**Proyecto - “WeWiza”: El Compañero de Compras Ideal**

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**

[1 Resumen 4](#_Toc168307401)

[1.1 Justificación del proyecto 4](#_Toc168307402)

[2 Objetivos 5](#_Toc168307403)

[2.1 Planificación Gantt 5](#_Toc168307404)

[2.2 Kanban + Scrum 6](#_Toc168307405)

[2.3 GitFlow 7](#_Toc168307406)

[3 Análisis/Elección Tecnológica 8](#_Toc168307407)

[3.1 Base de Datos 8](#_Toc168307408)

[3.1.1 Bases de Datos no elegidas 8](#_Toc168307409)

[3.1.2 Base de datos elegida (MongoDB) 9](#_Toc168307410)

[3.2 Creación API (Back-end) 10](#_Toc168307411)

[3.2.1 APIs no elegidas 10](#_Toc168307412)

[3.2.2 API elegida (FastAPI) 11](#_Toc168307413)

[3.3 Consumición API (Front-end) 11](#_Toc168307414)

[3.3.1 APIs no elegidas 11](#_Toc168307415)

[3.3.2 API elegida (Retrofit) 12](#_Toc168307416)

[3.4 Interfaz Android 12](#_Toc168307417)

[3.4.1 Interfaces no elegidas 12](#_Toc168307418)

[3.4.2 Interfaz elegida (Jetpack Compose) 13](#_Toc168307419)

[3.5 Competencia 14](#_Toc168307420)

[3.6 Requisitos 15](#_Toc168307421)

[3.6.1 Requisitos Funcionales 15](#_Toc168307422)

[3.6.2 Requisitos No Funcionales 15](#_Toc168307423)

[3.6.3 Requisitos de Información 15](#_Toc168307424)

[4 Diseño 16](#_Toc168307425)

[4.1 Diagrama de Clases y Modelo de Datos 16](#_Toc168307426)

[4.1.1 Backend 16](#_Toc168307427)

[4.1.2 Frontend 17](#_Toc168307428)

[4.2 Diagrama Entidad-Relación (Base de Datos) 18](#_Toc168307429)

[4.2.1 Backend MongoDB: 18](#_Toc168307430)

[4.2.2 Frontend Firebase: 18](#_Toc168307431)

[4.3 Mockup 19](#_Toc168307432)

[4.4 Tipos de fuentes de texto 19](#_Toc168307433)

[4.5 Diagrama de vistas 20](#_Toc168307434)

[4.6 Paleta de Color 20](#_Toc168307435)

[5 Desarrollo Back-end 21](#_Toc168307436)

[5.1 Planteamiento 21](#_Toc168307437)

[5.2 DatabaseManager 22](#_Toc168307438)

[5.3 Repositorios 23](#_Toc168307439)

[5.3.1 Markets 23](#_Toc168307440)

[5.3.2 Wewiza 23](#_Toc168307441)

[5.4 Servicios 24](#_Toc168307442)

[5.4.1 Procesamiento de productos 24](#_Toc168307443)

[5.4.2 Scrapping y Requisitos 25](#_Toc168307444)

[5.4.3 Cálculo diario 25](#_Toc168307445)

[5.5 Endpoints 26](#_Toc168307446)

[5.5.1 Root 26](#_Toc168307447)

[5.5.2 Categorías Top 26](#_Toc168307448)

[5.5.3 Productos Top 27](#_Toc168307449)

[5.5.4 Categorías 28](#_Toc168307450)

[5.5.5 Productos por categoría 28](#_Toc168307451)

[5.5.6 Producto por UUID 29](#_Toc168307452)

[5.5.7 Número de productos por mercado 29](#_Toc168307453)

[5.5.8 Producto con histórico por UUID 30](#_Toc168307454)

[5.5.9 Productos por mercado 31](#_Toc168307455)

[5.5.10 Productos por rango y mercado 31](#_Toc168307456)

[5.5.11 Producto (me gusta) 32](#_Toc168307457)

[5.5.12 Producto (no me gusta) 32](#_Toc168307458)

[5.5.13 Estado de la reacción de un producto por usuario 32](#_Toc168307459)

