Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Desarrollo de una aplicación full-stack para la geolocalización de productos alimenticios

Development of a full-stack application for the geolocation of food products

Tanausú Falcón Casanova

La Laguna, 14 de julio de 2023

D. **Eduardo Manuel Segredo González**, con N.I.F. 78.564.242-Z profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

Dña. **Gara Miranda Valladares**, con N.I.F. 78.563.584-T profesora Titular de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutora

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Desarrollo de una aplicación full-stack para la geolocalización de productos alimenticios*"

ha sido realizada bajo su dirección por D. **Tanausú Falcón Casanova**, con N.I.F. 79.153.845-N.

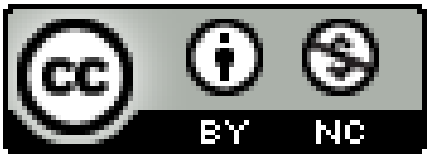
Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de julio de 2023

Agradecimientos

Agradezco a mi familia y amigos, especialmente a mi madre y hermana, por todo el apoyo durante estos años. A pesar de todos los obstáculos encontrados, habéis seguido ahí para darme el consuelo y la fuerza necesarios para sobreponerme a cualquier desafío. Mi vida ha cambiado tanto en tan poco tiempo, y llegar hasta este punto es un gran motivo de orgullo para mí.

Doy las gracias también a los tutores Eduardo Manuel Segredo y Gara Miranda por la oportunidad de realizar este proyecto, y por su comprensión y guía a lo largo del mismo.

Licencia



Resumen

Esta memoria detalla el proceso de desarrollo de un API REST para el almacenamiento y consulta de información sobre productos alimenticios, con sus valores nutricionales, precios en tiendas y ubicaciones, así como la creación de una aplicación móvil que haga uso de la misma.

Las funcionalidades implementadas por esta aplicación permiten añadir productos con sus valores nutricionales, al igual que el precio y geolocalización en supermercados y tiendas donde fueron encontrados. Además, el API recoge un histórico de precios, con el que se puede realizar la comparativa en distintas tiendas a lo largo del tiempo. El API REST se ha desarrollado con Node.js y Express. Para la aplicación móvil, se ha utilizado el framework de React Native, junto con el ecosistema de herramientas de Expo para facilitar el desarrollo tanto en la plataformas de iOS como Android. Por último, la base de datos elegida es MongoDB, haciendo uso del modelado de datos de Mongoose. Todo el código ha sido implementado con el lenguaje de programación TypeScript.

Con todo ello, se ha podido tener un mayor entendimiento de los APIs REST y el desarrollo de aplicaciones móviles. Las mayores dificultades se han encontrado en los inicios del proyecto, y se ha necesitado un tiempo de aprendizaje sobre el entorno de desarrollo móvil. Algunas mejoras a implementar es el añadido de imágenes para productos, la documentación del API, las pruebas en la aplicación y el rediseño de pantallas para mejorar la experiencia de usuario.

Palabras clave: API, REST, aplicación, móvil, productos, precios, información nutricional, comparador de precios, tiendas, geolocalización

Abstract

This memory details the development process of a REST API for storing and querying information about food products, with their nutritional values, prices in stores and locations, as well as the creation of a mobile application that uses it.

The features implemented by this application allow adding products with their nutritional values, besides the price and location in supermarkets and stores where they were found. In addition, the API collects a price history, with the posibility to compare prices in different stores over time. The API REST has been developed with Node.js and Express. For the mobile app, the React Native framework has been used, toguether with the Expo ecosystem of tools to facilitate development on both iOS and Android platforms. Finally, the database chosen is MongoDB, making use of Mongoose data modeling. All the code has been implemented with TypeScript programming language.

With all this, a better understanding of REST APIs and the development of mobile applications has been achieved. The greatest difficulties have been encountered at the beginning of the project, and it has taken time learning about the mobile development environment. Some improvements to be made are the addition of images for products, API documentation, in-app testing, and screen redesign to improve the user experience.

Keywords: API, REST, aplication, mobile, products, prices, nutritional information, price comparison, shops, geolocation

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Con el aumento de coste de calidad de vida, y consecuentemente, las variaciones del IPC (Índice de precios al consumidor [29]) durante los últimos años, existe una necesidad en la población de encontrar los productos a los mejores precios. Gracias al surgimiento de nuevas tecnologías, y al crecimiento de la accesibilidad a las mismas, han emergido distintas soluciones que pretenden resolver esta problemática, como comparadores de precios, buscadores de ofertas, entre otros. Además, a medida que la información se ha vuelto más accesible, la preocupación por la alimentación y la salud se ha expandido, convirtiéndose en una prioridad para muchas personas. Aunque la existencia de aplicaciones que recopilan la información nutricional de productos es conocida, muy pocas de estas disponen de información sobre sus precios, o se mantienen actualizados, siendo muchos de los alimentos más saludables los más caros.

En este trabajo, se recoge el desarrollo de una aplicación full-stack para la recopilación de la información nutricional, el registro de precios, y la geolocalización de productos alimenticios en tiendas y supermercados. Dicha aplicación permitirá escanear productos, visualizar los datos que otros usuarios hayan ingresado relativos a estos, y conseguir un histórico que muestre la evolución de precios del mismo. Este modelo colaborativo no solo permitirá poder comparar los costes en diferentes establecimientos, sino además obtener información actualizada sobre los mismos.

De esta manera, el proyecto constará de la creación de un API REST [9], que permitirá las operaciones de acceso y publicación de datos, estando estos securizados, así como la integración con aplicaciones front-end. En este sentido, se desarrollará una aplicación para plataformas móviles (centrándose así en una metodología *mobile first* [12]), que facilitará la interacción entre los usuarios y el API. Principalmente, se utilizarán tecnologías basadas en Javascript y TypeScript, con el uso de React Native [20] para el desarrollo móvil en sus versiones de iOS y Android.

El objetivo final es que los usuarios puedan encontrar información actualizada sobre los productos que deseen, y que esta información pueda ser utilizada en otros desarrollos.

1.1. Objetivos del proyecto

Se dividen a continuación los objetivos a conseguir en el proyecto:

- La disponibilidad de un API REST que permita realizar las operaciones de consulta y manejo de datos sobre productos, precios y tiendas. Este API podrá ser utilizado por cualquier sistema o usuario para su integración en otros desarrollos.
- Securizar los datos del API mediante un sistema de autenticación y roles, que limite el acceso a recursos privilegiados del mismo.
- El desarrollo de un frontend que haga uso del API y cumpla con los requisitos expuestos en la Sección 2.3, favoreciendo la creación de una aplicación móvil.

1.2. Estructura de la memoria

La memoria del proyecto se estructura de la siguiente manera:

- El Capítulo 2 relata algunos sistemas de evaluación de la calidad nutricional en los etiquetados, así como un estudio del arte sobre las aplicaciones relacionadas con la búsqueda de información nutricional y precios de productos alimenticios, realizando un análisis sobre los requisitos a implementar en la aplicación.
- En el Capítulo 3 se expone una breve introducción a los tipos de base de datos, la elección del mismo, y las entidades del modelo de datos con las validaciones llevadas a cabo.
- El Capítulo 4 muestra el desarrollo del API, con la estructura del proyecto, sus tecnologías, la conexión a la base de datos y el inicio del servidor web, así como la documentación de las rutas disponibles. También se describe la autorización de usuarios en la plataforma.
- Posteriormente, el Capítulo 5 relata la creación de la aplicación móvil, describiendo su arquitectura, las tecnologías empleadas, y los componentes y pantallas que la constituyen mostrando, además, diversos casos de uso.
- Más adelante, en el Capítulo 6 se describen las tecnologías empleadas para las pruebas del código y el despliegue del API en un servicio en la nube.
- El Capítulo 7 detalla las conclusiones tras el desarrollo del proyecto, así como las líneas futuras y mejoras sobre las que avanzar.
- Por último, en el Capítulo 9 se expone el presupuesto del proyecto.

Capítulo 2

Estado del arte

Habitualmente, las personas se hallan en la necesidad de encontrar los productos que necesitan al mejor precio. Debido al aumento y la variación de productos en los últimos años, han surgido muchas herramientas y aplicaciones que asisten en la toma de decisiones de compra de alimentos, ya sea por la comparación de sus valores nutricionales, dietas a seguir, alergias o intolerancias, o más comúnmente, por su precio.

Lamentablemente, pocas de estas herramientas poseen una base de datos centralizada con precios e información nutricional actualizada, que sirva como guía a los consumidores sobre qué productos comprar y dónde encontrarlos.

2.1. Evaluación de la calidad nutricional

Existen multitud de sistemas para la evaluación de la calidad nutricional en los productos alimenticios. Algunos vienen indicados en las etiquetas de productos procesados, que ayudan a la toma de decisiones de los consumidores a favorecer los más saludables y que los fabricantes pueden aportar de manera voluntaria. Este etiquetado nutricional se conoce como FOPL (del inglés *Front-of-pack labelling*). Algunos de los más conocidos [21] podrían ser el sistema NOVA o el Nutri-Score.

Nutri-Score [1] es uno de los sistemas de evaluación y modelos FOPL más extendidos en Europa. Consiste en un semáforo de colores, con las letras de la A hasta la E. Esta escala refleja de manera gradual de mayor (letra A en verde) a menor (letra E en naranja oscuro) la calidad nutricional del producto.

En su cálculo, se tiene en cuenta variables consideradas como favorables (frutas y hortalizas, proteínas, fibra, aceite de oliva, nuez), y componentes desfavorables (grasas saturadas, azúcares, sal y energía), por cada 100 gramos o 100 mililitros del producto. La información de los componentes desfavorables y favorables se encuentra en la información nutricional del envase del producto, siendo el Nutri-Score una traducción de estos valores nutricionales para mayor facilidad y compresión.

De esta manera, el Nutri-Score resulta especialmente útil para establecer una medida de la calidad nutricional de los productos de manera sencilla y fácil de comprender.

2.2. Aplicaciones relacionadas

Previo al desarrollo del proyecto, se ha llevado a cabo un estudio sobre algunas de estas aplicaciones, relacionadas con los objetivos a cumplir por el sistema de información a desarrollar, además de un análisis sobre las diferentes características con las que debería de cumplir.

2.2.1. Aplicaciones centradas en la información nutricional

Actualmente, existe una multitud de aplicaciones enfocadas a la alimentación y consumo de productos saludables. Muchas de estas permiten a sus usuarios la búsqueda y escaneo de productos, llevando a cabo una evaluación de su información nutricional.

Se exponen así una serie de aplicaciones relacionadas con la recopilación de información nutricional de productos alimenticios:

- **Yuka** [30]: Aplicación creada como proyecto independiente, tiene como objetivo evaluar la calidad de los productos mediante su información nutricional e ingredientes. Permite escanear productos alimenticios y cosméticos para descifrar su composición y evaluar sus efectos en la salud. Su sistema de evaluación se basa según los criterios de Nutri-Score, la presencia de aditivos, y si los productos son orgánicos. De esta manera, puntúa los productos con un baremo entre $1 - 100$, distinguiéndolos como excelente, bueno, malo, y mediocre. En caso de que algún producto no se encuentre en su sistema de datos, permite que los usuarios ingresen la información del mismo.

- **MyRealFood** [16]: Centrada en el movimiento *Realfooding*, permite escanear productos alimenticios, mostrar su información nutricional e ingredientes que contiene, otorgarles una puntuación, y determinar su porcentaje de procesado, ultraprocesado y real. Además, permite a sus usuarios compartir sus propias recetas saludables. Sigue los criterios del sistema de clasificación NOVA para su valoración de productos, y al igual que Yuka, permite que sus usuarios ingresen información de algún producto que no se encuentre en su base de datos.

- **Calorie Mama** [2]: Aplicación orientada al escaneo de productos, con la particularidad de su implementación con inteligencia artificial junto a *deep learning* para el reconocimiento de platos, calculando su valor nutricional a partir de una simple foto. Utiliza un API propia llamada *Food AI API* que mejora a medida que los usuarios añaden más fotos a su base de datos. Este API está disponible a través de un servicio de pago que ofrece.

