UNIVERSITAT DE BARCELONA

Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica

Universitat de Barcelona

CARDALIA: UNA APLICACIÓ WEB PER AL INTERCAMBI DE CARTES COL·LECCIONABLES

Albert Royo

Directora: Polyxeni Gkontra

Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 23 de gener de 2023

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres el gran soporte (y la paciencia) que me han dado durante los años de la carrera, especialmente en esta última etapa de realización del trabajo de fin de grado.

Por otro lado, agradezco a mis amigos Oriol y Sergi sus consejos para el diseño y desarrollo del proyecto. A Dani y a Trebla sus grandes ideas y el feedback para el diseño de la interfaz gráfica y la prueba de usabilidad de la aplicación.

Finalmente agradezco a Xenia, mi tutora del TFG su gran apoyo, implicación y consejos, pues a pesar de que el desarrollo web, no es su ámbito de trabajo habitual, valoro el esfuerzo que ha realizado para ayudarme a finalizar la carrera con un proyecto de calidad.

Resumen

Magic: The Gathering es un juego de cartas coleccionables (JCC) creado en 1993 y jugado por más de 40 millones de personas en todo el mundo. El juego contiene más de 20.000 cartas con diferentes efectos, y es considerado uno de los JCC más complejos que existen.

Wizards of the Coast es la empresa que lo comercializa. Esta ofrece principalmente, como productos de venta, mazos de iniciación y sobres de cartas, siendo este último el mecanismo principal para que los jugadores/as amplíen su colección de cartas y construyan sus barajas. Por otro lado, existe un mercado no oficial de compraventa de cartas sueltas formado por las tiendas especializadas en el juego y por distintas plataformas en línea.

En ocasiones, aunque exista este mercado, los jugadores/as no tienen la capacidad de adquirir las cartas que desean, ya sea por motivos económicos o por falta de producto. Pero al mismo tiempo, disponen de una gran cantidad de cartas que no usan y son útiles en el juego. Es por este motivo que este proyecto consiste en facilitar una herramienta en línea para que los jugadores/as de Magic puedan intercambiar cartas entre sí.

El proyecto presenta Cardalia, una aplicación web para mejorar la comunidad del juego Magic: The Gathering y la disponibilidad de sus cartas. El proyecto tiene como objetivo principal crear una aplicación web eficiente, fiable y robusta que permita a los usuarios realizar intercambios de cartas de Magic mediante una interfaz sencilla e intuitiva.

La aplicación basa su arquitectura en el estilo API REST. Funciona con dos programas distintos: CardaliaAPI, el backend de la aplicación programado en Go, encargado de guardar los datos de los usuarios y obtener la información de las cartas; y CardaliaWEB, el frontend programado con React, framework basado en javascript, encargado de facilitar una interfaz gráfica en línea que permita a los usuarios de la web intercambiar cartas entre sí.

Resum

Magic: The Gathering és un joc de cartes col·leccionables (JCC) creat al 1993 i jugat per més de 40 milions de persones a tot el món. El joc, disposa de més de 20.000 cartes amb diferents efectes, i és considerat un dels JCC més complexes que existeixen.

Wizards of the Coast és l'empresa que el comercialitza. Aquesta ofereix principalment, com a productes de venda, baralles d'iniciació i sobres de cartes, sent aquest últim el mecanisme principal perquè els jugadors/as ampliïn la seva col·lecció de cartes i construeixin les seves baralles. D'altra banda, existeix un mercat no oficial de compravenda de cartes soltes format per les botigues especialitzades en el joc i per diferents plataformes en línia.

A vegades, encara que existeixi aquest mercat, els jugadors/as no tenen la capacitat d'adquirir les cartes que desitgen, ja sigui per motius econòmics o per falta de producte. Però al mateix temps, disposen d'una gran quantitat de cartes que no utilitzen i són útils en el joc. És per aquest motiu que aquest projecte consisteix a facilitar una eina en línia perquè els jugadors/as de Magic puguin intercanviar cartes amb altres jugadors.

El projecte presenta Cardalia, una aplicació web per a millorar la comunitat del joc Magic: The Gathering i la disponibilitat de les seves cartes. El projecte té com a objectiu principal crear una aplicació web eficient, fiable i robusta que permeti als usuaris realitzar intercanvis de cartes de Magic mitjançant una interfície senzilla i intuïtiva.

L'aplicació basa la seva arquitectura en l'estil API REST. Funciona amb dos programes diferents: CardaliaAPI, el backend de l'aplicació programat en Go, encarregat de guardar les dades dels usuaris i obtenir la informació de les cartes; i CardaliaWEB, el frontend programat amb React, framework basat en javascript, encarregat de facilitar una interfície gràfica en línia que permeti als usuaris intercanviar cartes entre ells.

Abstarct

Magic: The Gathering is a collectable card game (CCG) created in 1993 and played by more than 40 million people worldwide. It has over 20,000 cards with different effects on the game, making it one of the most complex CCGs in the world.

Wizards of the Coast is the company that markets it. It offers mainly, as sale products, initiation decks and card envelopes, the latter being the main mechanism for players to expand their card collection and build their decks. On the other hand, there is an unofficial market for the sale of loose cards including the shops specializing in the game and various online platforms.

Sometimes, even if there is such a market, players do not have the ability to purchase the cards they want, either for economic reasons or due to a lack of product. At the same time, however, they have many cards that they do not use and are useful in the game. Therefore, this project consists of providing an online tool for Magic players to exchange cards with other players.

The project features Cardalia, a web application to enhance the Magic The Gathering game community and the availability of their cards. The project aims to create an efficient, reliable and robust web application that allows users to perform exchanges of Magic cards using a simple and intuitive interface.

The application bases its architecture on the REST API style. Works with two different programs: CardaliaAPI, the backend of the application programmed in Go, charged with storing user data and obtaining card information; and CardaliaWEB, the frontend programmed with React, javascript-based framework, charged with facilitating an online graphical interface that allows users to exchange cards with each other.

Índice

1. Introducción

1.1. Contextualización

Los juegos de cartas coleccionables o JCC son un estilo de juego que está cogiendo mucha fuerza en los últimos años y en mi caso, han estado presentes desde mi infancia. Después de, durante mucho de tiempo usar y ver evolucionar las herramientas que existen para comprar, vender o compartir información, he decidido crear un servicio que facilite la jugabilidad y potencie la comunidad de Magic: The Gathering.

Magic: The Gathering (Figura 1) es un juego de cartas coleccionables (JCC o CCG en inglés) diseñado en 1993 por Richard Garfield, profesor de matemáticas, y comercializado por la empresa Wizards of the Coast. Magic es el primer ejemplo de juego de cartas coleccionables moderno, con más de cuarenta millones de jugadores/as en cincuenta y dos países diferentes. Magic puede ser jugado por dos o más jugadores/as, cada uno de ellos usando un mazo individual. También existen varias versiones digitales que pueden jugarse en línea, en videoconsola o PC.

Figura 1 *Cartas de Magic: The Gathering*

En Magic los jugadores/as juegan cada uno con un conjunto personalizado de cartas, denominado baraja, o 'deck'. Como existen más de 20.000 cartas, uno de los aspectos que hace que jugar Magic sea muy divertido es que hay una gran cantidad de posibles barajas, por eso dependiendo de con qué mazo y contra qué mazo juegues, las partidas son muy diferentes entre sí, lo que lo hacen uno de los juegos más complejos del mundo [1].

Hoy en día existen múltiples herramientas en-línea muy útiles para los jugadores/as. Hay páginas web que sirven para crear tus colecciones o mazos de cartas, páginas para comprar y vender cartas, aplicaciones para jugar en línea e incluso APIs que facilitan el desarrollo de proyectos para la comunidad del juego.

1.2. Motivación

Wizards of the Coast lanza constantemente nuevas expansiones que incluyen cartas nuevas que hacen cambiar constantemente el meta-juego para la mayoría de los jugadores/as. El término "meta-juego" se refiere a las estrategias, tácticas y decisiones que se toman fuera del juego para mejorar la posibilidad de ganar. Este cambio constante, sumado a que a veces es un poco complicado adquirir las cartas que deseas para tu baraja, ya sea por motivos económicos o por falta de stock, hace que, para muchos jugadores/as, Magic acabe siendo una afición un poco cara.

Uno de los principales motivos por los que he decidido hacer el proyecto de fin de grado sobre este tema es porque desde hace tiempo utilizo de forma habitual herramientas en las que he detectado la falta de un servicio de intercambio.

Otra de las motivaciones para realizar este proyecto es el deseo de mejorar mis conocimientos en herramientas de desarrollo de software, específicamente el sector del desarrollo web.

1.3. Objetivos

El objetivo principal de este proyecto es crear una aplicación web eficiente, fiable y robusta que sea capaz de permitir a los usuarios realizar intercambios de cartas del juego de Magic: The Gathering mediante una interfaz gráfica sencilla e intuitiva.

Los objetivos específicos son los siguientes:

1.3.1. Funcionalidades de la aplicación web

- Registro de usuario.
- Iniciar sesión.
- Cerrar sesión.
- Mostrar datos de usuario.
- Modificar datos de usuario.
- Visualizar su colección.
- Modificar colección.
  - Añadir carta.
  - Modificar carta (copias, versión, condición, acabado).
  - Eliminar carta.
- Visualizar intercambios.
- Realizar oferta de intercambio.
- Modificar intercambio.

1.3.2. Ampliación de conocimientos

Otro de los objetivos específicos de este proyecto es el de ampliar y perfilar mis conocimientos en el desarrollo de software, y al mismo tiempo conseguir determinar cuál es el ámbito donde me siento más cómodo dentro del desarrollo web. Por ese motivo he decidido aprender distintos procedimientos, herramientas y lenguajes de programación que no había usado anteriormente.

1.4. Análisis de aplicaciones similares

1.4.1. Análisis de mercado

Wizards of the Coast solo vende cartas en sobres y mazos, pero existe un mercado activo de cartas sueltas entre los jugadores/as en muchas tiendas especializadas y en herramientas en línea. Aunque siempre se puede ir a comprar o vender cartas en estas tiendas, existe la

posibilidad que no dispongan de la carta que necesites o que el precio de esta sea muy elevado en caso de comprar, o que te ofrezcan un precio muy reducido en caso de vender. Por ese motivo hay muchos jugadores/as que prefieren usar las principales plataformas online para comprar y vender cartas como CardMarket.com, la más conocida a nivel europeo o Cardkindom.com, famosa en estados unidos.

CardMarket.com, con una media de 12 millones de visitas mensuales [2], es una página web alemana que proporciona una plataforma para la compraventa de cartas Magic y otros juegos y accesorios de cartas coleccionables. Los precios de las cartas dependen, en buena medida, de su utilidad, su rareza, su antigüedad y su estado físico. Estos varían mucho dependiendo de la carta en cuestión. En esta plataforma, los usuarios brindan información sobre cuántas y qué cartas poseen, en qué calidad y por cuánto quieren venderlas. A través de un sistema de listas de deseos, otros miembros pueden ver qué personas ofrecen los precios más baratos para una selección de cartas que desean. Después de realizar el pedido, esas cartas se envían por correo postal al comprador.

En un inicio pretendía hacer una página web para la compraventa de cartas coleccionables, permitiendo así a diferentes usuarios proponer ofertas para que otros, mediante un pago, las pudieran comprar. Sin embrago, tal y como ya he comentado, la compraventa es un servicio que actualmente está muy consolidado en el mercado europeo, pues existen múltiples plataformas web que la permiten.

1.4.2. Necesidad de intercambio

Existen distintos formatos de juego, en cada uno de los cuales se restringe un conjunto de cartas. La gran mayoría de jugadores/as juegan pocos de los muchos formatos que ofrece el juego, pero disponen de una gran cantidad de cartas que no usan pero que son útiles en otros formatos y por lo tanto tienen un valor notable.

Por este motivo he decidido crear un servicio que permita el intercambio de cartas en lugar de la compraventa, una alternativa que interesa a muchos jugadores/as pues estos en realidad no están interesados/as en ganar dinero sino en poder adquirir las cartas que les interesa y seguir jugando. Personalmente, por lo que he podido observar en los últimos meses, la alternativa del intercambio de cartas se está volviendo tendencia en algunas tiendas y establecimientos oficiales del juego.

Después de una investigación y de mi propia experiencia como jugador, he podido observar que a nivel europeo no existe tal servicio, aparte de algún foro especializado o página de venta de todo tipo de productos como Walapop o MilAnauncios, donde tales anuncios de intercambio son difíciles de encontrar o se pierden con el tiempo. Para superar este problema he decidido crear una página web que permitirá y que centralizará los intercambios de cartas.

2. Análisis y planificación del proyecto

2.1. Alcance

El objetivo principal del proyecto es diseñar y crear una aplicación web que permita el intercambio de cartas coleccionables. Para realizar este servicio un usuario tendrá la opción de crear un conjunto de las cartas que tiene, pero no las necesita y entonces buscar la carta que necesita.

El usuario, una vez haya encontrado otro usuario con la carta que necesita, podrá iniciar un intercambio iniciando de esta forma una oferta de intercambio.

El usuario que ha recibido una oferta de intercambio podrá seleccionar las cartas que le interesan del usuario que le ha realizado la oferta, si hay alguna. En caso de que así sea, si los dos usuarios llegan a un acuerdo se finalizar el intercambio.

La página web ofrece el servicio de buscar y encontrar usuarios para realizar intercambios. El paso final del contacto entre los usuarios y el envío de las cartas físicas no forma parte del alcance de este proyecto.

El proyecto no está enfocado en la estética de la interfaz gráfica sino en su funcionalidad y utilidad, pero se aplicarán los estilos de los distintos componentes públicos importados.

2.2. Análisis de requisitos

2.2.1. Usuarios

2.2.1.1. Estudio demográfico

Todos los jugadores y jugadoras de Magic son usuarios potenciales para la aplicación. Aunque Wizards of the Coast asegura que hay hasta un 38% de mujeres que juegan el juego [3], hay estudios que dicen que en la gran mayoría de los eventos en vivo es número se reduce [4]. La gran mayoría de jugadores/as tienen entre 19 y 25 años, aunque muchos de ellos empiezan a jugar a los 15 años y hay otros muchos entre 30 y 35 años, tal y como muestra un estudio realizado hace 7 años [5].

2.2.1.2. Actores

Entre los jugadores/as potenciales se puede distinguir dos actores distintos: los usuarios registrados y los usuarios no registrados. La única funcionalidad del usuario no registrado es registrarse.

Las funcionalidades principales de los usuarios registrados son iniciar sesión, visualizar y modificar su colección de cartas y realizar, modificar i finalizar ofertas de intercambio con otros usuarios.

2.2.2. Requisitos

He extraído los siguientes requisitos para la aplicación:

Registro / Inicio sesión

1. El usuario no registrado solo puede acceder a la página de registro y al inicio de sesión.
2. El usuario tiene tres datos: el nombre de usuario, su correo y la contraseña.
3. El usuario puede modificar su contraseña.

Inicio

Siendo la *Colección* el conjunto de cartas que dispone, pero no interesan a un usuario:

1. Al iniciar sesión, se podrán visualizar los datos del usuario y su *Colección*.
2. Los usuarios pueden visualizar y modificar sus datos de usuario.
3. Los usuarios pueden visualizar y modificar su colección.
4. Los usuarios pueden visualizar, modificar y eliminar intercambios de cartas

Intercambio

1. En caso de no disponer de una Colección, el usuario, no podrá realizar una oferta de intercambio.
2. El usuario que quiera realizar un intercambio buscará la carta que le interese en la pantalla principal y aparecerán todos los usuarios que la dispongan en su colección.
3. Un usuario podrá visualizar en la colección de otro usuario para seleccionar las cartas que le interesen y así empezar la oferta de intercambio.
4. Cualquier usuario puede tanto realizar ofertas de intercambio como recibirlas.
5. Cualquier usuario puede intercambiar una carta o conjunto de cartas.
6. El usuario que reciba una oferta de intercambio podrá seleccionar cartas del otro usuario para completar el intercambio, modificar la selección del otro usuario o cancelar la oferta de intercambio.
7. Los dos usuarios podrán indicar que quieren finalizar el intercambio.
8. Cuando los dos usuarios hayan indicado que quieren finalizar el intercambio, este aparecerá como intercambio finalizado.

2.2.2.1. Requisitos funcionales

Se presentan los siguientes requisitos funcionales:

Control de usuarios

- Registro de usuario.
- Iniciar sesión.
- Cerrar sesión.

Perfil de usuario

- Mostrar datos de usuario.
- Modificar datos de usuario.

Colección

- Visualizar su colección.
- Modificar colección.
  - Añadir carta.
  - Modificar carta (copias, versión, condición, acabado).
  - Eliminar carta.

Intercambio

- Visualizar intercambios.
- Realizar oferta de intercambio.
- Modificar intercambio.
- Cancelar intercambio.
- Aceptar intercambio (finalizarlo).

2.2.2.2. Requisitos no funcionales

Se presentan los siguientes requisitos no funcionales:

- Escalabilidad: el sistema será capaz de manejar un aumento en el tráfico o el uso.
- Disponibilidad: el sistema será accesible con los navegadores más usados mediante conexión a internet.
- Seguridad: el sistema protegerá los datos y la información del usuario de posibles ataques.
- Desempeño: el sistema realizará sus funciones de manera rápida y eficiente.
- Interoperabilidad: el sistema será capaz de trabajar con otros sistemas.
- Facilidad de uso: el sistema será fácil de usar y navegar para los usuarios.
- Mantenibilidad: el sistema será fácil de mantener y de actualizar.

2.3. Metodología usada

Agile

Actualmente existen varias metodologías a la hora de realizar un proyecto y conviene pensar cuál es la adecuada para ti. En mi caso he escogido utilizar la metodología Agile [6], ya que se adapta mucho a mi proyecto. En concreto, el proyecto está dirigido por las necesidades del mercado, es decir, no hay un cliente específico y eso permite adaptarse más rápida y cómodo a los cambios que pueden surgir y continuar con el desarrollo de este (integración continua).

Dentro de Agile hay distintos tipos de metodologías como pueden ser Scrum [7] y Kanban [8]. Son diferentes ya que Scrum contiene ciclos temporales cortos y de duración fija (sprint), además de que los roles dentro del equipo juegan un gran papel. En el inicio de cada sprint o etapa, se marcan un conjunto de tareas específicas a realizar, las cuales han sido valoradas según su dificultad. Al finalizar cada sprint, se mira si el resultado es el que se había planificado y se actúa en consecuencia.

En cambio, Kanban se basa en desarrollar y hacer entregas sincrónicas, enfocándose a un pequeño número de tareas de forma simultánea. El método Kanban consiste en crear una tabla de 3 columnas (To-Do, Doing y Done). Dentro de la tabla se introducen las historias de usuario

asociadas a grupos creados anteriormente en la columna de To-Do; una vez que se empieza la tarea pasa a la columna Doing y al finalizar se mueve al Done.

Finalmente he elegido seguir una metodología híbrida inspirada entre Kanban i Scrum, pero adaptada a una persona, ya que funciona mejor en mi caso. Haré uso de las herramientas visuales que propone el método Kanban junto con sprints propuestos por Scrum para asegurar un desarrollo continuo. He decidido adoptar sprints mensuales puesto que se trata de un trabajo individual y puesto que para el desarrollo de toda la aplicación he preferido actuar con cautela debido al hecho de tener que aprender lenguajes y herramientas nuevas.

Trello

Trello [9] es un software de administración de proyectos con interfaz web organizar proyectos.

En mi caso me ha sido de especial utilidad para llevar a cabo la metodología Kanban y tener en todo momento una buena gestión de las tareas que tenía que realizar.

La herramienta principal que me ayudará a visualizar el desarrollo y llevar el control del proyecto será Trello.

Git

Git [10] es un sistema de control de versiones distribuido que permite llevar un seguimiento de los cambios realizados en cualquier conjunto de archivos. Es ampliamente utilizado en el desarrollo de software para coordinar el trabajo entre varios programadores que trabajan en colaboración.

Aunque he trabajado yo solo en el proyecto, he usado Git i GitHub para disponer de un sistema de control de versiones para mi proyecto y para facilitar el despliegue de la aplicación.

Principios SOLID

También hay que comentar que he desarrollado toda la aplicación teniendo en cuenta los principios SOLID [11] aplicándolos siempre que fuera posible, pero priorizando la legibilidad del código.

SOLID es un conjunto popular de principios de diseño que se utilizan en el desarrollo de software orientado a objetos. SOLID es un acrónimo que significa cinco principios de diseño clave: principio de responsabilidad única, principio de sustitución de Liskov, principio de segregación de interfaces y principio de inversión de dependencias. Los cinco son comúnmente utilizados por ingenieros de software y proporcionan algunos beneficios importantes para los desarrolladores.

2.4. Recursos

React

React [12] es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página.

React intenta ayudar a los desarrolladores a construir aplicaciones que usan datos que cambian constantemente. Su objetivo es ser sencillo, declarativo y fácil de combinar. React sólo maneja

la interfaz de usuario en una aplicación; React es la vista en un contexto en el que se use el patrón MVC.

En un principio tenía pensado usar Vue [13] para desarrollar el FrontEND, debido a que es el framework (marco) que se nos enseñó en su día en la carrera. Pero, tras hacer un pequeño estudio, observé que existen frameworks más usados y mejor valorados por la comunidad.

Finalmente he decidido usar React principalmente por que junto con Angular es uno de los frameworks más usados globalmente y tiene una de las comunidades de soporte más grandes del mundo tal y como se muestra en la Figura 2[14].

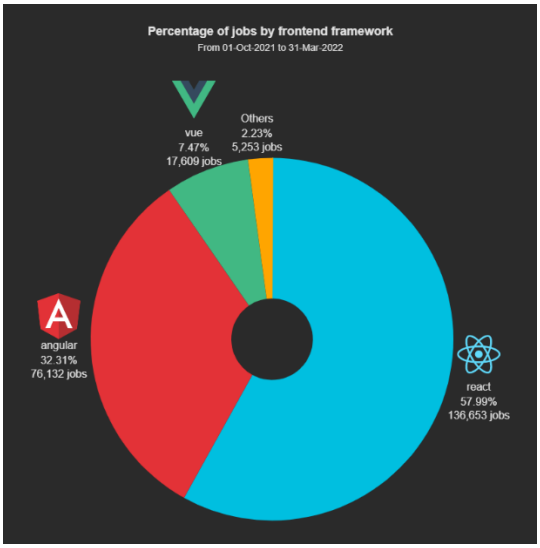

Figura 2 *Frameworks usados en puestos de trabajo de frontend*

Por otro lado, React dispone de Hooks [15]. Los Hooks son una incorporación en React 16.8. Permiten usar estado y otras características de React sin la necesidad de escribir una clase.

El lenguaje, y específicamente el uso de Hooks, me han sido de gran utilidad para obtener una aplicación escalada i entendible, pero al mismo tiempo funcional i optimizada.

Material-UI

Material UI [16] es una librería de componentes de React.js, de código abierto creada por Google, basada en Material Design, la cual nos brinda una serie de pautas y lineamientos que sientan la base para crear diseños profesionales para aplicaciones web en muy poco tiempo.

A parte de usar sus componentes para representar listas, tipografías, botones y muchos componentes más de mi aplicación, ha sido de especial utilidad el componente DataGrid de MUI X [17]. Este componente sirve para representar listados de datos con múltiples parámetros y permite ocultar una o varias columnas y ordenar su contenido según una columna.

Go

Go [18] es un lenguaje de programación desarrollado por Google concurrente y compilado con tipado estático inspirado en la sintaxis de C, pero con seguridad de memoria y recolección de basura.

De la misma forma que con React, al principio del proyecto tenía dudas de que lenguaje de programación usar para el backend. Estuve a punto de decantarme por Python, pues es el lenguaje que más he usado durante mis estudios de ingeniería informática y es uno de los que más domino. Finalmente tomé la decisión de decantarme por Go. Ello fue debido a que como parte del objetivo de este proyecto es el de ampliar mis conocimientos de programación, aunque Go no sea de los lenguajes más usados en el desarrollo web, me dispuse a aprenderlo para desarrollar la parte del backend de la aplicación.

Visual Studio Code

Visual Studio Code [19] es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.

He decidido usarlo porqué es el editor de código que más he usado durante toda la carrera y conozco bien su funcionamiento además que lo tengo personalizado a mi gusto.

MySQL

MySQL [20] es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base de datos de código abierto más popular del mundo.

He decidido usar MySQL para gestionar la base de datos del proyecto porque necesitaba una base de datos relacional y además por su sencillez y gran soporte por parte de la comunidad.

Heroku

Heroku [21] es una plataforma como servicio (PaaS) de computación en la nube, que fue desarrollada desde junio de 2007, con el objetivo de soportar solamente el lenguaje de programación Ruby, pero posteriormente se ha extendido el soporte a Java, Node.js, Scala, Python, PHP, Go y Clojure.

Las aplicaciones se corren desde un servidor Heroku usando Heroku DNS Server para apuntar al dominio de la aplicación. El servidor Git de Heroku maneja los repositorios de las aplicaciones que son subidas por los usuarios.

Los Dynos son piezas fundamentales del modelo de arquitectura de Heroku, son las unidades que proveen capacidad de cómputo dentro de la plataforma. Están basados en Contenedores Linux.

Todo y que hoy en día existen mejores alternativas, también gratuitas, a Heroku, he decidido usarlo porque ya tengo experiencia y además dispone de un gran soporte por parte de la comunidad.

Postman

Postman [22] es una plataforma de API para que los desarrolladores diseñen, construyan, prueben e iteren sus API. En abril de 2022, Postman constituye el centro de API público más grande del mundo.

Postman me ha sido de gran utilidad para probar los distintos endpoints de la aplicación a medida que iba desarrollando el proyecto.

API ScyFall

Scryfall es un motor de búsqueda de cartas de Magic. El sitio web Scryfall.com se introdujo en la World Wide Web en octubre de 2016.

A parte, Scryfall proporciona una API [23] similar a REST [24] para consultar los datos de las cartas mediante peticiones http. La API expone la información disponible del sitio web base en formatos fáciles de consumir.

Aunque existe una API oficial de Magic, he podido comprobar personalmente que Scryfall funciona mucho mejor. Es más intuitiva, más rápida, más completa y tiene una mejor documentación para desarrolladores.

2.5. Planificación temporal

2.5.1. Etapas

Aunque normalmente las tareas que se realizan en un sprint (etapa) se deciden en función de lo que se ha hecho en la etapa anterior, he diferenciado entre las siguientes etapas del proyecto:

- Etapa 1: Análisis de requisitos y diseño del proyecto.

- Etapa 2: En esta segunda etapa me he focalizado en tener un producto inicial funcional con las mínimas funcionalidades posibles, que permita tanto la búsqueda de cartas como la creación de una colección propia. Las tareas son las siguientes:
    - Implementar búsqueda de cartas
    - Implementar creación y modificación de la colección

- Etapa 3: La tercera etapa la he orientado a la gestión de usuarios, es decir, establecer una base de datos donde se guarde la información de cada usuario y crear paralelamente el conjunto de pantallas y métodos necesarios para poder facilitar esta gestión de usuarios. Por otro lado, esta etapa también se encarga de la puesta en producción de la página web, es decir el despliegue. Las tareas son las siguientes:
    - Implementación de usuarios
    - Creación base de datos
    - Despliegue de la web

- Etapa 4: Esta quarta etapa ha sido principalmente para desarrollar todo lo que implica el intercambio de cartas entre usuarios. En esta etapa, como el producto ya es casi el

definitivo empezaré a realizar pruebas de usabilidad con amigos que conocen el juego (usuarios potenciales). Las tareas son las siguientes:

- Implementación intercambios
- Pruebas de usabilidad

- Etapa 5: La etapa final consiste en aplicar los cambios propuestos por los usuarios para obtener un producto final de mayor calidad y redactar la memoria del proyecto. Las tareas son las siguientes:
  - Implementación del feedback de las pruebas de usabilidad
  - Mantenimiento de la web
  - Redacción de la memoria

2.5.2. Diagrama de Gantt

En la Figura 3 se muestra el diagrama de Gantt del proyecto.

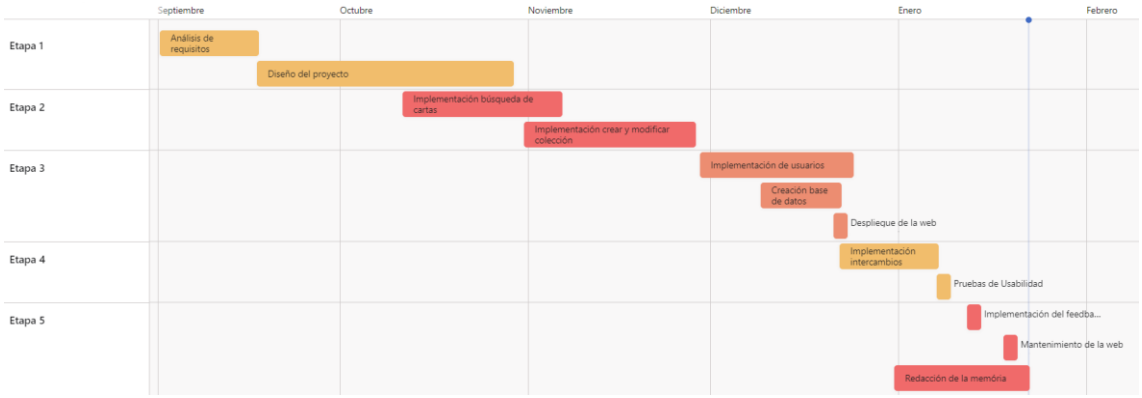

Figura 3 *Diagrama de Gantt*

3. Diseño

He decidido nombrar la aplicación con el nombre de *Cardalia*. Esta pretende crear una web sencilla pero funcional que sea capaz de permitir intercambio de cartas entre distintos usuarios. He decidido usar el inglés como idioma base para la aplicación, ya que es el más usado a nivel internacional.

3.1. Casos de uso

Registro de usuario

La Tabla 1 muestra el Caso de uso 1.

Identificador	UC1 - Registro de usuario (con datos de usuario)
Descripción	Se crea un usuario y se añade a la base de datos
Actores	Usuario no registrado, usuario registrado.
Precondición	Se ha pulsado el botón de registrarse.
Flujo principal	1. El usuario entra en el formulario de registro. 2. El usuario rellena los campos del formulario y pulsa el botón de registro. 3. La aplicación comprueba que los datos introducidos son correctos. 4. Se añaden los datos a la base de datos.
Flujo alternativo	3a. La aplicación comunica el error al usuario. 3b. El usuario modifica los datos erróneos.
Postcondición	El usuario se ha registrado correctamente.

Tabla 1 *Caso de uso 1*

Inicio de sesión

La Tabla 2 muestra el Caso de uso 2.

Identificador	UC2 - Iniciar sesión
Descripción	El usuario entra en su cuenta de usuario
Actores	Usuario registrado.
Precondición	Se ha pulsado el botón *Iniciar sesión*.
Flujo principal	1. Usuario entra en el formulario de inicio de sessión. 2. Usuario introduce su usuario y su contraseña. 3. La aplicación comprueba que los datos introducidos son correctos. 4. El ususario recibe un token de su sesión
Flujo alternativo	3a. La aplicación comunica el error al usuario. 3b. Usuario modifica los datos erróneos.
Postcondición	El usuario ha iniciado seseión correctamente.

Tabla 2 *Caso de uso 2*

Cerrar sesión

La Tabla 3 muestra el Caso de uso 3.

Identificador	UC3 - Cerrar sesión
Descripción	Usuario cierra su sesión.
Actores	Usuario registrado.
Precondición	Se ha pulsado el botón *Cerrar sesión*.
Flujo principal	1. Se cierra la sessión del usuario
Flujo alternativo	No existe
Postcondición	Se envía a UC3 - Iniciar sesión

Tabla 3 *Caso de uso 3*

Perfil de usuario

La Tabla 4 muestra el Caso de uso 4.

Identificador	UC4 - Perfil de usuario
Descripción	Visualiza el perfil de usuario.
Actores	Usuario registrado
Precondición	El usuario se encuentra en el Inicio y ha pulsado el botón *Tu perfil*.
Flujo principal	1. El usuario accede a la pagina de Perfil de usuario. 2. Se visualiza los datos del usuario 3. El usuario puede actualiza uno de sus datos personales UC6.
Flujo alternativo	No existe
Postcondición	Se ha mostrado la pagina de Perfil de usuario.

Tabla 4 *Caso de uso 4*

Actualizar datos de usuario

La Tabla 5 muestra el Caso de uso 5.

Identificador	UC5 – Actualizar datos usuario
Descripción	Actualiza los datos del usuario.
Actores	Usuario registrado
Precondición	El usuario esta en UC4 - Perfil de usuario.
Flujo principal	1. El usuario acede a la visualización de modificación de sus datos de usuario. 2. El usuario rellena los formularios de los datos que desea modificar. 3. El usuario pulsa el botón *Modificar*.
Flujo alternativo	No existe
Postcondición	Se han actualizado los datos del usuario

Tabla 5 *Caso de uso 5*

Visualizar colección

La Tabla 6 muestra el Caso de uso 6.

Identificador	UC6 – Visualizar Colección
Descripción	Visualiza la colección del usuario.
Actores	Usuario registrado
Precondición	El usuario accede a su colleción.
Flujo principal	1. El usuario visualiza su collección.
Flujo alternativo	2a. El usuario puede busca una carta UC6 – Buscar Carta para añadir.

	2b. El usuario puede modificar colección UC7 – Modificar Colección.
Postcondición	Se ha modificado la colección del usuario

Tabla 6 *Caso de uso 6*

Buscar carta para añadir

La Tabla 7 muestra el Caso de uso 7.

Identificador	UC7 – Buscar Carta para añadir
Descripción	Permite al usuario buscar una carta para añadir a la colección introduciendo su nombre.
Actores	Usuario registrado
Precondición	El usuario se encuentra en su colección .
Flujo principal	1. El usuario introduce en la barra de busqueda el nombre de la carta. 2. El usuario pulsa sobre la carta que desea de la búsqueda realizada.
Flujo alternativo	No existe
Postcondición	Se ha añadido la carta pulsada en el listado de tu colección

Tabla 7 *Caso de uso 7*

Modificar colección

La Tabla 8 muestra el Caso de uso 8.

Identificador	UC8 – Modificar Colección
Descripción	Modificar la Colección modificando sus cartas o añadiendo nuevas
Actores	Usuario registrado
Precondición	El usuario se encuentra en su colección.
Flujo principal	1. El usuario pulsa el botón de modificar carta. 2. El usuario modifica el numero de copias, la version, los extras o el estado de la carta. 3. El usuario guarda la carta modificada. 2. El usuario pulsa el botón *Guardar Colección*. 3. Se guarda en la base de datos la nueva Colección del usuario.
Flujo alternativo	1. El usuario modifica su Colección añadiendo cartas mediante la barra de búsqueda (UC7 – Buscar carta para añadir). 2. El usuario pulsa el botón *Guardar Colección*. 3. Se guarda en la base de datos la nueva Colección del usuario.
Postcondición	Se ha modificado la colección del usuario

Tabla 8 *Caso de uso 8*

Realizar oferta de intercambio

La Tabla 9 muestra el Caso de uso 9.

Identificador	UC9 - Realizar oferta de intercambio
Descripción	Realizar una oferta de intercambio a otro usuario
Actores	Usuario registrado
Precondición	El usuario se encuentra en la página de inicio
Flujo principal	1. El usuario busca la carta que desea conseguir 2. El usuario selecciona un usuario que contiene la carta buscada.

	3. El usuario accede a la colección del usuario que ha seleccionado. 4. El usuario selecciona las cartas que desea de la colección del usuario que ha seleccionado. 5. El usuario pulsa el botón *Realizar oferta*. 6. Se añade la oferta en la base de datos. 7. Se envía a UC5 – Visualizar intercambios.
Flujo alternativo	5a. El usuario pulsa el botón *Cancelar*.
Postcondición	Se a añadido el nuevo intercambio al listado de intercambios.

Tabla 9 *Caso de uso 9*

Visualizar intercambios

La Tabla 10 muestra el Caso de uso 10.

Identificador	UC10 – Visualizar intercambios
Descripción	Visualiza los intercambios pendientes y los intercambios finalizados que tengas con otros usuarios.
Actores	Usuario registrado
Precondición	El usuario accede a la pagina de Intercambios
Flujo principal	1. Se visualizan los los intercambios pendientes y los intercambios finalizados.
Flujo alternativo	2. El usuario puede escoger uno de los intercambios para visualizarlo UC10 – Visualizar intercambio.
Postcondición	Se ha mostrado la pagina de intercambios.

Tabla 10 *Caso de uso 10*

Visualizar intercambio

La Tabla 11 muestra el Caso de uso 11.

Identificador	UC11 – Visualizar intercambio
Descripción	Visualiza un intercambio con otro usuario
Actores	Usuario registrado
Precondición	El usuario ha pulsado sobre uno de los intercambios
Flujo principal	1. Se visualiza el intercambio pendiente con otro usuario. 2. El usuario puede modificar la selección de cartas tanto suya como del otro usuario UC11 - Modificar intercambio. 3. El usuario puede indicar que quiere finalizar intercambio UC12 – Finalizar intercambios. 4. El usuario puede cancelar el inercambio UC.
Flujo alternativo	1a. Se visualizan todos los intercambios finalizados con otro usuario.
Postcondición	Se ha mostrado la pagina del intercambio.

Tabla 11 *Caso de uso 11*

Modificar intercambio

La Tabla 12 muestra el Caso de uso 12.

Identificador	UC12 - Modificar intercambio

Descripción	Modificación de una oferta realizada, tanto una que haya hecho el mismo usuario u otra que le haya propuesto otro usuario.
Actores	Usuario registrado
Precondición	El usuario se encuentra en inicio. La oferta de intercambio aún no ha sido finalizada por el otro usuario
Flujo principal	1. El usuario selecciona una oferta pendiente. 2. El usuario accede al intercambio que ha seleccionado. 3. Se muestra el intercambio juntamente con el botón *Modificar intercambio*. 4. El usuario pulsa el botón *Modificar Oferta*. 5. Se actualiza la oferta en la base de datos. 6. Se envía a UC5 - Intercambios.
Flujo alternativo	4a. El usuario pulsa el botón *Cancelar* UC13 – Cancelar intercambios
Postcondición	Se ha modificado el intercambio

Tabla 12 *Caso de uso 12*

Finalizar intercambio

La Tabla 13 muestra el Caso de uso 13.

Identificador	UC13 – Finalizar intercambios
Descripción	Una vez hemos recibido la oferta de otro usuario, u otro usuario ha modificado la oferta que le habíamos realizado, hay que confirmar que los posibles cambios son correctos y finalizar el intercambio.
Actores	Usuario registrado
Precondición	Nos han hecho una oferta o han modificado una oferta nuestra. El usuario se encuentra en la oferta visualizada
Flujo principal	1. El usuario selecciona su checkbox de *Finalizar Oferta*. 2. El usuario pulsa el botón modificar oferta. 3. Se actualiza el estado del intercambio
Flujo alternativo	No existe
Postcondición	Intercambio fijado finalizado por parte del usuario

Tabla 13 *Caso de uso 13*

Cancelar intercambio

La Tabla 14 muestra el Caso de uso 14.

Identificador	UC14 – Cancelar intercambios
Descripción	Cancelar un intercambio
Actores	Usuario registrado
Precondición	El usuario está visualizando un intercambio UC10 – Visualizar intercambio
Flujo principal	2. El usuario pulsa el botón cancelar oferta. 3. Se envía al usuario a UC9 – Visualizar intercambios
Flujo alternativo	No existe
Postcondición	Se ha eliminado el intercambio de tu lista de intercambios.

Tabla 14 *Caso de uso 14*

3.2. Diagrama de casos de uso

En la Figura 4 se muestra el diagrama de casos de uso.

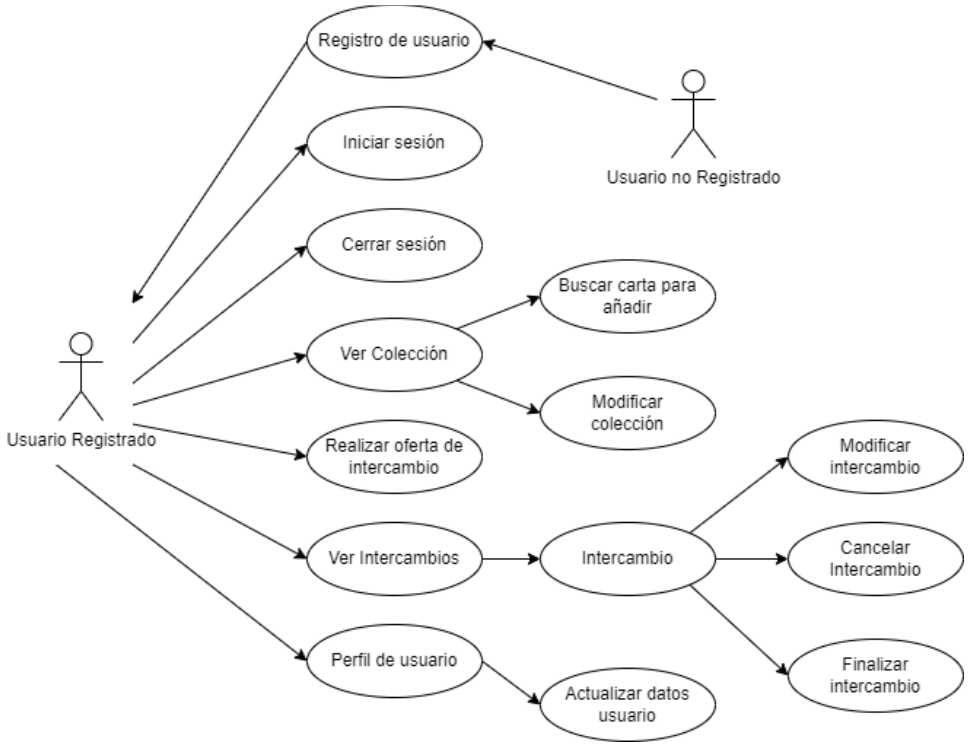

Figura 4 *Diagrama de Casos de uso*

3.3. Arquitectura general

La arquitectura de la aplicación se basa en el estilo RESTful API. En términos generales, la aplicación funciona ejecutando dos programas: CardaliaAPI, o backend, programado en Go, que permite tanto hacer consultas a una API pública llamada Scryfall, la cual contiene toda la información respecto las cartas del juego (imagen, versiones, etc), como hacer consultas a una base de datos creada para guardar la información de los usuarios. Por otro lado, la aplicación Cliente o frontend, que se comunica con el backend, que ofrecer a los usuarios una interfaz web sencilla con la que puedan hacer uso del servicio de intercambio.

En la Figura 5 se explica la arquitectura general de la aplicación.

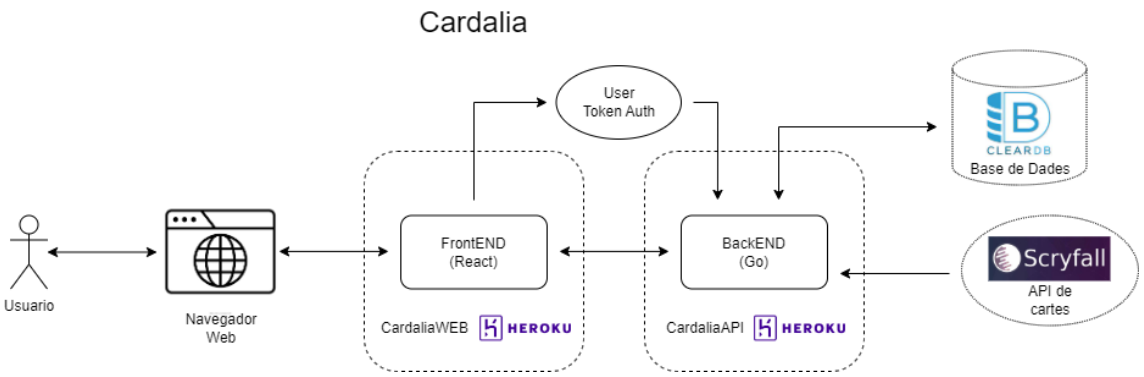

Figura 5 *Arquitectura general de Cardalia*

El usuario, mediante un navegador web, se comunica con el frontend. Este, mediante solicitudes HTTP, hace llamadas a el backend para adquirir, modificar o eliminar datos del usuario. Hay un middleware que comprueba la autenticidad del usuario que hace la llamada al backend cuando esta necesita autorización (token).

Por otra parte, el backend, cuando recibe una llamada del frontend, consulta en la base de datos para recoger o modificar datos del usuario y al mismo tiempo puede hacer consultas en la API de Scryfall para obtener toda la información de las cartas.

El backend y el frontend están en despliegues separados, es decir, los dos están desplegados en su propio espacio y son accesibles públicamente.

3.4. Diseño de la base de datos

En cuanto a la base de datos, tuve claro desde un inicio que quería algo sencillo, conocido y fácil de usar, es por ese motivo que he decidido usar MySQL como el lenguaje para programarla, pues una base de datos relacional es suficiente para cubrir mi proyecto.

Hay que tener en cuenta que existen múltiples bases de datos públicas para obtener información de las cartas de Magic. Por este motivo no he recreado yo mismo la base de datos con todas las cartas, pues existen servicios para consultar toda la información que se necesite de una carta, ya sea el nombre, la imagen de la carta, una versión específica, etc. En mi caso uso Scryfall.

Para el diseño de la base de datos he tenido en cuenta los siguientes requisitos:

- Existen varias versiones, acabado y condición física de una misma carta, y que los usuarios tienen que ser capaces de definir estos parámetros y modificarlos.
- Si hay múltiples copias de una misma carta, con la misma versión, misma condición, etc, estas no aparecerán en múltiples filas, sino en una sola y con el número de copias.

Los modelos creados se presentan en Figura 6.

Figura 6 *Modelos base de datos*

La tabla de usuarios, ***User***, tiene como clave primaria *IdUser* y su nombre de usuario *Username* y contraseña *Password*.

La tabla de cartas, ***CardOwnership***, representa la propiedad de una carta o de un conjunto de cartas idénticas de un usuario *IdUser* (una clave foránea de la tabla *User*). Tiene como clave primaria *IdCArd*. *IdVersion* e *IdOracle* son los IDs para identificar las cartas y recuperar su información en la API de Scryfall. El *IdVersion* referencia a la versión de la carta mientras que *IdOracle* referencia a la carta en sí. Es decir, una misma carta puede tener varias versiones distintas, pero todas las versiones de esa carta tienen el mismo efecto en el juego y el mismo *IdOracle*. El *Count* hace referencia al número de copias que tienes de una misma carta con la misma versión (*idVersio*), los mismos extras (*Extra*) y el mismo estado físico (*Condition*).

En el caso de la tabla de intercambios, ***Trade***, esta contiene una fila para cada fila de la tabla *CardOwnerwhip* que está en proceso de intercambio. Tiene como clave primaria *IdTrade*. *IdCard* es una clave foránea de *CardOwnership*. *IdUserOrigin* y *IdUserOwner* son claves foráneas que hacen referencia al usuario que quiere la carta i el usuario que la tiene respectivamente. *Select* sirve para determinar la cantidad de copias los usuarios seleccionan inicialmente cuando realiza una oferta de intercambio. Finalmente, el campo *Status* sirve para saber cuál es el estado del intercambio.

Por motivos de seguridad, me he propuesto no exponer públicamente nunca ninguna de las claves primarias o foráneas de la base de datos, es decir no usar las en el frontend.

3.5. Diseño de interfaz

3.5.1. Pantallas

La web Cardalia contará con múltiples páginas, algunas con barra de aplicación y otras sin ella. A continuación, se explica su contenido genérico.

- Barra de aplicación: Permite al usuario navegar por todas las pantallas de la aplicación. También permite cerrar sesión.

Páginas sin Barra de aplicación:

- Inicio de sesión: Permite iniciar sesión a los usuarios registrados. Contiene un formulario que el usuario debe rellenar para iniciar sesión.

- Registro: Permite el registro a los usuarios no registrados. Contiene un formulario que el usuario debe rellenar para registrarse.

Páginas con Barra de aplicación:

- Inicio: Se accede a esta página cuando un usuario inicia sesión. Esta contiene una breve descripción de la web y una barra para buscar la carta que se quiere obtener.
- Colección: Contiene la colección del usuario y la barra de búsqueda para añadirle cartas.
- Intercambios: Contiene todos los intercambios que tienes con otros usuarios.
- Intercambio: Permite visualizar un intercambio. También te permite modificar, eliminar y confirmar el intercambio.
- Perfil de usuario: Muestra la información del usuario y permite actualizarla.

Además, he decidido implementar un sistema genérico de predicción de cartas para los buscadores de la aplicación ya que la API de Scryfall tiene un método para autocompletar el nombre de las cartas.

3.5.2. Estilos

En cuanto la estética y la estilización de la aplicación web, no son de vital importancia para este proyecto y no están dentro de su alcance.

Por otro lado, existen distintos componentes de código abierto que tienen un estilo ya predeterminado como es el caso de los componentes de Material-UI.

Usando de base el estilo que dan estos componentes para aplicarlo a otros componentes o secciones hace que se logre un estilo general suficientemente correcto para el proyecto.

4. Implementación

4.1. Toy Aplication

Al principio, antes de empezar a desarrollar el backend, me propuse desarrollar una primera aplicación (Toy aplication) para probar el funcionamiento de la API de Scryfall. Esta es muy simple y sirve para crear una colección de cartas de forma interactiva mediante terminal (Figura 7). El programa hace llamadas a la API para coger la información de las cartas y guardar los datos de forma temporal durante la ejecución del programa.

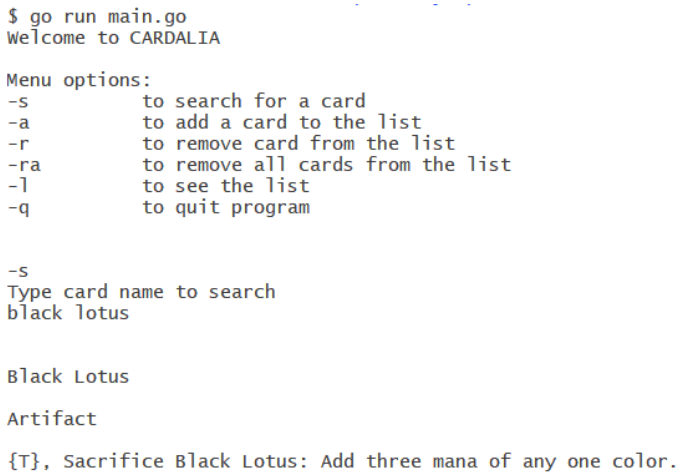

***Figura 7** Funcionamiento Toy application*

Una vez comprobado el correcto funcionamiento de la API de Scryfall, he podido proseguido con el inicio del desarrollo del backend.

4.2. Backend

El backend de la aplicación, CardaliaAPI, tiene la función de consultar en la base de datos y en la API de Scryfall las peticiones que hace CardaliaWEB, el frontend. El esquema del funcionamiento se ve en la Figura 8.

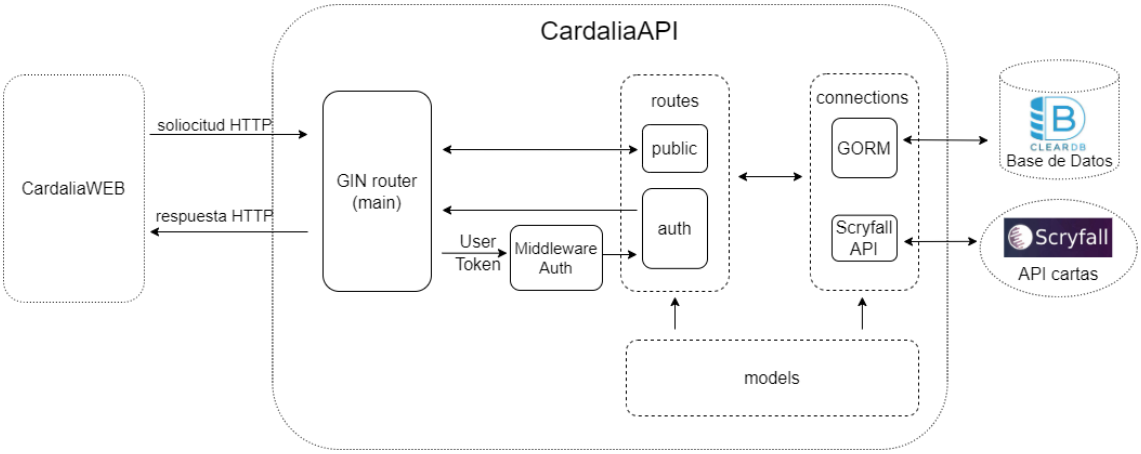

Figura 8 *Esquema funcionamiento backend*

Cuando *CardaliaWEB* envía una solicitud HTTP, esta es recibida por el *router* del backend, creado usando la librería Gin de Go [25] y situado en el *main* del programa.

En el *main*, aparte de la propia definición del *router*, se encuentran definidos todos los métodos de CardaliaAPI. Por otro lado, se definen las CORS [26] para permitir los métodos, cabeceras y parámetros que necesite la solicitud HTTP.

El *router* gestiona la solicitud y según el método que sea, se llama a la función de *routes* correspondiente. Por un lado, *public* se encarga de las peticiones públicas y *auth* se encarga de las que contienen token de usuario como parámetro de autorización.

JSON Web Token (JWT) es un estándar de Internet propuesto para crear datos con firma opcional y/o encriptación opcional del tipo JSON. Los tokens se firman con una clave privada o una clave pública/privada y contienen credenciales de seguridad de una sesión, como la identificación del usuario, y/u otros parámetros. Use un tutorial para realizarlo [27].

En Cardalia los tokens se firman con una clave privada, que se comprueba mediante un middleware, que permite autentificar las peticiones de los usuarios.

Por otro lado, dependiendo de la petición, se establece una conexión con la base de datos mediante GORM [28], el gestor de bases de datos de Go usado, para consultar los datos o modificarlos. Si la petición lo necesita, se consulta la API de Scryfall para obtener la información de las cartas que haga falta.

4.2.1. Modelo

El modelo del backend de la aplicación es representado en la Figura 9.

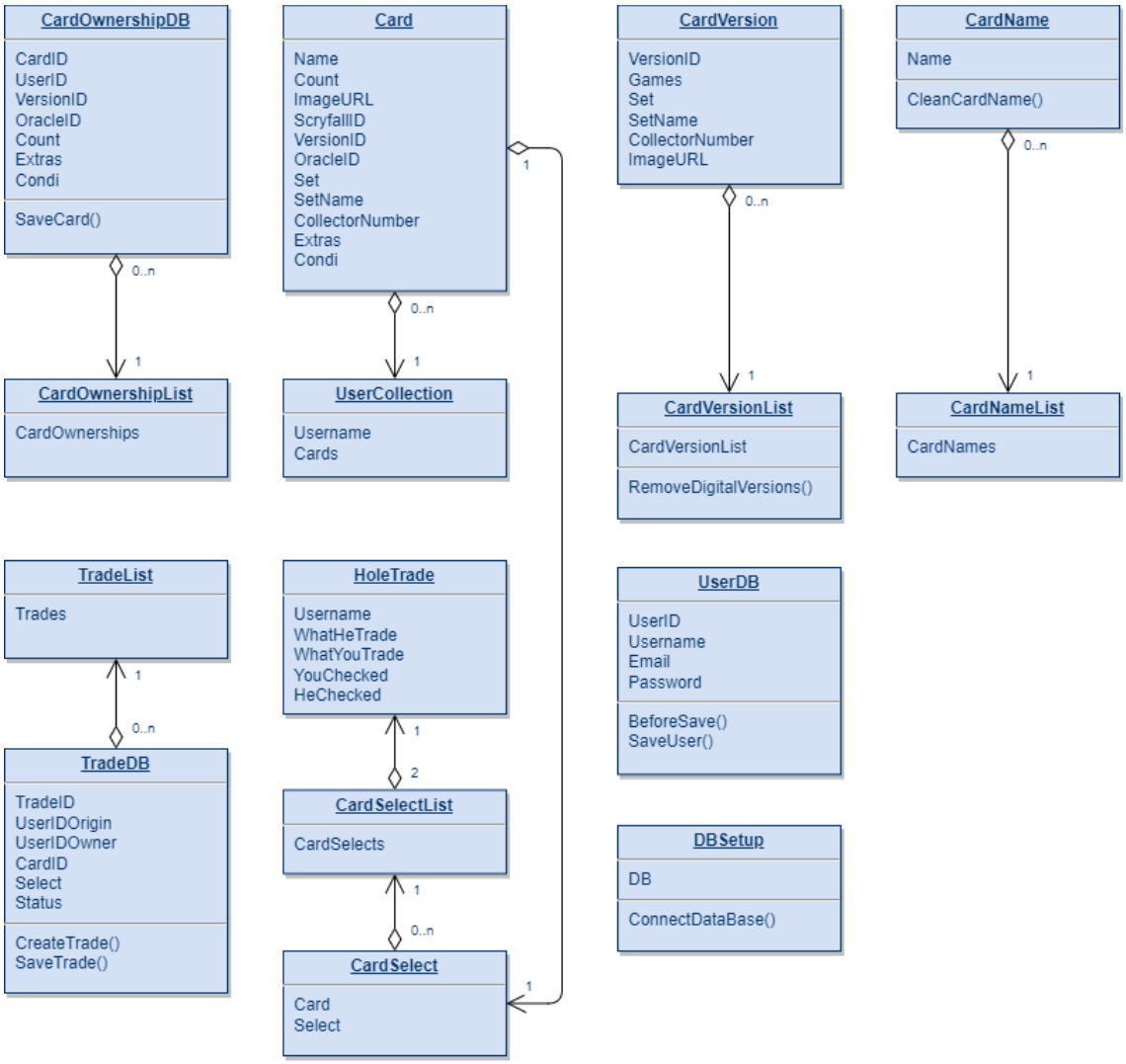

Figura 9 *Modelo backend*

4.2.2. Bases de Datos

Finalmente, el modelo de la base de datos está compuesto por las tablas *UserDB*, *CardOwnershipDB* y *TradeDB* descritas en la Figura 9 y presentan los siguientes cambios respecto al diseño inicial:

En la tabla ***UserDB*** se ha añadido el email del usuario como parámetro.

En la tabla ***CardOwnershipDB*** el parámetro *Condition* ha sido renombrado a *Condi* para evitar confusiones con las condiciones lógicas (palabra reservada de SQL). En esta tabla se puede buscar una carta de forma única usando el parámetro *CardID*, la clave única, o la combinación única de los parámetros *UserID*, *VersionID*, *Extras* y *Condi*.

En la tabla ***TradeDB*** se puede buscar un intercambio mediante su clave primaria *TradeID* o mediante la combinación de los parámetros *CardID* y UserIDOwner, la cual también es única.

El modelo *DBsetup* es el encargado de conectarse con la base de datos usando las variables del entorno adecuadas. Este contiene la variable DB, usada para referenciar las distintas acciones que permite hacer GORM.

4.2.2.1. Gestor Base de Datos

El gestor de bases de datos usado en este proyecto es GORM, una librería de Go usada para la comunicación con una base de datos del estilo MySQL.

GORM ofrece una interfaz de programación que traduce las instrucciones de Go para crear, consultar o modificar datos, a instrucciones SQL.

4.2.2.2. ClearDB

ClearDB [29] es un servicio en la nube de una base de datos, híbrida y/o local para aplicaciones basadas en MySQL. Almacena y administra la base de datos MySQL para que los usuarios no tengan que lidiar con cosas como servidores de base de datos, replicación, almacenamiento avanzado, soporte de TI y especialmente asuntos como fallas en la base de datos.

En mi caso la gestiono mediante un añadido de la página web de Heroku. El servidor está ubicado en Europa. Generalmente la conexión es un poco lenta, especialmente durante la noche.

4.2.3. Uso de la API Scryfall

Tal y como he comentado en múltiples ocasiones, uso la API de Scryfall en este proyecto para consultar la información de las cartas. En concreto he usado los siguientes métodos que ofrece la API.

- Búsqueda de carta por nombre exacto: Usado para recuperar toda la información de una carta según su nombre especificado en el parámetro *exactCardName*.
- Búsqueda de carta por id: Usado para recuperar toda la información de una carta según su id especificado en el parámetro *id*.
- Búsqueda de carta por nombre incompleto*:* Usado para recuperar un listado con los nombres de las cartas que contienen el parámetro *uncompletedCardname* como nombre completo o parcial de una carta.
- Búsqueda de versiones de carta por nombre exacto: Usado para recuperar todas las versiones de una carta con nombre *cardname*.

Todas las llamadas usadas, excepto *cards/autocomplete*, devuelven un objeto o un listado de objetos parecido al objeto Card descrito en el modelo, pero con muchos más parámetros que no he usado en el proyecto. De esta forma, usando *Card* se extraen únicamente los parámetros que interesan de la API.

4.2.4. Métodos HTTP

4.2.4.1. Métodos HTTP públicos

POST /register

Este método sirve para el registro de usuario. Este espera tres parámetros en el cuerpo de la solicitud: el nombre de usuario, el correo electrónico y la contraseña del usuario, los cuales son modelados como un *User*.

Su función es comprobar que los datos de registro sean correctos, y guardar-los en la base de datos, creando un nuevo usuario y encriptando su contraseña mediante la librería Bcrypt.

POST /login

Este método se encarga del inicio de sesión del usuario. Espera dos parámetros en el cuerpo de la solicitud, el nombre de usuario y su contraseña.

Su función es comprobar que los datos de inicio de sesión sean correctos y, en caso afirmativo, generar un token de usuario mediante la librería Jwt-go firmado con la clave privada y devolverlo al usuario.

GET /cards/:autocomplete

Este método sirve para obtener las cartas que busca un usuario. El valor de la query *autocomplete* determina el nombre de la carta o parte de él.

El método consulta en la API Scryfall las cartas que puedan existir que contengan el nombre determinado. Una vez obtenido el listado de posibles cartas, se vuelve a consultar en la API Scryfall para obtener de cada una de ellas los datos necesarios para modelarlos como *Card* y responder a la solicitud del usuario con un listado de estos objetos.

GET /cards/versions/:cardname

Este método sirve para obtener las versiones de una carta. El valor de la query *cardname* determina el nombre exacto de la carta.

El método consulta en la API Scryfall las versiones de la carta solicitada y responde a la solicitud con *cardVersionList*, un listado de *cardVersion*.

GET /user/collection/:username

Este método sirve para obtener la colección de un usuario. El valor de la query *username* determina el nombre del usuario.

El método consulta en la base de datos la colección del usuario especificado y devuelve el objeto *UserCollection*, que contiene el nombre de usuario y su listado de cartas en forma de *Card*.

4.2.4.2. Métodos HTTP privados

POST /user/password

Este método sirve para actualizar la contraseña de usuario. La solicitud HTTP contiene el token de autorización en la cabecera y la nueva contraseña como parámetros del cuerpo.

El método reemplaza la antigua contraseña del usuario por la nueva.

POST /user/collection

Este método sirve para que los usuarios guarden su colección de cartas. La solicitud HTTP contiene el token de autorización en la cabecera y la colección del usuario modificada como parámetros del cuerpo.

El método elimina las cartas que no encuentra en la colección enviada y posteriormente actualiza los elementos que ya existen y crea elementos nuevos para los que no existen.

GET /user/collection

Este método sirve para que los usuarios recuperen su colección de cartas. La solicitud HTTP contiene el token de autorización en la cabecera.

El método busca en la base de datos todas las cartas que pertenecen al usuario, recupera su información mediante la API Scryfall y las devuelve en forma de listado de *Card*.

GET /users/collections/:cardID

Este método sirve para obtener las colecciones de los usuarios que contienen una carta en concreto. El valor de la query *cardID* determina la carta. La solicitud HTTP contiene el token de autorización en la cabecera para saber cuál colección no tener en cuanta.

El método consulta en la base de datos todas las colecciones del usuario con la carta especificada y devuelve un listado del objeto *UserCollection*.

POST /user/trade

Este método sirve para realizar una oferta de intercambio. La solicitud HTTP contiene el token de autorización en la cabecera y la nueva oferta de intercambio como parámetro del cuerpo.

El método asocia un objeto *tradeDB* por cada selección que el usuario haya hecho y lo guarda en la base de datos.

PUT /user/trade

Este método sirve para modificar una oferta de intercambio. La solicitud HTTP contiene el token de autorización en la cabecera y la oferta de intercambio modificada como parámetro del cuerpo (*HoleTrade*).

El método asocia un objeto *tradeDB* por cada selección que el usuario haya hecho y lo actualiza en la base de datos, o crea un nuevo elemento para las selecciones que no existan.

DELETE /user/trade/:username

Este método sirve para eliminar una oferta de intercambio con un usuario. La solicitud HTTP contiene el token de autorización en la cabecera y el nombre de usuario como parámetro de la query (*username*).

El método elimina de la base de datos todos los intercambios pendientes (no finalizados) con el usuario especificado.

GET /user/trades

Este método sirve para que los usuarios recuperen sus intercambios. La solicitud HTTP contiene el token de autorización en la cabecera.

El método busca en la base de datos todos los intercambios que involucren al usuario que ha hecho la petición. Se responde con un listado de objetos del tipo *HoleTrade*, que contiene: el nombre del usuario con quien se realiza el intercambio (*Username*), dos listados de selección, uno para cada usuario (*WhatHeTrade* y *WhatYouTrade*) y dos valores booleanos que indican si los usuarios quieren finalizar la oferta (*YouChecked* y *HeChecked*).

4.3. Frontend

El frontend de la aplicación, CardaliaWEB, tiene la función de permitir al usuario internauta interactuar con la interfaz web mediante conjunto de páginas y componentes gráficos. Está desarrollada con React. El esquema del funcionamiento es de la Figura 10.

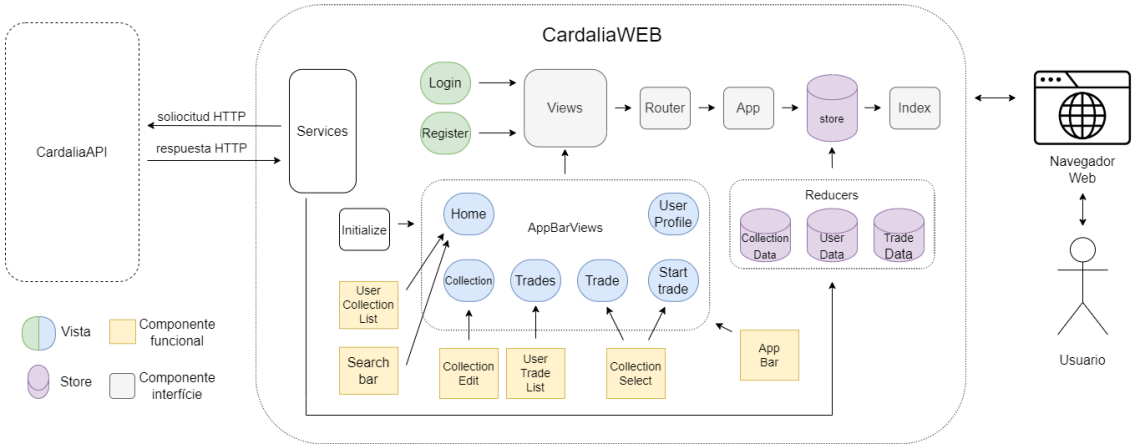

Figura 10 *Esquema frontend*

El usuario, mediante un navegador web, visualiza las distintas pantallas de la aplicación. El usuario no registrado solo puede visualizar las pantallas verdes, es decir, el registro (*Register*) y el inicio de sesión (*Login*). Por otro lado, el usuario registrado, una vez ha iniciado sesión, puede visualizar las pantallas azules, las cuales pertenecen al componente *AppBarViews* y contienen el componente funcional *AppBar*.

Se puede navegar entre las distintas vistas de la aplicación gracias al *Router* definido en *App*. Este permite la navegación entre vistas de varios componentes en la aplicación, permite cambiar la URL del navegador y mantiene la interfaz de usuario sincronizada con la URL.

He usado una Redux *Store* [30] para guardar información útil que se mantiene durante toda la navegación. Explico su uso en el apartado *4.3.3.2 Redux store*.

Service es el fichero que realiza las solicitudes al backend. Este es usado por las distintas vistas y componentes de la aplicación para hacer las peticiones al backend que haga falta.

Cuando un usuario inicia sesión en la aplicación, se actualiza *UserData* con los datos de usuario y su token de sesión. Después, se accede de forma predeterminada al inicio de la aplicación y el *Router* usa las rutas de *AppVarViews*. Este, llama a la función principal del fichero *Initialize*, la cual hace una solicitud al backend para coger la colección (mediante el método *GET /user/collection*) y los intercambios del usuario (mediante el método *GET /user/trades*) y los guarda en *CollectionData* y *TradeData* respectivamente.

4.3.1. Componentes Funcionales

CollectionEdit

Este componente es usado para representar la colección del propio usuario. Extiende el componente DataGrid de Material-UI.

Este componente usa los datos alojados en la *Store* (*CollectionData*) de la aplicación y es el encargado de permitir al usuario interactuar con su propia colección, modificándola a su antojo.

Tiene las siguientes columnas: *Copies*, *Name*, *Version*, *Extras*, *Condition*, y *Actions*.

El componente permite incrementar o decrementar el número de *Copies*. También permite modificar las columnas *Version*, *Extras* y *Condition* si se pulsa el botón *Modify* de la columna *Actions*. Para actualizar el componente debidamente, el usuario debe pulsar de nuevo el botón *Save* de la columna *Actions*.

Si se pulsa sobre el nombre de una de las cartas, se abre una ventana nueva en el navegador con la imagen de la carta.

CollectionSelect

Este componente es usado para representar la colección de los usuarios involucrados en un intercambio. Este extiende el componente DataGrid de Material-UI.

Usa los datos que recibe como *props*: la colección del otro usuario y todos los intercambios con ese usuario permitir al usuario seleccionar las cartas que dese de la otra colección para que sean intercambiadas.

Tiene las siguientes columnas: *Copies*, *Name*, *Version*, *Extras*, *Condition*, y *Select*.

El componente permite modificar la selección mediante la columna *Select*.

Si se pulsa sobre el nombre de una de las cartas, se abre una ventana nueva en el navegador con la imagen de la carta.

SearchBar

Este componente es usado para representar la búsqueda de cartas. Extiende el componente TextField de Material-UI.

Cuando un usuario introduce una palabra de más de 3 letras, se llama al método *GET /cards/:autocomplete* del backend para obtener los datos y se muestran los resultados debajo en formato de imagen.

AppBar

Este componente es usado para permitir a los usuarios registrados navegar por la página web. Extiende el componente AppBar de Material-UI.

Esta contiene de tres botones para navegar a las pantallas de *Home*, *Collection* y *Trades* y un desplegable para navegar a *UserProfile* o cerrar sesión. Usa los datos de la *Store UserData*.

UserTradeList

Este componente es usado para mostrar el listado de intercambios de un usuario.

Cada intercambio, contiene el nombre del usuario y el estado del intercambio. Si pulsamos sobre uno intercambio podremos navegar hasta la página del intercambio pulsado.

UserCollectionList

Este componente es usado para mostrar el listado colecciones que contienen la carta deseado por el usuario buscada en la página de inicio.

Contiene enlaces para navegar a cualquiera de las colecciones de usuario mostradas.

4.3.2. Paginas

Login

Esta es la página que renderiza el inicio de sesión. Contiene el formulario de inicio de sesión.

Cuando el usuario pulsa el botón de *Login*, se hace una llamada al backend usando el método *POST /login*. Se informa al usuario mediante una alerta si los datos son correctos o no, y en caso afirmativo se accede a la página de *Home*.

Register

Esta es la página que renderiza el registro de usuario. Contiene el formulario de registro de usuario.

Cuando el usuario pulsa el botón de *Register*, se hace una llamada al backend usando el método *POST /register*. Se informa al usuario mediante una alerta si los datos son correctos o no, y en caso afirmativo se redirige al usuario a la página de *Login*.

Home

Esta es la página que renderiza el inicio de la web. Contiene la información general de la web y el componente *SearchBar* para buscar las cartas que el usuario desea.

En cuanto el usuario introduce una palabra en el componente *SearchBar*, esta muestra los resultados. Si el usuario pulsa sobre una de las imágenes resultado, se hace una llamada al backend mediante el método *GET /users/collections/:cardID* para obtener y visualizar todas las colecciones de todos los usuarios que contienen la carta buscada mediante el componente *UserCollectionList*.

Collection

Esta es la página que renderiza la colección del usuario. Contiene un componente *SearchBar* para buscar las cartas y un componente *CollectionEdit* para mostrar la colección del usuario.

En cuando el usuario introduce una palabra en el componente *SearchBar*, esta muestra los resultados. Si el usuario pulsa sobre una de las imágenes resultado, se añade al *reducer CollectionData* para que el componente *CollectionEdit* se actualice con la información correspondiente.

Al pulsar el botón *Save collection* se hace una llamada al backend mediante el método *POST /user/collection* para guardar la colección modificada del usuario y se muestra el resultado mediante una alerta.

Trades

Esta es la página que renderiza el listado de intercambios del usuario. Contiene el componente *UserTradeList* con los datos del *reducer TradeData* y sirve para mostrar todos los intercambios del usuario.

Cada elemento de la lista contiene las cartas que se intercambian y el estado del intercambio. Si el usuario pulsa sobre uno de los elementos del componente, se redirige al usuario a la pantalla *Trade* del intercambio correspondiente.

Trade

Esta es la página que renderiza un intercambio pendiente o finalizado. Contiene dos componentes *CollectionSelect* para mostrar tanto la colección del usuario como la colección del usuario con quien se realiza el intercambio. Los datos de esta última se obtienen mediante una llamada al backend con el método *GET /user/collection/:username*.

El usuario puede seleccionar las cartas que quiera intercambiar de las dos colecciones mediante la columna *Select* del componente *CollectionSelect*. Puede marcar un componente checkBox para indicar que quiere finalizar el intercambio. Al pulsarlo sale una alerta para confirmar si el usuario realmente quiere finalizarla.

Al pulsar el botón *Modify*, se hace una llamada al backend con el método *PUT /user/trade* que actualiza todos los parámetros del intercambio. Se indica mediante una alerta el resultado de la llamada.

En caso de que el intercambio esté finalizado, se muestra solamente los dos componente *CollectionSelect* con la selección de las cartas intercambiadas que se hizo en su momento.

StartTrade

Esta es la página que renderiza la oferta de intercambio. Contiene un componente *CollectionSelect* para mostrar la colección del usuario con quien se realiza la oferta de intercambio. Los datos de la colección de este usuario se obtienen mediante una llamada al backend con el método *GET /user/collection/:username*.

El usuario puede seleccionar las cartas que quiera intercambiar de la colección del otro usuario mediante la columna *Select* del componente *CollectionSelect*.

Cuando el usuario pulsa el botón se llama al método *POST /user/trade* que crea un nuevo intercambio en la base de datos.

UserProfile

Esta es la página que renderiza el perfil del usuario. Usa los datos de la *Store UserData*.

Si el usuario introduce los datos del formulario de cambio de contraseña, primero se comprueba que los dos valores de la nueva contraseña sean iguales y seguidamente se realiza una llamada al backend con el método *PUT /user/password* para actualizar su contraseña

4.3.3. Recursos adicionales

4.3.3.1. Hooks

En la mayoría de los componentes uso múltiples Hooks.

UseState

El useState Hook de React nos permite rastrear el estado en un componente funcional sin la necesidad de crear clases. Estos contienen dos parámetros, el estado y el modificador de estado. Lo he usado en el proyecto para garantizar la consistencia de los datos entre las renderizaciones de los componentes.

UseEffect

UseEffect Hook permite realizar efectos secundarios en los componentes. Acepta dos argumentos: la función y sus dependencias. Los he usado en el proyecto para: obtención de datos, actualización directa del DOM, temporizadores y demás.

UseInitialize

UseInitialize Hook es un Hook personalizado que he creado para inicializar las *reducer stores* en cuanto un usuario accede a la aplicación. Funciona bien para no repetir llamadas innecesarias al backend.

4.3.3.2. Redux store

Para asegurar consistencia de datos entre las distintas pantallas de la web he decidido usar una Store de Redux.

Es un objeto que proporciona el estado de la aplicación. Este objeto es accesible con ayuda del proveedor en los archivos del proyecto. La única forma de cambiar el estado dentro de él es enviar una acción sobre él. En la Figura 11 [31] se muestra un diagrama de su funcionamiento.

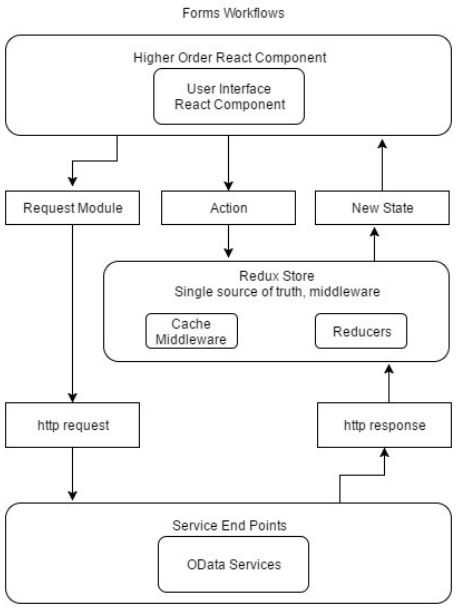

Figura 11 *Diagrama funcionamiento Redux store*

4.3.3.3. Alertas

Para que la web sea más fácil de usar y poder guiar al usuario durante todo el proceso de intercambio he usado alertas.

Concretamente he usado la librería SweetAlert2 [32] de React. Esta librería permite generar alertas de forma sencilla y con una capacidad de customización muy elevada.

Los errores y los mensajes generados en el backend son usados para informar al usuario las distintas acciones que realiza en el sistema mediante estas alertas.

4.4. Despliegue

Para hacer el despliegue de la aplicación y ponerla en producción, es decir, al alcance de todos los usuarios de internet he usado Heroku.

En un inicio mi propósito era hacer un mismo despliegue tanto para el backend como para el frontend. Alojarlos en el mismo sitio web i destinar un proceso para ejecutar el backend i otro para el frontend. De esta manera, una vez encontrada la configuración necesaria para hacer un despliegue simultáneo, este sería muy fácil y cómodo de realizar.

Por otro lado, el frontend de la aplicación sería el único capaz de hacer consultas al backend y el hecho de tener los dos servidores en el mismo sitio, podría provocar brechas de seguridad para la aplicación. Por esta falta de escalabilidad, encapsulación, seguridad y principalmente por su compleja configuración, finalmente he decidido no hacer el despliegue junto.

Tal y como comento en el apartado de diseño, el despliegue del backend y el frontend se hace de forma separada. Debido a este cambio, me di cuenta de que, para realizar un despliegue por separado, tanto el backend como el frontend tenían que estar en repositorios de GitHub distintos y yo tenía todo el proyecto junto en uno mismo.

Finalmente dispongo de dos repositorios privados en GitHub, CardaliaAPI, donde se ubica el backend de la aplicación y CardaliaWEB, que contiene el frontend. Estos están asociados con Heroku y, para realizar el despliegue de cada uno, se ha de realizar previamente un commit con los cambios al repositorio asociado a Heroku.

4.4.1. Despliegue del backend

El backend de la aplicación tiene su propio dominio situado en cardalia-api.herokuapp.com y es accesible públicamente.

Idealmente, aunque este disponga de un middleware para verificar la validez del JWT en una solicitud entrante, el backend debería estar alojado en un servidor privado, pues sería suficiente para cubrir las necesidades del proyecto y lo haría más seguro.

Entonces, para realizar el despliegue del backend, en primer lugar, se hace un nuevo *build* del backend el cual construye el archivo ejecutable *CardaliaAPI.exe*, ubicado en la carpeta *bin*. Seguidamente, se actualizan los cambios en el repositorio CardaliaAPI y finalmente se hace el despliegue de la rama principal) a Heroku. Los servidores de Heroku funcionan corriendo procesos (Dynos) que ejecutan los programas. El backend, dispone de un Dyno que se ocupa de ejecutar el archivo CardaliaAPI.

4.4.2. Despliegue del frontend

El frontend de la aplicación tiene su propio dominio situado en cardalia.herokuapp.com y es accesible públicamente.

El despliegue del frontend ha sido más sencillo de realizar ya que hay más tutoriales y más soporte para el caso de React.

Para realizar el despliegue del frontend, en primer lugar, se hace un nuevo *build* del backend el cual construye la carpeta *build*. Seguidamente, se actualizan los cambios en el repositorio CardaliaWEB y finalmente se hace el despliegue de la rama principal a Heroku. El frontend, dispone de un Dyno que se ocupa de ejecutar el DOM de la página web.

5. Pruebas y evaluación

5.1.1. Pruebas del backend

En un inicio, para probar el correcto funcionamiento del backend de la aplicación, he usado Postman. Es decir, una vez implementaba una nueva funcionalidad, me aseguraba que cada endpoint funcionaba como es debido, de forma manual, mediante la comprobación del resultado de las peticiones de Postman.

Pero llegado el momento, alrededor de la segunda etapa del proyecto i debido al crecimiento del código y de sus funcionalidades, comencé a utilizar pruebas automáticas para los distintos servicios de la aplicación mediante la librería Apitest [33] de Go.

He aplicado pruebas para cada uno de los distintos métodos de la aplicación. Para los métodos que modifican la base de datos, compruebo también que el resultado de esta sea el esperado.

Una vez creada una nueva funcionalidad, se ejecutan las pruebas para comprobar el correcto funcionamiento del frontend antes de realizar el nuevo commit de cambios.

5.1.2. Pruebas del frontend

De la misma forma que en el backend, en un inicio, para poder probar el funcionamiento del frontend, he usado el propio servidor de desarrollo de npm [34]. Es decir, a medida que desarrollaba y creaba la página web yo mismo probaba y aseguraba su correcto funcionamiento a nivel local y de forma manual.

Pero, llegado el momento, igualmente que en el backend, puesto que el código de la aplicación crecía y dejaba de tener el control sobre el resultado de cada función del frontend, empecé a usar Jest [35]. Jest, es una librería de JavaScript para hacer pruebas unitarias en las funcionalidades más importantes del frontend de la aplicación.

He realizado una prueba unitaria con Jest para la mayoría de Casos de uso. De esta forma se comprueba el correcto funcionamiento genérico del frontend.

Una vez creada una nueva funcionalidad, se ejecutan las pruebas para comprobar el correcto funcionamiento del backend antes de hacer el nuevo commit de cambios.

5.1.3. Pruebas de usabilidad

Por otra parte, se ha considerado necesario realizar pruebas de usabilidad. Estas consisten en probar con usuarios reales el funcionamiento de la aplicación.

Las pruebas de usabilidad son un método para evaluar la experiencia del usuario de un producto o sitio web. Al probar la usabilidad con un grupo representativo de usuarios o clientes, podremos saber si los usuarios reales serán capaces de usar el sitio web de manera fácil e intuitiva.

Previamente busqué información para hacer las pruebas de forma correcta. Basé mi sistema en el libro *About face 3: The essentials of interaction design* [36].

Para realizar estas pruebas, se han escogido tres usuarios con 5 a 17 años de experiencia como jugadores/as de Magic i se les ha realizado la misma prueba a todos ellos. Aunque es posible que el grupo representativo sea pequeño, los tres candidatos son usuarios potenciales de la

aplicación. No he visto necesario usar usuarios no potenciales para realizar estas pruebas pues estos no tendrán la necesidad de usar la aplicación.

El objetivo de estas pruebas es lograr finalizar un intercambio con otro usuario de una carta establecida previamente. Para facilitar el proceso de prueba se define previamente la carta que desean los usuarios probados, pues para realizar la prueba con la carta que ellos quieran, debería popular previamente la aplicación con muchos usuarios y muchas colecciones y he querido evitar eso.

Para lograr el objetivo establecido, se tienen que realizar las siguientes acciones o casos de uso: *Registro*, *Login*, *Modificar colección*, *Realizar oferta de intercambio*, *Finalizar oferta de intercambio* y finalmente *Cerrar sesión*.

Las métricas medidas en estas pruebas son las siguientes:

- Tiempo empleado en lograr el objetivo
- Numero de clics dados para lograr el objetivo

He usado la plantilla de la Tabla 15 para anotar los datos de las pruebas realizadas:

Nº de prueba		
Nombre		
Experiencia del juego		
Edad		
Tarea(Caso de uso)	**Nº de clics**	**Tiempo**
Registro		
Login		
Modificar colección		
Buscar carta para intercanviar		
Realizar oferta de intercambio		
Finalizar intercambio		
Cerrar sesión		
Total		
Comentarios o errores		

Tabla 15 *Plantilla prueba usabilidad*

Resultados de la prueba

La realización de estas pruebas ha sido muy útil pues he podido detectar múltiples errores de diseño de la interfaz gráfica e implementar soluciones para mejorar la navegación.

Aunque las métricas establecidas están bien para medir la dificultad de la prueba y han sido útiles para determinar en que pantallas los usuarios tienen mayor dificultad, lo más interesante y útil ha sido visualizar como los usuarios realizaban la prueba y sus comentarios posteriores.

Al ver que uno de los participantes tenía problemas con el inicio de sesión y el registro, he decidido poner alertas en la aplicación que te indiquen los posibles errores generados.

Por otro lado, a nivel general, no queda del todo claro como se ha de iniciar una oferta de intercambio. Solo uno de ellos fue capaz de encontrarlo pues en un inicio leyó bien la información de bienvenida mientras que los otros dos no. Esto ha hecho replantearme la

distribución de la página web para que sea más fácil de usar poniendo la barra de búsqueda de carta para un intercambio en la página de *Inicio* de la aplicación. También he puesto algún comentario indicativo para mejorar la navegación general (Por ejemplo, en la pantalla de Intercambios (*Trades*), he puesto un mensaje fijo avisando que los intercambios se empiezan en la página de *Inicio*).

Por último, tampoco queda del todo claro el funcionamiento del botón de Guardar Colección y Modificar Intercambio. Los usuarios no lo pulsaban en su debido momento. He pensado que la mejor solución es desactivar el botón de forma inicial y activar-lo si se detecta algún cambio. He aplicado este cambio mediante el uso de condiciones booleanas y comparando los estados iniciales de los distintos objetos.

6. Resultado

Finalmente, como resultados del proyecto realizado, la página web de Cardalia ubicada en cardalia.herokuapp.com, contiene las siguientes paginas o vistas.

6.1. Pantallas

Inicio de sesión (Login)

Esta página permite iniciar sesión a los usuarios registrados (Figura 12). Contiene un formulario que el usuario debe rellenar para iniciar sesión. Salta una alerta informando del estado de inicio de sesión al pulsar el botón de *Login*. Esta contiene un enlace para hacer registro si el usuario no está registrado.

Cuando el usuario inicie sesión, si lo hace con éxito, saltará una alerta indicando que se ha iniciado sesión correctamente y se enviará al usuario a la página de *Inicio*. En caso de que los datos introducidos no sean correctos, saltará una alerta indicándolo.

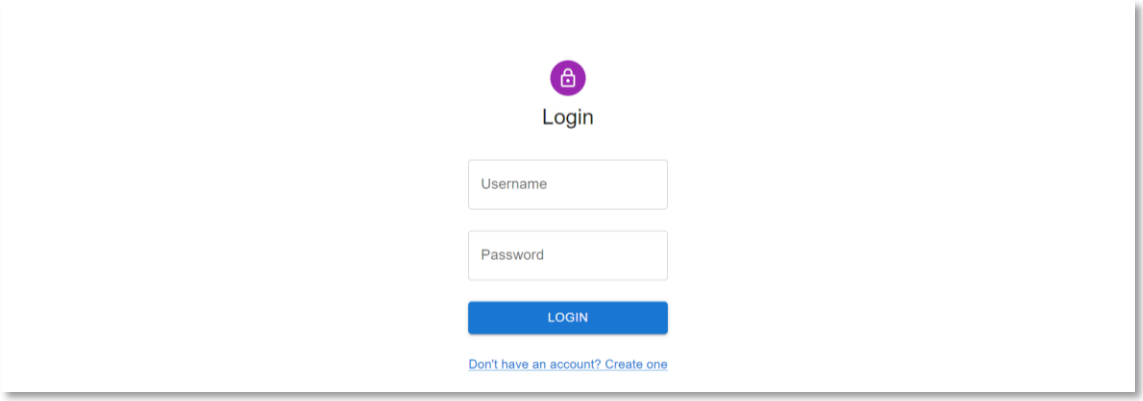

Figura 12 *Pantalla Login.*

Registro

Esta página permite el registro a los usuarios no registrados (Figura 13). Contiene un formulario que el usuario debe rellenar para registrarse pulsando el botón *Register*. Esta contiene un enlace para hacer el inicio de sesión si el usuario ya está registrado.

Cuando el usuario se registre, si lo hace con éxito, saltará una alerta indicando que se ha registrado correctamente y se envía al usuario a la página de *Inicio de sesión*. En caso de que los datos introducidos no sean correctos, saltará una alerta indicando que el nombre de usuario ya está registrado.

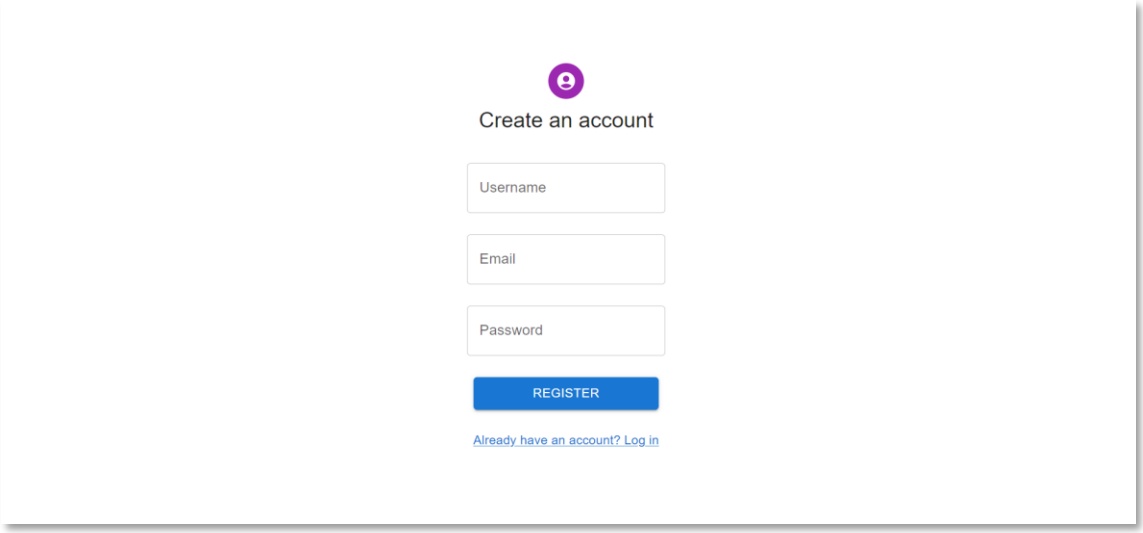

Figura 13 *Pantalla registro.*

Barra de navegación

Todas las pantallas exceptuando la de Registro y la de Inicio de sesión contendrán una barra de navegación (Figura 14). Esta contiene botones para ir al *Inicio* (*CARDALIA*) para ir a la *Colección* (*Your Collection*) y para ir a tus *Intercambios* (*Your Trades*). También dispone de un desplegable para ver el *Perfil del usuario* (*Profile*) y para cerrar sesión (*Logout*).

Figura 14 *Barra de navegación.*

Inicio

Esta página es a la que se acede cuando un usuario inicia sesión (Figura 15). Contiene una breve descripción de la web y una barra para buscar la carta que se quiere obtener. Cuando un usuario ya ha añadido cartas en su colección, viajará de nuevo al inicio para buscar cartas para intercambiar.

Al buscar una carta (Figura 16) y al pulsarla, aparecerán los usuarios que tienen esa carta en su colección junto con un botón (*cubo de basura*) para borrar la búsqueda (Figura 17). Si pulsamos sobre uno de los usuarios listados, se visualizará la pantalla de *Realizar oferta de intercambio*.

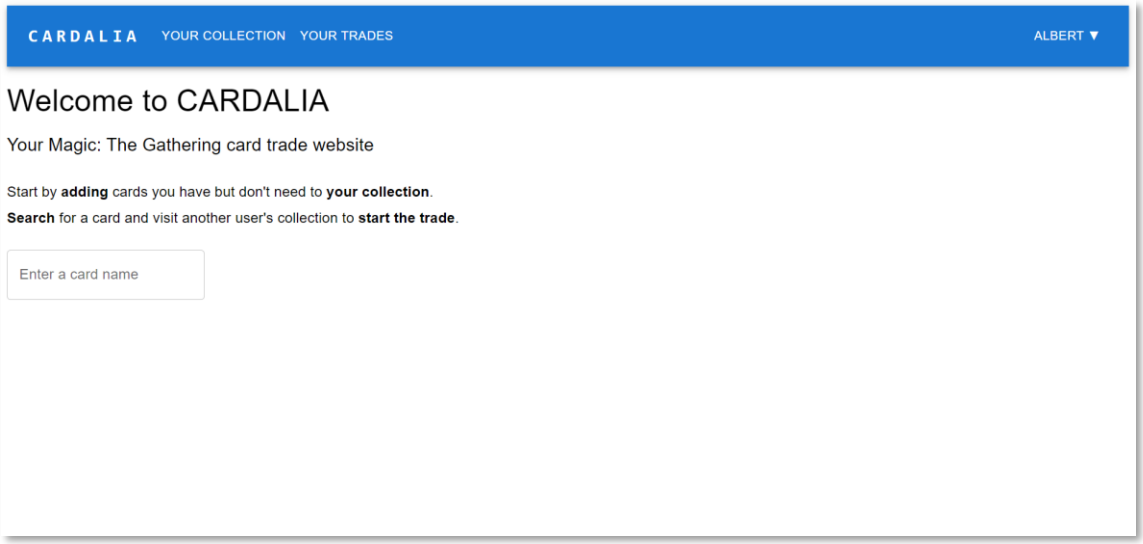

Figura 15 *Pantalla inicio. Ejemplo: El usuario acaba de iniciar sesión.*

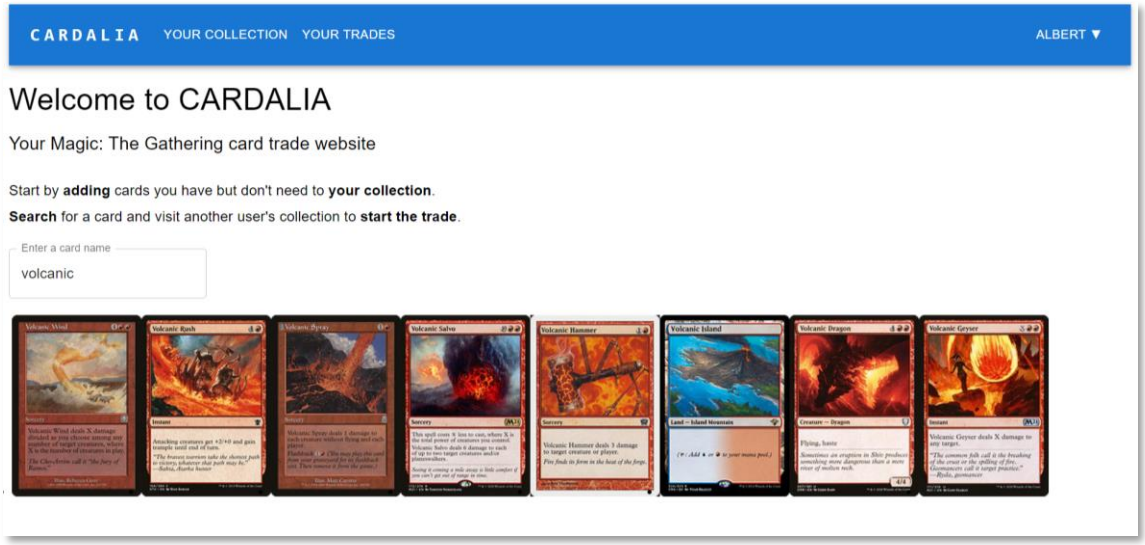

Figura 16 *Pantalla inicio. Ejemplo: El usuario busca la carta Volcanic Island para realizar una oferta de intercambio pues la desea.*

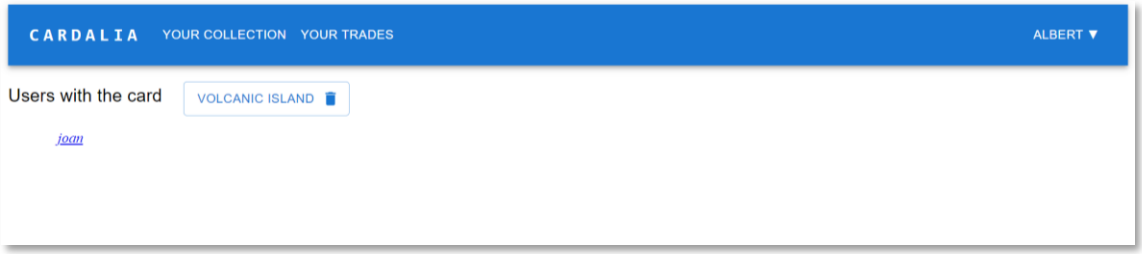

Figura 17 *Pantalla inicio. Ejemplo: Se muestra el usuario joan, que contiene al menos una carta llamada Volcanic Island.*

Colección

Esta página contiene la colección del usuario y la barra de búsqueda para añadirle cartas (Figura 18). Cuando un usuario quiere añadir una carta en su colección, escribe el nombre en la barra de búsqueda (Figura 19). Al pulsar sobre una de las cartas mostradas, esta se añade en una nueva fila si el usuario no la tenía previamente en su colección; o se incrementa el contador de la fila que contenía la carta, si esta ya estaba en la colección (Figura 20).

Se puede alterar el número de copias, la versión, el acabado y la condición física de una misma carta. Para modificar una carta (Figura 21), exceptuando el número de copias, se ha de pulsar previamente el botón modificar carta (*lápiz*) y luego, tras realizar los cambios pulsar el botón guardar carta (*disco*) (Figura 22). El usuario puede ordenar su colección por la columna que desee pulsando el botón que hay en cada una de estas. También puede ocultar alguna de las columnas mediante un menú que incorpora la propia tabla (Figura 23).

Una vez modificada la colección, el usuario ha de pulsar el botón *Save Collection* para guardar los cambios correctamente. Este botón estará inicialmente oculto y cuando se detecte un cambio en la colección aparecerá para poder pulsarlo.

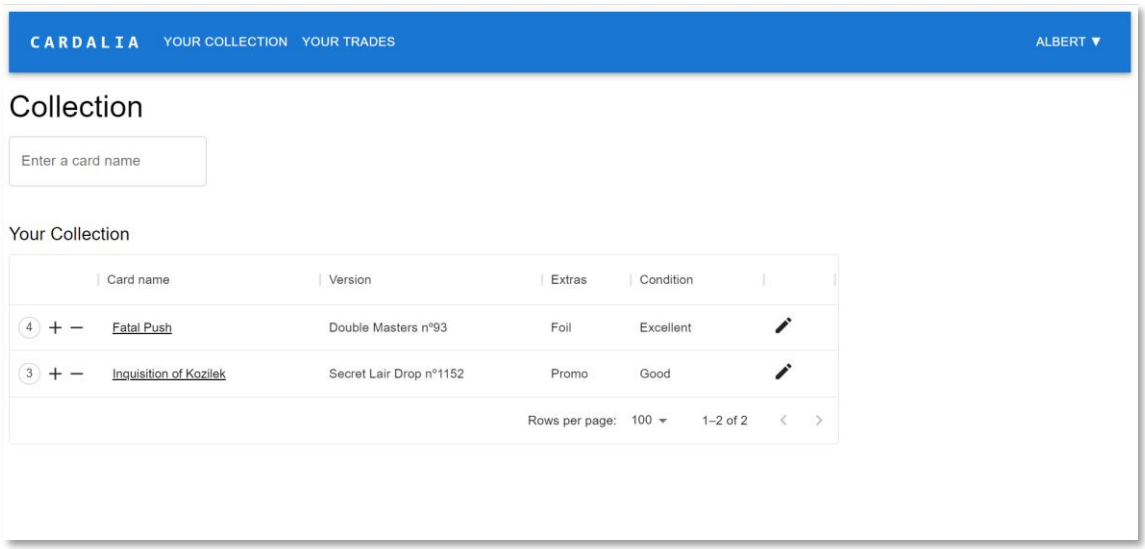

Figura 18 *Pantalla colección. Ejemplo: Sse muestra la colección de cartas del usuario albert.*

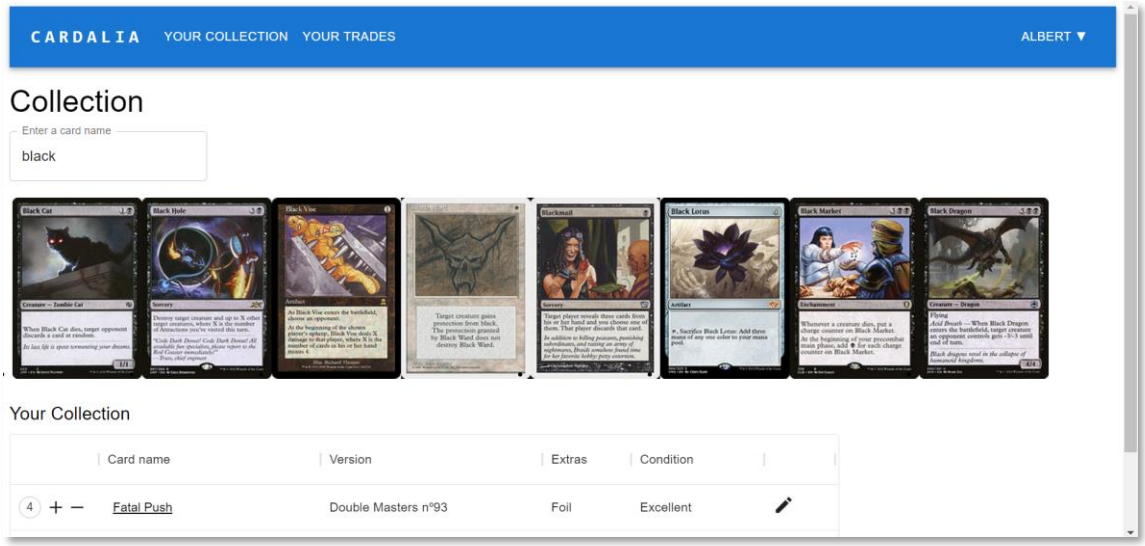

Figura 19 *Pantalla colección. Ejemplo: el usuario introduce en la barra de búsqueda la palabra black y se muestra las cartas que la contienen en su nombre.*

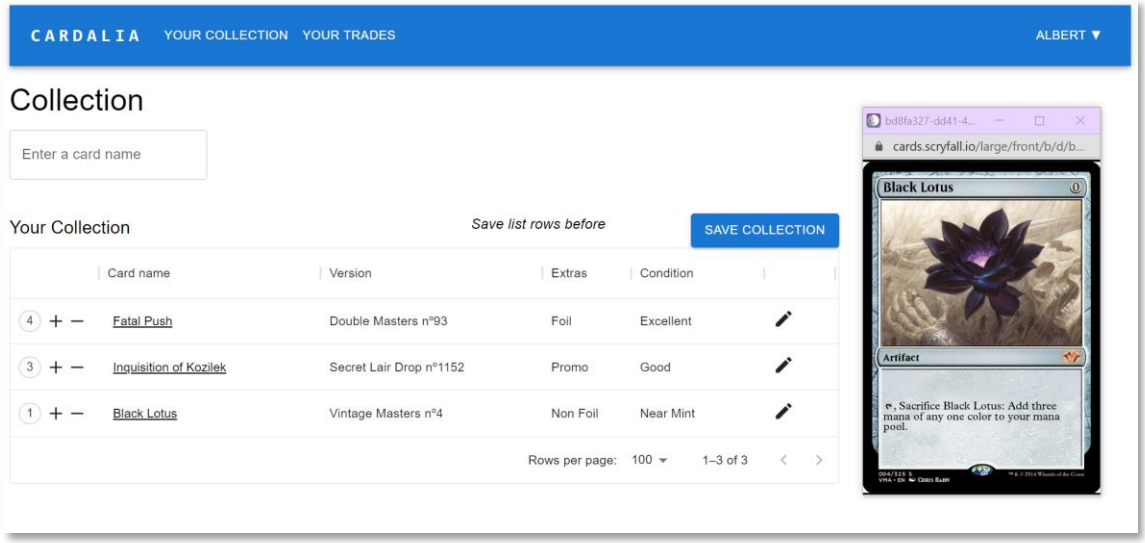

Figura 20 *Pantalla colección. Ejemplo: El usuario ha pulsado sobre la carta Black Lotus y la ha añadido en su colección. Al mismo tiempo ha aparecido el botón Save Collection. El usuario ha pulsado sobre el nombre de la carta y se ha abierto una nueva ventana del navegador con la imagen de la carta.*

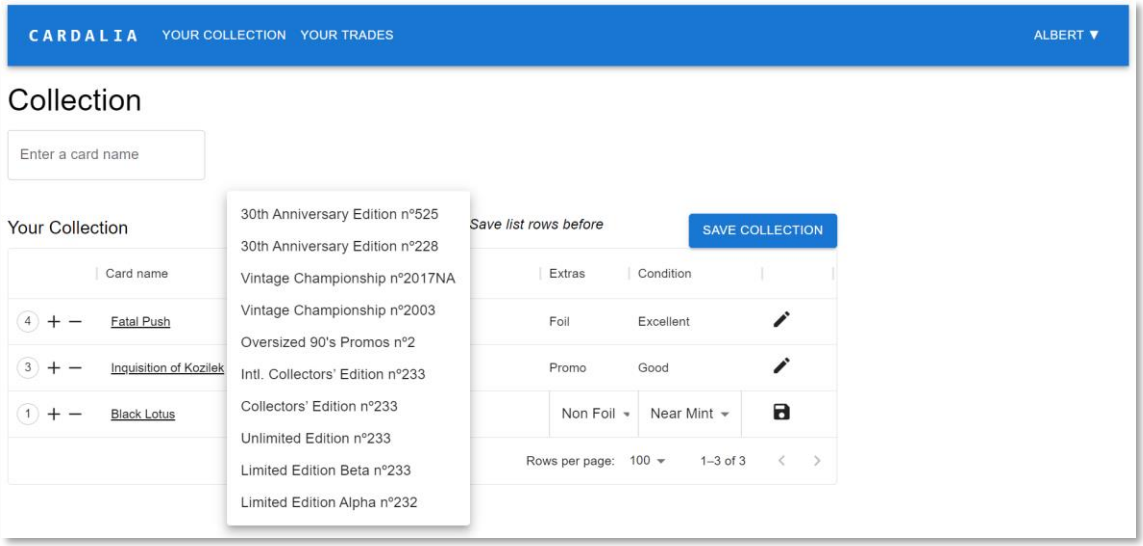

Figura 21 *Pantalla colección. Ejemplo: El usuario ha pulsado el botón modificar carta (lápiz) y ha abierto el desplegable para seleccionar otra versión.*

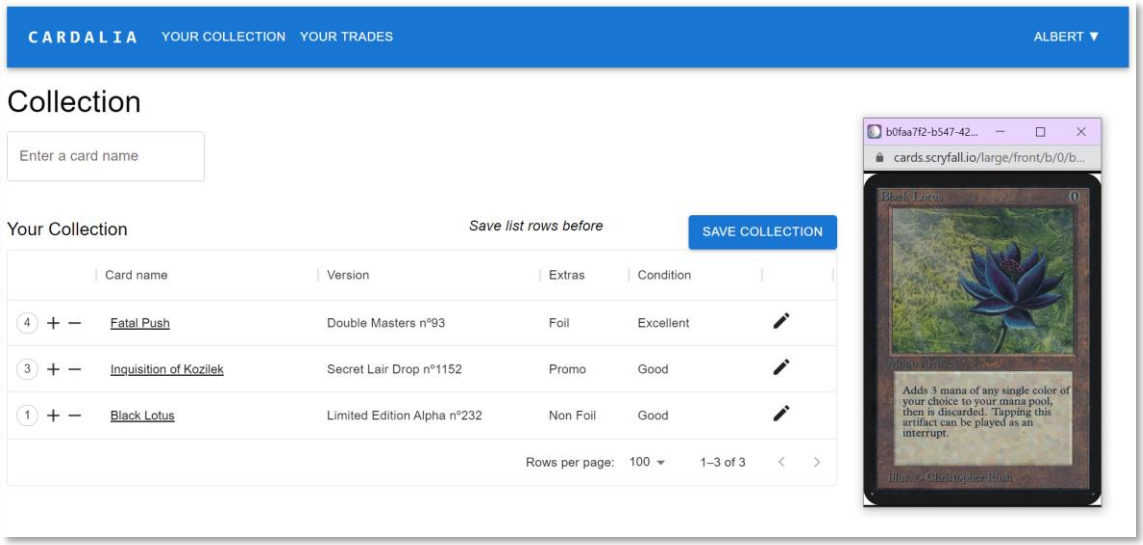

Figura 22 *Pantalla colección. Ejemplo: El usuario modificado la versión, el acabado (extras) y la condición física de la carta y ha pulsado el botón guardar carta (disquete). Seguidamente ha vuelto a abrir la imagen de la carta, con la versión cambiada)*

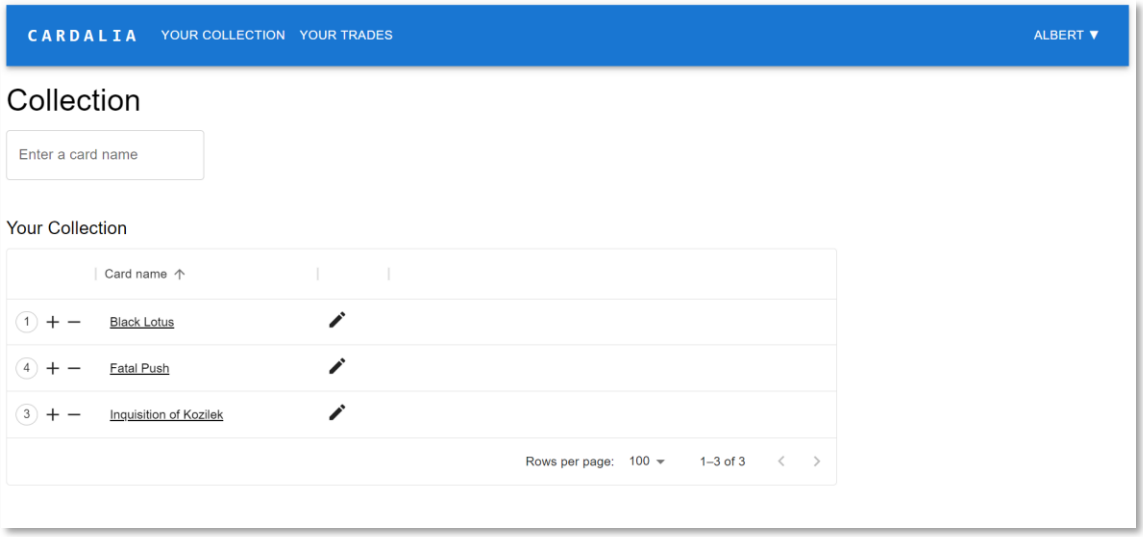

Figura 23 *Pantalla colección. Ejemplo: El usuario ha ocultado las columnas que no le interesan en el momento y ha reordenado la colección por el nombre de la carta)*

Realizar oferta intercambio

En esta pantalla se visualiza la colección de otro usuario. El usuario puede seleccionar el número de copias que desea de cada carta haciendo clic en la celda de la columna *Select*. Esto muestra un desplegable que donde el usuario puede seleccionar la cantidad de cartas que desea (Figura 24).

Una vez el usuario haya seleccionado al menos una copia de una carta, aparecerá el botón *Make Trade* para poder pulsarlo i realizar la oferta de intercambio.

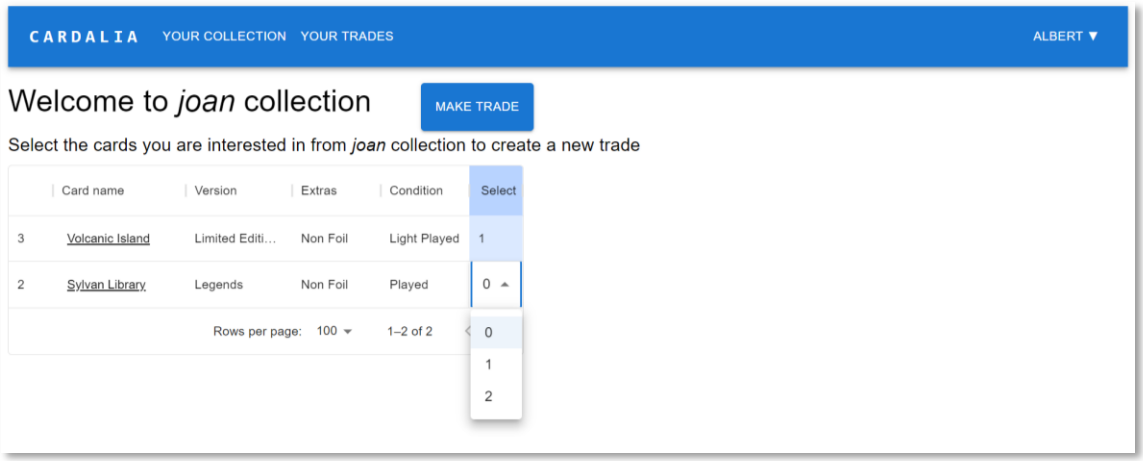

Figura 24 *Pantalla iniciar oferta intercambio. Ejemplo: El usuario albert selecciona las cartas de la colección del usuario joan.*

Intercambios

Esta página contiene todos los intercambios pendientes y finalizados que los usuarios tienen con otros usuarios (Figura 25). Estos aparecen en nombre de listado y contienen una pequeña descripción del estado del intercambio. Al pulsar sobre un elemento del listado se visualizará el intercambio en cuestión.

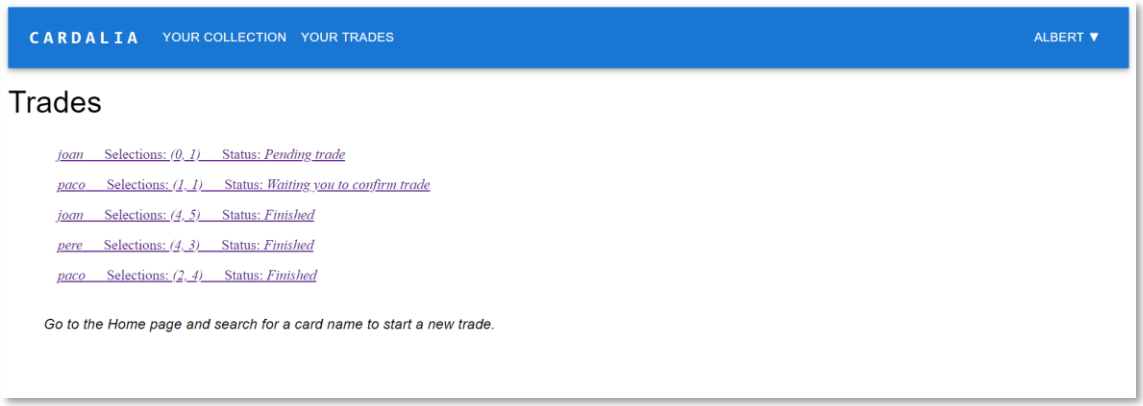

Figura 25 *Pantalla intercambio. Ejemplo: El usuario albert visualiza su listado de intercambios.*

Intercambio pendiente

Esta página te permite visualizar un intercambio (Figura 26). También permite modificar, eliminar y confirmar el intercambio. La página contiene dos colecciones, la del propio usuario y la del usuario con quien realiza el intercambio.

El usuario puede modificar la selección de la colección del otro usuario. Cuando el usuario quiera acabar con el intercambio por su parte, lo informará mediante el checkBox *Finish trade*, que mostrará un dialogo para confirmarlo (Figura 27). Para guardar los cambios del intercambio, el usuario pulsará el botón *Modify Trade* que se activará en cuanto se detecte un cambio en el intercambio.

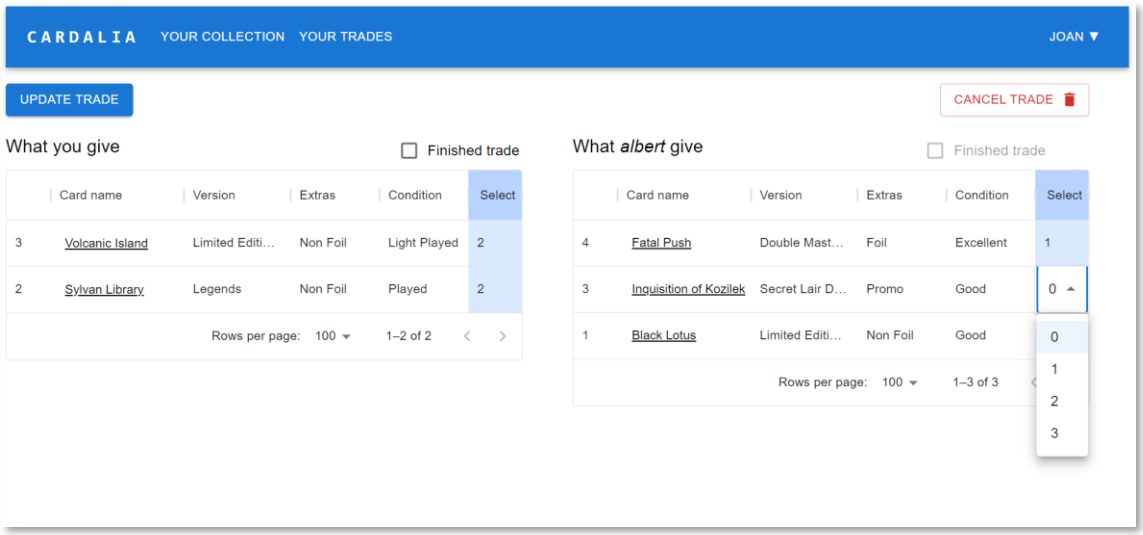

Figura 26 *Pantalla intercambio pendiente. Ejemplo: El usuario joan selecciona las cartas que dese de la colección de albert.*

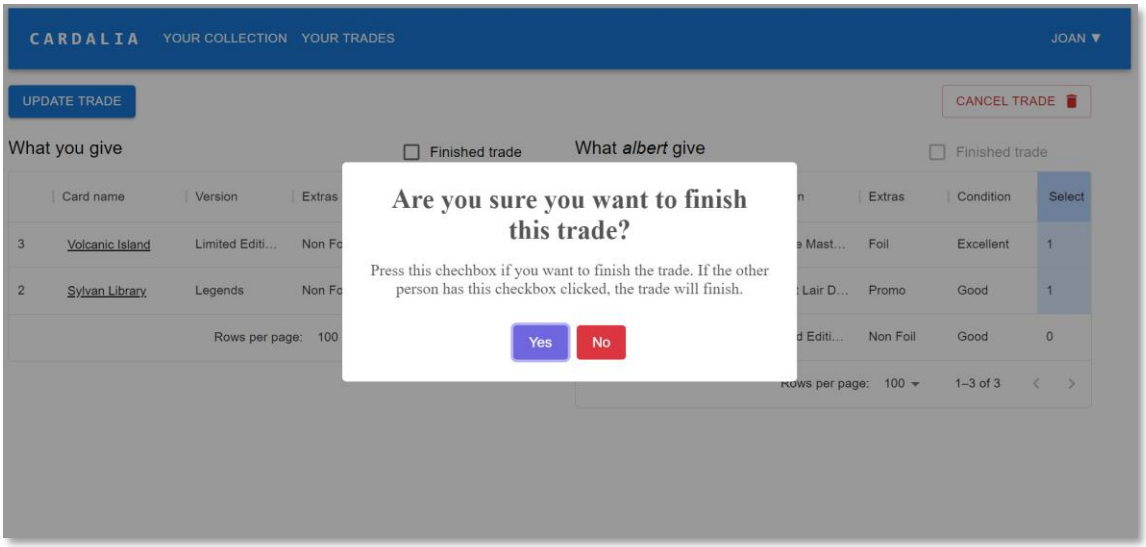

Figura 27 *Pantalla intercambio pendiente. Ejemplo: El usuario joan ha pulsado el check box Fisnished Trade y ha aparecido el dialogo para confirmar que quiere finalizar el intercambio)*

Intercambio finalizado

Esta página muestra las cartas de los intercambios finalizados con otro usuario y su correo de contacto (Figura 28). Contendrá todas las cartas de todas las ofertas finalizadas con ese usuario.

La página no contiene ningún botón y la selección de cartas está desactivada.

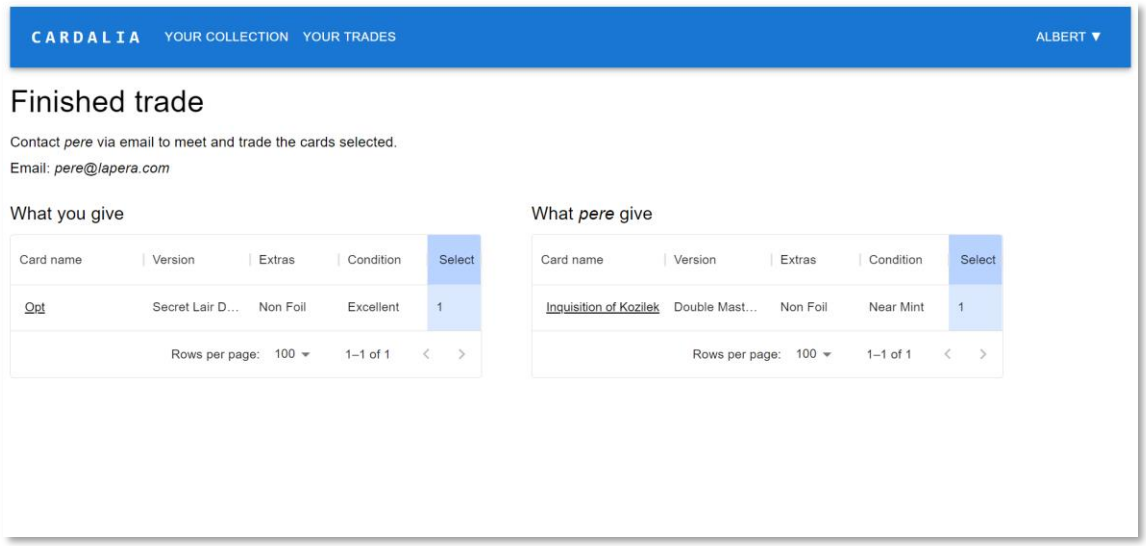

Figura 28 *Pantalla intercambio finalizado. Ejemplo: Se muestran las cartas intercambiadas con el usuario pere.*

Perfil de usuario

Esta página muestra la información del usuario y permite actualizar su contraseña (Figura 29). El usuario puede rellenar el formulario de cambio de contraseña (Figura 30) y pulsar el botón modificar contraseña para efectuar el cambio. Al realizarlo saltará una alerta indicando el resultado de la acción.

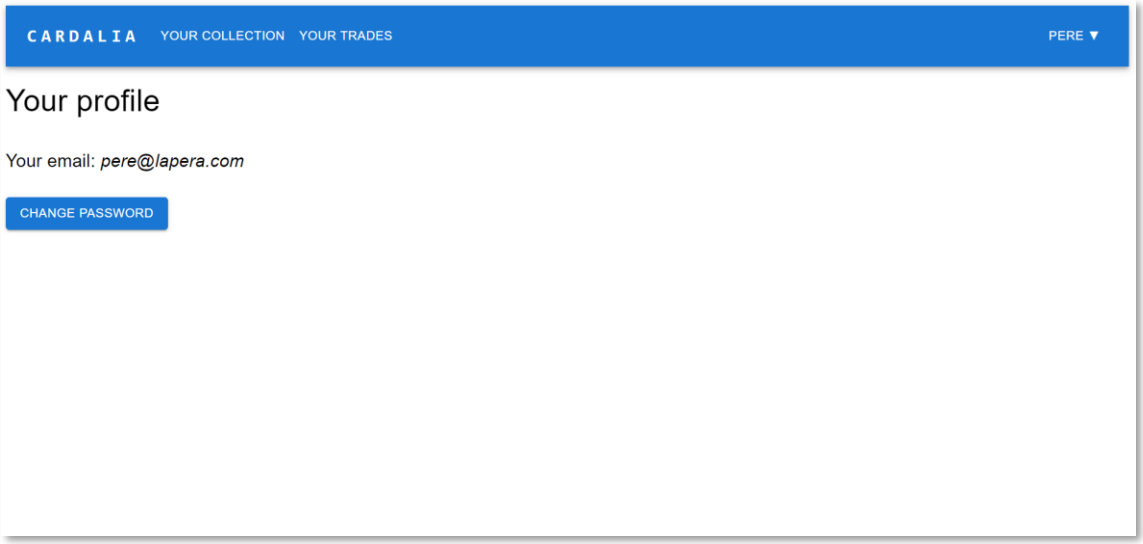

Figura 29 *Pantalla perfil de usuario.*

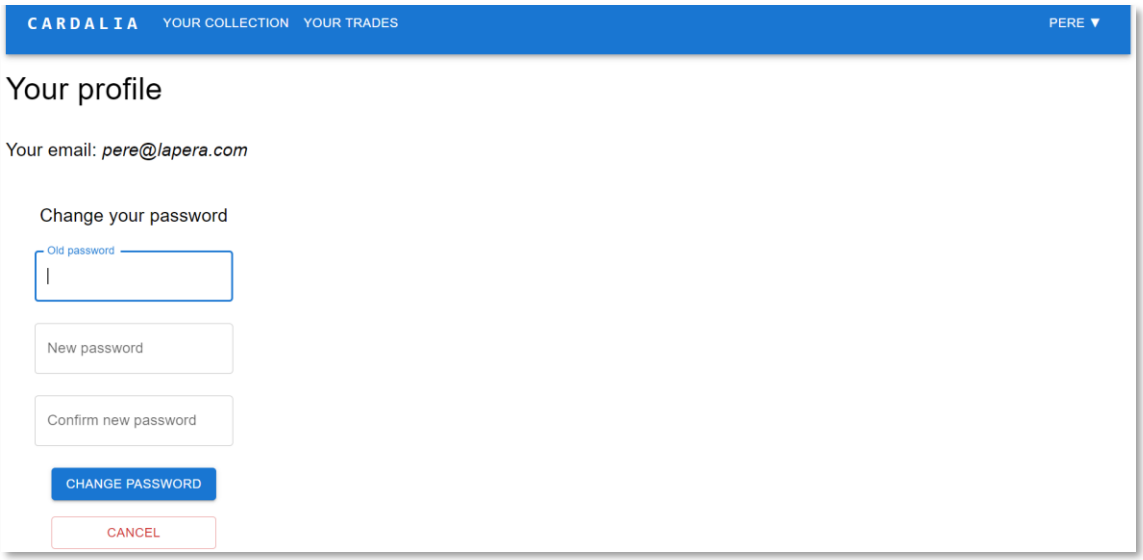

Figura 30 *Pantalla perfil de usuario. Ejemplo: El usuario ha pulsado sobre el botón Change password.*

7. Conclusión

7.1. Conclusiones

El producto final de este trabajo consiste en una aplicación interactiva y de fácil uso, que permite a los usuarios definir su colección e intercambiar cartas del juego de Magic con otros usuarios de la web. Los objetivos descritos al inicio del proyecto han sido cumplidos ya que la aplicación cumple con los requisitos y funcionalidades especificados.

Para el propósito de este trabajo se han elegido herramientas de desarrollo web como el framework React, que ha permitido el desarrollo de toda la interfaz gráfica de la web. El lenguaje Go junto con la API ScryFall y la base de datos MySQL han sido muy útiles para poder responder las peticiones que generan de los usuarios de la web. Al mismo tiempo, se ha seguido una metodología agile para hacer el desarrollo de forma progresiva y para asegurar el funcionamiento del software en cada una de las distintas etapas.

Por otro lado, la realización de este proyecto me ha permitido mejorar y aprender a usar todo un conjunto de herramientas y procedimientos muy útiles para el desarrollo de aplicaciones web, lo cual ha despertado en mí una cierta curiosidad y pasión. También me ha motivado a aprender dos lenguajes de programación nuevos que son ampliamente usados actualmente. Especialmente React. Al haber aprendido un lenguaje para el desarrollo web distinto de Vue, tengo otra perspectiva mucho más sana y organizada de cómo desarrollar interfaces web. En el caso de Go, aprenderlo me ha resultado bastante sencillo, pero podría haber obtenido el mismo resultado o uno incluso mejor de haber usado un lenguaje que ya conociera anteriormente como Python. Pero, aun así, estoy orgulloso de conocerlo pues tengo otros proyectos pendientes con este lenguaje.

Aunque este proyecto se haya enfocado al juego de Magic: The Gathering, seria relativamente sencillo escalar la aplicación para integrar otros juegos de cartas coleccionables, y proporcionar un servicio de intercambio de cartas de múltiples juegos.

7.2. Futuras mejoras

7.2.1. Cambios en la planificación

Despliegue preliminar

Uno de los puntos que plantearía de forma diferente al inicio sería planificar con anterioridad el despliegue de la aplicación. En un primer momento creía que para que el despliegue tuviera sentido, la aplicación tenía que estar más avanzada y por esa razón lo planifiqué para el final de la segunda etapa del proyecto.

Para asegurar una integración continua y un producto funcional desde el principio del proyecto, debería haber hecho el despliegue al finalizar la primera etapa o incluso al tener el proyecto base.

Por otro lado, el backend de la aplicación está desplegado en un servidor público. La razón por la que es público es debido a que en el momento de realizar el primer despliegue del

backend, me informé que Heroku daba la posibilidad de publicar o privatizar un servidor. Inicialmente me iba bien que fuera público, para poder probarlo, pero una vez tenía el proyecto más avanzado, comprendí que tenía que ampliar la tarifa subscripción de Heroku y pagar más, cosa que no pretendo hacer de momento.

7.2.2. Funcionalidades adicionales

Aunque el resultado de la aplicación sea suficiente, pues se han cumplido los objetivos marcados y la primera versión de la aplicación cumple con los requisitos y funcionalidades básicas para el correcto funcionamiento de la página web, sin embargo, me gustaría haber integrado más funcionalidades en la aplicación.

Confirmación de registro por correo electrónico

La confirmación de registro mediante correo electrónico es una excelente forma de aumentar la seguridad de la aplicación y mejorar la experiencia del usuario. Al requerir que los usuarios verifiquen su dirección de correo electrónico antes de poder acceder a la aplicación, se pueden evitar ataques de creación automática de cuentas mediante bots, reduciendo así la carga en el sistema y mejorando la seguridad general de la aplicación.

Además, la confirmación de registro mediante correo electrónico ofrece una mayor confianza para los usuarios, ya que les asegura que su registro ha sido procesado correctamente y que están listos para usar la aplicación. También se puede utilizar para enviar información importante sobre la cuenta del usuario, como instrucciones para restablecer la contraseña.

Para implementar esta funcionalidad, se puede utilizar un servicio de correo electrónico externo, como Sendgrid o Mailgun, para enviar correos electrónicos de confirmación a los usuarios recién registrados.

Integración de comparador de precios

La integración de un comparador de precios en la aplicación sería una gran adición para mejorar la experiencia del usuario y ayudar a evitar timos. El comparador de precios permitiría a los usuarios comparar los precios de las cartas en diferentes sitios web de compraventa, lo que les ayudaría a tomar decisiones informadas sobre sus intercambios.

La implementación de un comparador de precios puede ser relativamente sencilla gracias a la utilización de la propia API Scryfall. La API Scryfall proporciona información actualizada sobre los precios de las cartas en las principales webs de compraventa, lo que significa que no se necesita recopilar manualmente la información de los precios.

Uso de cookies

Añadir cookies para almacenar el token de usuario en lugar de usar una store con Redux puede ser una posible mejora de seguridad para la aplicación web. Las cookies son pequeños archivos de texto que se almacenan en el navegador del usuario y se envían de vuelta al servidor en cada solicitud de página. Esto significa que el token de usuario no se almacena en el lado del cliente, a diferencia de la store, sino que se transmite de forma segura a través del protocolo HTTP.

Sin embargo, es importante señalar que las cookies también tienen sus desventajas. Por ejemplo, las cookies son limitadas en tamaño y no pueden almacenar grandes cantidades de datos. Además, si el navegador del usuario tiene las cookies deshabilitadas, la aplicación no funcionará correctamente.

Obtención de certificados web

Utilizar certificados SSL/TLS para proteger las comunicaciones entre el cliente y el servidor es una forma de mejorar la seguridad de la aplicación web. Los certificados SSL/TLS proporcionan una capa adicional de cifrado para garantizar que la información transmitida entre el cliente y el servidor sea privada y no pueda ser interceptada por terceros.

Además, los certificados SSL/TLS también proporcionan autenticación del servidor, lo que significa que el cliente puede verificar la identidad del servidor antes de enviar información confidencial. Esto ayuda a prevenir ataques de phishing y otros ataques de suplantación de identidad.

Implementar certificados SSL/TLS en la aplicación web es relativamente sencillo y puede ser realizado mediante el uso de un proveedor de certificados. Una vez que el certificado ha sido instalado, se puede configurar el servidor web para que use el certificado y se habilite el protocolo HTTPS para todas las solicitudes de la aplicación.

Integración de un chat

La integración de un chat en la aplicación sería una excelente forma de mejorar la comunicación entre los usuarios y facilitar los intercambios. Un sistema de chat permitiría a los usuarios comunicarse fácilmente antes y después de finalizar un intercambio, lo que ayudaría a aclarar cualquier duda o inquietud y asegurar que ambas partes están de acuerdo con los términos de la transacción.

La implementación de un sistema de chat puede ser más compleja que otras funcionalidades ya que requiere la creación de un sistema de chat desde cero o la integración de uno ya existente. Existen muchas opciones disponibles para la creación de un sistema de chat, como desarrollar uno propio utilizando React o integrar una plataforma de chat ya existente como Firebase o Pusher.

Envío de cartas

Finalmente, para brindar un servicio de intercambio de calidad se tendría que establecer una tercera parte que hiciera de intermediario, para confirmar que los productos de ambas partes son los que se han pactado, o bien se podría encontrar algún sistema complejo de compromiso y deber mediante contratos.

8. Referencias

[1] Pérez, E. (2021, April 21). *"Magic: The Gathering" es el juego más complejo del mundo, tanto que ni siquiera las máquinas saben cómo ganar.* Xataka. Último acceso 10/01/23, de https://www.xataka.com/literatura-comics-y-juegos/magic-the-gathering-juego-complejo-mundo-que-siquiera-maquinas-saben-como-ganar-2

[2] *Trafic análisis.* Similarweb. Último acceso 15/01/23, de https://www.similarweb.com/es/

[3] *Markrosewater.* Tumblr. Último acceso 17/01/23, de https://markrosewater.tumblr.com/post/110840728088/do-you-guys-have-any-data-on-the-breakdown-of-the

[4] Ysabela Golden. *Gendered Experiences in Magic: The Gathering*. Google Docs. Último acceso 17/01/23, de https://docs.google.com/document/d/1EkHMzGbrzCsfzuOgEQ_gH_iHXKo8FPQQ4wXLTevlUmQ/edit

[5] *[results] magic: The gatheringTM player survey*. Reddit. Último acceso 17/01/23, de https://www.reddit.com/r/magicTCG/comments/3u1evf/results_magic_the_gathering_player_survey/

[6] Apke, L. (2015). *Understanding the agile manifesto*. Libro

[7] Ken Schwaber and Jeff Sutherland (2017). *Scrum Guide.* Scrumguides. Último acceso 13/01/23, de https://scrumguides.org/scrum-guide.html

[8] David J. Anderson (2010). *Kanban: Successful Evolutionary Change for Your Technology Business.* Libro

[9] *Manage your team's projects from anywhere.* Trello. Último acceso 17/01/23, de http://trello.com

[10] Let's build from here. Github. Último acceso 13/01/23, de https://github.com/

[11] Martin, R. C. (2011). *Agile software development, principles, patterns, and practices: International edition.* Libro

[12] *Getting Started*. React. Último acceso 13/01/23, de https://reactjs.org/docs/getting-started.html

[13] *Introduction*. Vue. Último acceso 13/01/23, de https://vuejs.org/guide/introduction.html

[14] *The most demanded frontend frameworks in 2022*. Devjobsscanner. Último acceso 13/01/23, de https://www.devjobsscanner.com/blog/the-most-demanded-frontend-frameworks-in-2022/

[15] *Introduction.* React Hooks. Último acceso 13/01/23, de https://reactjs.org/docs/hooks-intro.html

[16] *Getting Started*. MUI. Último acceso 13/01/23, de https://mui.com/material-ui/getting-started/overview/

[17] *Introduction*. MUI. Último acceso 13/01/23, de https://mui.com/x/introduction/

[18] *Documentation*. Go. Último acceso 13/01/23, de https://go.dev/doc/

[19] *Introduction*. Visual Studio. Último acceso 13/01/23, de https://code.visualstudio.com/docs

[20] *Documentation*. MySQL. Último acceso 13/01/23, de https://dev.mysql.com/doc/

[21] *Documentation*. Heroku. Último acceso 13/01/23, de https://devcenter.heroku.com/categories/reference

[22] *Getting Started*. Postman. Último acceso 13/01/23, de https://learning.postman.com/docs/getting-started/introduction/

[23] *Documentation*. Scryfall. Último acceso 19/01/23, de https://scryfall.com/docs/api

[24] Richardson, L., & Ruby, S. (2008). *Restful Web Services*. Libro

[25] *Documentation*. Gin-gonic. Último acceso 13/01/23, de https://gin-gonic.com/docs/

[26] W3C. *Cross-Origin Resource Sharing*. Último acceso 17/01/23, de https://www.w3.org/TR/2020/SPSD-cors-20200602/
[27] Nasrul, S. (2021, June 26). *Create your first Go REST API with JWT Authentication in Gin Framework.* Medium. Último acceso 17/01/23, de https://seefnasrul.medium.com/create-your-first-go-rest-api-with-jwt-authentication-in-gin-framework-dbe5bda72817
[28] *Documentation*. GORM. Último acceso 20/01/23, de https://gorm.io/docs/index.html
[29] *Navisite DBaaS - The premier database-as-a-service for open-source & commercial databases*. Cleardb. Último acceso 17/01/23, de https://www.cleardb.com
[30] *Documentation*. Redux. Último acceso 17/01/23, de https://redux.js.org/api/store
[31] Xourse. (2017, January 20). *ReactJS app architecture - xourse.* Medium. Último acceso 17/01/23, de https://medium.com/@Xourse/reactjs-app-architecture-7a33d7ae9834
[32] *SweetAlert2 Documentation*. Sweet alert 2. Último acceso 17/01/23, de https://sweetalert2.github.io/
[33] *Documentation*. Apitest. Último acceso 17/01/23, de https://apitest.dev/
[34] *Documentation*. Npm. Último acceso 17/01/23, de https://docs.npmjs.com/
[35] *Documentation*. Jest. Último acceso 17/01/23, de https://jestjs.io/es-ES/docs/getting-started
[36] Cooper, A., Reimann, R., & Cronin, D. (2011). *About face 3: The essentials of interaction design* (3rd ed.) Libro

Anexo

[1] Pruebas de usabilidad

Nº de prueba	1	
Nombre	Oriol Maristany	
Experiencia del juego	10 años	
Edad	25 años	
Tarea(Caso de uso)	**Nº de clics**	**Tiempo**
Registro	2	14s
Login	2	3s
Modificar colección	9	43s
Buscar carta para intercanviar	4	22s
Realizar oferta de intercambio	13	35s
Finalizar intercambio	4	17s
Cerrar sessión	3	10s
Total	37	144s
Comentarios o errores		
Le ha costado encontrar como seleccionar la carta al realizar la oferta de intercambio.		

Nº de prueba	2	
Nombre	Albert Nadales	
Experiencia del juego	5 años	
Edad	22 años	
Tarea(Caso de uso)	**Nº de clics**	**Tiempo**
Registro	3	12s
Login	3	7s
Modificar colección	14	71s
Buscar carta para intercanviar	7	31s
Realizar oferta de intercambio	4	22s
Finalizar intercambio	4	15s
Cerrar sessión	3	12s
Total	38	170s
Comentarios o errores		
Se ha entretenido modificando su colección. No tiene claro cómo se usa el botón guardar colección.		

Nº de prueba	3	
Nombre	Daniel Vazquez	
Experiencia del juego	17 años	
Edad	32 años	
Tarea(Caso de uso)	**Nº de clics**	**Tiempo**
Registro	3	16s
Login	14	48s
Modificar colección	11	33s
Buscar carta para intercanviar	9	42s
Realizar oferta de intercambio	8	31s
Finalizar intercambio	3	13s
Cerrar sessión	4	6s
Total	52	189s
Comentarios o errores		
Le ha costado el Login pues puso una mayúscula en su contraseña sin darse cuenta. Le ha costado encontrar donde iniciar un intercambio. No tiene claro el funcionamiento de algunos botones.		