UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA DEL SOFTWARE

# PTCG COLLECTOR: UNA APLICACIÓN MÓVIL PARA COLECCIONISTAS DE CARTAS POKÉMON

# PTCG COLLECTOR: A MOBILE APPLICATION FOR POKEMON CARD COLLECTORS


Realizado por

ISMAEL TORÉ FRANCO

Tutorizado por

GABRIEL JESÚS LUQUE POLO

JAMAL TOUTOUH EL ALAMIN

Departamento

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA



MÁLAGA, septiembre de 2023

UNIVERSIDAD
DE MÁLAGA

E.T.S. INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INFORMÁTICA GRADO EN INGENIERÍA DEL SOFTWARE

# PTCG COLLECTOR: UNA APLICACIÓN MÓVIL PARA COLECCIONISTAS DE CARTAS POKÉMON

# PTCG COLLECTOR: A MOBILE APPLICATION FOR POKEMON CARD COLLECTORS


Realizado por
**Ismael Toré Franco**

Tutorizado por
**Gabriel Jesús Luque Polo**
**Jamal Toutouh El Alamin**

Departamento
**Lenguajes y Ciencias de la Computación**

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2023



Fecha defensa: septiembre de 2023

# Resumen

Desde su creación Pokémon ha sido un producto que ha estado en boca de todos, ya sea por sus juegos, sus series animadas, o su *merchandising*. Debido a la alta popularidad de la marca decidieron sacar colecciones de cartas para que fuese una nueva forma de jugar, pero con el paso del tiempo, la funcionalidad de estas cartas evolucionó al coleccionismo. Actualmente debido al gran volumen de cartas existentes los coleccionistas más veteranos necesitan tener una forma de saber de una manera fácil y rápida que cartas tienen en su colección, y datos sobre la misma, y de esta necesidad surge la idea de desarrollar esta aplicación.

En el desarrollo de este proyecto se siguió las etapas de desarrollo software como son: Planificación, Análisis, Modelado, Implementación y Pruebas, haciendo que obtengamos el producto final. Este producto consiste en una aplicación móvil que actúa como base de datos para los usuarios, podrán ver todas las colecciones y cartas existentes, y podrán añadir las que ellos posean a su colección. Además, podrán añadir cartas a la *wishlist* y la aplicación les avisará cuando esté al precio deseado a través del correo electrónico. También, tendrán el apartado de estadísticas, con datos como por ejemplo el valor de su colección, valor de la carta más valiosa, etc. Como novedad se ha incorporado un sistema de logros y ranking para generar competencia entre los usuarios. Por último, tenemos el tasador de intercambios, el usuario introducirá las cartas que va a dar y recibir en el intercambio y la aplicación le dirá si es un intercambio justo. Y si aún no sabes que cartas dar, o que cartas vas a recibir, a través de algoritmo, la aplicación te sugerirá qué cartas puede dar o qué cartas te podrían dar para que sea un intercambio justo.

**Palabras clave:** Pokémon, coleccionismo, intercambios, Flask, Android Studio, Java

# Abstract

Since its creation, Pokémon has been a product that has been on everyone's lips, either for its games, its animated series, or its merchandising. Due to the high popularity of the brand, they decided to release card collections as a new way to play, but as time went by, the functionality of these cards evolved to collecting. Currently, due to the large volume of existing cards, veteran collectors need to have a way to know in an easy and quick way what cards they have in their collection, and data about it, and from this need arises the idea of developing this application.

In the development of this project, we followed the stages of software development such as: Planning, Analysis, Modeling, Implementation and Testing, so that we obtain the final product. This product consists of a mobile application that acts as a database for users, they will be able to see all existing collections and cards, and they will be able to add the ones they have to their collection. They will also be able to add cards to the wishlist and the application will notify them when the desired price is available via email. They will also have a statistics section, with data such as the value of their collection, the value of the most valuable card, etc. As a novelty, an achievement and ranking system has been incorporated to generate competition among users. Finally, we have the exchange appraiser, the user will enter the cards to give and receive in the exchange and the application will tell you if it is a fair exchange. And if you still do not know what cards to give, or what cards you are going to receive, through an algorithm, the application will suggest to you what cards to give or what cards you could be given to make it a fair exchange.

**Keywords:** Pokémon, collecting, trading, Flask, Android Studio, Java

# Índice

# 1

# Introducción

En este primer capítulo de la memoria se desarrollará una breve descripción de los motivos que llevaron a realizar este proyecto, los objetivos que se querían conseguir, la metodología empleada y la organización del documento.

## 1.1 Motivación

El mundo del coleccionismo siempre ha formado parte de nuestras vidas, ya sean objetos antiguos, coches, camisetas de fútbol, entre una gran infinidad de objetos. En este trabajo fin de grado (TFG) se abordará esta afición desde el punto de vista de las cartas Pokémon [1]. Éstas salieron al mercado por primera vez en 1996 en Japón, y dado su popularidad en 1999 se lanzaron en Estados Unidos y Europa.

En realidad, la funcionalidad original de estas cartas era para jugar partidas de uno contra uno, pero con el paso del tiempo la gente lo convirtió en una afición de coleccionismo. En la actualidad es un mercado que mueve muchísimo dinero, ya que hay muchas cartas en las que su valor está por encima de los 100.000$, teniendo la carta más cara un valor de 5.275.00$ [2] aproximadamente.

Además, podemos ver que es un mundo en el que la gente está muy interesada, ya que hay canales de YouTube en el que se dedican

exclusivamente a la apertura de cartas Pokémon y tienen más de un millón de suscriptores. Esto tampoco es algo que sea ajeno en España, donde tenemos ejemplos de creadores de contenido con vídeos que superan los dos millones de reproducciones [3]. Todo esto nos sirve para mostrar la gran masa de personas que mueve esta afición.

Además, esto va a más, a raíz de este gran interés se han creado empresas [4] encargadas de dar un valor a las cartas, en función de su estado. Hay incluso casos de personas que viven gracias a las cartas Pokémon [5], lo que es un claro ejemplo de la influencia de estas cartas en la vida de las personas.

Por todo ello, y la grandísima variedad de colecciones existentes hoy en día, los coleccionistas más longevos acaban con una gran cantidad de cartas que dificulta recordar cuáles tienes y cuáles no. Con este propósito de ayudar a los coleccionistas de este tipo de cartas nace esta aplicación. Aunque es cierto que ya existe alguna aplicación que trata la funcionalidad de ser una base de datos, muchos usuarios lo consideran insuficiente y les gustaría que la aplicación tuviese más funcionalidades. Por ello este proyecto será más completo, haciendo que sea una gran oportunidad de mercado para contentar a esos usuarios insatisfechos.

## 1.2 Objetivos

Como se ha mencionado en el apartado anterior, el objetivo del TFG será desarrollar una aplicación móvil para el almacenamiento de cartas Pokémon. En la aplicación se mostrará el listado de las distintas colecciones que hay, y en cada carta se mostrarán distintos datos sobre ella. Además, se mostrará el listado de cartas que cada usuario posee, pudiendo filtrarlas por colección, tipo del Pokémon, precio, artista y alguna característica más.

Una de las grandes novedades respecto a otras aplicaciones, es añadir un factor social implementando un sistema de logros y clasificación, lo cual fomentará la competitividad entre los usuarios para así alcanzar las posiciones más altas de dicha clasificación. También se podrá visualizar los datos estadísticos de tu colección, como cuántas cartas tienes en total, cuántas cartas únicas tienes, valor de la colección, carta más valiosa, entre otros datos.

Otra funcionalidad incluida será una *Wishlist* en la cual podrás incluir esas cartas que tanto deseas añadir a tu colección, y no solo eso, sino que la aplicación te avisará cuando esa carta baje de precio, para que así puedas adquirirla. Por último, se incluirá un tasador de intercambios en el cual podrás introducir tus cartas que vas a ofrecer y las cartas que esperas recibir, para así poder comprobar que es un intercambio justo. Además, la aplicación a desarrollar debe poder sugerir qué cartas puedes recibir u ofrecer para que el intercambio sea lo más equilibrado posible.

La arquitectura que se utilizará se trata de un cliente/servidor, donde la aplicación móvil actuará como *frontend* para interaccionar con los usuarios y se dispondrá de un servidor (*backend*) donde se almacenan los datos y se realicen las manipulaciones y cálculos necesarios. Además, se hará uso de una API (*Application Programming Interface*) externa con la cual podremos obtener todo lo relacionado con las cartas.

## 1.3 Metodología

Para el desarrollo de este proyecto, se eligió Scrum [6] como metodología, aunque se tuvo que adaptar para ser aplicada de forma individual. Se seleccionó esta metodología debido a la corta duración del proyecto, ya que se consideró que mejor se ajustaba a sus necesidades. Scrum se caracteriza por su enfoque incremental e iterativo, así como el uso de sprints de desarrollo, lo que proporciona flexibilidad para gestionar los cambios en el proyecto y garantizar la entrega continua de valor al cliente.

Las fases de trabajo seleccionadas para el desarrollo de este proyecto fueron: Planificación, Especificación y Análisis, Diseño, Implementación y Pruebas. Cada una de estas fases se corresponde a un sprint, a excepción de la fase de implementación que se han dividido en varios sprints, que se detallarán en su correspondiente capitulo. Al finalizar cada sprint de la implementación se tuvieron reuniones con el tutor, el cual actuaba como *Product Owner*. Es importante destacar que las pruebas se realizaron de manera simultánea con la implementación. En este documento, se describirá detalladamente el proceso seguido en cada una de estas fases.

## 1.4 Organización de la memoria

La memoria del Trabajo Fin de Grado está compuesta por los siguientes capítulos:

- **Capítulo 1. Introducción**. En este capítulo se describe mediante una breve introducción los contenidos que forman el documento y donde se explica la motivación, los objetivos y la organización de la memoria.
- **Capítulo 2. Tecnologías y herramientas.** En este capítulo se listarán y explicarán las distintas tecnologías y herramientas que han sido utilizadas en el desarrollo del proyecto.
- **Capítulo 3. Especificación y análisis.** En este capítulo se enumerarán los distintos requisitos; primero los funcionales, luego los no funcionales, y posteriormente se añadirán los casos de usos y diagramas de secuencia de algunos de estos requisitos.
- **Capítulo 4. Diseño del sistema.** En esta sección se describirá la arquitectura en la que se ha basado la aplicación y el diseño de la base de datos.
- **Capítulo 5. Implementación.** En este apartado se describen todos los pasos que se han dado en la implementación del proyecto.
- **Capítulo 6. Pruebas.** En este apartado se describen las pruebas que se han realizado durante el desarrollo del proyecto.
- **Capítulo 7. Conclusiones y líneas futuras.** En este último punto se evalúan los resultados obtenidos al finalizar el proyecto y se sacan

conclusiones sobre el aprendizaje obtenido, las dificultades encontradas en el proceso y qué posibles mejoras se podrían añadir al proyecto en el futuro.

En adición a estos capítulos, la memoria también está formada por un resumen de la propia memoria, un índice de los contenidos, una bibliografía y los distintos apéndices sobre el Manual de usuario y el Manual de Despliegue

# 2

# Tecnologías y Herramientas

Para el desarrollo de este proyecto se ha utilizado los lenguajes Python, junto al *framework* Flask, y Java. Además, se han hecho uso de algunas bibliotecas para poder ampliar la variedad de funcionalidades a implementar como Volley. Además, se ha usado MongoDB Atlas para la base de datos y Heroku para el despliegue. A continuación, se explicará brevemente cada lenguaje y las principales bibliotecas utilizadas.

## 2.1. Tecnologías

A continuación, se explicarán las distintas tecnologías utilizadas en el proyecto.

### 2.1.1 Python

Python [7] es un lenguaje de programación que destaca por ser un lenguaje de alto nivel. Se trata de un lenguaje muy versátil ya que es un lenguaje orientado a objetos, pero también permite la programación imperativa, y hasta en algunas situaciones específicas, la programación funcional. En la actualidad, Python se ha convertido en uno de los lenguajes más

*Figura 1. Logo Python*

populares y demandados en todo el mundo. Su versatilidad lo hace invaluable en una amplia variedad de campos, como puede ser el desarrollo web, la ciencia de datos, la inteligencia artificial, entre otros.

### 2.1.2 Flask

Flask [8] es un framework de Python para el desarrollo web. Se utiliza principalmente para crear aplicaciones web de manera sencilla y rápida. Flask proporciona las herramientas necesarias para construir aplicaciones web efectivas y escalables.

Algunas de las características clave de Flask son su simplicidad, extensibilidad, flexibilidad, su rápido desarrollo y el tener una amplia comunidad. Para el desarrollo de este proyecto se ha utilizado Flask en la parte del servidor para construir el API REST de la aplicación.


*Figura 2. Logo Flask*

### 2.1.3 Java

Java [9] al igual que Python es uno de los lenguajes de programación más extendidos y reconocidos en la actualidad. Java se caracteriza por ser un lenguaje orientado a objetos altamente versátil y compatible con múltiples plataformas. Dada su versatilidad uno de los principales usos de Java es en el desarrollo de aplicaciones para el ecosistema Android, justo por esto, se decidió su uso para este proyecto.


*Figura 3. Logo Java*

### 2.1.4 MongoDB Atlas

MongoDB [10] es una base de datos NoSQL de código abierto ampliamente utilizada que se destaca por su capacidad para manejar datos no estructurados y semiestructurados en forma de documentos JSON. En MongoDB, los datos se almacenan en colecciones, y cada documento puede tener una estructura diferente, lo que proporciona flexibilidad en el almacenamiento de datos. Su capacidad de escalabilidad


*Figura 4. Logo MongoDB Atlas*

horizontal y su rendimiento de lectura/escritura lo hacen especialmente adecuado para aplicaciones web y móviles de rápido crecimiento.

MongoDB Atlas [11], por otro lado, es el servicio de base de datos en la nube gestionado de MongoDB. Proporciona una forma sencilla de implementar y administrar clústeres de bases de datos MongoDB en la nube, sin la necesidad de lidiar con la infraestructura subyacente. Atlas ofrece características como copias de seguridad automáticas, escalabilidad vertical y horizontal, seguridad avanzada y monitoreo en tiempo real.

