Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Mención en Ingeniería del Software

Diseño y desarrollo de una app iOS para la gestión de colecciones de cartas para 'Magic: The Gathering'

Alumno: Carlos Gutiérrez Casado

Tutor: Joaquín Nicolás Adiego Rodríguez

Índice

1. Introducción **7**

1.1. Motivación y objetivos . . . 7
1.2. Alcance . . . 8
1.3. Funcionalidades de la App . . . 8
1.3.1. Escaneo de una carta . . . 8
1.3.2. Búsqueda manual de una carta . . . 8
1.3.3. Historial de búsquedas de cartas . . . 8
1.3.4. Cartas favoritas . . . 8

2. Estado del arte **9**

2.1. ManaBox . . . 9
2.2. Capturas de pantalla . . . 9
2.3. Descarga . . . 10
2.4. Comparación . . . 10
2.5. Tecnología . . . 10
2.5.1. Sistema operativo . . . 10
2.5.2. Lenguaje de programación . . . 11
2.5.3. Entorno de desarrollo . . . 13

3. Planificación del proyecto **15**

3.1. Metodología de trabajo . . . 15
3.1.1. Metodología Scrum . . . 15
3.2. Plan de trabajo . . . 17
3.2.1. Planificación de tareas . . . 17
3.3. Gestión de riesgos . . . 18
3.4. Costes del proyecto . . . 19
3.4.1. Costes de personal . . . 19
3.4.2. Costes hardware y elementos físicos . . . 19
3.4.3. Costes software . . . 19
3.4.4. Costes adicionales . . . 20
3.5. Rentabilidad . . . 20
3.6. Modelo de Negocio . . . 20
3.6.1. Publicidad gratuita . . . 20
3.6.2. Membresía con ventajas . . . 21
3.7. Conclusiones modelo de negocio . . . 21

4. Análisis del sistema **23**

4.1. Actores . . . 23
4.2. Requisitos . . . 23

4.2.1. Requisitos funcionales . . . 23
4.2.2. Requisitos no funcionales . . . 24
4.3. Casos de Uso . . . 25
4.3.1. Diagrama de Casos de Uso . . . 25
4.3.2. Especificaciones de casos de uso . . . 26

5. Diseño **32**
5.1. Arquitectura . . . 32
5.1.1. Arquitectura lógica . . . 32
5.1.2. Beneficios patrón MVVM . . . 32
5.1.3. Componentes principales . . . 32
5.2. Modelo de diseño . . . 33
5.2.1. Diagrama de clases . . . 33
5.2.2. Diagrama de secuencia . . . 35
5.3. Diseño de interfaces . . . 37
5.3.1. Pantalla de búsqueda . . . 37
5.3.2. Pantalla de escaneo . . . 38
5.3.3. Vista de resultados . . . 39
5.3.4. Pantalla de detalle de una carta . . . 40
5.3.5. Pantalla historial de cartas . . . 41
5.3.6. Pantalla cartas favoritas . . . 42

6. Implementación **43**
6.1. Requerimientos Hardware y Software . . . 43
6.1.1. Hardware . . . 43
6.1.2. Software . . . 43
6.2. Herramientas empleadas . . . 43
6.2.1. Herramientas para el desarrollo de la aplicación . . . 43
6.2.2. Herramientas de soporte . . . 43
6.3. Organización interna del proyecto . . . 43
6.4. Complicaciones en la implementación . . . 44

7. Pruebas **46**
7.1. Pruebas de caja blanca . . . 46
7.2. Pruebas de caja negra . . . 46
7.3. Pruebas usuario . . . 48

8. Instalación de la Aplicación **49**
8.1. Conclusiones . . . 49
8.2. Caso específico para este TFG . . . 50

9. Conclusiones y trabajo futuro **51**
9.1. Conclusiones 51
9.2. Trabajo futuro 51

10. Referencias **53**
10.1. Bibliografía 53
10.2. Webgrafía 53
10.2.1. Webs principales 53
10.2.2. Otras webs 54

A. Repositorio para el código **55**

Índice de tablas

1. Ejemplo de tabla de Riesgos en el Desarrollo de Aplicaciones Móviles 18
2. Costes hardware y elementos físicos 19
3. Costes software 19
4. Requisitos funcionales 24
5. Requisitos no funcionales 24
6. CU-01 Buscar carta por nombre 26
7. CU-02 Escanear carta por nombre 26
8. CU-03 Acceder al listado de resultados de cartas 27
9. CU-04 Seleccionar carta buscada o escaneada 27
10. CU-05 Mostrar imagen carta seleccionada 27
11. CU-06 Mostrar ambas caras de una carta con doble cara 28
12. CU-07 Mostrar ediciones de una carta 28
13. CU-08 Mostrar arte de una edición de una carta 28
14. CU-09 Mostrar valor de una carta de una edición 29
15. CU-10 Mostrar legalidades de la carta 29
16. CU-11 Almacenar historial de cartas consultadas 29
17. CU-12 Mostrar historial de cartas consultadas 30
18. CU-13 Seleccionar carta del historial de cartas consultadas 30
19. CU-14 Añadir carta al listado de cartas favoritas 30
20. CU-15 Eliminar carta del listado de cartas favoritas 31
21. CU-16 Mostrar listado de cartas favoritas 31
22. CU-17 Seleccionar carta del historial de cartas favoritas 31
23. Pantalla de búsqueda 37
24. Pantalla de escaneo 38
25. Vista de resultados 39
26. Pantalla de detalle de una carta 40
27. Pantalla historial de cartas 41
28. Pantalla cartas favoritas 42

Índice de figuras

1. Diagrama casos de uso 25
2. Diagrama casos de uso 34
3. Diagrama de secuencia CU-01 Buscar carta por nombre 35
4. Diagrama de secuencia CU-02 Buscar carta escaneando su nombre 35
5. Diagrama de secuencia CU-04 Seleccionar carta buscada o escaneada 36
6. Diagrama de secuencia CU-14 Añadir carta al listado de cartas favoritas 36

1. Introducción

Magic: The Gathering, normalmente llamado Magic, MTG o Cartas Magic es un juego de cartas coleccionables diseñado en 1993 por el matemático americano Richard Garfield, y comercializado por la empresa Wizards of the Coast.

Actualmente cuenta con más de seis millones de jugadores repartidos por más de cincuenta y dos países, una cantidad aproximada de más de 26000 cartas publicadas (lanzando al mercado cada 3 meses nuevas cartas y colecciones) en diferentes ediciones e idiomas, y dos versiones digitales del juego, Magic Arena y Magin Online.

Magic representa la batalla entre dos o más magos, cada uno de ellos es un jugador, los cuales pueden usar hechizos, como Tierras, Artefactos, Criaturas, Instantáneos, Conjuros, ..., representado individualmente en cada carta, ya sea física o digital (en las versiones digitales del juego).

En la actualidad, el comercio de cartas entre jugadores mueve millones de dólares, superando con creces al mercado de material sellado, material nuevo que se compra directamente a distribuidores oficiales, ya que cada carta varía de precio a lo largo del tiempo dependiendo del uso que se le de en determinados formatos, la cantidad de unidades de la misma, etc, estableciendo el récord actual por una carta en los 3 millones de dólares. Siendo este mercado de cartas uno de los mayores alicientes para los jugadores, y uno de los principales motivos para la compra de material sellado, como son los sobres, en los que entran 15 cartas aleatorias de una edición, buscando el jugador la obtención o de una carta que le pueda servir para mejorar su mazos, como una carta que tenga valor y pueda vender para comprar con ese dinero una carta específica que necesite.

Por esto, este proyecto busca e implementa una solución para la rápida consulta de la propia información de una carta, las ediciones que existen, así como el precio de cada una y su legalidad en los diferentes formatos del juego.

1.1. Motivación y objetivos

El objetivo de este proyecto es realizar una aplicación móvil para dispositivos iOS que permita escanear o buscar el nombre de una carta Magic, y obtener de forma rápida toda la información relativa a esa carta, ya sean diferentes ediciones, impresiones de la misma, precios medios de las diferentes ediciones y versiones y legalidades en las diferentes modo de juego.

El objetivo secundario es poder acceder a la carta buscada o escaneada, o cualquiera de sus ediciones, de forma directa en la web más usada en Europa para la compra/venta de productos de Magic the Gathering.

1.2. Alcance

La App sólo la podrán usar usuarios que tengan un dispositivo móvil con sistema operativo iOS, es decir, un iPhone de la marca Apple, y debido a la continua consulta de información sobre una base de datos externa, es necesario que dispongan de conexión a internet.

La aplicación va destinada a cualquier tipo de usuario, ya que la curva de aprendizaje para su uso es mínima, pero debido a su nicho de mercado tan específico, será usada por usuarios que jueguen o coleccionen cartas de Magic The Gathering, por lo que estarán familiarizados con toda la información que aparezca en la App.

1.3. Funcionalidades de la App

1.3.1. Escaneo de una carta

Permite al usuario escanear el nombre de una carta Magic, obteniendo así de forma rápida toda la información de esa carta.

1.3.2. Búsqueda manual de una carta

Permite al usuario buscar manualmente una carta por su título. El título introducido puede ser completo o parcial, obteniendo todas las posibles cartas que cumplan con ese título parcial.

1.3.3. Historial de búsquedas de cartas

Pantalla en el que aparecerá el listado de cartas consultadas por el usuario.

1.3.4. Cartas favoritas

Pantalla en el que aparecerá un listado de las cartas marcadas como Favoritas por el propio usuario.

2. Estado del arte

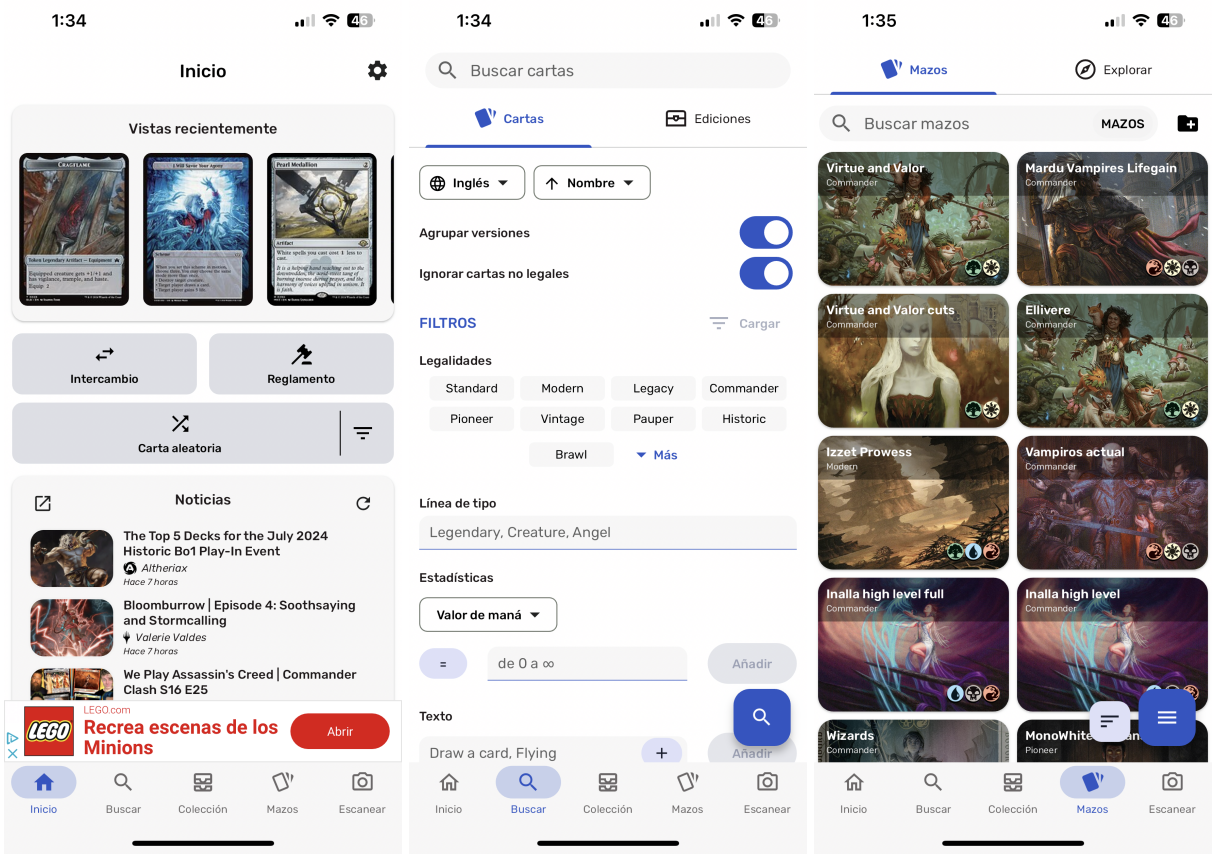
Una vez establecida la idea del proyecto, comparamos nuestro proyecto con las Apps similares disponibles en el mercado.