[5.5.14 Inicio mensual 32](#_Toc168307460)

[5.5.15 Sugerencias por producto 33](#_Toc168307461)

[5.6 Railway Oriented Programming 34](#_Toc168307462)

[5.7 Seguridad 35](#_Toc168307463)

[5.7.1 SSL 35](#_Toc168307464)

[5.8 Dependencias 35](#_Toc168307465)

[5.8.1 Servidor 35](#_Toc168307466)

[5.8.2 Scrapping 35](#_Toc168307467)

[5.8.3 Testing 35](#_Toc168307468)

[6 Desarrollo Front-end 36](#_Toc168307469)

[6.1 Interfaz 36](#_Toc168307470)

[6.1.1 Introducción 36](#_Toc168307471)

[6.1.2 Pantallas y composables 36](#_Toc168307473)

[6.2 Firebase 40](#_Toc168307483)

[6.2.1 Usuarios 40](#_Toc168307484)

[6.2.2 Gamificación 40](#_Toc168307485)

[6.3 Inversión de control/Inyector de dependencias 40](#_Toc168307486)

[6.4 Seguridad 40](#_Toc168307487)

[6.4.1 Firebase Auth OAuth 2.0 40](#_Toc168307488)

[7 Tests 41](#_Toc168307489)

[7.1 Rutas (End-Points) 41](#_Toc168307490)

[8 Implantación/Despliegue 42](#_Toc168307491)

[8.1 Alojamiento Servidor 42](#_Toc168307492)

[8.2 Docker 43](#_Toc168307493)

[8.2.1 Dockerfile Backend 43](#_Toc168307494)

[8.2.2 Doker-Compose 44](#_Toc168307495)

[8.3 Android 46](#_Toc168307496)

[9 Conclusiones y trabajo futuro 47](#_Toc168307497)

[10 Bibliografía 48](#_Toc168307498)

1. Resumen
   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.

1. 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.
  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”.

