UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Alcoy

CREACIÓN Y DESARROLLO DE UNA APLICACIÓN MÓVIL DE FIDELIZACIÓN DE CLIENTES PARA SUPERMERCADOS

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Sanz García, Sergi

Tutor/a: Linares Pellicer, Jordi Joan

Cotutor/a: Esparza Peidro, Javier

CURSO ACADÉMICO: 2022/2023

CREACIÓN Y DESARROLLO DE UNA APLICACIÓN MÓVIL DE FIDELIZACIÓN DE CLIENTES PARA SUPERMERCADOS

RESUMEN

La industria del retail ha experimentado un cambio significativo en los últimos años, principalmente impulsado por el gran avance tecnológico y augmento de la competencia en el mercado. En este contexto, las empresas se encuentran con un problema que los puede llevar terribles consecuencias y se trata de la dificultad de incrementar las ventas y de la retención de los clientes.

Teniendo como objetivo incrementar las ventas mediante la fidelización de los clientes habituales y captación de otros nuevo, se va a desarrollar una aplicación móvil de fidelización de clientes para supermercados. La función principal es el uso de varios incentivos, como descuentos, promociones y ofertas personalizadas. Con esto conseguimos que cada cliente sienta un trato personal, de valor añadido y diferenciado, simplemente por tener acceso a esos privilegios que se le conceden por el uso de la misma, obteniendo así un cliente satisfecho y fiel.

El proyecto implica el análisis de las diferentes tecnologías y la elección de las que van a ser usadas, el despliegue de la aplicación en un servidor, la creación y definición de la base de datos y el desarrollo de una interfaz fluida y fácil de usar.

PALABRAS CLAVE

- Aplicación móvil
- Frontend
- Backend
- Bases de datos
- Ingeniería software

CREACIÓ I DESENVOLUPAMENT D'UNA APLICACIÓ MÒBIL DE FIDELITZACIÓ DE CLIENTS PER A SUPERMERCATS

RESUM

La indústria del retail ha experimentat un canvi significatiu en els últims anys, principalment impulsat pel gran avanç tecnològic i increment de la competència en el mercat. En aquest context, les empreses es troben amb un problema que els pot portar terribles conseqüències i es tracta de la dificultat d'incrementar les vendes i de la retenció dels clients.

Tenint com a objectiu incrementar les vendes mitjançant la fidelització dels clients habituals i captació d'altres nou, es desenvoluparà una aplicació mòbil de fidelització de clients per a supermercats. La funció principal és l'ús de diversos incentius, com a descomptes, promocions i ofertes personalitzades. Amb això aconseguim que cada client senta un tracte personal, de valor afegit i diferenciat, simplement per tindre accés a aqueixos privilegis que se li concedeixen per l'ús d'aquesta, obtenint així un client satisfet i fidel.

El projecte implica l'anàlisi de les diferents tecnologies i l'elecció de les quals seran usades, el desplegament de l'aplicació en un servidor, la creació i definició de la base de dades i el desenvolupament d'una interfície fluida i fàcil d'usar.

PARAULES CALU

- Aplicació mòbil
- Frontend
- Backend
- Bases de dades
- Enginyeria software

CREATION AND DEVELOPMENT OF A CUSTOMER LOYALTY MOBILE APPLICATION FOR SUPERMARKETS

ABSTRACT

The retail industry has undergone significant change in recent years, mainly driven by technological advances and increased competition in the market. In this context, companies are facing a problem that can lead to terrible consequences: the difficulty of increasing sales and customer retention.

Aiming to increase sales through the loyalty of regular customers and attracting new ones, a customer loyalty mobile application for supermarkets will be developed. The main function is the use of various incentives, such as discounts, promotions and personalized offers. With this we get each customer to feel a personal, value-added and differentiated treatment, simply by having access to these privileges that are granted by the use of it, thus obtaining a satisfied and loyal customer.

The project involves the analysis of the different technologies and the choice of those to be used, the deployment of the application on a server, the creation and definition of the database and the development of a fluid and easy to use interface.

KEY WORDS

- Mobile application
- Frontend
- Backend
- Databases
- Software engineering

ÍNDICE

ÍNDICE DE FIGURAS

GLOSARIO DE ACRÓNIMOS

- API (del inglés, Application Programming Interface)
- SDK (del inglés, Software Development Kit)
- JS (JavaScript)
- TS (TypeScript)
- JSX (JavaScript XML)
- TSX (TypeScript XML)
- HTML (del inglés, HyperText Markup Language)
- CSS (del inglés, Cascading Style Sheets)
- PHP (del inglés, Hypertext Preprocessor)
- SQL (del inglés, Structured Query Language)
- UI (del inglés, User Interface)
- DOM (del inglés, Document Object Model)
- REST (del inglés, Representational State Transfer)
- UUID (del inglés, Universally Unique Identifier)
- ERP (del inglés, Enterprise Resource Planning)
- URL (del inglés, Uniform Resource Locator)

1. INTRODUCCIÓN

1.1 Motivación

En los últimos años, se ha producido una notable transformación en la industria del retail, motivada principalmente por los grandes avances tecnológicos y el incremento de la competencia en el ámbito comercial. En esta situación, las empresas se enfrentan a un desafío que puede acarrear graves repercusiones: la dificultad para aumentar las ventas diferenciándose de sus competidores.

El aumento de la cantidad de clientes es la estrategia fundamental para impulsar el crecimiento de una cadena de supermercados. Aunque esta idea parece simple, en realidad representa un desafío considerable, especialmente para los nuevos establecimientos ya que la mayoría de las personas ya tienen su supermercado preferido. Superar esta lealtad arraigada en los consumidores y atraer su atención hacia nuevos supermercados requiere una ejecución de estrategias diferenciadoras y atractivas.

Es en este punto donde aparecen las aplicaciones de fidelización, cuya implementación se convierte en una herramienta crucial para atraer y retener clientes. Estas aplicaciones se diseñan para recompensar la fidelidad de los consumidores, ofreciendo beneficios y ventajas exclusivas que promueven una relación duradera con la marca.

Desde los supermercados Economy Cash[1], propiedad de la empresa para la cual desempeño mi labor como Desarrollador de Software, se ha detectado que un número significativo de clientes que acuden a sus tiendas durante las primeras semanas tras la apertura de un nuevo establecimiento, dejan de hacerlo poco tiempo después. Es por este motivo que se ha decidido desarrollar una aplicación de fidelización, y así retener a estos clientes iniciales e incluso llamar la atención de nuevos.

Figura 1: ***Logotipo de supermercados Economy Cash*** *[Fuente propia]*

[1] https://www.economycash.es/

1.2 Estudio del mercado

Este tipo de aplicaciones está bastante extendido entre los supermercados, teniendo la mayoría de ellas el mismo objetivo y las mismas funcionalidades, aunque con enfoques algo distintos. Entre estos supermercados se pueden resaltar Carrefour, Consum y Aldi.

Conocer la competencia es algo muy importante, y para ello se han descargado y probado cada una de las aplicaciones. A continuación, se exponen diferentes funcionalidades que podemos encontrar en cada una de ellas.

1.2.3 Carrefour

Una aplicación muy completa, llamada Mi Carrefour[2]. Ofrece promociones y cupones, posibilidad de seleccionar tienda favorita, apartado de gasolineras donde puedes consultar las diferentes estaciones de repostaje, tienda online, membresía de usuarios que incluyen ofertas adicionales por una mensualidad e incluso un apartado de juegos.

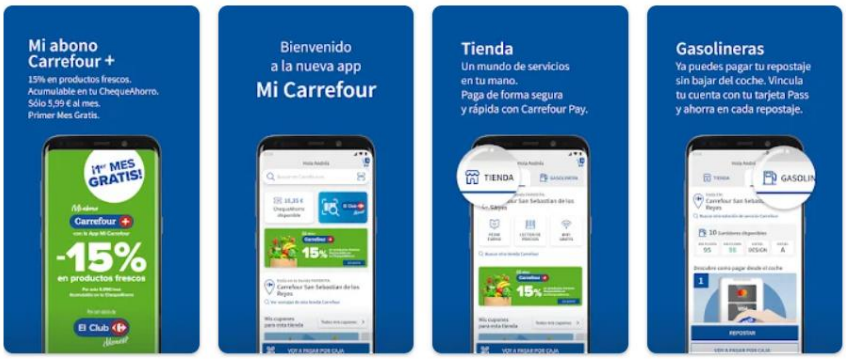

Figura 2: ***Aplicación de supermercados Carrefour*** *[Google Play]*

1.2.4 Consum

Se llama Mundo Consum[3] . No tiene tantas funcionalidades como la anterior, pero sigue más de cerca la sencillez. Ofrece funcionalidades como compra online, descuentos, acumulación de puntos, pedir turno en charcutería, carnicería o pescadería, lista de compra y consulta de tickets.

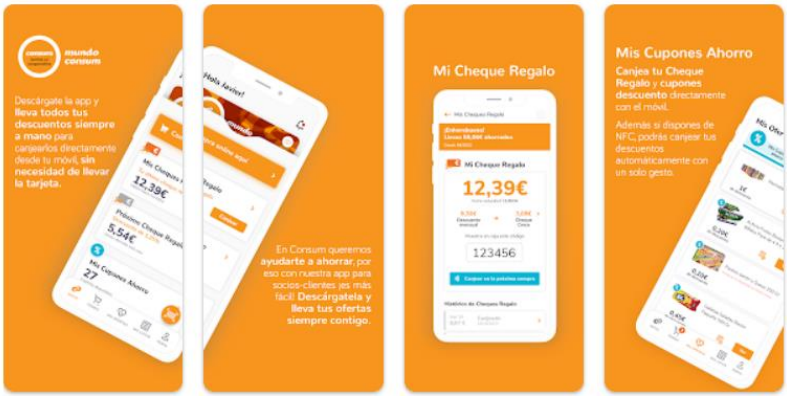

Figura 3: ***Aplicación supermercados Consum*** *[Google Play]*

[2] https://www.carrefour.es/clubcarrefour/app-mi-carrefour/
[3] https://www.consum.es/app-descuentos-ahorro

1.2.5 Aldi

La aplicación de Aldi[4] es quizás la más distinta, caracterizada por la sencillez y con funciones bastante más básicas entre las que se encuentran, entre otras, apartado de recetas, listado de compras o folleto de ofertas. En este caso no existe venta online, es más como un folleto convertido en aplicación.

Es la que más se aproxima a los estándares y objetivos que persigue Economy Cash.

Figura 4: ***Aplicación supermercados Aldi*** *[Google Play]*

1.3 Objetivos del proyecto

Este Trabajo Fin de Grado consiste en el desarrollo de una aplicación web de fidelización de clientes para los supermercados Economy Cash, dónde cada usuario tendrá acceso a diferentes incentivos como descuentos, promociones y ofertas personalizadas.

Se entiende como fidelización de clientes a la estrategia implementada por una empresa para mantener a sus clientes satisfechos y comprometidos con la marca de esta, creando así una relación duradera y sólida entre cliente y empresa.

Economy Cash persigue una filosofía de producto focalizada en la sencillez y minimalismo, sin funcionalidades innecesarias, evitando la complejidad y mostrando de forma clara toda la información, destacando lo más importante. De esta manera, se facilita al usuario el acceso a determinada información sin que le suponga un esfuerzo considerable.

Entre las funcionalidades más básicas y esenciales que suelen presentar este tipo de aplicaciones se encuentran: la acumulación de saldo o puntos, donde el usuario pueda canjear este importe a modo de descuento; promociones que se están aplicando; listado de establecimientos y más.

- Acumulación de saldo: el usuario acumula un saldo proporcional al importe de cada compra, pudiéndolo canjear a modo de cupón.

- Cupones canjeables: cada cliente tendrá unos cupones exclusivos que podrá canjear en las cajas.

[4] https://www.aldi.es/somos-aldi/conocenos/aldi-app.html

- Informar de promociones: la empresa debe poder mostrar promociones que estén activas.

- Mostrar los establecimientos existentes: el usuario tendrá acceso a un listado con todos los establecimientos Economy Cash.

- Consulta del ahorro acumulado: los clientes deben poder visualizar el ahorro que supone hacer uso de la aplicación.

- Consulta de histórico de compras: se debe mostrar un listado de compra diario.

La implementación de esta solución tecnológica es especialmente relevante, ya que, tras la aprobación del proyecto por parte de la empresa, tendrá un impacto real sobre esta al contribuir a fortalecer su posición competitiva en el mercado de supermercados, al mejorar la experiencia de compra de sus clientes y establecer relaciones comerciales sólidas.

A nivel personal, este proyecto ha sido todo un reto ya que he utilizado herramientas, tecnologías y lenguajes que no había usado anteriormente. También me he encargado de planificar, organizar y gestionar todo el desarrollo, e incluso convocar y liderar reuniones con compañeros de mi departamento y con otros departamentos como Marketing.

1.4 Estructura de la memoria

El resto del documento está organizado de la siguiente manera:

En el capítulo 2, se describe el problema a abordar en este proyecto de forma detallada, describiendo el motivo y causas por el que surge y el enfoque a seguir para resolverlo.

A continuación, en el capítulo 3, se presentan los detalles del diseño de la aplicación que se ha desarrollado para resolver el problema planteado en el apartado anterior. Se discuten aspectos fundamentales como la arquitectura de la aplicación, las tecnologías que se han elegido para el desarrollo del proyecto y las capas que lo componen.

En el capítulo 4, se realiza una exploración de la aplicación desarrollada, mediante explicaciones y capturas de pantalla, con el objetivo de presentar y demostrar el resultado alcanzado. Todas las capturas de pantalla irán acompañadas de una explicación de lo que podemos encontrar en cada una de ellas.

Por último, en el capítulo 5 se presentan las conclusiones obtenidas a partir del desarrollo del proyecto, así como las mejoras y las funcionalidades que se van a implantar en un futuro cercano.

2. ANÁLISIS

2.1 Contexto

Economy Cash presenta actualmente un total de 47 establecimientos, repartidos principalmente por el territorio de la Comunidad Valenciana, aunque también existen en la región de Murcia, Castilla-La Mancha y Tarragona.

Todos estos supermercados siguen la misma estructura en cuanto a comunicaciones. Cada tienda tiene un servidor que se comunica con central y con cada una de las cajas de la tienda. Estas cajas son gestionadas por una empresa externa, un punto clave para la implantación de la solución, es por ello que el primer paso que se debe abordar es conocer mediante reuniones los límites o requisitos que pueden aparecer por parte de esta empresa externa.

Conociendo estos puntos críticos que pueden aparecer y que habrá que gestionar en el momento de implantar el desarrollo, Economy Cash llega a un acuerdo con la empresa externa y se aprueba el desarrollo del proyecto.

En la *figura 5* se muestra un diagrama que representa las comunicaciones que se realizan a nivel general. El usuario utiliza la aplicación, que es alimentada por central. La aplicación se conecta con las cajas para poder identificar a los usuarios y así aplicar las promociones. Para ello la caja también se conectará con el servidor de la tienda, al cual tiene acceso central y la empresa externa que se encarga de las cajas.

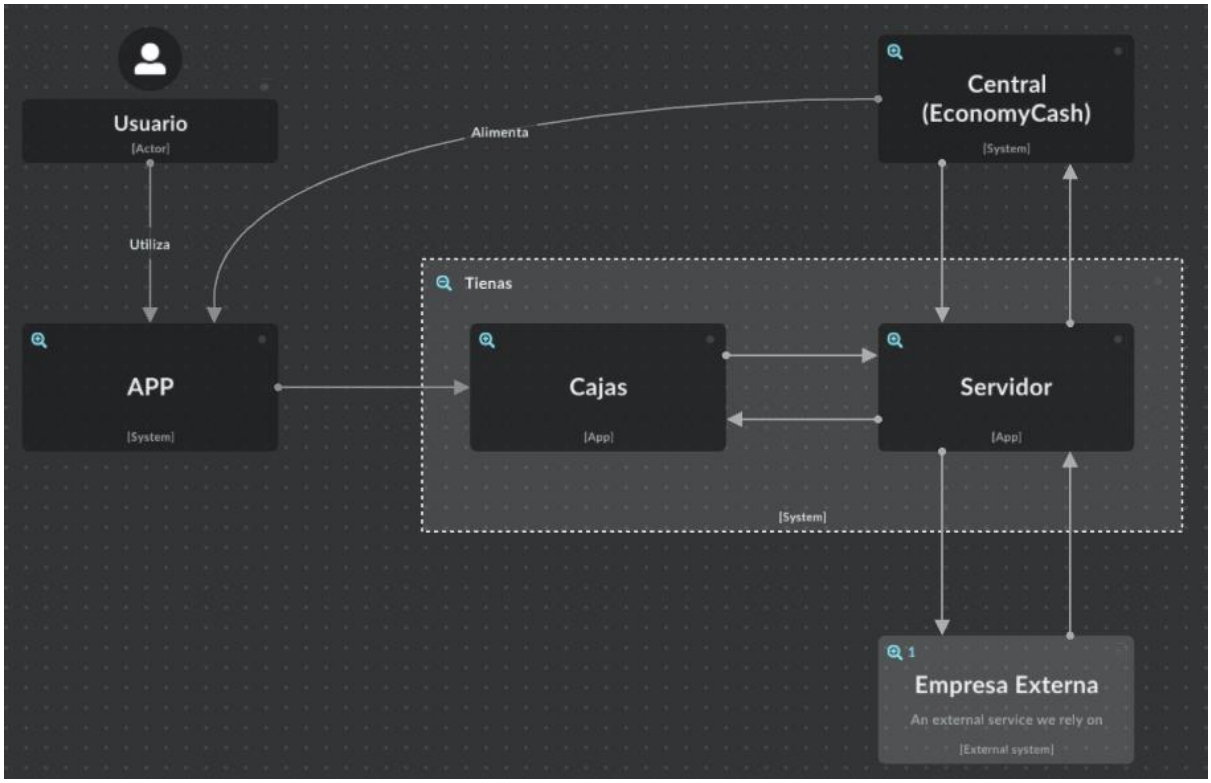

Figura 5: ***Diagrama de comunicaciones*** *[Fuente propia]*

2.2 Requisitos funcionales

Los requisitos funcionales mínimos que debe contener la aplicación quedan definidos en otra reunión con los responsables de los departamentos involucrados. Estos son:

- Uso de saldo acumulativo: El cliente debe tener un depósito donde se irá almacenando una cuantía monetaria de forma proporcional al importe de la compra. Este saldo se podrá canjear en cualquier caja de los establecimientos cuando el cliente lo considere, como si se tratase de un cupón de descuento. El importe descontado será igual al total del saldo acumulado o al total de la compra, es decir, el cliente no puede decidir qué cantidad se podrá descontar del total del ticket. Para canjear el saldo el cliente simplemente se tendrá que identificar como usuario en la caja.