### 2.1.5 Volley

Volley [12] es una biblioteca HTTP que facilita y agiliza el uso de redes en aplicaciones para Android. Se integra fácilmente con cualquier protocolo y, además, incluye compatibilidad con textos sin procesar, imágenes y JSON. Esto último ha sido el formato utilizado para las peticiones de este proyecto. Además de todo esto cuenta con una amplia documentación, lo que hacía más atractivo su uso para este proyecto.

### 2.1.6 Picasso

Picasso [13] es una biblioteca de Android ampliamente utilizada para la carga y visualización eficiente de imágenes en aplicaciones móviles. Esta biblioteca simplifica la gestión de imágenes al proporcionar una interfaz fácil de usar para descargar, almacenar en caché y mostrar imágenes en vistas de manera rápida y automática. Picasso también maneja la cancelación automática de solicitudes de imágenes en caso de que una vista ya no sea visible, lo que ayuda a optimizar el rendimiento de las aplicaciones y a reducir el consumo de ancho de banda. Se escogió utilizar esta biblioteca para el proyecto por su simplicidad y funcionalidad.

### 2.1.7 Heroku

Heroku [14] es una plataforma en la nube como servicio (PaaS) que permite a los desarrolladores implementar, administrar y escalar aplicaciones web y móviles de manera sencilla. Se destaca por su facilidad de uso y su enfoque en simplificar el proceso de implementación y gestión de aplicaciones, lo que permite a los equipos de desarrollo centrarse en el código y la lógica de la aplicación en lugar de preocuparse por la infraestructura subyacente.


*Figura 5. Logo Heroku*

Existen distintas formas de realizar el despliegue, como podría ser usando el CLI de Heroku, o como se ha hecho en este proyecto, conectando Heroku a un repositorio de GitHub con el código a desplegar.

### 2.1.8 Firebase

Firebase [15] es una plataforma de desarrollo de aplicaciones móviles y web ofrecida por Google. Proporciona una amplia gama de servicios y herramientas para simplificar la creación, el desarrollo y la administración de aplicaciones, incluyendo autenticación de usuarios, bases de datos en tiempo real, almacenamiento en la nube, análisis de usuarios y más. Firebase se destaca por su facilidad de uso, escalabilidad y la capacidad de acelerar el desarrollo de aplicaciones al proporcionar una infraestructura sólida y diversas funcionalidades listas para usar, lo que permite a los desarrolladores centrarse en crear experiencias de usuario excepcionales en lugar de preocuparse por la gestión de servidores y la infraestructura subyacente.


Figura 6. Logo Firebase

En este proyecto vamos a usar firebase para la autenticación de usuarios. A través del cliente se envía la petición al servidor, el cual está conectado con firebase y se crea el usuario, generando un identificador propio del usuario que será lo que guardemos en nuestra base de datos junto con el correo que también es guardado en firebase.

## 2.2 Herramientas

Una vez descritas las tecnologías que se han utilizado en este proyecto, pasamos a describir las distintas herramientas que han servido de apoyo para el desarrollo del proyecto. Se hace hincapié en la importancia de la API externa utilizada, ya que sin ella no habría sido posible realizar este proyecto.

### 2.2.1 Visual Studio Code

Visual Studio Code [16] es un editor de código fuente desarrollado por Microsoft y lanzado el 18 de noviembre de 2015. Esta herramienta puede ser utilizada desde Windows, Linux, macOS y Web. Entre sus funciones, incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.


*Figura 7. Logo Visual Studio Code*

Una de las características que ha impulsado la popularidad de esta herramienta es su alta compatibilidad con lenguajes de programación como pueden ser Python, C#, C/C++, Java, HTML, CSS, Ruby, PHP y muchos más. Entre esta variedad de lenguajes y que es un entorno simple de utilizar, ha hecho que se convierta en uno de los editores más populares y utilizados en la actualidad.

### 2.2.2 Android Studio

Android Studio [17] es un entorno de desarrollo integrado (IDE) altamente utilizado por los desarrolles de aplicaciones móviles Android. Diseñado para la plataforma Android, Android Studio ofrece una variedad de herramientas poderosas para la creación, edición, depuración y prueba de aplicaciones Android. Su interfaz intuitiva, el tener incorporado emuladores de dispositivos y una amplia gama de bibliotecas y recursos hacen que sea la elección principal para la comunidad de desarrolladores de Android, y por esto, se eligió esta herramienta para el desarrollo del proyecto. Además, Android Studio se mantiene actualizado con las últimas tecnologías de Android, lo que facilita la

*Figura 8. Logo Android Studio*

incorporación de nuevas características y funcionalidades en las aplicaciones móviles.

### 2.2.3 Postman

Postman [18] es una herramienta multitarea imprescindible para desarrolladores de software, simplificando la interacción con APIs al permitirles crear, enviar y gestionar solicitudes HTTP personalizadas, incluyendo solicitudes GET, POST, PUT, DELETE y PATCH, organizarlas en colecciones y automatizar pruebas para garantizar el correcto funcionamiento de las APIs. También facilita la documentación de las APIs y ofrece capacidades de colaboración en equipo. Postman es fundamental para el desarrollo, pruebas y monitoreo de APIs, lo que agiliza el proceso de construcción de aplicaciones web y móviles que dependen de servicios web, y se ha convertido en una herramienta estándar en la industria.

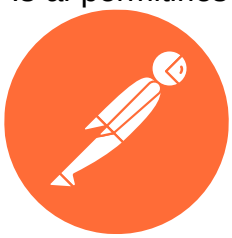


*Figura 9. Logo Postman*

### 2.2.4 Pokémon TCG API

Pokémon TCG API [19] es una fuente esencial de información para entusiastas y desarrolladores del juego de cartas Pokémon. Esta API brinda, de forma gratuita, acceso a datos detallados sobre cartas individuales, expansiones, conjuntos de cartas y reglas del juego. También proporciona información sobre estrategias de mazo, precios de cartas y más. Con su capacidad de búsqueda y filtrado, así como la documentación completa, Pokémon TCG API se convierte en una herramienta inestimable para los amantes del juego y para aquellos que deseen crear aplicaciones relacionadas con el juego de cartas. Aunque es cierto que pueden existir otras APIs más completas pero que no están disponibles para su uso actualmente, su disponibilidad gratuita la destacan en el campo, ya que muchas otras fuentes gratuitas, carecen de la amplitud y la calidad de información que ofrece esta API.

# 3

# Especificación y análisis

Como en cualquier proyecto software, un primer paso esencial es definir de forma clara cuáles son los requisitos (tanto funcionales como no funcionales) que debe cumplir. Una vez están claramente especificadas esos requisitos, se deben indicar los casos de uso del producto desarrollado. En este capítulo describiremos esos aspectos.

## 3.1 Actores del sistema

En este proyecto no se ha considerado diferenciar distintos actores del sistema, si no que existe el usuario registrado que podrá utilizar la aplicación y un usuario no registrado que hasta que no se registre no podrá hacer uso de la aplicación.

## 3.2 Requisitos funcionales

- RF-USER-01: El sistema debe permitir al usuario registrarse en la aplicación mediante su correo.
- RF-USER-02: El sistema debe permitir al usuario iniciar sesión utilizando su correo y su contraseña.
- RF-USER-03: El sistema debe permitir al usuario cerrar sesión.

- RF-CARD-01: La aplicación debe poder mostrar un listado de todas las colecciones y sus respectivas cartas que están disponible en la API.
- RF-CARD-02: El usuario podrá consultar los datos individuales de una carta. De cada carta debe mostrarse al menos, su nombre, numero en la Pokédex, tipo de carta, tipo del Pokémon, número en la colección, rareza, ilustrador/a, precio y enlace a cardmarket [20] para adquirir la carta.
- RF-CARD-03: El usuario podrá añadir cartas a su colección.
- RF-CARD-04: El usuario podrá editar la cantidad de cartas de una carta de su colección.
- RF-CARD-05: La aplicación debe poder mostrar un listado de las cartas filtradas al menos por, colección, tipo del Pokémon, tipo de carta, precio, ilustrador/a, generación, Pokémon y cartas de la colección del usuario.
- RF-CARD-06: La aplicación debe permitir eliminar cartas a la *Wishlist* del usuario, indicando el precio deseado por parte del usuario.
- RF-CARD-07: La aplicación debe permitir añadir cartas a la *Wishlist* del usuario, indicando el precio deseado por parte del usuario.
- RF-CARD-08: El usuario debe poder ir a la página de cardmarket de la carta seleccionada.
- RF-SYSTEM-01: El sistema debe ser capaz de otorgar la puntuación correspondiente al usuario al completar un logro.
- RF-SYSTEM-02: El sistema debe mostrar una clasificación que incluya todos los usuarios registrados y sus puntuaciones, ordenándolos de mayor a menor puntuación.
- RF-SYSTEM-03: El sistema debe mostrar datos estadísticos sobre la colección del usuario, incluyendo carta más valiosa, número total de cartas de la colección, número total de cartas únicas de la colección, valor total de la colección, número de cartas por cada tipo Pokémon, número de cartas por cada tipo de carta, número de cartas por cada rareza, colección en su colección, y número de colecciones completadas.
- RF-SYSTEM-04: El sistema debe enviar un correo al usuario cuando una carta de su *Wishlist* esté al mismo o menor precio que el precio deseado.
- RF-SYSTEM-05: La aplicación debe indicar si un intercambio es justo o no.

- RF-SYSTEM-06: La aplicación debe sugerir qué cartas puede dar el usuario en el intercambio si así lo ha indicado previamente el usuario.
- RF-SYSTEM-07: La aplicación debe sugerir qué cartas puede recibir el usuario en el intercambio si así lo ha indicado previamente el usuario.

## 3.3 Requisitos no funcionales

- RNF-01: El sistema debe tener acceso en todo momento a la API de las cartas.
- RNF-02: El sistema debe ofrecer seguridad y protección de datos de usuario, como contraseñas y datos personales.
- RNF-03: El sistema debe ofrecer un rendimiento y velocidad de trabajo que permita al usuario tener una experiencia fluida.
- RNF-04: El sistema debe ser escalable para manejar un crecimiento en el número de usuarios y cartas.
- RNF-05: El sistema debe proporcionar una interfaz de usuario intuitiva y amigable.

## 3.4 Casos de uso

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-01: Inicio de sesión |
| Actores | Usuario registrado |
| Descripción | El usuario introducirá su correo y su contraseña para iniciar sesión. |
| Precondiciones | El usuario aún no ha iniciado sesión y se encuentra en la pantalla de inicio de sesión. |
| Requisito | RF-USER-02 |

| | |
|---|---|
| Escenario Principal | 1. El usuario introduce su correo y su contraseña en la pantalla de inicio de sesión y pulsa el botón de inicio de sesión.<br>2. El usuario accede al menú principal de la aplicación con la sesión iniciada. |
| Escenario Alternativo | 2b. El correo o la contraseña es incorrecta, se muestra por pantalla un mensaje de error, se mantiene en la misma pantalla y no inicia sesión. |

*Tabla 1. Caso de uso: Iniciar sesión*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-02: Cerrar sesión |
| Actores | Usuario registrado |
| Descripción | El usuario con sesión iniciada podrá cerrar su sesión. |
| Precondiciones | El usuario tiene la sesión iniciada y se encuentra en el menú principal de la aplicación. |
| Requisito | RF-USER-03 |
| Escenario Principal | 1. El usuario presiona el botón de cierre de sesión.<br>2. La sesión es cerrada y el usuario vuelve a la pantalla de inicio de la aplicación. |
| Escenario Alternativo | No hay escenario alternativo. |

*Tabla 2. Caso de uso: Cerrar sesión*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-03: Registrar usuario |
| Actores | Usuario no registrado |

| | |
|---|---|
| Descripción | El usuario introduce sus datos y se registra en la aplicación. |
| Precondiciones | El usuario no está registrado y se encuentra en la pantalla de registro. |
| Requisito | RF-USER-01 |
| Escenario Principal | 1. El usuario introduce un nombre de usuario, su correo, y su contraseña 2 veces y pulsa el botón para registrarse.<br>2. El usuario es registrado y vuelve a la pantalla de inicio de la aplicación. |
| Escenario Alternativo | 2b. Ya existe una cuenta asociada a ese correo por lo que el registro no se hace, se muestra un mensaje de error y se mantiene en esa pantalla.<br>2c. Las contraseñas no coinciden por lo que el registro no se hace, se muestra un mensaje de error y se mantiene en esa pantalla. |

*Tabla 3. Caso de uso: Registrar usuario*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-04: Añadir carta a la colección del usuario |
| Actores | Usuario registrado |
| Descripción | El usuario agrega a su colección una carta. |
| Precondiciones | El usuario se encuentra en la pantalla de la carta desde el apartado de Colecciones o W*ishlist*. |
| Requisito | RF-CARD-03 |
| Escenario Principal | 1. El usuario presiona el botón para añadir a la colección.<br>2. El usuario introduce la cantidad de cartas, de esa carta que va a añadir.<br>3. La carta es añadida a la colección del usuario. |

| | |
|---|---|
| Escenario Alternativo | 2b. Si la cantidad es 0 o negativa se muestra un mensaje de error y se mantiene en la misma pantalla.<br>3b. Si la carta estuviese en la *wishlist*, esta además de ser añadida a la colección, será borrada de la *wishlist*. |

*Tabla 4. Caso de uso: Añadir carta a la colección del usuario*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-05: Añadir carta a la *wishlist* del usuario |
| Actores | Usuario registrado |
| Descripción | El usuario agrega a su *wishlist* una carta. |
| Precondiciones | El usuario se encuentra en la pantalla de la carta desde el apartado de colecciones. |
| Requisito | RF-CARD-06 |
| Escenario Principal | 1. El usuario presiona el botón para añadir a la *wishlist*.<br>2. El usuario elige a qué precio quiere que se le avise.<br>3. La carta es añadida a la wishlist del usuario. |
| Escenario Alternativo | 2b. En caso de introducir un precio no válido, se avisará al usuario mediante un mensaje de error y se mantendrá en la misma pantalla.<br>3b. Esa carta ya pertenece a la *wishlist* del usuario y se le avisa por un mensaje al usuario y por tanto no se añade de nuevo y se mantiene en la misma pantalla. |

