



UNIÓN EUROPEA  
Fondo Social Europeo  
El FSE invierte en tu futuro

---

## **Proyecto - “WeWiza”: Servimos productos de mercados locales españoles para desarrolladores y usuarios**

---

**CICLO FORMATIVO DE GRADO SUPERIOR  
Desarrollo de Aplicaciones Multiplataforma**

**Curso 2022-24**

Autor/a/es:

**JiaCheng Zhang y Ángel Maroto Chivite**

Tutor/a:

**Alberto Madera Chamorro**

Departamento de Informática y Comunicaciones  
**I.E.S. Luis Vives**



# CONTENIDO

---

<b>1</b>	<b>RESUMEN.....</b>	<b>4</b>
1.1	JUSTIFICACIÓN DEL PROYECTO.....	4
<b>2</b>	<b>OBJETIVOS .....</b>	<b>5</b>
2.1	PLANIFICACIÓN GANTT.....	5
2.2	KANBAN + SCRUM .....	6
2.3	GITFLOW.....	7
<b>3</b>	<b>ANÁLISIS/ELECCIÓN TECNOLÓGICA .....</b>	<b>8</b>
3.1	BASE DE DATOS .....	8
3.1.1	<i>BASES DE DATOS NO ELEGIDAS.....</i>	8
3.1.2	<i>BASE DE DATOS ELEGIDA (MONGODB).....</i>	9
3.2	CREACIÓN API (BACK-END) .....	10
3.2.1	<i>APIs NO ELEGIDAS.....</i>	10
3.2.2	<i>API ELEGIDA (FASTAPI) .....</i>	11
3.3	CONSUMICIÓN API (FRONT-END) .....	11
3.3.1	<i>APIs NO ELEGIDAS.....</i>	11
3.3.2	<i>API ELEGIDA (RETROFIT) .....</i>	12
3.4	INTERFAZ ANDROID .....	12
3.4.1	<i>INTERFACES NO ELEGIDAS.....</i>	12
3.4.2	<i>INTERFAZ ELEGIDA (JETPACK COMPOSE) .....</i>	13
3.5	COMPETENCIA.....	14
3.6	REQUISITOS .....	15
3.6.1	<i>REQUISITOS FUNCIONALES .....</i>	15
3.6.2	<i>REQUISITOS No FUNCIONALES.....</i>	15
3.6.3	<i>REQUISITOS DE INFORMACIÓN.....</i>	15
<b>4</b>	<b>DISEÑO.....</b>	<b>16</b>
4.1	DIAGRAMA DE CLASES Y MODELO DE DATOS .....	16
4.1.1	<i>BACKEND.....</i>	16
4.1.2	<i>FRONTEND.....</i>	17
4.2	DIAGRAMA ENTIDAD-RELACIÓN (BASE DE DATOS) .....	18
4.2.1	<i>BACKEND MONGODB:.....</i>	18
4.2.2	<i>FRONTEND FIREBASE:.....</i>	18
4.3	MOCKUP .....	19
4.4	TIPOS DE FUENTES DE TEXTO .....	19
4.5	DIAGRAMA DE VISTAS .....	20
4.6	PALETA DE COLOR.....	20
<b>5</b>	<b>DESARROLLO BACK-END.....</b>	<b>21</b>
5.1	PLANTEAMIENTO .....	21
5.2	DATABASEMANAGER.....	22
5.3	REPOSITORIOS.....	23
5.3.1	<i>MARKETS.....</i>	23
5.3.2	<i>WEWIZA.....</i>	23
5.4	SERVICIOS.....	24
5.4.1	<i>PROCESAMIENTO DE PRODUCTOS .....</i>	24
5.4.2	<i>SCRAPPING Y REQUISITOS .....</i>	25
5.4.3	<i>CÁLCULO DIARIO .....</i>	25
5.5	ENDPOINTS.....	26
5.5.1	<i>ROOT.....</i>	26
5.5.2	<i>CATEGORÍAS Top .....</i>	26
5.5.3	<i>PRODUCTOS Top.....</i>	27
5.5.4	<i>CATEGORÍAS.....</i>	28
5.5.5	<i>PRODUCTOS POR CATEGORÍA.....</i>	28
5.5.6	<i>PRODUCTO POR UUID.....</i>	29



5.5.7	NÚMERO DE PRODUCTOS POR MERCADO .....	29
5.5.8	PRODUCTO CON HISTÓRICO POR UUID.....	30
5.5.9	PRODUCTOS POR MERCADO.....	31
5.5.10	PRODUCTOS POR RANGO Y MERCADO.....	31
5.5.11	PRODUCTO (ME GUSTA) .....	32
5.5.12	PRODUCTO (NO ME GUSTA).....	32
5.5.13	ESTADO DE LA REACCIÓN DE UN PRODUCTO POR USUARIO.....	32
5.5.14	INICIO MENSUAL.....	32
5.5.15	SUGERENCIAS POR PRODUCTO .....	33
5.6	RAILWAY ORIENTED PROGRAMMING.....	34
5.7	SEGURIDAD.....	35
5.7.1	SSL.....	35
5.8	DEPENDENCIAS.....	35
5.8.1	SERVIDOR.....	35
5.8.2	SCRAPPING.....	35
<b>6</b>	<b>DESARROLLO FRONT-END .....</b>	<b>36</b>
6.1	INTERFAZ.....	36
6.1.1	INTRODUCCIÓN .....	36
6.1.2	PANTALLAS Y COMPOSABLES.....	36
6.1.2.1	PANTALLA DE BIENVENIDA.....	36
6.1.2.2	PANTALLA DE INICIO DE SESIÓN .....	37
6.1.2.3	PANTALLA DE REGISTRO .....	37
6.1.2.4	PANTALLA DE DESTACADOS .....	38
6.1.2.5	PANTALLA DE BÚSQUEDA.....	38
6.1.2.6	PANTALLA DE LISTAS DE USUARIO .....	39
6.1.2.7	PANTALLA DE PERFIL .....	39
6.1.2.8	PANTALLA DE AJUSTES .....	40
6.1.2.9	PANTALLA DE PRODUCTOS .....	40
6.1.2.10	PANTALLA DE DETALLES DE PRODUCTO.....	41
6.1.2.11	PANTALLA DE LISTA DE USUARIO .....	41
6.1.2.12	PANTALLA DE SUGERENCIAS.....	42
6.1.2.13	PANTALLA DE SOPORTE .....	42
6.1.2.14	PANTALLA DE MANTENIMIENTO .....	43
6.2	FIREBASE .....	43
6.2.1	USUARIOS.....	43
6.3	INVERSIÓN DE CONTROL/INYECTOR DE DEPENDENCIAS .....	44
6.4	SEGURIDAD.....	44
6.4.1	FIREBASE AUTH OAuth 2.0.....	44
<b>7</b>	<b>TESTS .....</b>	<b>45</b>
7.1	RUTAS (END-POINTS) .....	45
<b>8</b>	<b>IMPLANTACIÓN/DESPLIEGUE.....</b>	<b>45</b>
8.1	ALOJAMIENTO SERVIDOR.....	45
8.2	DOCKER .....	46
8.2.1	DOCKERFILE BACKEND.....	46
8.2.2	DOKER-COMPOSE .....	47
8.3	ANDROID.....	49
8.3.1	LIBRERÍAS. ....	49
<b>9</b>	<b>CONCLUSIONES Y TRABAJO FUTURO .....</b>	<b>50</b>
<b>10</b>	<b>CÓDIGO FUENTE DEL PROYECTO.....</b>	<b>50</b>
<b>11</b>	<b>BIBLIOGRAFÍA.....</b>	<b>51</b>

# 1 RESUMEN

## 1.1 JUSTIFICACIÓN DEL PROYECTO

La elección de este proyecto se basa en la necesidad creciente de los consumidores de optimizar sus gastos en la compra de alimentos, especialmente en el contexto económico donde la inflación y los cambios en los precios son cada vez más frecuentes.

Adquirimos el nombre de “Wewiza” siendo un acrónimo del inglés “We are wizards”, es decir “Somos magos”, ya que el servicio que proponemos es crear una entidad o comunidad donde intentamos a llegar a todo público desde desarrolladores a usuarios, respectivamente dando servicio mediante un servidor API Rest o desde una aplicación móvil.

La aplicación que se propone para el usuario es un ejemplo de cómo utilizar nuestro servicio API Rest donde se busca ofrecer una solución práctica y conveniente para ayudar a los usuarios a tomar decisiones informadas sobre dónde comprar sus productos alimentarios al mejor precio.



La aplicación se focaliza en comparar precios entre distintos supermercados y mercados alimentarios en España. Además, incluye una dinámica de gamificación que invita a los usuarios a verificar la correspondencia entre los precios sugeridos en la aplicación y los precios reales.

Los usuarios tienen la capacidad de buscar productos específicos y recibir recomendaciones sobre dónde pueden encontrar el mejor precio para dichos productos. Asimismo, la aplicación ofrece funcionalidades adicionales, como la creación de listas de compras, para las cuales se brindan sugerencias de soluciones optimizadas.

Por último, la aplicación ofrece una experiencia centrada en la eficiencia y la economía para el usuario, ayudándoles a tomar decisiones informadas sobre sus compras de manera ágil y confiable.



## 2 OBJETIVOS

- Adquirir conocimientos en el desarrollo de aplicaciones móviles, gestión de bases de datos y despliegue mediante contenedores y alojamiento de servidores.
- Proporcionar a los usuarios una herramienta eficaz para optimizar sus gastos en la compra de alimentos, permitiéndoles tomar decisiones informadas sobre dónde adquirir los productos al mejor precio.
- Desarrollar una aplicación intuitiva y fácil de usar que permita a los usuarios buscar y comparar precios de productos alimentarios en diferentes supermercados y mercados en España.
- Implementar una funcionalidad de gamificación que motive a los usuarios a participar en la verificación de precios de productos, generando un “feedback” útil para mejorar la precisión y confiabilidad.
- Facilitar la creación de listas de compras personalizadas, ofreciendo sugerencias y alternativas basadas en los precios y ubicación del usuario.

### 2.1 PLANIFICACIÓN GANTT

Mediante 5 “Sprints” semanales, hemos trabajado con una metodología ágil.

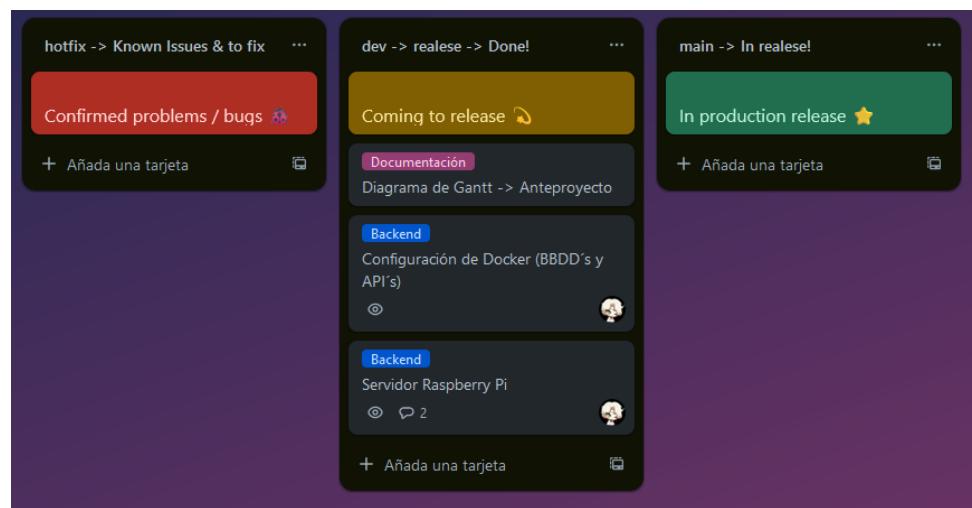
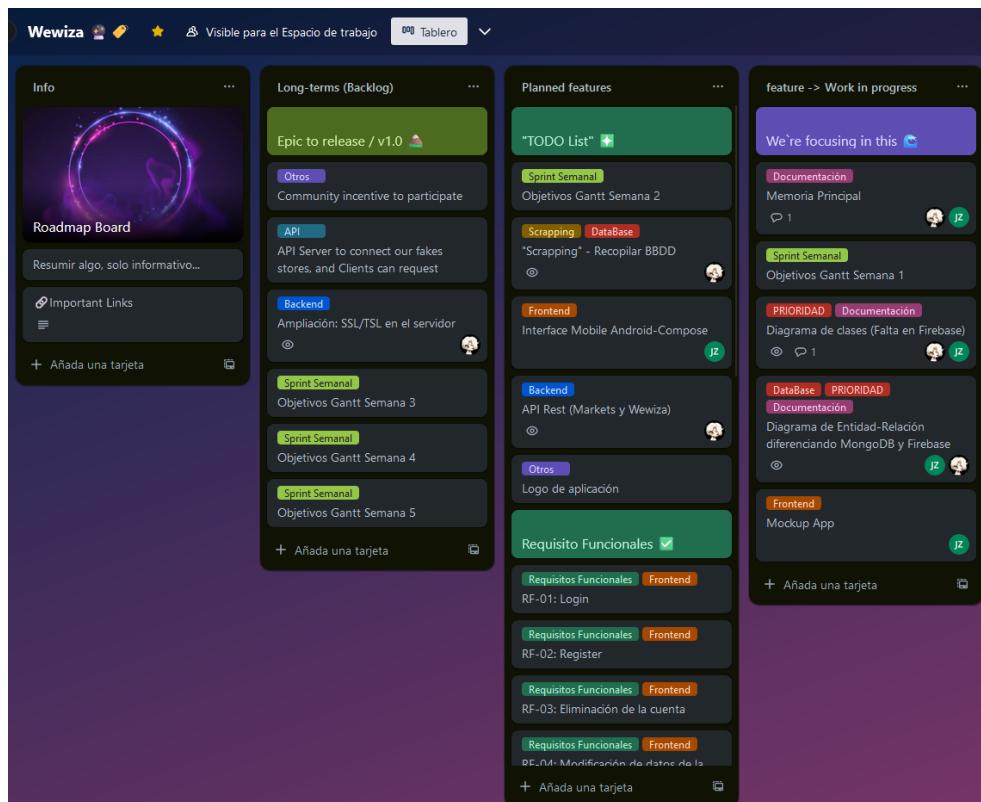
Al acabar un “Sprint” semanal, realizamos una retrospectiva de dicho “Sprint”, para verificar nuestros avances o dificultades, mediante objetivos marcados en “Trello”.