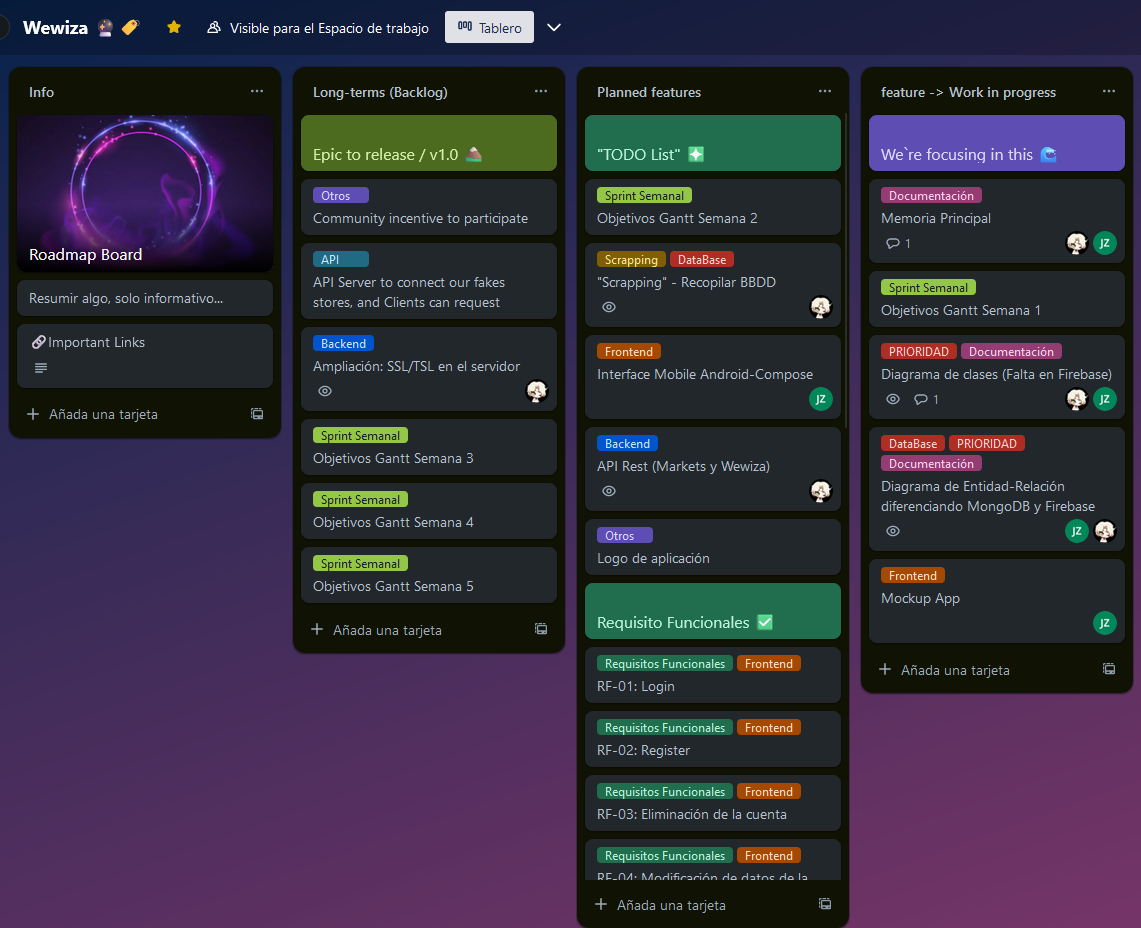
Escala de tiempo

Descripción generada automáticamente con confianza baja

* 1. 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 produción.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* 1. 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”: trabajado realizado con el sumatorio de “features”, en espera de ser verificado por el equipo.
  + “Docs”: documentación del proyecto
  + “Hotfix”: errores o “bugs” puntuales que requiren de un arreglo inmediato.
  + “Main”: trabajo verificado y dispuesto para lanzar a produción.

Diagrama

Descripción generada automáticamente

1. Análisis/Elección Tecnológica
   1. Base de Datos
      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.
    1. Base de datos elegida (MongoDB)

Logotipo, nombre de la empresa

Descripción generada automáticamente

* + - 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.
  1. Creación API (Back-end)
     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.
    1. 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.
  1. Consumición API (Front-end)
     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 asincrónico.
      * Menos recursos de la comunidad y documentación.
    1. API elegida (Retrofit)

Imagen que contiene Texto

Descripción generada automáticamente

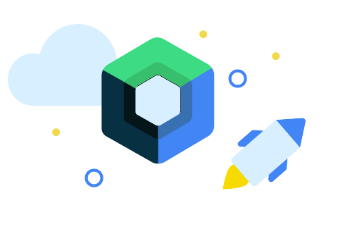
* + - Ventajas:
      * Fácil de usar, interfaz simple y declarativa.
      * Se integra fácilmente con bibliotecas de serialización JSON.
      * Comunidad de usuarios y una documentación extensa.
    - Desventajas:
      * Es sincrónico por defecto, lo que significa que las solicitudes se realizan 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
  1. Interfaz Android
     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
    1. 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.
  1. Competencia

**SpriceSpy**, I**dealo** 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.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteTexto

Descripción generada automáticamenteCaptura de pantalla de un celular

Descripción generada automáticamenteImagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamenteLogotipo

Descripción generada automáticamenteTexto

Descripción generada automáticamente

* 1. Requisitos
     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
  + 1. 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.
  + 1. 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.

1. Diseño
   1. Diagrama de Clases y Modelo de Datos
      1. Backend

* **CategoriaBackMarket** 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.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

* + 1. 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.

Interfaz de usuario gráfica

Descripción generada automáticamente

* 1. Diagrama Entidad-Relación (Base de Datos)
     1. Backend MongoDB:
* **GananciaPorcentaje** y **Ganancia** son campos que aparecen en endpoints particulares.