*Tabla 5. Caso de uso: Añadir carta a la wishlist del usuario*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-06: Editar cantidad de cartas |
| Actores | Usuario registrado |

| | |
|---|---|
| Descripción | El usuario edita la cantidad de cartas de una carta de su colección. |
| Precondiciones | El usuario está en la pantalla de carta desde mi colección. |
| Requisito | RF-CARD-04 |
| Escenario Principal | 1. El usuario presiona el botón editar.<br>2. El usuario introduce una nueva cantidad de cartas.<br>3. Se actualiza la cantidad de cartas de esa carta del usuario. |
| Escenario Alternativo | 3b. Si la cantidad es 0 se eliminará esa carta de la colección del usuario y volverá al listado de cartas del apartado "Mi colección".<br>3c. Si la cantidad es negativa saldrá un mensaje de error y no se actualizará la cantidad de cartas y se mantendrá en la misma pantalla. |

*Tabla 6. Caso de uso: Editar cantidad de cartas*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-07: Filtrar cartas del usuario |
| Actores | Usuario registrado |
| Descripción | El usuario podrá filtrar las cartas por diferentes características. |
| Precondiciones | El usuario se encuentra en la pantalla de "Mi colección" o "Cartas a recibir" del tasador. |
| Requisito | RF-CARD-05 |
| Escenario Principal | 1. El usuario selecciona el tipo de filtro que quiere utilizar.<br>2. El usuario selecciona o introduce los datos necesarios según el filtro que haya elegido.<br>3. Se muestran las cartas filtradas. |

| | |
|---|---|
| Escenario Alternativo | 2b. Si introduce algún dato incorrecto no se mostrarán cartas, se mostrará un mensaje de error y se mantendrá en la misma pantalla. |

*Tabla 7. Caso de uso: Filtrar cartas del usuario*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-08: Eliminar carta de la *wishlist* |
| Actores | Usuario registrado |
| Descripción | El usuario elimina una carta de su *wishlist.* |
| Precondiciones | El usuario se encuentra en la pantalla de carta desde su *wishlist.* |
| Requisito | RF-CARD-07 |
| Escenario Principal | 1.El usuario presiona el botón de eliminar de la *wishlist.*<br>2.La carta es eliminada de la wishlist del usuario. |
| Escenario Alternativo | No hay escenario alternativo. |

*Tabla 8. Caso de uso: Eliminar carta de la wishlist*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-09: Ir a cardmarket |
| Actores | Usuario registrado |
| Descripción | El usuario puede acceder a la página para poder comprar la carta si así lo desea. |
| Precondiciones | El usuario se encuentra en la pantalla de la carta desde el apartado de "Colecciones" o "W*ishlist".* |
| Requisito | RF-CARD-08 |

| | |
|---|---|
| Escenario Principal | 1. El usuario pulsa el botón para ser enviado a la página web de cardmarket.<br>2. El usuario es enviado a la página de cardmarket de la carta en cuestión. |
| Escenario Alternativo | No hay escenario alternativo. |

*Tabla 9. Caso de uso: Ir a cardmarket*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-10: Tasar un intercambio (sin sugerencias) |
| Actores | Usuario registrado |
| Descripción | El usuario introducirá las cartas que va a dar, y las que va a recibir en el intercambio y la aplicación le dirá si es un intercambio justo o no. |
| Precondiciones | El usuario debe encontrarse en la pantalla del "Tasador". |
| Requisito | RF-SYSTEM-05 |
| Escenario Principal | 1. El usuario selecciona que sí sabe que cartas va a dar.<br>2. El usuario selecciona las cartas de su colección que va a dar en el intercambio.<br>3. El usuario selecciona el botón "Finalizar".<br>4. El usuario selecciona que sí sabe que cartas va a recibir.<br>5. El usuario selecciona las cartas que va a recibir.<br>6. El usuario selecciona el botón "Finalizar".<br>7. Se muestra la pantalla con las cartas a dar y su valor total, las cartas a recibir y su valor total y si el intercambio es justo o no. |

| | |
|---|---|
| Escenario Alternativo | 2b. El usuario introduce un número de cartas no válido por lo que se muestra un mensaje de error y se mantiene en la misma pantalla.<br>3b. Se mostrará un mensaje de error si el usuario no ha seleccionado ninguna carta y se mantiene en la misma pantalla.<br>6b. Se mostrará un mensaje de error si el usuario no ha seleccionado ninguna carta y se mantiene en la misma pantalla. |

*Tabla 10. Caso de uso: Tasar un intercambio (sin sugerencias)*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-11: Tasar un intercambio (sugerencia de que cartas puedes dar) |
| Actores | Usuario registrado |
| Descripción | El usuario introducirá las cartas que va a recibir en el intercambio y la aplicación a través de un algoritmo le mostrará una sugerencia de las cartas que puede dar de su colección para que sea un intercambio justo, en el caso de que sea posible. |
| Precondiciones | El usuario debe encontrarse en la pantalla del "Tasador". |
| Requisito | RF-SYSTEM-06 |
| Escenario Principal | 1. El usuario selecciona que no sabe que cartas va a dar.<br>2. El usuario selecciona que sí sabe que cartas va a recibir.<br>3. El usuario selecciona las cartas que va a recibir.<br>4. El usuario selecciona el botón "Finalizar".<br>5. Se muestra la pantalla con las cartas a dar obtenidas por el algoritmo y su valor total, las cartas a recibir y su valor total y si el intercambio es justo o no. |

| | |
|---|---|
| Escenario Alternativo | 4b. Se mostrará un mensaje de error si el usuario no ha seleccionado ninguna carta y se mantendrá en la misma pantalla. |

*Tabla 11. Caso de uso: Tasar un intercambio (sugerencia de que cartas puedes dar)*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-12: Tasar un intercambio (sugerencia de que cartas puedes recibir) |
| Actores | Usuario registrado |
| Descripción | El usuario introducirá las cartas que va a dar en el intercambio y la aplicación a través de un algoritmo le mostrará una sugerencia de las cartas que podría recibir para que sea un intercambio justo. |
| Precondiciones | El usuario debe encontrarse en la pantalla del "Tasador". |
| Requisito | RF-SYSTEM-07 |
| Escenario Principal | 1. El usuario selecciona que sí sabe que cartas va a dar.<br>2. El usuario selecciona las cartas de su colección que va a dar en el intercambio.<br>3. El usuario selecciona el botón "Finalizar".<br>4. El usuario selecciona que no sabe que cartas va a recibir.<br>5. El usuario selecciona si va a querer una búsqueda rápida o lenta<br>6. Se muestra la pantalla con las cartas a dar obtenidas por el algoritmo y su valor total, las cartas a recibir y su valor total y si el intercambio es justo o no. |
| Escenario Alternativo | 2b. El usuario introduce un número de cartas no válido por lo que se muestra un mensaje de error y se mantendrá en la misma pantalla. |

| | |
|---|---|
| | 3b. Se mostrará un mensaje de error si el usuario no ha seleccionado ninguna carta y se mantendrá en la misma pantalla. |

*Tabla 12. Caso de uso: Tasar un intercambio (sugerencia de que cartas puedes recibir)*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-13: Mostrar cartas de una colección |
| Actores | Usuario registrado |
| Descripción | La aplicación muestra el listado de todas las cartas de una colección. |
| Precondiciones | El usuario se encuentra en el menú principal. |
| Requisito | RF-CARD-01 |
| Escenario Principal | 1. El usuario presiona el botón de colecciones.<br>2. Por defecto se elige la colección "Base" y se muestra por pantalla el listado de cartas de la colección. |
| Escenario Alternativo | 2b. Si el usuario selecciona otra colección, se mostrarán las cartas de dicha colección en la misma pantalla. |

*Tabla 13. Caso de uso: Mostrar cartas de una colección.*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-14: Mostrar datos de una carta. |
| Actores | Usuario registrado |
| Descripción | El usuario selecciona una carta para ver sus datos. |
| Precondiciones | El usuario se encuentra en la pantalla de "Colecciones", "Mi Colección" o "*Wishlist*". |

| | |
|---|---|
| Requisito | RF-CARD-02 |
| Escenario Principal | 1. El usuario selecciona cualquier carta.<br>2. Se muestran por pantalla los datos de la carta seleccionada. |
| Escenario Alternativo | No hay escenario alternativo. |

*Tabla 14. Caso de uso: Mostrar datos de una carta.*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-15: Completar logro |
| Actores | Usuario registrado |
| Descripción | El sistema al detectar que se completa un logro le otorga la puntuación correspondiente al usuario que lo ha completado. |
| Precondiciones | El usuario acaba de añadir una carta a su colección. |
| Requisito | RF-SYSTEM-01 |
| Escenario Principal | 1. Se completa el logro al añadir esa carta.<br>2. Se muestra un mensaje por pantalla de que se ha completado el logro.<br>3. Internamente se le añade al usuario la puntuación del logro completado. |
| Escenario Alternativo | 1b. No se completa ningún logro por lo que no pasa nada. |

*Tabla 15. Caso de uso: Completar logro*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-16: Mostrar clasificación |
| Actores | Usuario registrado |
| Descripción | El sistema muestra al usuario la clasificación en la que se muestran todos los usuarios registrados en la aplicación con sus respectivas puntuaciones ordenados de mayor a menor puntuación. |
| Precondiciones | El usuario se encuentra en el menú principal. |
| Requisito | RF-SYSTEM-02 |
| Escenario Principal | 1. El usuario presiona el botón “Ranking”<br>2. Se muestra la pantalla con la clasificación. |
| Escenario Alternativo | No hay escenario alternativo. |

*Tabla 16. Caso de uso: Mostrar clasificación.*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-17: Mostrar estadísticas del usuario |
| Actores | Usuario registrado |
| Descripción | El sistema muestra los datos estadísticos de la colección del usuario. |
| Precondiciones | El usuario se encuentra en el menú principal. |
| Requisito | RF-SYSTEM-03 |
| Escenario Principal | 1. El usuario presiona el botón “Estadísticas”<br>2. Se muestra la pantalla con los cuatro datos principales.<br>3. El usuario presiona el botón “Más estadísticas”. |

| | |
|---|---|
| | 4. Se muestra la pantalla con el resto de las estadísticas. |
| Escenario Alternativo | No hay escenario alternativo. |

*Tabla 17. Caso de uso: Mostrar estadísticas del usuario.*

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-18: Enviar notificación a través de un correo. |
| Actores | Usuario registrado |
| Descripción | El sistema detecta que una carta de la *wishlist* del usuario se encuentra al precio deseado. |
| Precondiciones | El usuario tiene cartas en su *wishlist*. |
| Requisito | RF-SYSTEM-04 |
| Escenario Principal | 1. El sistema actualiza el precio de las cartas guardadas en la base de datos.<br>2. El sistema detecta que una carta ha bajado de precio y se encuentra al precio deseado o por debajo del precio deseado.<br>3. El sistema envía un correo al usuario notificando que la carta X de su *wishlist* está al precio deseado. |
| Escenario Alternativo | 2b. Ninguna carta está al precio deseado o más barata, por lo que no pasa nada. |

*Tabla 18. Caso de uso: Enviar notificación a través de un correo.*

# 4

# Diseño del sistema

En este apartado se explicará detalladamente la estructura y el modelo del sistema del proyecto, dando los motivos por lo que se ha decidido optar por estas opciones.

## 4.1 Arquitectura de la aplicación

Teniendo en cuenta las funcionalidades que puede realizar la aplicación, se podía elegir entre dos opciones principales, el tener una arquitectura Modelo-Vista-ViewModel (MVVM) o una arquitectura cliente-servidor. Finalmente se optó por esta última ya que nos facilitaba la implementación del proyecto. Tendríamos el servidor que actúa como una API REST y se encargará de manejar las solicitudes de datos, autenticación y de todas las operaciones relaciones con los datos. Y por el otro lado tendríamos el cliente, el cual se trataría de la propia aplicación móvil, la cual se comunicará con el servidor indicando las peticiones que quiere realizar, y será el servidor como hemos mencionado anteriormente, el encargado de procesar y realizar esas peticiones devolviendo el resultado obtenido al cliente.

Los beneficios que nos asegura esta elección es que nos garantiza la disponibilidad, la seguridad y escalabilidad de la aplicación, favoreciendo un acceso rápido a la API de cartas y facilitando la actualización de la base de datos.

En la siguiente imagen podemos ver una representación visual de la arquitectura del sistema, donde tenemos los clientes, que son los usuarios con sus teléfonos móviles, el servidor y la base de datos.

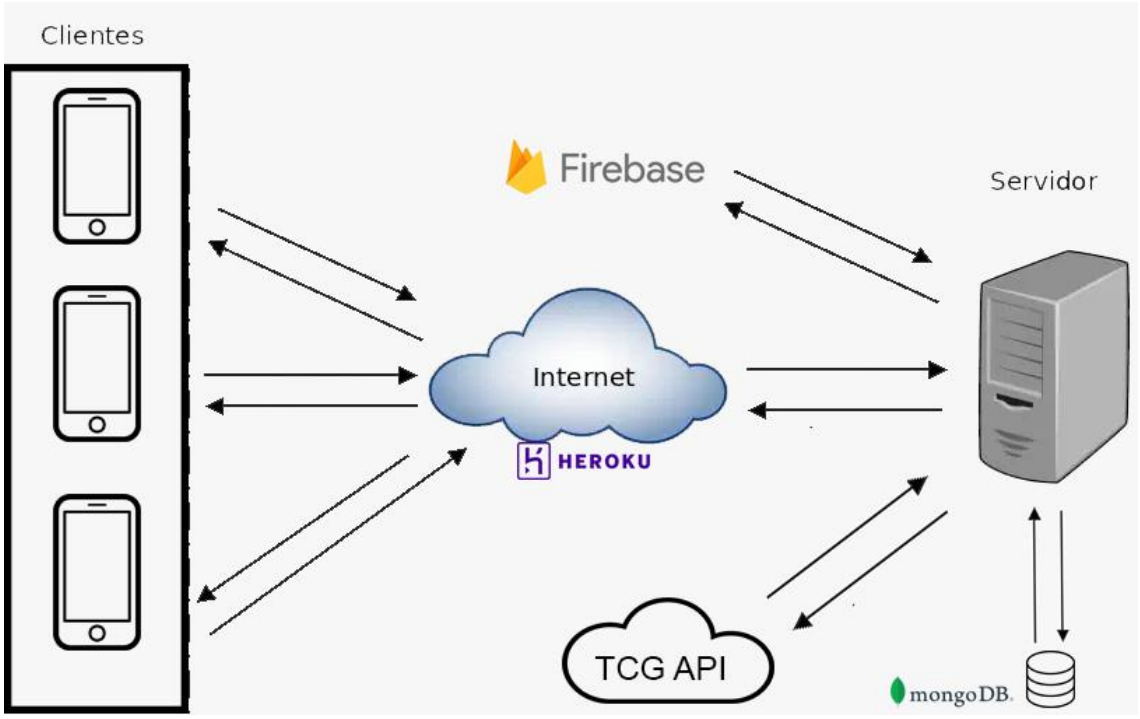



*Figura 10. Representación de la arquitectura*

El cliente se corresponde con la aplicación móvil desarrollada en Android utilizando el lenguaje de programación Java, la cual es usada por los usuarios desde sus teléfonos móviles. Desde la aplicación, recogemos los datos del servidor que hemos solicitado a través de peticiones HTTP.