- **El CoCo** [4]: Aplicación móvil que informa de la calidad nutricional de los productos escaneados, orientada a fomentar el consumo consciente de alimentos. Al igual que otras aplicaciones, escanea los códigos de barras de productos para consultar su valor nutricional.

- **Otras aplicaciones**: como **MyHealth Watcher** [15], que permite personalizar las dietas mediante un perfil personal y actividades físicas, **Vegan Pocket** [23], que permite identificar alimentos veganos, **Foodvisor** [7], que realiza un recuento de calorías mediante una foto, entre otras aplicaciones similares enfocadas a dietas y pérdida de peso.

2.2.2. Aplicaciones centradas en la comparativa de precios

Existen una variedad de aplicaciones para la consulta de precios de productos disponibles en supermercados y tiendas, algunas permiten las comparativas de precios entre establecimientos para informar de las compras más baratas y dónde encontrarlas. A continuación, se exponen alguna de estas aplicaciones:

- **SoySuper** [22]: Permite a los usuarios crear listas de compra de productos, y ofrece el precio total de las mismas en diferentes supermercados de líneas registradas como AlCampo, El Corte Inglés o Mercadona. La búsqueda de productos puede realizarse mediante el escaneo del código de barras o el ingreso del nombre de los mismos. Los establecimientos disponibles dependen del código postal de los usuarios, para el envío de la compra a domicilio.
- **OCU Market** [18]: Aplicación enfocada a la búsqueda de productos de calidad al mejor precio. Creada por la Organización de Consumidores y Usuarios, facilita los precios de los productos de supermercados registrados cercanos al usuario mediante su localización. Permite escanear y buscar productos, ver sus ingredientes e información nutricional, y proporciona una valoración de escala saludable, mediante criterios de Nutri-Score e índice NOVA, además de los aditivos añadidos.
- **Tiendeo** [8]: Aplicación que ofrece ofertas asociadas en diferentes supermercados de los productos, incluyendo además listas de compra.
- **Aplicaciones propias de los supermercados**: Muchas líneas de supermercados disponen de sus propias aplicaciones gratuitas, donde lanzan ofertas de productos que los usuarios pueden validar. Ejemplos de estas serían Lidl, Dia, Mercadona, Carrefour, AlCampo, Eroski, entre otras.

2.3. Definición de requisitos

En las secciones anteriores, se han descrito algunas aplicaciones que poseen funcionalidades deseables para el sistema a conseguir. El desarrollo de cualquier producto software requiere del estudio y valoración de los requisitos imprescindibles que se pretenden lograr.

De esta manera, cumpliendo con el objetivo final del proyecto, los requisitos que el sistema debe alcanzar se pueden resumir en los siguientes puntos:

- El escaneo y almacenamiento de productos alimenticios.
- La obtención y consulta de información alimenticia sobre productos escaneados.
- La obtención y consulta de precios de productos escaneados.
- La comparativa de precios entre productos de diferentes establecimientos.
- La obtención de un histórico de precios de los productos.
- La geolocalización de productos en tiendas y supermercados.

Realizando una comparativa entre los objetivos y las funcionalidades de algunas de las aplicaciones expuestas anteriormente, se puede reflejar las características que cada una posee, y obtener una referencia sobre las posibles aproximaciones a tomar. En la Tabla 2.1 se muestran las funcionalidades que cada una ofrece.

Tabla 2.1: Características de las aplicaciones

	Yuka	MyRealFood	SoySuper	OCU Market
Escaneo de productos alimenticios	✓	✓	✓	✓
Consulta de información nutricional	✓	✓	*	✓
Consulta de precios	×	✓	✓	✓
Comparativa de precios	×	✓	✓	✓
Obtención de histórico de precios	×	×	×	×
Geolocalización de productos	×	*	*	✓

* Indica apartados cuya aplicación es parcial o no se integra explícitamente.

Se puede observar como entre todas las aplicaciones, OCU Market es la más completa, debido a la implementación de su funcionalidad de mapa para encontrar supermercados cercanos a la ubicación del usuario y mostrar los precios del producto en los mismos. Sin embargo, la información de estos precios es escasa, y parece centrarse en supermercados de grandes superficies. Además, el rendimiento de la aplicación parece no ser adecuada en todos los dispositivos.

Por otra parte Yuka, aún teniendo funcionalidades más acotadas, posee una interfaz agradable, tiene un gran registro de productos en su base de datos, es fácil de usar, y parece estar mucho mejor optimizada a los dispositivos móviles.

Se desearía que la aplicación a desarrollar poseyera las funcionalidades de OCU Market, con la simpleza y rendimiento de Yuka. Además, ninguna recoge un histórico de precios con los que el usuario pueda estudiar la evolución de los mismos. Esto aportaría gran valor al sistema al ser una característica diferenciadora entre estas aplicaciones.

Capítulo 3

Base de datos

Como se comentó en secciones anteriores, la necesidad de la persistencia de datos para almacenar las entidades del sistema plantea la pregunta de qué bases de datos utilizar. Existen diferentes opciones, como Firebase o mySQL, entre otras. Dentro del tipo de bases de datos a escoger, existen dos alternativas:

- **Bases de datos relacionales** [25]: se basan en el modelo relacional, es decir, utiliza un esquema tabular, donde cada fila es un registro con una clave única, y las columnas contienen los atributos de los datos. Ejemplos de este tipo de bases de datos serían MariaDB, MySQL, PostgreSQL u Oracle Database.
- **Bases de datos no relacionales** [27]: también conocidas como NoSQL, se caracterizan por no usar tablas, sino colecciones de documentos. Esto hace que resulte sencillo el desarrollo de modelos, mayor escalabilidad y flexibilidad, y el posterior trabajo con los mismos. Algunos ejemplos podrían ser Cassandra, MongoDB o CouchDB.

3.1. MongoDB

Entendiendo las necesidades del sistema a desarrollar, se ha decidido por la utilización de una base de datos no relacional gracias a la facilidad y flexibilidad de su uso. Entre las posibilidades existentes, MongoDB [13] fue la decisión que se tomó para almacenar las entidades, siendo frecuentemente utilizado para el desarrollo de un API REST. Además, gracias a la librería de Mongoose [14], que se integra fácilmente con MongoDB y Node.js [17], permite la validación y el modelado de los datos, estableciendo relaciones entre los mismos, así como consultas propias que facilitan en gran medida el desarrollo.

3.2. Entidades del modelo de datos

El modelo de datos estará compuesto por cuatro entidades principales, siendo estas las listadas a continuación:

- `Account`: Para la autenticación de usuarios en el servidor, será necesario almacenar la información relacionada con las cuentas de los mismos. Los datos a modelar serán el nombre del usuario, su email, contraseña, y su rol. Los roles de los usuarios podrán ser de administrador, o de usuario regular. La diferencia entre ellos serán

los recursos y peticiones que tendrán disponible en el API, como la modificación o borrado de los datos.

- `Shop`: Guardar la información de las tiendas permitirá saber los productos que se encuentran en cada una y en que lugares se pueden encontrar. Por ello, se guardará el nombre de la tienda, sus productos, y la localización en la que se encuentra, almacenada a modo de latitud, longitud y una dirección opcional, esto es, lo esencial para poder geolocalizar la misma.
- `Product`: La información de un producto vendrá dada por su código de barras distintivo, nombre, marca, ingredientes, nutrientes, un indicador para saber si son bebidas, así como su NutriScore.
- `Receipt`: El recibo será el indicador del precio de un producto en una tienda en una fecha determinada. Se almacenará así el producto, precio, tienda en la que fue encontrado, el día y hora, así como el usuario que lo encontró. Esto nos permitirá realizar un seguimiento del precio de un producto a lo largo del tiempo en las distintas tiendas donde los usuarios lo encuentren.

También se han creado otras dos entidades para facilitar las abstracciones de los datos, siendo estas:

- `Location`: Asignada a una única tienda, almacenará la latitud, longitud y dirección de la tienda.
- `Nutrients`: Indica la información nutricional relevante sobre un producto, tanto sus nutrientes como micro-nutrientes y minerales.

3.3. Información nutricional de productos

En todos los alimentos envasados existe una etiqueta obligatoria con la información alimentaria [3], que indican las sustancias que poseen. Esto permite realizar la compra informada de los productos y sus contenidos.

Esta información con el aporte calórico y los nutrientes se muestra habitualmente en el siguiente orden:

- Valor energético, en Kcal (calorías).
- Grasas totales.
- Grasas saturadas.
- Hidratos de carbono.
- Azúcares.
- Proteínas.
- Sal.

Obligatorios	
Nutrientes	**Unidad**
Calorías	Kilocalorías
Grasas totales	Gramos
Grasas saturadas	Gramos
Hidratos de carbono	Gramos
Azúcares totales	Gramos
Proteína	Gramos

Tabla 3.1: Nutrientes a considerar como obligatorios y sus medidas.

Opcionales	
Nutrientes	**Medida**
Grasas trans	Gramos
Azúcares añadidos	Gramos
Sal	Gramos
Sodio	Miligramos
Fibra alimentaria	Gramos
Colesterol	Miligramos
Vitamina D	Microgramos
Calcio	Miligramos
Hierro	Miligramos
Potasio	Miligramos

Tabla 3.2: Nutrientes y micronutrientes a considerar como opcionales y sus medidas.

Particularmente, se puede añadir información adicional, como fibra alimentaria, el colesterol, otras grasas, y micronutrientes como vitaminas y minerales como el hierro. De esta manera, el modelo de datos debería de considerar todos los posibles nutrientes y micro-nutrientes que las etiquetas de los productos contengan, tanto de manera obligatoria como opcional, por si algún usuario quisiera introducir esta información.

También se tiene en consideración que en muchas regiones esta información nutricional no se presente en el mismo orden, ni que los nutrientes aquí obligatorios lo sean en otros lugares. Por ello, se debe tener en cuenta los nutrientes fundamentales y comunes entre las etiquetas de diferentes regiones, relegando como opcionales aquellas que no lo sean.

En la Tablas 3.1 y 3.2 se exponen los nutrientes y micronutrientes (con sus respectivas medidas) que se pueden encontrar en la mayoría de etiquetas en los alimentos envasados, y los que se tendrán en consideración como obligatorios para el modelado de los datos.

Cuando se realiza la declaración de propiedades nutritivas en las etiquetas, el contenido se expresa por porción y por 100 gramos. Para la estandarización de las medidas, se tomarán las cantidades por 100 gramos como referencia.

A su vez, la información sobre vitaminas y sales minerales se expresa tanto en gramos como por porcentaje de las cantidades diarias recomendadas (CDR). En este caso, se tomarán los gramos como medida.

3.4. Códigos de barras

Los códigos de barras son identificadores que distinguen de manera inequívoca un producto. Esto permite saber la producción, origen y destino del mismo, y facilita su circulación en establecimientos.

Entre los sistemas de código de barras, el estándar internacional para productos alimenticios es el EAN (European Article Numbering), creado en Europa, y compatible con el sistema Universal Product Code (UPC) que se utiliza en América del Norte [28].

El sistema EAN-13 es la versión más extendida, y consta de 13 cifras. Por su parte, el UPC (normalmente referido a UPC-A) abarca 12 dígitos. Dada las necesidades del sistema para la identificación única de productos alimenticios, se debe tener en consideración ambas codificaciones para el modelo cuando los usuarios ingresen un nuevo producto.

3.5. Modelado de objetos con Mongoose

A continuación, se modelarán y definirán las validaciones a cumplir por cada una de las entidades de la base de datos, donde se indicarán los tipos de datos de sus atributos, el requerimiento obligatorio u opcional de los mismos, así como otras restricciones a cumplir.

