *git clone <URL_del_repositorio>*.

2. Ejecutar el comando *docker-compose up*1 para desplegar el proyecto.

[1] El comando *docker-compose up* a veces es necesario ejecutarlo añadiendo *sudo* delante, ya que en algunos sistemas es necesario ejecutarlo como *root* o administrador. En caso de querer desplegar la aplicación en segundo plano se debe usar la opción *-d*.

D.3.3. Generar APK

Para generar el APK simplemente habrá que ejecutar el comando *flutter build apk –release*. Esto generará un apk llamado *app-release.apk*en la ruta */build/app/outputs/flutter-apk/*. Solo queda instalarlo en un dispositivo físico Android y abrirlo.

D.4. Ficheros Docker

En esta sección se explican los ficheros *Dockerfile* y *docker-compose.yml* del proyecto Flutter y los del proyecto Node.js.

D.4.1. Ficheros Docker del proyecto front-end

La primera instrucción que contiene el *Dockerfile* de la Figura D.1 establece la imagen base para construir la imagen de Docker. Lo siguiente a realizar es la instalación de las dependencias de Flutter y del propio Flutter, así como la inclusión de las rutas a los ejecutables de Flutter y su SDK de Dart para que sean accesibles desde cualquier ubicación dentro del contenedor. Posteriormente, se realiza la comprobación de que el SDK de Flutter y de Dart, entre otras cosas, estén instalados correctamente, y la actualización de la versión de Flutter instalada a la más reciente. Tras esto, se realiza la copia de toda la aplicación al contenedor y se compila la aplicación web. Por último, se crea una imagen de Nginx [71] que sirve la aplicación web compilada en la anterior instrucción.

En la Figura D.2 se muestra el contenido del fichero *docker-compose.yml*. En este fichero se define el servicio con nombre ”front” y se indica como debe construirse la imagen del contenedor, especificando el directorio en el que se encuentran los archivos necesarios para su construcción e indicando cual es el *Dockerfile*. Por último, se indica el puerto en el que se ejecutará la aplicación y desde el cual será accesible.

```
# Stage 1 - Install dependencies and build the app in a build environment
FROM debian:latest AS build-env

# Install flutter dependencies
RUN apt-get update
RUN apt-get install -y curl git wget unzip libgconf-2-4 gdb libstdc++6 libglu1-mesa fonts-droid-fallback
lib32stdc++6 python3 sed
RUN apt-get clean

# Clone the flutter repo
RUN git clone https://github.com/flutter/flutter.git /usr/local/flutter

# Set flutter path
ENV PATH="${PATH}:/usr/local/flutter/bin:/usr/local/flutter/bin/cache/dart-sdk/bin"

# Run flutter doctor
RUN flutter doctor -v
RUN flutter channel master
RUN flutter upgrade

# Copy files to container and build
RUN mkdir /app/
COPY . /app/
WORKDIR /app/
RUN flutter build web --release

# Stage 2 - Create the run-time image
FROM nginx:1.21.1-alpine
COPY --from=build-env /app/build/web /usr/share/nginx/html
```

Figura D.1: *Dockerfile* del proyecto front-end

```
services:
  front:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3000:80"
```

Figura D.2: Docker compose del proyecto front-end

D.4.2. Ficheros Docker del proyecto back-end

En la Figura D.3 se muestra el contenido del Dockerfile del proyecto Node.js. La primera instrucción que contiene especifica la imagen base que se va a utilizar para construir una nueva imagen Docker. En este caso, la imagen base es una versión de Node.js basada en Debian Bookworm [72] ligera. Posteriormente, se establece el directorio de trabajo dentro del contenedor Docker, se instala el paquete *dos2unix*[1] para que a la hora de instalar los módulos necesarios en el contenedor no haya problemas con las terminaciones de línea DOS/Windows. A continuación, se copia el fichero *package.json* al directorio de trabajo del contenedor y se instalan las dependencias del proyecto según lo especificado en el *package.json*. El propósito de las siguientes instrucciones, es la copia del proyecto al completo en el directorio de trabajo del contenedor, y del *script wait-for-it.hs* en el directorio de binarios del contenedor. Posteriormente se utiliza el comando *dos2unix* para convertir todas las terminaciones de línea estilo DOS/Windows presentes en el proyecto a terminaciones Unix, y también las del ejecutable *wait-for-it.sh*. Por último, se indica el puerto en el que la aplicación escucha para conexiones y se inicia la aplicación Node.js llevándose antes a cabo la ejecución del *script process_scenarios_and_modular.js* y utilizando *wait-for-it.sh* para que la construcción de este contenedor espere a que el servicio *mysqldb* esté listo.

```
FROM node:bookworm-slim

WORKDIR /myapp

RUN apt-get update && apt-get install -y dos2unix

COPY package.json ./
RUN npm install

COPY . .
COPY wait-for-it.sh /usr/local/bin/wait-for-it.sh

RUN dos2unix /usr/local/bin/wait-for-it.sh && chmod +x /usr/local/bin/wait-for-it.sh

RUN find /myapp -type f -exec dos2unix {} \;

EXPOSE 3000
CMD ["sh", "-c", "wait-for-it.sh mysqldb:3306 -- node process_scenarios_and_modular.js && npm start"]
```

Figura D.3: Dockerfile del proyecto back-end

En el *docker-compose.yml* de la Figura D.4 se pueden observar dos servicios, uno para la base de datos de la aplicación y otro para el proyecto back-end. Para el servicio de la base de datos (llamado *mysqldb*) se utiliza la imagen oficial de Docker para MySQL, se le pone un nombre al contenedor (*db*), se indica cual es el fichero que contiene las variables de entorno, y por último, se definen algunas configuraciones adicionales para el servicio:

[1] Convierte ficheros de texto con terminaciones de línea estilo DOS/Windows (CLRF) a terminaciones de línea estilo Unix (LF).