El servidor, desarrollado con el lenguaje de programación Python, y más en concreto Flask, es el encargado de procesar las peticiones HTTP solicitadas por el cliente y se conecta con la base de datos para actualizar la información necesaria. También, estará conectado con firebase para la autenticación de los usuarios tal y como se comentó previamente. Además, también es el encargado de realizar las peticiones a la API externa utilizada para obtener todo lo relacionado con las cartas Pokémon.

## 4.2 Diseño de la base de datos

La aplicación hace uso de una Base de Datos no relacional en MongoDB. Dado que se trata de una base de datos relacional no se puede representar mediante un diagrama entidad-relación, por lo que se mostrará en la siguiente imagen las distintas colecciones que forman la base de datos.

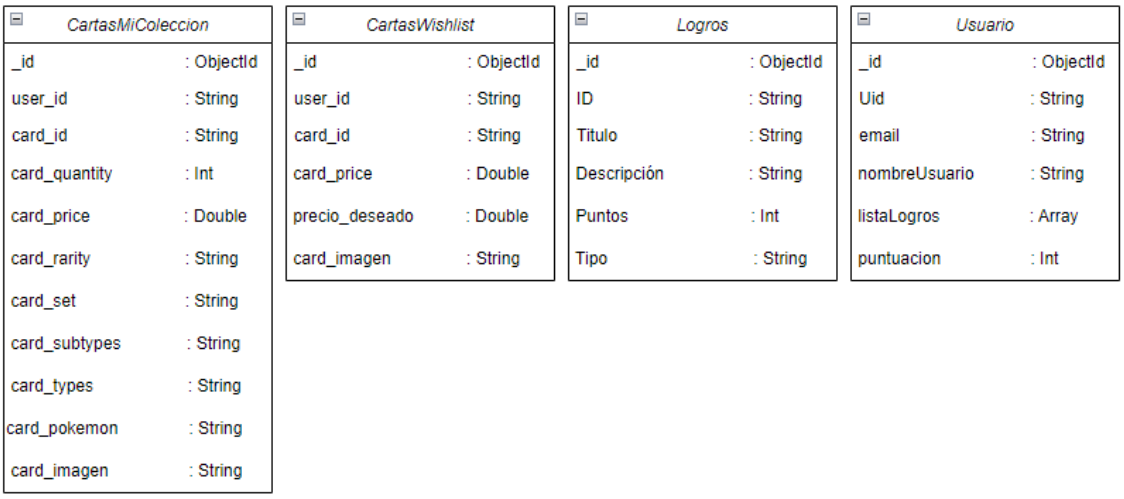



*Figura 11. Diagrama de la base de datos*

Como se ha comentado con anterioridad, al tratarse de una base de datos no relacional no existen relaciones explícitas entre los atributos de cada colección. A continuación, se describirá los distintos elementos que forman la base de datos.

### 4.2.1 CartasMiColección

Esta colección representa las cartas que han sido añadida a la colección de cada usuario. Y a continuación se explicará que representa cada atributo.

- _id: Id que genera MongoDB por defecto. Este id identifica cada objeto de la colección. Este id es de tipo "*ObjectId*", tipo propio de MongoDB.
- user_id: Atributo de tipo "*String*" que representa el id del usuario al que se le va a añadir la carta a su colección.
- card_id: Atributo de tipo "*String*" que representa el id de la carta que va a añadirse a la colección del usuario.

- card_quantity: Atributo de tipo “Int” que representa la cantidad de cartas que el usuario tiene de la carta que va a añadir a su colección.
- card_price: Atributo de tipo “Double” que representa el precio de la carta.
- card_rarity: Atributo de tipo “*String*” que representa la rareza de la carta.
- card_set. Atributo de tipo “*String*” que representa el id de la colección a la que pertenece la carta.
- card_subtypes: Atributo de tipo “*String*” que representa el subtipo de la carta.
- card_types: Atributo de tipo “*String*” que representa el tipo de la carta.
- card_pokemon: Atributo de tipo “*String*” que representa el Pokémon de la carta.
- card_imagen: Atributo de tipo “String” que representa la *url* de la imagen de la carta.

### 4.2.2 CartasWishlist

Esta colección representa las cartas que han sido añadida a la *wishlist* de cada usuario. Y a continuación se explicará que representa cada atributo.

- _id: Id que genera MongoDB por defecto. Este id identifica cada objeto de la colección. Este id es de tipo “*ObjectId*”, tipo propio de MongoDB
- user_id: Atributo de tipo “*String*” que representa el id del usuario al que se le va a añadir la carta a su colección.
- card_id: Atributo de tipo “*String*” que representa el id de la carta que va a añadirse a la colección del usuario.
- card_price: Atributo de tipo “Double” que representa el precio de la carta.
- precio_deseado: Atributo de tipo “Double” que representa el precio máximo que el usuario estaría dispuesto a pagar por la carta. Se utiliza para avisar al usuario cuando la carta esté por ese precio o menor y se procede al envío de un correo electrónico.

- card_imagen: Atributo de tipo “String” que representa la *url* de la imagen de la carta.

### 4.2.3 Logros

Esta colección representa cada logro que está registrado en la aplicación. Y a continuación se explicará que representa cada atributo.

- _id: Id que genera MongoDB por defecto. Este id identifica cada objeto de la colección. Este id es de tipo “*ObjectId*”, tipo propio de MongoDB.
- ID: Atributo de tipo “String” que representa el id del logro.
- Titulo: Atributo de tipo “String” que representa el título del logro.
- Descripción: Atributo de tipo “String” que representa una breve descripción sobre qué se debe hacer para conseguir el logro.
- Puntos: Atributo de tipo “Int” que representa la cantidad de puntos que otorga obtener el logro.
- Tipo: Atributo de tipo “String” que representa el tipo de logro que es. Finalmente, por falta de tiempo este atributo no se ha utilizado en el proyecto.

### 4.2.4 Usuario

Esta colección representa cada usuario que ha sido registrado en la aplicación. Y a continuación se explicará que representa cada atributo.

- _id: Id que genera MongoDB por defecto. Este id identifica cada objeto de la colección. Este id es de tipo “*ObjectId*”, tipo propio de MongoDB.
- Uid: Id que genera *Firebase* a cada usuario registrado, y es utilizado en las colecciones CartasMiColección y CartasWishlist como el atributo user_id. Este id es el utilizado durante la aplicación para identificar al usuario que ha iniciado sesión.
- email: Atributo de tipo “String” que representa el email utilizado por el usuario que se ha registrado. Para el inicio de sesión se utilizará este email.

- nombreUsuario: Atributo de tipo “String” que representa el *nickname* del usuario y que será utilizado para la clasificación.
- listaLogros: Atributo de tipo “Array de logros” y representa el listado de logros que el usuario ha conseguido.
- puntuación: Atributo de tipo “Int” que representa la puntuación total del usuario, y será utilizado para la clasificación.

# 5

# Implementación

En este apartado se explicará paso por paso cómo se ha desarrollado el proyecto, describiendo los distintos procedimientos que se han seguido. Cada apartado estará dedicado a cada iteración que se ha realizado. Cada funcionalidad que ha sido implementada ha seguido el mismo patrón de implementación, un primer diseño de la interfaz en Android Studio que actúa como boceto inicial, posteriormente se programa la funcionalidad en cuestión, y se termina realizando el diseño final de la interfaz.

## 5.1 Arranque del proyecto

Como cada proyecto, primero partimos de una idea que queremos llevar a cabo y enumeramos los requisitos que queremos que formen parte nuestro proyecto para así poder saber con certeza que vamos a necesitar en el desarrollo del proyecto.

### 5.1.1 Creación del proyecto

Para la creación del proyecto se han utilizado dos herramientas principales, Visual Studio Code para el servidor y un proyecto en Android Studio para el cliente.

En Visual Studio Code se ha creado un proyecto Python, más en concreto utilizando Flask, y con esto se ha desarrollado el servidor. Este servidor solamente consta de una clase principal que es app.py en el cual se han

instalado todas las librerías necesarias para la programación del servidor como por ejemplo Flask, PyMongo, request, jsonify, etc.

Y posteriormente se ha creado el proyecto para el cliente en Android Studio, para esto se ha creado un nuevo proyecto haciendo uso de la versión 32 del SDK (Kit de desarrollo de software que genera un conjunto de herramientas útiles para la programación).

### 5.1.2 Base de datos

Una vez creados los proyectos anteriores, se ha creado un clúster gratuito de Amazon Web Services (AWS) en MongoDB Atlas. Dentro de este clúster, se ha creado una base de datos llamada PTCGCollector. Esta, está formada por las colecciones Usuarios, CartasMiColeccion y CartasWishlist que estarán inicializadas vacías. Además de estas colecciones habrá una colección más llamada Logros, la cual si estará inicializada con los datos de los distintos logros existentes.

Una vez que se ha creado se ha hecho la conexión del servidor con la base de datos haciendo uso de la biblioteca PyMongo, de MongoClient y de la URL que genera la base de datos (véase Figura 12).

```
app = Flask(__name__)
app.config['MONGO_URI']='mongodb+srv://ismael:' + config.PASSWORD + '@cluster0.byvnlpx.mongodb.net/PTCGDatabase'

mongo = PyMongo(app)
```

*Figura 12. Conexión con la base de datos*

### 5.1.3 Peticiones HTTP a través de la librería Volley

Como se explicó en el apartado de tecnologías, para realizar las peticiones entre el cliente y el servidor se ha hecho uso de la biblioteca Volley. Esta biblioteca se ha añadido al proyecto de Android Studio como una dependencia, y se utiliza para cualquier petición que se quiera hacer entre el cliente y el servidor.

Para realizar las siguientes peticiones se necesitan los siguientes elementos (Véase Figura 13):

- URL del punto de acceso a la consulta que se quiere realizar.
- Crear una cola de peticiones (Request Queue)
- Se crea y ejecuta la llamada con el tipo de petición que se va a hacer, es decir si la consulta devuelve un array de objetos de tipo JSON o un solo objeto de tipo JSON y espera una respuesta del servidor.
- Tras hacer la petición hay dos opciones, que se realice con éxito haciendo que pase al método onResponse de la petición, o por el contrario falle y se ejecutaría el método onErrorResponse.

```
RequestQueue queue = Volley.newRequestQueue( context: this);
    JsonArrayRequest request = new JsonArrayRequest(Request.Method.GET, url,  jsonRequest: null,
        new Response.Listener<JSONArray>() {
            @Override
            public void onResponse(JSONArray response) {
                // Procesar los datos de la carta y crear un objeto Carta
                try {
                    JSONObject jsonObject = response.getJSONObject( index: 0);
                    precioAlto = jsonObject.getString( name: "highest_price");
                    totalCartas = jsonObject.getString( name: "total_cards");
                    totalCartasUnicas = jsonObject.getString( name: "total_unique_cards");
                    double valorTotal = jsonObject.getDouble( name: "total_value");
                    DecimalFormat decimalFormat = new DecimalFormat( pattern: "0.00");
                    formattedValorTotal = decimalFormat.format(valorTotal);
                    mostrarEstadisticas();
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
        },
        new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) { error.printStackTrace(); }
        });
queue.add(request);
```

*Figura 13. Ejemplo de Request*

## 5.2 Iteración 1

En esta primera iteración se comenzó con el desarrollo de la API REST y además se implementó las primeras funcionalidades en la aplicación. Se realizó la extracción de todas las colecciones del API REST externa utilizada además

de las cartas, añadiendo consultas para extraer cada set de forma individual y cada carta de forma individual. Por tanto, se agregó la funcionalidad de Colecciones, la cual extraía todas las colecciones existentes y las cartas pertenecientes a esa colección, y al clicar en cada carta extrae sus datos. En la Figura 14, podemos observar cómo es el proceso de la petición GET para obtener todos los sets, primero se hace una llamada al API con el atributo orderBy='releaseDate' para que devuelva los sets ordenados de más antiguos a los más recientes. Posteriormente creamos una lista vacía, y recorriendo la lista de sets, por cada set creamos un diccionario con los datos que nos interesa de cada set, en este caso su id, nombre y cantidad de cartas que tiene el set. Finalmente añadimos este diccionario a la lista de sets, y esto es lo que devolverá la consulta.