2.1. ManaBox

ManaBox es una aplicación móvil y una herramienta web diseñada para jugadores de Magic: The Gathering (MTG), siendo la más usada en el mercado, y a la que más se parece el proyecto desarrollado para este TFG.
La aplicación ofrece una serie de funciones para ayudar a los jugadores a gestionar sus colecciones de cartas y construir mazos, como funciones principales.
Al ser una aplicación poderosa y versátil, se ha convertido en una herramienta esencial para los jugadores de Magic: The Gathering. Su combinación de características avanzadas, facilidad de uso y accesibilidad la han convertido en la opción preferida para la gestión de cartas y la construcción de mazos.

2.2. Capturas de pantalla

En este apartado vamos a exponer una serie de capturas de pantalla en las que se podrá observar la interfaz de usuario de la aplicación



2.3. Descarga

ManaBox es una aplciación disponible tanto para el sistema operativo Android como iOS.
La web de descarga que nos redirige tanto a AppStore de Apple como a Google Play de Android es `https://manabox.app`

2.4. Comparación

La alternativa a nuestro proyecto que actualmente se encuentra en el mercado, ManaBox, como decimos está más enfocada a tener una gestión detallada de una colección propia dividida en mazos ya construidos de cartas, obteniendo informes detallados de cada mazo (estadísticas, potencial, etc ..). Siendo una opción disponible, pero secundaria, la funcionalidad de poder buscar/escanear una carta y su información.

La mayor diferencia con nuestro proyecto radica en la sencillez y rapidez que ofrecemos a la hora de poder buscar directamente una carta por si nombre, o directamente escanearla, obteniendo de una forma rápida y sencilla toda la información de la misma, incluyendo su precio medio de mercado y facilitando su compra, al enlazar directamente la carta buscada con la mayor página web de compra/venta de cartas Magic de Europa.

Otra diferencia importante respecto a ManaBox, es la complejidad frente a la sencillez y simplicidad, tanto en términos de búsqueda y consulta de cartas, como la usabilidad de disponer un historial de consultas o un listado de favoritos, evitando al usuario tener que realizar numerosas búsquedas redundantes de cartas.

2.5. Tecnología

2.5.1. Sistema operativo

Arquitectura

- El sistema operativo iOS, desarrollado por Apple , es uno de los sistemas operativos móviles más populares y avanzados del mundo. Lanzado por primera vez en 2007 junto con el iPhone original, iOS ha evolucionado significativamente para incluir una amplia gama de dispositivos, desde iPads hasta iPods y Apple Watches, pasando por las distintas gamas de ordenadores Mac.
- La arquitectura de iOS está diseñada para maximizar el rendimiento y la seguridad. Se compone de cuatro capas principales:
    - **Capa de Núcleo (Core OS)**: Proporciona servicios de bajo nivel, como la gestión de memoria, redes y el sistema de archivos. Esta capa se basa en el kernel XNU (X is Not Unix).
    - **Capa de Servicios Principales (Core Services)**: Incluye servicios esenciales como la base de datos SQLite, Core Data para la gestión de datos, y tecnologías de redes.

- **Capa de Medios (Media)**: Proporciona servicios para gráficos, audio y video. Esto incluye frameworks como Core Graphics, AVFoundation y Core Animation.
- **Capa de Cocoa Touch/SwiftUI**: Contiene las bibliotecas necesarias para la construcción de interfaces de usuario (UI). Incluye UIKit o SwiftUI.

Características clave

- iOS se distingue por una serie de características avanzadas:
  - **Interfaz de Usuario Intuitiva**: iOS es conocido por su facilidad de uso, con una interfaz intuitiva y accesible.
  - **App Store**: Ofrece millones de aplicaciones, lo que ha fomentado un ecosistema muy activo de desarrolladores.
  - **Seguridad**: Incorpora medidas avanzadas de seguridad, como el cifrado de datos, autenticación biométrica (Face ID y Touch ID), y un entorno de ejecución seguro.
  - **Rendimiento y Estabilidad**: Optimizado para funcionar de manera fluida en dispositivos Apple, garantizando una experiencia de usuario consistente y confiable.
  - **Integración con el Ecosistema Apple**: Permite una integración sin problemas con otros dispositivos Apple, como Mac, Apple Watch y Apple TV, a través de servicios como Handoff, Continuity y AirDrop.

Conclusiones

- iOS sigue siendo un pilar fundamental en la tecnología móvil, ofreciendo una combinación única de rendimiento, seguridad y usabilidad.
- Su evolución constante y su capacidad para adaptarse a las necesidades cambiantes de los usuarios y desarrolladores aseguran que continuará siendo una fuerza dominante en la industria tecnológica.

2.5.2. Lenguaje de programación

Para el desarrollo de este proyecto se ha utilizado el lenguaje de programación Swift junto con el framework SwiftUI.
SwiftUI es un framework moderno de desarrollo de interfaces de usuario (UI) creado por Apple para sus plataformas, incluidas iOS, macOS, watchOS y tvOS. Presentado en 2019, SwiftUI marca una evolución significativa en la forma en que se diseñan y construyen las interfaces de usuario en el ecosistema Apple.

Contexto

- SwiftUI se introdujo como parte de la estrategia de Apple para simplificar y modernizar el desarrollo de aplicaciones. Antes de SwiftUI, los desarrolladores utilizábamos principalmente UIKit para iOS y AppKit para macOS, junto con lenguajes de programación como Objective-C (lenaguaje que

aprendí en mis prácticas de empresa del Grado) y Swift. Aunque son frameworks muy poderoso, requerían una considerable cantidad de código y configuración manual.

- SwiftUI busca abordar estas complejidades ofreciendo un enfoque declarativo y más intuitivo para la creación de interfaces de usuario.

Principios fundamentales de SwiftUI

- **Enfoque declarativo**: En lugar de describir cómo construir la interfaz paso a paso, los desarrolladores declaran lo que la interfaz debería hacer. Esto hace que el código sea más legible y fácil de mantener.
- **Reactividad**: La interfaz de usuario se actualiza automáticamente cuando el estado de la aplicación cambia. Esto se logra mediante el uso de data bindings y observables.
- **Compatibilidad multiplataforma**: Con SwiftUI, los desarrolladores pueden crear interfaces de usuario para todas las plataformas de Apple con un solo conjunto de herramientas y un lenguaje unificado.

Características clave

- SwiftUI se distingue por una serie de características principales:

  - **Interfaz declarativa**: Permite describir la interfaz de usuario de manera clara y concisa.
  - **Actualizaciones en tiempo real**: Gracias a su integración con XCode, el IDE de Apple que explicaremos más adelante, los desarrolladores pueden ver los cambios en la interfaz de usuario en tiempo real mientras codifican. Aunque esto requiere un gran procesamiento en tiempo real.
  - **Animaciones simples**: SwiftUI facilita la creación de animaciones y transiciones con código simple.
  - **Accesibilidad integrada**: Proporciona herramientas para que las aplicaciones sean accesibles para todos los usuarios.
  - **Interoperabilidad con UIKit y AppKit**: Aunque SwiftUI está diseñado para ser usado de manera independiente, también puede integrarse con UIKit y AppKit, permitiendo a los desarrolladores adoptar SwiftUI gradualmente.

Conclusiones

- SwiftUI representa un avance significativo en el desarrollo de interfaces de usuario para las plataformas de Apple. Su enfoque declarativo, reactividad y compatibilidad multiplataforma lo hacen una herramienta poderosa y accesible para desarrolladores de todos los niveles.
- Este análisis proporciona una base sólida para comprender la importancia y las características de SwiftUI, así como su impacto en el desarrollo de aplicaciones.

2.5.3. Entorno de desarrollo

El entorno de desarrollo integrado usado en este proyecto es XCode, el entorno de desarrollo integrado (IDE) oficial de Apple Inc. para el desarrollo de aplicaciones en macOS, iOS, watchOS y tvOS. Lanzado por primera vez en 2003, Xcode se ha convertido en una herramienta esencial para desarrolladores que trabajan en el ecosistema de Apple.

Componentes principales

- **Editor de código**: Proporciona un entorno avanzado para escribir y editar código, con soporte para múltiples lenguajes de programación, incluyendo Swift y Objective-C. Ofrece características como el autocompletado, resaltado de sintaxis y la navegación de código.
- **Interface Builder**: Una herramienta visual para diseñar y probar interfaces de usuario. Permite arrastrar y soltar elementos de la interfaz, en caso de usar UIKit, o de ver los cambios en tiempo real en caso de usar SwiftUI.
- **Debugger**: Un depurador integrado que ayuda a los desarrolladores a identificar y corregir errores en el código. Ofrece características como puntos de interrupción (breakpoints), inspección de variables y análisis de memoria.
- **Simulator**: Emuladores de dispositivos iOS, watchOS y tvOS que permiten probar aplicaciones sin necesidad de hardware físico. Permite simular diferentes configuraciones de dispositivo y escenarios de uso.
- **Playgrounds**:Un entorno interactivo para probar y experimentar con código en Swift. Es ideal para aprendizaje y desarrollo rápido de aplicaciones. Los playgrounds son posiblemente el mejor recurso para hacer pruebas rápidas de desarrollo de funciones y métodos.

Características clave

- **Integración con Git**: Soporte para sistemas de control de versiones como Git, facilitando la colaboración y gestión del código fuente.
- **Soporte para Swift y Objective-C**: Ofrece un entorno optimizado para trabajar con ambos lenguajes de programación.
- **Compilación y ejecución rápida**: Herramientas de optimización que mejoran los tiempos de compilación y ejecución de aplicaciones.
- **Documentación y ayuda integrada**: Acceso a documentación completa y tutoriales directamente desde el IDE.
- **App Store Connect**: Integración para gestionar y distribuir aplicaciones en el App Store.

Conclusiones

- XCode es un entorno de desarrollo robusto y completo, ha facilitado la creación de aplicaciones de alta calidad para el ecosistema Apple.
- La introducción de nuevas tecnologías como Swift y SwiftUI ha permitido que los desarrolladores tengan acceso a las últimas herramientas y frameworks disponibles.
- Destaca por su facilidad de uso, simplicidad y potencia.

3. Planificación del proyecto

En este apartado vamos a describir la metodología empleada para la realización del proyecto, así como su distribución en las diferentes fases de trabajo.

3.1. Metodología de trabajo

Para este proyecto se ha decidido seguir una metodología agile, en concreto la metodología Scrum, que explicaremos con más detalle a continuación.

3.1.1. Metodología Scrum

La metodología Scrum es un marco ágil ampliamente utilizado para el desarrollo de software, incluyendo el desarrollo de aplicaciones móviles. Este enfoque se centra en la colaboración, la flexibilidad y la entrega rápida de valor al cliente.

Principios y Estructura

1. Equipos Auto gestionados y Multifuncionales
   Los equipos Scrum están formados por desarrolladores, diseñadores, testers y otros roles necesarios para completar el trabajo. Estos equipos son auto gestionados y colaboran estrechamente.

2. Roles en Scrum

   - **Product Owner**: Responsable de definir y priorizar las características y requisitos de la aplicación (Product Backlog) según el valor que aportan al cliente.
   - **Scrum Master**: Facilita el proceso Scrum, elimina impedimentos y asegura que el equipo siga los principios ágiles.
   - **Equipo de Desarrollo**: Encargado de diseñar, desarrollar, probar e implementar las funcionalidades de la App.

3. Eventos de Scrum

   - **Sprint**: Ciclo de trabajo fijo, generalmente de dos a cuatro semanas, durante el cual se desarrolla un incremento del producto.
   - **Planificación del Sprint**: Reunión para determinar qué elementos del Product Backlog se abordarán en el próximo Sprint y cómo se trabajarán.
   - **Reunión Diaria (Daily)**: Reunión de pie de 15 minutos donde el equipo sincroniza actividades y ajusta su plan de trabajo diario.
   - **Revisión del Sprint (Sprint Review)**: Reunión al final del Sprint donde se presenta el incremento desarrollado y se recibe feedback del Product Owner.

- **Retrospectiva del Sprint (Sprint Retrospective)**: Reunión para reflexionar sobre el Sprint que terminó y buscar maneras de mejorar el proceso y la colaboración en el próximo Sprint.

4. Aplicación de Scrum en el Desarrollo de Apps Móviles

   - **Iteraciones Cortas y Flexibles**: La naturaleza iterativa de Scrum permite que las Apps móviles se desarrollen en pequeños incrementos, lo que facilita la adaptación a los cambios rápidos del mercado y a las nuevas tecnologías.
   - **Entrega Continua de Valor**: Al final de cada Sprint, se presenta una versión funcional de la App, lo que permite al Product Owner revisar y proporcionar feedback continuo, asegurando que el producto evoluciona según las necesidades del usuario.
   - **Priorización y Adaptación**: El Product Backlog se prioriza continuamente, asegurando que el equipo trabaja en las funcionalidades de mayor valor para los usuarios. Esto es crucial en el desarrollo de Apps móviles, donde las demandas del usuario pueden cambiar rápidamente.
   - **Colaboración y Comunicación Constante**: Las reuniones diarias y otros eventos de Scrum fomentan la comunicación constante dentro del equipo, lo que es esencial para resolver problemas rápidamente y mantener a todos alineados con los objetivos del proyecto.
   - **Pruebas y Calidad**: La integración de testers en el equipo de desarrollo asegura que las funcionalidades se prueben continuamente, lo que mejora la calidad y la estabilidad de la App.

5. Beneficios de Usar Scrum en el Desarrollo de Apps Móviles

   - **Mayor Flexibilidad y Adaptabilidad**: La capacidad de ajustar prioridades y cambiar direcciones rápidamente en respuesta a las necesidades del mercado y el feedback de los usuarios.
   - **Mejora Continua**: La retrospectiva del Sprint permite identificar áreas de mejora continua en el proceso de desarrollo.
   - **Satisfacción del Cliente**: Entregar incrementos funcionales y recibir feedback constante asegura que el producto final cumpla con las expectativas del cliente y del usuario final.
   - **Transparencia y Visibilidad**: Las reuniones y los artefactos de Scrum proporcionan una visión clara del progreso y los desafíos, facilitando la toma de decisiones informadas.

   En resumen, Scrum proporciona una estructura efectiva para gestionar la complejidad y la dinámica del desarrollo de Apps móviles, asegurando que los equipos puedan entregar productos de alta calidad de manera eficiente y alineada con las necesidades del mercado.

3.2. Plan de trabajo

El plan de trabajo se utiliza para poder planificar, organizar y gestionar correctamente la evolución del proyecto. En este caso, siguiendo una metodología Scrum, se ha decidido seguir una planificación de tareas por Sprints, que se detalla a continuación.

3.2.1. Planificación de tareas

1. **Sprint 1: Planificación (Semana 1 - Semana 2)**
    - Definir el alcance del proyecto: 5 horas
    - Requisitos y especificaciones: 5 horas
    - Planificación del proyecto y diseño del diagrama de Gantt: 10 horas
2. **Sprint 2: Diseño (Semana 3 - Semana 5)**
    - Diseño de la arquitectura de la aplicación: 10 horas
    - Diseño de la interfaz de usuario (UI): 20 horas
    - Revisión y ajustes del diseño: 10 horas
3. **Sprint 3: Desarrollo (Semana 6 - Semana 11)**
    - Configuración del entorno de desarrollo: 5 horas
    - Desarrollo del módulo de búsqueda: 60 horas
    - Desarrollo del módulo de escaneo de cartas: 55 horas
    - Desarrollo del módulo de historial de cartas: 10 horas
    - Desarrollo del módulo de cartas favoritas: 10 horas
    - Integración de módulos: 10 horas
4. **Sprint 4: Pruebas (Semana 12 - Semana 13)**
    - Pruebas funcionales y de integración: 20 horas
    - Pruebas de usabilidad y ajustes: 20 horas
5. **Sprint 5: Documentación (Semana 14 - Semana 15)**
    - Documentación técnica: 35 horas
    - Revisión y corrección de la documentación: 15 horas

3.3. Gestión de riesgos

La gestión de riesgos nos permite analizar, identificar y gestionar posibles riesgos e imprevistos que puedan surgir durante el desarrollo del proyecto.

Para ejemplificar una posible gestión de riesgos en un entorno profesional real, compuesto por varios equipos de desarrollo, he realizado una tabla en la que se plasma una gestión de riesgos real, pudiendo aplicar a este proyecto algún riesgo identificado que aparece en dicha tabla, ya que al estar el proyecto desarrollado por una única persona, desde las fase de análisis, hasta la posterior implementación y testeo, hay riesgos que sería muy difícil que se produjeran.

Riesgo	**Probabilidad**	**Impacto**	**Estrategia de Mitigación**
Cambio de requisitos del cliente	Alta	Alta	Realizar reuniones frecuentes con el cliente para revisar y ajustar los requisitos. Usar el Product Backlog para priorizar y gestionar los cambios.
Problemas de comunicación en el equipo	Media	Media	Fomentar reuniones diarias (Daily Scrum) y utilizar herramientas de colaboración como Slack o Microsoft Teams. Realizar actividades de team building.
Retrasos en la entrega de componentes críticos	Alta	Alta	Implementar la integración continua y las pruebas automatizadas para identificar problemas rápidamente. Descomponer las tareas en entregas más pequeñas y manejables.
Baja calidad del código	Media	Alta	Realizar revisiones de código regulares y establecer estándares de codificación. Integrar pruebas unitarias y de integración en el proceso de desarrollo.
Falta de experiencia en tecnología específica	Media	Alta	Proporcionar formación y recursos adicionales al equipo. Contratar consultores o expertos temporales si es necesario.

Tabla 1: Ejemplo de tabla de Riesgos en el Desarrollo de Aplicaciones Móviles

3.4. Costes del proyecto

3.4.1. Costes de personal

Los costes de personal son los costes relacionados a las personas que han participado en el desarrollo del proyecto. Normalmente en un entorno profesional en el desarrollo de una App móvil se estipula un coste por hora de cada desarrollador/analista/diseñador UI/etc participante, siendo ese coste distinto si el empleado tiene categoría junior, mid o senior, y a mayores un coste por hora relacionado con la jefatura de proyecto, siendo este coste mayor que los anteriores.

En concreto, en este proyecto los costes son nulos debido a que no se ha realizado en un entorno empresarial real y a que todos los roles han sido cubiertos por la misma persona.

3.4.2. Costes hardware y elementos físicos

Estos costes son los relacionados con el hardware necesario para la realización del proyecto, y además, en este caso en concreto, han sido necesarias una serie de cartas Magic para poder ir probando la App.

Descripción	Coste(€)
MacBook Pro	1900
Monitor MSI	250
iPhone13	700
Teclado Apple	85
Raton Apple	65
Cartas Magic	10

Tabla 2: Costes hardware y elementos físicos

3.4.3. Costes software

Costes relacionados con el software necesario para la realización del proyecto.

Descripción	Coste(€)
macOS	0
XCode	0
iOS	0
GitHub	0
Scryfall API	0
Overleaf	0

Tabla 3: Costes software

3.4.4. Costes adicionales

Costes indirectamente relacionados con el proyecto, pero indispensables para la realización del mismo, como puede ser conexión a Internet, 45€/mes y gasto en electricidad 49€/mes.

3.5. Rentabilidad del proyecto Los gastos futuros que puede implicar este proyecto:

- **Cuenta Apple Developer Program**: Es necesario tener una cuenta suscrita al Apple Developer Program poder publicar la App en la Apple Store, esto implica un coste anual de 99 dólares, permitiéndonos publicar un número indeterminado de Apps, así como de versiones de una misma App.
- **Sueldos nuevo personal**: Si se quieren aumentar las funcionalidades de la App, como por ejemplo desarrollar una gestión de colecciones propias distribuidas por mazos, ediciones etc.. requeriría una grandiosa carga de trabajo para una persona solo, por lo que habría que contratar un equipo que pudiera ayudar al desarrollo, diseño e implementación de las nuevas funcionalidades.

3.5. Rentabilidad

Esta App de momento no se va a comercializada, por lo que ahora mismo la rentabilidad sería nula. Pero en un futuro si se decidiese comercializar, se podrían explorar diferentes vías de rentabilidad:

- Gestión de usuarios de pago, permitiendo a los usuarios con una cuenta acceder a determinadas funcionalidades de la App que no estén habilitadas para los usuarios sin cuenta, fomentando mediante este sistema de contenido limitado el que los usuarios accediesen a pagar una cuenta.
- Publicidad en la App para los usuarios sin cuenta, haciendo que dicha publicidad no fuera visible por los usuarios con cuenta, fomentando así que la gente pagase una cuenta de usuario

3.6. Modelo de Negocio

El modelo de negocio con este tipo de Apps suele basarse en dos estrategias principalmente:

- Publicidad gratuita por la App.
- Membresía mensual o anual con ciertas ventajas.

3.6.1. Publicidad gratuita

Es el método más usado, liberar las funcionalidades básica de la App sin coste de descarga ni coste de membresía, pero a cambia la visualización de publicidad en la propia App. Es un método muy efectivo ya que con las funcionalidades básicas la gente suele conformarse, si que es verdad que en este tipo de App, estas funcionalidades básicas irían destinadas a jugadores muy casuales de Magic: The Gathering.
Para jugadores experimentados y no solo casuales, este tipo de funciones básicas se quedan cortas, es ahí donde entraría el segundo tipo de modelo de negocio.

3.6.2. Membresía con ventajas

Dado que el juego Magic The Gathering, es un juego en el que se suele gastar dinero, este tipo de modelo negocio es muy muy rentable, ya que, como hemos expuesto antes, salvo los jugadores más casuales, que apenas jueguen, el resto de jugadores tienen una gran cantidad de cartas y mazos, por lo que tener un control preciso de todas sus cartas, su valor y sus mazos es algo sumamente importante.
Mediante este modelo de negocio se intenta abordar la problemática que se encuentra un jugador asiduo a Magic, para gestionar y controlar su colección de cartas, ya que con las funcionalidades básicas no le va a ser posible.
Este modelo de negocio es más fructífero que la publicidad en la App, ya que, según un estudio realizado entre mi grupo de amigos y la gente que va a jugar a la tienda, 8 de cada 10 personas pagan por la suscripción de la App que hay en el mercado parecida a la App desarrollada para este proyecto, ManaBox, la cual tiene una opción PRO que valora en 2,49 euros mensuales o 24,99 euros en un pago único.
Las diferencia que se podrían fijar entre la versión gratuita y la de pago, es el limitar el número de cartas/mazos/colecciones/carpetas que pueda gestionar un usuario de la aplicación.
Teniendo en cuenta que no podemos simplificar en exceso las funcionalidades de la versión gratuita, ya que esto podría implicar que mucho usuarios se sintieran obligados a pagar si o si una versión PRO, hecho que llevaría a suscitar cierta animadversión respecto a la plataforma.
La versión Pro aparte de ofrecer un número ilimitado de cartas/mazos/colecciones gestionadas, es imperativo que no haya publicidad, ya que es lo que más afecta a la experiencia de usuario dentro de una App.

3.7. Conclusiones modelo de negocio

Las conclusiones a las que se puede llegar después de estudiar el modelo de negocio son que este tipo de aplicaciones, por su ecosistema y los potenciales usuarios son una fuente de ingresos bastante segura, ya que la proporción de usuarios que usan este tipo de Apps de forma gratuita respecto a los que la usan con al versión PRO es muy muy baja.
Como he mencionado antes, haciendo un estudio en al tienda en la que juego yo tanto de mi grupo de amigos como de jugadores en general que van ahí a jugar, he calculado que sólo de esta gente, la App con la que podríamos comparar este proyecto, genera aproximadamente unos 80-100€ al mes.
Todo esto teniendo en cuenta que es una tienda pequeña de barrio, en una ciudad no muy grande como Valladolid, implica una fuente de ingresos bastante sustancial si lo extrapolamos a las tiendas que hay en una ciudad como Madrid, dado que en la zona centro hay unas 5-6 tiendas de cartas Magic, con mucha más afluencia que en la que he hecho el estudio.
Si hiciéramos un calculado aproximado de una tienda de estas, fácilmente, por tienda el ingreso podría ser de unos 400 euros, teniendo en cuenta la gente que acude a torneos que organizan en estas tiendas, y la gente que acude a los torneos que organiza la tienda a la que voy yo (el número de participantes es fácilmente 4-5 veces más).

Es decir, implantar un modelo de negocio para sacar beneficios a una aplicación móvil de este tipo, sería muy sencillo, y se tendría beneficios asegurados, dado la inversión económica necesaria para llevarlo a cabo.

4. Análisis del sistema

En este punto se expondrán los aspectos considerados antes de empezar el diseño e implementación del proyecto. Incluyendo actores, requisitos y especificaciones que debe cumplir el mismo.

4.1. Actores

Representan los agentes que participan en el sistema.

- **Usuario**: Cualquier persona que use la App.
- **Scryfall**: Página web que contiene los datos de todas las cartas Magic.
- **UserDefaults**: Estructura nativa de iOS que permite almacenar datos de forma consistente, mientras esos datos no sean lo suficientemente grandes como para necesitar una base de datos externa.

4.2. Requisitos

Los requisitos de un proyecto de software son las funciones, características y restricciones que debe cumplir el producto final.
En otras palabras, los requisitos definen qué debe hacer el software, cómo debe verse y las condiciones que deben cumplirse para que se considere exitoso.
Se dividen en dos categorías, **Requisitos funcionales** y **Requisitos no funcionales**
Como breve resumen introductorio, los requisitos funcionales definen lo que el sistema debe hacer, mientras que los requisitos no funcionales describen cómo debe hacerlo.

4.2.1. Requisitos funcionales

Los requisitos funcionales describen qué debe hacer el sistema de software de manera específica. Se centran en las funciones y características tangibles que el software debe proporcionar.
Los requerimientos funcionales son cruciales porque determinan cómo los usuarios interactuarán con el software y qué operaciones podrán realizar. Además, sirven como base para el diseño e implementación del sistema.
En este proyecto se han identificado los siguientes requisitos funcionales:

ID	Descripción
RF-01	El Sistema permitirá buscar cartas mediante el nombre o palabras clave
RF-02	El Sistema permitirá buscar cartas mediante el reconocimiento y escaneo de su nombre
RF-03	El Sistema mostrará una lista de resultados coincidentes con el nombre buscado/escaneado
RF-04	El Sistema permitirá la selección de un resultado de la tabla de resultados de cartas
RF-05	El Sistema mostrará una imagen de la carta seleccionada
RF-06	El Sistema permitirá poder ver ambas caras de las cartas que tengan varias caras
RF-07	El Sistema mostrará un listado de las diferentes ediciones de la carta
RF-08	El Sistema permitirá ver el arte de una edición de una carta de forma ampliada
RF-09	El Sistema mostrará el valor actual medio de cada edición de la carta en el mercado
RF-10	El Sistema mostrará la legalidad de la carta en los diferentes modos de juego
RF-11	El Sistema guardará un historial de las cartas consultadas
RF-12	El Sistema permitirá acceder a un historial de cartas consultadas
RF-13	El Sistema permitirá seleccionar una carta del listado del historial de cartas consultadas
RF-14	El Sistema permitirá añadir una carta al listado de cartas favoritas
RF-15	El Sistema permitirá eliminar a un listado de cartas favoritas
RF-16	El Sistema mostrará un listado de cartas favoritas
RF-17	El Sistema permitirá seleccionar una carta del listado de cartas favoritas

Tabla 4: Requisitos funcionales

4.2.2. Requisitos no funcionales

Los requisitos no funcionales describen cómo debe comportarse el sistema, en lugar de las funciones específicas que debe realizar.
Abordan características como rendimiento, seguridad, usabilidad, escalabilidad y otros aspectos de calidad.
Por ejemplo, los requisitos no funcionales pueden especificar la rapidez de respuesta del sistema, la eficiencia del uso de recursos o la seguridad del software.
Los requisitos no funcionales son las restricciones o requisitos impuestos al sistema.
Se han identificado los siguientes requisitos no funcionales:

ID	Descripción
RNF-01	El Sistema deberá ser intuitivo y fácil de usar
RNF-02	El Sistema podrá ser ejecutado en versiones actuales de iOS
RNF-03	El Sistema será adaptable a los diferentes tamaños de pantalla de dispositivos iOS
RNF-04	El Sistema deberá tener un tiempo de respuesta inferior a 5 segundos
RNF-05	El Sistema deberá gestionar de forma eficiente el almacenamiento de datos

Tabla 5: Requisitos no funcionales

4.3. Casos de Uso

En este apartado trataremos el modelo de los casos de uso del sistema en base a los requisitos previamente establecidos.

4.3.1. Diagrama de Casos de Uso

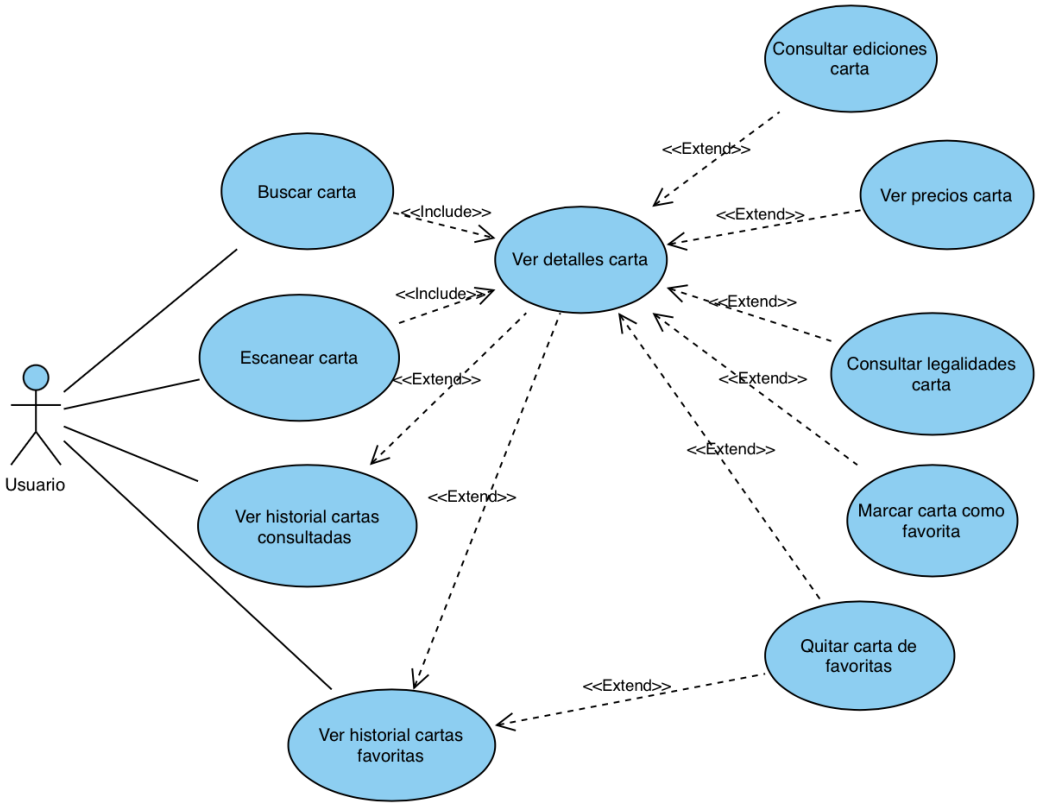


Figura 1: Diagrama casos de uso

4.3.2. Especificaciones de casos de uso

Identificador	CU-01 Buscar carta por nombre
Descripción	Caso de uso correspondiente al RF-01
Actores	Usuario
Precondición	
Flujo principal	1. El Usuario empieza a escribir en el cuadro de texto 2. El Sistema lee los datos y activa el botón buscar si hay más de tres caracteres escritos 3. El Usuario pulsa el botón buscar 4. El Sistema busca las cartas cuyo nombre contenga el texto buscado
Flujo alternativo	
Postcondición	El usuario verá un listado de cartas cuyos nombres contienen el texto buscado

Tabla 6: CU-01 Buscar carta por nombre

Identificador	CU-02 Buscar carta escaneando su nombre
Descripción	Caso de uso correspondiente al RF-02
Actores	Usuario
Precondición	
Flujo principal	1. El Usuario pulsa el botón de escanear en la pantalla de búsqueda 2. El Sistema abre la cámara del dispositivo mostrando en la pantalla la vista de la cámara con un recuadro central donde encuadrar el nombre de la carta 3. El Usuario centra el nombre de la carta en el recuadro de escaneo central de la pantalla 4. El Sistema reconoce el texto automáticamente y realiza la búsqueda.
Flujo alternativo	
Postcondición	El usuario verá un listado de cartas cuyos nombres contienen el texto escaneado

Tabla 7: CU-02 Escanear carta por nombre

Identificador	CU-03 Acceder al listado de resultados de cartas
Descripción	Caso de uso correspondiente al RF-03
Actores	Usuario
Precondición	El Usuario ha escaneado o buscado previamente el nombre de una carta
Flujo principal	1. El Sistema muestra un listado de cartas que contengan en el nombre el texto escaneado o buscado
Flujo alternativo	1a. Si no hay ninguna carta que contenga en el nombre el texto escaneado o buscado, el Sistema mostrará un mensaje avisando que no hay cartas disponibles
Postcondición	

Tabla 8: CU-03 Acceder al listado de resultados de cartas

Identificador	CU-04 Seleccionar carta buscada o escaneada
Descripción	Caso de uso correspondiente al RF-04
Actores	Usuario
Precondición	El Sistema muestra como resultado el listado de cartas
Flujo principal	1. El Usuario selecciona una carta del listado de cartas
Flujo alternativo	
Postcondición	La carta seleccionada aparecerá en el historial de cartas consultadas

Tabla 9: CU-04 Seleccionar carta buscada o escaneada

Identificador	CU-05 Mostrar imagen carta seleccionada
Descripción	Caso de uso correspondiente al RF-05
Actores	Usuario
Precondición	El Usuario ha seleccionado una carta del listado de cartas
Flujo principal	1. El Sistema mostrará una imagen de la carta buscada o seleccionada en grande
Flujo alternativo	
Postcondición	

Tabla 10: CU-05 Mostrar imagen carta seleccionada

Identificador	CU-06 Mostrar ambas caras de una carta con doble cara
Descripción	Caso de uso correspondiente al RF-06
Actores	Usuario
Precondición	
Flujo principal	1. El Sistema muestra un botón en la parte inferior de la imagen de una carta que tenga doble cara 2. El Usuario pulsa el botón de reverso de la carta 3. El Sistema, mediante una animación, mostrará la otra cara de la carta
Flujo alternativo	
Postcondición	

Tabla 11: CU-06 Mostrar ambas caras de una carta con doble cara

Identificador	CU-07 Mostrar ediciones de una carta
Descripción	Caso de uso correspondiente al RF-07
Actores	Usuario
Precondición	El Usuario a seleccionado una carta del listado de cartas
Flujo principal	1. El Sistema muestra debajo de la imagen principal de la carta un listado de ediciones de la carta
Flujo alternativo	
Postcondición	

Tabla 12: CU-07 Mostrar ediciones de una carta

Identificador	CU-08 Mostrar arte de una edición de una carta
Descripción	Caso de uso correspondiente al RF-08
Actores	Usuario
Precondición	El Usuario a seleccionado una carta del listado de ediciones de una carta
Flujo principal	1. El Sistema muestra en grande la imagen de la carta de esa edición
Flujo alternativo	
Postcondición	

Tabla 13: CU-08 Mostrar arte de una edición de una carta

Identificador	CU-09 Mostrar valor de una carta de una edición
Descripción	Caso de uso correspondiente al RF-09
Actores	Usuario
Precondición	
Flujo principal	1. El Sistema muestra los precios medio de la carta en cada edición tanto foil como no foil.
Flujo alternativo	
Postcondición	