Nombre de Tarea	Duración	1 Semana					2 Semana					3 Semana					4 Semana					5 Semana								
		L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	
DOCUMENTACIÓN																														
1 Diagrama de clases	2 horas																													
2 Documentación técnica	6 horas																													
3 Presentación diapositiva	2 horas																													
	Total: 10h.																													
BACK-END																														
4 Configuración del servidor	15 horas																													
5 Configuración de BBDD	10 horas																													
6 Recopilación y almacenamiento de datos en la BBDD	25 horas																													
7 Crear servicio API REST	45 horas																													
	Total: 95h.																													
FRONT-END																														
8 Mockup interfaz	25 horas																													
9 Configuración Firebase	10 horas																													
10 Desarrollo de interfaz gráfica	50 horas																													
11 Consumir servicio API REST	10 horas																													
	Total: 95h.																													
	Total: 200h.																													

## 2.2 KANBAN + SCRUM

Mediante “Trello” hemos aplicado brevemente el método “Scrum” como el “Back-log” y “Sprints” semanales:

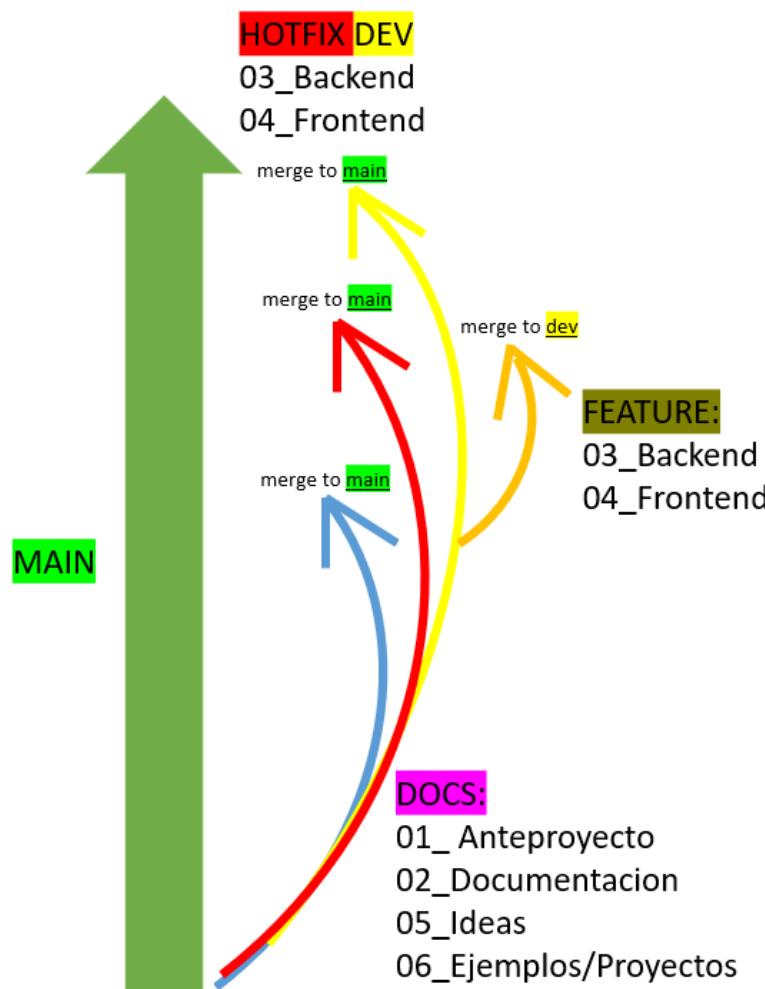
- “Back-log”: donde explicamos de forma muy general la tarea-
- “TODO list”: detallamos en profundidad la tarea del “Back-log”.
- “In progress”: el trabajo que estamos realizando.
- “Coming to release”: trabajado realizado, en espera de ser verificado por el equipo.
- “Done to release”: trabajo verificado y dispuesto para lanzar a producción.



## 2.3 GITFLOW

Trabajar con “Git” para mantener un control de versiones a lo largo de los “Sprints” es totalmente compatible con la metodología “Scrum” ya que podemos disponer de varias ramas diferenciadas por la fase del ciclo de vida del desarrollo en el que nos encontramos.

- “Feature”: características independientes.
- “Dev”: trabajo realizado realzado con el sumatorio de “features”, en espera de ser verificado por el equipo.
- “Docs”: documentación del proyecto
- “Hotfix”: errores o “bugs” puntuales que requieren de un arreglo inmediato.
- “Main”: trabajo verificado y dispuesto para lanzar a producción.



## 3 ANÁLISIS/ELECCIÓN TECNOLÓGICA

### 3.1 BASE DE DATOS

#### 3.1.1 BASES DE DATOS NO ELEGIDAS



##### ❖ Ventajas:

- ✓ Potente soporte para características avanzadas como transacciones ACID, integridad referencial y procedimientos almacenados.
- ✓ Fuerte enfoque en la integridad de los datos y la seguridad.
- ✓ Escalabilidad y rendimiento sólidos para aplicaciones de alto volumen y alta concurrencia.

##### ❖ Desventajas:

- Curva de aprendizaje más pronunciada.
- Menor popularidad y adopción.



##### ❖ Ventajas:

- ✓ Amplia adopción y comunidad activa.
- ✓ Rendimiento rápido para aplicaciones de tamaño mediano.
- ✓ Facilidad de uso y amplia compatibilidad con herramientas y lenguajes de programación.

##### ❖ Desventajas:

- Limitaciones en escalabilidad y rendimiento para grandes volúmenes de datos.
- Menor soporte para características avanzadas.



❖ Ventajas:

- ✓ Compatibilidad con MySQL y mejora de características adicionales.
- ✓ Mejora continua y desarrollo comunitario activo.
- ✓ Escalabilidad y rendimiento debido al almacenamiento en columnas mejorado.

❖ Desventajas:

- Dificultades en migraciones y portabilidad.
- Documentación menos extensa.

### 3.1.2 BASE DE DATOS ELEGIDA (MONGODB)



❖ Ventajas:

- ✓ Modelo de datos flexible y escalabilidad horizontal.
- ✓ Alto rendimiento para operaciones de lectura y escritura en entornos de alta carga.
- ✓ Buena integración con aplicaciones modernas y “frameworks” de desarrollo.

❖ Desventajas:

- Falta de soporte transaccional ACID completo en comparación con sistemas de bases de datos relacionales.
- Menor rendimiento para operaciones complejas de análisis.

## 3.2 CREACIÓN API (BACK-END)

### 3.2.1 APIs NO ELEGIDAS



#### ❖ Ventajas:

- ✓ Framework completo y con todas las funciones, que incluye características como autenticación, ORM, administración de usuarios y más.
- ✓ Buena documentación y comunidad activa.

#### ❖ Desventajas:

- Mayor curva de aprendizaje inicial.
- Menos flexibilidad para proyectos pequeños.



#### ❖ Ventajas:

- ✓ Ligero y minimalista.
- ✓ Amplia gama de extensiones disponibles para añadir funcionalidades específicas según las necesidades del proyecto.
- ✓ Facilidad de aprendizaje y uso.

#### ❖ Desventajas:

- Requiere una mayor responsabilidad por parte del desarrollador en la toma de decisiones de diseño y elección de herramientas.

### 3.2.2 API ELEGIDA (FASTAPI)



❖ Ventajas:

- ✓ Alto rendimiento y eficiencia gracias a su integración con Pydantic, lo que permite la generación de código altamente optimizado.
- ✓ Fácil creación de API RESTful con una sintaxis declarativa y tipado estático.
- ✓ Integración sencilla con herramientas de documentación automática como Swagger y ReDoc.

❖ Desventajas:

- Relativamente nueva conlleva una menor cantidad de recursos y bibliotecas disponibles.
- Curva de aprendizaje moderada debido a su enfoque en la programación asíncrona y en el uso de Pydantic para validación de datos.

### 3.3 CONSUMICIÓN API (FRONT-END)

#### 3.3.1 APIs NO ELEGIDAS



❖ Ventajas:

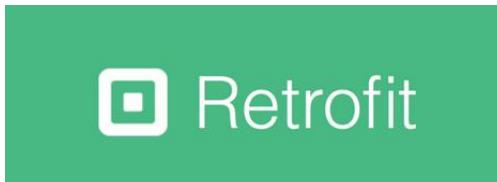
- ✓ Está integrado con las corutinas de Kotlin, lo que permite realizar solicitudes de red de forma asíncrona y sin bloqueo de forma nativa.
- ✓ Es muy flexible y modular.
- ✓ Ofrece una API moderna y orientada a funciones que se adapta bien al estilo de programación funcional de Kotlin.

❖ Desventajas:

- Curva de aprendizaje al principio para comprender y utilizar eficazmente el enfoque asíncrono.
- Menos recursos de la comunidad y documentación.



### 3.3.2 API ELEGIDA (RETROFIT)



#### ❖ Ventajas:

- ✓ Fácil de usar, interfaz simple y declarativa.
- ✓ Se integra fácilmente con bibliotecas de serialización JSON.
- ✓ Comunidad de usuarios y una documentación extensa.

#### ❖ Desventajas:

- Es sincrónico por defecto, lo que significa que las solicitudes se realizan en el hilo principal de la aplicación a menos que se especifique lo contrario, por lo que manejar operaciones asincrónicas puede requerir el uso de bibliotecas adicionales o patrones de programación más complejos

## 3.4 INTERFAZ ANDROID

### 3.4.1 INTERFACES NO ELEGIDAS



#### ❖ Ventajas:

- ✓ Desarrollo multiplataforma.
- ✓ Utiliza el motor de renderizado Skia para generar interfaces de usuario altamente fluidas y responsivas.
- ✓ Reutilización de componentes atractivos.
- ✓ Función de Hot Reload de Flutter permite realizar cambios en tiempo real en la aplicación durante el desarrollo, lo que acelera el proceso de iteración y depuración.

#### ❖ Desventajas:

- Las aplicaciones tienden a ser más grandes en tamaño en comparación con las aplicaciones nativas.
- Menor cantidad de recursos y bibliotecas de terceros.



❖ Ventajas:

- ✓ Gran cantidad de recursos y documentación disponibles.
- ✓ Permite separar claramente el diseño de la interfaz de usuario de la lógica de la aplicación.
- ✓ Compatible con versiones anteriores de Android.

❖ Desventajas:

- Puede volverse verboso y difícil de mantener en aplicaciones con interfaces de usuario complejas.
- Limitación para previsualizar o emular las interfaces.
- Limitación para desarrollar interfaces más modernas y dinámicas

### 3.4.2 INTERFAZ ELEGIDA (JETPACK COMPOSE)



❖ Ventajas:

- ✓ Se integra perfectamente con el ecosistema de Android y las herramientas de desarrollo de Google.
- ✓ Utiliza un enfoque declarativo para la creación de interfaces de usuario.
- ✓ Incluye una función de previsualización en tiempo real.

❖ Desventajas:

- Relativamente nueva y puede requerir actualizaciones significativas en las aplicaciones existentes para adoptarla.
- Aunque simplifica la creación de interfaces de usuario en muchos aspectos, puede requerir tiempo para familiarizarse con sus conceptos y patrones de diseño.



### 3.5 COMPETENCIA

**PriceSpy**, **Idealo** y **Chollometro** abarcan una gama amplia de productos, principalmente productos tecnológicos, ropa o incluso alimentación muy específica (alcohol, café, chocolates), pero no se especializan en ningún sector.

Al no encontrar una herramienta realmente accesible y funcional que asista al usuario en su día a día en las compras alimentarias del hogar, "WeWiza" entra cubriendo esa necesidad.



Daily Deals

Filter Popularity Search deals

116 results

Category	Product Name	Brand	Rating	Price
Portable Speakers	Ultimate Ears Hyperboom Bluetooth Speaker	Harvey Norman	★★★★★ (7)	\$388.00
Portable Speakers	Sonos Move WiFi Bluetooth Speaker	JB Hi-Fi	★★★★★ (16)	\$545.00
Routers	TP-Link Deco Voice X20 Whole-Home Mesh WiFi System (2-pack)	Computer Lounge	★★★★★	\$299.00
Home Security Cameras	Google Nest Cam Outdoor or Indoor Battery	PB Tech	★★★★★ (6)	\$239.00



Apple AirPods Pro 2

desde 229,00 €

Comparar 32 ofertas

9,0/10 Nota media

32 ofertas

Ver todo

229,00 € IVA incl.

Envío inmediato Envío gratuito



Chollometro

Destacados Popular Nuevo Los más populares

Consola Xbox Series S 80,99 € | Microsoft

Gafas Northweek gratis pagando solo gastos de envío GRATIS | Northweek

Apple iPhone 13 versión 128GB solo 579€ 579 € | MediaMarkt

Cupón 15€ en compras de más de 21 m



## 3.6 REQUISITOS

### 3.6.1 REQUISITOS FUNCIONALES

- RF-01: Acceso a la aplicación “Log in”.
- RF-02: Registro a la aplicación.
- RF-03: Eliminación de la cuenta.
- RF-04: Modificación de datos de la cuenta.
- RF-05: Buscador de productos.
- RF-06: Agregar productos a lista de la compra del usuario.
- RF-07: Eliminar productos de la lista de la compra del usuario.
- RF-08: Sistema de niveles de usuario en función de la participación con la aplicación.
- RF-09: Filtrar los productos en función de categorías, tiendas y precio.
- RF-10: Detalles de un producto con productos relacionados y si hay un producto similar en otra tienda a un precio inferior.
- RF-11: Participación en confirmar o negar que el producto corresponde con el real.
- RF-12: Sugerir varias rutas donde realizar la compra en función de los máximos mercados a los que visitar y los productos que tiene el usuario en su lista de la compra.
- RF-13: Recomendar productos en la interfaz principal.
- RF-14: Posibilidad de contactar con los desarrolladores.
- RF-15: Gráfica de históricos mensuales en detalles de producto

### 3.6.2 REQUISITOS NO FUNCIONALES

- RNF-01: Aplicación de móvil Android.
- RNF-02: “Back-end” en Python con persistencia de datos mediante NoSQL (MongoDB).
- RNF-03: “Front-end” en Jetpack-Compose con persistencia de datos mediante NoSQL (Firebase).
- RNF-04: Despliegue en contendores Docker.
- RNF-05: Alojamiento de servidor.

### 3.6.3 REQUISITOS DE INFORMACIÓN

- Dispondremos de 4 entidades principales:
  - RI-01: Productos: serán proporcionados por los distintos comercios.
  - RI-02: Categorías: serán tratadas en función de cada estructura en los distintos mercados.
  - RI-03: Usuarios: dispondrán de una lista de la compra.
  - RI-04: Lista de compra: cada usuario podrá tener varias listas de compra personalizadas.