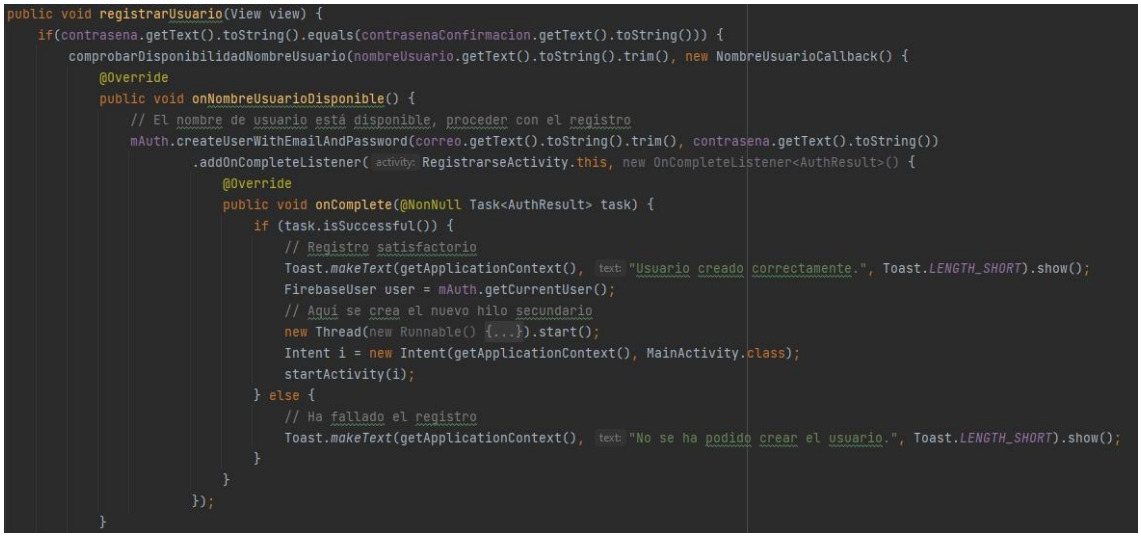
```python
@app.route('/sets' , methods=['GET'])
def find_all_sets():
    sets = Set.where(orderBy='releaseDate')
    set_list = []
    for set in sets:
        set_dict = {
            'id': set.id,
            'name': set.name,
            'total': set.total
        }
        set_list.append(set_dict)
    return jsonify(set_list)
```

*Figura 14. Petición GET para obtener todos los sets*

## 5.3 Iteración 2

En esta segunda iteración se realizó la implementación del registro, e inicio de sesión de usuarios y se estudió cómo manejar el token de inicio de sesión durante la sesión. Como se explicó en el apartado de tecnologías, para esto se utilizó un servicio externo, Firebase. A través del cliente y del servidor, se realizan las peticiones y se consigue crear al usuario, dentro de Firebase. Además, guardamos en la base de datos el nombre de usuario, correo y el id que nos genera Firebase para cada usuario. Este id será el que utilizaremos durante toda la aplicación para asociar al usuario con sus datos.

Una vez que se logró un funcionamiento correcto del registro e inicio de sesión, y se guardaban los datos correctamente en la base de datos, se realizaron los diseños de las pantallas de registro e inicio de sesión.



```
public void registrarUsuario(View view) {
    if(contrasena.getText().toString().equals(contrasenaConfirmacion.getText().toString())) {
        comprobarDisponibilidadNombreUsuario(nombreUsuario.getText().toString().trim(), new NombreUsuarioCallback() {
            @Override
            public void onNombreUsuarioDisponible() {
                // El nombre de usuario está disponible, proceder con el registro
                mAuth.createUserWithEmailAndPassword(correo.getText().toString().trim(), contrasena.getText().toString())
                        .addOnCompleteListener( activity: RegistrarseActivity.this, new OnCompleteListener<AuthResult>() {
                            @Override
                            public void onComplete(@NonNull Task<AuthResult> task) {
                                if (task.isSuccessful()) {
                                    // Registro satisfactorio
                                    Toast.makeText(getApplicationContext(),  text: "Usuario creado correctamente.", Toast.LENGTH_SHORT).show();
                                    FirebaseUser user = mAuth.getCurrentUser();
                                    // Aquí se crea el nuevo hilo secundario
                                    new Thread(new Runnable() {...}).start();
                                    Intent i = new Intent(getApplicationContext(), MainActivity.class);
                                    startActivity(i);
                                } else {
                                    // Ha fallado el registro
                                    Toast.makeText(getApplicationContext(),  text: "No se ha podido crear el usuario.", Toast.LENGTH_SHORT).show();
                                }
                            }
                        });
            }
```

*Figura 15. Registro de usuario*

En la Figura 15 podemos ver el proceso para registrar un usuario. Primero se hace la comprobación de si las contraseñas coinciden, posteriormente se comprueba si el nombre de usuario está disponible, y si todo es correcto se realiza la creación del usuario con el correo y contraseña escritos por el usuario, y esto se almacena en Firebase. Aunque no se vea en la figura, si el usuario está escogido o las contraseñas no coinciden, se mostrará el mensaje indicando el error que ha sucedido, y no se realizará el registro.

## 5.4 Iteración 3

Para esta tercera iteración se ha realizado la implementación de todos los filtros mencionados en el requisito RF-CARD-05. Este trabajo se ha podido facilitar mucho gracias a la API externa, ya que permite realizar peticiones filtradas según los parámetros que se le pidieran, de esta forma no era necesario traer todas las cartas y filtrarlas desde el servidor, si no que desde el servidor ya se recoge de la API el listado de cartas filtradas, haciendo que las peticiones sean mucho más rápidas y fluidas.

Uno de los mayores problemas que se tuvo al realizar las consultas con las cartas fue la cantidad de cartas que recogía la petición, lo que hacía que tardara mucho en procesarse. Para solucionar esto se ha realizado consultas paginadas, es decir por cada petición se recogen veintiún cartas, que son las que se muestran por cada pantalla, y tendremos dos botones, uno anterior y otro siguiente. El botón anterior no se podrá pulsar si está en la página uno, y el botón siguiente por cada página se hace una consulta si existe una carta veintidós, lo que haría indicar que si habría página siguiente y por tanto estaría habilitado, en caso contrario no se podría pulsar. La consulta solo se realiza una vez por página y se guarda el resultado para no tener que realizar la consulta cada vez que se quiera ver una página.

En algunos de los filtros, se tuvo que hacer un tratamiento previo de los datos, como por ejemplo el filtro de Tipo del Pokémon, en pantalla se muestran los nombres en español, pero internamente se hace una traducción del tipo seleccionado ya que en la API los datos aparecen en inglés. Otro filtro con un tratamiento extra sería el de Generación, ya que en pantalla se muestra 1º Generación, 2º Generación, etc. Pero internamente se están pasando al servidor los números de la Pokedex que corresponden con el inicio y fin de la generación.


*Figura 16. Filtro por Colección*


*Figura 17. Filtro por Precio*

Como se puede ver en las figuras anteriores, la pantalla del filtro se adapta al filtro que está seleccionado, ya que para algunos será necesario introducir algún dato manualmente, como en el caso de la Figura 17 el precio mínimo y el precio máximo o si no es necesario simplemente se muestra el *spinner* con las opciones disponibles como se ve en la Figura 16.

## 5.5 Iteración 4

Para esta iteración se ha implementado los datos estadísticos del usuario, y con esto nos permite realizar la implementación del sistema de logros y posteriormente la clasificación.

Primero se comenzó con la contabilización de los datos del usuario, para ello se realizó una consulta en la que se contabilizan todos los datos mencionados en el requisito RF-SYSTEM-03, esto se consigue a través de dos peticiones distintas, una primera en la que se obtiene, el valor de la carta más cara, número total de cartas, número de cartas únicas y valor de la colección del usuario, y una segunda consulta para obtener el resto de los datos estadísticos. En ambos casos, la idea es similar, extraer todas las cartas del usuario e ir realizando las comprobaciones necesarias para obtener los datos de cada una.

```
    if set_id not in sets_dict:
        sets_dict[set_id] = 0

    sets_dict[set_id] += 1

num_sets = 0
num_sets_completos = 0

for set_id, num_cartas_usuario in sets_dict.items():
    set_details = find_set_by_id(set_id)  # Llamada al método para obtener detalles del set

    num_total_cartas_set = set_details['total']

    if num_cartas_usuario == num_total_cartas_set:
        num_sets_completos += 1

    num_sets += 1

stats_dict['Sets'] = num_sets
stats_dict['Sets completos'] = num_sets_completos
```

*Figura 18. Obtener los sets completos*

Cabe destacar el trabajo que se debe realizar para la comprobación de las colecciones completadas, ya que es el dato más difícil de conseguir. Parte del código empleado es el que se puede ver en la Figura 18. Comenzamos creando un diccionario de colecciones llamado sets_dict, en el cual se guardan el id de la colección y la cantidad de cartas de esa colección que posee el usuario. En ese primer if, que se encuentra dentro del bucle de comprobar los datos de las cartas, si el id no pertenece al diccionario se añade con valor cero, ya que posteriormente se aumentará en uno su valor. Se inicializan las variables de num_sets y num_sets_completos, y se realiza un bucle for, para recorrer todo el diccionario. Dentro del bucle for primero obtenemos el número total de cartas que tiene la colección y comprobamos si las cartas que tiene el usuario de esa colección son igual al número total, en caso afirmativo se considera colección completa y aumenta la variable en uno, y si no, no se hace nada, y finalmente se aumenta en uno el número de sets.

## 5.6 Iteración 5

En la quinta iteración se ha implementado el añadir cartas a la *wishlist* del usuario, indicando previamente a qué precio quiere que se le avise, pudiendo elegir un precio introducido manualmente por el usuario, o cuando baje la carta de precio, no importa cuánto. Y la pestaña de la propia *wishlist* en la que se muestran todas las cartas que el usuario ha añadido y donde se puede editar el precio deseado, si así lo desea el usuario.

Además de esto se implementó un método que actualiza los precios de las cartas almacenadas en la base de datos, y además realiza la comprobación de si esos precios se corresponden o son menores al precio deseado de una carta de la *wishlist* del usuario, y en el caso de que sea así, se realizará el envío de un correo electrónico avisando al usuario de la carta que ha bajado de precio junto con un enlace de la carta en cardmarket por si desea comprarla.

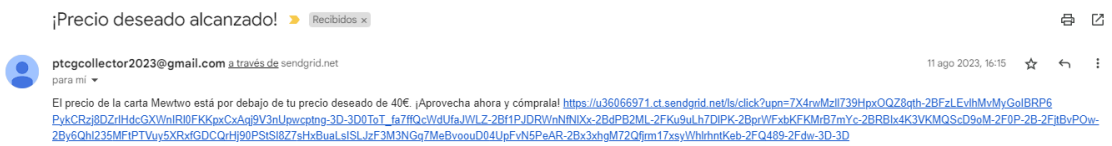



*Figura 19. Correo electrónico enviado por la aplicación.*

Para realizar esto, se hizo uso de una web externa llamada sengrid. Haciendo uso de su API pública podemos enviar correos al usuario tal y como vemos en la Figura 19. Y en la Figura 20 se muestra el código utilizado para conseguirlo.

```
if new_card_price <= card_desired_price:
    sendgrid_api_key = config.SENGRID_API_KEY

    # Configura el remitente y el destinatario del correo electrónico
    sender_email = 'ptcgcollector2123@gmail.com'
    recipient_email = email

    # Crea el contenido del correo electrónico
    subject = '¡Precio deseado alcanzado!'
    message = f'El precio de la carta {card_name} está por debajo de tu precio deseado de {card_desired_price}€. ¡Aprovecha ahora y cómprala! {cardmarket_link}'

    # Envía el correo electrónico utilizando SendGrid
    try:
        sg = SendGridAPIClient(sendgrid_api_key)
        email = Mail(from_email=sender_email, to_emails=recipient_email, subject=subject, plain_text_content=message)
        response = sg.send(email)
        print(f'Correo electrónico enviado a {recipient_email}, estado: {response.status_code}')
    except Exception as e:
        print(f'Error al enviar el correo electrónico: {str(e)}')
```

*Figura 20. Código para el envío de correo*

Por último, para poder lanzar el método de forma automática una vez al día se ha tenido que crear un *scheduler*. Primero se inicia, posteriormente se borran todos los trabajos que pudiera tener para evitar que se lance el mismo trabajo varias veces a la vez, y por último se añade el trabajo que se quiere realizar, en nuestro caso el método updated_cards_prices_and_send_email y se añade un *trigger* con la hora que se quiere lanzar, en este caso a las 16:15. Por último se añadió un método para detener el planificador en el caso de que se apague el servidor. Este código se puede ver en la Figura 21.

```
scheduler = BackgroundScheduler()
scheduler.start()
scheduler.remove_all_jobs()  # Detener el planificador y eliminar todas las tareas existentes

scheduler.add_job(
    updated_cards_prices_and_send_email,
    trigger=CronTrigger(hour=16, minute=15),
    id='actualizar_precios_correo',
    name='Actualizar precios y enviar correos a las 16:15',
    replace_existing=True
)

# Detener el planificador al finalizar el programa
@atexit.register
def shutdown_scheduler():
    scheduler.shutdown()
```

*Figura 21. Scheduler*

## 5.7 Iteración 6

En esta sexta y última iteración se implementó la funcionalidad más compleja del proyecto, el tasador de intercambios. Esta funcionalidad puede actuar de tres formas distintas y vamos a hablar por separado de cada una de ellas.