Tabla 14: CU-09 Mostrar valor de una carta de una edición

Identificador	CU-10 Mostrar legalidades de la carta
Descripción	Caso de uso correspondiente al RF-10
Actores	Usuario
Precondición	
Flujo principal	1. El Usuario selecciona el botón de Legalidades debajo de la imagen de la carta consultada 2. El Sistema mostrará en una tabla un listado de los modos de juego mas jugados en Magic: The Gathering indicando al legalidad de la carta para cada modo de juego.
Flujo alternativo	
Postcondición	

Tabla 15: CU-10 Mostrar legalidades de la carta

Identificador	CU-11 Almacenar historial de cartas consultadas
Descripción	Caso de uso correspondiente al RF-11
Actores	Usuario
Precondición	El Usuario selecciona una carta del listado de cartas buscadas o escaneadas
Flujo principal	1. El Sistema almacena la carta seleccionada en un historial de cartas consultadas
Flujo alternativo	
Postcondición	

Tabla 16: CU-11 Almacenar historial de cartas consultadas

Identificador	CU-12 Mostrar historial de cartas consultadas
Descripción	Caso de uso correspondiente al RF-12
Actores	Usuario
Precondición	
Flujo principal	1. El Usuario selecciona la opción de historial en el TabBar inferior de la App 2. El Sistema muestra un listado con el historial de cartas consultadas
Flujo alternativo	
Postcondición	

Tabla 17: CU-12 Mostrar historial de cartas consultadas

Identificador	CU-13 Seleccionar carta del historial de cartas consultadas
Descripción	Caso de uso correspondiente al RF-13
Actores	Usuario
Precondición	
Flujo principal	1. El Usuario selecciona una de las cartas del historial de cartas consultadas
Flujo alternativo	
Postcondición	El Sistema mostrará los detalles de la carta seleccionada

Tabla 18: CU-13 Seleccionar carta del historial de cartas consultadas

Identificador	CU-14 Añadir carta al listado de cartas favoritas
Descripción	Caso de uso correspondiente al RF-14
Actores	Usuario
Precondición	El Usuario ha buscado o escaneado una carta
Flujo principal	1. El Sistema muestra un listado de cartas como resultado de la búsqueda o escaneo del nombre de una carta 2. El Usuario selecciona una carta del listado de resultados 3. El Sistema muestra el detalle de esa carta seleccionada con un botón en forma de estrella al lado del nombre de la carta, botón de favoritos, sin rellenar 4. El Usuario pulsa el botón de favoritos 5. El Sistema guarda la carta en un listado de favoritas
Flujo alternativo	
Postcondición	Al consultar la carta de nuevo aparecerá el botón de favoritos relleno

Tabla 19: CU-14 Añadir carta al listado de cartas favoritas

Identificador	CU-15 Eliminar carta del listado de cartas favoritas
Descripción	Caso de uso correspondiente al RF-15
Actores	Usuario
Precondición	El Usuario ha buscado o escaneado una carta
Flujo principal	1. El Sistema muestra un listado de cartas como resultado de la búsqueda o escaneo del nombre de una carta 2. El Usuario selecciona una carta del listado de resultados 3. El Sistema muestra el detalle de esa carta seleccionada con un botón en forma de estrella al lado del nombre de la carta, botón de favoritos, relleno porque esa carta es favorita 4. El Usuario pulsa el botón de favoritos 5. El Sistema elimina la carta del listado de favoritas
Flujo alternativo	
Postcondición	Al consultar la carta de nuevo aparecerá el botón de favoritos sin rellenar

Tabla 20: CU-15 Eliminar carta del listado de cartas favoritas

Identificador	CU-16 Mostrar listado de cartas favoritas
Descripción	Caso de uso correspondiente al RF-16
Actores	Usuario
Precondición	
Flujo principal	1. El Usuario selecciona la opción de favoritas en el TabBar inferior de la App 2. El Sistema muestra un listado de cartas favoritas
Flujo alternativo	
Postcondición	

Tabla 21: CU-16 Mostrar listado de cartas favoritas

Identificador	CU-17 Seleccionar carta del historial de cartas favoritas
Descripción	Caso de uso correspondiente al RF-17
Actores	Usuario
Precondición	
Flujo principal	1. El Usuario selecciona una de las cartas del historial de cartas favoritas
Flujo alternativo	
Postcondición	El Sistema mostrará los detalles de la carta seleccionada

Tabla 22: CU-17 Seleccionar carta del historial de cartas favoritas

5. Diseño

5.1. Arquitectura

5.1.1. Arquitectura lógica

En este proyecto se ha elegido una arquitectura MVVM, es decir, Model View ViewModel, actualmente el más usado en el desarrollo de aplicaciones móviles con metodología Agile, debido a su separación clara de la lógica de la interfaz de usuario y la lógica de negocios.

- **Model**: Representa los datos y la lógica de negocios. En este caso, incluye los datos de las cartas de Magic, como nombre, descripción, imagen, etc.
- **View**: La interfaz de usuario (UI) que presenta los datos al usuario y permite la interacción.
- **ViewModel**: Actúa como un intermediario entre el Model y el View, manejando la lógica de presentación y la gestión de datos para la UI.

5.1.2. Beneficios patrón MVVM

- **Separación de responsabilidades**: Como ya se ha dicho anteriormente, Claramente separa la lógica de la UI, la lógica de negocio y la gestión de datos.
- **Mantenibilidad**: Facilita la actualización y el mantenimiento del código, permitiendo cambios en la UI sin afectar la lógica de negocio.
- **Testeabilidad**: Permite pruebas unitarias efectivas, especialmente para la lógica de presentación en el ViewModel.
- **Escalabilidad**: La arquitectura es modular y facilita la adición de nuevas funcionalidades o la modificación de las existentes.

5.1.3. Componentes principales

1. Interfaz de Usuario (UI)
    - **Pantalla de búsqueda**: Permite al usuario buscar cartas por nombre o características.

- **Pantalla de escaneo**: Utiliza la cámara del dispositivo para escanear el nombre de las cartas físicas y obtener así sus datos.
- **Vista de resultado**: Ubicada en al pantalla de búsqueda, muestra un listado de los resultados obtenido de la búsqueda manual o escaneo.
- **Pantalla de detalles de al carta**: Muestra detalles completos de una carta seleccionada.

2. Servicios y APIs

   - **Servicio de búsqueda**: Conecta con la API de Scryfall para obtener resultados de cartas que cumplan con el parámetro buscado.
   - **Servicio de escaneo**: Utiliza técnicas de reconocimiento de texto para identificar el nombre de una carta.
   - **UserDefaults**: Es una clase de la API de iOS utilizada para almacenar configuraciones simples y datos de usuario persistentes. Es parte del framework Foundation y proporciona una manera conveniente de almacenar pares clave-valor. Estos datos se guardan en un archivo en el dispositivo del usuario y se mantienen entre las ejecuciones de la aplicación. Se ha optado por esta solución ya que los datos almacenados son muy simples y no sería del todo práctico utilizar una base de datos completa.

3. Gestión de datos

   - **Modelo de datos**: Estructura de datos que define las propiedades de una carta de Magic.
   - **Repositorios de datos**: Administra la obtención, almacenamiento y actualización de datos en los UserDefaults.

5.2. Modelo de diseño

5.2.1. Diagrama de clases

Como se puede ver en el diagrama, las principales clases del proyecto son:

- **Card**: Clase principal de la Aplicación. Compuesta por los datos principales de una carta.
- **Legalities**: Clase que contiene las legalidades de la carta.
- **Prices**: Clase que contiene los diferentes precios de una carta.
- **Images**: Clase que contiene las URLs de las imágenes de una carta.
- **CardPrints**: Clase que contiene las diferentes imágenes de una carta.
- **CardPrintsInfo**: Clase que contiene la información de una carta que tiene varias caras.
- **RelatedUris**: Clase que contiene la dirección URL para buscar la información de esa carta en la web www.edhrec.com.

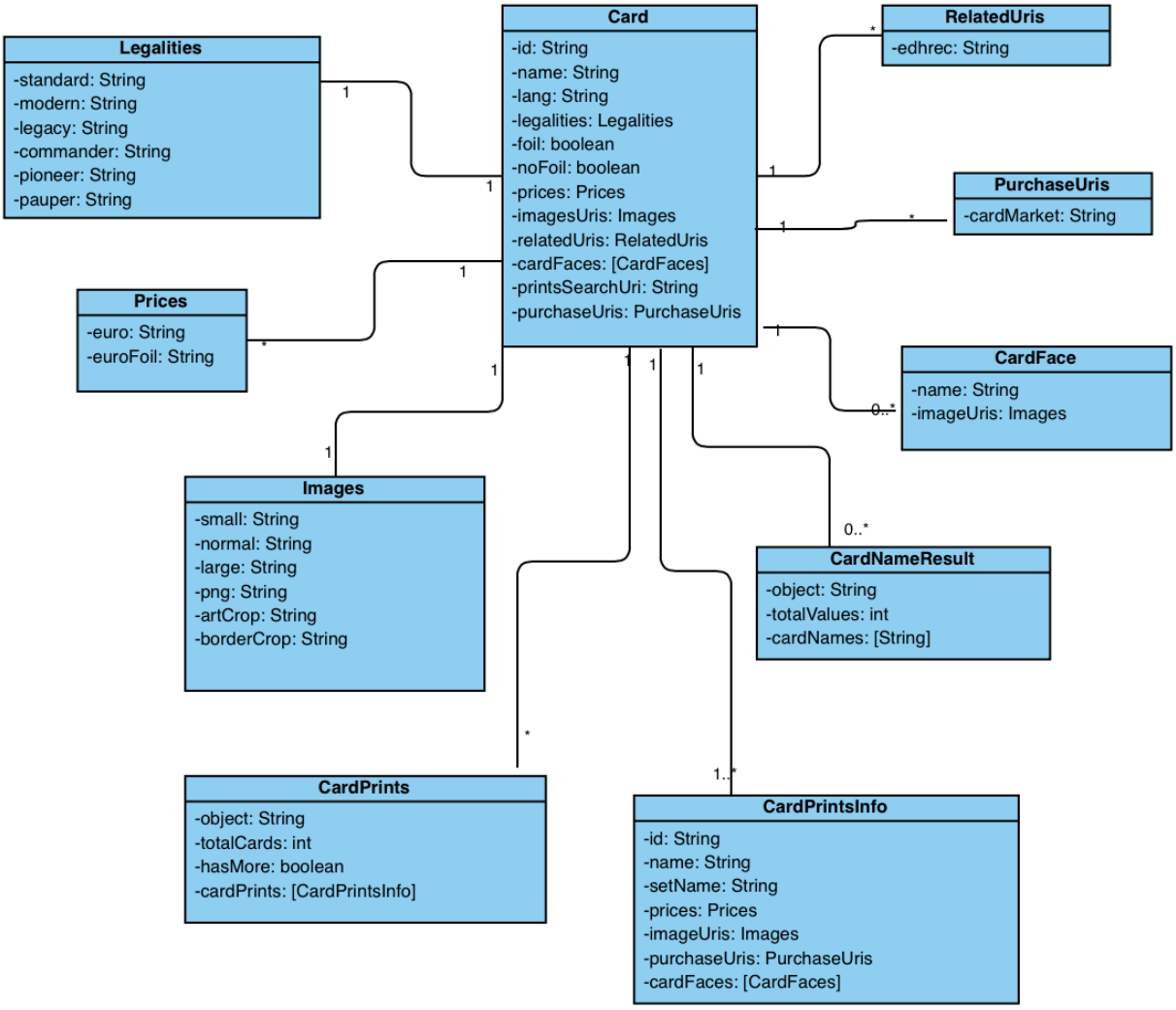


Figura 2: Diagrama casos de uso

- **PurchaseUris**: Clase que contiene la dirección URL de esa carta en la web ww.cardmarket.com, el mayor mercado online de compra venta de cartas Magic.
- **CardFace**: Clase que contiene la información de las caras de una carta con doble cara.
- **CardNameResult**: Clase resultante de la búsqueda de posibles cartas intriduciendo un text en el buscado, conteniendo un array con todos los posibles nombres de las cartas que tiene relación con ese texto buscado

5.2.2. Diagrama de secuencia

El diagrama de secuencia es un tipo de diagrama usado para modelar interacción entre objetos en un sistema según UML. Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso.
A menudo es útil para complementar a un diagrama de clases, pues el diagrama de secuencia se podría describir de manera informal como .[el] diagrama de clases en movimiento", por lo que ambos deben estar relacionados entre sí.
En este documento vamos a mostrar los diagramas de secuencia referentes a los casos de uso más importantes.