## 4 DISEÑO

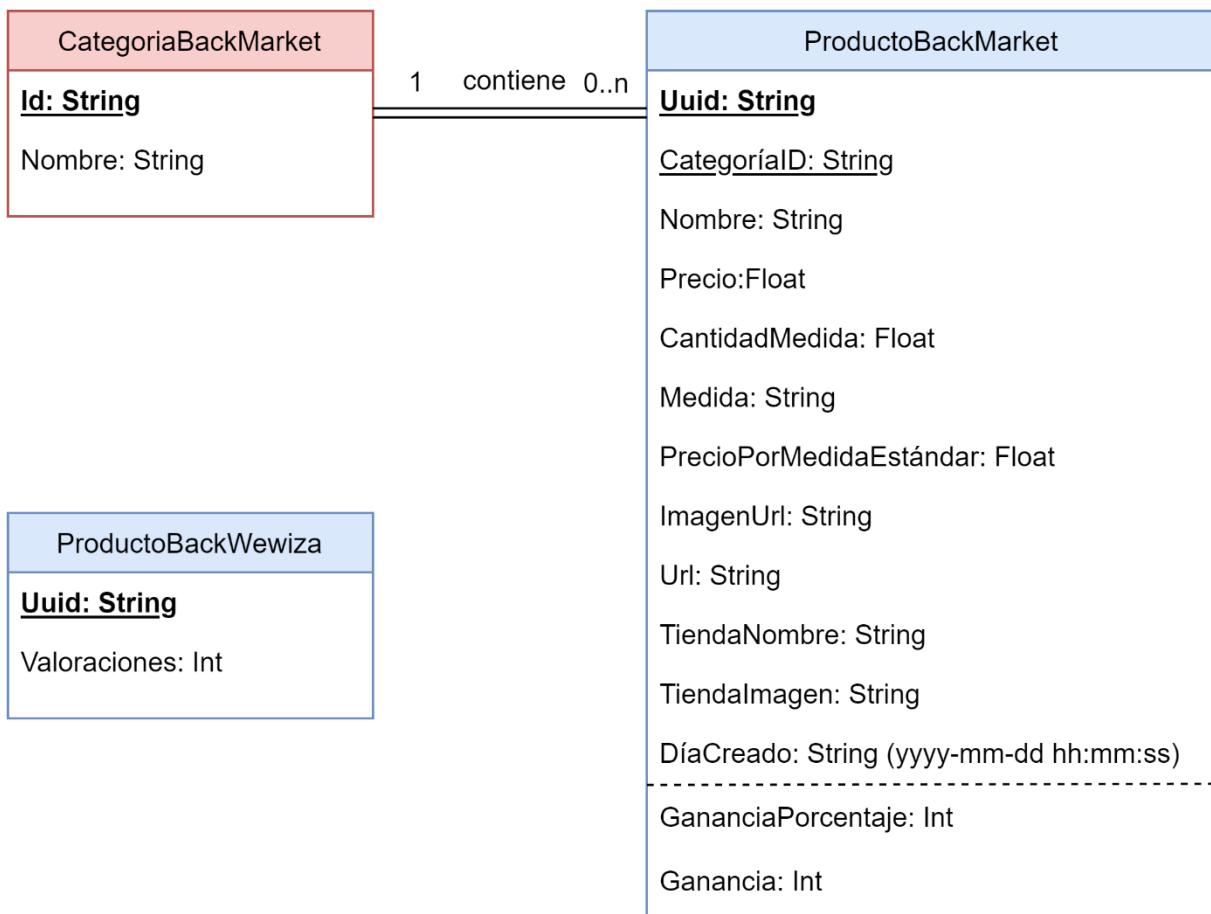
### 4.1 DIAGRAMA DE CLASES Y MODELO DE DATOS

#### 4.1.1 BACKEND

- **CategoríaBackMarket** y **ProductoBackMarket**: corresponden a las API-market, es decir el “backend” de cada mercado.

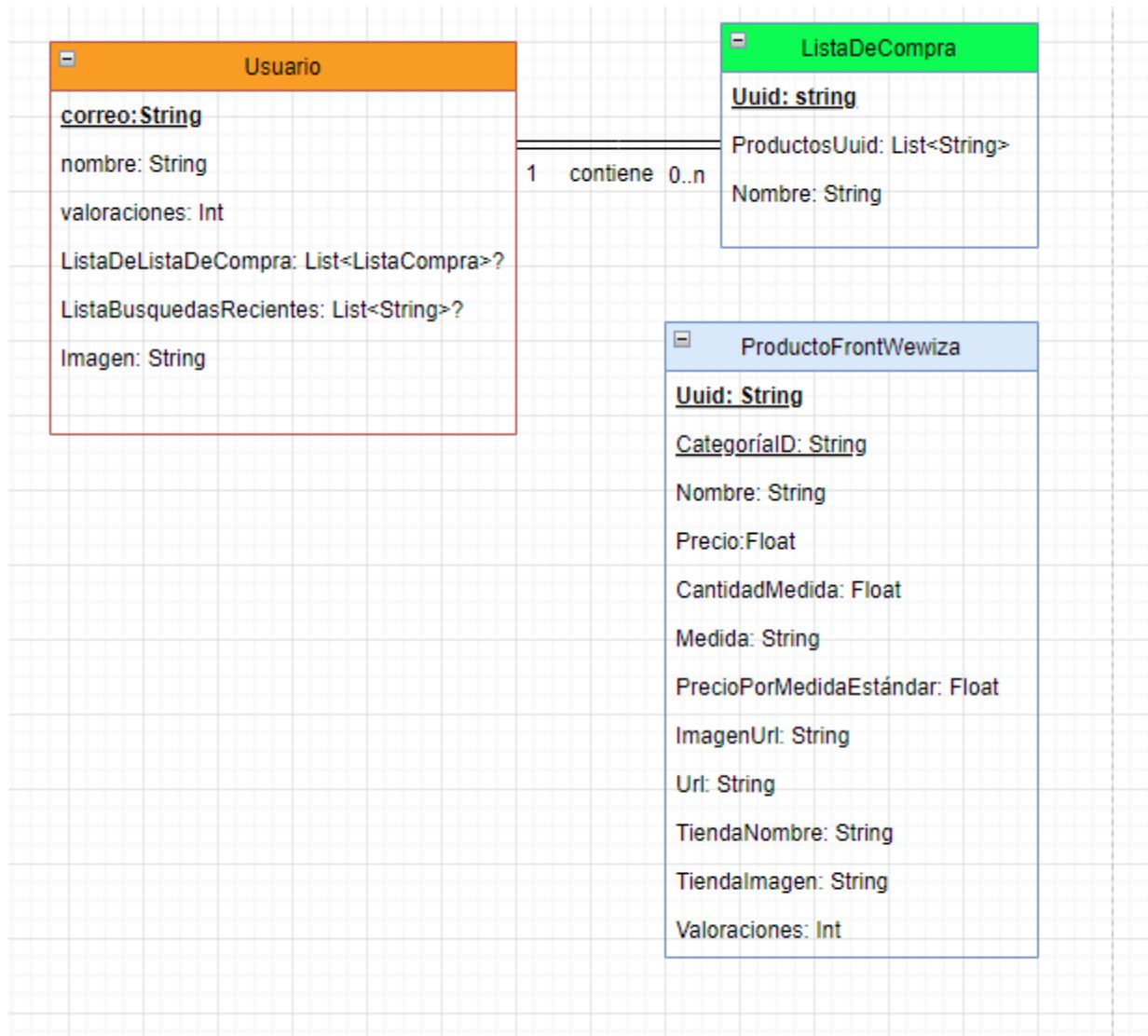
En nuestra implementación hemos definido que un producto solamente pueda tener referencia a una categoría ya que no requerimos de más, aunque una ampliación considerable sería ampliar el catálogo de categorías así dando lugar a que los productos puedan hacer referencia a una lista de categorías.

- **ProductoBackWewiza**: corresponde a la clase utilizada en nuestra API-wewiza, podemos reducir su complejidad ya que delegamos el almacenaje central de datos a los mercados.
  - **GananciaPorcentaje** y **Ganancia** son campos que aparecen en endpoints particulares.



## 4.1.2 FRONTEND

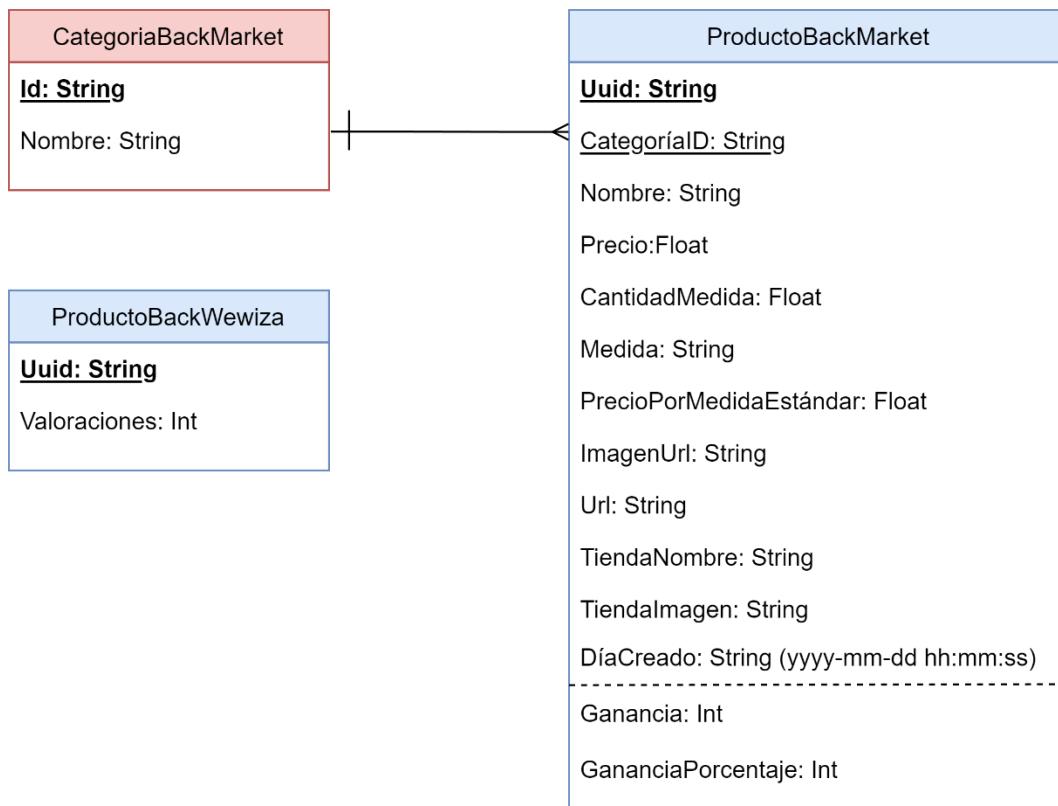
- **Usuario**: siempre se almacenará un usuario independientemente del tipo de log-in.
- **ListaDeCompra**: un usuario podrá disponer de varias listas de compras donde se almacenará el UUID del producto, para posteriormente hacer la llamada a la API.
- **ProductoFrontWewiza**: al pasar por la API-market y API-wewiza recogemos todos los datos para poderlos presentar.
  - **GananciaPorcentaje** y **Ganancia** son campos que aparecen en endpoints particulares.



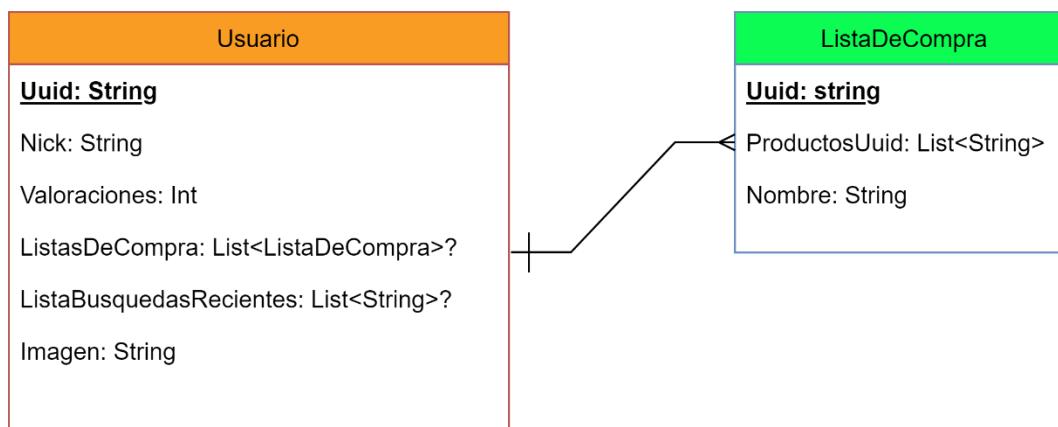
## 4.2 DIAGRAMA ENTIDAD-RELACIÓN (BASE DE DATOS)

### 4.2.1 BACKEND MONGODB:

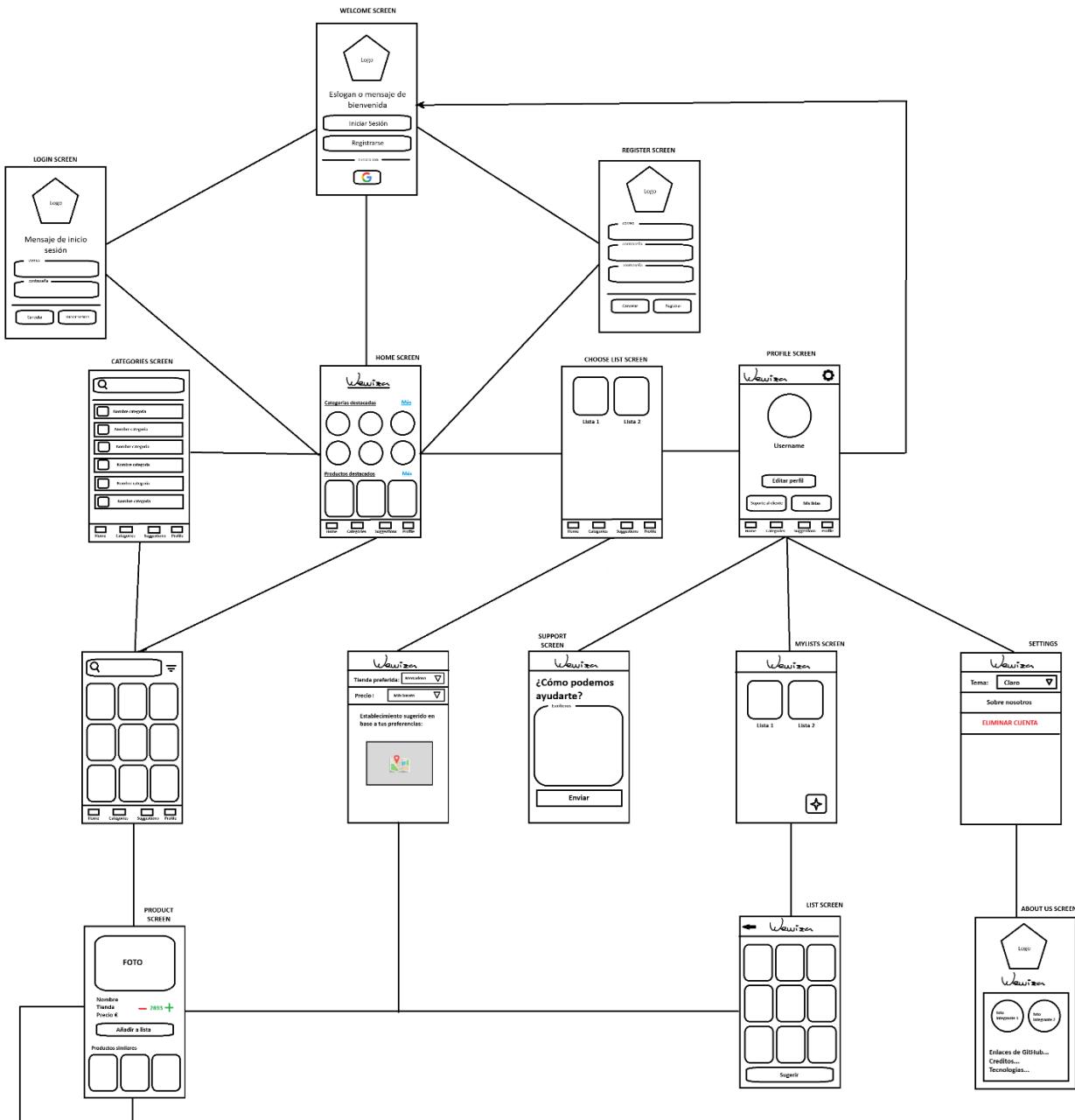
- **GananciaPorcentaje** y **Ganancia** son campos que aparecen en endpoints particulares.