Interfaz de usuario gráfica, Texto

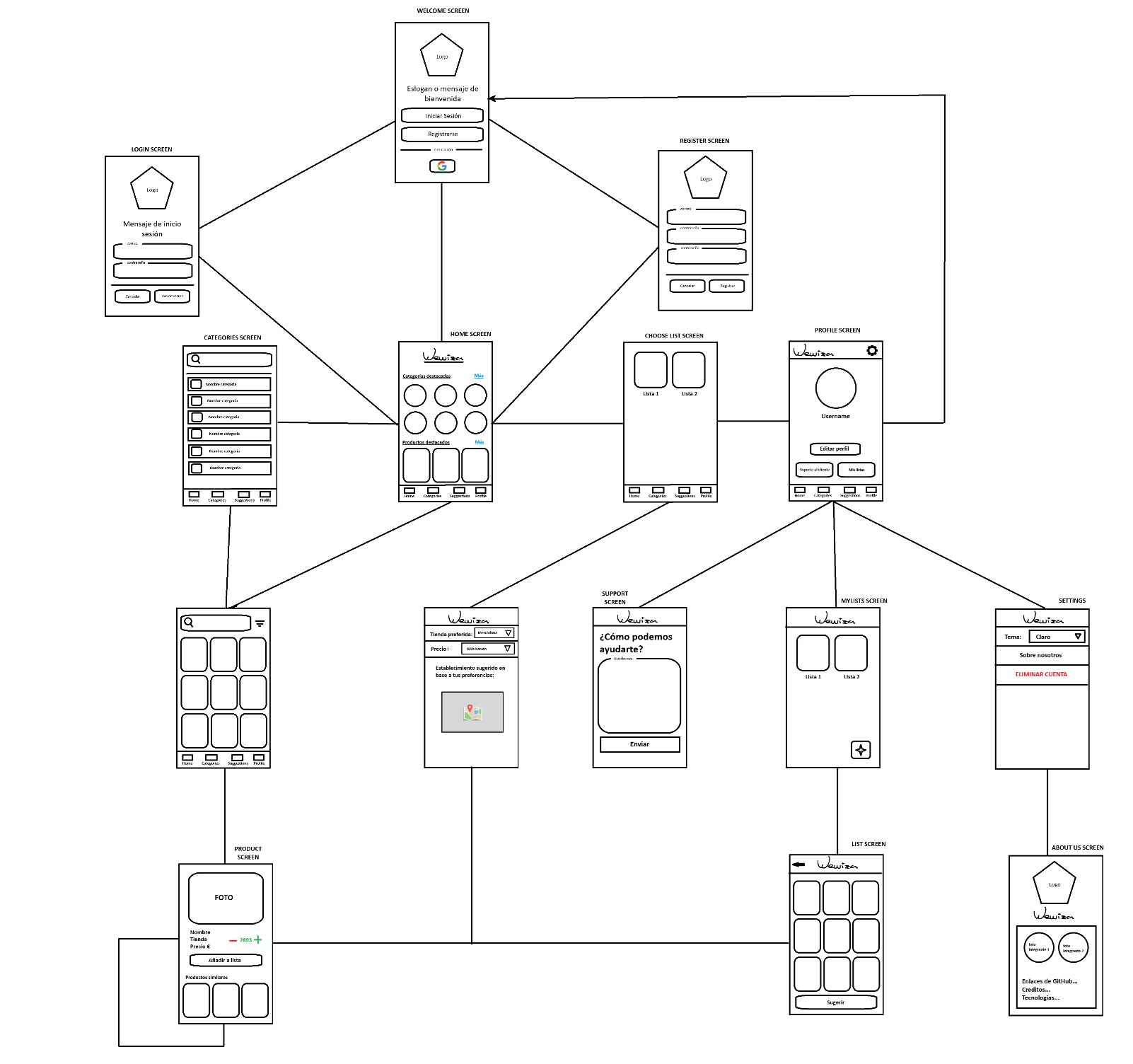
Descripción generada automáticamente

* + 1. Frontend Firebase:

Imagen que contiene Texto

Descripción generada automáticamente

* 1. Mockup



* 1. 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
  1. Diagrama de vistas