- Buscar carta por nombre

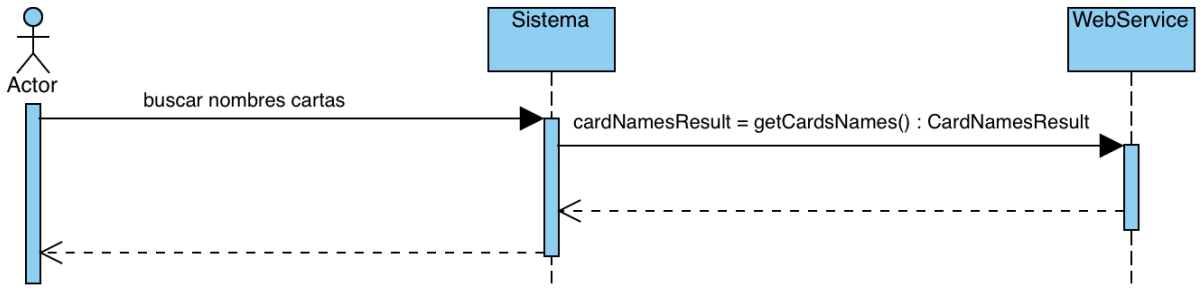


Figura 3: Diagrama de secuencia CU-01 Buscar carta por nombre

- Escanear nombre de carta

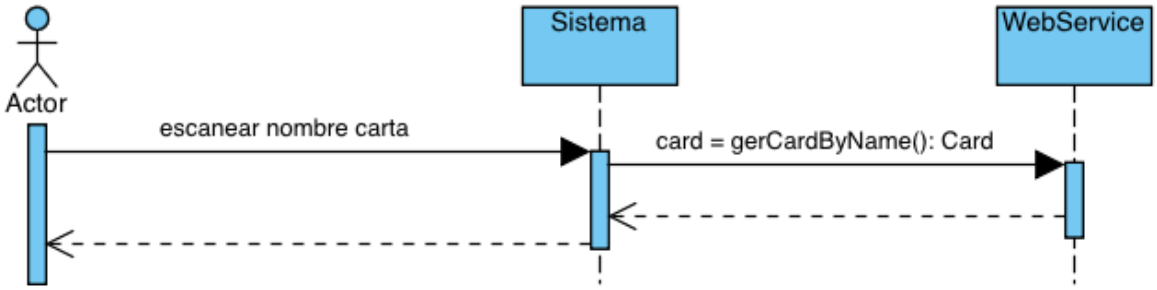


Figura 4: Diagrama de secuencia CU-02 Buscar carta escaneando su nombre

- Seleccionar carta buscada

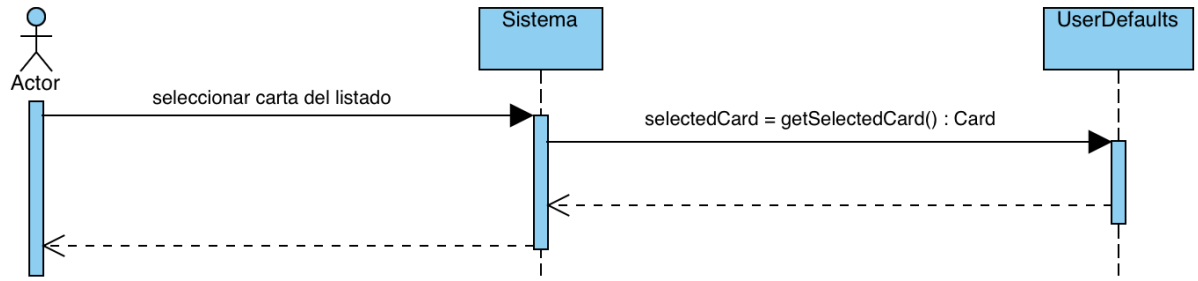


Figura 5: Diagrama de secuencia CU-04 Seleccionar carta buscada o escaneada

- Marcar carta como favorita

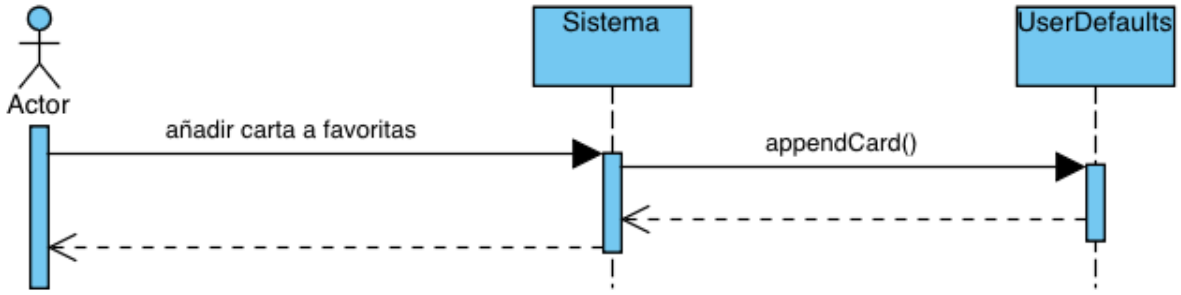


Figura 6: Diagrama de secuencia CU-14 Añadir carta al listado de cartas favoritas

5.3. Diseño de interfaces

En este apartado se van a mostrar todas las interfaces de la aplicación, explicando cada una de ellas.

5.3.1. Pantalla de búsqueda

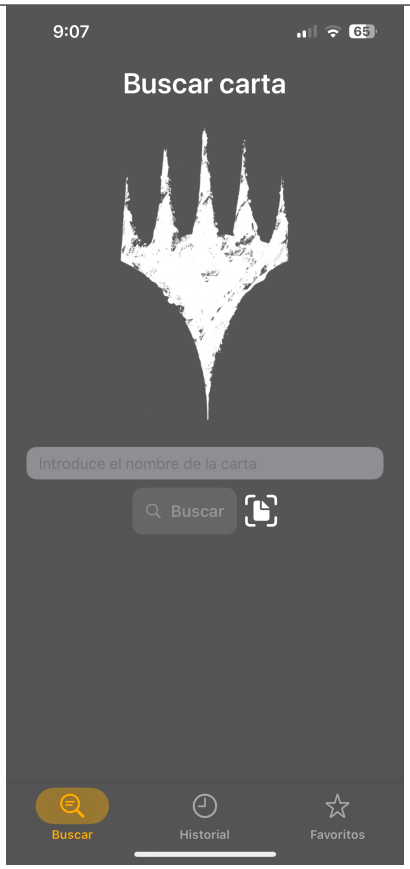
Descripción	Pantalla inicial de la aplicación. En ella el usuario puede buscar manualmente, mediante un campo de texto, una carta por nombre. En esta pantalla también aparece el botón para escanear una carta y cambiar el modo de visualización de los resultados.
Eventos	- Cuando el Usuario introduzca al menos tres caracteres se activará el botón de buscar y le permitirá realizar la búsqueda. - Si el usuario selecciona el botón de escanear, se abrirá la pantalla de escaneo. - Si el usuario busca una carta aparecerá en la parte inferior un listado con los resultados que coincidan con el texto del nombre de carta buscado.
Capturas	

Tabla 23: Pantalla de búsqueda

5.3.2. Pantalla de escaneo

Descripción	Pantalla compuesta por una vista con fondo difuminado dejando un recuadro en el centro para situar en él el nombre de la carta a escanear. El escaneo se produce de forma automática.
Eventos	Una vez la aplicación ha escaneado la carta de forma automática, esa pantalla desaparecerá y mostrará la pantalla principal con el resultado del escaneo.
Capturas	

Tabla 24: Pantalla de escaneo

5.3.3. Vista de resultados

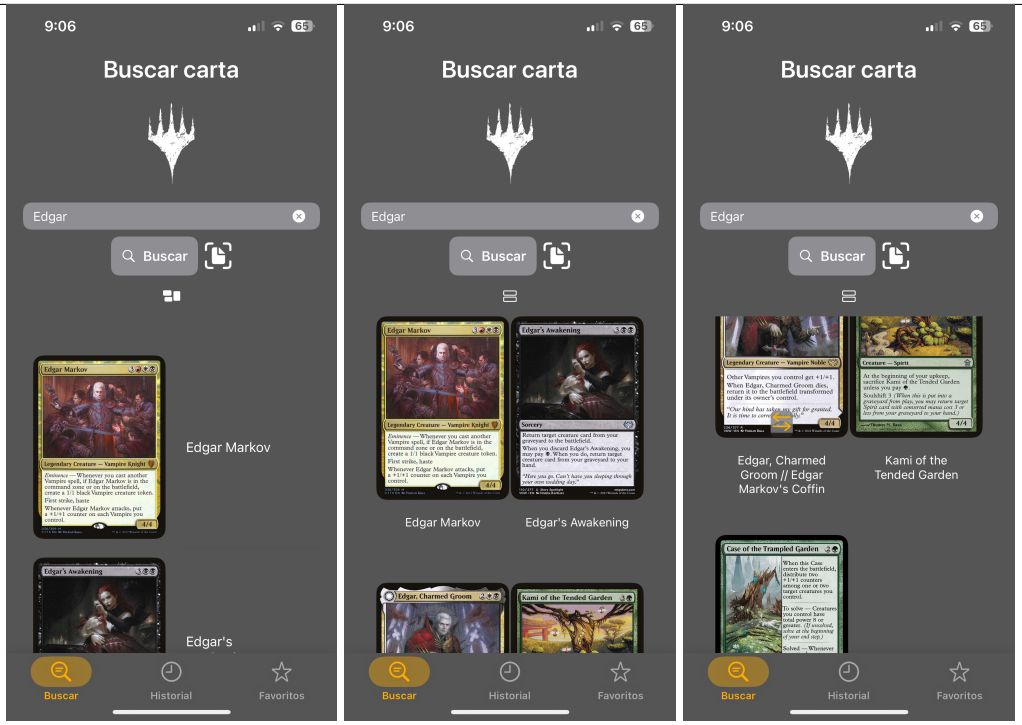
Descripción	Vista que aparece en la parte inferior de la pantalla de búsqueda, compuesta por un botón para cambiar la disposición de los resultados(modo listado o modo grid), y un listado con los resultados de las cartas, imagen y nombre, buscadas.
Eventos	- Cuando el usuario seleccione una carta, navegará hacia al pantalla de detalle de carta. - Si una carta tiene doble cara, en el propio listado de resultados podrá ver ambas caras de la carta, mediante el botón que indica que tiene varias caras.
Capturas	