- Cupones de descuento: Estos serán totalmente exclusivos para los clientes que usen la aplicación, ya que para poder canjearlos deben facilitar el código del cupón en la caja, y si este usuario tiene el cupón activado se aplicará. Estos cupones deben ser visibles mínimamente de dos maneras: una reducida, donde en un primer vistazo el usuario sea capaz de ver que le va a aportar el cupón; y una vista del detalle del cupón, donde se mostrará toda la información referente a este. Cuando un cliente se dé de alta, se le asignará automáticamente un cupón de descuento de bienvenida.

- Un apartado para promocionar cosas de interés: Debe existir una sección donde se pueda mostrar, en formado de banners, información que la empresa considere oportuna en ese momento. Puede ser un producto nuevo, productos que están en descuento o cualquier otro tipo de publicidad.

- Artículos en bajada de precio: Espacio dedicado a mostrar una cantidad reducida de productos que están en bajada de precio.

- Listado de tiendas: es necesario tener un lugar donde el usuario pueda consultar todas las tiendas existentes y poderlas localizar.

- Consulta del ahorro acumulado: debe existir un apartado dedicado a informal al usuario del ahorro que le está suponiendo para su economía el uso de esta aplicación. Se debe poder ver el ahorro acumulado durante el año, el ahorro del mes en curso y el ahorro por cada mes transcurrido del año. Este apartado puede ser clave para obtener clientes leales al supermercado, ya que el usuario es consciente en todo momento del ahorro que está consiguiendo y se verá motivado a seguir comprando en Economy Cash.

- Grupos de usuarios: la aplicación debe ser capaz de manejar grupos de usuarios. Estos se crear principalmente para asociar ciertas promociones a ciertos usuarios, una posible clasificación podría ser por la cantidad de compras mensuales.

En la *figura 6* se muestran las funcionalidades en modo de diagrama de casos de uso. Central representa a la empresa, puede tratarse tanto de un proceso automatizado como de una persona física, la tienda representa a todas las cajas que existen en esta y el usuario es el cliente que hace uso de la aplicación.

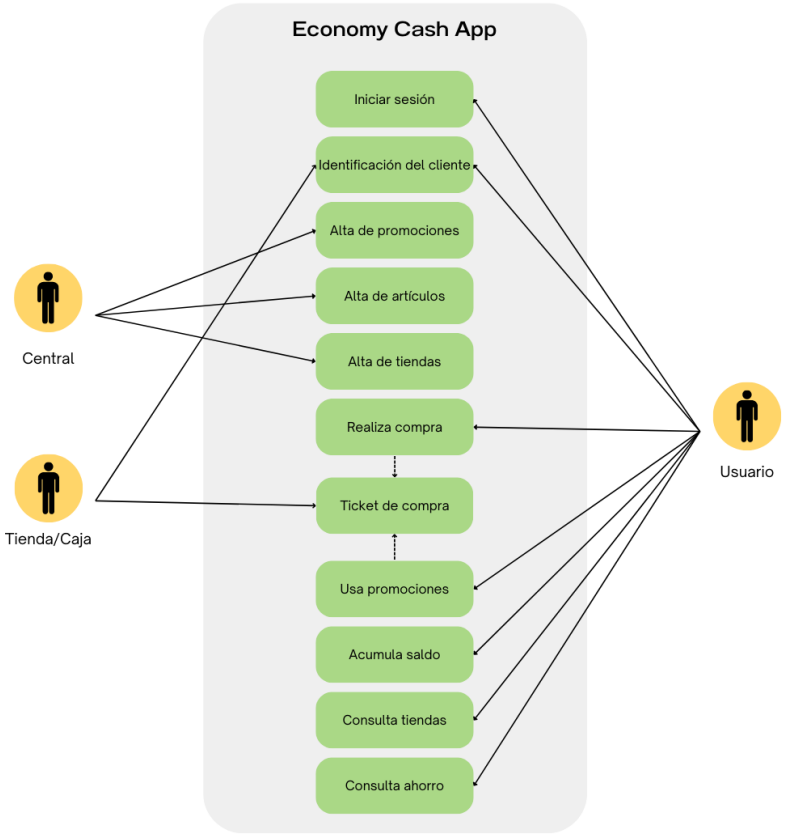

Figura 6: ***Diagrama de casos de uso*** *[Fuente propia]*

2.3 Requisitos no funcionales

La empresa no marca que lenguajes o tecnologías se deben utilizar para la implementación de esta aplicación, pero si quiere que se priorice el uso de aquellas herramientas que ya se hayan utilizado en proyectos anterior, para así partir de cierto conocimiento, agilizando así el proceso y para que se puedan incorporar en un futuro el resto de componentes del equipo de desarrollo.

Para poder llegar al máximo número de usuarios se requiere una aplicación multiplataforma, pudiendo llegar a usuarios de dispositivos Android y iOS.

Se necesita que la aplicación tenga un alto grado de disponibilidad. Para abordar este tema se tendrá que trabajar de forma conjunta con los administradores de sistemas. Pero a modo de resumen, la aplicación se desplegará en el servidor propio de la empresa, con esto se consigue una comunicación directa entre los administrados y el servidor, pudiendo dar soporte de forma inmediata a posibles caídas o errores.

La empresa de Economy Cash quiere una aplicación muy sencilla e intuitiva. El objetico es que el uso de la aplicación no suponga de un esfuerzo para el usuario. Esto se consigue reduciendo al máximo el número de pasos o pulsaciones que se tienen que hacer para obtener la información deseada. A demás, el aspecto de las vistas tiene que ser minimalista, mostrando solo información que es realmente relevante. Un exceso de información se puede transformar en una mala experiencia para el usuario. Se debe hacer uso de los colores que proporcione la empresa, que serán colores corporativos.

3. DISEÑO

3.1 Arquitectura de la aplicación

El proyecto se ha desarrollado utilizando una arquitectura de tres capas, un diseño software que separa la aplicación en tres capas encargadas de distintas funciones y que además se comunican entre sí. Esto permite lograr una solución más fácil de entender, mantener y escalar. Estas tres son:

- Capa de presentación, compuesta por los elementos con los que interactúa el usuario.
- Capa de lógica de negocio, encargada de procesar los datos, de definir las reglas de negocio y de realizar las funciones necesarias para cumplir con el propósito de la aplicación.
- Capa de datos, responsable de almacenar y gestionar los datos de la base de datos de la aplicación.

En los próximos apartados se entrará más en detalle en cada una de las capas, explicando cómo se aplica en la aplicación desarrollada.

En la *figura 7* se presenta un esquema de cómo es la arquitectura de tres capas. El componente que recibe el nombre de Economy Cash representa a todo el sistema fuera de la aplicación, donde se incluye la API de la empresa que será la encargada de comunicarse con la capa de negocios de la aplicación y así realizar consultas o modificaciones en la base de datos de la aplicación.

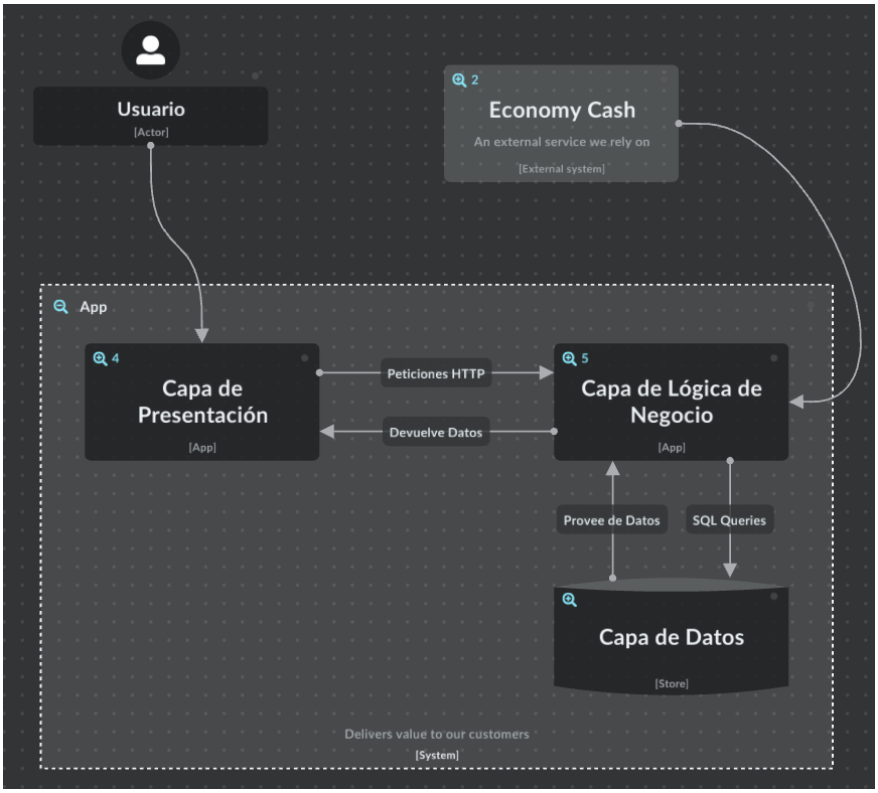

Figura 7: ***Representación de arquitectura de tres capas.*** *[Fuente propia]*

3.2 Tecnologías utilizadas

La selección de las herramientas y tecnologías que se utilizarán en el desarrollo de un proyecto es un proceso fundamental. Esta elección puede tener un impacto significativo en el éxito y la eficacia del proyecto. Para el desarrollo de la aplicación objeto de este trabajo, se ha realizado un análisis exhaustivo de las diferentes tecnologías disponibles, y se han seleccionado cuidadosamente las más adecuadas para alcanzar los objetivos planteados, teniendo en cuenta también el contexto tecnológico de la empresa.

En este apartado se exponen las diferentes tecnologías que se han seleccionado para el desarrollo de la aplicación.

Para el desarrollo de interfaz se ha tomado la decisión de utilizar Ionic[5],concretamente en su versión Ionic 7. Es un SDK de desarrollo de aplicaciones móviles que utiliza tecnologías web como HTML, CSS y JavaScript, permitiendo así crear aplicaciones multiplataforma. Este tipo de aplicaciones reciben el nombre de aplicaciones híbridas. Ionic es de código abierto, y destaca por tener una amplia biblioteca de componentes de interfaz de usuario que están optimizados para dispositivos móviles. Estos componentes son altamente personalizables y permiten construir ágilmente interfaces atractivas y muy consistentes.

A demás de Ionic se ha seleccionado React[6] en su versión 18.2 como biblioteca de JavaScript. React es conocido por su enfoque en la construcción de componentes reutilizables y su capacidad para manejar eficientemente los cambios de estado de la interfaz.

Esta biblioteca se basa en una serie de fundamentos que son clave para facilitar la creación de interfaces de usuario interactivas y eficientes. Estoy fundamentos incluyen:

- **Componentes**: al igual que Ionic, React también utiliza una arquitectura basada en componente, que representan de forma independiente cada parte de la interfaz de usuario. Permiten dividir la interfaz en piezas más pequeñas, facilitando el desarrollo, mantenimiento y la reutilización de código.

- **Propiedades**: se conocen como "props" y son atributos de configuración par el propio componente. Al realizar la instancia del componente se heredan desde un nivel superior. Estas propiedades son inmutables.

- **Estado**: en un componente es la representación de este mismo en un momento determinado, como una instancia del propio componente. Existen componentes con estado llamados "stateful" y sin estado llamados "stateless".

- **Virtual DOM**: React mantiene una representación virtual del DOM, en lugar de utilizar el DOM del navegador. Esto permite a React determinar qué partes del DOM han cambiado haciendo una comparación entre el contenido de la representación actual y el contenido de la nueva representación virtual.

- **React Hooks**: Proporcionan una forma de utilizar el estado y otras características de React en componentes funcionales sin tener que escribir una clase. Con el uso de Hooks se puede reutilizar la lógica de estado y efectos secundarios. Existen diferentes Hooks ya incorporados en React, entre ellos los más utilizados en el proyecto son: "useState", proporcionando una forma de declarar una variable de estado y una

[5] https://ionicframework.com/
[6] https://es.react.dev/

función para actualizarla; “useEffect”, permite realizar efectos secundarios en los componentes, utilizado en su mayoría para hacer llamadas a API.

- **TSX** (Type Script XML): Es una extensión del lenguaje de programación TypeScript, y son archivos que contienen código TypeScript junto a elementos de sintaxis JSX (JavaScript XML), una extensión de JavaScript que permite escribir elementos de interfaz de usuario utilizando una sintaxis muy similar a XML o HTML.

Con la combinación de Ionic y React se logra una sinergia muy consistente y poderosa, permitiendo desarrollar aplicaciones híbridas que se ejecutan de manera nativa en múltiples plataformas, como iOS y Android, todo a partir de una misma base de código, manteniendo al mismo tiempo una coherencia en la apariencia y experiencia del usuario.

Todo esto ha sido decisivo en la elección de estas tecnologías, ya que escribiendo el código una sola vez, se puede ejecutar en dispositivos de múltiples plataformas. Esto permite un desarrollo mucho más rápido y eficiente, ya que no es necesario desarrollar diferentes versiones de la aplicación para cada plataforma.

Figura 8: ***Logo Ionic***

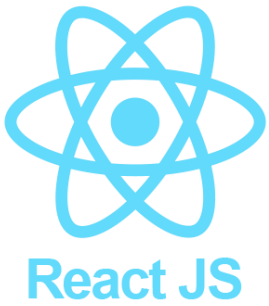

Figura 9: ***Logo React***

El lenguaje por el que se ha optado para programar sobre React ha sido TypeScript[7], un lenguaje fuertemente tipado [1] construido a un nivel superior que JavaScript. Dado que TypeScript es un superconjunto de JavaScript, todo código escrito con JS es válido también con TS. Sin embargo, no es válido en el sentido inverso.

En la *Figura 10* se expone de forma clara las principales diferencias que presentan estos dos lenguajes.

[7] https://www.typescriptlang.org/

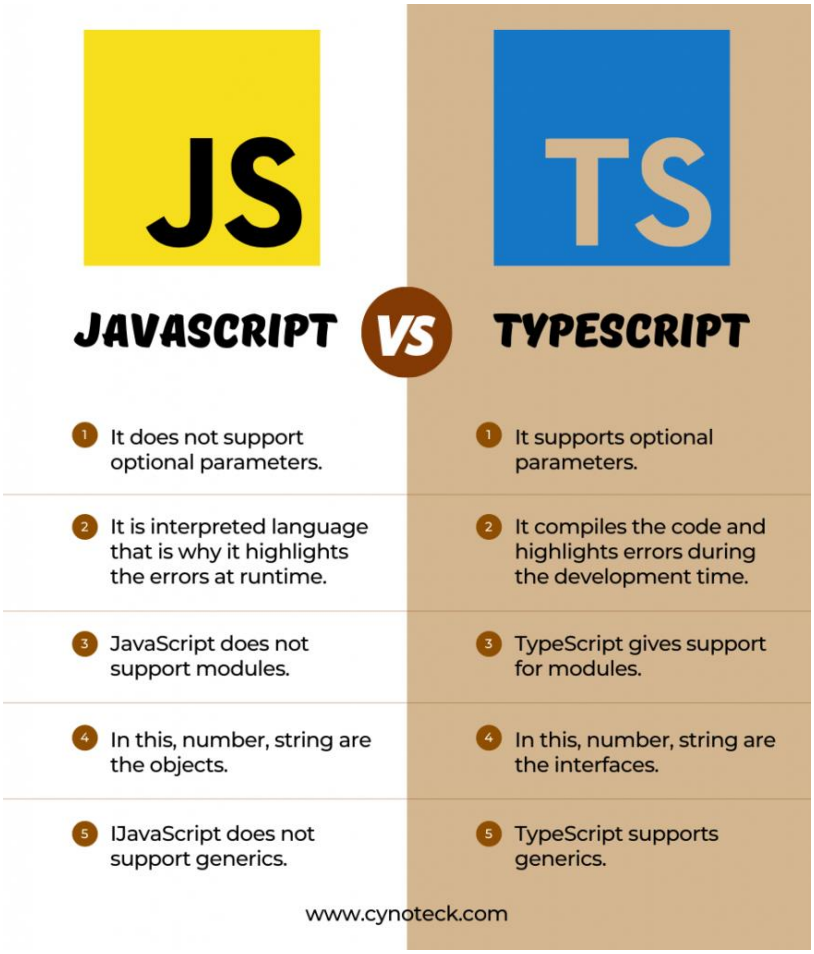

Figura 10: ***Diferencias entre JS y TS*** *[https://cynoteck.com/es/blog-post/typescript-vs-javascript-vs-ecmascript-know-the-difference/]*

Puesto que se trata de una aplicación web [2], los elementos React se complementan con código HTML, el cual se encarga de definir el significado y la estructura de una web mediante etiquetas o marcas. Estas permiten expresar diferentes partes de un documento, como la cabecera, el cuerpo, los encabezados, los párrafos, y otros elementos esenciales que conforman el contenido completo de una página web. Y complementando a este se usa CSS [3] que toma el control de la parte de presentación y del diseño visual, pudiendo establecer colores, fuentes, márgenes, tamaños y posiciones de los elementos HTML, permitiendo personalizar el aspecto estético de la aplicación web.

Figura 11: ***Logos HTML y CSS*** *[https://www.freepnglogos.com/pics/html5-logo]*

Para la capa de lógica de negocio se ha optado por Laravel[8], un framework de código abierto ampliamente reconocido que ha ganado popularidad en la comunidad de desarrollo web. Se destaca por su enfoque moderno y por tener potentes utilidades lo que permite a los desarrolladores agilizar la creación y el mantenimiento de aplicaciones web personalizadas de alta calidad. Con una arquitectura depurada y una amplia variedad de bibliotecas útiles, ofrece una base sólida para proyectos de cualquier tamaño, enfatizando en la cálida del código, la facilidad del mantenimiento y la escalabilidad. El lenguaje que se utiliza en este framework es PHP[9], destacado por ser un lenguaje de script el cual se ejecuta en el servidor, antes de enviar la página web al navegador del usuario. Esto permite que PHP interactúe con las bases de datos, maneje formularios o genere contenido dinámico.

Figura 12: ***Logo Laravel***

Figura 13: ***Logo PHP***

Estas herramientas comentadas se aplican usando un editor de código. Para el desarrollo de este proyecto se ha utilizado Visual Studio Code (VS Code), una herramienta muy popular y utilizada en el desarrollo de software debido a su gran versatilidad, rendimiento y un amplio catálogo de extensiones. El editor proporciona soporte integrado para un listado de lenguajes de programación muy extenso.

En cuanto a la implementación de la capa de datos, se ha optado por emplear una base de datos relacional, cuya estructura organiza los datos en tablas compuestas por filas y columnas. Cada una de estas filas en una tabla representa un registro de datos, y cada columna corresponde con un atributo específico de los mismos. Un elemento esencial en este tipo de base de datos son los valores únicos denominados "claves", los cuales se utilizan para identificar y relacionar los registros en distintas tablas. Estas bases de datos se caracterizan por su eficiencia para almacenar y recuperar datos, lo que resulta sumamente beneficioso para el funcionamiento óptimo del sistema en cuestión.

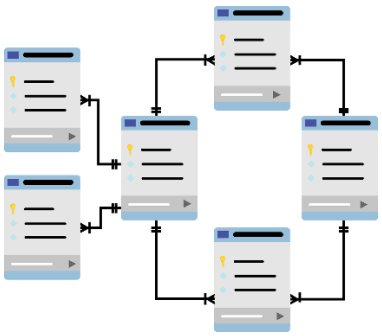

Figura 14: ***Representación de una base de datos relacional*** *[https://www.aluracursos.com/blog/base-de-datos-relacional]*

[8] https://laravel.com/
[9] https://www.php.net/

Estas bases de datos relacionales requieren de un sistema de gestión, y para el proyecto se ha seleccionado MariaDB[10]. Se caracteriza por ser un sistema de gestión de código abierto creado por los desarrolladores originales de MySQL y por ser compatible con muchos lenguajes de programación, como C, C#, Java, Python, PHP y Perl, y con los principales sistemas operativos incluyendo Windows, Linux y macOS.

Figura 15: ***Logo de MariaDB***

Figura 16: ***Logo de HeidiSQL***

Para administrar de forma gráfica nuestra base de datos MariaDB se ha utilizado HeidiSQL[11]. Es un administrador de bases de datos gráfico gratuito y de código abierto. Esta herramienta es muy poderosa y a la vez fácil de usar para administrar bases de datos, tablas, columnas, registros y usuarios. Además, permite ejecutar consultas, crear respaldos e incluso restaurar bases de datos. Las consultas se realizan utilizando el lenguaje SQL [5], un Lenguaje de Consulta Estructurada. Este tipo de lenguaje de programación permite tanto manipular como descargar datos de una base de datos, teniendo a su vez capacidad para hacer cálculos avanzada e incluso álgebra.

En todo proyecto de desarrollo software, es crucial tener un control sobre el código que se va desarrollando, y para ello se hace uso de la herramienta Git[12]. Es un sistema de control de versiones distribuida, que permite gestionar y mantener el histórico de cambios que se realizan en el código fuente del proyecto. Aunque Git es exprimido especialmente en proyectos colaborativos, también es útil y valioso para desarrolladores individuales, sobre todo si la complexidad del proyecto es considerablemente elevada, como es el caso.

Git es alojado por GitHub[13] un servicio basado en la nube esencial para aprovechar al máximo las capacidades de Git e incluso mejorar la gestión de proyectos de desarrollo de software de cualquier tamaño y naturaleza. Este permite a los desarrolladores rastrear y gestionar cambios de código de manera eficiente, agilizando así el desarrollo del proyecto.

Figura 17: ***Git y GitHub*** *[https://dev.to/suchintan/git-and-github-tutorial-beginner-to-advanced-part-1-32m6]*

[10] https://mariadb.org/
[11] https://www.heidisql.com/
[12] https://git-scm.com/
[13] https://github.com/

Otra herramienta que se ha utilizado en el proyecto para facilitar su desarrollo es Postman[14]. Es una plataforma gratuita para quienes trabajan individualmente, aunque sí tiene un coste adicional para trabajar de forma colaborativa. Esta herramienta facilita la creación, prueba, documentación y gestión de API. La principal función de Postman es que permite crear y enviar peticiones HTTP a cualquier API utilizando los diferentes métodos propios del protocolo HTTP como GET, POST, PUT, DELETE, etc., pudiendo configurar parámetros, cabeceras y datos de solicitud.

Figura 18: ***Logo de Postman***

Por la complejidad del proyecto y con el objetivo de tener una buena planificación del desarrollo se ha hecho uso de Jira[15], una plataforma de gestión de proyectos y seguimiento de problemas. Esta permite llevar una organización de hitos y de tareas a realizar de forma visual, pudiendo así ubicar temporalmente cada una de estas, ordenarlas por prioridad e incluso crear dependencias entre ellas.

Figura 19: ***Logo Jira***

Todo el proyecto se ha desarrollado de forma local en un servidor virtual Windows Server 2022 desplegado sobre el hardware propio de la empresa. Por motivos de seguridad y por ser un proyecto en fase de desarrollo, solo es posible acceder al servidor desde la red interna de la empresa.

[14] https://www.postman.com/
[15] https://www.atlassian.com/es/software/jira

3.3 Capa presentación

Esta capa también es conocida como capa de interfaz de usuario o UI. El propósito principal de esta capa es proporcionar una experiencia interactiva y amigable para que los usuarios puedan interactuar con las funcionalidades de la aplicación de una mamera efectiva.

La capa de presentación se encuentra en la parte superior de la arquitectura multicapa, y se comunica con la capa de lógica de negocio para obtener y presentar la información de una manera adecuada.

A continuación, se detallan las principales responsabilidades de la capa de presentación que se incluyen en el proyecto desarrollado:

- **Interacción con el usuario**: Proporciona una interfaz gráfica mediante la cual los usuarios pueden interactuar con el software, incluyendo elementos como botones, formularios o menús desplegables.

- **Visualización de datos**: Se encarga de mostrar la información de una manera comprensible y manteniendo una cierta estética. Se incluyen gráficos, listas o etiquetas con valores.

- **Gestión del flujo de navegación**: La capa de presentación asume la responsabilidad de cómo los usuarios se mueven dentro del software y que pueden realizar acciones en cada paso.

Como ya se ha indicado en el apartado anterior, la capa de interfaz de este proyecto se ha implementado mediante Ionic y React. Con la finalidad de adquirir los conocimientos necesarios para poder desarrollar una aplicación en React se ha realizado el curso "React - The Complete Guide 2023" [6] de Udemy[16].

Con esto ya se puede proceder a diseñar la interfaz de usuario. El primer paso que se ha seguido es definir el flujo de la aplicación. Este no tiene por qué ser el resultado final, pero es fundamental tener algo definido sobre lo que poder partir, sirviendo como objetivo al que llegar, punto de partida o plantilla.

La *figura 20* que se muestra a continuación corresponde al diagrama de flujo de la aplicación que se ha realizado para el proyecto.

[16] https://www.udemy.com/

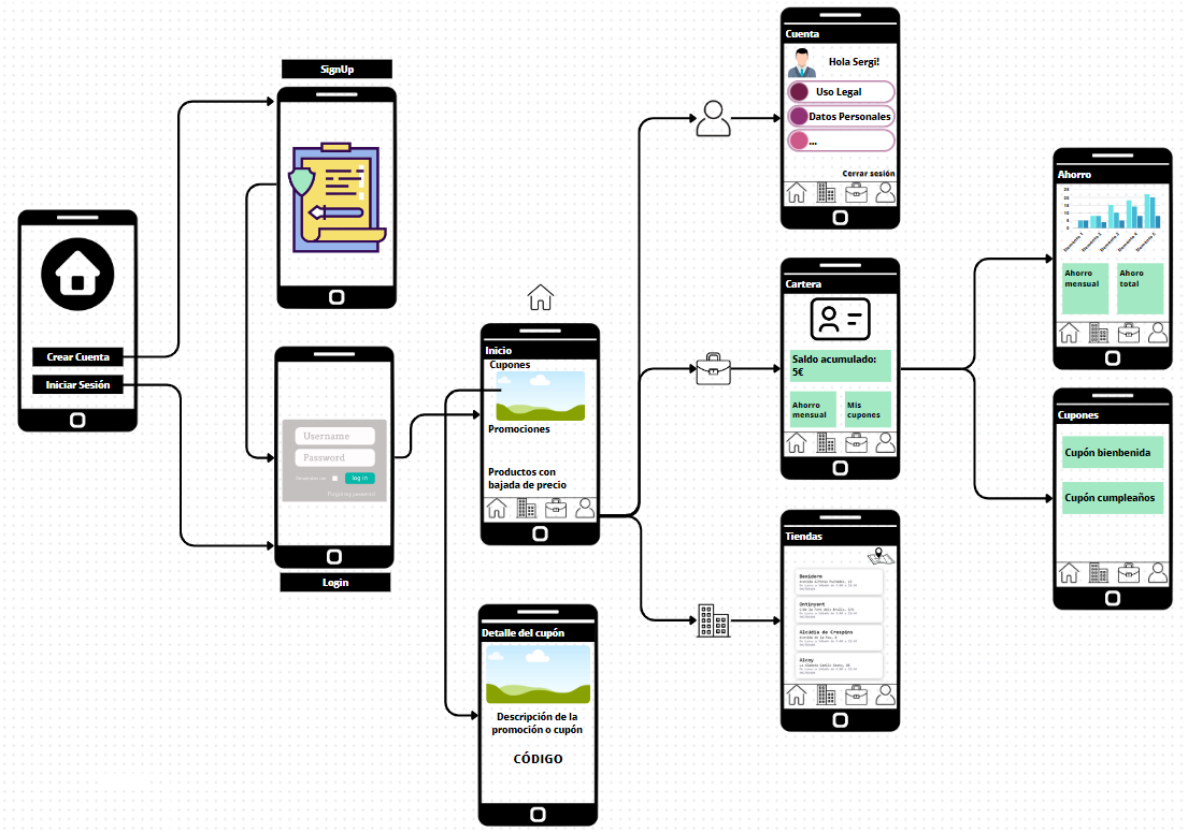

Figura 20: ***Diagrama de flujo de la aplicación*** *[Fuente propia]*

La aplicación parte de una primera vista, donde el usuario puede crear una cuenta o iniciar sesión. Tras el inicio de sesión se muestra la página "Inicio" en la que se distinguen tres apartados: cupones únicos para el usuario que podrá canjear en las cajas de las tiendas; promociones que están activas en las tiendas; productos que están en ese momento en bajada de precio. Desde aquí se puede navegar a otras tres páginas: un listado de las tiendas existentes, con información relevante de cada una; la cartera, donde el usuario puede consultar el saldo en su cuenta y tiene accesos a información adicional, como todos los cupones disponibles o el ahorro acumulado por usar la aplicación; el perfil del usuario, donde se pueden consultar datos personales, condiciones legales o cerrar sesión.

La interfaz de usuario de la aplicación ha sido creada a partir de este diagrama, y siguiendo la arquitectura basada en componentes de React se ha llegado a la estructura de directorios que se detalla a continuación.

Las carpetas principales son las presentadas en la *figura 21,* cuyo contenido se detallará a continuación.

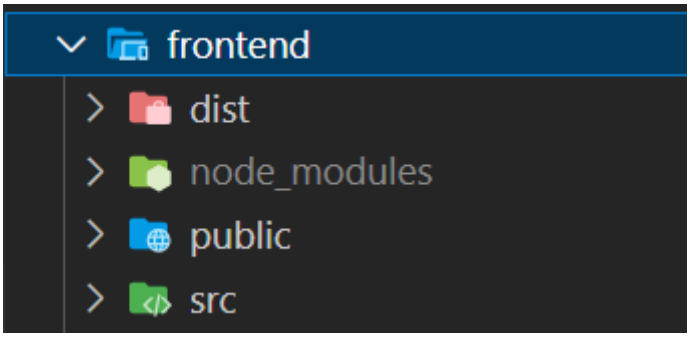

Figura 21: ***Estructura de directorios principal de la interfaz.***

La primera de ellas es “dist”, que contendrá los archivos finales generados tras el proceso de construcción, listos para ser desplegados en un servidor web o para su distribución. En este directorio de momento no se encuentran archivos relevantes.

En segundo lugar, se encuentra la carpeta “node_modules”. Esta, junto a todo su contenido es generada automáticamente, y contiene todas las dependencias de Node.js y paquetes externos necesarios para el correcto funcionamiento del proyecto. Todas las dependencias se incluyen en el archivo “package.json” y pueden ser, entre otras, bibliotecas de React, herramientas de compilación, plugins de Ionic, librerías de terceros y cualquier otro paquete que se requiera para la aplicación.

Seguidamente se sitúa la carpeta llamada “public”, destinada a almacenar recursos estáticos y públicos que deben estar accesibles directamente desde la raíz del servidor, sin un procesamiento adicional por parte de la aplicación. En esta carpeta se han dejado los archivos referentes a la tipografía de la aplicación en la carpeta “fonts”, y las imágenes que se utilizan en la interfaz en la carpeta “imgs”, evitando así ese procesamiento por parte de la aplicación.

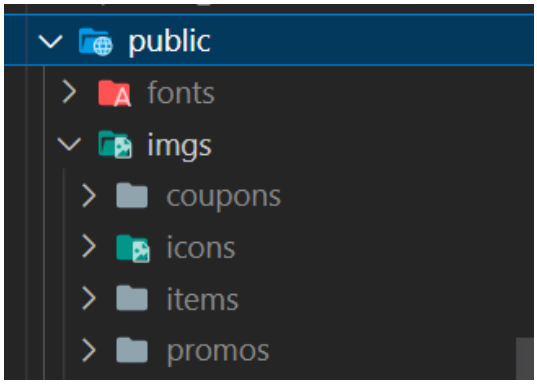

Figura 22: ***Estructura del directorio “public”***

Por último, se presenta la carpeta “src”. Es la más importante, ya que es donde está el código fuente principal de la aplicación. Dentro de esta carpeta se encuentran los archivos y subcarpetas mostradas en la *figura 23,* y se detallarán a continuación.

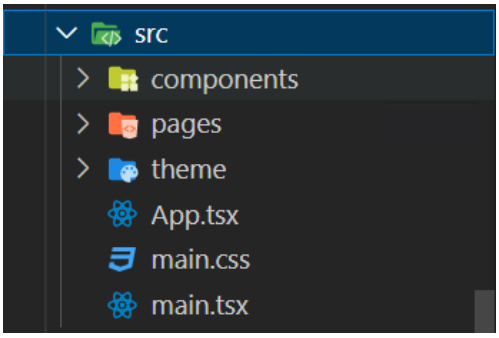

Figura 23: ***Estructura de directorio "src"***

El primer archivo mostrado es "App.tsx", el punto de entrada principal de la aplicación y lugar donde se define y se configura el componente raíz. El archivo raíz está envuelto por las configuraciones de enrutamiento y otras configuraciones globales. En el proyecto desarrollado también contiene lógica adicional, concretamente la encargada de permitir o no el cambio entre ciertas páginas de la aplicación dependiendo si el usuario ha iniciado sesión o no.

En el archivo "main.tsx" se importa el componente raíz de la aplicación ("App.tsx") y se lo renderiza al DOM. Se utiliza para inicializar y arrancar la aplicación. Este archivo también importa estilos globales, es por eso que existe la presencia del archivo "main.css".

A parte de estos archivos se encuentran tres carpetas. En la carpeta "theme" encontraremos archivos destinados para definir el estilo y la apariencia visual de la aplicación. En el caso del proyecto existe nicamente un archivo llamado "variables.css" el cual contiene variables de estilo que se utilizan en toda la aplicación.

La carpeta "pages" existe ya que se ha implementado una aplicación con múltiples páginas. Esta carpeta está compuesta por los componentes propios de cada una de las páginas acompañados por un archivo CSS que aplican estilo.

En la *figura 24* se muestran los diferentes archivos que forman la estructura de la carpeta "pages".

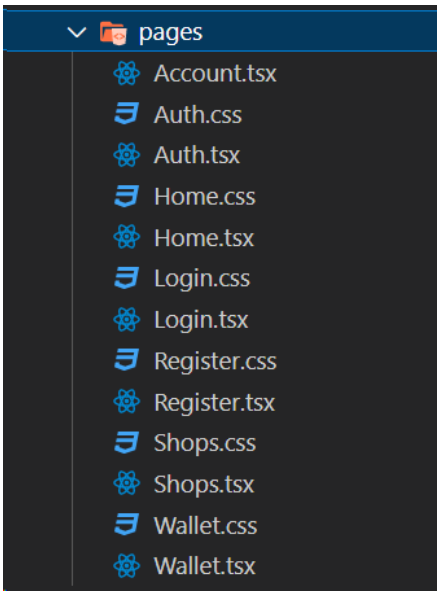

Figura 24: ***Contenido de la carpeta "pages"***

A continuación, se exponen los diferentes componentes que corresponden con cada una de las páginas y el contenido que se va a encontrar:

- **Auth.tsx**: Es la página que se muestra por defecto, y da a elegir entre dos botones, uno que permite crear una cuenta y otro que sirve para iniciar sesión.

- **Register.tsx**: Es la página que se mostrará tras pulsar el botón destinado para crear una nueva cuenta.

- **Login.tsx**: Página donde el cliente se identificará como usuario de la aplicación.

- **Register.tsx**: Página que permite a los clientes crear una nueva cuenta

- **Home.tsx:** Componente que se renderiza al iniciar sesión. La página se divide en tres secciones: la primera es un espacio destinado a los cupones que tiene disponible el cliente, el segundo es información que los supermercados consideren oportuno destacar, pudiendo ser desde una promoción hasta informar de un nuevo producto, y por último una sección donde se muestran productos que están en bajada de precio.

- **Shops.tsx**: Página donde se muestra en forma de listado de tarjetas todos los establecimientos existentes, pudiendo filtrarlos por provincias. Estas tarjetas contienen información relevante de cada una de las tiendas, como el nombre, la dirección o el horario de apertura.

- **Wallet.tsx**: Es quizás la página más compleja, ya que la profundidad del árbol de flujo aumenta. En esta página se tiene acceso a diferente información: una tarjeta del cliente donde se puede ver un código QR que servirá para identificarse en las tiendas; la cantidad de saldo que tiene disponible el usuario; información referente a la cantidad de ahorro que el cliente está consiguiendo; listado de todos los cupones que tiene disponibles; un sencillo historial de compra, en el que el usuario puede ver por día el valor de compra y el importe descontado mediante los descuentos.

- **Account.tsx**: La página más sencilla de todas, ya que se va a mostrar un menú con diferentes apartados, donde el único que se ha implantado es el de información del usuario. El resto de los apartados están destinados a incluir información como protección de datos, información legal o contenido que la empresa considere oportuno incluir.

A continuación, se muestra el código fuente de una de las páginas, concretamente el código que da pie al componente “Home.tsx”. Este componente es ideal como primer ejemplo, ya que es muy sencillo y se puede entender a la perfección el funcionamiento de los componentes.

```
import { IonContent, IonPage, IonLabel } from "@ionic/react";
import SliderPromos from "../components/Promos/SliderPromos";
import SliderCoupons from "../components/Promos/SliderCoupons";
import GridItems from "../components/Items/GridItems";
import AppInHeader from "../components/UI/HeaderIn";
import "./Home.css";
import "swiper/css";
import "swiper/css/pagination";
```

```
import "swiper/css/free-mode";
import "swiper/css/effect-cards";

const Home: React.FC = () => {
  return (
    <IonPage>
      <AppInHeader title="Promociones"></AppInHeader>
      <IonContent>
        <IonLabel className="items-title">Mis cupones</IonLabel>
        <SliderCoupons></SliderCoupons>
        <IonLabel className="items-title">Destacados</IonLabel>
        <SliderPromos></SliderPromos>
        <br />
        <IonLabel className="items-title">
          Productos en bajada de precio
        </IonLabel>
        <GridItems></GridItems>
      </IonContent>
    </IonPage>
  );
};


export default Home;
```

Se puede observar que las primeras líneas de código son destinadas para la importación de otros componentes o de estilos relacionados que se quieran usar. Seguidamente se crea el componente funcional "Home", que devuelve una estructura compuesta por otros componentes.

Por último, se presenta la carpeta nombrada como "components" que va a contener todos los componentes reutilizables que se utilizan en las diferentes partes de la aplicación. Como se ve en la *figura 25* estos componentes se organizan dentro de otras carpetas, consiguiendo así una estructura más organizada y coherente. Se puede ver la subcarpeta "Promos" desplegada, mostrándose como ejemplo de los componentes que se pueden encontrar en cada una de estas carpetas.

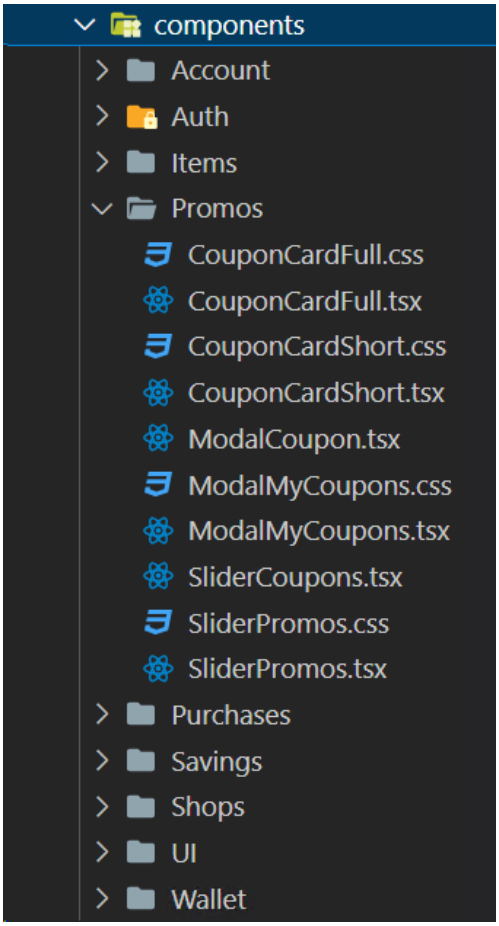

Figura 25: ***Contenido de la carpeta "components"***

Las carpetas creadas dentro de "components" son:

- **Account**: formado por los componentes que se utilizarán dentro de la página "Account".

- **Auth**: incluye los componentes que se encargan de la interfaz para que el usuario pueda crear una cuenta o iniciar sesión.

- **Items**: se sitúan los componentes que retornan contenido referente a los productos que se muestran en la página "Home"

- **Promos**: carpeta donde se encuentran los componentes relacionados con las promociones.

- **Purchases**: componentes usados para el histórico de compras.

- **Savings**: se encuentran los componentes referentes al ahorro del usuario.

- **Shops**: incluye aquellos componentes relacionados con el listado de tiendas.

- **UI**: carpeta donde se sitúan los componentes generales de la aplicación.

- **Wallet**: componentes relacionados con la página "Wallet".

Ya que todos los componentes siguen una estructura similar, a continuación, se muestra el código fuente de dos de ellos.

El primero de ellos es el componente llamado “HeaderIn.tsx” que se situado dentro de la carpeta “UI”. Este retorna la cabecera para las páginas de la aplicación una vez se ha iniciado sesión. Se ha seleccionado este ejemplo ya que su código fuente es muy sencillo, la reutilización de este se ve de forma muy clara y además es ideal para entender el uso de “props”.

```
import { IonHeader, IonItemDivider, IonTitle, IonToolbar } from
"@ionic/react";
import "./Header.css";

const AppInHeader = (props: any) => {
  return (
    <IonHeader>
      <IonToolbar className="header__toolbar">
        <IonTitle>{props.title}</IonTitle>
      </IonToolbar>
      <IonItemDivider className="header__divider"></IonItemDivider>
    </IonHeader>
  );
};

export default AppInHeader;
```

Este componente se utiliza en cada una de las páginas, ya que todas incluyen la misma cabecera, pero con un título diferente. En lugar de escribir este código cuando se desee mostrar la cabecera, con su correspondiente hoja de estilos, hay que importar y hacer referencia al componente, reduciéndolo así a tan solo dos líneas.

Para poder mostrar el título correcto que corresponde con la página mostrada se resuelve con el uso de los “props”. Para ello, desde un componente padre, hay que referenciar al componente hijo pasándole el parámetro en cuestión, y así desde el componente hijo tendremos acceso al valor del parámetro que le ha llegado desde el componente padre. Este comportamiento se puede ver en el primer ejemplo, el cual se referencia al componente hijo “AppInHeader”.

El segundo ejemplo es el componente “SliderPromos.tsx” dentro de la carpeta “Promos”. Este componente se encarga de construir un carrusel donde aparecen las imágenes de las promociones. El código fuente se presenta a continuación

```
import React, { useEffect, useState } from "react";
import { IonImg, IonList, IonListHeader, IonSkeletonText } from
"@ionic/react";
import axios from "axios";
import "./SliderPromos.css";

const SliderPromos = () => {
  const [error, setError] = useState(null);
```

```
  const [isLoaded, setIsLoaded] = useState(false);
  const [items, setItems] = useState([]);
  const token = sessionStorage.getItem("token");
  const userId = sessionStorage.getItem("userId")

  useEffect(() => {
    axios
      .get("http://127.0.0.1:8000/api/getPromos", {
        headers: {
          Authorization: `Bearer ${token}`,
        },

      })
      .then((response) => {
        setIsLoaded(true);
        setItems(response.data);
      })
      .catch((error) => {
        setError(error);
      });
  }, []);

  if (error) {
    return <div>Error: {error}</div>;
  } else if (!isLoaded) {
    return (
      <IonList>
        <IonListHeader>
          <IonSkeletonText
            animated={true}
            style={{ width: "95%", height: "150px" }}
          ></IonSkeletonText>
        </IonListHeader>
      </IonList>
    );
  } else {
    let promosContent = items.map((promo: any) => (
      <IonImg
        src={"imgs/promos/" + promo.image_name}
        key={promo.id}
        className="promo-image"
      ></IonImg>
    ));
    return <div className="promos-container">{promosContent}</div>;
  }
};
export default SliderPromos;
```

En este código fuente aparecen tres nuevos conceptos que no se han visto en los ejemplos anteriores.

El primero que se va a comentar es “useState”, un Hook que permite añadir el estado de React al componente funcional pudiendo actualizar el estado de una variable. En el código se usa por ejemplo para almacenar en una variable el valor obtenido tras una petición HTTP. En este caso la variable toma el nombre de “items” y su valor se actualiza mediante “setItems”.

La segunda es la realización de una petición HTTP. Para ello se usa Axios, una biblioteca de cliente HTTP que permite realizar solicitudes a la API del proyecto de forma fácil e intuitiva. En este caso se trata de una petición “get” que contiene una cabecera donde el usuario se autentica contra la API mediante su token de sesión generado por la lógica de negocio.

Esta petición a la capa de negocio es rodeada por “useEffect”. Es un Hook de React que se ejecuta después de que el componente se haya renderizado, y permite realizar efectos secundarios en componentes funcionales, como en este caso realizar una solicitud a la API.

Para el uso de los componentes de Ionic se ha hecho uso de la documentación oficial [7], en la que se exponen todos los componentes disponibles y como hacer uso de ellos.

Para implementar las hojas de estilo de cada uno de los componentes ha sido muy útil la web de W3Schools [8], en la que hay todo un apartado dedicado a tutoriales de CSS.

3.4 Capa lógica de negocio

Esta es la campa situada en la parte central de la aplicación, y se comunica con la capa de presentación y con la capa de datos. Se encarga de tomar decisiones basadas en las reglas establecidas por el negocio y manipula los datos para asegurarse de que las operaciones se realicen siguiendo las políticas y procedimientos establecidos. Es responsable de procesar y coordinar las solicitudes de los usuarios, realizadas desde la capa de presentación, y de acceder a la capa de datos para obtener o almacenar información y producir así los resultados adecuados.

Está implementada con el framework de código abierto Laravel, que usa PHP como lenguaje de programación. La versión utilizada es la última disponible, Laravel 10 que requiere mínimo la versión 8.1 de PHP. A pesar de presentar la capacidad de renderizar vistas, para este proyecto se ha decido utilizar esta herramienta exclusivamente para la implantación de la capa de negocio y el desarrollo de la API.

A continuación, se presentan algunos detalles sobre la estructura de archivos que sigue Laravel y se han aplicado en el proyecto:

- **Controladores**: se sitúan en el directorio “app/Http/Controllers” y son los responsables de manejar las solicitudes HTTP y coordinar la lógica de negocio. Cada uno de estos controladores contiene el código fuente necesario para obtener el resultado deseado, como obtener información o crear nueva información.

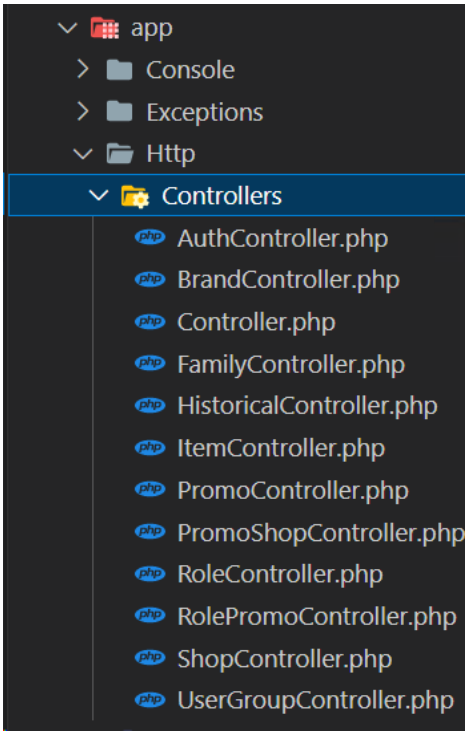

Figura 26: ***Estructura de archivos de los controladores***

- **Rutas:** se utilizan para relacionar cada petición HTTP con el controlador y función correspondiente. En el caso de este proyecto las rutas están definidas en el archivo "routes/api.php" que son las rutas de la API, y están protegidas para que solo puedan hacer uso de ellas los usuarios que han iniciado sesión. Puesto que el encargado de autenticar a los usuarios en esta capa es Sanctum (componente de Laravel), para proteger las rutas se debe hacer uso del middleware "auth:Sanctum", que verificará si el usuario está autenticando y tiene un token válido para poder acceder a la ruta. A continuación, se muestra un ejemplo de cómo se deben usar las rutas para manejar información referente a las tiendas y que además están protegidas:

```
Route::middleware(['auth:sanctum'])->group(function () {
    //Shops
    Route::get('/shops/{shop}', [ShopController::class, 'show']);
    Route::post('/shops', [ShopController::class, 'store']);
    Route::put('/shops/{shop}', [ShopController::class, 'update']);
    Route::delete('/shops/{shop}', [ShopController::class, 'delete']);
});
```

- **Modelos:** En Laravel los modelos se encuentran en el directorio "app/Models", tal y como se muestra en la *figura 27* y representan las tablas de la base de datos, utilizándose para interactuar con estas.

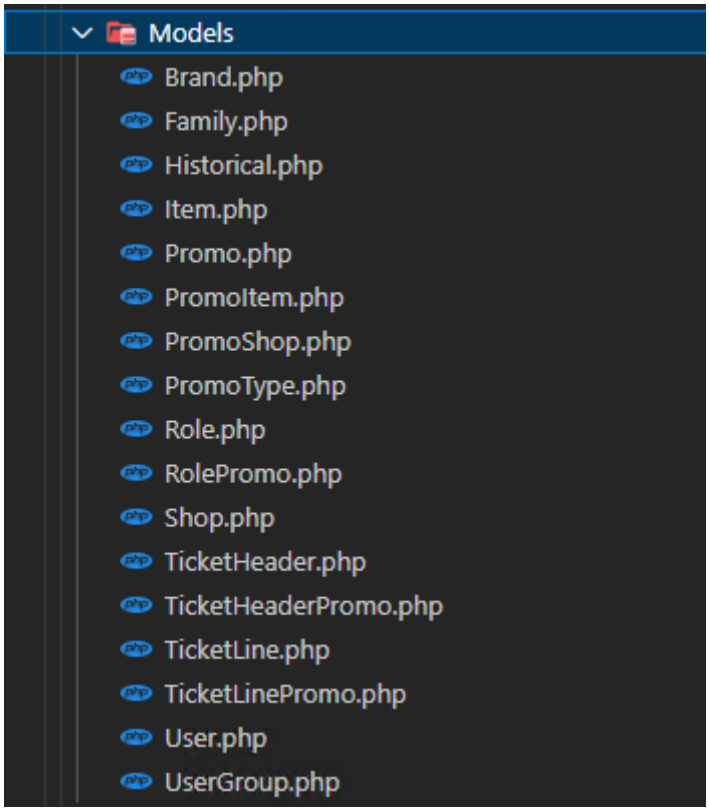

Figura 27: ***Listado de modelos*** *[Fuente propia]*

A demás también contienen la lógica que definen la relaciones que existen entre los diferentes modelos. Seguidamente se muestra un ejemplo de un modelo en el proyecto desarrollado. Este es el modelo de las promociones ("Promo.php") y en él se cambian las propiedades del identificador del modelo, esto se hace porque se va a hacer uso de UUID en lugar de un valor incremental, por eso se deshabilita la opción de incremento y se cambia el tipo de dato a texto. Este UUID se genera de forma automática haciendo uso de la función que sigue. A continuación, se define el atributo "fillable", una propiedad que se utiliza para especificar que atributos de un modelo pueden ser utilizados en la creación o actualización de registros en la base de datos. Por último, se presentan las relaciones que existen con otros modelos:

```php
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Ramsey\Uuid\Uuid;

class Promo extends Model
{
    use HasFactory;
    public $incrementing = false;
    protected $keyType = 'string';
```

```
    // ----- Generate UUID ----- //
    protected static function boot()
    {
        parent::boot();
        static::creating(function ($model) {
            $model->id = Uuid::uuid4()->toString();
        });
    }

    protected $fillable = [
        'code', 'title', 'subtitle', 'description', 'image_name', 'type',
'value', 'value_unit', 'status', 'start_date', 'end_date'
    ];

    //  |promos| 1 ------ 0..n |promo_shops|
    //  Relation one to many with promo_shops
    public function promoShop(): HasMany
    {
        return $this->hasMany(PromoShop::class);
    }

    //  |promos| 1 ------ 0..n |ticket_header_promos|
    //  Relation one to many with ticket_header_promos
    public function ticketHeaderPromo(): HasMany
    {
        return $this->hasMany(TicketHeaderPromo::class);
    }

    //  |promos| 1 ------ 0..n |ticket_line_promos|
    //  Relation one to many with ticket_line_promos
    public function ticketLinePromo(): HasMany
    {
        return $this->hasMany(TicketLinePromo::class);
    }

    //  |promos| 1 ------ 0..n |promo_items|
    //  Relation one to many with promo_items
    public function promoItem(): HasMany
    {
        return $this->hasMany(PromoItem::class);
    }
}
```

- **Validación de datos:** estas validaciones se pueden utilizar desde los mismos controladores, con el objetivo de validar los datos enviados por la capa de

presentación. Si la validación falla, Laravel responde con el mensaje de error correspondiente de forma automática.

Ya que la capa de lógica de negocios se sitúa en el centro de la arquitectura, esta debe poder comunicarse con los diferentes sistemas, es por ello que se hace uso de una API.

Una API es un conjunto de reglas, protocolos y herramientas que permiten que diferentes sistemas puedan interactuar entre sí. Funciona como un intermediario que facilita el intercambio de datos o funcionalidades entre diferentes componentes software. El uso de este conjunte de mecanismos ofrece diferentes beneficios [9]:

- **Integración**: Las API son empleadas para integrar nuevas aplicaciones con sistemas ya existentes. Con esto se acelera el proceso de desarrollo, ya que no es necesario crear las funcionalidades desde cero, sino que se pueden aprovechar las funcionalidades existentes en las API.

- **Innovación**: La llegada de una aplicación puede provocar cambios significativos en sectores enteros, es por ello por lo que las empresas deben adaptarse ágilmente para respaldar la rápida implementación de servicios innovadores. Para lograrlo, tienen la opción de realizar modificaciones en la API sin la necesidad de reescribir todo el código.

- **Ampliación**: Las API brindan a las empresas una oportunidad única para atender responder a las necesidades de sus clientes en diversas plataformas. Al proporcionar acceso a funciones específicas o datos internos, facilitan la integración y la mejoran la experiencia del usuario.

- **Facilidad de mantenimiento**: Una API actúa como puente de enlace entre dos sistemas diferentes, los cuales tendrán que realizar cambios internos para evitar que la API se vea afectada. De esta forma si una parte realiza algún cambio en el código, no afectará a la otra.

Concretamente se ha implementado una API RESTful, un tipo concreto de API que sigue los principios de arquitectura REST, que se basa en el protocolo HTTP.

Para la API se han implementado las llamadas estándares para cada modelo. Estas son:

- **Obtener toda la información**. Se hace uso de la clase '"index" del controlador del modelo, y esta petición retorna la información de un modelo. Por ejemplo si hacemos esta llamada para las tiendas, obtendremos como resultado una respuesta en formato JSON con todos los registros de la tabla. Para esto se utiliza la función GET del protocolo HTTP.

- **Obtener información filtrada**. Se obtiene mediante la clase "show" del controlador del modelo, y a partir de unos valores de entrada se filtrarán los datos para así obtener la respuesta deseada. También se hace uso de GET.

- **Crear nuevo registro**. La clase encargada de esto es la llamada "store", situada en el controlador del modelo, y permite crear nuevas entradas de datos en una tabla. Los datos a crear llegan a modo de parámetros, que se utilizarán para crear el nuevo registro. Se hace uso de la función POST del protocolo HTTP. En el *Anexo 1* se muestra como ejemplo el código fuente que se utiliza para crear una nueva tienda.

- **Actualizar registros**. Mediante la clase del controlador del modelo "update" se pueden actualizar datos ya existentes. De debe pasar como parámetro el registro que se va a actualizar y los nuevos datos. Se utiliza la función PUT del protocolo HTTP.

- **Eliminar registros**. Para eliminar un registro existente se hace uso de la clase "destroy" del controlador del modelo. Para poder eliminar el registro se debe hacer llegar a la clase el registro que se desea eliminar. Se utiliza la función DELETE del protocolo HTTP.

A parte de estas llamadas estandarizadas también se crean otras personalizada, como por ejemplo la llamada para obtener las promociones que tiene activas un cliente. Esta utiliza una clase personalizada situada en el controlador del modelo. La estructura de esta clase es algo más compleja, ya que se deben cruzar varias tablas para obtener el resultado deseado. Como ejemplo, en el *Anexo 2* se comparte el código fuente de la clase que recupera los cupones de un usuario.

Para interactuar con la API y realizar las pruebas correspondientes se utiliza la herramienta Postman, una aplicación de escritorio que permite simplificar el proceso de desarrollo, prueba y documentación de la API.

En la *figura 28* se muestra una captura de Postman, en la que se ve una estructura de carpetas, donde cada una de ellas se refiere a una identidad de la base de datos, conteniendo cada una de estas las funciones necesarias. Se puede ver que en "Brands" existe un archivo para crear, mostrar, actualizar y eliminar registros referentes a las marcas de los productos.

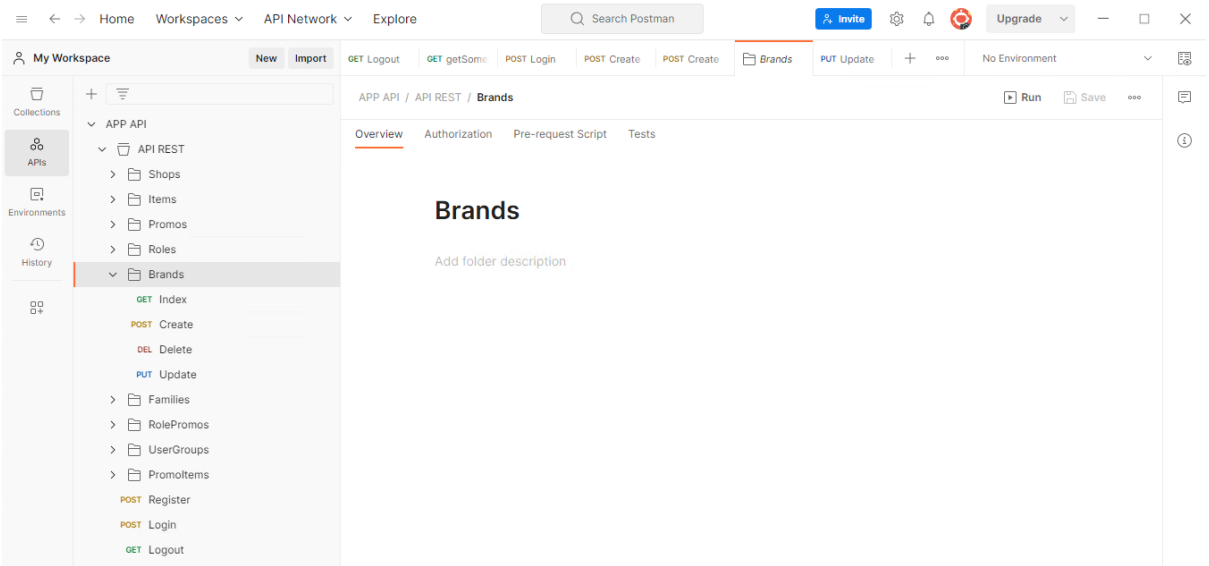

Figura 28: ***Captura de pantalla de Postman*** *[Fuente propia]*

3.5 Capa de datos

La capa de datos es la responsable de interactuar con la fuente de datos pudiendo recuperar, almacenar, actualizar o eliminar la información que sea necesaria para el funcionamiento correcto de la aplicación.

La implantación de este proyecto también ha implicado la creación de esta fuente de datos. Se trata de una base de datos relacional [10], que como se ha explicado en el apartado Tecnologías utilizadas, son aquellas bases de datos que se organizan en tablas con filas y columnas y se relacionan entre sí mediante claves.

Para llegar a esta base de datos se ha partido de un diagrama conceptual, clave para obtener una visión clara de la organización de los datos y definir como se relacionan entre sí. El diagrama servirá como guía para llegar al diseño final de la base de datos.

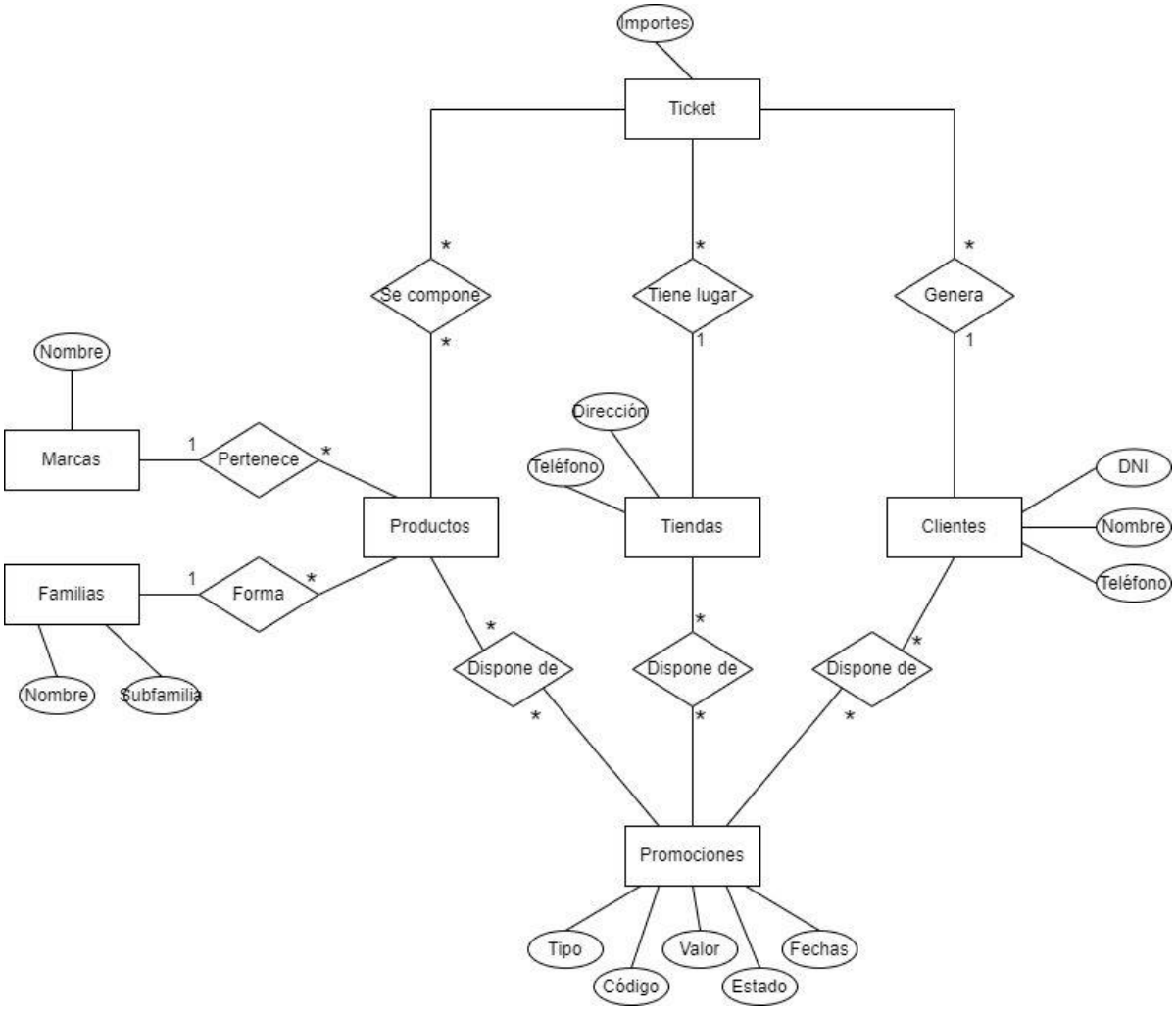

Figura 29: ***Diagrama conceptual de la base de datos*** *[Fuente propia]*

En la *figura 29* se puede ver el diagrama y este se puede explicar de la siguiente manera:

- Un ticket se compone de muchos productos, tiene lugar en una tienda y es generado por un cliente.

- Un producto puede componer varios tickets, pertenece a una marca, pertenece a una familia y puede disponer de una promoción.

- Una tienda puede producir muchos tickets y puede disponer de muchas promociones.

- Un cliente puede generar muchos tickets y puede disponer de muchas promociones.

Partiendo de este esquema conceptual, se ha llegado al modelo físico de datos que se presenta en la *figura 30*. Un modelo físico es una representación detallada de la base de datos y define la estructura concreta de las tablas, el tipo de datos de cada campo, las claves primarias, las claves foráneas, los índices y el tipo de relación que une las diferentes tablas.

Se ha perseguido una base de datos normalizada, con el objetivo de obtener un diseño eficiente y de evitar la redundancia de datos.

Para la nomenclatura de la base de datos se ha utilizado, tanto en el nombre de las tablas como en el nombre de las columnas, el estilo de escritura "snake case" que consiste en separar las palabras con el uso del guion bajo. Este tema se discute en una infinidad de blogs y artículos, donde cada uno defiende un estilo de nomenclatura diferente, pero en el caso de este proyecto se ha decidido seguir este estilo ya que es el que se utiliza en el resto de las bases de datos de la empresa.

También se ha decidido utilizar UUID para el campo que identifica cada una de las tablas. Es un tipo de identificador único de 128 bits que se representa como una cadena de caracteres hexadecimales de 32 dígitos y separados por guiones. El uso de estos identificadores en lugar del típico entero incremental es tema de gustos, pero es interesante, ya que se evita la dependencia de la base de datos para generar un valor incremental y, además de eliminarlo, se facilita la escalabilidad del problema reduciendo los cuellos de botella relacionados con la generación de identificadores.

Del diagrama físico presentado se sacan dos conceptos diferentes de tablas. El primero de ellos son las tablas fundamentales, compuestas por:

- **ticket_headers**: almacena los datos referentes a la cabecera de los tickets. Con columnas como el total del importe del ticket, el usuario que realiza la compra o la tienda donde se ha comprado.

- **ticket_lines**: representa las líneas del ticket. Algunos de campos que contiene son el artículo, el importe por línea o la cantidad de producto comprado.

- **shops**: almacena los establecimientos. Con columnas como el nombre de la tienda, el horario y la dirección.

- **promos**: son las promociones que existen o han existido. Contiene columnas como el título de la oferta, el importe de descuento, el tipo de descuento o la descripción de la promoción.

- **items**: donde se almacena la información de los artículos. Presenta campos como el nombre del artículo o el precio.

- **families**: son las familias de los artículos. Dentro de esta tabla se identifica el código de la familia, el código de la subfamilia y sus respectivos nombres.

- **brands**: todas las marcas de los artículos. Formada por columnas como el nombre de la marca o una descripción de esta.

- **users**: representa a los usuarios de la aplicación. En esta se encuentra columnas relacionadas con los datos personales del usuario.

- **user_groups**: representa a grupos de usuarios.

- **roles**: es la tabla que identifica a un único usuario o a un grupo de usuarios. Estas tres últimas forman parte de una relación polimórfica.

- **historicals**: almacena una especie de histórico diario de las compras que realiza cada usuario. Contiene columnas como el importe total de compra del día o el importe ahorrado.

Y el otro concepto son las tablas intermedias, utilizadas para representar relaciones muchos a muchos entre dos tablas principales. Se podrían definir como un enlace entre las tablas principales permitiendo relaciones complejas entre ellas. Por lo general estas tablas están formadas por los identificadores de las tablas que se relacionan y son nombradas usando también los nombres de las tablas relacionadas.

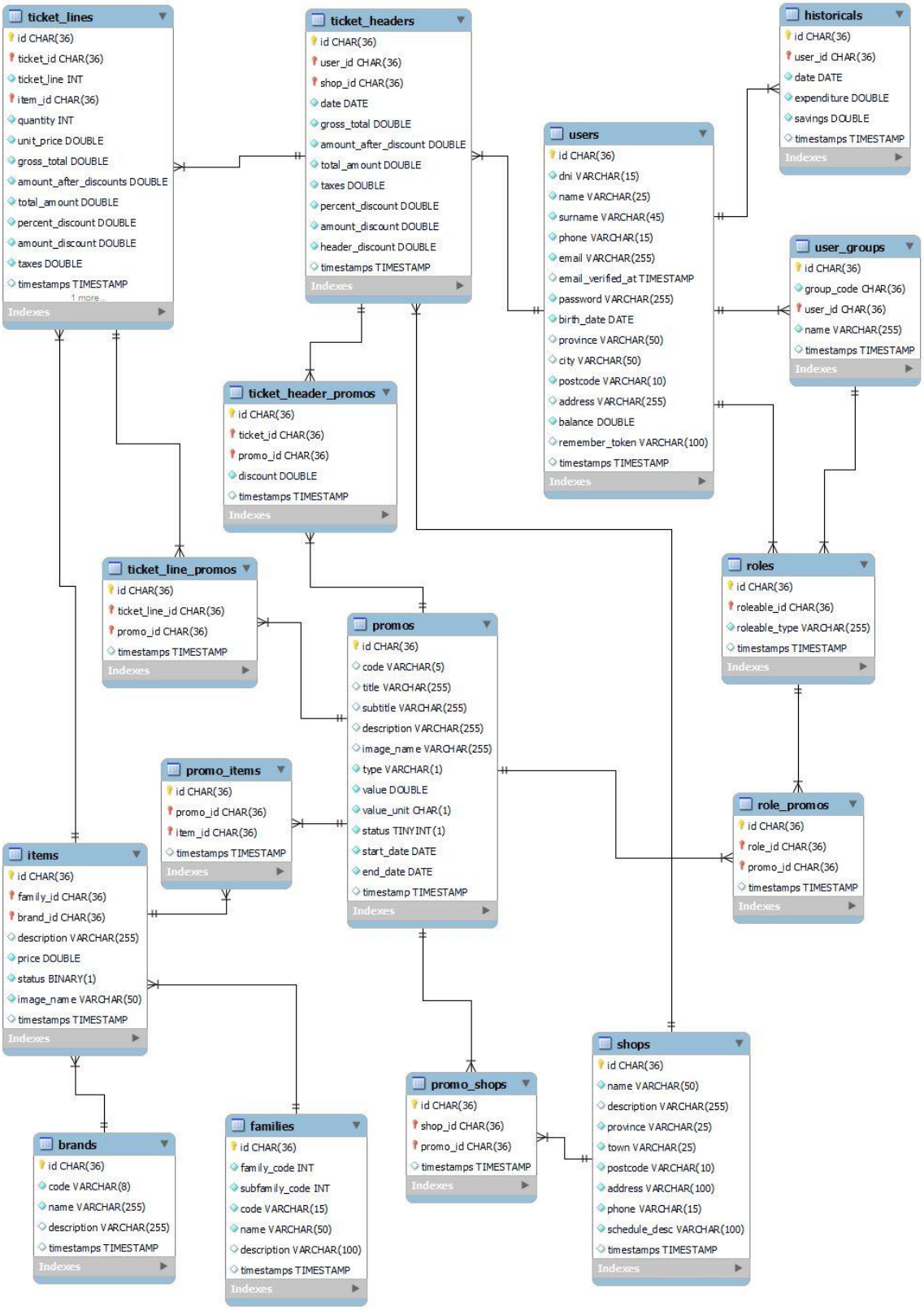

Figura 30: ***Modelo físico de la base de datos*** *[Fuente propia]*

Una vez el diseño de la base de datos está claro, se procede a su creación. Para ello, el primer paso es crear desde la herramienta de gestión de bases de datos HeidiSQL,

mencionado anteriormente en el apartado “Tecnologías utilizadas”. En este proceso se definen las credenciales para acceder a la base de datos creada y se indica la dirección IP donde se despliega el servicio, así como el puerto.

Con la base de datos creada ya se puede conectar la capa de datos con la base de datos, y se consigue haciendo uso del archivo “.env”, indicando simplemente el nombre de la conexión, el nombre de la base de datos, las credenciales, el host y el puerto.

Ahora ya se puede proceder a crear totas las tablas necesarias, y para ello se hará uso de las migraciones de Laravel. Estas no son más que archivos de código fuente PHP situados en el directorio “app/database/migrations”.

Se ha decidido hacer uso de ellos ya que ofrecen una serie de ventajas muy significantes en el desarrollo de un proyecto como el implementado.

Algunas de estas ventajas son:

- Control de versiones: al tratarse de archivos de código fuente, también son incluidos en el sistema de control de versiones, permitiendo llevar un registro de los cambios que se realizan en la estructura de la base de datos.

- Reversión de cambios: las migraciones permiten deshacer cambios en la estructura de la base de datos mediante el comando “rollback”, pudiendo volver a versiones anteriores.

- Sincronización con el código fuente: con el uso de las migraciones se consigue una estructura de la base de datos sincronizada con el código fuente de la aplicación, y esto garantiza que los cambios en el esquema de la base de datos se realicen de forma controlada.

Las migraciones pueden crear las tablas que componen la base de datos o incluso modificar tablas ya creadas. En el *Anexo 3* se muestra un ejemplo de migración para crear una tabla.

Ahora que ya existe la base de datos, solo queda llenarla con datos. Para ello se ha utilizado un proceso ETL (Extracción, Transformación y Carga) haciendo uso de los comandos de Laravel. Estos comandos son archivos de código fuente que permiten crear funciones que se ejecutan tras la ejecución de un comando.

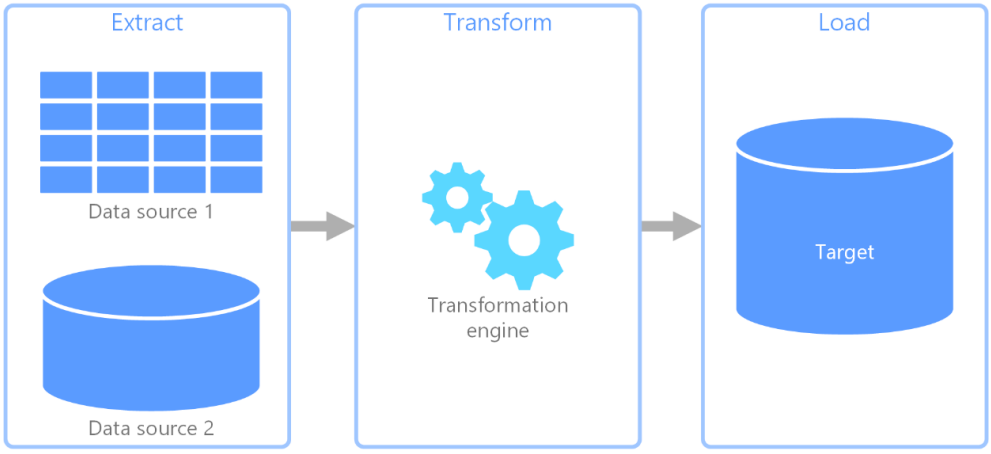

Figura 31: ***Representación del proceso ETL*** *[https://learn.microsoft.com/es-es/azure/architecture/data-guide/relational-data/etl]*

Los datos se han extraído de la base de datos central de la empresa, que para poder acceder a ellos se ha configurado una nueva conexión en el archivo ".env".

Teniendo acceso a esta nueva fuente de datos, solo falta quedarse con los datos necesarios. Por ejemplo, uno de los comandos creados se ha utilizado para obtener la información referente a los artículos. El primer paso consta en saber que datos son los que interesan obtener y así poder crear las consultas necesarias. Una vez tenemos los datos obtenidos de la base de datos origen, hay que hace una limpieza y transformación de ellos, preparándolos para seguidamente insertarlos en la base de datos de la aplicación. En este proceso también se debe comprobar si es necesario crear información adicional referente al nuevo artículo como, por ejemplo, si la marca del producto ya existe o no, y si no existe también se debe crear.

Con la base de datos creada, y con las tablas llenas de los datos de interés, ya se puede acceder desde la capa de lógica de negocio a esta capa de datos.

En el *anexo 4* se muestra como ejemplo el script utilizado para rellenar la tabla de artículos.

4. RESULTADO

En este apartado se va a realizar un tour por la aplicación resultante tras el desarrollo.

La primera página que se muestra es la *figura 32*, la cual muestra dos botones.

Figura 32: ***Primera página de la aplicación*** *[Fuente propia]*

Si se pulsa sobre el primer botón, con el texto "Crear cuenta", aparecerá la *figura 33*, una nueva página con un formulario donde el usuario se podrá registrar introduciendo los datos necesarios. Estos datos

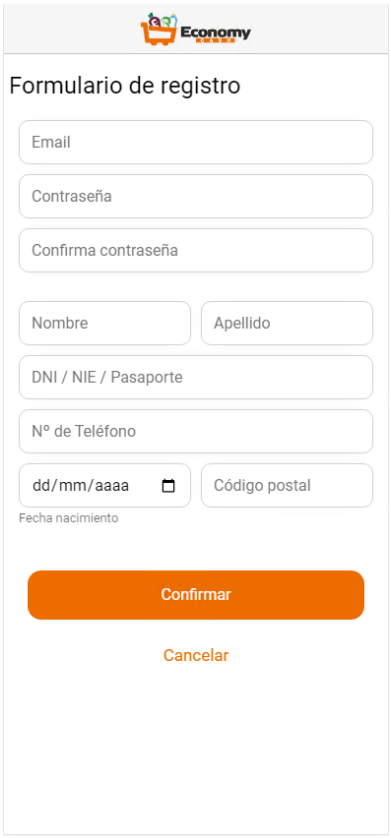

Figura 33: ***Formulario de registro*** *[Fuente propia]*

Todos los campos son obligatorios, por lo que si el usuario intenta confirmar la creación de la cuenta sin rellenar alguno de ellos se obtendrá como resultado la *figura 34*, la cual indica que el campo se ha de rellenar.

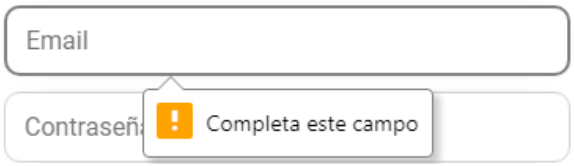

Figura 34: ***Campo requerido*** *[Fuente propia]*

También se controla el valor que se está asignando a cada campo. Por ejemplo para el email, se debe introducir un valor que sea del tipo email, incluyendo por ejemplo el símbolo arroba. Si la validación no se supera se muestra al usuario como en la *figura 35*.

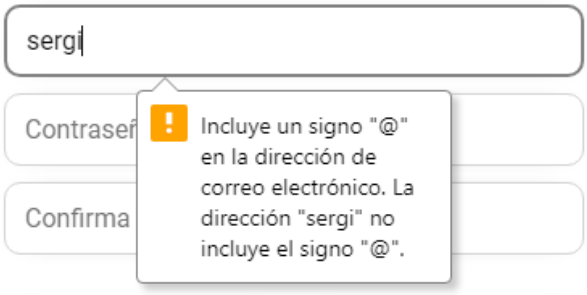

Figura 35: ***Validación de campos*** *[Fuente propia]*

También se comprueba si los campos email, DNI y teléfono ya existen en la base de datos. Si es así se informa al usuario que ya está dado de alta.

Al pulsar sobre “Cancelar” se vuelve a la pantalla anterior.

Con el segundo botón o tras confirmar el registro, se muestra el formulario de la *figura 36*, el cual permite al usuario iniciar sesión mediante el email y la contraseña.

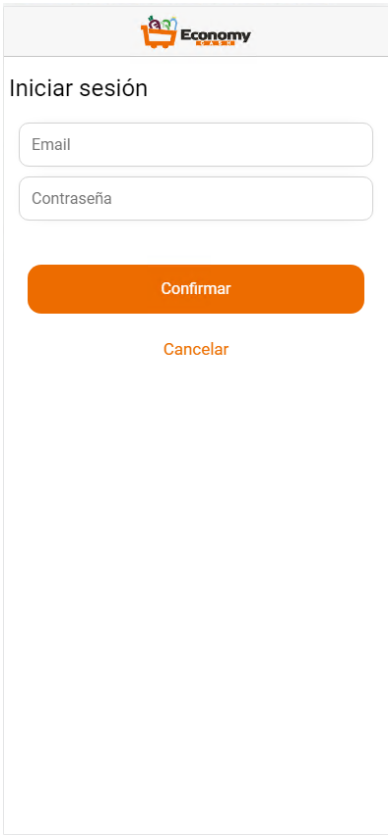

Figura 36: ***Formulario para iniciar sesión*** *[Fuente propia]*

Tras iniciar sesión la aplicación muestra la página de la *figura 37*, que siempre es la primera que se muestra al entrar a la aplicación.

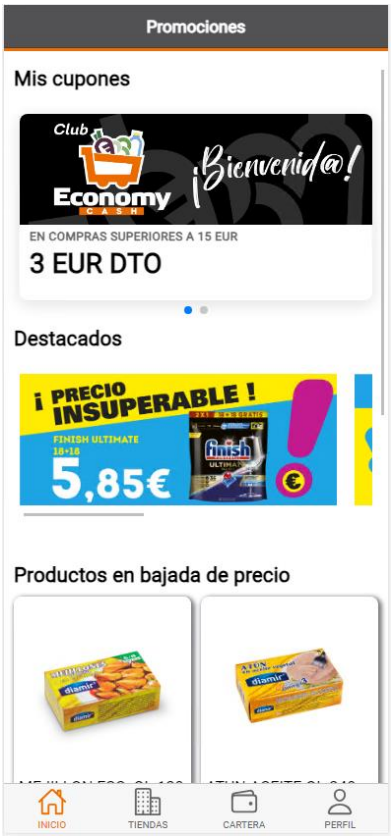

Figura 37: ***Página de inicio de la aplicación*** *[Fuente propia]*

En esta página se muestran tres apartados:

1. **Mis cupones**: es el espacio dedicado a mostrar los cupones que tiene el cliente de una form resumida, mostrando únicamente la imagen, el título y el subtítulo. Estos cupones se envuelven de un componente destinado a crear un carrusel de cupones.

2. **Destacados**: información que la empresa quiere destacar a modo de banners. Se muestran en forma de slider horizontal.

3. **Productos en bajada de precio**: son una colección de 6 productos que se encuentran en bajada de precio. Cada producto se muestra en forma de tarjeta con su imagen, el precio original y el precio tras el descuento. Se muestra más en detalle en la *figura 38*.

Figura 38: ***Apartado de productos*** *[Fuente propia]*

Sin moverse todavía de esta página se puede acceder al detalle de los cupones mostrados en la primera sección simplemente pulsando sobre el cupón en cuestión. Al hacerlo se muestra la toda la información como en la *figura 39*. Este es un modal done aparece información como la descripción del cupón, la fecha de validez y el código QR y código de texto que se pueden utilizar para canjear cada cupón. Para volver a la página anterior se haría uso del botón situado a la izquierda de la cabecera.

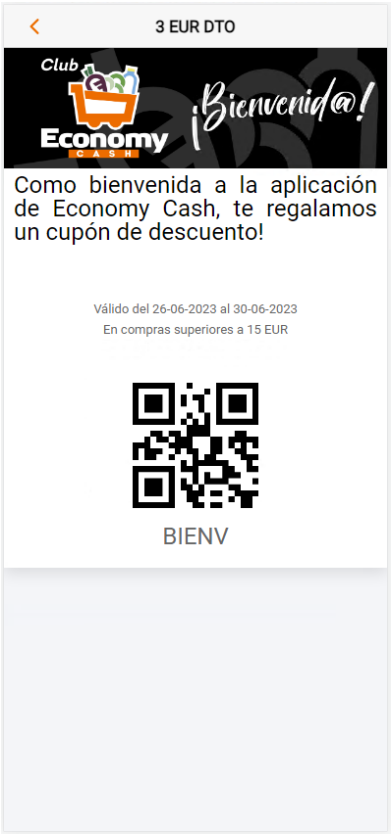

Figura 39: ***Detalle del cupón*** *[Fuente propia]*

Haciendo uso de los botones que se presentan en la *figura 40* situados en la parte inferior, se puede navegar entre las diferentes páginas de la aplicación.

Figura 40: ***Botones de navegación entre páginas*** *[Fuente propia]*

Al pulsar sobre TIENDAS se cambia a la página de tiendas, donde hay un sencillo listado con las diferentes tiendas que forman parte de la cadena de supermercados Economy Cash, tal y como se muestra en la *figura 41*. A demás del listado en la parte superior hay un filtro desplegable que permite seleccionar la provincia que interese, y así se mostrarán únicamente los establecimientos de esa provincia.

Figura 41: ***Listado de tiendas*** *[Fuente propia]*

Como se puede ver el listado de tiendas está formado por diferentes tarjetas donde se da información de cada una de las tiendas. Entre la información se encuentra la dirección del establecimiento, y al pulsar sobre esta se abrirá la ubicación en Google Maps.

Otra página es la nombrada como CARTERA, que es la correspondiente a la *figura 42*.

Figura 42: ***Página cartera*** *[Fuente propia]*

Lo primero que se ve en esta página es una especie de tarjeta con el logo de la empresa, que al pulsar sobre "VER TARJETA" esta se voltea y muestra el nombre del usuario y el código QR que lo identifica. Tal y como se muestra en la *figura 43*.

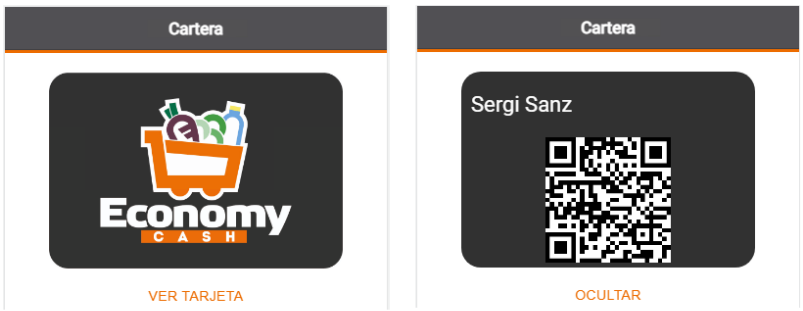

Figura 43: ***Las dos caras de la tarjeta*** *[Fuente propia]*

Seguido de la tarjeta se encuentra el saldo del que dispone el usuario. Este está rodeado de una pequeña descripción.

Por último, se encuentran dispuestas tres tarjetas que operan como botones, y al ser pulsadas, desencadenan acciones visibles en la página. Estos botones son:

- **Mis Cupones**: Al presionarse se muestra un modal que ocupa toda la pantalla donde se muestra un listado de todos los cupones del usuario. El listado es formado por tarjetas que contienen toda la información necesaria para saber que ofrece el cupón y para poder hacer uso de él, ya que se muestra el código del cupón. Este modal corresponde a la *figura 44*.

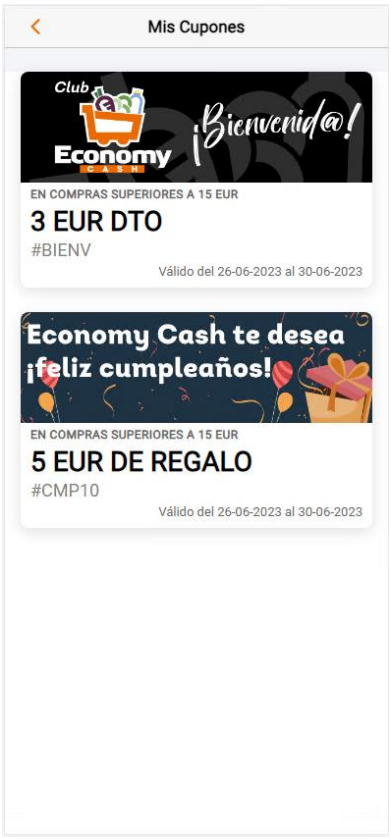

Figura 44: ***Listado de cupones*** *[Fuente propia]*

- **Mis Compras**: con su activación se abre el modal que corresponde con la *figura 45*, en el que se muestra una simple tabla con un registro diario del importe total que han supuesto las compras ese día y el importe del descuento que se ha aplicado. Los días sin compra no aparecerán en la tabla.

Figura 45: ***Histórico de compras*** *[Fuente propia]*

- **Ver Ahorro**: se vuelve a desplegar un modal, esta vez con información relacionada con el ahorro que está acumulando el cliente. Corresponde a la *figura 46* y se puede distinguir dos partes. La primera de ellas se forma por dos tarjetas, donde una muestra el importe total que se ha acumulado durante el año actual y otra que muestra el valor acumulado durante el mes en curso. La segunda parte corresponde a un gráfico de barras que representa el importe de descuentos que se han acumulado.

Figura 46: ***Ahorro acumulado*** *[Fuente propia]*

Por último se encuentra la página PERFIL, correspondiente a la *figura 47* En este apartado se muestra un listado de puntos de acceso que llevarán al usuario a distintas vistas.

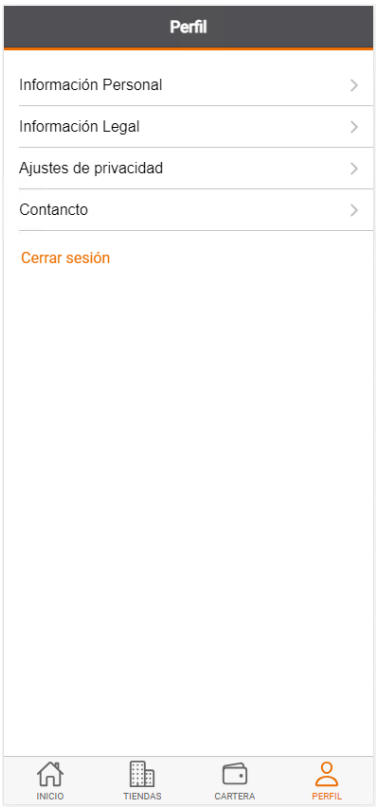

Figura 47: ***Página Perfil*** *[Fuente propia]*

De estas solo se ha desarrollado un modal para que el usuario pueda consultar su información personal. El resto de puntos se han puesto a modo de ejemplo, ya que estos temas los debe definir la empresa en un momento más avanzado.

En esta página también se puede cerrar sesión pulsando sobre "Cerrar sesión", y aparecerá el mensaje de la *figura 48*.

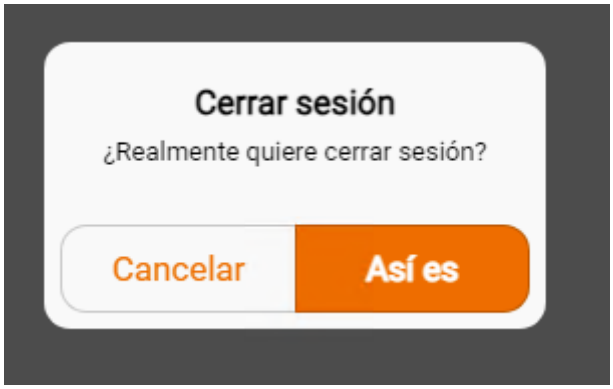

Figura 48: ***Mensaje para confirmar cerrar sesión*** *[Fuente propia]*

Todas las capturas mostradas se han realizado desde lo que sería un dispositivo con sistema operativo iOS. Para dispositivos Android la aplicación se comporta exactamente igual, pero algunos elementos se mostrarán ligeramente diferentes.

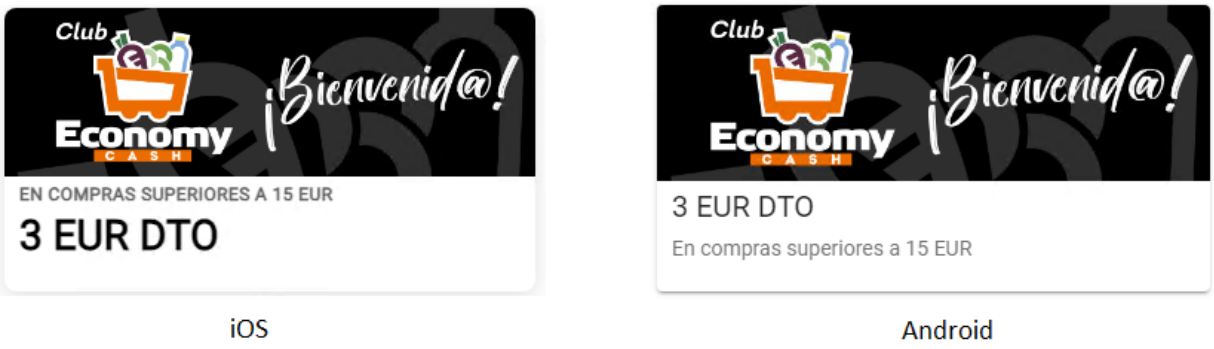

Figura 49: ***Comparación de estilos entre sistemas operáticos*** *[Fuente propia]*

5. CONCLUSIONES Y TRABAJOS FUTUROS

A lo largo de esta memoria, se ha abordado el desarrollo de una aplicación web de fidelización de clientes para los supermercados Economy Cash, con el objetivo de aumentar la cantidad de clientes leales a esta cadena de supermercados.

Este desarrollo ha supuesto un desafío significativo, ya que se han tenido que aprender nuevos lenguajes de programación y nuevas tecnologías para implementar la aplicación, pero también ha sido toda una oportunidad para poner en práctica los conocimientos que se han adquirido en el transcurso de grado.

Al tratarse de un proyecto real, la elección de las tecnologías utilizadas también ha sido un punto clave y crítico. Esto a supuesto un proceso de investigación que ha aportado mucho conocimiento, de los cuales se ha tenido que obtener conclusiones suficientemente sólidas como para optar por ciertas tecnologías o herramientas.

La complejidad notablemente elevada del proyecto ha exigido también aprender a planificar y gestionar eficientemente los recursos y el tiempo disponible, convirtiéndose en un aspecto crucial para garantizar un progreso contante y la finalización exitosa del proyecto. Entre otras cosas se ha aprendido a priorizar tareas y a definir hitos a corto y medio plazo.

La implantación de esta aplicación no ha hecho más que comenzar, ya que este proyecto es una primera entrega de la aplicación, quedándose internamente en la empresa y si es aprobada, avanzará su desarrollo e integración. Es por ello que algunos temas se quedan fuera del TFG, como la seguridad, el tratamiento de datos personales o la implementación junto a empresas externas que se ven involucradas. Estos temas, al presentar intervención interna de la empresa, se abarcarán más adelante.

Durante el desarrollo de la aplicación han surgido ideas de posibles funcionalidades que se han valorado como viables para implantar en un futuro cercano. Entre estas se pueden destacar:

- Posibilidad de añadir productos a una lista de favoritos. Estos productos se marcarán como favoritos de una manera muy sencilla, y el usuario será notificado si estos productos tienen una nueva promoción activada.

- Creación de listas de compra. Cada usuario de la aplicación podrá crear distintas listas de productos. Por ejemplo, podrían crear una lista de compra para la compra semanal, otra para la reunión familiar del fin de semana y otra para el cumpleaños de un amigo

- El uso de Machine Learning para ofrecer ofertas totalmente personalizadas. La idea es hace llegar al usuario ofertas de productos que realmente le interesen, a partir de algoritmos de venta cruzada o de predicción de gustos. Esta funcionalidad supondrá un reto interesante, ya que los conocimientos de este ámbito son limitados, aunque también supone una motivación extra.

- Dedicar un apartado a recetas. Estas irán rotando cada cierto tiempo, y ofrecerán al cliente un diverso surtido de ideas de recetas que podrá llevar a cabo haciendo uso de los productos que ofrecen los supermercados Economy Cash. A demás se implantaría la opción de añadir todos los ingredientes necesarios a una lista de compra

de forma automática. Con esto se consigue a un cliente activo, que consulta la aplicación para obtener nuevas ideas de que comer, facilitándole además la compra de los productos.

- La creación de hitos u objetivos para conseguir recompensas. Estos logros tendrán que ser alcanzados por los clientes comprando en los supermercados y acumulando una cantidad de ahorro o importe para así obtener las recompensas en cuestión. Esto supondría una especie de juego para el usuario, haciendo de la compra un proceso más divertido, ya que podrá ir alcanzando diferentes niveles de experiencia, que se transformará en un rango de cliente.

Todas estas funciones se estudiarán en profundidad tras la aprobación por parte de la empresa, participando en la implementación de estas el resto de las componentes del equipo de desarrollo, agilizando así el proceso.

A pesar de que se ha conseguido un diseño bastante robusto, sencillo e intuitivo, está planeada la incorporación del equipo de diseño del departamento de marketing, con el objetivo de crear un diseño mucho más atractivo y con una finalidad a la imagen de la marca insuperable. Este proceso se ha sacado del TFG ya que esto hubiera supuesto un aumento muy notable en el tiempo de desarrollo, pudiendo llegar a ser inviable.

Mientras el diseño se finaliza, algunos componentes del equipo de desarrollo se encargarán de integrar las funcionalidades necesarias en el ERP de la empresa, para así poder generar información, como crear promociones o gestionar usuarios, desde central.

BIBLIOGRAFÍA

[1] Wikipedia, la enciclopedia libre. (2022). *Tipado fuerte*. Recuperado de: https://es.wikipedia.org/w/index.php?title=Tipado_fuerte&oldid=147689333
[2] Rafael Marín. (2022). *¿Qué es una web app? tipos y diferencias con un sitio web Estándar. Recuperado de:* https://www.inesem.es/revistadigital/informatica-y-tics/que-es-una-web-app/
[3] *Mdn*. (2023). *CSS.* https://developer.mozilla.org/es/docs/Web/CSS
[4] Team, T. E. (2022) *What is a relational database? A deep dive, Medium.* Recuperado de: https://medium.com/educative/what-is-a-relational-database-a-deep-dive-3247d26869a6
[5] Martín Escofet, C. (s. f.). *El lenguaje SQL.* Recuperado de: https://pdfcursos.com/pdf/0018-lenguaje-sql.pdf
[6] Maximilian Schwarzmüller (s. f.). *React - The Complete Guide 2023 (incl. React Router & Redux)*. Recuperado de: https://www.udemy.com/course/react-the-complete-guide-incl-redux/
[7] Ionicframework. (s. f.).*UI Components.* Recuperado de: https://ionicframework.com/docs/components
[8] W3Schools. (s. f.). *CSS Tutorial*. Recuperado de: https://www.w3schools.com/css/
[9] Lahtela, M., & Kaplan, P. (Provenance). (s. f.). *¿Qué es una API?* Recuperado de: https://aws.amazon.com/es/what-is/api/
[10] León Osorio Rivera, F. (2008). *Base de datos relacionales. Colombia: Instituto Tecnológico Metropolitano, ITM.*

ANEXOS

Anexo 1 – Ejemplo de creación de una tienda

Cuando se trata de dar de alta nueva información mediante una solicitud HTTP, el método utilizado es POST, que envía los datos en el cuerpo de la solicitud HHTP en lugar de enviarlos en la URL. Esto hace a estos datos no visibles en la URL, lo que proporciona cierta privacidad y seguridad.

En este ejemplo se presenta la creación de una nueva tienda. Primero se expone el código fuente que se utiliza para crear la información, implementado en el controlador del modelo de tiendas en el directorio "app/Http/Controllers/ShopController.php".

```
public function store(Request $request)
    {
        $rules = [
            'name' => 'required|string|min:1|max:50',
            'province' => 'required|string|min:1|max:25',
            'town' => 'required|string|min:1|max:50',
            'postcode' => 'required|string|min:1|max:10',
            'address' => 'required|string|min:1|max:100',
            'phone' => 'required|string|min:9|max:15',
            'schedule_desc' => 'required|string|min:1|max:100'
        ];
        $validator = Validator::make($request->input(), $rules);
        if ($validator->fails()) {
            return response()->json([
                'status' => false,
                'errors' => $validator->errors()->all()
            ],400);
        }


        $shop = new Shop($request->input());
        $shop->save();
        return response()->json([
            'status' => true,
            'message' => 'Shop created successfully'
        ],200);
    }
```

El primer paso que se realiza en este método es definir unas reglar referentes a los datos recibidos de la solicitud HTTP, como si es un valor requerido, el tipo de dato o su longitud.

Estas reglas se aplicarán haciendo uso de la herramienta "Validator" proporcionada por Laravel. Si alguna de estas reglas se incumple, el método termina retornando el mensaje de error correspondiente. Si la validación es correcta, seguidamente se procede a crear el nuevo registro. Para ello se crea una nueva instancia del modelo "Shop", al cual se le pasan los datos que llegan desde la solicitud HTTP.

Por último se devuelve un mensaje confirmando la creación de la nueva tienda.

Para enlazar esta clase con la petición HTTP es necesario definir una ruta en el archivo "api.php" dentro de la carpeta "routes". En este caso se realiza de la siguiente manera:

```
Route::post('/shops', [ShopController::class, 'store']);
```

Para comprobar que funciona correctamente se puede utilizar Postman, que permite realizar la petición HTTP de forma muy sencilla.

Como la ruta se ha definido dentro del middleware "auth:Sanctum", es necesario iniciar sesión para poder hacer uso de la ruta. Se realizará utilizando la solicitud HTTP destinada a ello, también desde Postman, tal y como se muestra en la figura 50. En esta se pueden distinguir tres apartados. El primero que corresponde a la URL que se utiliza para la solicitud. Seguidamente se encuentra un apartado "Params" que es donde se definen los parámetros que queremos pasar en la petición con sus correspondientes valores. Por último se sitúa la respuesta obtenida. En este caso como respuesta se confirma el inicio de sesión con los datos del usuario y con el token que recibe la sesión ("accessToken"). Este token es la clave que permitirá realizar solicitudes HTTP que estén protegidas con el middleware "auth:Sanctum".

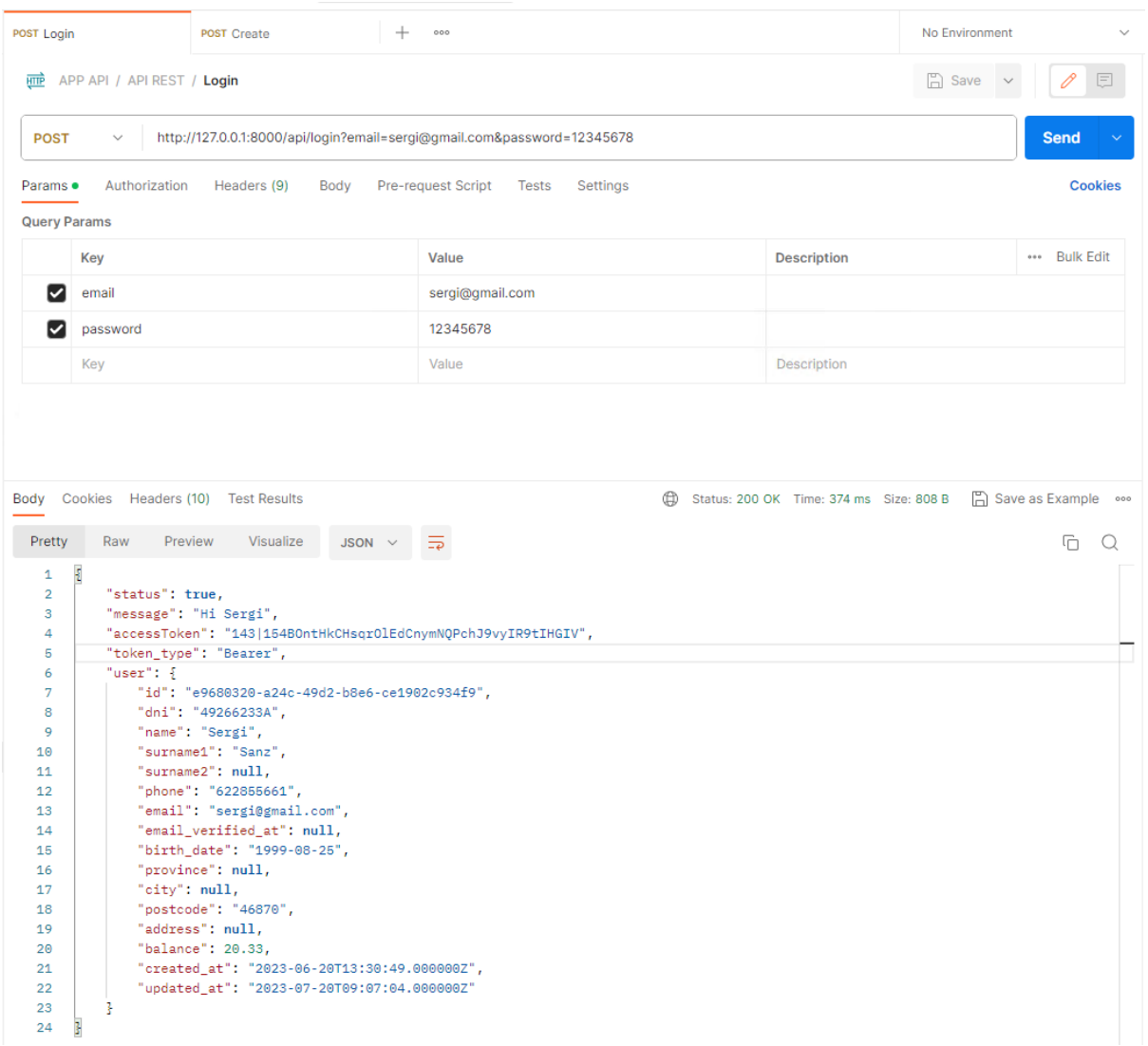

Figura 50: ***Solicitud "login" desde Postman*** *[Fuente propia]*

Con todo esto ya se puede realizar la solicitud para crear una nueva tienda. Para ello, en la petición de Postman encargada se tiene que abrir el apartado “Authorization”, seleccionar el tipo de token “Bearer Token” e introducir el token del usuario autenticado tal y como se muestra en la *figura 51*.

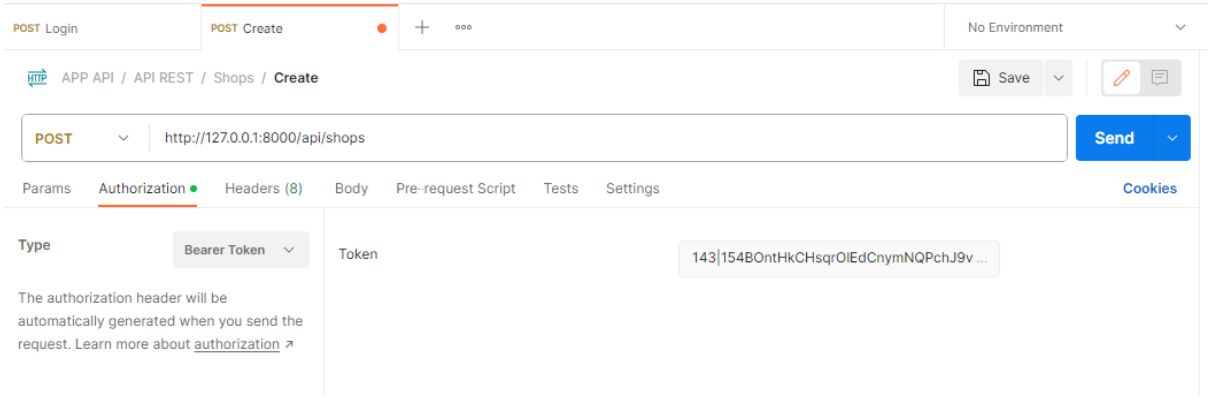

Figura 51: ***Introducir token*** *[Fuente propia]*

Para finalizar, en la *figura 52* se muestra la petición HTTP final, con todos los parámetros y datos de ejemplo.

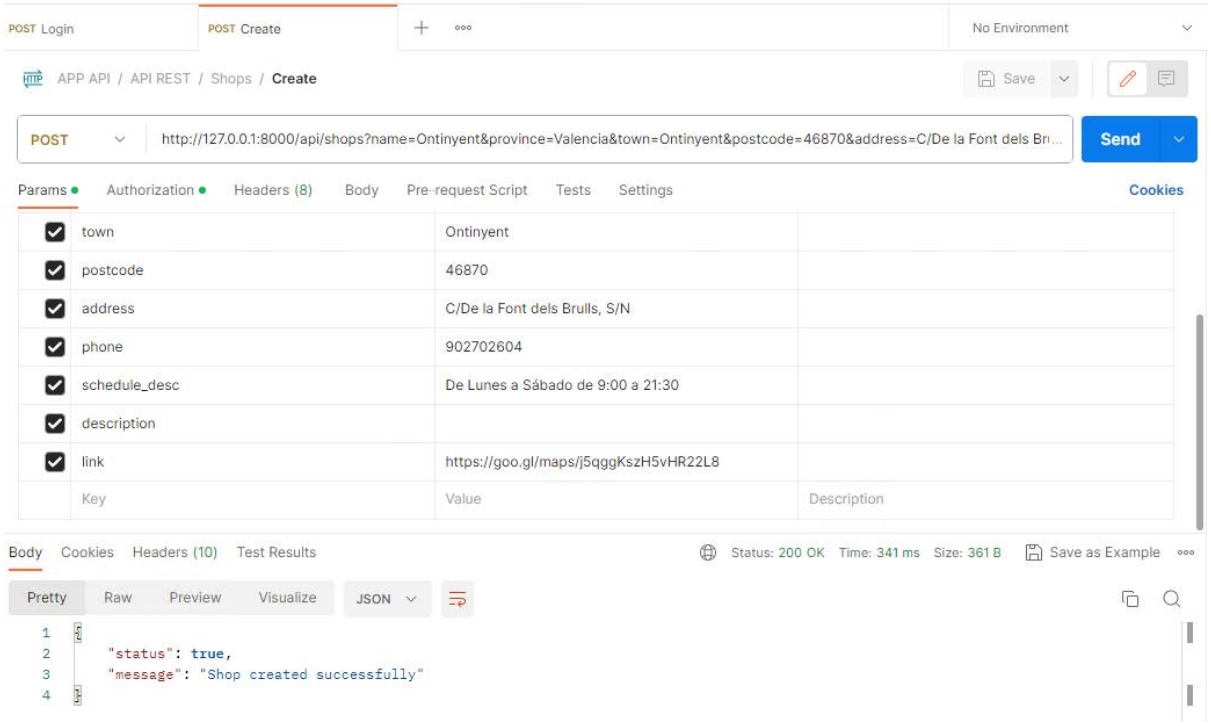

Figura 52: ***Creación de una tienda desde Postman*** *[Fuente propia]*

Anexo 2 – Ejemplo de "get" para recuperar los cupones del usuario

El código fuente que se utiliza para obtener los cupones que tiene activos un usuario se encuentra en el controlador "RolePromoController.php". Y es el presentado a continuación:

```
// Get all the promos (coupons) associated with the authenticated user
    public function authCoupons()
    {
        $usuario = Auth::user();
        $role = Role::where('roleable_id', $usuario->id)->pluck('id');
        $rolePromos = RolePromo::where('role_id', $role)
->pluck('promo_id');
        $promos = Promo::whereIn('id', $rolePromos)->where('status',1)
->where('type','S')->get();
        foreach ($promos as $promo) {
            //  Formatting the dates start_date and end_date like d-m-Y
            $dateStr = $promo['start_date'];
            $date = date_create($dateStr);
            $result = $date->format('d-m-Y');
            $promo['start_date'] = $result;

            $dateStr = $promo['end_date'];
            $date = date_create($dateStr);
            $result = $date->format('d-m-Y');
            $promo['end_date'] = $result;
        }
        return $promos;
    }
```

Este método es bastante sencillo, en él se obtiene el usuario autentificado y a partir de este se obtienen los cupones que tiene asociados y están activos.

La petición HTTP se puede encontrar por ejemplo en el componente "ModalMyCoupons.tsx", tal y como se muestra a continuación:

```
  useEffect(() => {
    axios
      .get("http://127.0.0.1:8000/api/myCoupons", {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      })
      .then((response) => {
        setIsLoaded(true);
        setItems(response.data);
      })
      .catch((error) => {
        setError(error);
      });
  }, []);
```

Y se accede a partir de la ruta:

```
Route::get('/myCoupons', [RolePromoController::class, 'authCoupons']);
```

Anexo 3 – Ejemplo de migración para crear una tabla

A continuación, se muestra un ejemplo de como crear una tabla haciendo uso de las migraciones. Como ejemplo se va a utilizar la tabla de promociones, llamada "promos". El primer paso es crear el archivo que contendrá el código fuente y para ello se hará uso del siguiente comando:

> php artisan make:migration [nombre_de_migración]

En este comando debemos sustituir *nombre_de_migración* por el nombre de la migración. Es importante seguir una nomenclatura para nombrar a las migraciones, para así mantener un orden que facilitará el desarrollo de la aplicación. Como se trata de la creación de una tabla se seguirá la convención *create_[table_name]_table*, obteniendo como resultado el siguiente comando:

> php artisan make:migration create_promos_table

Tras ejecutar el comando se generará un archivo con un nombre parecido a "2023_06_01_080627_create_promos_table.php". El valor que se ha concatenado al inicio corresponde a la fecha y hora de creación, con el objetivo de mantener el orden por creación de los archivos. Es muy importante seguir un orden adecuado en el momento de ejecutar las migraciones, consiguiendo evitar conflictos y asegurar la consistencia de la base de datos.

Con el archivo creado ya se puede escribir el código fuente, el cual se presenta a continuación.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('promos', function (Blueprint $table) {
            $table->uuid('promo_id')->primary()
                ->comment('Identifier of the promo.');
            $table->string('code', 5)->unique()->nullable()
                ->comment('5-character code used to redeem the promotion in
shop.If null, the promotion is automatically applied. ');
            $table->string('description', 100)->nullable()
                ->comment('Description of the promotion');
            $table->string('type', 1)
                ->comment('S: for single-use promotions | M: for multiple-
use promotions');
            $table->float('value', 6, 2)->default('0')
                ->comment('Amount of the discount');
            $table->char('value_unit', 1)
                ->comment('Indicates the unit of the amount. P: Percent | M:
Monetary');
```

```
            $table->boolean('status')
                ->comment('1 for activated promotions and 0 for deactivated
promotions')
                ->default('0');
            $table->date('start_date')
                ->comment('Date when the promo is activated');
            $table->date('end_date')
                ->comment('Date when the promo is deactivated');
            $table->timestamps();
        });
    }


    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('promos');
    }
};
```

En este bloque de código se definen dos funciones. Una que se ejecuta al hacer la migración y otra para revertir la migración. En la función "up" se define el nombre de la tabla, y cada una de las columnas que la forman, indicando algunas propiedades como el nombre de la columna, el tipo de dato y comentarios referentes a la información almacenada en cada columna.

Las migraciones pendientes de migrar se ejecutan haciendo uso del comando:

>php artisan migrate

Anexo 4 – Script para rellenar tabla de artículos

En el contexto de Laravel estos scripts son nombrados como comandos y se encuentran en el directorio ”app/Console/Commands”.

En este anexo se pone como ejemplo el script utilizado para llenar la tabla de artículos a partir de la base de datos de central. La creación de un artículo también puede incluir o no la creación de la familia y de la marca.

Para ello se ha utilizado el código fuente que se presenta a continuación, en el cual se realiza primero la consulta a las tablas de la base de datos de centrar que se necesita para obtener los datos que se desean. A partir de estos datos obtenidos, se crea, si no existen, la familia y la marca correspondiente, y finalmente se crea el artículo.

```php
<?php
namespace App\Console\Commands;
use App\Models\Brand;
use App\Models\Family;
use App\Models\Item;
use Illuminate\Console\Command;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Storage;

class InsertItems extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'app:insert-items';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Command description';

    /**
     * Execute the console command.
     */
    public function handle()
    {
        // Query to obtain the items
        $itemsAll = DB::connection('oracle')->table('sop_price_lines as a')
            ->select('b.part_code as part_code', 'a.sales_price as price',
DB::raw("REPLACE(c.part_desc_1, '§', 'o') as name"), 'c.sf_brand as
brand_code', 'd.code_description as brand', 'c.product_class as family')
            ->join('sop_prices as b', 'a.price_record_number', '=',
'b.price_record_number')
            ->join('product_master as c', 'b.part_code', '=',
'c.part_code')
            ->join('sf_brands as d', 'c.sf_brand', '=', 'd.sf_brand')
            ->where('b.price_code', 'K1')
```

```
            ->whereIn('c.product_class', function ($query) {
                $query->select('product_class')
                    ->from('product_classes')
                    ->where('company_code', '5')
                    ->where('nd_master_class', 'G06')
                    ->where(DB::raw('LENGTH(product_class)'), 6)
                    ->whereRaw("REGEXP_LIKE(product_class, '^[0-9]+$')");
            })
            ->where('c.activation_status', 'A')
            ->get();

        foreach ($itemsAll as $item) {
            $familyId = Family::where('subfamily_code', $item->family)
->value('id');
            $item->family = $familyId;


            // Check if the brand already exists
            $existsBrand = Brand::where('code', $item->brand_code)
->exists();
            // If not exists, create brand
            if (!$existsBrand) {
                $newBrandData['code'] = $item->brand_code;
                $newBrandData['name'] = $item->brand;
                $newBrand = new Brand($newBrandData);
                $newBrand->save();
            }
            // Get the brand id
            $brandId = Brand::where('code', $item->brand_code)
->value('id');
            $item->brand = $brandId;

            // Check if the item already exists
            $existsItem = Item::where('code', $item->part_code)->exists();
            // If not exists, create item
            if (!$existsItem) {
                $newItemData['code'] = $item->part_code;
                $newItemData['family_id'] = $item->family;
                $newItemData['brand_id'] = $item->brand;
                $newItemData['name'] = $item->name;
                $newItemData['price'] = $item->price;
                $newItemData['image_name'] = $item->part_code . ".jpg";

                $newItem = new Item($newItemData);
                $newItem->save();
            }
        }
    }
}
```

Para ejecutar el script se utiliza el comando *php artisan* seguido del valor de la variable "signature" definida también el código feunte. En este caso sería:

> php artisan app:insert-items