### 4.2.2 FRONTEND FIREBASE:



## 4.3 MOCKUP

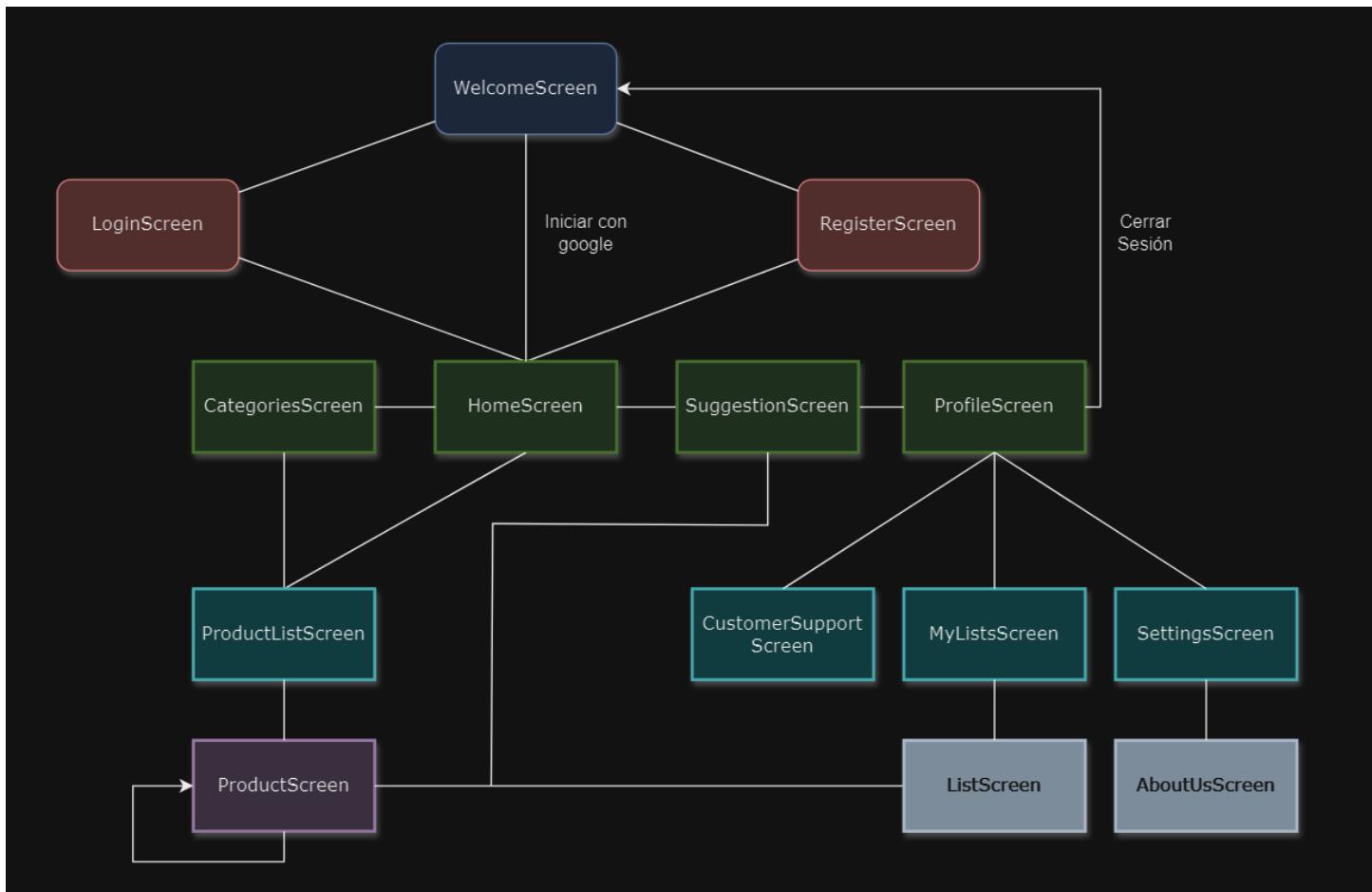


## 4.4 TIPOS DE FUENTES DE TEXTO

- Títulos de grado 1 (H1): Mermaid1001
  - Títulos de grado 2 (H2): TT Neue Bold
  - Texto estándar: TT Firs Neue Regular

## 4.5 DIAGRAMA DE VISTAS

Flujo de navegación que podrá realizar el usuario en la aplicación móvil:



## 4.6 PALETA DE COLOR



#470459

RGB 71, 4, 89

#8538A6

RGB 133, 56, 166

#7386BF

RGB 115, 134, 191



#F2CB57

RGB 242, 203, 87

#F2F2F2

RGB 242, 242, 242



## 5 DESARROLLO BACK-END

### 5.1 PLANTEAMIENTO

Se ha llevado a cabo simulando la situación en la que cada mercado nos facilita su correspondiente API, al no disponer de esa facilidad nos hemos visto envueltos en la premisa de recolectar los datos recogiéndolos de las páginas web públicas de cada mercado con una finalidad meramente didáctica.

Una vez recolectado los datos planteamos que cada mercado tendrá su backend personal con sus características específicas por lo que hemos construido diversas API Rest monolíticas de cada uno.

Cada mercado presentaba una organización de productos mediante categorías con un criterio muy específico por lo que se ha tenido que realizar un análisis en profundidad de cada mercado y llevar a cabo un tratamiento de datos agrupando en categorías que hemos considera las más adecuadas y de búsqueda frecuente del usuario:

ID	Nombre	
aceite_especias_y_salsas	Aceite, especias y salsas	congelados
agua_y_refrescos	Agua y refrescos	conservas_caldos_y_cremas
aperitivos	Aperitivos	frutas
arroz_legumbres_y_pasta	Arroz, legumbres y pasta	huevos_leche_batidos_y_mantequilla
azucar_caramelos_y_chocolate	Azúcar, caramelos y chocolate	marisco_y_pescado
bebé	Bebé	mascotas
bodega	Bodega	panaderia_y_pasteleria
cacao_cafe_e_infusiones	Cacao, café e infusiones	pizzas_y_platos_preparados
carne	Carne	postres_y_yogures
cereales_y_galletas	Cereales y galletas	verduras
charcuteria_y_quesos	Charcutería y quesos	zumos
congelados	Congelados	

Las categorías no agrupadas serían las siguientes: [cuidado de cabello, facial y corporal / fitoterapia y parafarmacia / limpieza y hogar / maquillaje](#).

Finalmente se terminaría construyendo la API Rest monolítica de Wewiza donde actuaría como nexo para encontrar, filtrar y proporcionar el servicio de sugerencias de productos de los diversos mercados al usuario.



## 5.2 DATABASEMANAGER

Disponemos de la clase “DatabaseManager” con la responsabilidad de:

- Crear bases de datos en MongoDB.
- Crear colecciones en MongoDB con “Schemas” de validación.
- Conexión a las bases de datos de MongoDB.
- Cierre de conexión a las bases de datos de MongoDB.

```
class DatabaseManager:  
    def __init__(self, connection_host, database_name) → None:  
        self.connection_host = connection_host  
        self.database_name = database_name  
        self.client = None  
        self.database = None  
  
    def __create_database(self) → None:  
        if self.database_name not in self.client.list_database_names():  
            self.client[self.database_name]  
  
    def create_collection_with_validation(self, collection_name, new_validator=None) → None:  
        """  
        Only is going to be executed in the repositories when instantiating  
        """  
        if collection_name not in self.database.list_collection_names():  
            self.database.create_collection(collection_name, validator=new_validator)  
  
    def connect_database(self) → Database:  
        self.client = MongoClient(self.connection_host)  
        self.database = self.client[self.database_name]  
        self.__create_database()  
        return self.database  
  
    def close_database(self) → None:  
        if self.client:  
            self.client.close()
```

Los “Schemas” son validadores para aplicar rigidez y consistencia a las colecciones de las bases de datos de MongoDB así teniendo un mayor control sobre que documentos son aceptados:

```
product_schema = {  
    "$jsonSchema": {  
        "bsonType": "object",  
        "title": "Product Object Validation",  
        "required": [  
            "uuid",  
            "category_id",  
            "name",  
            "price",  
            "quantity_measure",  
            "measure",  
            "price_by_standard_measure",  
            "image_url",  
            "store_name",  
            "store_image_url",  
        ],  
        "properties": {  
            "uuid": {  
                "bsonType": "string",  
                "description": "must be a string and is required",  
            },
```

## 5.3 REPOSITORIOS

Se delega al “Repository” las consultas y modificaciones a la base de datos mediante un CRUD para manipular correctamente los productos:

### 5.3.1 MARKETS

```
class ProductRepository:      You, 2 months ago • [Update] Folder restructuring
    def __init__(self, db_manager: DatabaseManager, collection_name) → None:
        self.db_manager = db_manager
        self.collection_name = collection_name
        self.__setup_collection_validation()

    def __setup_collection_validation(self) → Any | Result:...
        def insert_product(self, product_data) → Result:... → CREATE
        def get_size(self) → Result:...
        def get_all_products_by_query(self, query) → Result:...
        def get_all_products(self) → Result:...
        def get_all_products_by_market(self, market_name) → Result:...
        def get_all_products_by_category_id(self, category_id) → Result:...
        def get_products_by_range(self, init_num, end_num) → Result:...
        def get_product_by_uuid(self, uuid) → Result:...
        def get_products_by_name(self, product_name) → Result:...
        def get_products_by_date(self, date) → Result:...

        def delete_product(self, query) → Result:... → DELETE
        def update_product(self, query, new_data) → Result:... → UPDATE
```

The diagram illustrates the ProductRepository class structure. It highlights several methods with colored boxes and arrows indicating their function:

- CREATE:** `insert_product(self, product_data)` (blue box, blue arrow)
- READ:** `get_size(self)`, `get_all_products_by_query(self, query)`, `get_all_products(self)`, `get_all_products_by_market(self, market_name)`, `get_all_products_by_category_id(self, category_id)`, `get_products_by_range(self, init_num, end_num)`, `get_product_by_uuid(self, uuid)`, `get_products_by_name(self, product_name)`, `get_products_by_date(self, date)` (green box, green arrow)
- UPDATE:** `update_product(self, query, new_data)` (yellow box, yellow arrow)
- DELETE:** `delete_product(self, query)` (red box, red arrow)

### 5.3.2 WEWIZA

```
class ProductLikesRepository:
    def __init__(self, db_manager: DatabaseManager, collection_name) → None:...

    def __setup_collection_validation(self) → Any | Result:...
        def insert_products_json(self, products_data) → Result:... → CREATE
        def get_all_products_by_query(self, query) → Result:...
        def get_all_products(self) → Result:...
        def get_product_by_query(self, query) → Result:...
        def get_products_by_uuids(self, uuids) → Result:...

        def update_product(self, query, new_data) → Result:... → UPDATE
        def delete_product(self, query) → Result:... → DELETE
        def delete_all_products_by_actual_month(self) → Result:...
```

The diagram illustrates the ProductLikesRepository class structure. It highlights several methods with colored boxes and arrows indicating their function:

- CREATE:** `insert_products_json(self, products_data)` (blue box, blue arrow)
- READ:** `get_all_products_by_query(self, query)`, `get_all_products(self)`, `get_product_by_query(self, query)`, `get_products_by_uuids(self, uuids)` (green box, green arrow)
- UPDATE:** `update_product(self, query, new_data)` (yellow box, yellow arrow)
- DELETE:** `delete_product(self, query)`, `delete_all_products_by_actual_month(self)` (red box, red arrow)

## 5.4 SERVICIOS

### 5.4.1 PROCESAMIENTO DE PRODUCTOS

Desde el servicio de productos delegamos el procesamiento de cualquier solicitud de los endpoints y ajustamos la respuesta recogiendo los datos del repositorio.

- Se realiza el CRUD estándar mencionado en los repositorios aplicando distintos filtros o cálculos, un ejemplo de uso sería el método `get_products_with_good_profit`, donde se realizan consultas básicas de **READ** al repositorio y se realiza el **CALCULO** de ganancias respecto al histórico pasado:

```
class ProductService:    You, 2 months ago • [Add] Schema validation in MongoDB, ProductServ...
    def __init__(self, product_repository: ProductRepository) -> None:
        self.product_repository = product_repository

    def get_products_with_good_profit(self) -> list:
        actual_month = datetime.datetime.now().strftime("%Y-%m")
        result_actual_products = self.product_repository.get_products_by_date(...)

        if result_actual_products.is_failure():
            ...

        last_month = (datetime.datetime.now() - datetime.timedelta(days=30)).strftime("%Y-%m")
        result_last_products = self.product_repository.get_products_by_date(last_month)
        if result_last_products.is_failure():
            ...

        ##### Difference key ["price_by_standard_measure"] and calculate the profit #####
        last_products_list_json = result_last_products.value
        actual_products_list_json = result_actual_products.value

        for last_product in last_products_list_json:
            for actual_product in actual_products_list_json:
                if (
                    last_product["uuid"] == actual_product["uuid"]
                    and last_product["price_by_standard_measure"]
                    != actual_product["price_by_standard_measure"]
                ):
                    actual_product["profit"] = (
                        last_product["price_by_standard_measure"]
                        - actual_product["price_by_standard_measure"]
                    )
                    actual_product["profit_percentage"] = (
                        actual_product["profit"]
                        / last_product["price_by_standard_measure"]
                    ) * 100

        # Only get the actual products that has key profit
        actual_products_list_json = [
            product for product in actual_products_list_json if "profit" in product
        ]

    return actual_products_list_json
```

## 5.4.2 SCRAPPING Y REQUISITOS

Cada mercado dispone de un servicio particular ya que la **disposición de los elementos HTML en cada página web son totalmente distintos**, por lo que habría que realizar un estudio a toda la estructura HTML y minuciosamente buscar las etiquetas específicas de donde recoger la información de las páginas webs públicas y totalmente accesibles a todo público.

Todos los mercados tienen estandarizada la implementación [de la FACHADA](#):

```
# Chrome Service
service = Service()
options = webdriver.ChromeOptions()
driver_chrome = webdriver.Chrome(service=service, options=options)

# Fachade
database_manager = DatabaseManager(CONNECTION_MONGO, DATABASE_NAME)
product_repository = ProductRepository(database_manager, COLLECTION_NAME)

# Services
product_service = ProductService(product_repository)
scrapping_service = ScrappingService(driver_chrome, product_service)

scrapping_service.run_scrapping_ahorramas()
```

- Se ha utilizado la [librería](#) de Python ([BeautifulSoup](#)) para recolectar los datos públicos de las páginas web de los distintos mercados.
- Requiere de un dispositivo que admita ejecutar [Google Chrome](#) ya que necesitamos descargar y ejecutar [chromedriver](#) mediante el servicio utilizando la [librería Selenium](#).
- Finalmente, para ejecutar la recolección se debe realizar manualmente mediante una máquina que disponga de los requisitos anteriores y un entorno de ejecución Python.
  - Sistema Operativo:
    - ✓ Windows 10 o superior
  - Python 3.6 o superior.

## 5.4.3 CÁLCULO DIARIO

Diarialmente a las **03:00 am** el servidor realiza el cálculo de **los productos TOP** ya que es un proceso complejo debido al volumen de datos que disponemos de productos.

Mediante la librería [APScheduler](#) nos facilita la ejecución de funciones agendadas:

```
scheduler = BackgroundScheduler()
trigger = CronTrigger(hour=3, minute=0)
scheduler.add_job(calculate_daily_tasks, trigger)
scheduler.start()
```



## 5.5 ENDPOINTS

Desde el servicio de Wewiza se realizan llamadas a los mercados que disponen de endpoints privados, donde se enruta a los distintos mercados para realizar solicitudes y decidir qué hacer con ellas.

### **Servicios ofrecidos por Wewiza:**

Desde Wewiza disponemos los siguiente endpoints como servicios para facilitar al desarrollador.

#### 5.5.1 ROOT

<https://wewiza.ddns.net>

Informamos sobre la disposición de la documentación facilitada por FastAPI:

##### Response body

```
{  
  "docs": [  
    {  
      "endpoint": "/docs",  
      "description": "To see all endpoints allowed and documentation"  
    }  
  ]  
}
```

#### 5.5.2 CATEGORÍAS TOP

<https://wewiza.ddns.net/categories/top>

Mediante una lista de modelos categorías top, son calculadas en función de los productos top:

##### Response body

```
[  
  {  
    "id": "agua_y_refrescos",  
    "name": "Agua y refrescos",  
    "icon": "https://raw.githubusercontent.com/JiaChengZhang14/Wewiza-Icons/44/  
  },  
  {  
    "id": "azucar_caramelos_y_chocolate",  
    "name": "Azúcar, caramelos y chocolate",  
    "icon": "https://raw.githubusercontent.com/JiaChengZhang14/Wewiza-Icons/44/  
  },  
  {  
    "id": "postres_y_yogures",  
    "name": "Postres y yogures",  
    "icon": "https://raw.githubusercontent.com/JiaChengZhang14/Wewiza-Icons/44/  
  }  
]
```

### **5.5.3 PRODUCTOS TOP**

<https://wewiza.ddns.net/products/top>

Devuelve una lista de modelos producto donde puede haber dos casos:

- Productos más gustados son aquellos que en el **profit** y **profit percentage** son igual a 0:

```
[{"id": 1, "product": {"name": "Aqua con gas Perrier 0,5l", "category_id": "agua_y_refrescos", "price": 1.07, "quantity_measure": 0.5, "measure": "l", "price_by_standard_measure": 2.1, "image_url": "https://www.ahorramas.com/dw/image/v2/BFH_PRD/on/demandware.static/-/wcsstore-ProductImage/051-34266_001.jpg", "url": "https://www.ahorramas.com//agua-con-gas-perrier-05l-34266.html", "store_name": "Ahorramas", "store_image_url": "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQ", "date_created": "2024-05-21 19:07:21", "profit": 0, "profit_percentage": 0, "num_likes": 3}}, {"id": 2, "product": {"name": "Aqua con gas Perrier 0,5l", "category_id": "agua_y_refrescos", "price": 1.07, "quantity_measure": 0.5, "measure": "l", "price_by_standard_measure": 2.1, "image_url": "https://www.ahorramas.com/dw/image/v2/BFH_PRD/on/demandware.static/-/wcsstore-ProductImage/051-34266_001.jpg", "url": "https://www.ahorramas.com//agua-con-gas-perrier-05l-34266.html", "store_name": "Ahorramas", "store_image_url": "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQ", "date_created": "2024-05-21 19:07:21", "profit": 0, "profit_percentage": 0, "num_likes": 3}}]
```

- Productos con una **ganancia** en comparación con el scrapping mensual anterior:

```
[  
]  
{  
    "uuid": "166b989d-9e6d-4827-9c59-6568be6793ce",  
    "category_id": "postres_y_yogures",  
    "name": "Cheesecake con ar\u00e1ndanos Hacendado",  
    "price": 1.81,  
    "quantity_measure": 190,  
    "measure": "g",  
    "price_by_standard_measure": 8.68,  
    "image_url": "https://prod-mercadona.imgix.net/images/fc22",  
    "url": "https://tienda.mercadona.es/categories/111",  
    "store_name": "Mercadona",  
    "store_image_url": "https://mirasol-centre.com/nousite/wp-",  
    "date_created": "2024-05-18 23:16:15",  
    "profit": 0.32,  
    "profit_percentage": 6,  
    "num_likes": 1  
}
```



## 5.5.4 CATEGORÍAS

<https://wewiza.ddns.net/categories>

Devuelve un modelo **categories** que contiene una lista de modelos **category**:

```
{  
  "categories": [  
    {  
      "id": "aceite_especias_y_salsas",  
      "name": "Aceite, especias y salsas",  
      "icon": "https://raw.githubusercontent.com/JiaChen/.../  
    },  
    {  
      "id": "agua_y_refrescos",  
      "name": "Agua y refrescos",  
      "icon": "https://raw.githubusercontent.com/JiaChen/.../  
    }  
  ]  
}
```

## 5.5.5 PRODUCTOS POR CATEGORÍA

[https://wewiza.ddns.net/products/category/id/{category\\_id}](https://wewiza.ddns.net/products/category/id/{category_id})

Cada Mercadona como clave con sus correspondientes productos filtrados por las categorías:

The diagram illustrates a transformation process. On the left, there is a JSON object with three keys: "mercadona", "ahorramas", and "carrefour", each pointing to a list of products. An arrow points from this structure to the right, where a single Mercadona store is shown with its unique identifier, category ID, product name, price, quantity measure, and other details.

```
{  
  "mercadona": [ ... ],  
  "ahorramas": [ ... ],  
  "carrefour": [ ... ]  
}  
  
{  
  "mercadona": [  
    {  
      "uuid": "24bd5b31-474e-4ca2-83d4-489f10b5d868",  
      "category_id": "aceite_especias_y_salsas",  
      "name": "Aceite de oliva 0,4º Hacendado",  
      "price": 9.01,  
      "quantity_measure": 1,  
      "measure": "l",  
      "price_by_standard_measure": 8.95,  
      "image_url": "https://prod-mercadona.imgix.net/images/",  
      "url": "https://tienda.mercadona.es/categories/112",  
      "store_name": "Mercadona",  
      "store_image_url": "https://mirasol-centre.com/nousite",  
      "date_created": "2024-05-18 23:26:14",  
      "num_likes": 0  
    },  
    ...  
  ]  
},  
...
```



## 5.5.6 PRODUCTO POR UUID

<https://wewiza.ddns.net/product/id/{uuid}>

Devolvemos el modelo producto encontrado:

Response body

```
{  
    "uuid": "24bd5b31-474e-4ca2-83d4-489f10b5d868",  
    "category_id": "aceite_especias_y_salsas",  
    "name": "Aceite de oliva 0,4º Hacendado",  
    "price": 9.01,  
    "quantity_measure": 1,  
    "measure": "l",  
    "price_by_standard_measure": 8.95,  
    "image_url": "https://prod-mercadona.imgix.net/images/  
    "url": "https://tienda.mercadona.es/categories/112",  
    "store_name": "Mercadona",  
    "store_image_url": "https://mirasol-centre.com/nousit/  
    "date_created": "2024-05-18 23:26:14",  
    "num_likes": 0  
}
```

En caso de no encontrarlo devolvemos un documento básico con información:

```
{"name": "Product not found"}
```

## 5.5.7 NÚMERO DE PRODUCTOS POR MERCADO

[https://wewiza.ddns.net/size/{market\\_name}](https://wewiza.ddns.net/size/{market_name})

Devolvemos un número entero con los productos que disponga el mercado dentro del marco mensual en el que hemos realizado el “scrapping”:

Response body

```
2965
```



## 5.5.8 PRODUCTO CON HISTÓRICO POR UUID

<https://wewiza.ddns.net/product/details/id/{uuid}>

Devolvemos una lista con el histórico del producto, donde el último producto de la lista es el más actual:

```
[  
 {  
   "uuid": "e7c068a8-7edf-4fdb-b459-fb4331d01710",  
   "category_id": "aceite_especias_y_salsas",  
   "name": "Aceite de oliva 0,4º Hacendado",  
   "price": 8.95,  
   "quantity_measure": 1,  
   "measure": "l",  
   "price_by_standard_measure": 8.95,  
   "image_url": "https://prod-mercadona.imgix.net/images/1a84cd7052b68873985104ac24",  
   "url": "https://tienda.mercadona.es/categories/112",  
   "store_name": "Mercadona",  
   "store_image_url": "https://mirasol-centre.com/nousite/wp-content/uploads/2017/0",  
   "date_created": "2024-04-05 00:00:00"  
 },  
 {  
   "uuid": "24bd5b31-474e-4ca2-83d4-489f10b5d868",  
   "category_id": "aceite_especias_y_salsas",  
   "name": "Aceite de oliva 0,4º Hacendado",  
   "price": 9.01,  
   "quantity_measure": 1,  
   "measure": "l",  
   "price_by_standard_measure": 8.95,  
   "image_url": "https://prod-mercadona.imgix.net/images/1a84cd7052b68873985104ac24",  
   "url": "https://tienda.mercadona.es/categories/112",  
   "store_name": "Mercadona",  
   "store_image_url": "https://mirasol-centre.com/nousite/wp-content/uploads/2017/0",  
   "date_created": "2024-05-18 23:26:14",  
   "num_likes": 0  
 }]  
 ]
```



## 5.5.9 PRODUCTOS POR MERCADO

[https://wewiza.ddns.net/products/market/{market\\_name}](https://wewiza.ddns.net/products/market/{market_name})

Devolvemos una lista con todos los productos dentro del marco mensual del “scrapping” mediante un mercado:

```
[  
 {  
     "uuid": "fe145dc6-6cd0-4799-82eb-6c8b33c0d212",  
     "category_id": "postres_y_yogures",  
     "name": "Bífidus desnatado con nueces y cereales Hacendado 0% m",  
     "price": 1.47,  
     "quantity_measure": 500.0,  
     "measure": "g",  
     "price_by_standard_measure": 2.6,  
     "image_url": "https://prod-mercadona.imgix.net/images/a33f37143",  
     "fit=crop&h=300&w=300",  
     "url": "https://tienda.mercadona.es/categories/105",  
     "store_name": "Mercadona",  
     "store_image_url": "https://mirasol-centre.com/nousite/wp-conte",  
     "logo-Mercadona.png",  
     "date_created": "2024-05-18 23:16:06",  
     "num_likes": 0  
 },  
 { ...  
 },  
 { ...  
 }]
```

## 5.5.10 PRODUCTOS POR RANGO Y MERCADO

[https://wewiza.ddns.net/products/market/{market\\_name}/range/{init\\_num}/{end\\_num}](https://wewiza.ddns.net/products/market/{market_name}/range/{init_num}/{end_num})

Devolvemos una lista con todos los productos dentro del marco mensual del “scrapping” actual mediante un rango y mercado para facilitar la recuperación de todos los productos con mayor rapidez:

**! Los rangos mínimo y máximo no pueden ser el mismo.**

```
[  
 {  
     "uuid": "fe145dc6-6cd0-4799-82eb-6c8b33c0d212",  
     "category_id": "postres_y_yogures",  
     "name": "Bífidus desnatado con nueces y cereales Hacendado 0%",  
     "price": 1.47,  
     "quantity_measure": 500,  
     "measure": "g",  
     "price_by_standard_measure": 2.6,  
     "image_url": "https://prod-mercadona.imgix.net/images/a33f37143",  
     "fit=crop&h=300&w=300",  
     "url": "https://tienda.mercadona.es/categories/105",  
     "store_name": "Mercadona",  
     "store_image_url": "https://mirasol-centre.com/nousite/wp-conte",  
     "logo-Mercadona.png",  
     "date_created": "2024-05-18 23:16:06",  
     "num_likes": 0  
 },  
 { ...  
 },  
 { ...  
 },  
 { ...  
 }]
```



### 5.5.11 PRODUCTO (ME GUSTA)

[https://wewiza.ddns.net/like/{uuid}/email/{email\\_user}](https://wewiza.ddns.net/like/{uuid}/email/{email_user})

Mediante el **UUID** del producto y un nick, email o cualquier identificador de un usuario, devolvemos “**True**” en caso de que se haya dado me gusta correctamente o “**False**” en caso de que anteriormente ese producto ya fue dado me gusta por el mismo usuario.

### 5.5.12 PRODUCTO (NO ME GUSTA)

[https://wewiza.ddns.net/unlike/{uuid}/email/{email\\_user}](https://wewiza.ddns.net/unlike/{uuid}/email/{email_user})

Mediante el **UUID** del producto y un nick, email o cualquier identificador de un usuario, devolvemos “**True**” en caso de que se haya dado **NO** me gusta correctamente o “**False**” en caso de que anteriormente ese producto ya fue dado **NO** me gusta por el mismo usuario.

### 5.5.13 ESTADO DE LA REACCIÓN DE UN PRODUCTO POR USUARIO

[https://wewiza.ddns.net/reaction/email/{email\\_user}/product/id/{uuid}](https://wewiza.ddns.net/reaction/email/{email_user}/product/id/{uuid})

Mediante el **UUID** del producto y un nick, email o cualquier identificador de un usuario, devolvemos un **String** “**liked**” en caso de que ese producto le guste a ese usuario, “**unliked**” en caso de que **NO** o “**none**” en caso de que no haya ninguna reacción.

En resumen, estados posibles:

- “**liked**”: al usuario en cuestión le gusta el producto.
- “**unliked**”: al usuario en cuestión **NO** le gusta el producto.
- “**none**”: el usuario en cuestión todavía no ha realizado ninguna acción sobre el producto.

### 5.5.14 INICIO MENSUAL

[https://wewiza.ddns.net/start\\_month/password/{password}](https://wewiza.ddns.net/start_month/password/{password})

PASSWORD: **wewiza**

Al iniciar el periodo mensual y al finalizar el “scrapping” mensual se debe llamar a este endpoint para iniciar los “likes” de los nuevos productos.



## 5.5.15 SUGERENCIAS POR PRODUCTO

[https://wewiza.ddns.net/suggest/id/{uuid}?filter\\_markets={market\\_name}](https://wewiza.ddns.net/suggest/id/{uuid}?filter_markets={market_name})

Como máximo la sugerencia de respuesta será de 3 productos, puede haber menos o directamente ninguno.

Mediante un producto buscamos productos similares con un precio menor, donde podemos filtrar mediante una “query” en el endpoint los distintos mercados.

Para las búsquedas de productos similares se han utilizado las siguientes técnicas:

- **LEMATIZACIÓN:** mediante esta técnica podemos encontrar palabras que dan la misma información, pero de distinta forma presentada, ejemplo: limones -> limón.
- **REGEX:** mediante las palabras encontradas con la lematización, se formula un regex para buscar aquellos productos que correspondan también las mismas palabras, sabiendo que es casi imposible que coincidan todas las palabras buscamos que tengan una coincidencia del **50% o más**.

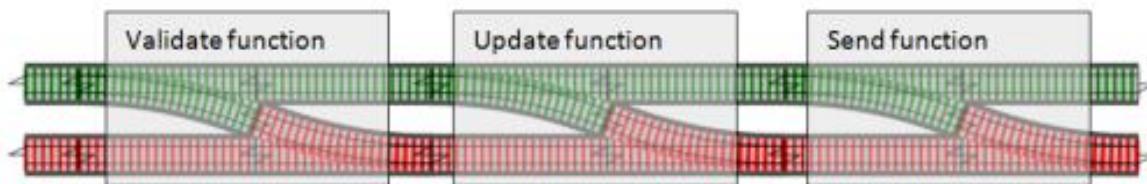
“.\*(palabra\_lematizada\_1 | palabra\_lematizada\_2 | ... | palabra\_lematizada\_N).\*”

Productos que sugiere wewiza más baratos y similares al que ha preguntado el usuario:

```
[{"id": "1", "name": "Pan rallado Hacendado", "category_id": "panaderia_y_pasteleria", "price": 1.11, "quantity_measure": 750, "measure": "g", "price_by_standard_measure": 1.2, "image_url": "https://prod-mercadona.imgix.net/images/categories/62/icon.png", "url": "https://tienda.mercadona.es/categories/62", "store_name": "Mercadona", "store_image_url": "https://mirasol-centre.com/nous", "date_created": "2024-05-18 23:17:31"}, {"id": "2", "name": "Pan de molde integral", "category_id": "panaderia_y_pasteleria", "price": 1.2, "quantity_measure": 500, "measure": "g", "price_by_standard_measure": 1.2, "image_url": "https://prod-mercadona.imgix.net/images/categories/62/icon.png", "url": "https://tienda.mercadona.es/categories/62", "store_name": "Mercadona", "store_image_url": "https://mirasol-centre.com/nous", "date_created": "2024-05-18 23:17:31"}, {"id": "3", "name": "Pan de molde integral", "category_id": "panaderia_y_pasteleria", "price": 1.2, "quantity_measure": 500, "measure": "g", "price_by_standard_measure": 1.2, "image_url": "https://prod-mercadona.imgix.net/images/categories/62/icon.png", "url": "https://tienda.mercadona.es/categories/62", "store_name": "Mercadona", "store_image_url": "https://mirasol-centre.com/nous", "date_created": "2024-05-18 23:17:31"}]
```

## 5.6 RAILWAY ORIENTED PROGRAMMING

Mediante (ROP) enfocamos la programación centrando el flujo de datos y posibles errores de una manera más estructurada y sencilla de comprender



En este ejemplo podemos apreciar en nuestra función que estamos insertando un producto en formato Json, esta función se encuentra en un servicio y se redirige al repositorio:

```
def create_product_to_mongo_recieving_json(self, product_json: str) → Any | None:
    product_dict = json.loads(product_json)

    result = self.product_repository.insert_product(product_dict)
    if result.is_failure():
        print("Failed to insert product:", result.error)
        return result.error
    else:
        return result.value
```

Llegando al repositorio, siempre disponemos de los dos caminos, el camino del éxito y el camino del error:

```
def insert_product(self, product_data) → Result:
    try:
        database = self.db_manager.connect_database()
        collection = database[self.collection_name]
        result = collection.insert_one(product_data)
        # Because ObjectId needs to be converted to string
        inserted_id = str(result.inserted_id)
        self.db_manager.close_database()
        return Result.success(inserted_id)
    except Exception as e:
        return Result.failure(str(e))
```

## 5.7 SEGURIDAD

### 5.7.1 SSL

Para las conexiones al servidor se realizan mediante HTTPS, por lo que requerimos de un certificado SSL, en nuestro caso se auto firmó con [OpenSSL](#).

Desde el dispositivo Android en el que se realizarán las peticiones a los distintos “endpoints”, requerirá de la instalación del certificado para que el servidor confíe en él.



## 5.8 DEPENDENCIAS

### 5.8.1 SERVIDOR

Se han seleccionado las dependencias teniendo en cuenta que se ajusten al procesador ARMv7 del servidor:

- TextBlob (0.8.0): <https://textblob.readthedocs.io/en/dev/install.html>
- FastAPI (0.68.0): <https://fastapi.tiangolo.com/>
- Requests (2.26.0): <https://pypi.org/project/requests/2.26.0/>
- Uvicorn (0.15.0): <https://pypi.org/project/uvicorn/0.15.0/>
- Aiofiles (0.8.0): <https://pypi.org/project/aiofiles/0.8.0/>
- Pymongo (3.11.4): <https://pypi.org/project/pymongo/3.11.4/>
- Python-dotenv (1.0.1): <https://pypi.org/project/python-dotenv/>
- Python-on-rails (1.0.1): <https://pypi.org/project/python-on-rails/>
- APScheduler (3.10.0): <https://pypi.org/project/APScheduler/3.10.0/>

### 5.8.2 SCRAPPING

- Selenium (4.10.0): <https://pypi.org/project/selenium/4.10.0/>
- BeautifulSoup4 (4.10.0): <https://pypi.org/project/beautifulsoup4/4.10.0/>



## 6 DESARROLLO FRONT-END

### 6.1 INTERFAZ

Para la creación de la interfaz gráfica de usuario (GUI), se ha optado por utilizar Jetpack Compose. Jetpack Compose es una moderna herramienta de UI toolkit para Android que simplifica y acelera el desarrollo de interfaces de usuario. A continuación, se presenta una descripción completa de cómo se ha utilizado Jetpack Compose en este proyecto.



#### 6.1.1 INTRODUCCIÓN

- Descripción General:** Jetpack Compose es un kit de herramientas de UI completamente declarativo que facilita la creación de interfaces de usuario de manera más intuitiva y eficiente. Permite definir la UI en Kotlin, lo que hace que el código sea más sencillo y menos propenso a errores.
- Ventajas:** Jetpack Compose ofrece varias ventajas sobre los enfoques tradicionales, como una mejor integración con Kotlin, menos código boilerplate, una mejor capacidad de prueba y una actualización más sencilla de los componentes de la UI.

#### 6.1.2 PANTALLAS Y COMPOSABLES



##### 6.1.2.1 Pantalla de bienvenida

Es una pantalla donde el usuario puede elegir registrarse con Google, registrarse con un correo o en el caso de que ya tenga una cuenta creada, iniciar sesión con el correo. Para las opciones de registro e inicio de sesión con correo, se han creado componentes personalizados para poder navegar hacia las siguientes vistas.

```
@Composable
private fun LoginButton(
    viewModel: WelcomeScreenViewModel,
    navController: NavController
) {
    Button(
        onClick = { viewModel.navigateToLoginScreen(navController) },
        modifier = Modifier
            .padding(top = 20.dp)
            .padding(start = 20.dp)
            .padding(end = 20.dp)
            .fillMaxWidth()
            .height(50.dp),
        colors = ButtonDefaults.buttonColors(
            containerColor = MaterialTheme.colorScheme.primary,
            contentColor = Color.White // Color del texto en el botón
        ),
        shape = MaterialTheme.shapes.medium
    ) {
        Text(text = "Iniciar Sesión", style = TextStyle(fontSize = 20.sp), fontFamily = FirsNeue)
    }
}
```

Ejemplo de código



### 6.1.2.2 Pantalla de inicio de sesión

Pantalla donde el usuario podrá iniciar sesión con su correo y contraseña registrados previamente en Wewiza.

```
@Composable
fun LoginScreenBodyContent(viewModel: LoginScreenViewModel, navController: NavController) {
    val context : Context = LocalContext.current
    Box(modifier = Modifier.run {
        background(MaterialTheme.colorScheme.surface)
    }) {
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(top = 40.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Image(
                painter = painterResource(id = R.drawable.loading),
                contentDescription = "Logo",
                modifier = Modifier.size(300.dp)
            )
            Text(
                text = "¡Inicia sesión en Wewiza!", style = TextStyle(
                    fontSize = 30.sp,
                    fontWeight = FontWeight.ExtraBold,
                    fontFamily = Butler,
                    color = Color.Black
                )
            )
            LoginTextInputs(viewModel)
            Spacer(modifier = Modifier.size(20.dp))
            LoginScreenButtons(viewModel, navController, context)
        }
    }
}
```

Ejemplo de código.



### 6.1.2.3 Pantalla de registro

Pantalla donde el usuario podrá registrarse con un correo y una contraseña de al menos 6 caracteres.

```
@Composable
fun RegisterScreenBodyContent(viewModel: RegisterScreenViewModel, navController: NavController) {
    val context : Context = LocalContext.current
    Box(modifier = Modifier.run {
        background(MaterialTheme.colorScheme.surface)
    }) {
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(top = 40.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Image(
                painter = painterResource(id = R.drawable.loading),
                contentDescription = "Logo",
                modifier = Modifier.size(300.dp)
            )
            Text(
                text = "¡Regístrate en Wewiza!", style = TextStyle(
                    fontSize = 30.sp,
                    fontWeight = FontWeight.ExtraBold,
                    fontFamily = Butler,
                    color = Color.Black
                )
            )
            RegisterTextInputs(viewModel)
            Spacer(modifier = Modifier.size(20.dp))
            RegisterScreenButtons(viewModel, navController, context)
        }
    }
}
```

Ejemplo de código



**Categorías Destacadas**

Ver más

- Agua y refrescos
- Azúcar, caramelos y chocolate
- Bebé

**Productos Destacados**

Ver más

- MÁS GUSTADO!

Destacado Buscador Mis listas Mi perfil

#### 6.1.2.4 Pantalla de destacados

Esta pantalla presenta categorías y productos destacados. Las categorías se destacan en función de los productos destacados. Los productos destacados se eligen por su popularidad entre los usuarios y su alto porcentaje de beneficio, proporcionando a los usuarios acceso rápido a los artículos más relevantes y rentables.

```
@Composable
fun HomeBodyContent(
    navController: NavController,
    viewModel: HomeScreenViewModel,
) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .background(MaterialTheme.colorScheme.surface)
    ) {
        WewizaLogoSection()
        FeaturedCategoriesSection(navController, viewModel)
        FeaturedProductsSection(navController, viewModel)
    }
}
```

Ejemplo de código: Muestra la estructura de la vista.

Buscar Categoría

- Aceite, especias y salsas
- Agua y refrescos
- Aperitivos
- Arroz, legumbres y pasta
- Azúcar, caramelos y chocolate
- Bebé
- Bodega
- Cacao, café e infusiones
- Carne

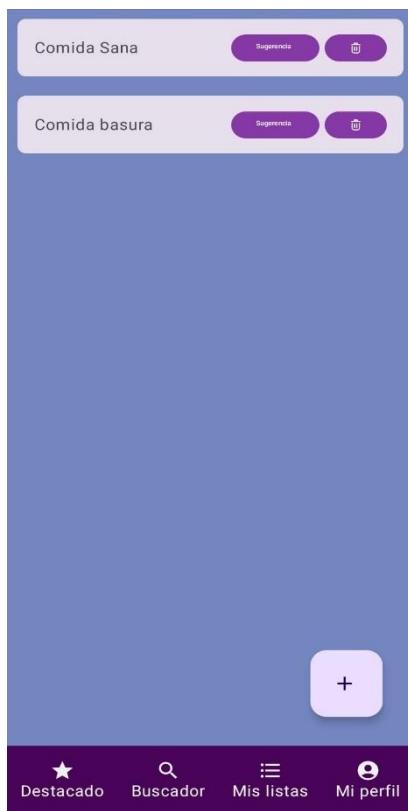
Destacado Buscador Mis listas Mi perfil

#### 6.1.2.5 Pantalla de búsqueda

Esta pantalla está dedicada a la búsqueda de categorías. Al seleccionar una categoría, el usuario es redirigido a la vista correspondiente de productos dentro de esa categoría.

```
@Composable
fun CategoryItem(
    category: Category,
    viewModel: CategoriesScreenViewModel,
    navController: NavController
) {
    //can only retrieve svg from github repo with this method.
    val painter: AsyncImagePainter = rememberAsyncImagePainter(
        model = ImageRequest.Builder(context = LocalContext.current)
            .data(category.icon)
            .decoderFactory(SvgDecoder.Factory())
            .build()
    )
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp),
        shape = RoundedCornerShape(8.dp),
        onClick = {viewModel.navigateToProductsScreen(category.id, navController)}
    ) {
        Row(
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier.padding(16.dp)
        ) {
            Image(
                painter = painter,
                contentDescription = category.name,
                modifier = Modifier.size(24.dp)
            )
            Spacer(modifier = Modifier.width(30.dp))
            Text(
                text = category.name,
                modifier = Modifier.padding(top = 8.dp)
            )
        }
    }
}
```

Ejemplo de código: Componente personalizado de categoría.



### 6.1.2.6 Pantalla de listas de usuario

Pantalla dedicada al usuario para la creación y eliminación de listas de compra. Habrá una opción de sugerencias, donde se le sugieran productos más asequibles de la franquicia de supermercados que el usuario elija.

```
@Composable
fun MyListsScreenBodyContent(
    viewModel: MyListsScreenViewModel,
    navController: NavController,
    showDialog: MutableState<Boolean>,
) {
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(bottom = 70.dp)
    ) {
        Column {
            val myLists: MutableList<ShoppingList> by remember { viewModel.myLists }

            LazyColumn {
                items(myLists.size) { index ->
                    MyListItem(
                        myLists[index], viewModel, navController
                    )
                }
            }
            Box(
                modifier = Modifier
                    .fillMaxSize(), contentAlignment = Alignment.BottomEnd
            ) {
                AddNewListButton(showDialog, viewModel)
            }
        }
    }
}
```

Ejemplo de código.



### 6.1.2.7 Pantalla de perfil.

Pantalla donde se muestran los datos del perfil del usuario, tales como el nombre de usuario y la foto de perfil, ambas modificables. También dispondrá de acceso a las listas de usuario (pantalla anterior), un acceso para poder contactar con personas para soporte y un acceso a los ajustes.

```
@Composable
fun ProfilePicture(
    imageUrl: String,
    context: Context,
    isEditing: Boolean,
    onImageSelected: (String) -> Unit
) {
    val pickImageContract = ImagePickerContractsContract(Info = ActivityContracts.GetContent())
    // Aquí puedes marcar el url de la imagen seleccionada
    // Por ejemplo, puedes convertir a una cadena y asignarla a selectedImage
    val url: Uri? by remember { mutableStateOf(null) }
    val painter: Painter = if (imageUrl.isNotBlank()) {
        rememberImagePainter(url)
    } else {
        rememberImagePainter(
            data = ImageUrl(
                builder = {
                    crossfade(true)
                    placeholder(R.drawable.defaultprofilepic)
                }
            )
        )
    }

    Image(
        painter = painter,
        contentDescription = null,
        modifier = Modifier
            .size(150.dp)
            .clip(CircleShape)
            .clickable {
                Toast
                    .makeText(context, "Imagen clickeada", Toast.LENGTH_SHORT)
                    .show()
            }
            .clickable {
                if (isEditing) {
                    pickImageContract.launching("image/*")
                }
            },
        contentScale = ContentScale.Crop,
    )
}
```

Ejemplo de código: Imagen de perfil.



### 6.1.2.8 Pantalla de ajustes.

Pantalla donde el usuario podrá realizar acciones como cerrar sesión, eliminar su cuenta o ver más sobre nosotros.

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SettingsScreenBodyContent(
    viewModel: SettingsScreenViewModel,
    navController: NavController,
    mainActivity: MainActivity
) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .background(MaterialTheme.colorScheme.surface)
            .padding(vertical = 70.dp)
    ) {
        val context : Context = LocalContext.current

        Spacer(modifier = Modifier.height(20.dp))

        Divider(color = Color.Black, thickness = 1.dp)

        Spacer(modifier = Modifier.height(20.dp))
        AboutUsSection(navController)
        Spacer(modifier = Modifier.height(20.dp))

        Divider(color = Color.Black, thickness = 1.dp)

        Spacer(modifier = Modifier.height(20.dp))
        SignOutSection(viewModel, context, navController, mainActivity)
        Spacer(modifier = Modifier.height(20.dp))

        Divider(color = Color.Black, thickness = 1.dp)

        Spacer(modifier = Modifier.height(20.dp))
        DeleteAccountSection(viewModel, navController, context, mainActivity)
        Spacer(modifier = Modifier.height(20.dp))
    }
}
```

Ejemplo de código: división de la pantalla por secciones.



### 6.1.2.9 Pantalla de productos.

Pantalla donde el usuario puede buscar el producto que deseé, tanto por nombre o por filtros de categoría, tienda y precio. Al pulsar sobre un producto, el usuario es transportado a la pantalla de detalles de producto.

```
@Composable
fun ProductScreenBodyContent(
    viewModel: ProductScreenViewModel,
    navController: NavHostController,
    searchText: String,
    isMercadonaChecked: Boolean,
    isHorramasChecked: Boolean,
    isCarrefourChecked: Boolean,
    selectedCategories: MutableState<Map<String, Boolean>>,
    priceSortOrder: String
) {
    val products : List<Product> = viewModel.allProductsList.collectAsState()
    val isProductsLoading : Boolean = viewModel.isProductsLoading.collectAsState()

    val filteredProducts : List<Product> = products
        .filter { it.name.contains(searchText, ignoreCase = true) }
        .filter {
            ((it.store_name == "Mercadona" && isMercadonaChecked) ||
            (it.store_name == "Ahorramás" && isHorramasChecked) ||
            (it.store_name == "Carrefour" && isCarrefourChecked)) &&
            selectedCategories.value[it.category_id] == true
        }
        .sortedWith(
            when (priceSortOrder) {
                "Ascendente" -> compareBy { it.price }
                "Descendente" -> compareByDescending { it.price }
                else -> compareBy { it.uid } // Default order if no sorting is selected
            }
        )

    if (isProductsLoading) {
        Box(modifier = Modifier.fillMaxSize()){
            CircularProgressIndicator(modifier = Modifier.align(Alignment.Center))
        }
    } else{
        LaunchedEffect(products){
            delay( 1000 )
        }
        ProductGrid(
            products = filteredProducts,
            isMercadonaChecked,
            isHorramasChecked,
            isCarrefourChecked,
            selectedCategories,
            viewModel,
            navController
        )
    }
}
```

Ejemplo de código.

**Wewiza**



**Cebolla dulce malla 700g**

Precio: 2.29 €  
Precio por medida: 3.27 €/Kg AhorraMas

👎 0 👍



Ejemplo de código: Pantalla de detalles de producto.

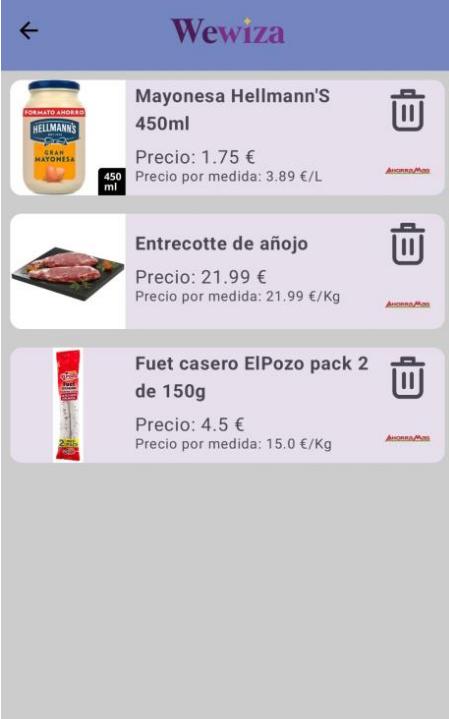
```

@Composable
fun GraphicField(viewModel: ProductDetailsScreenViewModel) {
    val productHistoryDetails : MutableList<Product> by sharedViewModel.productHistoryDetails
    val isDataLoaded : Boolean = productHistoryDetails.isNotEmpty()
    Log.d("tag: "ProductDetailsScreen", msg: "ProductHistoryDetails: $productHistoryDetails")
    Log.d("tag: "ProductDetailsScreen", msg: "CurrentProduct: ${sharedViewModel.getCurrentProduct().uuid}")

    if (isDataLoaded) {
        val sortedDetails : List<Product> = productHistoryDetails.sortedBy { it.date_created }
        val (entries : List<Entry>, monthMap : Map<Float, String>) = viewModel.prepareChartData(sortedDetails)
        Log.d("tag: "ProductDetailsScreen", msg: "Entries: $entries")
        LineChartView(entries, monthMap)
    } else {
        CircularProgressIndicator()
    }
}

```

**Wewiza**



Total: 28,24 € Sugerencias

#### 6.1.2.10 Pantalla de detalles de producto.

Pantalla donde se muestran los datos del producto más en detalle, junto a su precio, precio por medida y opciones para poder valorar el producto. El producto puede ser añadido a una lista que el usuario haya creado previamente. Se muestra una gráfica donde se puede ver la evolución del precio del producto a lo largo de los meses.

```

@Composable
fun GraphicField(viewModel: ProductDetailsScreenViewModel) {
    val productHistoryDetails : MutableList<Product> by sharedViewModel.productHistoryDetails
    val isDataLoaded : Boolean = productHistoryDetails.isNotEmpty()
    Log.d("tag: "ProductDetailsScreen", msg: "ProductHistoryDetails: $productHistoryDetails")
    Log.d("tag: "ProductDetailsScreen", msg: "CurrentProduct: ${sharedViewModel.getCurrentProduct().uuid}")

    if (isDataLoaded) {
        val sortedDetails : List<Product> = productHistoryDetails.sortedBy { it.date_created }
        val (entries : List<Entry>, monthMap : Map<Float, String>) = viewModel.prepareChartData(sortedDetails)
        Log.d("tag: "ProductDetailsScreen", msg: "Entries: $entries")
        LineChartView(entries, monthMap)
    } else {
        CircularProgressIndicator()
    }
}

```

Ejemplo de código: Gráfica.

#### 6.1.2.11 Pantalla de lista de usuario.

Tras haber añadido uno o varios productos a la lista de usuario, se mostrarán en esta pantalla junto a ciertos detalles y la opción de eliminarlos de la lista. Además, al presionar el botón de sugerencias, se navegará hacia la pantalla de sugerencias.

```

fun ListScreenListScreenViewModel: ListScreenViewModel, navController: NavController) {
    val selectedProductsIds : ShoppingList? = sharedViewModel.selectedList.value
    var productsList : List<Product> by remember { mutableStateOf(emptyList<Product>()) }
    Log.d("tag: "ListScreen", msg: "Selected Products: ${selectedProductsIds?.products.size}")
    var loading : Boolean by remember { mutableStateOf(false) }

    if (selectedProductsIds.products.isNotEmpty()) {
        LaunchedEffect(selectedProductsIds) {
            loading = true
            val products : List<Product> = listScreenViewModel.getProductsFromList(selectedProductsIds)
            productsList = products
            Log.d("tag: "ListScreen", msg: "Products: $products")
            loading = false
        }
    }

    MyLightTheme {
        Scaffold(
            topBar = {
                Constants.TopBarWithLogo(navController)
            },
        ) {
            if (loading) {
                Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
                    CircularProgressIndicator()
                }
            } else {
                ListScreenBodyContent(
                    listScreenViewModel,
                    navController,
                    productsList
                ) { updatedProductsList ->
                    productsList = updatedProductsList
                }
            }
        }
    }
}

```

Ejemplo de código.



## Wewiza

Mayonesa Hellmann's 450ml  
Precio: 1.75 € Precio/medida: 3.89 €/L

No hay sugerencias para este producto

Entrecot de añojo  
Precio: 21.99 € Precio/medida: 21.99 €/Kg

Albóndigas de Añojo Círculo de Calidad 360 g  
Precio: 4.65 € Precio/medida: 12.92 €/Kg

Aceptar sugerencias

### 6.1.2.12 Pantalla de sugerencias.

Pantalla donde a partir del mercado seleccionado por el usuario, se mostrarán unas sugerencias de los productos de la lista que se haya elegido. Al aceptar la sugerencia, los productos originales de la lista se sustituirán por los productos sugeridos. Por cada producto, puede haber hasta 3 sugerencias y el usuario tendrá que elegir entre una de esas 3.

```
@Composable
fun AcceptSuggestionsButton(
    viewModel: SuggestionScreenViewModel,
    navController: NavController,
    shoppingListUUID: String
) {
    Button(
        onClick = {
            viewModel.acceptSuggestions(shoppingListUUID)
            viewModel.navigateToMyListsScreen(navController)
        },
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        shape = RoundedCornerShape(percent = 50), // Bordes redondeados
        colors = ButtonDefaults.buttonColors(
            containerColor = MaterialTheme.colorScheme.primary, // Color de fondo del botón
            contentColor = Color.White // Color del texto del botón
        )
    ) {
        Text(
            text = "Aceptar sugerencias",
            fontSize = 18.sp,
            fontWeight = FontWeight.Bold,
            modifier = Modifier.padding(vertical = 8.dp)
        )
    }
}
```

Ejemplo de código: botón de aceptar sugerencias.

¿Cómo podemos ayudarte?

Escribe tu mensaje aquí

Enviar ➤

### 6.1.2.13 Pantalla de soporte.

En caso de que el usuario tenga alguna duda o sugerencia sobre la aplicación, podrá enviar un mensaje directo a los desarrolladores.

```
@Composable
fun SendButton(viewModel: CustomerSupportScreenViewModel, userMsg: MutableState<TextFieldValue>, context: Context) {
    Button(onClick = {
        viewModel.sendMessage(userMsg.value.text, context)
        userMsg.value = TextFieldValue("") // Clear the text field after sending the message
    }) {
        Row(
            modifier = Modifier
                .size(270.dp, 40.dp)
                .align(Alignment.CenterVertically), // This will center the items vertically
            horizontalArrangement = Arrangement.Center, // This will center the items horizontally
            verticalAlignment = Alignment.CenterVertically // This will center the items vertically
        ) {
            Text(
                text = "Enviar",
                modifier = Modifier.weight(1f),
                fontSize = 20.sp,
                fontFamily = FirsNeue
            )
            Icon(
                imageVector = Icons.Default.Send,
                contentDescription = "Enviar",
                modifier = Modifier.size(24.dp)
            )
        }
    }
}
```

Ejemplo de código: botón de enviar.



# Wewiza

Servidor en mantenimiento.  
Disculpe las molestias.

## 6.1.2.14 Pantalla de mantenimiento.

Pantalla que se mostrará al usuario en caso de que acceda a la aplicación y el servidor esté bajado por cuestiones de mantenimiento. De esta manera nos aseguramos de la que aplicación no falle, aunque el servidor no esté operativo.

```
@Composable
fun ServerMaintenanceScreen() {
    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier.fillMaxSize(),
        verticalArrangement = androidx.compose.foundation.layout.Arrangement.Center
    ) {
        Image(painter = painterResource(id = R.drawable.logo_letras), contentDescription = "logo")
        Spacer(modifier = Modifier.height(80.dp))
        Text(
            text = "Servidor en mantenimiento. Disculpe las molestias.",
            color = MaterialTheme.colorScheme.primary,
            fontWeight = FontWeight.Bold,
            fontSize = 24.sp,
            textAlign = TextAlign.Center
        )
    }
}
```

Ejemplo de código: Pantalla de mantenimiento.

## 6.2 FIREBASE

### 6.2.1 USUARIOS

Desde la aplicación, los usuarios pueden iniciar sesión para crear una cuenta. Esta cuenta se almacena en colecciones de *Firebase Firestore*.

The screenshot shows a user document in the Firestore database. The structure is as follows:

- + Iniciar colección
- + Agregar campo
  - email: "prueba@gmail.com"
  - imageUrl: ""
  - name: "prueba"
- recentSearches
  - reviews: 12
- shoppingListsList
  - 0
    - name: "lista nueva" (cadena)
  - products
    - 0 "0018121b-c540-4548-a936-0ef92a49f278"
    - uuid: "23b5f758-00c8-4038-b340-f8d1bbca390b"

#### Dentro de Firebase se almacenarán:

- **Correo:** se usará como elemento identificador principal.
- **Imagen:** Enlace hacia la foto de perfil del usuario (al iniciar sesión con Google, la imagen por defecto será la de la cuenta de Google).
- **Nombre:** nombre de usuario.
- **Búsquedas recientes:** se usará para poder mostrar al usuario sus últimas búsquedas (se ha decidido implementar en futuras versiones de la app).
- **Contador de participaciones:** se usará para monitorizar la participación del usuario y poder entregar recompensas visuales al usuario. (se ha decidido implementar en futuras versiones de la app).
- **Listas:** Los usuarios disponen de la funcionalidad de crear listas de compra donde pueden almacenar diversos productos. Es posible la creación de varias listas de la compra.

## 6.3 INVERSIÓN DE CONTROL/INYECTOR DE DEPENDENCIAS

Durante el desarrollo de nuestra aplicación, hemos identificado la necesidad de implementar un inyector de dependencias para optimizar la gestión de nuestras dependencias y mejorar la modularidad del código. Después de una exhaustiva investigación y múltiples sesiones de debate, el equipo ha decidido utilizar Koin como la solución más adecuada.



Koin es un liviano e intuitivo marco de inyección de dependencias diseñado específicamente para aplicaciones Android y Kotlin. Facilita la gestión de dependencias mediante un DSL (Domain Specific Language) de Kotlin, permitiendo definir y proporcionar dependencias de manera sencilla y eficiente, sin la necesidad de utilizar archivos de configuración XML adicionales. Esta elección se alinea con nuestra visión de mantener un código limpio, modular y fácil de mantener, asegurando una mayor escalabilidad y flexibilidad en el desarrollo de la aplicación.

```
val appModule = module {
    viewModel { WelcomeScreenViewModel() }
    viewModel { HomeScreenViewModel() }
    viewModel { LoginScreenViewModel() }
    viewModel { RegisterScreenViewModel() }
    viewModel { AboutUsScreenViewModel() }
    viewModel { SuggestionScreenViewModel() }
    viewModel { CategoriesScreenViewModel() }
    viewModel { ProfileScreenViewModel() }
    viewModel { SettingsScreenViewModel() }
    viewModel { MyListsScreenViewModel() }
    viewModel { CustomerSupportScreenViewModel() }
    viewModel { ProductsScreenViewModel() }
    viewModel { ProductDetailsScreenViewModel() }
    viewModel { SharedViewModel() }
    viewModel { ListScreenViewModel() }
}
```

← Declaración de los ViewModel en el módulo de Koin.

```
startKoin {
    androidContext(this@MainActivity)
    modules(appModule)
}
```

↑ Iniciar Koin con su modulo en el MainActivity

## 6.4 SEGURIDAD

### 6.4.1 FIREBASE AUTH OAUTH 2.0

Firebase utiliza el protocolo OAuth 2.0, un estándar de la industria para la autorización segura, que permite a las aplicaciones obtener acceso a los recursos del usuario sin exponer sus credenciales. Cuando un usuario inicia sesión con Google a través de Firebase, es redirigido a la página de inicio de sesión de Google, donde ingresa sus credenciales de manera segura. Firebase nunca maneja directamente la contraseña del usuario.

Una vez autenticado, Google proporciona a Firebase un token de acceso seguro. Este token permite que Firebase identifique al usuario sin comprometer la seguridad de sus credenciales. Además de Google, Firebase admite otros métodos de autenticación basados en OAuth 2.0, como Facebook, Twitter y GitHub, ofreciendo una experiencia de usuario segura y fluida. Utilizando Firebase y OAuth 2.0, se asegura una gestión confiable y segura de la autenticación y autorización de usuarios.

## 7 TESTS

### 7.1 RUTAS (END-POINTS)

Para comprobar las rutas se ha utilizado la extensión de Visual Studio Code:

- Thunder-Client: <https://www.thunderclient.com>

The screenshot shows the Thunder Client extension in Visual Studio Code. On the left, a sidebar lists recent activities, including several GET requests to the domain `wewiza.ddns.net`. In the center, a request details pane is open for a GET request to `https://wewiza.ddns.net/like`. The status bar indicates `Status: 200 OK`, `Size: 1.57 MB`, and `Time: 15.74 s`. The response body is displayed as a JSON object:

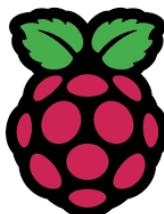
```
1 [  
2 {  
3   "uuid": "fe145dc6-6cd0-4799-82eb  
-6c8b33c0d212",  
4   "category_id": "postres_y_yogures",  
5   "name": "Bifidus desnatado con nueces y  
cereales Hacendado 0% m.g.",  
6   "price": 1.47,  
7   "quantity_measure": 500.0,  
8   "measure": "g",  
9   "price_by_standard_measure": 2.6,  
10  "image_url": "https://prod-mercadona.in  
.net/images  
/a33f37143476a71a09c5e90074d44ed4.j  
pg?fit=crop&h=300&w=300",  
11  "url": "https://tienda.mercadona.es"
```

## 8 IMPLANTACIÓN/DESPLIEGUE

### 8.1 ALOJAMIENTO SERVIDOR

Para el alojamiento del “backend”, se optó por utilizar una Raspberry Pi 3 como servidor principal.

Está ejecutando una distribución Debian como sistema operativo, lo que proporciona un entorno estable y bien soportado para mi aplicación



#### Raspberry Pi OS (Legacy, 32-bit)

A port of Debian Bullseye with security updates and desktop environment (Recommended)

Publicado: 2024-03-12

- **Servidor de Aplicaciones:** Se configuró “Uvicorn” como servidor de aplicaciones para ejecutar el backend.

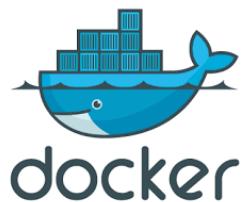
Uvicorn es un servidor ASGI (Asynchronous Server Gateway Interface) que ofrece una ejecución asíncrona y eficiente para aplicaciones Python.

- **Base de Datos:** se implementó una base de datos MongoDB acorde a la arquitectura ARM del procesador de la Raspberry Pi 3, mediante la imagen de Docker: “apcheamitru/arm32v7-mongo”.
- **UFW:** facilitó la seguridad para disponer de los puertos abiertos deseados y configurados.
- **SSH:** nos facilitó la conexión con el servidor de forma remota.
- **DNS:** se registró la <IP Pública> facilitada por la compañía proveedora, al dominio: **wewiza.ddns.net**

## 8.2 DOCKER

Mediante el despliegue en contenedores permitimos que nuestro proyecto adquiera:

- **Portabilidad:** encapsulación de la aplicación y sus dependencias.
- **Consistencia:** garantizamos que se ejecute en el mismo entorno.
- **Aislamiento:** evitamos conflicto con otras aplicaciones y seguridad.
- **Escalabilidad:** creación, eliminación y funcionalidades específicas de contenedores ligeros.
- **Automatización:** mediante la orquestación de contenedores podemos crear un flujo de automatización.



### 8.2.1 DOCKERFILE BACKEND

- Api-Markets:

```
FROM python:3.11-bullseye

WORKDIR /market_01

COPY ./api_market_01/requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 8081

COPY . .

CMD [ "python", "./api_market_01/main.py" ]
```

- Api-Wewiza:

```
FROM python:3.11-bullseye

WORKDIR /wewiza

COPY ./api_wewiza/requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 8080

COPY . .

CMD [ "python", "./api_wewiza/main.py" ]
```



## 8.2.2 DOKER-COMPOSE

Mediante la orquestación de contenedores facilitamos independencia de servicios y automatizamos tareas.

- Api-Markets:

Cada market dispondrá de una red privada para que la única forma de llegar a los datos del contenedor MongoDB sea de manera segura mediante la API del market correspondiente.

Se compartirá una red pública con la API-Wewiza para que las API puedan entre ellas establecer conexión y solicitudes.

```
version: "3"
services:
  api_market_01:
    build: ./api-market-01
    restart: always
    ports:
      - "8081:8081"
    environment:
      - PORT=8081
    command: uvicorn api_market_01.main:app --host 0.0.0.0 --port 8081 --reload
    depends_on:
      - mongo_market_01
    networks:
      - public-wewiza-network
      - private-market-01-network

  mongo_market_01:
    image: apcheamitru/arm32v7-mongo:3.2.20
    restart: always
    ports:
      - "27021:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: root
      MONGO_INITDB_DATABASE: market_01_db
    volumes:
      - mongo_database:/data/db # Directorio de datos de MongoDB
    networks:
      - private-market-01-network

  volumes:
    mongo_database:

networks:
  public-wewiza-network:
    external: true
  private-market-01-network:
    driver: bridge
```



- Api-Wewiza:

Se compartirá una red pública con las API-Market para que las API puedan entre ellas establecer conexión y solicitudes.

```
version: "3"
services:
  api_wewiza:
    build: ./api_wewiza
    restart: always
    ports:
      - "443:443"
      - "587:587" # SMTP for email
    environment:
      - PORT=443
    volumes:
      - ./certs/ssl:/certs/ssl
    networks:
      - public-wewiza-network
      - private-wewiza-network
    command: uvicorn api_wewiza.main:app --host 0.0.0.0 --port 443 --ssl-keyfile=/certs/ssl/server.key --ssl-certfile=/certs/ssl/server.cer --reload
    depends_on:
      - mongo_wewiza

  mongo_wewiza:
    # apcheamitru/arm32v7-mongo:3.2.20 # mongo:6.0.13
    image: apcheamitru/arm32v7-mongo:3.2.20
    restart: always
    ports:
      - "27020:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: root
      MONGO_INITDB_DATABASE: wewiza_db
    volumes:
      - mongo_database:/data/db # Directorio de datos de MongoDB
    networks:
      - private-wewiza-network

  volumes:
    mongo_database:

networks:
  public-wewiza-network:
    external: true
  private-wewiza-network:
    driver: bridge
```



## 8.3 ANDROID

Hemos decidido utilizar Android Studio para el desarrollo de nuestra aplicación. Android Studio, como el entorno de desarrollo integrado (IDE) oficial de Google para Android, ofrece una amplia gama de herramientas y funcionalidades avanzadas que optimizan y facilitan el proceso de desarrollo. Su integración perfecta con las bibliotecas y servicios de Google, junto con su soporte para múltiples configuraciones de dispositivos y emuladores, garantiza un flujo de trabajo eficiente y productivo. Además, Android Studio facilita la construcción y el despliegue del APK, permitiendo un ciclo de desarrollo más ágil. Esta elección nos permitirá crear una aplicación robusta y de alta calidad, alineada con las mejores prácticas de desarrollo para la plataforma Android.



### 8.3.1 LIBRERÍAS.

Para poder cumplir con las tareas y los objetivos propuestos, ha sido necesario el uso de estas librerías durante el desarrollo de la aplicación. (solo se han recopilado las librerías adicionales añadidas al proyecto).

- **Firebase:** El uso de Firebase Auth y Firebase Storage ha sido necesario para el manejo y gestión de usuario, así como para la creación y/o modificación de perfiles de usuario. Adicionalmente también se han usado los servicios de Google para poder realizar el inicio de sesión con una cuenta de Google.
- **Retrofit:** Retrofit es una biblioteca de cliente HTTP para Android. Durante el proyecto se ha utilizado para consumir la API propia de Wewiza y realizar llamadas de manera asíncrona durante la ejecución de la aplicación. También se ha hecho uso del *Interceptor* para la monitorización del estado de las consultas, de manera que facilite la depuración del código.
- **Coil:** Coil Image Leader, es una biblioteca moderna de carga de imágenes para Android. Se ha utilizado para la carga de imágenes en formato .png y .svg.
- **MPAndroidCharts:** MPAndroidCharts es una biblioteca que permite la creación de diversos gráficos a aplicaciones Android. Se ha utilizado para la creación de graficas con el histórico de datos y mostrar gráficamente la evolución del precio de los productos del mercado.



## 9 CONCLUSIONES Y TRABAJO FUTURO

---

Respecto a la planificación del proyecto pudimos desarrollar todos los requisitos funcionales, no funcionales y de información, gracias a los distintos métodos ejecutados como realizar diagramas de Gantt, trabajar con metodologías ágiles y dispones de un entorno de control de versiones como Git.

Ha sido una experiencia totalmente gratificante aplicar el ciclo del desarrollo de software en un proyecto de más envergadura.

Para los puntos que hubiéramos gustado desarrollar serían los siguiente:

- Carrusel de tutorial sobre el funcionamiento de la aplicación.
- Crear un sistema de gamificación más robusto e innovador.
- Disponer de modo oscuro.
- Aplicar pruebas de software.
- Facilitar al usuario el uso de la aplicación sin tener que instalar el certificado del servidor.
- Mejorar la UI y UX de la aplicación.
- Mejorar las respuestas desde el servidor con una estructuración de un documento estándar para todas las respuestas, así facilitar el desarrollo al consumir los endpoints.

Finalmente, aunque los datos recogidos son meramente orientativos ya que los merados no actualizan constantemente sus productos, este proyecto queremos mantenerlo vivo y actualizado, para poderlo publicar y que cualquier desarrollador pueda hacer uso de la API Rest e implementar sus soluciones frontales o que los usuarios puedan instalarse la aplicación y facilitarle su día a día.

## 10 CÓDIGO FUENTE DEL PROYECTO

---



<https://github.com/agchivite/Wewiza/tree/main>



## 11 BIBLIOGRAFÍA

---

- <https://aprendepython.es/pypi/scraping/beautifulsoup/>
- <https://www.openssl.org/>
- <https://medium.com/@mariovanrooij/adding-https-to-fastapi-ad5e0f9e084e>
- <https://ubuntu.com/download/raspberry-pi>
- <https://help.skysilk.com/support/solutions/articles/9000182486--basic-forward-a-port-with-ufw-in-ubuntu-18-04-linux>
- <https://www.draytek.com/support/knowledge-base/5214>
- <https://www.noip.com/es-MX>
- <https://www.youtube.com/>
- <https://chatgpt.com/>
- <https://www.yougetsignal.com/tools/open-ports/>
- <https://changhsinlee.com/pytest-mock/>
- <https://www.linkedin.com/pulse/testing-python-using-pytest-mock-claudio-shigueo-watanabe/>
- <https://developer.chrome.com/docs/chromedriver/downloads?hl=es-419>
- <https://stackoverflow.com/>
- <https://developer.android.com/develop/ui/compose/documentation?hl=es-419>