Tabla 25: Vista de resultados

5.3.4. Pantalla de detalle de una carta

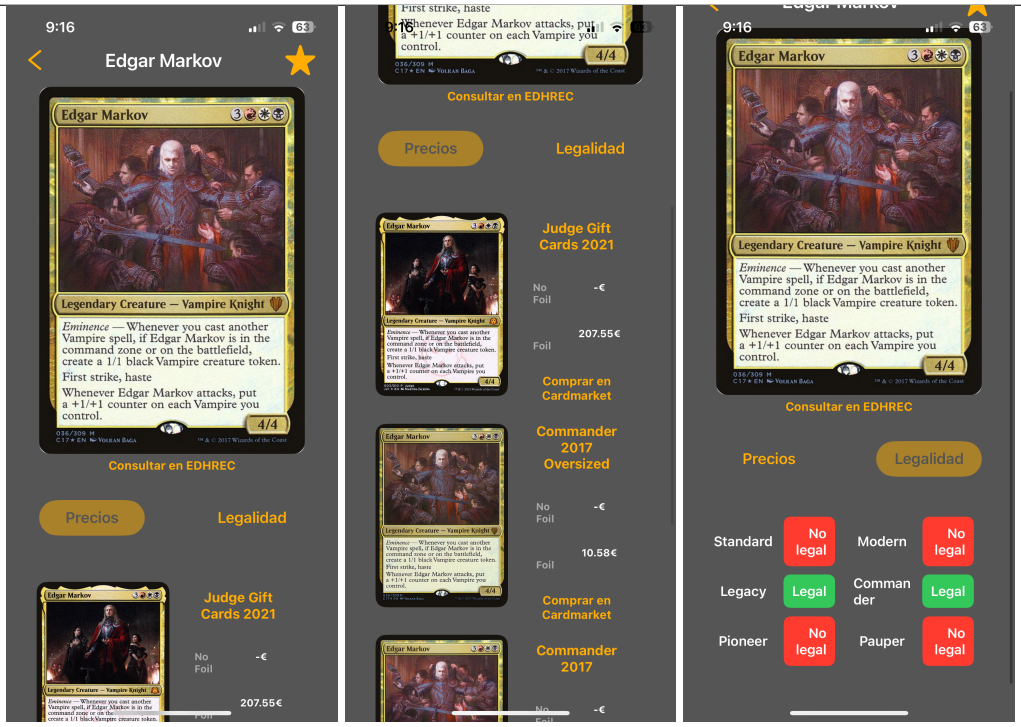
Descripción	- Pantalla principal de la aplicación donde aparecerán los detalles de la carta. - Si es carta con doble cara permitirá ver ambas caras de la carta mediante un botón. - Al lado del nombre de la carta aparecerá el botón que permite añadir la carta a favoritos. - En la parte inferior se podrán ver las diferentes ediciones con sus correspondientes precios si está seleccionada la opción de precios. - En la parte, si está seleccionada la opción de legalidades se podrán consultar las legalidades de la carta en los diferentes formatos. - Se podrán ver dos hipervínculos que permiten abrir la página web del mayor mercado europeo de compra venta de cartas, y la página más consultada mundialmente para ver los detalles, mazos y sinergias que se pueden hacer con esa carta.
Eventos	Una vez se accede a esta pantalla, la carta consultada pasa a formar parte del historial de cartas consultadas, así como si el usuario la marca como favorita, pasará a formar parte de las cartas favoritas.
Capturas	

Tabla 26: Pantalla de detalle de una carta

5.3.5. Pantalla historial de cartas

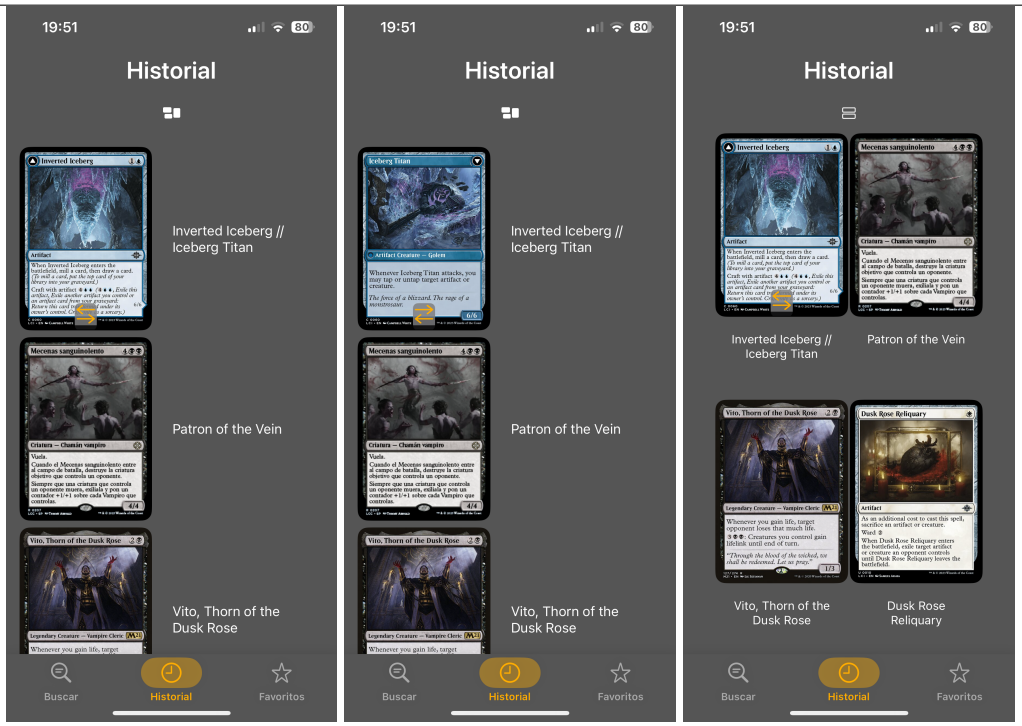
Descripción	Pantalla en la que se mostrará un listado con el historial de cartas consultadas.
Eventos	- El usuario puede selecciona el modo de visualización del historial de cartas, o como listado o como un grid. - El usuario puede seleccionar una carta del listado, abriéndose el detalle de la misma.
Capturas	

Tabla 27: Pantalla historial de cartas

5.3.6. Pantalla cartas favoritas

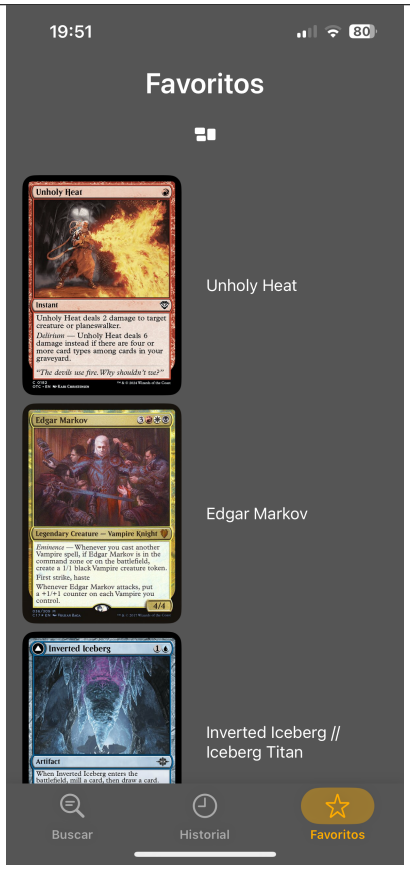
Descripción	Pantalla en la que se mostrará un listado con todas las cartas favoritas marcadas por el usuario.
Eventos	- El usuario puede selecciona el modo de visualización de las cartas favoritas, o como listado o como un grid. - El usuario puede seleccionar una carta del listado de cartas favoritas, abriéndose el detalle de la misma.
Capturas	

Tabla 28: Pantalla cartas favoritas

6. Implementación

6.1. Requerimientos Hardware y Software

6.1.1. Hardware

- Dispositivo iPhone SE o iPhone 8 y superiores.
- Conexión a internet estable, Wi-Fi o datos.
- Memoria dispositivo al menos 50Mb.

6.1.2. Software

- Sistema operativo iOS 17.0 o superior.

6.2. Herramientas empleadas

6.2.1. Herramientas para el desarrollo de la aplicación

El desarrollo de la aplicación se ha realizado en XCode, el IDE nativo de Apple para el desarrollo de aplicaciones nativas para iOS, que soporta tanto Objective-C como Swift en cuanto a lenguajes de programación.

El lenguaje de desarrollo utilizado es Swift complementado con el framework SwiftUI, que es la última tecnología lanzada por Apple para el desarrollo de aplicaciones nativas para iOS.

6.2.2. Herramientas de soporte

Se han utilizado las siguientes herramientas de soporte para la realización del proyecto:

- **Overleaf**: Es un editor laTeX colaborativo basado en la nube que se utiliza para escribir, editar y publicar documentos. Permitiendo una compilación en tiempo real y una exportación automática en diversos formatos. Se ha utilizado para escribir la memoria del proyecto.
- **GitHub**: Plataforma que ofrece almacenamiento de repositorios en la nube. Se ha utilizado para ir almacenando el desarrollo del proyecto a medida que iba avanzando.

6.3. Organización interna del proyecto

En este apartado vamos a describir la organización interna del proyecto, es decir, como están distribuidos los archivos dentro del proyecto. La estructura base del proyecto se divide en las siguientes carpetas:

- **MTGCardScan** - Carpeta del proyecto general.

- **CommonViews** - Carpeta que contienes vistas comunes como pueden ser celdas, TabBar, la vista de carga, etc ...
- **DependencyInjection** - Carpeta donde se encuentra las dependencias y el inyector de dependencias.
- **DDBBRepositories** - Carpeta donde se encuentran las bases de datos locales relativas a cartas, historial y favoritas.
- **Extensions** - Carpeta donde se crean todas las extensiones que pueden ser de utilidad.
- **Model** - Carpeta que alberga todo el modelo de datos de la App separado por clases.
- **Networking** - Carpeta donde se encuentra la gestión de llamadas a la API y los repositorios web.
- **Screens** - Carpeta donde se encuentran todas las clases de la App referentes a cada pantalla de la misma. Esta carpeta esta compuesta de una subcarpeta por cada pantalla de la aplicación que a su vez contiene los siguientes ficheros:
  - **View** - Vista de la pantalla a la que corresponde.
  - **VM** - El encargado de comunicar a la View con el Interactor.
  - **Interactor** - El encargado de comunicarse con la API y los Servicios Web.
- **Utils** - Carpeta que engloba utilidades que nos permiten evitar redundancia de código, tener un código más limpio etc...
- **WebRepositories** - Carpeta que engloba los repositorios web que se utilizan en la App.

6.4. Complicaciones en la implementación

La mayor complicación en cuanto a la implementación del código, ha sido la funcionalidad de las cartas con doble cara, provocado por diferentes motivos

- Complicación del modelo de datos: En la API, la obtención de los datos de las cartas seguía, para todas las cartas el mismo modelo base del objeto *Card*, salvo para los casos de cartas dobles, que el objeto cambiaba en el campo con el que se obtienen las imágenes de la carta, y además, bajo mi forma de ver, un cambio sin mucho sentido.

En el objeto *Card* hay un campo *cardFaces*, que es un Array de un objeto *CardFaces*, el cual si está relleno o aparece es que la carta tiene más de una cara, obteniendo las imágenes de las caras dentro del objeto *cardFaces* mediante su campo *imagesUris*, en vez de poder venir esos datos de las imágenes a mostrar en el campo *imageUris* del objeto Card global. Este escenario, bajo mi punto de vista no tiene mucho sentido, ya que perfectamente se podría haber implementado dicha información en el campo de primer nivel *imagesUris* del objeto Card, como para las cartas de una sola cara.

- Gestión animación de la doble cara: Otro punto complicado ha sido hacer una gestión sencilla, sin gran carga de recursos y a la vez visualmente agradable y llamativa para la gestión de la visualización de ambas caras de la carta.

Optando por una solución con una animación tipo flip de la carta, que simula el giro que se da a una carta en el mundo real para ver su reverso.

- Gestión visualización listado de resultados: La gestión de la visualización del listado de resultados también ha sido compleja, ya que el cuadrar tanto ver las celdas como un listado lineal, o verlas en forma de grid, y que encajaran todos los elementos de forma orgánica, ha supuesto algún quebradero de cabeza, ya que la gestión del espacio, al cargarse imágenes, y alguna de ellas con las posibilidad de girarse haciendo una animación mostrando otra carta, al principio no mantenían bien todas las proporciones, tanto de la propia carta como del grid, siendo la solución, y como pasa muchas veces en programación, pensar en lo más básico y estableciendo ciertos límites de tamaño (para que todas las celdas tengn el mismo tamaño) dejando que se reajustaran los componentes del grid completamente solos.

7. Pruebas

7.1. Pruebas de caja blanca

Las pruebas de caja blanca, también conocidas como pruebas estructurales, son un enfoque de pruebas de software que se centra en la verificación de la estructura interna del código. A diferencia de las pruebas de caja negra, que evalúan la funcionalidad de una aplicación sin conocer su implementación interna, las pruebas de caja blanca requieren conocimiento detallado del código fuente. Este tipo de pruebas se utiliza para garantizar que todos los caminos posibles a través del código sean probados y que todos los errores internos se identifiquen y corrijan.
Las pruebas de caja blanca son fundamentales para identificar errores lógicos y asegurar que el código funcione como se espera en todos los posibles escenarios. En el desarrollo de una aplicación de consulta de cartas, como es el caso de este proyecto, este enfoque ayuda a garantizar que las funcionalidades de búsqueda y filtrado se comporten correctamente, proporcionando una experiencia de usuario satisfactoria.