En la primera, el usuario sabe de antemano las cartas que va a entregar y las cartas que va a recibir en el intercambio, por lo que simplemente deberá introducirlas en sus respectivas pantallas, para que finalmente en la pantalla final se diga si el intercambio es justo o no,

En la segunda, el usuario sabe que cartas va a recibir, pero no sabe que cartas podría dar, en este caso, el sistema a través del algoritmo que se muestra en la Figura 22, este algoritmo es bastante simple, recoge todas las cartas de la colección del usuario y las ordena por precio, y se van añadiendo las cartas que al añadirse al conjunto solución no haga que sobrepase el precio objetivo.

```
cards = sorted(cards, key=lambda card: card['price'], reverse=True)

selected_cards = []
total_value = 0.0

for card in cards:
    while total_value + card['price'] <= price_target and card['quantity'] > 0:
        selected_card = next((c for c in selected_cards if c['id'] == card['id']), None)
        if selected_card:
            selected_card['quantity'] += 1
        else:
            selected_cards.append({
                'id': card['id'],
                'quantity': 1,
                'price': card['price'],
                'imagen': card['imagen']
            })
        total_value += card['price']
        card['quantity'] -= 1
```

*Figura 22. Algoritmo cartas a dar*

En la tercera y última posibilidad, el usuario no sabe que cartas va a recibir en el intercambio, para ello se abren dos posibilidades, que el usuario quiera una búsqueda rápida o una lenta. La rápida es similar al algoritmo mostrado anteriormente, pero en lugar de buscar en las cartas del usuario, se hace con las cartas de la *wishlist* del usuario. Esta búsqueda es instantánea, pero puede producir soluciones no óptimas, ya que puede ser que el usuario tenga pocas cartas en la *wishlist*, o que no sean muy valiosas como para llegar al valor objetivo. Sin embargo la búsqueda lenta, como indica su nombre es lenta, porque además de hacer esta búsqueda en la *wishlist* si no ha encontrado una solución óptima hace lo siguiente: primero recoge todas las cartas del usuario, y las ordena por colección, comprueba cuantas cartas tiene por colección, recoge las cartas totales de todas las colecciones existentes y hace la diferencia con las cartas de la colección que tiene el usuario, ordenando de menor a mayor el número de cartas restantes para completar la colección y recorriendo en ese orden las colecciones en busca de cartas que no superen el valor objetivo y así ofrecer una solución óptima. Parte de este proceso es el que se muestra en la Figura 23.

```python
# Obtener las diferencias de cartas que le faltan al usuario por colección y ordenar de menor a mayor
collection_diff = []
for set_info in sets:
    set_id = set_info['id']
    total_cards = set_info['total']
    user_cards_count = collection_count[set_id]
    diff = total_cards - user_cards_count
    collection_diff.append((set_id, diff))

collection_diff = sorted(collection_diff, key=lambda x: x[1])

for set_info in collection_diff:
    set_id = set_info[0]
    cards_in_collection_response = find_cards_by_set(set_id)
    cards_in_collection = cards_in_collection_response.get_json()

    # Filtrar las cartas que el usuario no tenga y ordenar por precio
    available_cards = []
    for card in cards_in_collection:
        cardmarket_info = card.get('cardmarket')
        if cardmarket_info is not None:
            prices_info = cardmarket_info.get('prices')
            if prices_info is not None and prices_info.get('lowPrice') is not None:
                if card['id'] not in [user_card['id'] for user_card in formatted_cards] and prices_info['lowPrice'] <= price_target:
                    available_cards.append(card)

    available_cards = sorted(available_cards, key=lambda x: x['cardmarket']['prices']['lowPrice'], reverse=True)
```

*Figura 23. Fragmento de código del algoritmo lento.*

Para finalizar con esta iteración, se ha realizado el despliegue del servidor con Heroku, tal y como se ha explicado anteriormente. El método utilizado para realizar esto ha sido a través de un repositorio GitHub (véase Figura 24).

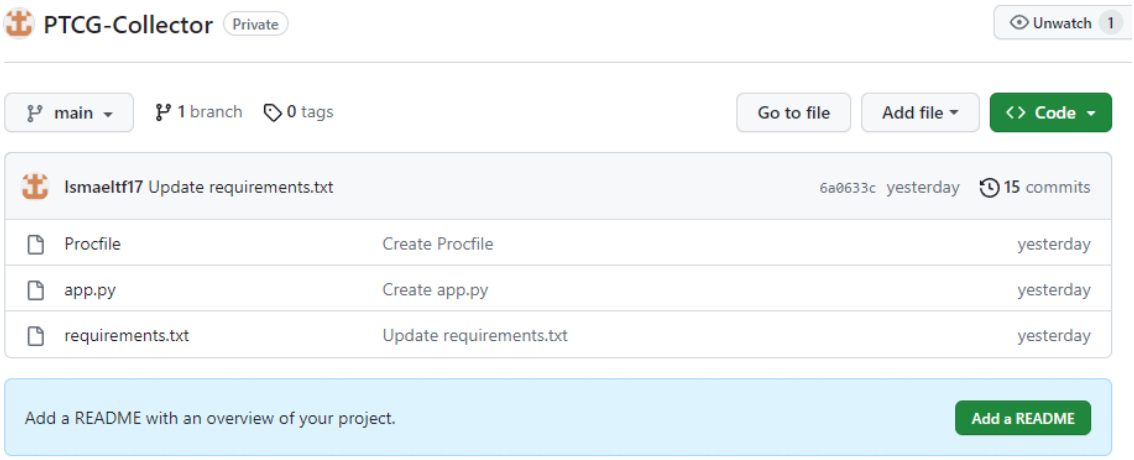



*Figura 24. Repositorio GitHub*

Una vez que tenemos el repositorio creado con el código subido, simplemente debemos ir a Heroku, crear una aplicación y elegir la conexión con GitHub, se autoriza la conexión y elegimos la rama “main” para desplegarla, y finalmente con eso, si no hay ningún problema en el código ya estaría desplegado. Todo este proceso será explicado con mucho más destalle en el Apéndice del Manual de despliegue.

No podemos olvidar, que estamos trabajando con variables secretas, como pueda ser la contraseña para la conexión con la base de datos, o el *api key* que tienen que ser privados, para ello en Heroku podemos ir al apartado configuración y añadir estas variables.

# 6

# Pruebas

En este apartado se describirán las pruebas que se han realizado en el apartado del servidor.  Además, se realizaron pruebas de usabilidad, que ya se explicarán en su apartado.

## 6.1 Pruebas REST API

Como se comentó anteriormente estas pruebas se fueron realizando de forma paralela a su implementación. Para ello se ha usado la herramienta Postman.

Se ha creado una colección de pruebas y dentro de ella se han creado 8 directorios diferentes, para el tipo de dato que maneja la consulta, además de una consulta que no encajaba con ninguno de los directorios anteriores. En estos directorios, se han agregado todas las peticiones que se han realizado para probar que todas las consultas funcionaban de forma correcta. En total se han realizado 32 consultas con sus respectivos *endpoints.* En la Figura 25, se observa cómo se ha estructurado la colección de pruebas, mostrando además las peticiones que se han realizado en los directorios de Logros y *Wishlist*.


*Figura 25. Pruebas REST API*

El proceso de prueba para cada servicio es bastante sencillo y fácil de entender. En primer lugar, se especifica la dirección web donde se encuentra alojado el servicio. En este caso, todos los servicios están alojados en el mismo servidor de Heroku, y la dirección base es la siguiente: "https://ptcgcollector-68d462cc0459.herokuapp.com". Luego, para acceder a un servicio específico, se agrega a esta dirección la colección de datos a la que se quiere acceder, el nombre del método que se desea utilizar y se indica el tipo de solicitud que se va a realizar, que puede ser GET, POST o PUT. Si es necesario enviar información al servicio, se incluye un objeto JSON en el cuerpo de la solicitud.

Una vez hecho esto se envía la petición y hay dos escenarios, en el caso de que la consulta sea correcta devuelve lo que venga especificado en la consulta acompañado de un código 200, pero en el caso de que falle, nos devolverá un error 400 o 500 dependiendo si la consulta no está formulada de forma correcta o si ha sido error del servidor respectivamente además de un mensaje indicando que ha provocado el error.

## 6.2 Pruebas de usabilidad

Las pruebas de usabilidad son evaluaciones sistemáticas en las que los usuarios interactúan con un producto, como un sitio web o una aplicación, para identificar problemas de diseño, usabilidad y experiencia del usuario. Estas pruebas permiten obtener información valiosa sobre cómo los usuarios reales interactúan con el producto, lo que ayuda a los diseñadores y desarrolladores a mejorar la accesibilidad, la eficiencia y la satisfacción del usuario, garantizando así que el producto sea más intuitivo y útil para su público objetivo.

Para realizar estas pruebas se han escogido un grupo de personas ajenas al desarrollo de la aplicación, los cuales podrán probar la aplicación y tras esto rellenar un cuestionario sobre la experiencia. Esta selección de personas se hace teniendo en cuenta que existan personas que, estén familiarizadas con las cartas Pokémon y con aplicaciones de este tipo, personas que están familiarizadas con las cartas Pokémon, pero que nunca han usado este tipo de aplicaciones y personas que no están familiarizados con las cartas Pokémon ni con este tipo de aplicaciones. Por tanto, se han escogido, dos personas familiarizadas con las cartas Pokémon y con aplicaciones de este tipo, dos personas familiarizadas con las cartas Pokémon, pero no con este tipo de aplicaciones y dos personas que no están familiarizadas con las cartas Pokémon ni con este tipo de aplicaciones.

La evaluación de usabilidad se ha llevado a cabo siguiendo el estándar de pruebas denominado SUS, que son las siglas en inglés de "System Usability Scale" (Escala de Usabilidad del Sistema) [21]. Este método es una herramienta sencilla y valiosa tanto para los usuarios como para los desarrolladores, permitiendo una evaluación exhaustiva del rendimiento de la aplicación. El

cuestionario que los usuarios deben completar después de probar la aplicación consta de diez preguntas, con cinco opciones de respuesta posibles. El contenido del cuestionario es el siguiente:

Preguntas:

1. Tengo la impresión de que querría usar esta aplicación de forma regular.
2. Me parece que la aplicación es demasiado complicada.
3. Creo que es fácil utilizar esta aplicación.
4. Siento que no podría usar esta aplicación sin ayuda de otra persona.
5. Opino que las funciones en esta aplicación están integradas de manera adecuada.
6. Encuentro una notable inconsistencia en la aplicación.
7. Creo que la mayoría de las personas aprenderían a usar la aplicación en poco tiempo.
8. La aplicación me resultó bastante molesta de usar.
9. Me sentí muy seguro al usar la aplicación.
10. Fue necesario aprender muchas cosas antes de poder comenzar con esta aplicación.

Respuestas posibles:

1. Nada de acuerdo
2. En desacuerdo
3. Indiferente
4. De acuerdo
5. Muy de acuerdo

A continuación, se va a realizar la valoración de los resultados obtenidos. Estos resultados se calculan de acuerdo con el siguiente estándar:

1. Sumar las respuestas de los enunciados impares y restarle 5.
2. Sumar las respuestas de los enunciados pares y restárselo a 25.
3. Sumar ambos resultados previos y multiplicar por 2,5.

Tras obtener los resultados de todos los encuestados, se ha calculado que el valor obtenido se corresponde con 85 sobre 100. La media de estas pruebas se sitúa en el valor de 68, por lo que se podría concluir que las pruebas han sido incluso más satisfactorias de lo esperado.

Además de estas pruebas, se pidió a los encuestados que rellenar un apartado con los problemas que habían encontrado y algunas sugerencias que creen que pudieran venir bien de cara al desarrollo de la aplicación. Quiero destacar algunas de estas propuestas que me han parecido muy interesantes y que se estudiará de incorporar en el futuro:

- Añadir en el apartido "Mi colección" la opción de poder elegir si quieres que se muestren todas las cartas o solo las cartas de tu colección en los distintos filtros.
- Colores identificativos de los logros, por ejemplo, si un logro es de tipo planta que sea de tonalidad verde.
- En las pantallas paginadas, añadir la posibilidad de moverte entre páginas a través de un deslizamiento lateral, además de los botones ya implementados.
- Adaptar el diseño de la aplicación para el modo oscuro.
- Mensajes de error más descriptivos.
- Añadir algún tipo de mensaje en la pantalla si tu *Wishlist* está vacía.
- Añadir gráficos en las estadísticas.
- Botón Home.
- Al editar la cantidad de una carta, solo debería de actualizarse ese valor, y no toda la pantalla.
- Dialogo de confirmación al cerrar sesión.
- Añadir el filtrado de cartas al apartado "Colección".
- Botones de la pantalla de cartas pocos intuitivos.

# 7

# Conclusiones y líneas futuras

Para finalizar en este último apartado se van a realizar unas reflexiones finales sobre el trabajo realizado y el resultado obtenido finalmente, además de pensar de cara al futuro que se podría realizar para mejorar el producto final que se ha obtenido en el desarrollo actual.

## 7.1 Resultados obtenidos

Como resultado de la realización de este Trabajo de Fin de Grado, se ha logrado crear una aplicación móvil de gestión de cartas Pokémon, se han diseñado e implementado funcionalidades específicas para el género de colección, tratando de conservar las ideas originales que surgieron al comienzo de realizar este proyecto. Todo esto se ha logrado siguiendo la metodología establecida inicialmente y realizando iteraciones según se consideraron necesario, con el objetivo de lograr un sistema final lo más completo posible.

Respecto a la propia aplicación y su facilidad de uso, los resultados obtenidos cumplen con las expectativas iniciales. Las interfaces de usuario han sido diseñadas de forma atractiva para aquellos interesados en el universo Pokémon, y las características de la aplicación se integran de manera coherente con el propósito central del juego. Se ha evidenciado que la aplicación satisface

criterios exigentes en cuanto a su usabilidad, proporcionando una experiencia del usuario que cumple con las expectativas.

En resumen, los resultados alcanzados en este proyecto se consideran satisfactorios en todos los aspectos, ofreciendo a los usuarios una plataforma efectiva y atractiva para gestionar sus colecciones y cartas Pokémon, así como para participar en estadísticas, logros, rankings y en la función de tasación de intercambios.

## 7.2 Conocimientos adquiridos

El desarrollo de este proyecto me ha permitido afianzar conocimientos previos que podía tener con Python y sobre todo con Java, ya que con este último lenguaje si he trabajado con mayor frecuencia, mientras que con el primero tenía ideas más básicas de haberlo usado en algún proyecto anterior del grado. Sin embargo, nunca había realizado un proyecto de tal complejidad y de forma individual lo que ha hecho que tenga que enfrentarme a problemas que no me habían surgido con anterioridad y tener que enfrentarlos solo, pero que aun así he podido realizar obteniendo un resultado satisfactorio.

Otra cosa que he aprendido en el transcurso del proyecto ha sido a conectar las distintas partes de la arquitectura, es decir a tener un cliente conectado a un servidor, y que este servidor a su vez esté conectado a una base de datos, y a una API externa, ya que anteriormente solo había trabajado con ellas de forma separada pero no en conjunto como ahora.

Y, por último, he podido aplicar los conocimientos adquiridos en asignaturas anteriores como ha podido ser Análisis y Diseño de Algoritmos para la realización de mi propio algoritmo heurístico de búsqueda para las sugerencias de los intercambios, aunque es cierto que ha sido algo más sencillo de lo que planteaba inicialmente, pero se ha optado por optimizar el tiempo de búsqueda. Igualmente, esto me ha servido para reforzar y aplicar esos conocimientos previos.