3.5.1. Modelado de usuarios

La entidad de usuarios posee cuatro atributos:

- `Username`: Será el identificador único de los usuarios, y será por tanto requerido en todos. Gracias a Mongoose y su opción de `validate`, se ha podido definir una restricción para aceptar como válido un nombre de usuario entre cuatro a veinte caracteres.
- `Email`: El email de los usuarios también será requerido, y nuevamente se ha añadido una restricción para sólo aceptar correos electrónicos que se puedan considerar como válidos gracias a una expresión regular.
- `Password`: La contraseña de los usuarios será requerida y validada para considerarse como segura, esto es, con un mínimo de ocho caracteres, con al menos un número, mayúscula y minúscula gracias a una expresión regular.
- `Role`: Los roles de los usuarios se validan gracias a la opción `enum` de Mongoose, que permite que el atributo tome el valor de un array de valores. En este caso, se han indicado los valores de un enumerado con roles de regular y administrador. Por defecto, cualquier usuario tendrá el rol de regular.

3.5.2. Modelado de tiendas

Las tiendas tienen las siguientes propiedades:

- `Name`: Los nombres de la tienda serán los identificadores únicos de cada una.

- `Products`: Los productos que una tienda posea se almacenan mediante un array de IDs de la entidad de productos que se encuentren en la misma. MongoDB proporciona un ID a cada objeto del dominio que actúa como su propio identificador único, y con el que se puede establecer este tipo de relaciones.
- `Location`: actúa como referencia hacia la ID de la localización donde se encuentra la tienda, que almacena principalmente su latitud, entre los valores numéricos de -90 a 90, y su longitud, entre -180 y 180. Estas restricciones son posibles gracias a las opciones de `min` y `max` de Mongoose. También existe un campo de dirección que no es obligatorio.

3.5.3. Modelado de productos

El modelado de productos requiere de:

- `Barcode`: El código de barras actuará como identificador único de cada producto. Gracias a la propiedad `unique` de Mongoose, dos productos no podrán tener un mismo identificador. Los códigos de barras son el EAN-13 y UPC-A, que abarcan de 12 a 13 dígitos y se validan con una expresión regular y la propiedad `match`.
- `Name`: El nombre del producto será necesario para que los usuarios puedan identificarlo y deberá ser obligatorio.
- `Brand`: La marca de los productos se deberá de indicar obligatoriamente.
- `Nutrients`: Los nutrientes de un producto se relacionarán por una referencia de su ID. Los obligatorios serán aquellos indicados en la Tabla 3.1.
- `Beverage`: Un indicador obligatorio que especifica si un producto es una bebida o no según sus valores a `true` o `false`.
- `NutriScore`: La puntuación del NutriScore es opcional, pero dada su aportación, deberán tomar los valores de un enumerado, entre las letras A, B, C, D, o E.

3.5.4. Modelado de recibos

Por último, para el modelado de recibos existen las propiedades de:

- `Price`: El valor numérico y requerido del precio de un producto.
- `Date`: Fecha en el que se encontró el producto, representada con el tipo `Date` de JavaScript.
- `Product`: Referencia por ID a la instancia del producto que se encontró.
- `Shop`: Referencia por ID a la instancia de la tienda donde se encontró el producto.
- `User`: Referencia por ID que indica el usuario que encontró el producto.

De esta manera, todas las entidades del dominio quedan modeladas para su uso. Para cualquier operación, Mongoose se encargará de ejecutar las validaciones necesarias, impidiendo que los objetos incumplan sus restricciones.

3.6. Despliegue en MongoDB Atlas

Para el acceso a la base de datos en cualquier entorno de producción, esto es, que los usuarios utilicen, será necesario su despliegue. Al no disponer de una infraestructura física, la mejor opción es confiar en algún servicio en la nube que maneje la complejidad de su despliegue y administración. Para MongoDB, existe un servicio llamado MongoDB Atlas.

Para su uso, es necesario el registro en la plataforma. Una vez se registra, se crea una organización y un clúster para ejecutar el servidor de base de datos MongoDB. Entre las diferentes opciones y dado el alcance de este proyecto, se ha seleccionado un clúster compartido, que es el modelo gratuito que MongoDB Atlas ofrece.

Cuando se crea, se debe especificar un usuario administrador de la base de datos, así como las IPs de conexión a la misma. MongoDB Atlas proporciona una URI de conexión a la base de datos, que contiene la contraseña con la que se harán las peticiones a la base de datos, y que debe mantenerse oculta como secreto en el servidor, lo que se explicará en secciones posteriores.

De esta manera, quedan completamente definidas las validaciones y configuraciones realizadas en la base de datos.

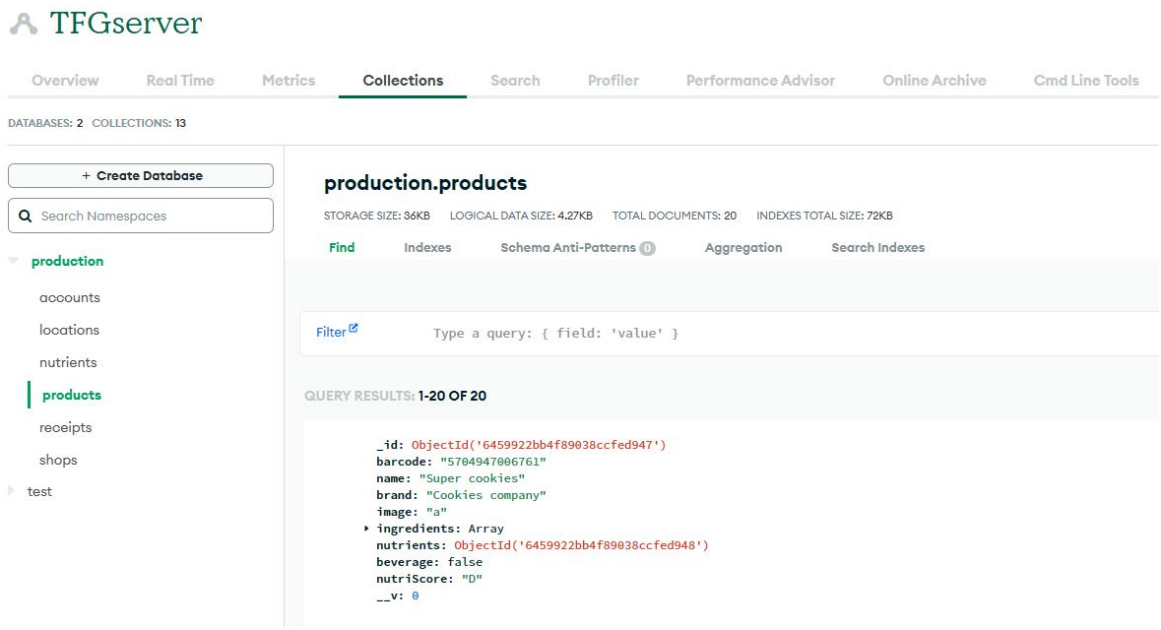


Figura 3.1: Colecciones de datos en MongoDB Atlas

Capítulo 4

Desarrollo del API

Como se comentó en capítulos anteriores, se requiere de la creación de un API REST que permita a las aplicaciones cliente acceder y realizar las diferentes operaciones CRUD [26] correspondientes sobre la base de datos MongoDB. En el entorno de Node.js, el framework de Express [6] es uno de los más populares para la creación de servidores web y APIs, y el que se usará para este proyecto.

Con Express, se instanciará el servidor y se habilitarán las diferentes rutas que administrarán las peticiones de los clientes. Todo ello, con el lenguaje de programación de TypeScript.

4.1. API REST

Un API [24] describe un intermediario entre dos sistemas, que permite a una aplicación comunicarse con otra y realizar operaciones específicas. De esta manera, se pretende crear un servidor web que sirva un API con información en formato JSON, y permita a las aplicaciones clientes acceder y trabajar con los recursos de la base de datos.

Las operaciones que permitirá realizar este sistema entran dentro de lo que se conoce como un diseño REST (transferencia de estado representacional). Las principal característica de un API REST es que permite las operaciones denominadas como CRUD, esto es, creado (*create*), lectura (*read*), actualización (*update*) y borrado (*delete*) de los datos.

Además, esta arquitectura permite el desacoplamiento del cliente y servidor, siendo completamente independientes entre sí. El cliente realiza peticiones HTTP hacia la URI del servidor, pudiendo ser peticiones para lectura (GET), creación de recursos (POST), actualización de los mismos (PATCH) y su borrado (DELETE). Cada solicitud debe incluir toda la información necesaria para que pueda ser procesada, por lo que el API no guarda ningún tipo de estado y no almacena información relacionada con una solicitud del cliente.

Se puede apreciar así un sistema en capas, como se muestra en la Figura 4.1, donde el servidor recibe las peticiones del cliente, e indica las operaciones necesarias sobre la base de datos, enviando la respuesta acorde.

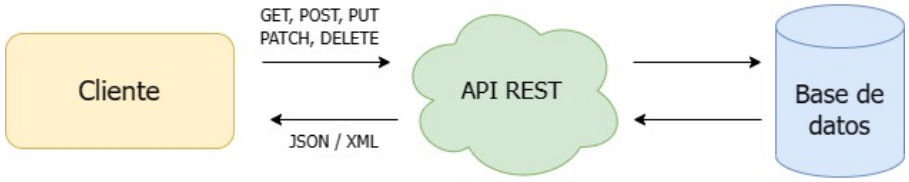


Figura 4.1: Esquema de API REST

4.2. Estructura del proyecto

El proyecto del API REST del backend[1] se estructura como se muestra en la Figura 4.2.

El fichero `mongoose.ts` contiene la instancia de la base de datos con la que se establece la conexión a la URI de MongoDB Atlas, la cual se ha almacenado en una variable de entorno local en un fichero oculto `.env`. Esta URI no debe ser expuesta por seguridad, y en GitHub la variable debe ser almacenada en los secretos del repositorio, siendo estos inaccesibles para cualquier usuario que no sea administrador del mismo.

La carpeta `middleware` contiene toda la lógica que se procesa entre la recepción de la petición de un cliente, y la ruta que se va a encargar de procesarla y enviar la respuesta. Como se verá más adelante, aquí es donde se harán las validaciones de la autenticación y permisos de los usuarios a través de tokens y roles correspondientemente.

En `models` se almacena toda la lógica de modelado y las validaciones de Mongoose comentadas en secciones anteriores. Las funciones empleadas a lo largo del código se residen en `utils`, como las validaciones a través expresiones regulares de correos electrónicos o de una contraseña segura, así como el encriptado y desencriptado de estas últimas.

Por último, en `routers` se encuentran todos los *endpoints* disponibles para que los usuarios puedan interactuar con los datos y realizar las peticiones al servidor. Las rutas que recibirán las peticiones serán sobre las entidades de cuentas (`account.ts`), productos (`product.ts`), recibos (`receipt.ts`) y tiendas (`shop.ts`). También se habilita una ruta por defecto (`default.ts`) que recibirá todas aquellas peticiones a rutas que no están disponibles. En capítulos posteriores se discutirá sobre los distintos códigos de estado de respuesta HTTP que el servidor devuelve, así como las diferentes rutas disponibles.

4.3. Conexión con la base de datos

El paquete de MongoDB proporciona el *driver* para interactuar con un servidor MongoDB desde NodeJS. La conexión con la base de datos se declara en el fichero de `mongoose.ts`, que utiliza la URI de MongoDB almacenada en una variable de entorno del fichero `.env` como secreto. Se posee también una URI para la base de datos dedicada a las pruebas del backend, de las que se hablarán más adelante.

```
const { MONGO_DB_URI, MONGO_DB_URI_TEST, NODE_ENV } = process.env;
```

[1] Repositorio del proyecto del API: `https://github.com/tanafc/tfg-backend`

Figura 4.2: Estructura de ficheros del proyecto del API REST.

Una vez se tiene la URI, para instanciar la conexión a la base de datos de MongoDB, simplemente se utiliza el método de `connect` del paquete de mongoose, que recibe la URI y las opciones de la conexión.

```
mongoose
    .connect(connectionUrl, {
        autoIndex: true,
  })
  .then(() => {
    console.log("Connection to MongoDB server established");
  })
  .catch((err) => {
    console.log("Unnable to connect to MongoDB server");
    console.log(err);
  });

const db = mongoose.connection;

export default db;
```

Se exporta la base de datos para poder iniciar la conexión cuando el servidor de Express se instancia, además de ser más sencillo a la hora de realizar las pruebas.

4.4. Servidor Web de Express

Como se comentó anteriormente, Express permite la creación de servidores Web y APIs en Node.js de manera muy sencilla. Para iniciar un servidor en Express, se debe instanciar desde el propio método de `express`. Esto se realizará desde el fichero de `app.ts`, donde además se importarán todos los `routers` de la aplicación, que contienen las rutas que procesan las peticiones de los clientes, y se indica al servidor de Express que utilice dichas rutas mediante el método `use`.

```
import * as express from "express";
import { accountRouter } from "./routers/account";
import { defaultRouter } from "./routers/default";
import { productRouter } from "./routers/product";
import { receiptRouter } from "./routers/receipt";
import { shopRouter } from "./routers/shop";

const app = express();

app.use(express.json());
app.use(accountRouter);
app.use(shopRouter);
app.use(productRouter);
app.use(receiptRouter);
app.use(defaultRouter);

export default app;
```

Una vez más, se exporta la aplicación de Express para ser iniciada desde el fichero de `index.ts`. En este fichero, será donde se importe la instancia de la base de datos de MongoDB, así como el servidor para empezar a recibir las peticiones mediante el método `listen` desde el puerto de elección, que en este caso, se le indica con una variable de entorno, siendo por defecto el puerto 8000.

```
import app from "./app";
import "./database/mongoose";

const port = process.env.PORT || 8000;

const server = app.listen(port, () => {
  console.log(`Server up on port ${port}`);
});

export default server;
```

De esta manera, desde el fichero `package.json`, que contiene los paquetes instalados y las dependencias del proyecto, así como los scripts a ejecutar, se declara un script `start` que permite iniciar el servidor. En el mismo, se declara también la variable `NODE_ENV` que indica el entorno de ejecución, para distinguir entre la base de datos de producción y la que ejecuta las pruebas en desarrollo.

```
"start": "NODE_ENV=production node dist/index.js",
"test": "NODE_ENV=test mocha --timeout 10000 --exit",
```

Gracias a Node Package Manager, con el comando `npm start` se iniciará el servidor, conectando con la base de datos de MongoDB y empezando a escuchar las peticiones en el puerto designado.

4.5. Códigos de respuesta HTTP

Cada petición HTTP a uno de los *endpoints* del API devolverá los datos acordes al mismo, así como un código de estado de respuesta HTTP [11] que indica si se ha completado satisfactoriamente o no la petición. Para entender las diferentes respuestas que el API devuelve, se exponen algunos de los códigos de respuesta utilizados y que se encuentran frecuentemente en servidores web.

4.5.1. Respuestas satisfactorias

Las respuestas que se engloban en el rango de 200-299 indican peticiones resueltas satisfactoriamente. En el caso del API, se emplean los diferentes códigos:

- **200 OK**: Indica que la solicitud ha tenido éxito, variando su significado dependiendo del método HTTP, ya sea GET, POST, PATCH o DELETE.
- **201 Created**: La solicitud ha tenido éxito y se ha creado un nuevo recurso dada la petición. Es utilizada cuando se procesan peticiones de POST.

4.5.2. Errores de los clientes

El rango de códigos 400-499 definen peticiones que han sido procesadas por el servidor, pero que no han sido correctamente formuladas o percibidas como error por parte el cliente. Algunos de los errores utilizados son:

- **400 Bad Request**: El cliente ha enviado una petición con una sintaxis inválida, que el servidor ha sido incapaz de interpretar.
- **401 Unauthorized**: Se devuelve para aquellas peticiones cuyo cliente no tiene los permisos necesarios para acceder, y para los cuales es necesario estar autenticado. Este código se da en el API cuando el usuario no posee el rol adecuado, o su token de sesión ha caducado.
- **404 Not Found**: El servidor no ha podido encontrar el contenido solicitado por la petición. Uno de los más recurrentes en la web. En el caso del API, por la inexistencia del recurso en la base de datos.
- **409 Conflict**: Se indica a peticiones que tienen conflicto con el estado del servidor. En el API, es utilizado cuando se intenta crear algún recurso mediante POST que ya existe en la base de datos, como un producto con el mismo código de barras, o tienda con el mismo nombre.

4.5.3. Errores del servidor

Por último, los códigos de respuesta en el rango 500-599 indican errores inesperados por parte del servidor debido a la petición recibida. En el API se pueden encontrar los siguientes:

- **500 Internal Server Error**: Indica que el servidor ha encontrado una situación incapaz de manejar. El API responde con este código al haber un fallo al realizar operaciones sobre la base de datos MongoDB, al ser incapaz de acceder a ella, u otro caso.
- **501 Not Implemented**: La petición solicitada no está soportada por el servidor o no existe. Cuando se envía una petición al API a una ruta inexistente, la ruta por defecto `default` capturará dicha petición y devolverá este código de estado.

Estos serían todas los códigos de respuesta manejados por el API REST implementada.

4.6. Rutas del API REST

A continuación, se mostrarán los puntos de acceso (*endpoints*) disponibles en el API, con los que se interactuará con los recursos del servidor para cada una de las entidades. La creación de rutas en un servidor de Express es muy sencillo, y requiere de la declaración de una instancia (`Router`) con los métodos `get`, `post`, `delete` y `patch`.

4.6.1. Rutas de cuenta

- `GET /account`: Obtiene información de la cuenta del usuario. Es necesaria la autorización por token.
- `POST /signup`: Registro del usuario en la aplicación.
- `POST /login`: Inicio de sesión en la aplicación.
- `PATCH /account`: Cambio de información de la cuenta para nombre de usuario, contraseña y email. Es necesario el token de autorización.
- `DELETE /account`: Borrado de la cuenta. Es necesario el token de autorización.

4.6.2. Rutas de productos

- `GET /products`: Devuelve información sobre un producto por su código de barras. Es necesario token de autorización.
- `GET /products/:id`: Devuelve información sobre un producto por su ID. Es necesario el token de autorización.
- `GET /products-all`: Responde con una lista de códigos de barras y nombres de productos similares a la cadena que se le envía. La autorización por token es necesaria.
- `POST /products`: Crea un nuevo producto y sus nutrientes. Es necesaria la autorización por token.

- `DELETE /products`: Borrado de un producto y sus nutrientes. También se eliminan todos los recibos relacionados, así como las entradas del producto en las tiendas. Es necesario el token de autorización y el rol de administrador.
- `PATCH /products`: Actualización de los atributos de un producto. Las modificaciones posibles son el nombre, marca, ingredientes, nutrientes y NutriScore. Es necesario el token de autorización y el rol de administrador.

4.6.3. Rutas de tiendas

- `GET /shops`: Obtiene información de una tienda por su nombre. Es necesaria la autorización por token.
- `GET /shops/:id`: Obtiene información de una tienda por su ID. Es necesaria la autorización por token.
- `GET /shops-all`: Obtiene un listado de tiendas con nombres similares a la petición. Es necesaria la autorización por token.
- `POST /shops`: Crea una nueva tienda con su localización. Es necesaria la autorización por token.
- `POST /shops/products`: Añade un producto existente a una tienda, creando un nuevo recibo con su precio y fecha. Es necesaria la autorización por token.
- `DELETE /shops`: Borra una tienda y su localización. Reservado para usuarios con rol administrador.
- `DELETE /shops/products`: Elimina un producto de la lista de productos en una tienda. También se borran los recibos asociados. Reservado para usuarios con rol administrador.
- `PATCH /shops`: Modifica el atributo de nombre de una tienda. Reservado para usuarios con rol administrador.

4.6.4. Rutas de recibos

- `GET /receipts/:id`: Devuelve información sobre un recibo por ID. Es necesario token de autorización.
- `GET /newest-receipts`: Devuelve un listado de los últimos recibos registrados en las tiendas que disponen de un producto, indicando el código de barras en la petición. Es necesario un token de autorización.
- `GET /receipts`: Devuelve un listado de recibos que dependen de los parámetros que se le pasen a la petición. Estos pueden ser:
  - `shop`: nombre de una tienda.
  - `product`: código de barras de un producto.
  - `sdate` y `edate`: fechas iniciales y finales por las que filtrar los recibos.
  - `minprice` y `maxprice`: rango de precios por los que filtrar los recibos.

    - `limit`: limita el número de recibos que devuelve la petición.
    - `skip`: ignora un número de entradas determinado al buscar en la base de datos.
- `DELETE /receipts`: Elimina una lista de recibos que dependen del producto, la tienda, los filtros de fecha y el rango de precios. Esta reservado para usuarios con rol administrador.
- `DELETE /receipts/:id`: Elimina un recibo por su ID. Reservado para usuarios con rol administrador.

4.7. Autorización de usuarios

Para la securización de los datos en el API, se ha implementado una serie de *middlewares* que validan la autenticación de los usuarios gracias a la implementación de tokens de sesión, así como un sistema de roles que comprueba los permisos que cada usuario posee.

4.7.1. Validación de tokens de sesión

JSON Web Token (JWT) es un estándar de RFC 7519 [10] para la creación de tokens de acceso. Permite así la identificación de un determinado usuario, además de los privilegios del mismo. Se trata de una cadena con tres partes separadas por un punto, cada una con su propia información codificada:

- `Header`: indica el algoritmo (HS256) y tipo de token.
- `Payload`: almacena información relacionada con el usuario y sus privilegios, así como otra información a añadir.
- `Signature`: la comprobación de firma que valida el token.

Las ventajas de utilizar JWT a otros mecanismos de autenticación de usuarios es que el servidor no necesita almacenar ningún tipo de información. El servidor crea el token JWT con un secreto propio y lo manda al cliente, el cual deberá enviar las peticiones a recursos protegidos con el token en el encabezado. De esta manera, el servidor comprueba la firma del token, y en caso de ser válida, responde con dicho recurso.

El paquete `jsonwebtoken`[2] de `npm` permite la creación y firma (método `sign`) y verificación (método `verify`) de tokens JWT con múltiples opciones para la elección del algoritmo, el tiempo de expiración de dicho token, la información del payload, entre otras propiedades. En la implementación de JWT, se ha optado por un tiempo de expiración de dos horas, después de las cuales el servidor no validará el token.

El secreto que se utiliza para la firma de los tokens es almacenado en el fichero `.env`, que no debe ser expuesto, y en producción, se establece como secreto en el repositorio de GitHub.

[2] Librería de jsonwebtoken: `https://www.npmjs.com/package/jsonwebtoken`

4.7.2. Validación de roles

El acceso a recursos privilegiados de la aplicación, como pueden ser rutas destinadas a la actualización o eliminación de productos o tiendas, están protegidos por un sistema de verificación de roles, y solo esta disponibles para usuarios con rol administrador, y no regulares.

Cada petición a alguno de estos recursos invocará el middleware de autenticación de token JWT del fichero `authJwt.ts` para identificar al usuario, y una vez verificado, un siguiente *middleware* en `authRole.ts` examina su rol, denegando el acceso en caso de no poseer el privilegio.

4.7.3. Encriptación de contraseñas

Para el almacenamiento seguro de las contraseñas de los usuarios en la base de datos, el paquete de `bcrypt`[3] de `npm` ofrece múltiples funcionalidades con el fin de establecer un encriptado seguro de las mismas.

Mediante su método de `hash`, se puede encriptar la contraseña de manera asíncrona, y de esta manera almacenarla en la base de datos. Para verificar la contraseña del usuario cuando se reciba en alguna petición, se recupera el hash almacenado y se compara con la contraseña mediante el método `compare` de la librería.

[3]Librería de bcrypt: `https://www.npmjs.com/package/bcrypt`

Capítulo 5

Desarrollo de la aplicación móvil

Durante este capítulo se mostrará el desarrollo de la aplicación móvil, que implementará los requisitos definidos en la Sección 2.3, haciendo uso de los *endpoints* del API mostrados en la Sección 4.6.

El framework de React Native [20] permitirá la creación de la aplicación con soporte para las plataformas de Android e iOS. A su vez, el ecosistema de herramientas de Expo [5] asegura una experiencia de desarrollo agradable gracias a las utilidades que proporciona, como depuración, manejo de librerías y la posibilidad de ejecutar la aplicación desde un dispositivo gracias a la plataforma de Expo Go, donde se han realizado las pruebas del mismo.

5.1. Estructura del proyecto

La estructura del proyecto de la aplicación móvil[1] se muestra en la Figura 5.1.

En el directorio `api` se encuentran los ficheros con las llamadas a los distintos *endpoints* del API, almacenadas dentro de `account`, `products`, `shops` y `auth` para el inicio de sesión y registro. Estas peticiones se realizan con la ayuda de la librería de Axios[2].

Axios es un Cliente HTTP basado en promesas. Para su uso, en primer lugar se ha iniciado una instancia con la URL hacia el API, en el fichero `httpClient.ts`.

```
import axios from "axios";

const httpClient = () => {
  const httpClient = axios.create({
    baseURL: ""
  });

  return httpClient;
};
```

Para hacer uso de estas peticiones, se han creado una serie de *hooks*. Desde la propia documentación de React [19], los *hooks* permiten “enganchar el estado de React y el ciclo

[1] Repositorio del proyecto de la aplicación móvil: `https://github.com/tanafc/tfg-frontend-app`
[2] Librería de Axios: `https://axios-http.com/es/docs/intro`

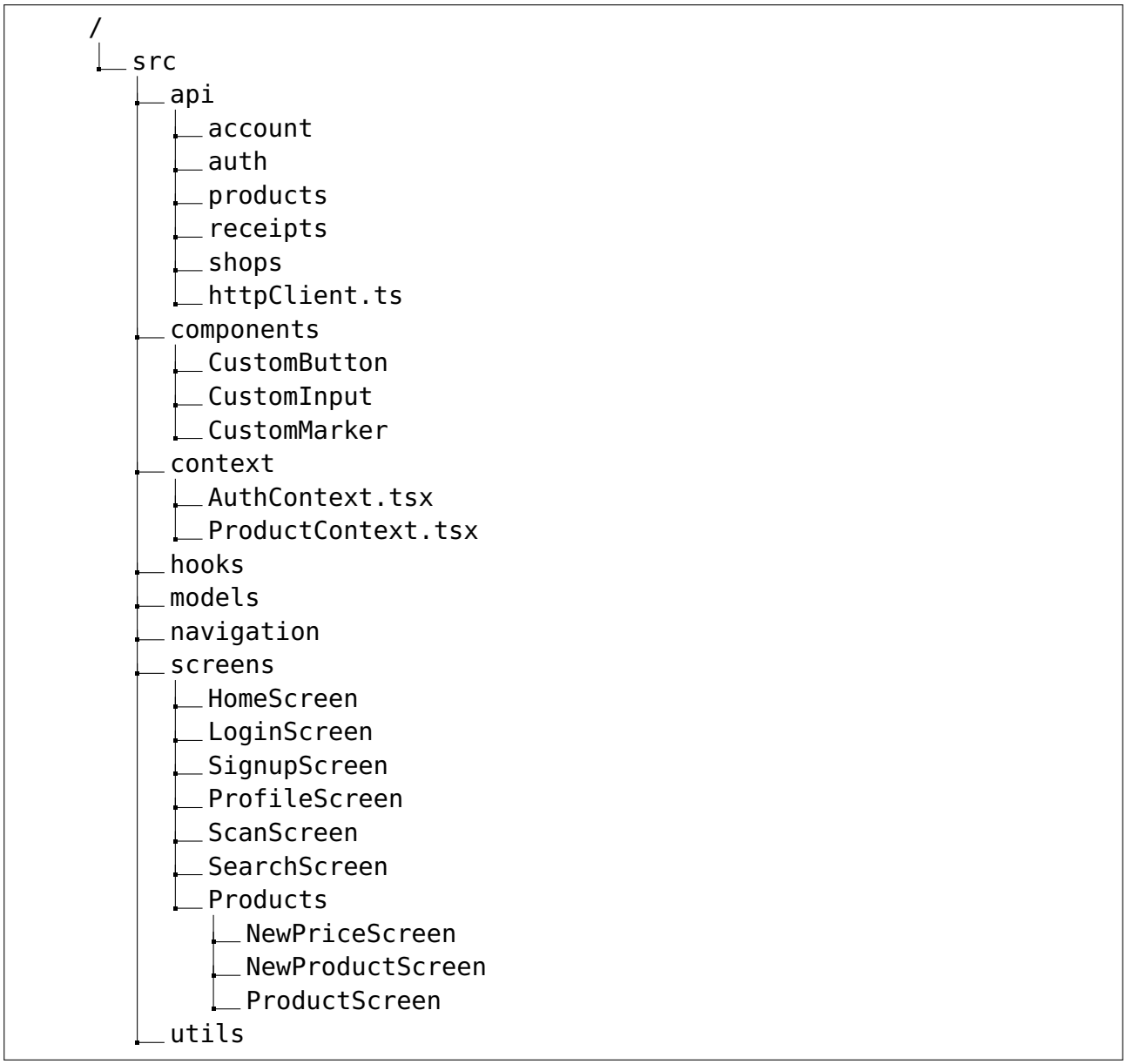


Figura 5.1: Estructura de ficheros del proyecto de la aplicación.

de vida desde componentes de función". Esto hace posible determinar el estado actual de los componentes y realizar las llamadas al API acordes a los datos de la aplicación. Todas las implementaciones de estos se pueden encontrar en la carpeta de `hooks`.

El directorio de `context` almacena todos los contextos con los estados de la aplicación que se deseen almacenar. En React, un contexto se utiliza para compartir estado que es considerado global, y que es utilizado en una multitud de componentes. En este caso, se han creado dos contextos, uno para almacenar los productos que han sido escaneados `ProductContext`, y otro para almacenar la sesión y la información de la cuenta del usuario `AuthContext`.

En `models` se pueden encontrar todos los tipos y enumerados del modelo de datos de la aplicación, características que TypeScript proporciona y que reduce el número de errores que se puede encontrar a lo largo del desarrollo. Los modelos incluyen los datos recibidos del API de productos, tiendas, recibos y cuentas, así como el tipado de la navegación.

Toda la lógica relacionada con la navegación se almacena en el directorio `navigation`. Gracias a la librería de React Navigation[3], se han podido añadir dos tipos de navegación al *stack* de pantallas. Un navegador nativo, el cual utiliza los botones para retroceder de los propios dispositivos, y un navegador de menú horizontal, el cual alterna sobre las principales páginas de *Home*, *Profile*, *Scan*, y *Search*. En la Figura 5.2 se muestra el menú horizontal con la navegación de las mismas.

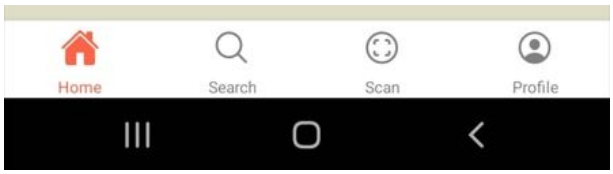


Figura 5.2: Menú de navegación de pantallas en la aplicación.

Por último, en el directorio de `screens` se almacenan los componentes de React con todas las pantallas de la aplicación, que hacen uso de los componentes generales de la carpeta `components`. En la sección posterior se visualizará el resultado de estos.

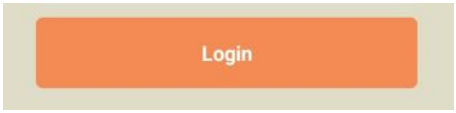
5.2. Componentes de la aplicación

Las pantallas de la aplicación hacen uso de componentes que se reutilizan a lo largo de las mismas. Estos son botones, campos de texto, y marcadores para el mapa de localizaciones de precios en tiendas. A continuación, se exponen estos componentes personalizados.

5.2.1. Botones personalizados

Se muestran los diseños de los botones, así como sus diferentes estados.

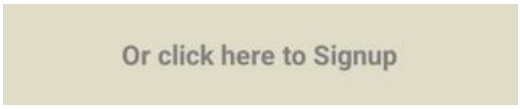
[3]Librería de React Navigation: `https://reactnavigation.org/`



(a) Botón primario habilitado

(b) Botón primario deshabilitado

(c) Botón primario con estado *loading*

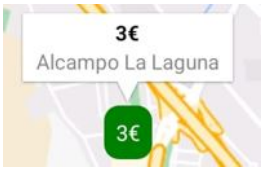


(d) Botón secundario

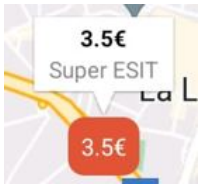
Figura 5.3: Estilo y estados de los botones

5.2.2. Marcadores personalizados

En el mapa de localizaciones de precios, se han creado marcadores personalizados para la visualización de precios. Las variaciones de estos son los siguientes.



(a) Marcador de precio más barato



(b) Marcador de precio

Figura 5.4: Estilos y estados de los marcadores

El marcador de color verde indica el precio del producto más barato entre todos los supermercados, mientras que el color naranja fuerte indica el resto.

5.2.3. Iconos de la aplicación

Para los iconos de la aplicación, se ha hecho uso de Ionicons[4], una librería de iconos de código abierto que se puede utilizar tanto en aplicaciones web, iOS, Android y en escritorio, siendo muy sencillo su uso en Expo al estar integrado en el propio ecosistema de herramientas. Tiene una licencia MIT y está construido por Ionic.

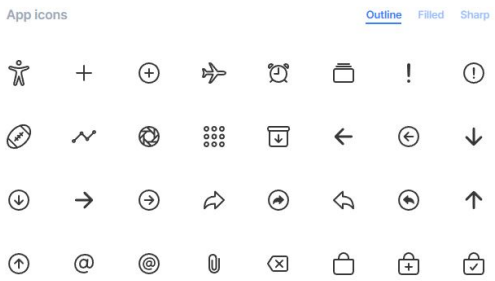


Figura 5.5: Iconos de la librería Ionicons

[4]Ionicons: `https://ionic.io/ionicons`

5.3. Escaneo de productos

Con el fin de poder escanear los códigos de barras de los productos con la cámara de los dispositivos móviles, se ha hecho uso de una herramienta de Expo llamada `expo-barcode-scanner`[5].

Este módulo proporciona un componente de React que renderiza un visor para la cámara del dispositivo (frontal o posterior), y escanea los códigos de barras que aparecen en el marco. Para mejorar el rendimiento del módulo se pueden indicar los formatos de códigos a escanear. En este caso, y como se vió en la Sección 3.4, solo se necesita escanear la codificación referida a UPC-A y UPC-EAN, lo que se puede indicar al componente `BarCodeScanner` con los parámetros `upc_a`, `upc_e` y `upc_ean`, que se declara en la pantalla de `Scan` del menú principal.

```
<BarCodeScanner
  barCodeTypes={[
    BarCodeScanner.Constants.BarCodeType.upc_a,
    BarCodeScanner.Constants.BarCodeType.upc_e,
    BarCodeScanner.Constants.BarCodeType.upc_ean,
  ]}
  onBarCodeScanned={scanned ? undefined : handleBarCodeScanned}
  style={[StyleSheet.absoluteFillObject, styles.container]}
>
  {/** Se incluye un marco para el enfoque */}
  <View style={styles.layerTop} />
  <View style={styles.layerCenter}>
    <View style={styles.layerLeft} />
    <View style={styles.focused} />
    <View style={styles.layerRight} />
  </View>
  <View style={styles.layerBottom} />
</BarCodeScanner>
```

Además, permite añadir componentes dentro del visor del escáner, siendo posible incluir un marco para ayudar al usuario a enfocar el código de barras en el centro del encuadre.

5.4. Gráficas de precios

La visualización del histórico de precios es posible gracias a la ayuda de la librería de Victory. Esta librería ofrece componentes de React para gráficos modulares y de visualización de datos. Tiene su propia implementación en React Native llamada Victory Native[6].

Victory Native permite así renderizar gráficas de datos importando los componentes correspondientes. Para la visualización de los diagramas en la aplicación, se utiliza el gráfico de líneas `VictoryLine` y el diagrama de dispersión `VictoryScatter`. Se pueden

[5]Librería de escáner: `https://docs.expo.dev/versions/latest/sdk/bar-code-scanner/`
[6]Librería de Victory Native: `https://formidable.com/open-source/victory/docs/native/`

emplear los dos tipos de gráficas para el mismo conjunto de datos, que en este caso, serían los precios en el eje vertical y las fechas en el eje horizontal, indicando así la evolución de los precios de un producto.

5.5. Geolocalización de tiendas y precios

Entre las herramientas que proporciona Expo, existe una librería para determinar la localización del dispositivo, llamada `expo-location`[7]. Esta proporciona acceso para leer la información de geolocalización, sondear la ubicación actual o suscribir eventos de actualización de la ubicación del dispositivo.

Es necesario requerir de los permisos de geolocalización del dispositivo, lo cual es posible con los métodos de `requestForegroundPermissionAsync`, que hará una petición para solicitar la información. En caso de que el usuario acepte los permisos, el método de `getCurrentPositionAsync` de la librería devolverá un objeto `LocationObject`, que dispondrá de la latitud, longitud y altitud del dispositivo.

Esto permite realizar el cálculo de una región donde se encuentra el usuario, tomando sus coordenadas de latitud y longitud actuales, y aplicando un valor delta a cada magnitud. Con ello, se tendrá una zona de búsqueda donde mostrar las localizaciones de tiendas con el producto. Sin embargo, es necesario el uso de otra librería para renderizar una vista de mapa donde sean observables, llamada `react-native-maps`[8]. Gracias a su componente de `MapView`, proporciona una vista de Google Maps, a la que se le puede indicar una región inicial, así como marcadores personalizados.

Algo a tener en cuenta, es que mientras su uso es gratuito mediante la aplicación de Expo Go, en un entorno de producción donde se despliegue la aplicación esta funcionalidad de Google Maps no funcionaría, debido a la necesidad de un API key que es proporcionada por el servicio de Google Cloud API.

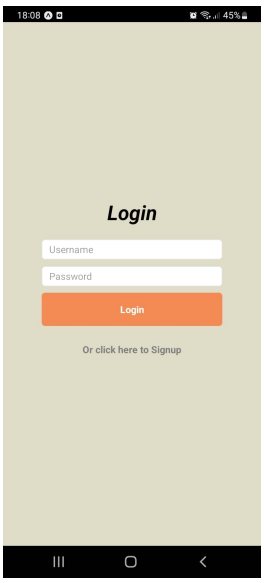
5.6. Pantallas y casos de uso

A continuación, se exponen las pantallas de la aplicación, así como los casos de uso de la misma.

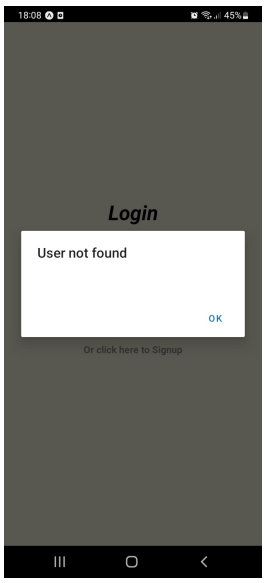
5.6.1. Iniciar sesión en la aplicación

La pantalla de inicio de sesión será la primera pantalla del *stack* de navegación, y la que se presentará al usuario en un primer momento.

[7] Librería de ubicaciones: `https://docs.expo.dev/versions/latest/sdk/location/`
[8] Librería de mapas: `https://docs.expo.dev/versions/latest/sdk/map-view/`



(a) Campos de inicio de sesión

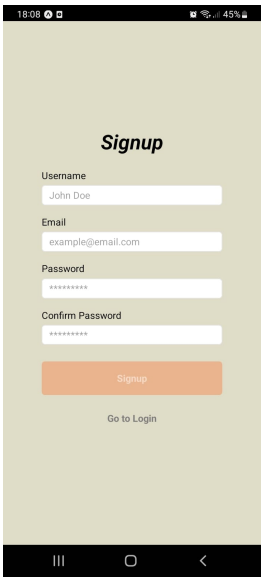


(b) Inicio de sesión fallido

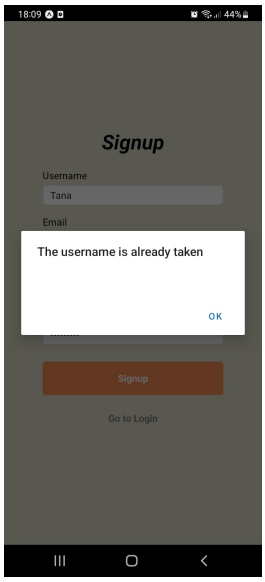
Figura 5.6: Campos de inicio de sesión

5.6.2. Registrarse en la aplicación

La pantalla de registro de la aplicación consistirá de varios campos de texto donde el usuario ingresará el nombre, email y contraseña válidos, con un campo extra para repetir esta última. En caso de que el nombre de usuario ya esté en uso, la aplicación notificará del caso.



(a) Campos de registro



(b) Registro de cuenta fallido

Figura 5.7: Pantalla de registro

5.6.3. Ver productos escaneados

Cuando el usuario inicia sesión se encontrará con la página de inicio, donde se mostrarán todas las entradas de los productos que ha escaneado en la sesión, pudiendo entrar en las pantallas individuales de cada producto con su información.

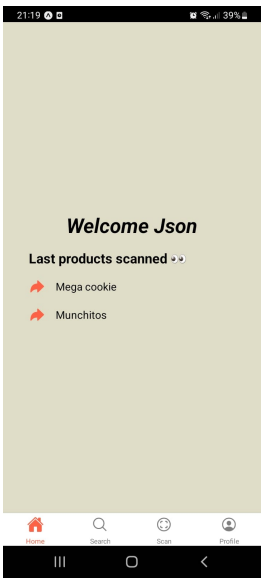


Figura 5.8: Pantalla de inicio

5.6.4. Ver perfil de cuenta

En la pantalla de perfil, el usuario podrá observar la información relativa a su cuenta, así como realizar el cierre de su sesión para volver a la pantalla de inicio de sesión.

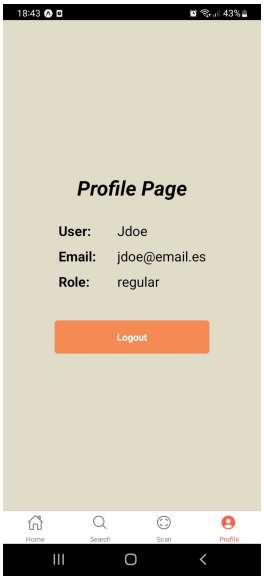
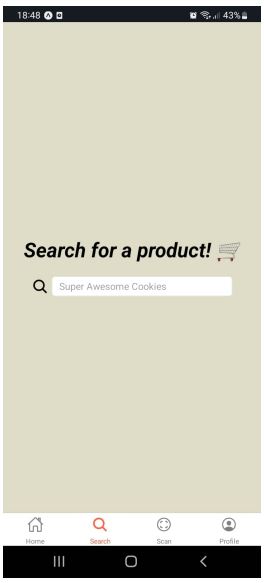


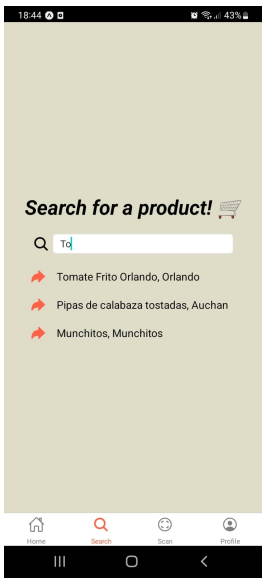
Figura 5.9: Pantalla de perfil

5.6.5. Buscar productos

Para la búsqueda de productos, el usuario puede ingresar el nombre del producto en un buscador, el cual mostrará los resultados disponibles. Si se toca en alguno de los resultados, la aplicación navegará hasta la página del producto con toda la información del mismo.



(a) Resultados vacíos



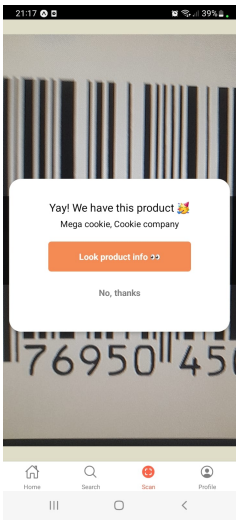
(b) Resultados de la búsqueda

Figura 5.10: Pantalla de búsqueda de productos

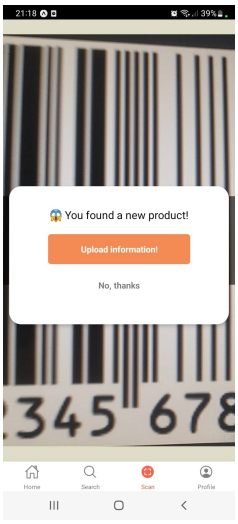
5.6.6. Escanear productos

La pantalla de escaneo de producto requerirá de los permisos del dispositivo a la cámara. Si estos son otorgados, se mostrará el visor de la cámara con el marco para centrar el código de barras.

(a) Marco de escáner



(b) Producto disponible



(c) Nuevo producto

Figura 5.11: Pantalla de escáner

Si se dispone de la información relativa al producto escaneado, se le indicará al usuario y se le guiará hacia la pantalla de información del producto. En caso contrario, se le pedirá ingresar la información sobre el mismo.

5.6.7. Ver información del producto

En la pantalla de información de un producto, existen varios selectores con los que el usuario puede interactuar para mostrar información correspondiente a datos nutricionales, los precios del producto en tiendas, y la localización de los mismos.

Información nutricional

Se muestra toda la información sobre la marca, nutrientes, micro-nutrientes, y su puntuación de NutriScore si ha sido indicada. Los datos nutricionales opcionales disponibles dependerán de los aportados por los usuarios.

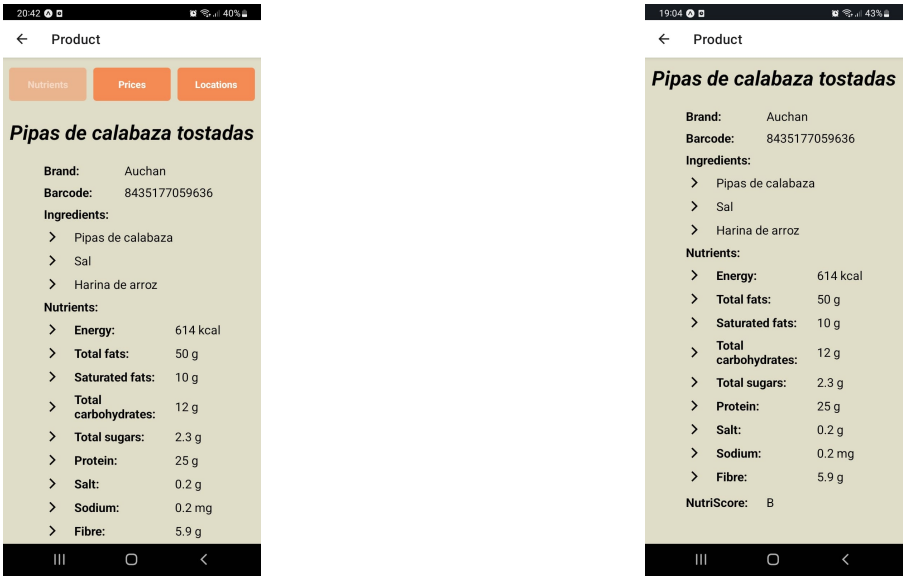
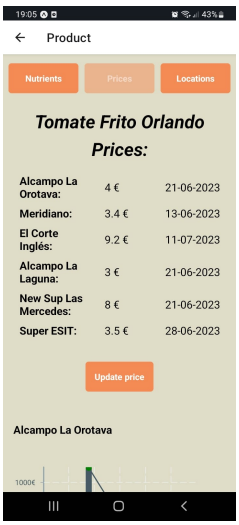


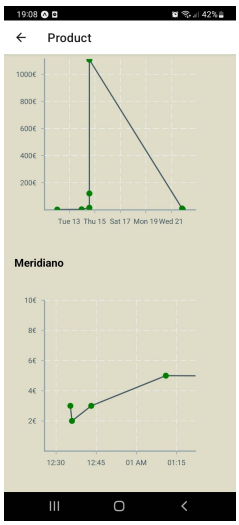
Figura 5.12: Pantalla de información nutricional

Información de precios

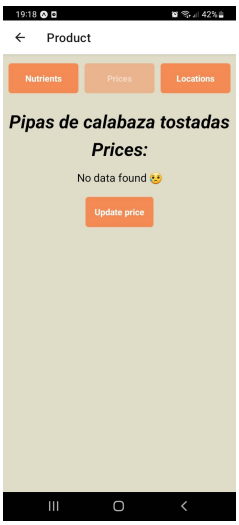
La pantalla de precios se divide en dos secciones. En la parte superior, se indican los últimos precios actualizados en cada una de las tiendas en la que están disponibles. En la parte inferior, se incluyen gráficos de líneas con el historial de los precios en cada una de las tiendas, siendo estas responsivas para que el usuario pueda desplazarse y ampliar los datos en las mismas.



(a) Últimos precios



(b) Gráfico de líneas



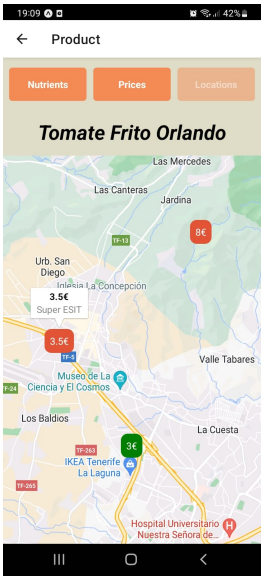
(c) Sin información disponible

Figura 5.13: Pantalla de precios del producto

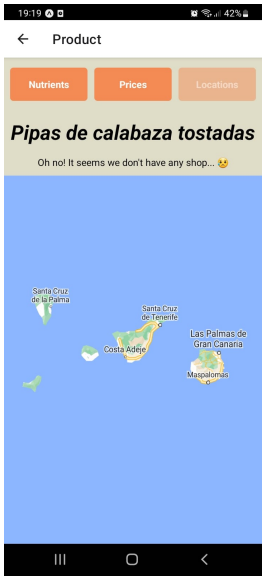
El botón de *Update price* llevará al usuario a una pantalla para actualizar el precio del producto si lo desea.

Información de localizaciones

En la pantalla de localizaciones, si se dispone de los permisos de geolocalización del dispositivo, los usuarios podrán ver un mapa de la región cercana a su zona donde se podrán desplazar y observar los marcadores con los precios de las tiendas. Si no se dispone de los permisos, se ha dispuesto una zona por defecto.



(a) Mapa con marcadores

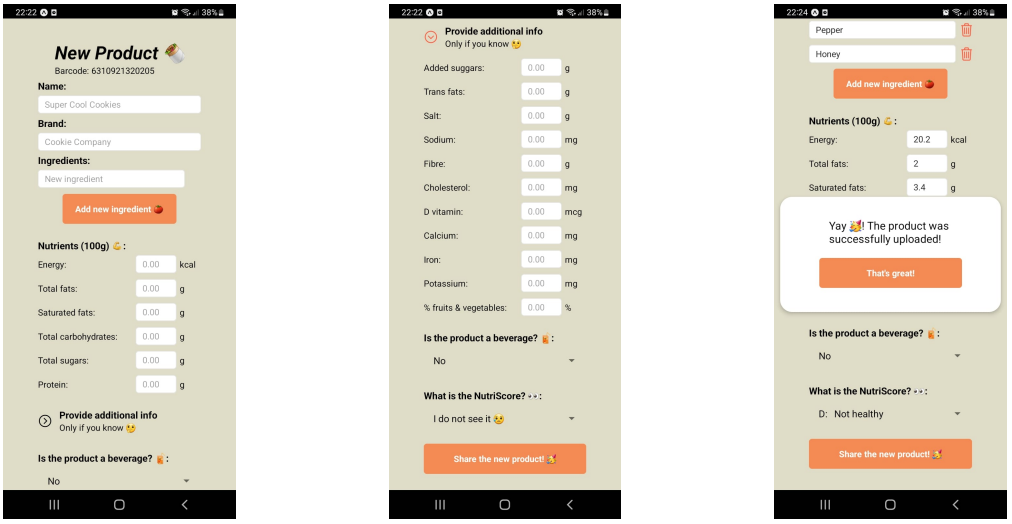


(b) Sin información disponible

Figura 5.14: Pantalla de localizaciones del producto

5.6.8. Añadir un nuevo producto

Para añadir un nuevo producto, el usuario debe escanear un código de barras que no esté registrado en el API. Una vez que lo encuentre, se le guiará hasta la pantalla de nuevo producto, donde deberá rellenar todos los campos para poder subir el producto satisfactoriamente. También se añaden campos opcionales de micro-nutrientes por si al usuario le interesara aportarlos.

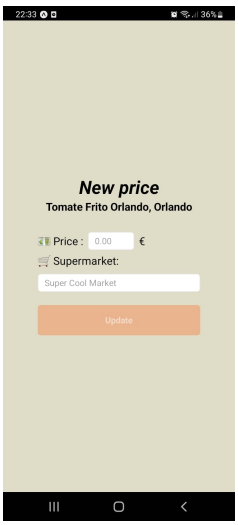


(a) Campos a ingresar (b) Nutrientes opcionales (c) Nuevo producto subido

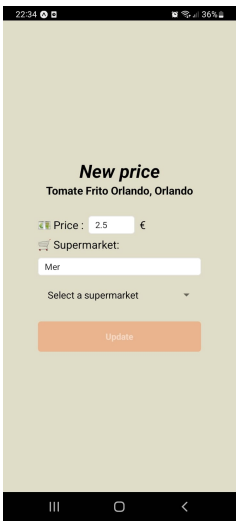
Figura 5.15: Pantalla de nuevo producto

5.6.9. Añadir un precio y localización a un producto

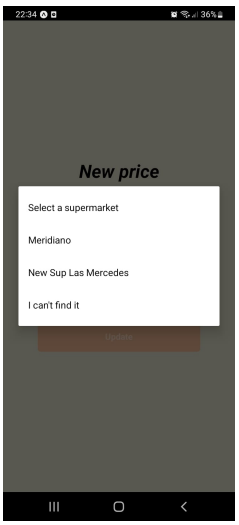
Como se observó en la pantalla de información de precios, el botón de *Update price* permitirá al usuario navegar hasta la pantalla de nuevo precio. Aquí, el usuario deberá indicar el precio al que encontró el producto, así como ingresar el nombre del supermercado. Un selector aparecerá e indicará los supermercados disponibles en base al nombre que el usuario ingrese. En caso de que no encuentre el supermercado, seleccionando la opción de *I can't find it*, aparecerán unos nuevos campos para indicar un nuevo supermercado, así como la latitud y longitud del mismo. Si los permisos de ubicación están concedidos, el usuario puede seleccionar la chincheta para rellenar estos campos automáticamente.



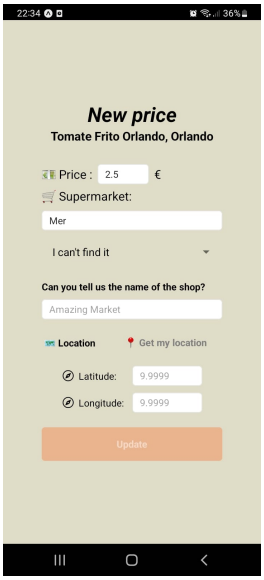
(a) Campos a ingresar



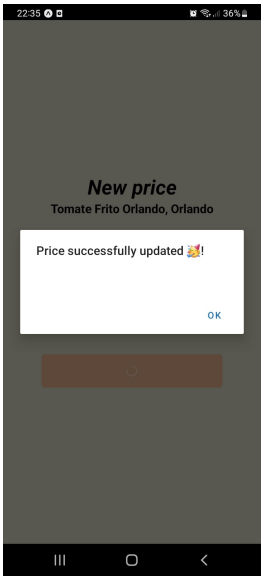
(b) Selector de tiendas



(c) Tiendas encontradas



(d) Tienda no encontrada



(e) Precio subido

Figura 5.16: Pantalla de nuevo precio

Esto hará que, en caso de que el supermercado seleccionado ya existiera, se actualice el precio en este. Sin embargo, si la tienda es una nueva indicada por el usuario, se creará añadiendo la nueva superficie en el mapa con el precio correspondiente.

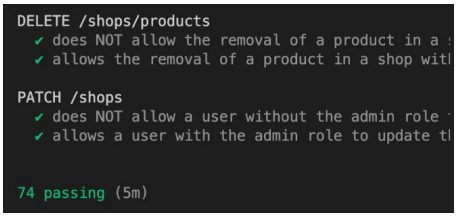
Capítulo 6

Pruebas y despliegue del API

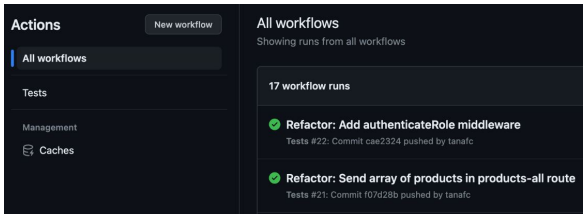
6.1. Pruebas en Mocha y SuperTest

Las pruebas en el desarrollo de software otorgan un gran beneficio al asegurar el correcto funcionamiento del mismo y su mantenibilidad. Para comprobar que la API cumple con todos los requisitos necesarios, y que su comportamiento es el esperado, se han diseñado pruebas para cada uno de los *endpoints* mediante las herramientas de `Mocha`[1] y `SuperTest`[2].

Mocha es un framework de pruebas de JavaScript para Node.js. Junto con la librería de SuperTest, permite realizar las pruebas sobre una aplicación de Express. Además, se ha implementado integración continua mediante *workflows* de GitHub Actions, donde por cada avance en la rama principal se ejecutan las pruebas para detectar los fallos lo antes posible.



(a) Tests en entorno local



(b) Workflow de GitHub Actions

Figura 6.1: Integración continua de la API

6.2. Despliegue en Railway

Existen múltiples servicios en la nube que permiten el despliegue de APIs, como Cyclic[3], Heroku[4], o Fly.io[5].

Para este proyecto, se ha escogido Railway[6], uno de los servicios más populares y utilizados. Su integración con GitHub permite un despliegue contínuo a cada actualización

[1] Mocha: `https://mochajs.org/`
[2] SyperTest: `https://www.npmjs.com/package/supertest`
[3] Cyclic.sh: `https://www.cyclic.sh/`
[4] Heroku: `https://www.heroku.com/`
[5] Fly.io: `https://fly.io/`
[6] Railway: `https://railway.app/`

de la rama indicada a la plataforma, con opciones de configuración como variables de entorno y métricas. El registro en el servicio se realiza mediante una cuenta de GitHub, dando los permisos al mismo para la lectura del repositorio, ya sea público o privado. Su plan gratuito ofrece 200 horas de uso.

Una vez desplegada la aplicación, se obtiene un dominio fijo al cual se pueden enviar las peticiones desde cualquier cliente.

6.3. Acceso a la aplicación

Para la utilización de la aplicación, será necesaria la ejecución del API REST en local, conectada a una instancia de MongoDB, y la instalación de las dependencias de Expo.

Desde el proyecto del API REST[7], se realizará la instalación de las dependencias mediante `npm install`. Se deberá crear alguna instancia de MongoDB y añadir su URL a las variables del fichero `.env`, como se muestra en la Sección 4.3. La ejecución del servidor con `npm start` hará que se empiecen a escuchar las peticiones en el puerto por defecto 8000, como en la Figura 6.2.

Una vez iniciado el backend, para el uso de la aplicación móvil se requerirá de la instalación previa de la herramienta de Expo CLI. Una vez instalada, desde el proyecto de la aplicación[8] se ejecuta el comando `npm install` y se inicia mediante `npm start`. La consola mostrará la ejecución de Metro Bundler (Figura 6.3), que proporcionará un código QR a escanear por las aplicación de Expo Go (Android) o la aplicación de cámara (iOS). Al escanearlo, se iniciará la *build* correspondiente, cargando la misma en el dispositivo para su prueba.

Algo a tener en consideración es que se debe editar la URL base del fichero `httpClient.ts` del frontend con la dirección del API en ejecución, que en un entorno local correspondería a una dirección IP privada, de la forma `http://192.168.0.0:8000`.

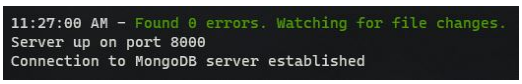


Figura 6.2: Ejecución del API

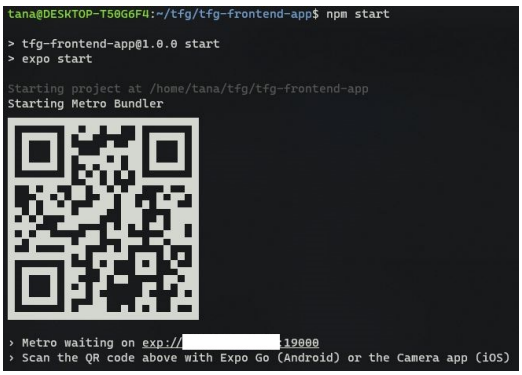


Figura 6.3: Ejecución de Metro Bundler de Expo

[7]Repositorio del proyecto del API: `https://github.com/tanafc/tfg-backend`
[8]Repositorio del proyecto de la aplicación móvil: `https://github.com/tanafc/tfg-frontend-app`

Capítulo 7

Conclusiones y líneas futuras

El presente trabajo ha consistido en el desarrollo de una aplicación full-stack, con la implementación de un API REST y una aplicación móvil, que permite compartir y obtener información nutricional sobre productos, datos sobre sus precios, y ubicaciones de tiendas donde pueden encontrarse.

Con la realización de este proyecto, se ha podido tener un mejor entendimiento del desarrollo de un API REST, con problemáticas avanzadas como la autenticación de usuarios o el almacenamiento seguro de contraseñas. A su vez, la creación de una aplicación móvil me ha permitido experimentar con frameworks de desarrollo que desconocía, como es React Native y Expo.

Mi poca experiencia en el desarrollo frontend, y aún menos en aplicaciones móviles, me ha llevado a realizar un trabajo de estudio para comprender muchos de los conceptos fundamentales, como componentes y estados en React, así como las características de React Native y Expo para la creación de una aplicación móvil. Uno de los mayores desafíos a los que me enfrenté fue el comienzo del desarrollo de la aplicación, pues no sabía como hacer las pruebas ni observar los cambios en las pantallas, así como el manejo de la navegación en las mismas. Me apoyé en gran medida al entorno de Expo, que me ofreció muchas facilidades, y que recomiendo a todo aquel que se inicie en el desarrollo móvil con React Native.

Con todo ello, se ha podido alcanzar exitosamente los objetivos propuestos en el proyecto, desde el diseño e implementación del API REST, la gestión de usuarios, el desarrollo de frontend con la creación de la aplicación, las pruebas y el despliegue del API.

De esta manera, existen mejoras que se pueden realizar tanto al API como a la aplicación móvil. Algunas de estas son:

- La documentación del API y sus *endpoints*.
- El añadido de imágenes para los productos.
- La realización de tests para la aplicación móvil.
- La localización de traducciones en la aplicación.
- La opción de filtrado por categorías de productos.
- La optimización de las gráficas del histórico de precios.
- El rediseño de pantallas para mejorar la experiencia del usuario.

Capítulo 8

Summary and Conclusions

The current work has consisted of the development of a full-stack application, with the implementation of a REST API and a mobile application, which allows sharing and obtaining nutritional information about products, data on their prices, and store locations where they can be found.

With the completion of this project, it has been possible to have a better understanding of the development of a REST API, including advanced issues such as user authentication or secure storage of passwords. At the same time, the creation of a mobile application has allowed me to experiment with development frameworks that I was not familiar with, such as React Native and Expo.

My little experience in frontend development, and even less in mobile applications, has led me to carry out a study work to understand many of the fundamental concepts, such as components and states in React, as well as the features of React Native and Expo for the creation of a mobile application. One of the biggest challenges I faced was the beginning of the development of the application, because I did not know how to test or observe the changes in the screens, as well as how to manage the navigation in them. I relied heavily on the Expo environment, which offered me a lot of facilities, and that I recommend to anyone starting out in mobile development with React Native.

With all this, it has been possible to successfully achieve the goals proposed in the project, from the design and implementation of the REST API, user management, frontend development with the creation of the application, and the testing and deployment of the API.

Improvements can be made to both the API and the mobile application. Some of these are:

- Documentation of the API and its endpoints.
- The addition of images for products.
- The implementation of test for the mobile app.
- Translation localization in the application.
- An option for filter products by categories.
- The optimization of price history graphs.
- The redesign of screens to improve user experience.

Capítulo 9

Presupuesto

A continuación, se expone el presupuesto para la ejecución del proyecto, asimilando una duración de 4 meses de desarrollo y 1 año de actividad de la API REST y la aplicación móvil.

Concepto	Presupuesto
Desarrollo de API y aplicación móvil	2.000 € x 4 meses
Clúster dedicado de MongoDB Atlas	51,18 € x 12 meses
Despliegue en Railway Pro	17,96 € x 12 meses
Disponibilidad de la app en App Store	88,89 € x 1 año
Disponibilidad de la app en Play Store	22,45 €
Total	**8.941,02 €**

Tabla 9.1: Presupuesto de ejecución el proyecto

El costo de desarrollo se estima a 2.000€ al mes. El clúster dedicado de la base de datos de MongoDB Atlas tiene un coste de 51.18€ al mes. A su vez, el despliegue del API en Railway mediante su plan Pro tiene un costo de 17.96€ al mes. Por último, para subir la aplicación a la App Store se debe pagar una tarifa de 88,89€ al año, mientras que para Play Store se paga 22,45€ una única vez. En total, el proyecto se presupuesta por 8.941,02€.

Bibliografía

[1] AESAN. “Información sobre el modelo Nutri-Score”. Accedido: 27-Jun-2023. [En línea]. Disponible: https://www.aesan.gob.es/AECOSAN/web/para_el_consumidor/seccion/informacion_Nutri_Score.htm.

[2] Calorie Mama. “A smart camera app that uses deep learning to track nutrition from food images”. Accedido: 5-Jun-2023. [En línea]. Disponible: https://www.caloriemama.ai/#CalorieMama.

[3] Comunidad de Madrid. “El etiquetado obligatorio de los alimentos”. Accedido: 27-Jun-2023. [En línea]. Disponible: https://www.comunidad.madrid/servicios/salud/etiquetado-obligatorio-alimentos#:~:text=La%20informaci%C3%B3n%20nutricional%20incluir%C3%A1%20el,incluir%20antes%20de%20esa%20fecha.

[4] El Coco. “Compra con cabeza, come sano”. Accedido: 5-Jun-2023. [En línea]. Disponible: https://elcoco.es/.

[5] Expo. “The world’s most loved ecosystem of tools that help you develop, review & deploy”. Accedido: 7-Jul-2023. [En línea]. Disponible: https://expo.dev/.

[6] Express. “Infraestructura web rápida, minimalista y flexible para Node.js”. Accedido: 12-Jul-2023. [En línea]. Disponible: https://expressjs.com/es/.

[7] Foodvisor. “Your personal nutrition guide”. Accedido: 5-Jun-2023. [En línea]. Disponible: https://www.foodvisor.io/en/.

[8] Google Play. “Tiendeo - Catálogos y Ofertas”. Accedido: 5-Jun-2023. [En línea]. Disponible: https://play.google.com/store/apps/details?id=com.geomobile.tiendeo&hl=es&gl=US&pli=1.

[9] IBM. “¿Qué es una API REST?”. Accedido: 20-Jun-2023. [En línea]. Disponible: https://www.ibm.com/es-es/topics/rest-apis.

[10] IETF. “JSON Web Token (JWT)”. Accedido: 13-Jul-2023. [En línea]. Disponible: https://datatracker.ietf.org/doc/html/rfc7519.

[11] MDN Web Docs. “Códigos de estado de respuesta HTTP”. Accedido: 7-Jul-2023. [En línea]. Disponible: https://developer.mozilla.org/es/docs/Web/HTTP/Status.

[12] MDN Web Docs. “Mobile First”. Accedido: 20-Jun-2023. [En línea]. Disponible: https://developer.mozilla.org/es/docs/Glossary/Mobile_First.

[13] MongoDB. “MongoDB: The Developer Data Platform”. Accedido: 12-Jul-2023. [En línea]. Disponible: https://www.mongodb.com/.

[14] Mongoose. “Elegant mongodb object modeling for node.js”. Accedido: 12-Jul-2023. [En línea]. Disponible: https://mongoosejs.com/.

[15] myHealthWatcher. “¡Di hola a la vida sana!”. Accedido: 5-Jun-2023. [En línea]. Disponible: https://myhealthwatcher.es/.

[16] MyRealFood. “La app para mejorar tu estilo de vida”. Accedido: 5-Jun-2023. [En línea]. Disponible: https://myrealfood.app/.

[17] Node.js. “Entorno de ejecución para JavaScript”. Accedido: 12-Jul-2023. [En línea]. Disponible: https://nodejs.org/es.

[18] OCU Market. “OCU Market: ahorra en tu compra del super y come más sano”. Accedido: 5-Jun-2023. [En línea]. Disponible: https://www.ocu.org/info/apps-ocu/ocu-market.

[19] React. “La biblioteca para interfaces de usuario web y nativas”. Accedido: 7-Jul-2023. [En línea]. Disponible: https://es.react.dev/.

[20] React Native. “Learn once, write anywhere.”. Accedido: 7-Jul-2023. [En línea]. Disponible: https://reactnative.dev/.

[21] Carmen Romero Ferreiro, David Lora Pablos, and Agustín Gómez de la Cámara. Two dimensions of nutritional value: Nutri-score and nova. *Nutrients*, 13(8), 2021.

[22] SoySuper. “Tu supermercado online. Fácil, fácil”. Accedido: 5-Jun-2023. [En línea]. Disponible: https://soysuper.com/.

[23] Vegan Pocket. “Vegan Pocket - Is It Vegan?”. Accedido: 5-Jun-2023. [En línea]. Disponible: https://veganpocketapp.com/.

[24] Wikipedia. “API”. Accedido: 21-Jun-2023. [En línea]. Disponible: https://es.wikipedia.org/wiki/API.

[25] Wikipedia. “Base de datos relacional”. Accedido: 21-Jun-2023. [En línea]. Disponible: https://es.wikipedia.org/wiki/Base_de_datos_relacional.

[26] Wikipedia. “CRUD”. Accedido: 21-Jun-2023. [En línea]. Disponible: https://es.wikipedia.org/wiki/CRUD.

[27] Wikipedia. “NoSQL”. Accedido: 22-Jun-2023. [En línea]. Disponible: https://es.wikipedia.org/wiki/NoSQL.

[28] Wikipedia. “Universal Product Code”. Accedido: 21-Jun-2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Universal_Product_Code.

[29] Wikipedia. “Índice de precios al consumidor”. Accedido: 21-Jun-2023. [En línea]. Disponible: https://es.wikipedia.org/wiki/%C3%8Dndice_de_precios_al_consumidor.

[30] Yuka. “La aplicación que escanea tus productos alimentarios”. Accedido: 5-Jun-2023. [En línea]. Disponible: https://yuka.io/es/.