Las comprobaciones y pruebas realizas durante el proceso de desarrollo han sido:

- Comprobación de las peticiones a los servicios web en tiempo de ejecución.
- Comprobación de funcionalidades mostradas al usuario.
- Comprobación de carga de imágenes.
- Comprobación de adaptación a los diferentes tipos de tamaño de pantalla.
- Comprobación del parseo de datos recibidos.

7.2. Pruebas de caja negra

Las pruebas de caja negra, también conocidas como pruebas funcionales, son un método de prueba de software en el cual el tester examina la funcionalidad del sistema sin conocer su estructura interna, diseño o implementación. El objetivo principal es verificar que el software cumple con los requisitos especificados y funciona correctamente en todos los escenarios posibles.

En las pruebas de caja negra, los testers se enfocan en las entradas y salidas del sistema, diseñando casos de prueba basados en las especificaciones del software y los requerimientos del usuario. No necesitan saber cómo se implementan internamente las funcionalidades, sino sólo que el sistema produce las salidas correctas cuando se le dan las entradas correctas.

Las pruebas de caja negra más importantes realizadas han sido:

- **Búsqueda por nombre exacto**
    - **Entrada**: Nombre de carta *Black Lotus*.

- **Salida**: El listado de resultados muestra la carta *Black Lotus*.

- **Búsqueda por parte del nombre**
  - **Entrada**: Nombre de carta *Lotus*.
  - **Salida**: La lista de resultados muestra todas las cartas que contienen *Lotus* en el nombre, como *Black Lotus* y *Lotus Petal*.

- **Escaneo de carta válida**
  - **Entrada**: Escanear una carta física *Brutal Cathar*.
  - **Salida**: El listado de resultados muestra la carta *Brutal Cathar*.

- **Selección carta del listado de resultados**
  - **Entrada**: Seleccionar carta *Edgar Markov* del listado de resultados.
  - **Salida**: Se muestra la pantalla de detalle de la carta *Edgar Markov*.

- **Carta consultada se añade al historial de cartas**
  - **Entrada**: Seleccionar carta *Edgar Markov* del listado de resultados mostrando los detalles de la carta.
  - **Salida**: Seleccionar la opción del TabBar de Historial de cartas, aparece la carta *Edgar Markov*.

- **Carta marcada como favorita aparece en el listado de cartas favoritas**
  - **Entrada**: Seleccionar el botón de favorito en la pantalla de detalle de la carta *Edgar Markov*.
  - **Salida**: Seleccionar la opción del TabBar de cartas favoritas, aparece la carta *Edgar Markov* en el listado de cartas favoritas.

- **Ejemplo de prueba fallida - Escaneo de carta válida**
  - **Entrada**: Escanear una carta física *Brutal Cathar*.
  - **Salida**: El sistema no reconoce bien el nombre de la carta, debido al brillo o al enfoque, no muestra la carta *Brutal Cathar* en el listado de resultados.

- **Ejemplo de prueba fallida - Carta consultada NO se añade al historial**
  - **Entrada**: Seleccionar carta *Edgar Markov* del listado de resultados mostrando los detalles de la carta.
  - **Salida**: Seleccionar la opción del TabBar de Historial de cartas, no aparece la carta *Edgar Markov*, debido a un problema de persistencia de datos.

- **Ejemplo de prueba fallida - Carta marcada como favorita NO se añade al listado de cartas favoritas**

    - **Entrada**: Seleccionar el botón de favoritos en el detalle de la carta consultada, textitEdgar Markov.
    - **Salida**: Seleccionar la opción del TabBar de cartas favoritas, no aparece la carta *Edgar Markov*, debido a un problema de persistencia de datos.

7.3. Pruebas usuario

Las pruebas de usuario son una fase crucial en el ciclo de vida del desarrollo de software, enfocadas en evaluar la experiencia del usuario final con la aplicación. Estas pruebas se realizan para garantizar que el producto sea intuitivo, funcional y satisfactorio desde la perspectiva del usuario.

La pruebas realizadas han sido:

- **Pruebas de Usabilidad**: Centradas en como el usuario interactuaría con la aplicación, su facilidad de uso, la eficiencia y la satisfacción del usuario.
- **Pruebas de Funcionalidad**: Verifican que la aplicación funcione según lo esperado y cumpla con los requisitos especificados. Se han enfocado en aspectos funcionales como la búsqueda,la visualización de detalles, etc...
- **Pruebas de Experiencia de Usuario (UX)**: Centradas la estética de la interfaz, la accesibilidad, y la coherencia del diseño.

8. Instalación de la Aplicación

Para instalar una aplicación iOS hay varias opciones dependiendo de las necesidades específicas y del estado del desarrollo de la aplicación.
Primero se expondrán todas las opciones, y posteriormente se justificará la opción posible para este TFG.

1. **Uso de XCode**: Es la opción principal y más directa para desarrollar y probar aplicaciones en un dispositivo iOS.

   - **Desarrollo y pruebas**

     a) Conectar dispositivo iOS (iPhone o iPad) al Mac usando un cable USB o via WiFi.

     b) Abrir proyecto en XCode.

     c) Seleccionar dispositivo en el menú de destino en la barra de herramientas de XCode.

     d) Hacer click en el botón de Run"(o usar el atajo de teclado Cmd + r) para compilar y ejecutar la aplicación en tu dispositivo.

   - **Desarrollo y pruebas**

     a) Tener una cuenta de desarrollador de Apple (puede ser una cuenta gratuita, pero tiene limitaciones).

     b) Asegurarse de que el dispositivo está registrado en XCode. Si es la primera vez que lo conectas, puede que tengas que confiar en el dispositivo y registrarlo como dispositivo de desarrollo.

2. **TestFlight**: TestFlight es una plataforma de Apple para la distribución beta de aplicaciones. Permite a los desarrolladores invitar a usuarios internos y externos para probar la aplicación antes de su lanzamiento oficial. Siendo necesario tener una cuenta de desarrollador con suscripción.

3. **Ad Hoc Distribution**: La distribución Ad Hoc permite instalar una aplicación en un número limitado de dispositivos iOS sin pasar por la App Store. Siendo necesario tener una cuenta de desarrollador con suscripción.

4. **App Store Distribution**: La distribución a través de la App Store es el método más formal y adecuado para aplicaciones que están listas para ser publicadas y usadas por el público en general.

8.1. Conclusiones

Las opciones de XCode y TestFlight son las más prácticas y comunes, ya que permiten una fácil distribución y prueba de la aplicación sin necesidad de pasar por el proceso completo de revisión de la App Store. La opción de Ad Hoc puede ser útil si necesitas instalar la aplicación en varios dispositivos sin utilizar TestFlight. La distribución a través de la App Store es ideal si tu aplicación está lista para un lanzamiento público y deseas que esté disponible para un público más amplio.

8.2. Caso específico para este TFG

En el caso que nos compete, que es la distribución de la App desarrollada para este TFG, sólo serían posibles la opción de XCode, ya que actualmente no dispongo de cuenta de suscripción de pago que cuesta anualmente unos 100 dólares.

9. Conclusiones y trabajo futuro

9.1. Conclusiones

Técnicamente, a pesar de llevar trabajando casi 8 años como desarrollador iOS, nunca había utilizado el reconocimiento de texto nativo en iOS, y la verdad me ah sorprendido muy gratamente el nivel de precisión que tiene.

En cuanto al desarrollo del proyecto me ha venido bien para refrescar ciertas cosas, ya que actualmente me dedico más a la Jefatura de Proyectos que al propio desarrollo desde 0 de una App, por lo que me ha gustado volver a implicarme en la creación de una App desde 0, y sobretodo de una App que seguiré desarrollando cuando tenga algo de tiempo libre, ya que es algo que voy usar tanto yo, como muchos conocidos que jugamos a Magic The Gathering.

Si es cierto que ciertas cosas que he utilizado no eran necesarias para una App estructuralmente sencilla, como puede ser al inyección de dependencias, pero quería hacer una estructura en la App lo más parecida posible a como puede ser la estructura de una App móviles con miles de usuarios activos al día como las que estoy acostumbrado a hacer, para en el caso de seguir desarrollando este proyecto por mi cuenta, que la App fuera fácilmente escalable, y su arquitectura no fuera un problema.

A nivel personal me ha costado mantener la planificación inicial, pero he tenido que cumplirla prácticamente rajatabla ya que sino me habría sido imposible compaginar la gran carga de trabajo y responsabilidad que tengo en mi actual trabajo con el desarrollo de este TFG.

9.2. Trabajo futuro

En este apartado vamos a hablar de las posible mejoras que se podrían implementar en la aplicación, si el desarrollo continuase en un futuro:

- **Búsqueda completa de cartas -** Ampliar las posibilidades de búsqueda:
  - Por tipo de carta (Criatura, Tierra, Instantáneo ...)
  - Por coste de maná
  - Por color (negro, blanco, azul, rojo, verde)
  - Por edición

  De esta forma no reducimos la búsqueda exclusivamente al nombre de la carta, por mucho que esta sea la forma de búsqueda más usada.
- **Gestión de colecciones -** Poder almacenar toda la colección de cartas de un usuario permitiendo su clasificación por ediciones, categorías, o según la preferencia del usuario.

- **Gestión de mazos -** Permitir al usuario crear mazos, dependiendo del tipo de modalidad de juego y reflejando las estadísticas del mazo que puedan ser de utilidad para el usuario en su construcción.
- **Simulador de partidas -** Permitir al usuario, una vez creado un mazo, tener una especia de simulador de partidas en solitario para que el usuario pueda ver el desempeño del mazo en una partida simulada.
- **Sección de noticias -** Incluir uan sección de noticias relacionadas con el munde de Magic The Gathering, pudiendo ser noticias relativas a nuevos lanzamientos, como a datos de torneos importantes, etc...
- **Versión Android -** Realizar la misma aplicación pero para dispositivos Android.

10. Referencias

En este apartado se van a enumerar las referencias consultadas para la realización de este proyecto.

10.1. Bibliografía

En el desarrollo de este proyecto he consultado varios libros, que son los que suelo consultar en mi día a día profesionalmente.
Siendo los más usados los dos primero del listado, ambos pertenecientes a Paul Hudson, actualmente una de las mayores fuentes de conomcimiento sobre desarrollo iOS.

- Paul Hudson - Advanced iOS - Libro de consulta digital
- Paul Hudson - Beyond Code - Libro de consulta digital
- Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Ed. PRENTICE-HALL INTERNATIONAL EDITION, ISBN 9780132350884.
- Arlow, Jim, Neustadt, Ila - UML 2

10.2. Webgrafía

Aparte de la bibliografía ya comentada, se han consultado varias páginas web, mostrando a continuación las más relevantes

10.2.1. Webs principales

- Hacking with Swift, la web del anteriormente mencionado Paul Hudson, que se convertido en la web de referencia de consulta en el desarrollo para aplicaciones del ecosistema Apple, libros sobre desarrollo para aplicaciones iOS, iPadOs y macOS - `https://www.hackingwithswift.com`
- ScryFall, API que contiene toda la información de las cartas para hacer las consultas - `https://scryfall.com/docs/api`
- Documentación oficial de SwiftUI - `https://developer.apple.com/documentation/swiftui/`
- Documentación oficial XCode - `https://developer.apple.com/xcode/`
- StackOverflow - `https://stackoverflow.com/questions/tagged/swiftui`

10.2.2. Otras webs

- Wikipedia, MVVM - https://en.wikipedia.org/wiki/Modelviewviewmodel
- Wikipedia, pruebas software - https://es.wikipedia.org/wiki/Pruebas_de_software
- Wikipedia, diagramas de clases - https://es.wikipedia.org/wiki/Diagrama_de_clases
- Wikipedia, diagramas de secuencia - https://es.wikipedia.org/wiki/Diagrama_de_secuencia
- Wikipedia, pruebas de caja blanca -https://es.wikipedia.org/wiki/Pruebas_de_caja_blanca
- Wikipedia, pruebas funcionales -https://es.wikipedia.org/wiki/Pruebas_funcionales

Anexo

A. Repositorio para el código

El código de este proyecto se encuentra en el repositorio de GitHub - `https://github.com/carguti/MTGCardScan`