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

Diagrama

Descripción generada automáticamente

* 1. Forma

     Descripción generada automáticamentePaleta de Color

Gráfico

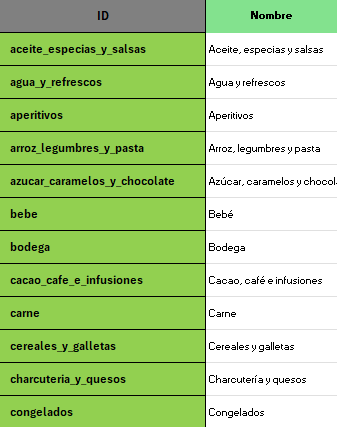
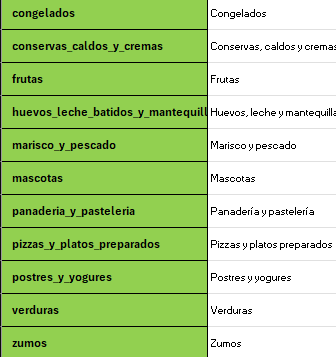
Descripción generada automáticamente

1. Desarrollo Back-end
   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:



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.

* 1. 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.

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:

Texto

Descripción generada automáticamente

* 1. Repositorios

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

* + 1. Markets

Texto

Descripción generada automáticamente

* + 1. Wewiza

Interfaz de usuario gráfica

Descripción generada automáticamente

* 1. Servicios
     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:

Texto

Descripción generada automáticamente

* + 1. 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.

Texto

Descripción generada automáticamenteTodos los mercados tienen estandarizada la implementación **de la FACHADA**:

* Se ha utilizado la **librería** de Python ([BeautifulSoap](https://aprendepython.es/pypi/scraping/beautifulsoup/)) para recolectar los datos públicos de las páginas web de los distintos mercados.
* Requiere de un dispositivo que admita ejecutar [Google Chrome](https://www.google.com/intl/es_es/chrome/) ya que necesitamos descargar y ejecutar [chromedriver](https://developer.chrome.com/docs/chromedriver/downloads?hl=es-419) mediante el servicio utilizando la **librería** [Selenium](https://selenium-python.readthedocs.io/).
* 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.
  + 1. Cálculo diario

Diariamente 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](https://pypi.org/project/APScheduler/3.10.0/) nos facilita la ejecución de funciones agendadas:

Texto

Descripción generada automáticamente

* 1. 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.

* + 1. Root

<https://wewiza.ddns.net>

Texto

Descripción generada automáticamenteInformamos sobre la disposición de la **documentación** facilitada por FastAPI:

* + 1. Categorías Top

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

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

Texto

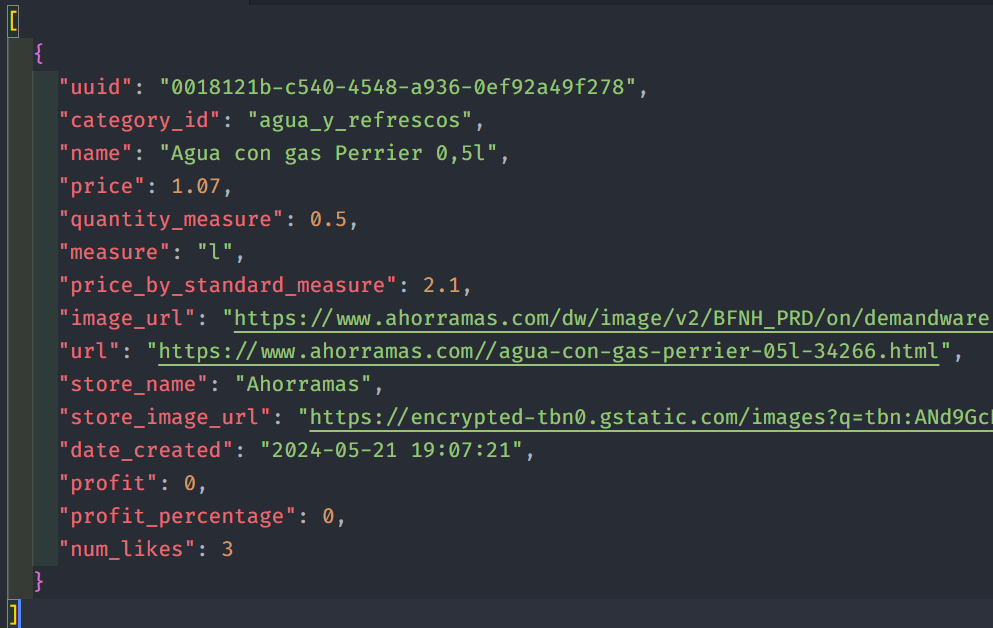
Descripción generada automáticamente

* + 1. 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:



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

Texto

Descripción generada automáticamente

* + 1. Categorías

https://wewiza.ddns.net/categories

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

Texto

Descripción generada automáticamente

* + 1. Productos por categoría

https://wewiza.ddns.net[/products/category/id/{category\_id}](https://wewiza.ddns.net/docs#/default/get_products_by_category_products_category_id__category_id__get)

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

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteTexto

Descripción generada automáticamente

* + 1. Producto por UUID

https://wewiza.ddns.net[/product/id/{uuid}](https://wewiza.ddns.net/docs#/default/get_product_by_id_product_id__uuid__get)

Texto

Descripción generada automáticamente Devolvemos el modelo producto encontrado:

Imagen que contiene Texto

Descripción generada automáticamenteEn caso de no encontrarlo devolvemos un documento básico con información:

* + 1. Número de productos por mercado

https://wewiza.ddns.net/size/{market\_name}

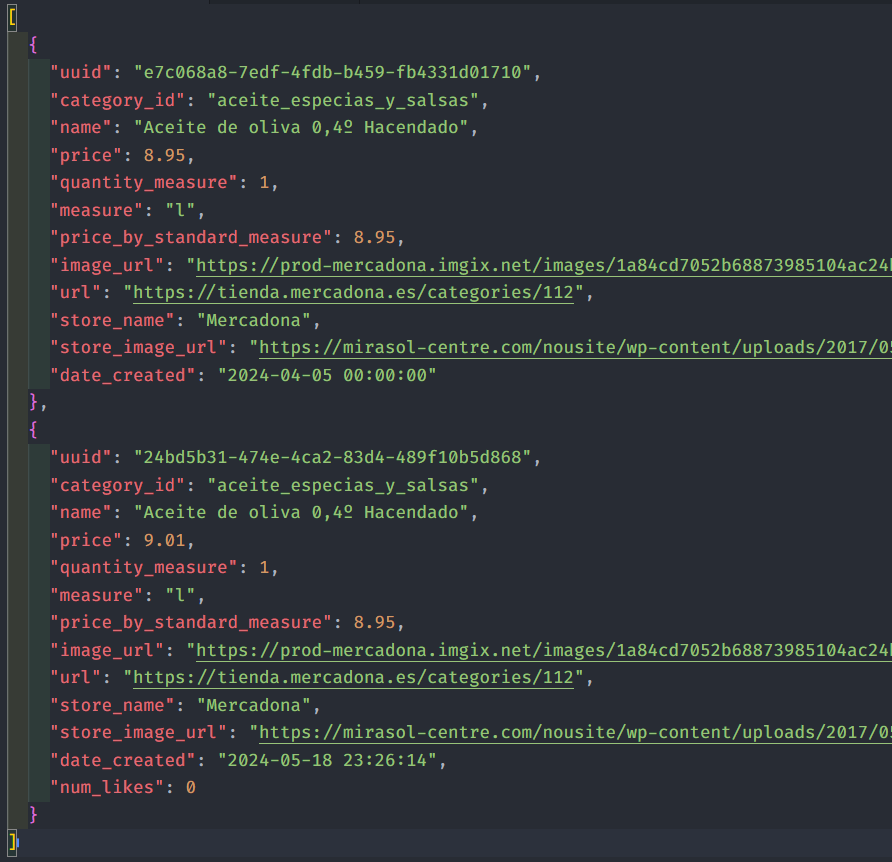
Devolvemos **un número entero** con los productos que disponga el mercado dentro del **marco mensual** en el que hemos realizado el “scrapping”:

Texto

Descripción generada automáticamente

* + 1. Producto con histórico por UUID

https://wewiza.ddns.net[/product/details/id/{uuid}](https://wewiza.ddns.net/docs#/default/get_product_by_id_product_id__uuid__get)

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

* + 1. Productos por mercado

https://wewiza.ddns.net[/products/market/{market\_name}](https://wewiza.ddns.net/docs#/default/get_all_products_by_market_products_market__market_name__get)

Texto

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

* + 1. Productos por rango y mercado

https://wewiza.ddns.net[/products/market/{market\_name}/range/{init\_num}/{end\_num}](https://wewiza.ddns.net/docs#/default/get_products_by_range_products_market__market_name__range__init_num___end_num__get)

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.

Texto

Descripción generada automáticamente

* + 1. Producto (me gusta)

https://wewiza.ddns.net[/like/{uuid}/email/{email\_user}](https://wewiza.ddns.net/docs#/default/like_product_like__uuid__email__email_user__get)

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.

* + 1. Producto (no me gusta)

https://wewiza.ddns.net[/unlike/{uuid}/email/{email\_user}](https://wewiza.ddns.net/docs#/default/like_product_like__uuid__email__email_user__get)

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.

* + 1. Estado de la reacción de un producto por usuario

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

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.
  + 1. Inicio mensual

[https://wewiza.ddns.net/start\_month/password/{password}](https://wewiza.ddns.net/start_month/password/%7bpassword%7d)

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.

* + 1. Sugerencias por producto

https://wewiza.ddns.net[/suggest/id/{uuid}](https://wewiza.ddns.net/docs#/default/get_suggest_products_suggest_id__uuid__get)?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:

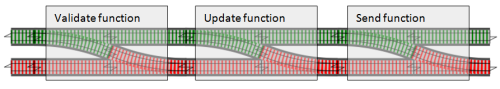
* **LEMATIZACIÓN:** mediante esta técnica podemos encontrar palabras que dan la misma información, pero de distinta forma presentada, ejemplo: limones -> limón.
* **REGEX:** mediante las palabras encontradas con la lematización, se formula un regex para buscar aquellos productos que correspondan también 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).\*”

Texto

Descripción generada automáticamenteProductos que sugiere wewiza más baratos y similares al que ha preguntado el usuario:

* 1. 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:

Texto

Descripción generada automáticamente

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

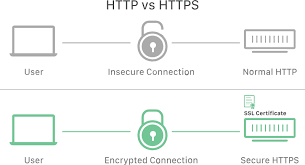
Texto

Descripción generada automáticamente

* 1. Seguridad
     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](https://www.openssl.org/).

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.

* 1. Dependencias
     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/>
  + 1. Scrapping
* Selenium (4.10.0): <https://pypi.org/project/selenium/4.10.0/>
* BeatifulSoup4 (4.10.0): <https://pypi.org/project/beautifulsoup4/4.10.0/>
  + 1. Testing
* Pytest (6.2.5): <https://pypi.org/project/pytest/6.2.5/>
* Pytest-mock (3.6.1): <https://pypi.org/project/pytest-mock/3.6.1/>

1. Desarrollo Front-end
   1. Interfaz

Para la creación de la interfaz gráfica de usuario (GUI), se ha optado por usar 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.

* + 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.
  + 2. Pantallas y composables





#### Pantalla de bienvenida

Es una pantalla donde el usuario puede elegir registrarse con Google, registrase 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.

Texto

Descripción generada automáticamente

Ejemplo de código



#### Pantalla de inicio de sesión

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

Texto

Descripción generada automáticamente

Ejemplo de código.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

#### Pantalla de registro

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

Texto

Descripción generada automáticamente

Ejemplo de código

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

#### 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.

Texto

Descripción generada automáticamente

Ejemplo de código: Muestra la estructura de la vista.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

#### 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.

Texto

Descripción generada automáticamente

Ejemplo de código: Componente personalizado de categoría.



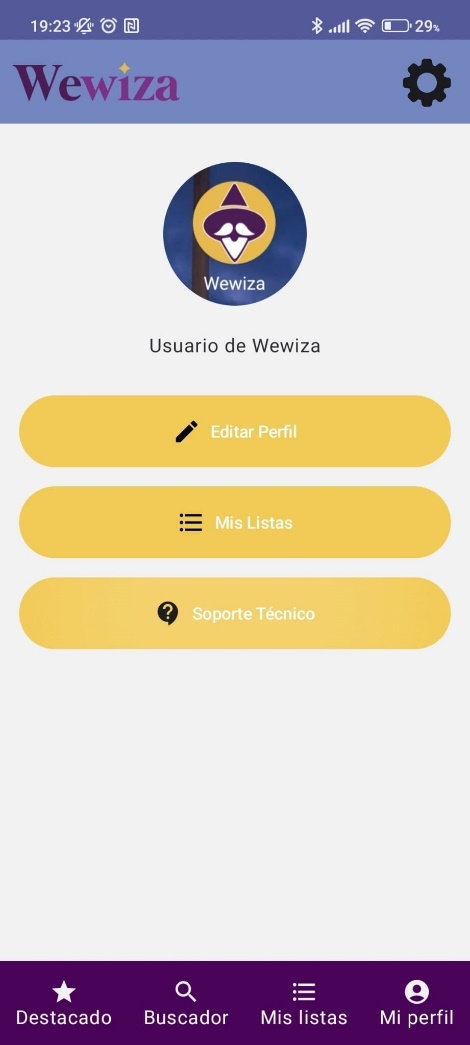
#### Pantalla de listas de usuario

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 del la franquicia de supermercados que el usuario elija.

Texto

Descripción generada automáticamente

Ejemplo de código.



#### 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.

Texto

Descripción generada automáticamente

Ejemplo de código: Imagen de perfil.

* 1. Firebase
     1. Usuarios

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

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**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 decido 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 decido 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.

* 1. 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.

Texto

Descripción generada automáticamente

← *Declaración de los ViewModel en el módulo de Koin.*

Texto

Descripción generada automáticamente

*↑ Iniciar Koin con su modulo en el*

* 1. Seguridad
     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.

1. Tests
   1. Rutas (End-Points)

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

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

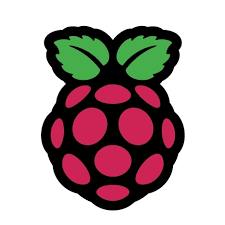
Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

1. Implantación/Despliegue
   1. Alojamiento Servidor

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

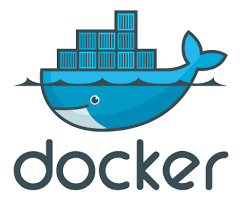
Está ejecutando una distribución Debian como sistema operativo, lo que proporciona un entorno estable y bien soportado para mi aplicación



* **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 asincrónica y eficiente para aplicaciones Python.

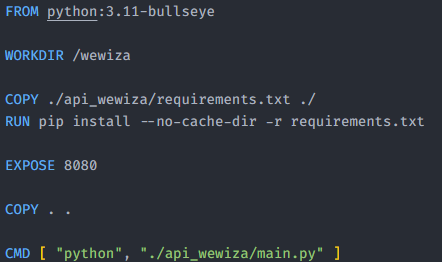
* **Base de Datos:** se implementó una base de datos MongoDB acorde a la arquitectura ARM del procesador de la Raspberry Pi 3, mediante la imagen de Docker: “apcheamitru/arm32v7-mongo”.
* **UFW:** facilitó la seguridad para disponer de los puertos abiertos deseados y configurados.
* **SSH**: nos facilitó la conexión con el servidor de forma remota.
* **DNS:** se registró la <IP Pública> facilitada por la compañía proveedora, al dominio: **wewiza.ddns.net**
  1. 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.
  + 1. Dockerfile Backend
* Texto

  Descripción generada automáticamenteApi-Markets:

* Api-Wewiza:



* + 1. 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.

Texto

Descripción generada automáticamente

* Api-Wewiza:

Texto

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

* 1. Android

Icono

Descripción generada automáticamenteHemos 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.

1. Conclusiones y trabajo futuro

Respecto a la planificación del proyecto pudimos desarrollar todos los requisitos funciones, 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.

1. 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>