## 7.3 Dificultades encontradas

El tener que trabajar con tecnologías que no se habían utilizado previamente, el conseguir esa conexión entre las distintas partes del proyecto ha sido una dificultad, no muy problemática, ya que al final esto ha hecho que se haya podido aprender cosas nuevas, que realmente es el objetivo final de este trabajo.

La mayor dificultad encontrada en el desarrollo de este proyecto ha sido el de una API externa con la conseguir toda la información de las colecciones y las cartas Pokémon, ya que en un inicio se planteó utilizar una de dos opciones que eran las más tentadoras, ya que por su documentación se podía ver que eran muy completas y podía permitir que se realizasen todas las funcionalidades planteadas al inicio del proyecto, pero el problema surge que cuando se solicita el acceso a estas APIs se nos deniega el acceso ya que dejaron de ser de acceso público recientemente, Debido a este se tuvo que buscar otra API que si fuese pública y que nos permitiera realizar las funcionalidades planteadas. Finalmente se consiguió una, que ha sido la que se ha utilizado, que no en su totalidad, pero sí nos ha permitido cumplir con la mayoría de los requisitos y funcionalidades que se querían realizar en la aplicación.

Otro problema relacionado con el anterior es la lentitud de algunas solicitudes a esa API externa. Cuando esa solicitud su respuesta era de un gran número de cartas, esta solicitud se realizaba de forma muy lenta, provocaba una degradación muy importante de la experiencia del usuario al usar la aplicación. Finalmente, se optó por realizar consultas paginadas, y así acelerar estos tiempos de espera y reducirlos casi a cero.

La última dificultad encontrada en el desarrollo de este proyecto ha sido el despliegue, desde un inicio se quería realizar a través de la plataforma Heroku, pero como esta pasó a ser de pago buscaron alternativas que permitieran un despliegue de un servidor de las características de este proyecto. Finalmente, y tras intentarlo con distintas herramientas y no conseguirlo con ninguna por

problemas de compatibilidad, se decidió desplegarlo con Heroku igualmente y así acabar con esta problemática.

## 7.4 Líneas futuras

Para el futuro hay ciertos aspectos de la aplicación que se podrían mejorar, y que no se ha podido hacer ahora por la limitación de tiempo. A continuación, se comentarán algunas de las posibles propuestas a implementar en el futuro.

Una de las primeras propuestas sería intentar de nuevo conseguir una de las APIs mencionadas anteriormente, que son más completas que la que se ha podido utilizar, lo cual haría que la información que se le pueda dar al usuario sobre las cartas sea mayor que la que se puede obtener actualmente.

Otra idea sería la de utilizar algún tipo de caché que pudiese guardar las cartas durante el uso de la aplicación y el tratamiento asíncrono de las peticiones para así poder reducir el tiempo de espera para el usuario.

Por último, se considera que se podría mejorar mucho la interfaz gráfica de todas las pantallas, pudiendo conseguir un acabado mucho más profesional y hacer que sea una propuesta visual mucho más atractiva para los usuarios.

Una vez que se haya conseguido realizar estas optimizaciones en la aplicación se quiere subir a Google Play para que la aplicación esté abierta al público.

# Referencias


[1] Wikipedia, “Juego de cartas coleccionables Pokémon”, URL: https://es.wikipedia.org/wiki/Juego_de_cartas_coleccionables_Pok%C3%A9mon#:~:text=La%20primera%20edici%C3%B3n%20del%20juego,en%20Estados%20Unidos%20y%20Europa. Último acceso: 25/08/2023

[2] Nintenderos, “Conoce las cartas de Pokémon más caras que existen”, URL: https://www.nintenderos.com/2022/11/conocelas-cartas-de-pokemon-mas-caras-que-existen/ Último acceso: 25/08/2023

[3] Youtube, “PokeRev”, URL: https://www.youtube.com/@PokeRev/featured Último acceso: 25/08/2023

[4] European Grading, “European Grading”, URL: https://europeangrading.com/es/ Último acceso: 25/08/2023

[5] El Output, “Vender cartas Pokémon como medio de vida: Gana 345.000 dólares en un año” URL: https://eloutput.com/noticias/cultura-geek/thomas-kovacs-cartas-pokemon-especulacion/ Último acceso: 25/08/2023

[6] Scrum.org, “The Home of Scrum”, URL: https://www.scrum.org/resources/what-is-scrum Último acceso: 27/08/2023

[7] Python.org, URL: https://www.python.org/ Último acceso: 27/08/2023

[8] Flask, URL: https://flask.palletsprojects.com/en/2.3.x/ Último acceso: 27/08/2023

[9] Java, URL: https://www.java.com/es/ Último acceso: 2708/2023

[10] MongoDB, https://www.mongodb.com/es Último acceso: 28/08/2023

[11] MongoDB Atlas, URL: https://www.mongodb.com/atlas/database Último acceso: 28/08/2023

[12] Volley Android, URL: https://developer.android.com/training/volley?hl=es-419 Último acceso: 27/08/2023

[13] Picasso Android, URL: https://desarrollador-android.com/librerias/square/picasso/ Último acceso: 27/08/2023

[14] Heroku, URL: https://www.heroku.com/ Último acceso: 31/08/2023

[15] Firebase, URL: https://firebase.google.com/?hl=es-419 Último acceso: 31/08/2023

[16] Visual Studio Code, URL: https://es.wikipedia.org/wiki/Visual_Studio_Code Último acceso: 28/08/2023

[17] Android Studio, URL: https://developer.android.com/studio Último acceso: 27/08/2023

[18] Potsman, URL: https://www.postman.com/product/what-is-postman/ Último acceso: 29/08/2023

[19] Pokemon TCG API, URL: https://docs.pokemontcg.io/ Último acceso: 31/08/2023

[20] Cardmarket, URL: https://www.cardmarket.com/es/Pokemon Último acceso: 31/08/2023

[21] Jeff Sauro, “Measuring Usability with the System Usability Scale (SUS)”, URL: https://measuringu.com/sus/ Último acceso: 31/08/2023

# Apéndice A

# Manual de usuario

En esta sección, se presenta el manual de usuario que proporciona directrices detalladas sobre cómo utilizar la aplicación desarrollada. El manual se organiza de acuerdo con las funciones clave de la aplicación y ofrece imágenes del juego que acompañan las instrucciones paso a paso para una comprensión más clara.

## A.1. Registro e inicio de sesión

En este primer apartado vamos a mostrar las primeras pantallas que ve el usuario cuando inicia la sesión. Nada más entrar en la aplicación verá la pantalla de inicio, en la cual podrá elegir si iniciar sesión o registrase si aún no lo ha hecho (ver Figura 26).

Si el usuario selecciona el botón de registro, accederá a la pantalla para poder registrarse, en ella deberá de introducir, los datos que aparecen en la Figura 27. Si no hay ningún problema con los datos, el usuario se registrará correctamente, en caso contrario, si el nombre de usuario o el correo ya está registrado o si las contraseñas no coinciden se mostrará por pantalla un mensaje de error, como se puede ver en la Figura 28, y el usuario no se registrará.

Si el usuario selecciona el botón para iniciar sesión, avanzará a la pantalla de inicio de sesión donde tendrá que introducir su correo y contraseña como se puede ver en la Figura 29. Si los datos son correctos accederá a su cuenta, y en caso contrario se mostrará un mensaje de error como el de la Figura 30.


*Figura 26. Manual de usuario. Inicio*


*Figura 27. Manual de usuario. Registro*


*Figura 28. Manual de usuario. Registro fallido*


*Figura 29. Manual de usuario. Inicio de sesión*

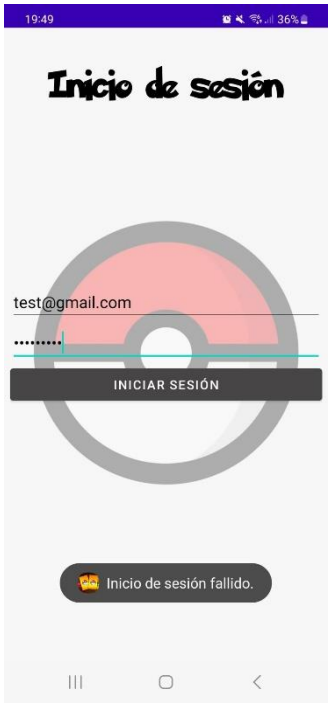



*Figura 30. Manual de usuario. Inicio de sesión fallido.*

## A.2. Menú principal

Una vez que el usuario haya iniciado sesión correctamente accederá a la pantalla del menú principal (Figura 31). En esta pantalla existen ocho botones distintos, siete de ellos corresponden a distintas funcionalidades de la aplicación, y un último botón que es para cerrar sesión. En los distintos apartados se explicará con detalle las distintas secciones a las que dirige al usuario cada uno de los botones del menú principal.


*Figura 31. Manual de usuario. Menú principal*

## A.3. Colecciones

Al acceder a la pantalla Colecciones, se mostrará un *spinner* con un listado de todas las colecciones existentes, por defecto al entrar en esta sección se selecciona la colección "Base" y se muestran sus cartas tal y como se puede observar en la Figura 32. El usuario podrá seleccionar cualquiera de las cartas que aparecen y se les mostrará la pantalla de la carta (ver Figura 33)

En esta pantalla aparece la carta ampliada de tamaño, junto a sus datos y tres botones. El primer botón lleva al usuario a la página web de la carta en *Cardmarket* por si desea comprarla (Figura 34). El segundo botón es para añadir esta carta a su colección, donde le aparecerá un recuadro para indicar cuantas cartas de esta carta quiere añadir (Figura 35), y el último botón es para añadirlo al su *wishlist*, donde aparecerá un recuadro en el podrá elegir a qué precio quiere que se le avise por correo electrónico (Figura 36).


*Figura 32. Manual de usuario. Colecciones*


*Figura 33. Manual de usuario. Carta Colecciones*


*Figura 34. Manual de usuario. Cardmarket*

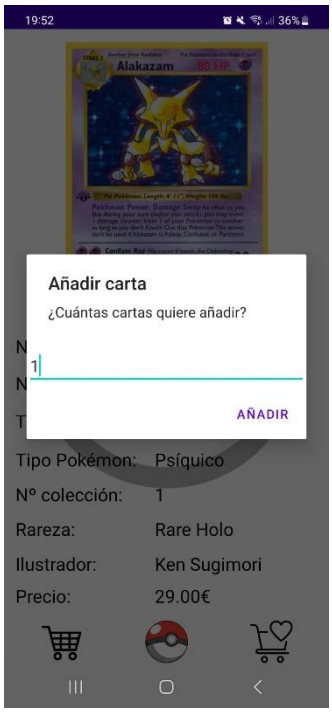



*Figura 35. Manual de usuario. Añadir carta*

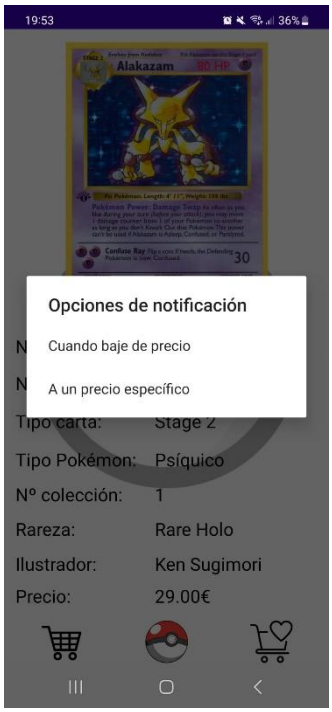



*Figura 36. Manual de usuario. Añadir Wishlist*

## A.4. Mi colección

En la pantalla Mi colección el usuario podrá visualizar una pantalla muy similar a la de Colecciones (Figura 37), pero con sus diferencias. La primera de ella es que solo aparecerán a color las cartas que el usuario tenga en su colección, las que no aparecerán en blanco y negro. Además, estas últimas cartas no pueden ser seleccionadas, solo son seleccionables las cartas que pertenecen a la colección del usuario, y al seleccionarlas se mostrarán sus datos (Figura 38) y podrá editar la cantidad si así lo desea el usuario, si la cantidad es 0, se eliminará de la colección del usuario. La segunda diferencia son los filtros, el usuario dispondrá de distintos filtros para poder mostrar las cartas según su conveniencia. Los filtros disponibles son los que se ven en la Figura 39.


*Figura 37. Manual de usuario. Mi colección*


*Figura 38. Manual de usuario. Carta mi colección*


*Figura 39. Manual de usuario. Filtros*

## A.5. Wishlist

En esta sección el usuario podrá observar el listado de las cartas que ha añadido a su *wishlist* personal (Figura 40), pudiendo seleccionar cada una de las cartas en las que podrá ver los datos de la carta, podrá editar su precio deseado, y además tendrá tres botones como la carta en la pantalla colecciones, con la variación del tercer botón que en este caso sirve para eliminar la carta de la *wishlist* (Figura 41).


*Figura 40. Manual de usuario. Wishlist*


*Figura 41. Manual de usuario. Carta wishlist*

## A.6. Estadísticas

Para este apartado el usuario podrá visualizar los cuatro datos principales sobre su colección, que son los datos de carta más valiosa, número total de cartas, número total de cartas únicas y el valor total de la colección. Además de esto, como podemos ver en la Figura 42 hay un botón para ver más estadísticas, que, al presionarlo el usuario será enviado a otra pantalla donde se mostrarán otros datos sobre su colección (Figura 43).


*Figura 42. Manual de usuario. Estadísticas*


*Figura 43. Manual de usuario. Más estadísticas*

## A.7. Logros y Ranking

En estas dos funcionalidades, primero el usuario podrá acceder a la sección de logros, en la cual podrá ver el listado de logros existentes (Figura 44), pudiendo ver en ellas el nombre, una descripción y la puntuación que recibiría al completar cada logro. Además, podrá ver que logros ha completado ya, ya que estos se distinguen porque están de color verde. Cuando un usuario al añadir una carta completa un logro, se mostrará un mensaje por pantalla indicando que ha completado un logro y el nombre del logro completado (Figura 45). Luego en la sección del ranking, el usuario podrá ver en que puesto de la clasificación se encuentra respecto al resto de los usuarios registrados en la aplicación. En la Figura 46 se puede ver como se muestra la clasificación, destacando el nombre del usuario que ha iniciado sesión respecto de los demás usuarios registrados.