- Se establece la contraseña del usuario root de MySQL y se crea una base de datos con el nombre especificado por la variable de entorno correspondiente.
- Se mapea el puerto especificado en la variable de entorno *MYSQLDB_LOCAL_PORT* del sistema *host* al puerto especificado en *MYSQLDB_DOCKER_PORT* del contenedor para poder acceder a MySQL en el contenedor a través del puerto definido en el host.
- Se monta un volumen para persistir los datos de MySQL, de esta forma, los datos no se pierden al reiniciar el contenedor. Y, se mapea el fichero *db.sql* a */docker-entrypoint-initdb.d/init.sql* dentro del contenedor, ya que MySQL ejecutará cualquier *script* SQL ubicado en ese directorio cuando se inicie por primera vez. De esa forma, se inicializa la base de datos.

Al contenedor del servicio del proyecto back-end (llamado *app*) se le da el nombre *back*, se indica que depende del servicio de la base de datos y se establece una conexión de red entre ambos contenedores. Posteriormente, se indica que se debe construir la imagen del contenedor *app* utilizando el *Dockerfile* ubicado en el directorio actual. A continuación, se especifica la política de reinicio del contenedor: *unless-stopped*, que indica que Docker intentará reiniciar el contenedor siempre que no haya sido detenido manualmente. Al igual que en el servicio anterior, se indica el fichero que contiene las variables de entorno, y se mapea el puerto especificado en la variable de entorno *NODE_LOCAL_PORT* del sistema *host* al puerto especificado en *NODE_DOCKER_PORT* del contenedor. Finalmente, se definen las variables necesarias para llevar a cabo la configuración de la conexión a la base de datos desde la aplicación back-end.

D.5. Rutas para acceder a la aplicación

Tras haber desplegado el proyecto back-end y la aplicación web en el proyecto front-end, las rutas para acceder tanto a la documentación de la API de Node.js como a la aplicación web son las siguientes:

- Documentación de la API: *http://<ip_de_la_máquina>:8080/api-doc/*
- Aplicación web: *http://<ip_de_la_máquina>:3000/*

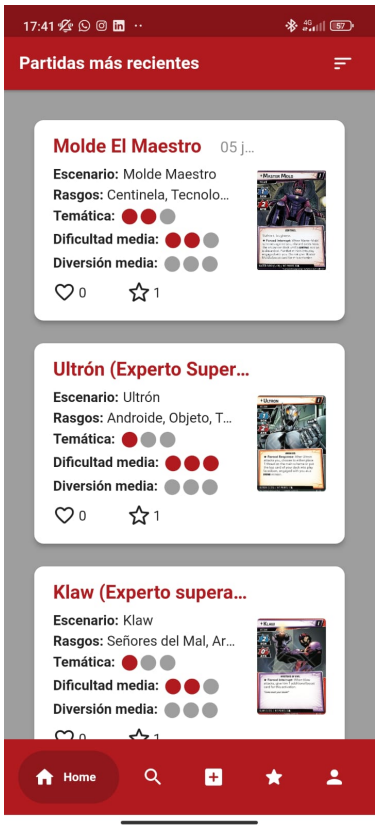
```yaml
services:
  mysqldb:
    image: mysql
    container_name: db
    env_file: .env
    environment:
      - MYSQL_ROOT_PASSWORD=$MYSQL_ROOT_PASSWORD
      - MYSQL_DATABASE=$MYSQLDB_DATABASE
    ports:
      - $MYSQLDB_LOCAL_PORT:$MYSQLDB_DOCKER_PORT
    volumes:
      - mysql:/var/lib/mysql
      - ./db/db.sql:/docker-entrypoint-initdb.d/init.sql
  app:
    container_name: back
    depends_on:
      - mysqldb
    links:
      - mysqldb
    build: ./
    restart: unless-stopped
    env_file: .env
    ports:
      - $NODE_LOCAL_PORT:$NODE_DOCKER_PORT
    environment:
      - DB_HOST=$MYSQLDB_HOST
      - DB_USER=$MYSQLDB_USER
      - DB_PASSWORD=$MYSQL_ROOT_PASSWORD
      - DB_NAME=$MYSQLDB_DATABASE
      - DB_PORT=$MYSQLDB_DOCKER_PORT
volumes:
  mysql:
```

Figura D.4: Docker compose del proyecto back-end

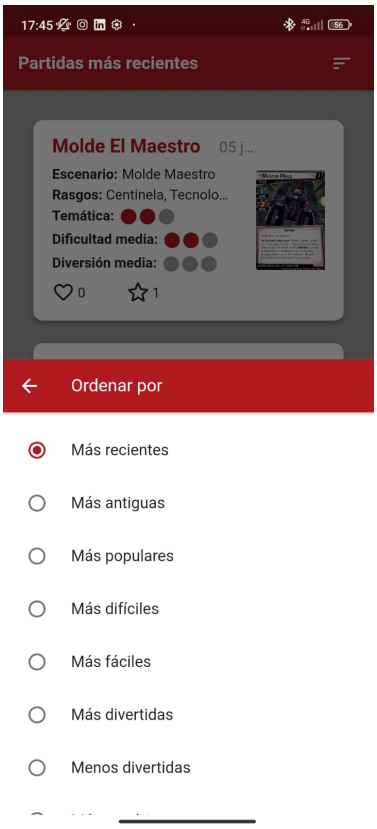
Apéndice E

Manual de uso

Lo primero que ve el usuario al abrir la aplicación es la pantalla principal en la que se muestran las diferentes partidas (ver Figura E.1a). En esta pantalla, además de poder ver los escenarios publicados por diferentes usuarios, estas partidas se pueden ordenar por sus diferentes campos clicando en el botón de la esquina superior derecha de la pantalla como se muestra en la Figura E.1b. Como es de esperar, también se puede acceder a los detalles de estas partidas clicando sobre ellas, navegando así a la pantalla que se muestra en la Figura E.2.

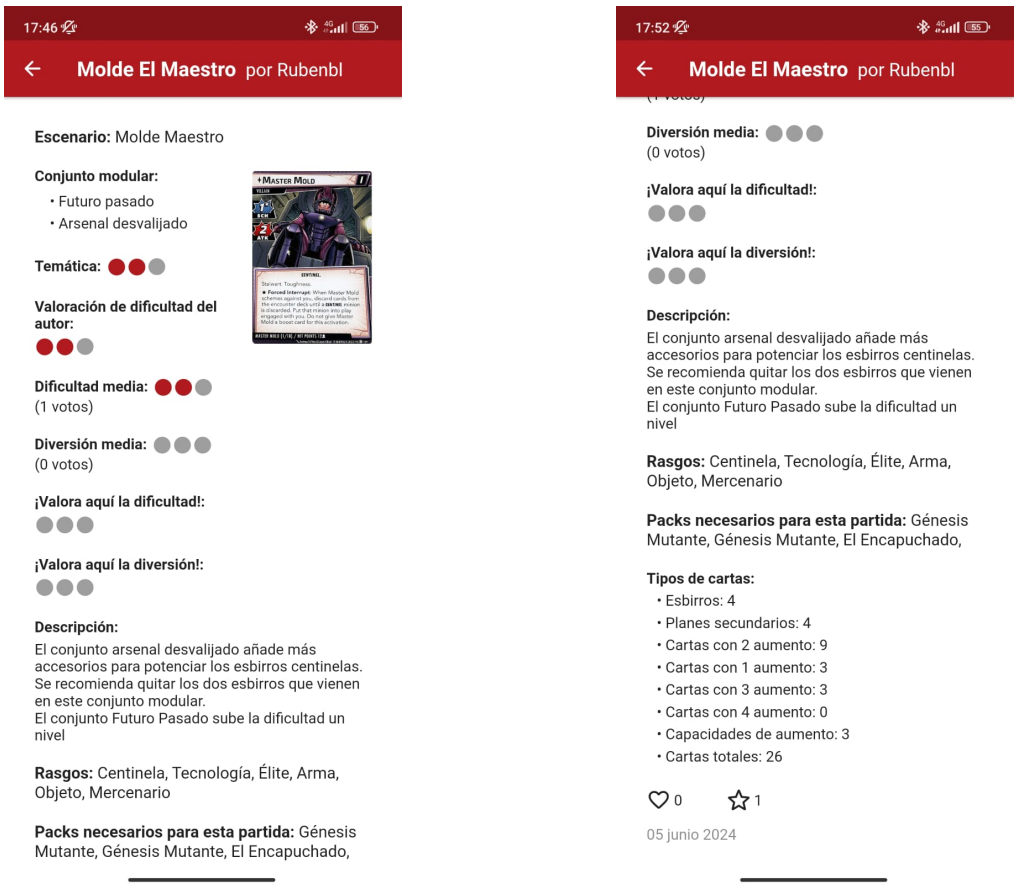


(a) Pantalla principal



(b) Pantalla principal (ordenar partidas)

Figura E.1: Pantalla principal

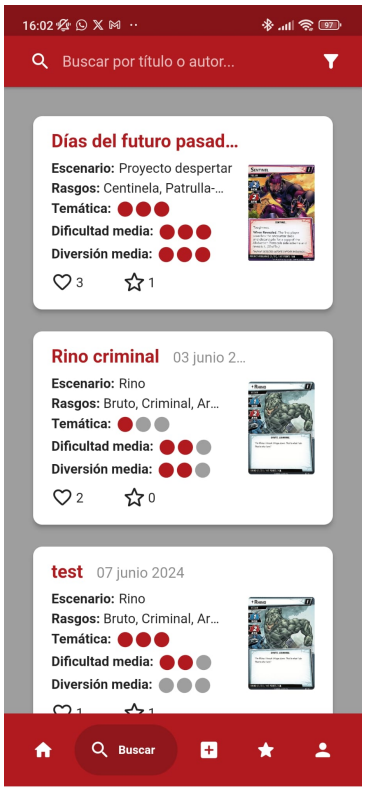


(a) Pantalla 1

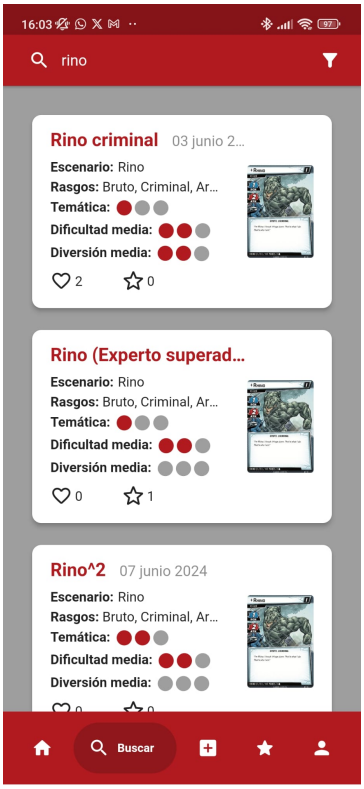
(b) Pantalla 2

Figura E.2: Pantalla de detalles de una partida

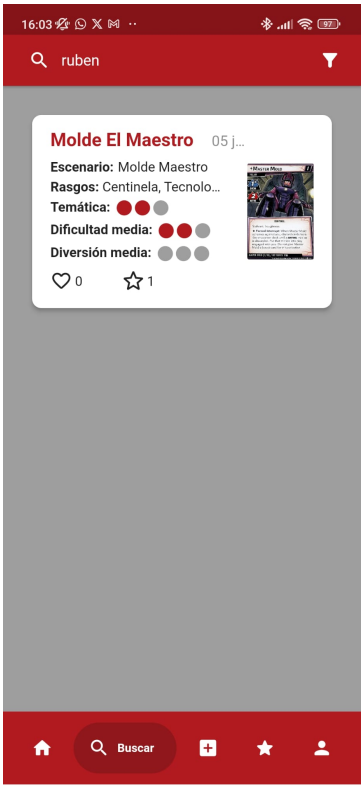
Si se navega hacia la pantalla de búsqueda lo primero que se ve es lo que aparece en la Figura E.3a. En esta pantalla, se pueden realizar tanto búsquedas por título de la partida como por nombre de usuario del autor de una partida como se muestra respectivamente en las Figuras E.3b y E.3c. En la barra de arriba se puede ver la entrada de texto por la que se busca, y un icono de filtrado en la esquina superior derecha. Si se clica este último se accede a la pantalla de filtrado de partidas por campos (ver Figura E.3d). Al realizar un filtrado por campos, como el que se muestra en la Figura E.3e habiendo filtrado por partidas con el escenario del villano Rino seleccionado, se navegará automáticamente a la pantalla anterior conteniendo esta únicamente las partidas con los campos que concuerdan con los filtros establecidos. Ahora en esta pantalla se muestra una tarjeta que indica que los filtros están aplicados, y en la que aparece un botón para poder eliminar los filtros establecidos (ver Figura E.3f).



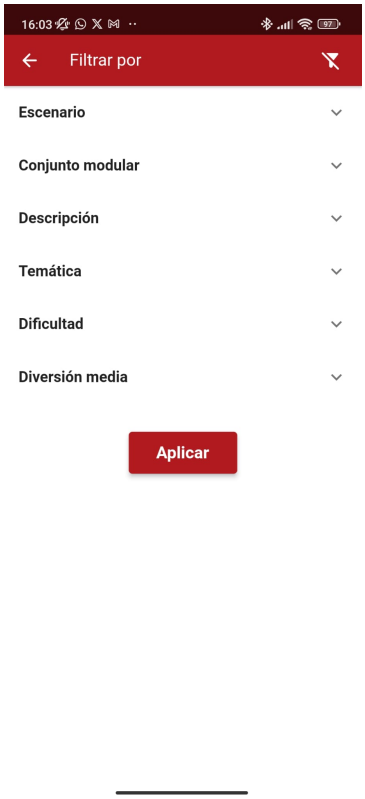
(a) Pantalla de búsqueda



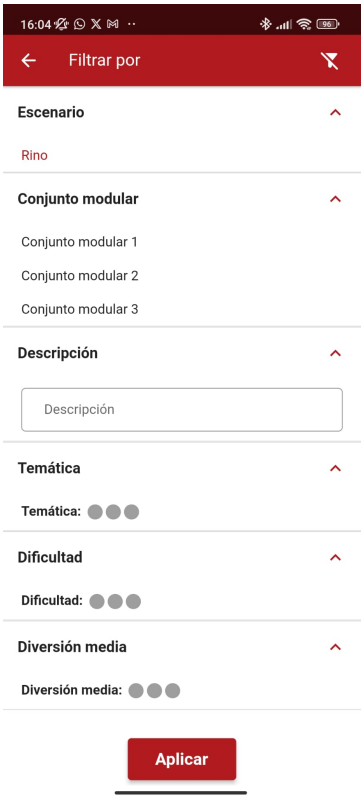
(b) Pantalla de búsqueda (búsqueda por título)



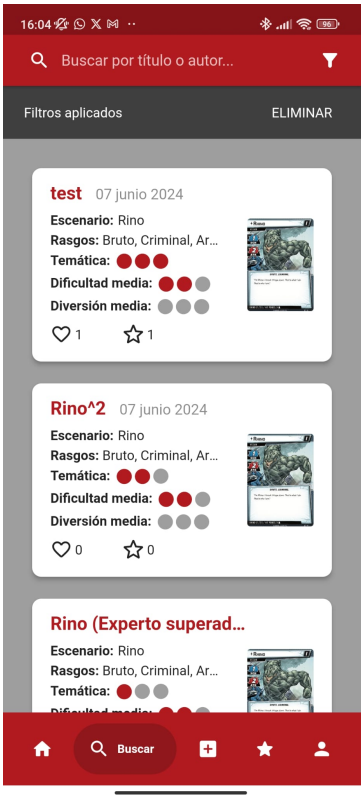
(c) Pantalla de búsqueda (búsqueda por autor)



(d) Pantalla de búsqueda (filtros)



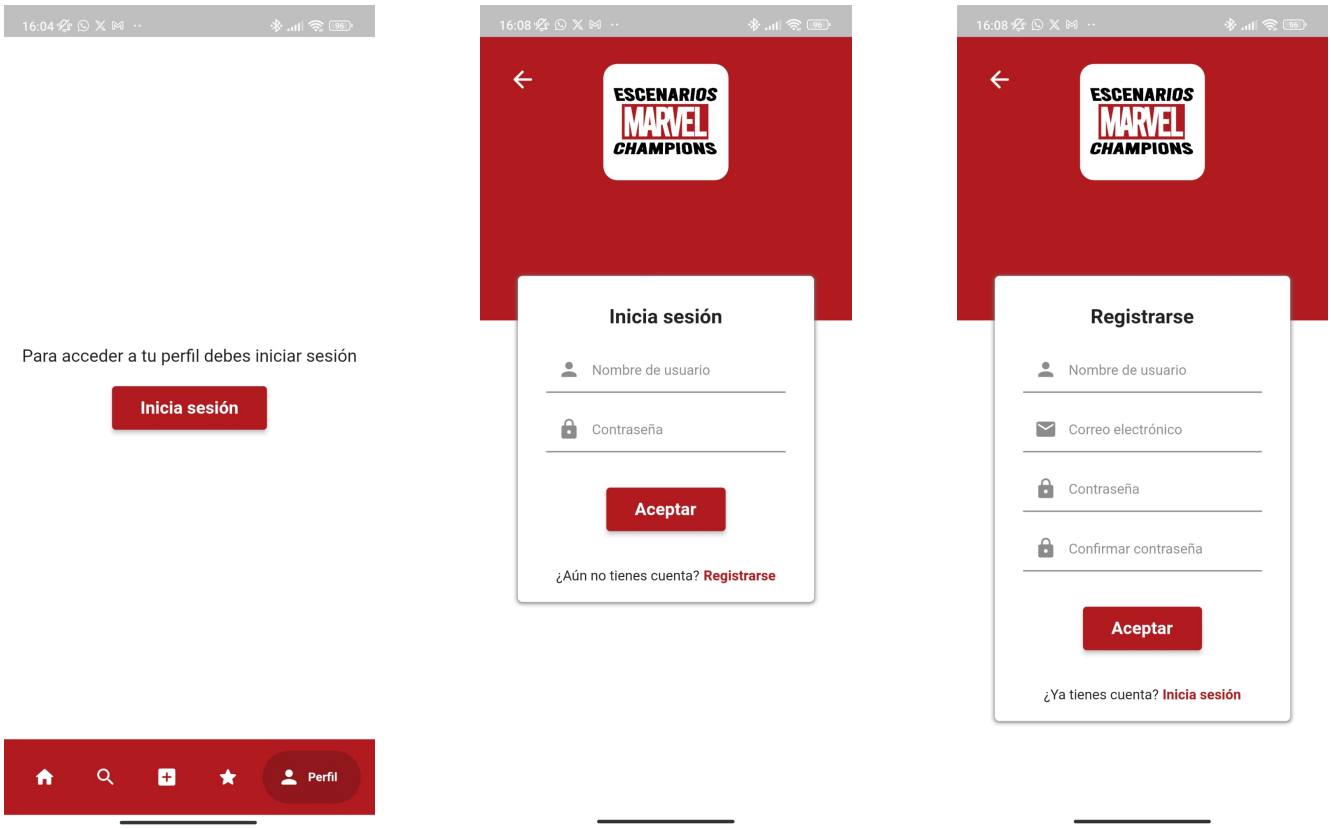
(e) Pantalla de búsqueda (filtros abiertos)



(f) Pantalla de búsqueda (filtros aplicados)

Figura E.3: Pantalla de búsqueda

Para el resto de funcionalidades es necerio iniciar sesión en la aplicación. Si se accede a cualquiera de las secciones restantes se muestra una pantalla como la mostrada en la Figura E.4a. Al clicar sobre el botón de inicio de sesión se accederá a la pantalla de la Figura E.4b, y si se desea realizar el registro, clicando en ”Registrarse” se accederá a la correspondiente pantalla (ver Figura E.4c).

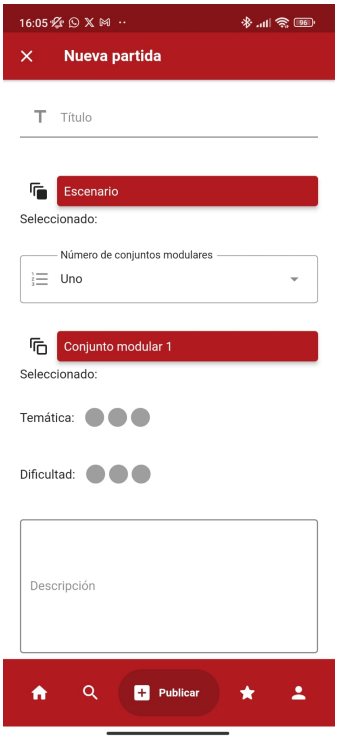


(a) Pantalla sin sesión iniciada (b) Pantalla de inicio de sesión (c) Pantalla de registro

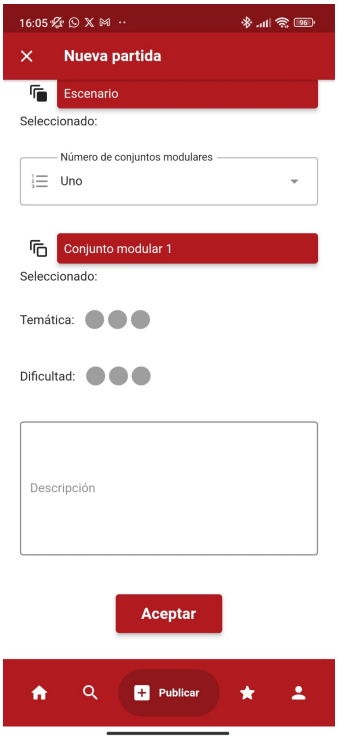
Figura E.4: Pantallas relacionadas con la sesión del usuario

Navegando hacia la pantalla para crear una nueva publicación, se muestra el formulario necesario para crearla (ver Figura E.5). Tras indicar la información que se desea que contenga esta nueva partida y clicar el botón de aceptar, la partida se creará y estará disponible para ser visualizada por todos los usuarios.

Las pantallas de selección de escenario y de conjuntos modulares tanto en la búsqueda por filtrado mencionada anteriormente como en el formulario para crear una nueva partida se muestran en las Figuras E.6a y E.6b respectivamente.



(a) Pantalla para crear una nueva partida 1



(b) Pantalla para crear una nueva partida 2

Figura E.5: Pantalla para crear una nueva partida

(a) Pantalla de selección de escenario

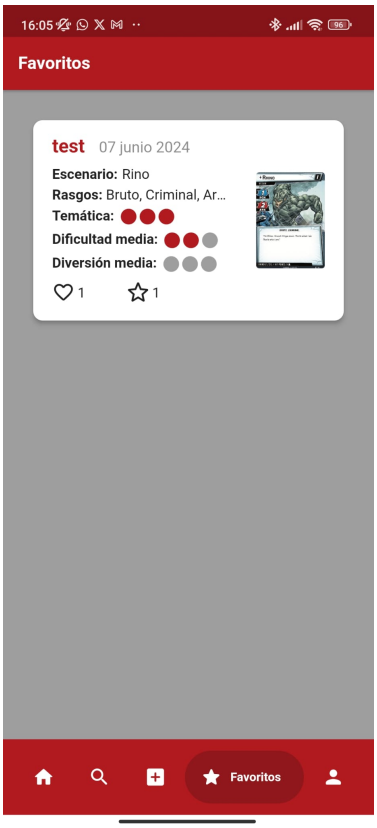
(b) Pantalla de selección de conjunto modular

Figura E.6: Pantallas de selección de escenario y de conjunto modular

Si se navega a la sección de favoritos se mostrará la pantalla que se ve en la Figura E.7. Aquí se

muestran las partidas añadidas a favoritos en caso de haber añadido alguna. Al igual que en la pantalla principal y en la de búsqueda, las tarjetas de las partidas son clicables y pulsando en ellas se accede a la pantalla de detalle de esas partidas.

(a) Pantalla de favoritos sin partidas añadidas



(b) Pantalla de favoritos con una partida añadida

Figura E.7: Pantallas de favoritos

En la pantalla del perfil se muestra el nombre usuario y la imagen (esta parte no está implementada ya que no forma parte de los objetivos del proyecto) y las partidas que ha creado el usuario, si es que ha creado alguna (ver Figura E.8). En esta pantalla, también se puede cerrar la sesión clicando el botón que aparece en la esquina superior derecha.

Por último, si se accede a la pantalla de detalles de una partida creada por el propio usuario, en la esquina superior derecha apareceran dos iconos, uno para eliminar la partida y otro para editarla (ver Figura E.9a). Si se pulsa en el icono del lapiz, se navega a la pantalla para editar la partida (ver Figura E.9b), y si se pulsa en el icono de la papelera, aparece un diálogo que pide confirmación al usuario para eliminar la partida (ver Figura E.9c).

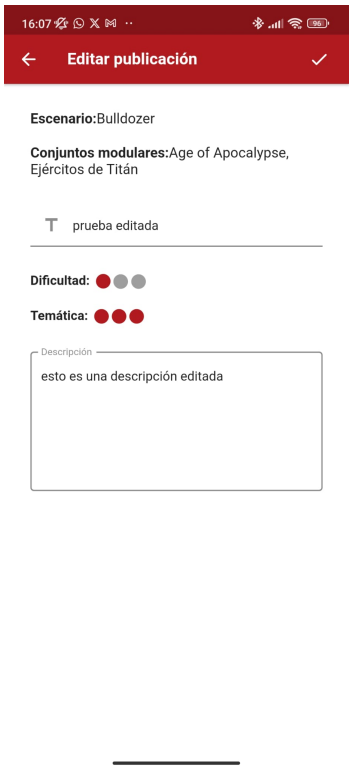
Cabe destacar que no se han añadido las capturas de la aplicación en web ya que la interfaz de usuario es prácticamente igual que en móvil, excepto el menú, que en web aparece en el lado izquierdo de la pantalla en vez de en la parte inferior. También hay pequeños cambios en el estilo de algunos botones y en la disposición de cierta información, pero nada importante que destacar. En la Figura E.10 se muestra el menú de la aplicación web en un navegador desde un ordenador y desde un móvil.

(a) Pantalla de perfil sin partidas

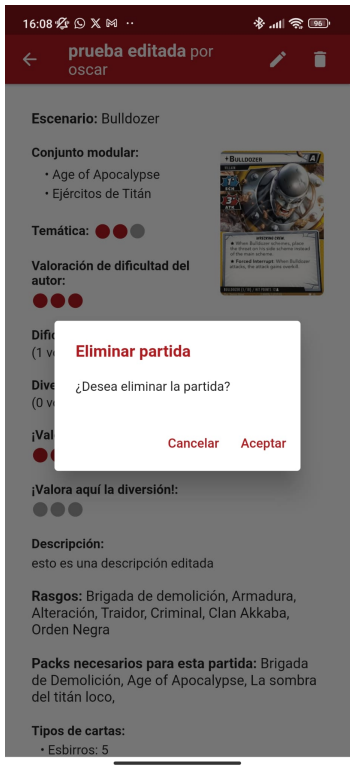
(b) Pantalla de perfil con una partida

Figura E.8: Pantalla de perfil

(a) Pantalla de detalles de una partida propia

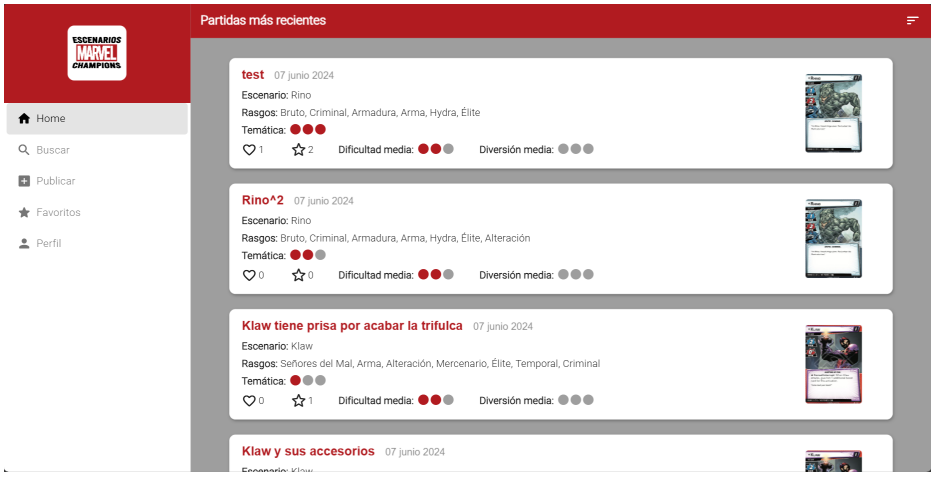


(b) Pantalla para editar una partida

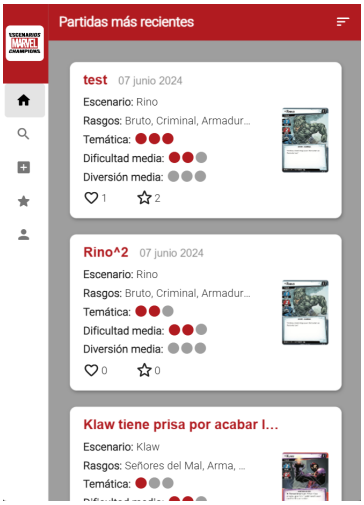


(c) Pantalla de detalles de una partida propia (eliminar)

Figura E.9: Pantalla de detalles de una partida propia



(a) Menú web



(b) Menú web desde navegador móvil

Figura E.10: Menú en la aplicación web

Bibliografía

[1] Marvel champions: The card game. Visitado: 2024-06-15. [Online]. Available: https://www.fantasyflightgames.com/en/products/marvel-champions-the-card-game/

[2] Marvelcdb. Visitado: 2024-05-28. [Online]. Available: https://marvelcdb.com/

[3] Board game stats. Visitado: 2024-05-28. [Online]. Available: https://www.bgstatsapp.com/

[4] Marvel champions tracker. Visitado: 2024-05-28. [Online]. Available: https://marvelchampionstracker.com/

[5] M champions lcg statistics. Visitado: 2024-05-28. [Online]. Available: https://play.google.com/store/apps/details?id=com.ingamedi.mvchamplcg&hl=es_419&gl=US&pli=1

[6] Flutter. Visitado: 2024-06-15. [Online]. Available: https://flutter.dev/

[7] Comparativa flutter vs react native. Visitado: 2024-02-07. [Online]. Available: https://www.linkedin.com/pulse/flutter-vs-react-native-2023-which-better-mobile-app/

[8] Comparativa flutter vs kotlin multiplatform. Visitado: 2024-02-07. [Online]. Available: https://scand.com/company/blog/kotlin-multiplatform-vs-flutter-making-the-right-choice-for-your-app/#:~:text=Flutter%20primarily%20focuses%20on%20Android,aligns%20with%20your%20platform%20strategy

[9] Node.js. Visitado: 2024-06-15. [Online]. Available: https://nodejs.org/en

[10] Javascript. Visitado: 2024-06-15. [Online]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript

[11] Ventajas node.js. Visitado: 2024-02-07. [Online]. Available: https://www.dongee.com/tutoriales/ventajas-y-desventajas-de-nodejs/

[12] Mysql web oficial. Visitado: 2024-02-12. [Online]. Available: https://www.mysql.com/

[13] Docker. Visitado: 2024-05-28. [Online]. Available: https://www.docker.com/

[14] Diseño centrado en el usuario: qué es, etapas y ejemplos. Visitado: 2024-06-15. [Online]. Available: https://blog.hubspot.es/website/diseno-centrado-usuario

[15] Diseño centrado en el usuario: una guía práctica. Visitado: 2024-06-15. [Online]. Available: https://carlosvicente.es/blog/diseno-centrado-en-el-usuario-una-guia-practica/

[16] Metodología iterativa e incremental. Visitado: 2024-02-06. [Online]. Available: https://medium.com/sue%C3%B1os-graficos/dise%C3%B1o-iterativo-la-metodolog%C3%ADa-que-perfeccionar%C3%A1-tus-proyectos-21034b0d277e

[17] Asus tuf gaming fx505dt-bq051. Visitado: 2024-02-07. [Online]. Available: https://www.asus.com/es/laptops/for-gaming/tuf-gaming/asus-tuf-gaming-fx505dd-dt-du/

[18] Xiaomi redmi note 11 pro. Visitado: 2024-02-07. [Online]. Available: https://www.mi.com/es/product/redmi-note-11-pro/

[19] Api android. Visitado: 2024-02-07. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/11926878?hl=es#:~:text=Cuando%20actualices%20tu%20aplicaci%C3%B3n%2C%20tendr%C3%A1s,nivel%2033%20de%20la%20API).

[20] Instalación de android studio. Visitado: 2024-02-07. [Online]. Available: https://developer.android.com/studio/install?hl=es-419

[21] Instalación de visual studio code. Visitado: 2024-02-07. [Online]. Available: https://code.visualstudio.com/download

[22] Instalación de mysql. Visitado: 2024-02-07. [Online]. Available: https://dev.mysql.com/downloads/installer/

[23] Android studio web oficial. Visitado: 2024-02-12. [Online]. Available: https://developer.android.com/

[24] Astah professional web oficial. Visitado: 2024-02-12. [Online]. Available: https://astah.net/products/astah-professional/

[25] Google chrome web oficial. Visitado: 2024-02-12. [Online]. Available: https://www.google.com/intl/es_es/chrome/

[26] Gitlab. Visitado: 2024-02-12. [Online]. Available: https://about.gitlab.com/

[27] Git. Visitado: 2024-06-16. [Online]. Available: https://www.git-scm.com/

[28] Overleaf editor online. Visitado: 2024-02-12. [Online]. Available: https://www.overleaf.com/

[29] Postman web oficial. Visitado: 2024-02-12. [Online]. Available: https://www.postman.com/

[30] Visual studio code web oficial. Visitado: 2024-02-12. [Online]. Available: https://code.visualstudio.com/

[31] Sueldo medio desarrollador software junior. Visitado: 2024-02-13. [Online]. Available: https://www.glassdoor.es/Sueldos/junior-software-developer-sueldo-SRCH_KO0,25.htm

[32] Precio astah professional. Visitado: 2024-02-13. [Online]. Available: https://astah.net/pricing/individual/

[33] Balsamiq. Visitado: 2024-06-18. [Online]. Available: https://balsamiq.com/

[34] Documentación oficial pub.dev de getx. Visitado: 2024-05-08. [Online]. Available: https://pub.dev/packages/get

[35] Usos de getx. Visitado: 2024-05-08. [Online]. Available: https://flutteracademy.app/exprime-flutter-con-getx/

[36] Dart programming lenguage | dart. Visitado: 2024-06-16. [Online]. Available: https://dart.dev/

[37] Estructura de un proyecto flutter. Visitado: 2024-05-12. [Online]. Available: https://medium.com/@jaimetellezb/flutter-estructura-de-proyecto-96ae96becf33#:~:text=dart_tool,la%20versi%C3%B3n%202.0%20de%20Dart.

[38] Explicación y diferencias entre pubspec.yaml y pubspec.lock. Visitado: 2024-05-12. [Online]. Available: https://stackoverflow.com/questions/63794639/difference-between-pubspec-lock-and-pubspec-yaml

[39] Fichero de opciones de análisis. Visitado: 2024-05-12. [Online]. Available: https://medium.com/comunidad-flutter/c%C3%B3digo-efectivo-en-tu-aplicaci%C3%B3n-flutter-desde-el-principio-b88715e90019

[40] Swagger. Visitado: 2024-06-18. [Online]. Available: https://swagger.io/

[41] Ficheros generados al crear un proyecto node.js. Visitado: 2024-05-12. [Online]. Available: https://medium.com/williambastidasblog/estructura-de-una-api-rest-con-nodejs-express-y-mongodb-cdd97637b18b

[42] Arquitectura limpia. Visitado: 2024-05-12. [Online]. Available: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

[43] Arquitectura limpia orientada a flutter. Visitado: 2024-05-12. [Online]. Available: https://nescalro.medium.com/entendiendo-a-la-arquitectura-limpia-7877ad3a0a47

[44] Arquitectura controller-service-repository. Visitado: 2024-05-14. [Online]. Available: https://tomcollings.medium.com/controller-service-repository-16e29a4684e5

[45] ¿que es un patrón de diseño? Visitado: 2024-05-16. [Online]. Available: https://refactoring.guru/es/design-patterns/what-is-pattern

[46] Clasificación de los patrones de diseño. Visitado: 2024-05-16. [Online]. Available: https://refactoring.guru/es/design-patterns/classification

[47] Explicación del patrón singleton. Visitado: 2024-05-16. [Online]. Available: https://medium.com/@pabloulloacastro/programaci%C3%B3n-orientada-a-objetos-patr%C3%B3n-singleton-423e2755614b

[48] Esquema del patrón singleton. Visitado: 2024-05-16. [Online]. Available: https://es.wikipedia.org/wiki/Singleton

[49] Explicación del patrón observador. Visitado: 2024-05-18. [Online]. Available: https://medium.com/@bromanv/patr%C3%B3n-observer-1f7f48958285

[50] Esquema del patrón observador. Visitado: 2024-05-18. [Online]. Available: https://reactiveprogramming.io/blog/es/patrones-de-diseno/observer

[51] Librería get_rx de getx. Visitado: 2024-05-18. [Online]. Available: https://pub.dev/documentation/get/latest/get_rx/get_rx-library.html

[52] Clase obx de getx. Visitado: 2024-05-18. [Online]. Available: https://pub.dev/documentation/get_server/latest/get_server/Obx-class.html

[53] Explicación del patrón estrategia. Visitado: 2024-05-18. [Online]. Available: https://medium.com/all-you-need-is-clean-code/patr%C3%B3n-estrategia-strategy-pattern-654c6e9d2abe

[54] Esquema del patrón estrategia. Visitado: 2024-05-18. [Online]. Available: https://reactiveprogramming.io/blog/es/patrones-de-diseno/strategy

[55] Explicación del patrón mvvm. Visitado: 2024-05-18. [Online]. Available: https://medium.com/@muzammalshahzad494/mvvm-design-pattern-d2fdda9028f

[56] Esquema del patrón mvvm. Visitado: 2024-05-18. [Online]. Available: https://www.researchgate.net/figure/The-Model-View-ViewModel-MVVM-architectural-pattern-In-MVVM-the-View-layer-is_fig3_275258051

[57] Explicación del patrón dao/dto. Visitado: 2024-05-21. [Online]. Available: https://www.oscarblancarteblog.com/2018/12/10/data-access-object-dao-pattern/

[58] ¿qué es un diagrama de despliegue? Visitado: 2024-05-26. [Online]. Available: https://miro.com/es/diagrama/que-es-diagrama-despliegue-uml/

[59] Librería flutter_test. Visitado: 2024-05-26. [Online]. Available: https://api.flutter.dev/flutter/flutter_test/flutter_test-library.html

[60] Librería jest. Visitado: 2024-05-26. [Online]. Available: https://jestjs.io/es-ES

[61] P. Efthymiou, *Clean Mobile Architecture*. Petros Efthymiou, 2022. [Online]. Available: https://www.petrosefthymiou.com/product-page/clean-mobile-architecture

[62] ¿qué son los mocks? Visitado: 2024-05-26. [Online]. Available: https://medium.com/orbismobile/conociendo-stubs-y-mocks-30689161f8a3#:~:text=Son%20aquellas%20funciones%20que%20se,realmente%20dependen%20de%20nuestro%20c%C3%B3digo.

[63] mocktail. Visitado: 2024-05-28. [Online]. Available: https://pub.dev/packages/mocktail

[64] Supertest. Visitado: 2024-05-28. [Online]. Available: https://www.npmjs.com/package/supertest

[65] ¿qué son los tests de usabilidad? Visitado: 2024-05-30. [Online]. Available: https://maze.co/guides/usability-testing/#:~:text=Usability%20testing%20is%20a%20type,%2Dto%2Duse%20products%20are.

[66] ¿la usabilidad puede medirse? escala sus y test de usuario. Visitado: 2024-06-04. [Online]. Available: https://www.flat101.es/blog/diseno-ux/la-usabilidad-puede-medirse-escala-sus-y-test-de-usuario/#:~:text=El%20sistema%20de%20escala%20de,con%20la%20interfaz%20de%20estudio.

[67] Cómo medir la usabilidad con un sus. Visitado: 2024-06-16. [Online]. Available: https://www.uifrommars.com/como-medir-usabilidad-que-es-sus/

[68] Measuring usability with the sus. Visitado: 2024-06-17. [Online]. Available: https://measuringu.com/sus/

[69] ¿qué es la escala de likert? Visitado: 2024-06-17. [Online]. Available: https://www.zendesk.com.mx/blog/que-es-escala-de-likert/

[70] Make android apps | flutter. Visitado: 2024-06-03. [Online]. Available: https://docs.flutter.dev/get-started/install/windows/mobile?tab=download

[71] Nginx. Visitado: 2024-06-18. [Online]. Available: https://nginx.org/en/

[72] Debian bookworm. Visitado: 2024-06-18. [Online]. Available: https://www.debian.org/releases/bookworm/