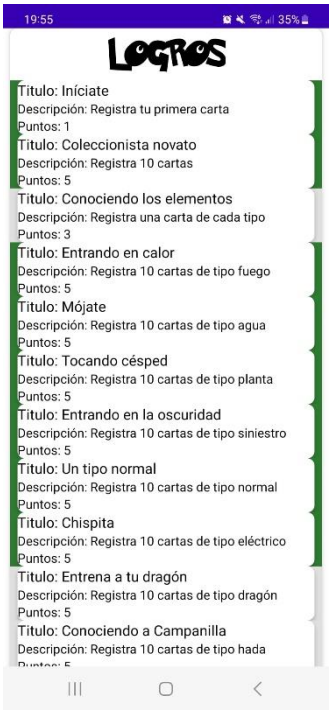



*Figura 44. Manual de usuario. Logros*


*Figura 45. Manual de usuario. Logro completado*


*Figura 46. Manual de usuario. Ranking*

## A.8. Tasador

En esta última funcionalidad, y la más compleja de la aplicación, al pulsar en ella, el usuario accederá a una primera pantalla en la que podrá leer información de cómo funcionará el tasador (Figura 47). Una vez que lo haya leído y entienda su funcionamiento pulsará el botón iniciar y comenzará el procedimiento del tasador. Primero verá una pantalla intermedia (Figura 48), en la que tendrá que indicar si sabe las cartas que va a entregar en el intercambio, si es que no, pues se pasará directamente a la pantalla de cartas a recibir (Figura 52). Esta pantalla tiene una interfaz muy similar al de la Figura 37, solo que en este caso si se pueden pulsar todas las cartas. En el caso de que haya seleccionado sí, pasará a la pantalla de cartas a dar (Figura 49). En esta pantalla el usuario podrá observar todas las cartas que pertenecen a su colección, al pulsar sobre ellas se mostrará un recuadro indicando cuantas cartas tiene en su posesión y el usuario podrá indicar cuantas quiere dar (Figura 50). En el caso de que el usuario indique que quiere dar todas las cartas, se mostrará otro mensaje informando al usuario para que confirme si es lo que quiere hacer (Figura 51).

Una vez que el usuario ya ha seleccionado todas las cartas que va a dar, debe pulsar el botón Finalizar y pasará a una segunda pantalla intermedia, exactamente igual que la primera, pero en este caso es sobre las cartas a recibir. Si pulsa sí, pasará a la pantalla de la Figura 52 y tendrá que seleccionar las cartas que va a recibir. En el caso de que pulse no, aparecerá una pantalla (Figura 53) en la que se informa al usuario de los dos tipos de búsqueda posibles, y dos botones para que el usuario elija.

Una vez hecho todo esto pasaremos a la pantalla final, en la que se mostrarán las cartas que va a dar, las cartas que va a recibir, pudiendo deslizar lateralmente el listado de cartas para verlas al completo, y debajo de cada carta aparece la cantidad de cartas que dará o recibirá. Debajo de cada listado de cartas aparece el valor total, y finalmente en la parte inferior se muestra un mensaje que indica si el intercambio que se va a realizar es justo o no. Todo esto se puede ver en la Figura 54.

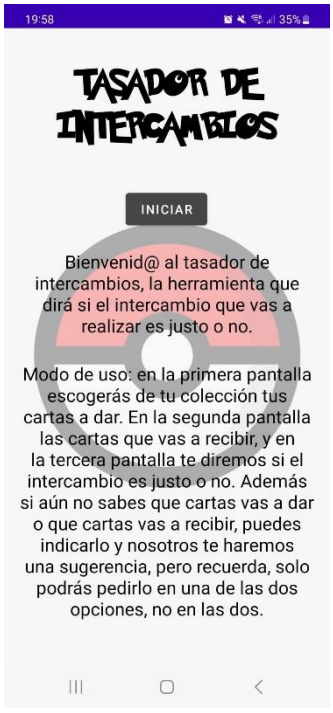



*Figura 47. Manual de usuario. Tasador de intercambios*


*Figura 48. Manual de usuario. Pantalla intermedia*


*Figura 49. Manual de usuario. Cartas que dar*

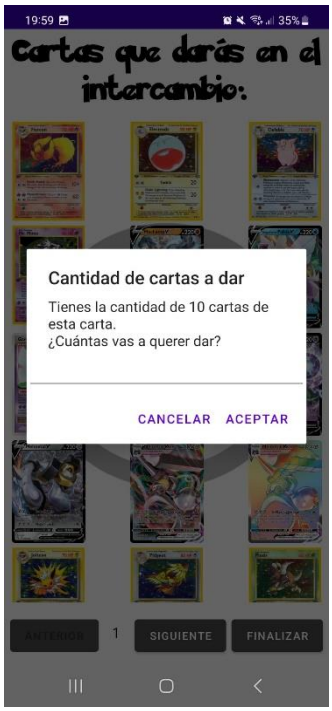



*Figura 50. Manual de usuario. Cantidad de cartas a dar*


*Figura 51. Manual de usuario. Mensaje de aviso*


*Figura 14 Manual de usuario. Cartas que recibir*

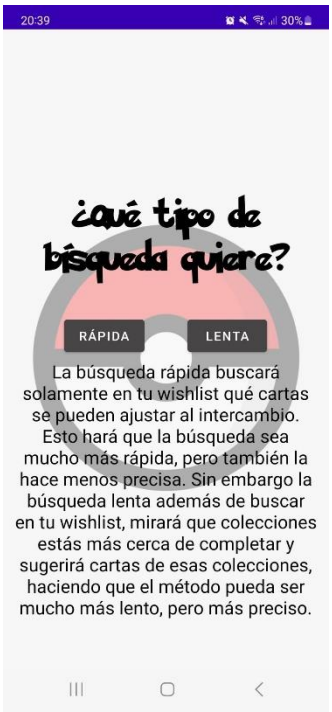



*Figura 53. Manual de usuario. Tipo de búsqueda*


*Figura 54. Manual de usuario. Valoración final*

# Apéndice B

# Manual de despliegue

A continuación, se detallan las guías para iniciar el sistema, tanto para poner en funcionamiento el servidor en la plataforma Heroku como para crear la aplicación cliente mediante el uso de Android Studio.

## B.1. Despliegue del servidor

Tal y como se ha explicado previamente en el Apartado 5.7, para comenzar con el despliegue del servidor lo primero que se hizo fue subir el proyecto del servidor a un repositorio de GitHub. Para ello se creó un repositorio, y se subieron los dos archivos que forman el servidor: `app.py` y `requirements.txt`. Además de estos dos archivos se añadió un tercer archivo `Procfile`, este archivo se trata de un archivo específico utilizado por Heroku para el despliegue. Todo esto se puede ver en la Figura 24 mostrada previamente.

Antes de realizar el despliegue se debe comprobar que los archivos están actualizados, para ello se podrá realizar un *commit* del proyecto, o como en este caso se trata de un proyecto muy sencillo en cuanto cantidad de archivos, se pueden subir directamente al repositorio desde GitHub.

Una vez que el código del repositorio está actualizado, se accede a Heroku iniciando sesión con la cuenta personal. Actualmente es de pago, por lo que habrá que elegir el plan de pago que se quiere utilizar, en este caso se usará el plan básico. Una vez elegido esto podemos pasar a crear una aplicación, se elige un nombre, un servidor y en caso de que se desee, un pipeline y se crea. (Figura 55).

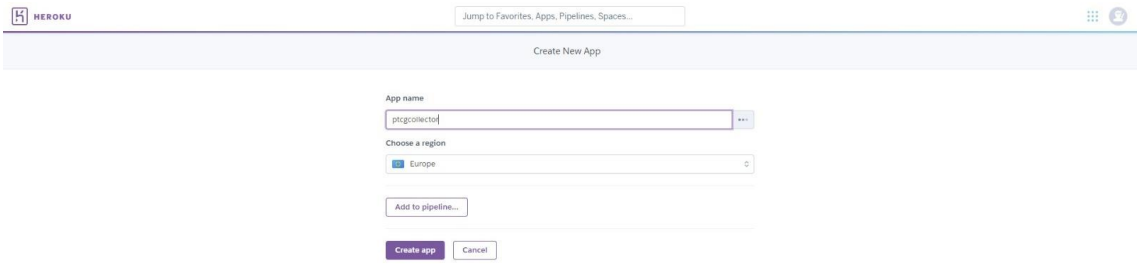



*Figura 55. Manual de despliegue. Creación de aplicación*

Tras tener la aplicación creada, veremos la pantalla para realizar el despliegue, se conecta a un pipeline si no se ha hecho previamente al crear la aplicación, si así se desea, y posteriormente se realiza la conexión con GitHub (Figura 56). Otra opción habría sería usando Heroku CLI, pero se optó por la conexión con un repositorio GitHub. Una vez realizada esta conexión, se elige el repositorio que se quiere desplegar y por último se elige la rama a desplegar y pulsamos el botón para realizar el despliegue. Adicionalmente, se puede seleccionar la opción de que se despliegue automáticamente el servidor al detectar Heroku cualquier cambio en la rama seleccionada, para este despliegue se ha decidido activar esta opción, tal y como se puede ver en la Figura 57.

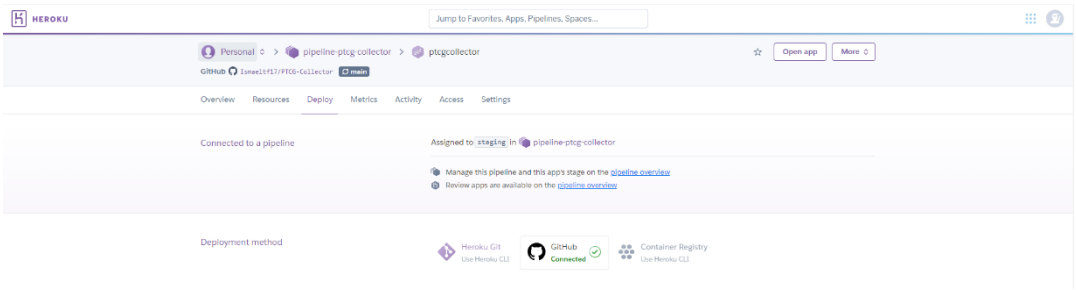



*Figura 56. Manual de despliegue. Conexión con GitHub*

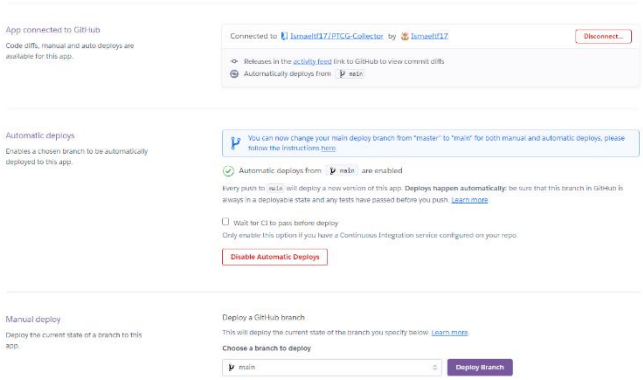


*Figura 57. Manual de despliegue. Despliegue del servidor*

Una vez que se han realizado estos pasos, ya se encuentra desplegado el servidor en la URL: https://ptcgcollector-68d462cc0459.herokuapp.com/ donde se encuentran alojados los servicios REST que la aplicación móvil ofrece.

## B.2. Generación aplicación móvil

En lo que respecta a la aplicación móvil creada en Android, ya se ha generado un archivo APK que contiene la aplicación completa. Sin embargo, si es necesario, también es posible generar nuevamente este archivo desde Android Studio. Para hacerlo, primero debes abrir el proyecto desarrollado en Android Studio, se selecciona la opción de “Build” en la barra de herramientas, que abrirá un desplegable con múltiples opciones, y seleccionamos “Build(s) / APK(s)” y dentro de esto la opción “Build APK(s)” (Figura 58).

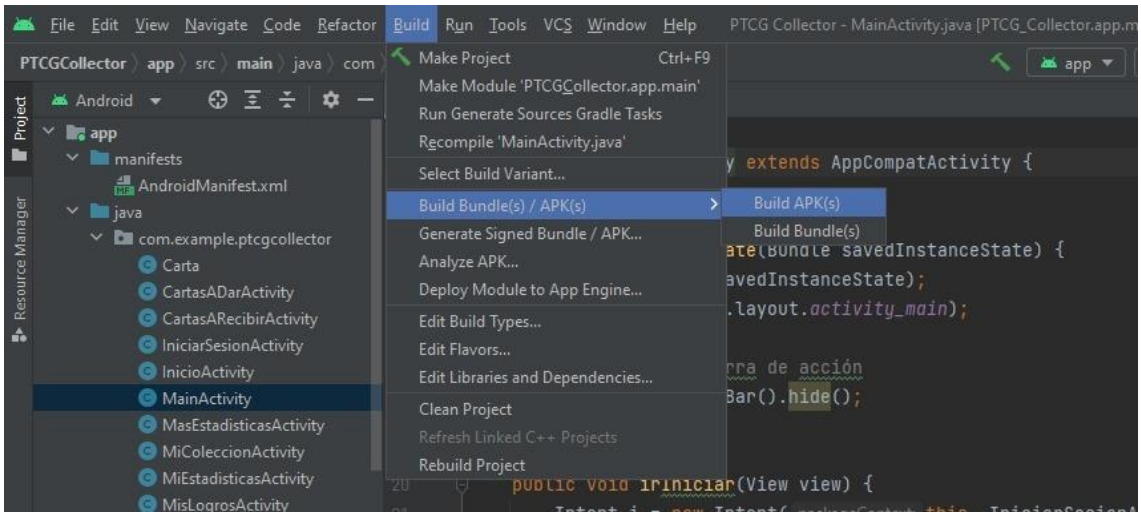



*Figura 58. Manual de despliegue. Generación del APK de la aplicación*

Además de la posibilidad de generar un APK, también existe la opción de subir la aplicación a Google Play. Como se ha comentado en el Apartado 7.4, la idea final de la aplicación será conseguir esto, pero actualmente no se ha realizado.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA