**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](#_Toc162299865)

[1.1 Justificación del proyecto 4](#_Toc162299866)

[2 Objetivos 5](#_Toc162299867)

[2.1 Planificación Gantt 5](#_Toc162299868)

[2.2 Kanban + Scrum 6](#_Toc162299869)

[2.3 GitFlow 7](#_Toc162299870)

[3 Análisis/Elección Tecnológica 8](#_Toc162299871)

[3.1 Base de Datos 8](#_Toc162299872)

[3.1.1 PostgreSQL 8](#_Toc162299873)

[3.1.2 MySQL 8](#_Toc162299874)

[3.1.3 MariaDB 9](#_Toc162299875)

[3.1.4 MongoDB 9](#_Toc162299876)

[3.2 Creación API (Back-end) 10](#_Toc162299877)

[3.2.1 Django 10](#_Toc162299878)

[3.2.2 Flask 10](#_Toc162299879)

[3.2.3 FastAPI 11](#_Toc162299880)

[3.3 Consumición API (Front-end) 11](#_Toc162299881)

[3.3.1 Retrofit 11](#_Toc162299882)

[3.3.2 Ktor 12](#_Toc162299883)

[3.4 Interfaz Android 12](#_Toc162299884)

[3.4.1 Flutter 12](#_Toc162299885)

[3.4.2 XML 13](#_Toc162299886)

[3.4.3 Jetpack Compose 13](#_Toc162299887)

[3.5 Competencia 14](#_Toc162299888)

[3.6 Requisitos 15](#_Toc162299889)

[3.6.1 Requisitos Funcionales 15](#_Toc162299890)

[3.6.2 Requisitos No Funcionales 15](#_Toc162299891)

[3.6.3 Requisitos de Información 15](#_Toc162299892)

[4 Diseño 16](#_Toc162299893)

[4.1 Diagrama de Clases y Modelo de Datos 16](#_Toc162299894)

[4.1.1 Backend 16](#_Toc162299895)

[4.1.2 Frontend 17](#_Toc162299896)

[4.2 Diagrama Entidad-Relación (Base de Datos) 18](#_Toc162299897)

[4.2.1 Backend MongoDB: 18](#_Toc162299898)

[4.2.2 Frontend Firebase: 19](#_Toc162299899)

[4.3 Mockup 20](#_Toc162299900)

[4.4 Tipos de fuentes de texto 20](#_Toc162299901)

[4.5 Diagrama de vistas 21](#_Toc162299902)

[4.6 Paleta de Color 21](#_Toc162299903)

[5 Desarrollo Back-end 22](#_Toc162299904)

[5.1 Planteamiento 22](#_Toc162299905)

[5.2 DatabaseManager 23](#_Toc162299906)

[5.3 Repositorios 24](#_Toc162299907)

[5.3.1 ProductosMarkets 24](#_Toc162299908)

[5.3.2 ProductosWewiza 24](#_Toc162299909)

[5.4 Servicios 25](#_Toc162299910)

[5.4.1 Scrapping 25](#_Toc162299911)

[5.4.2 Procesamiento de productos 25](#_Toc162299912)

[5.5 Endpoints 26](#_Toc162299913)

[5.6 Implementación basada en eventos (Reactividad) 27](#_Toc162299914)

[5.7 Railway Oriented Programming 28](#_Toc162299915)

[5.8 Seguridad 29](#_Toc162299916)

[5.8.1 SSL 29](#_Toc162299917)

[5.8.2 JWT 29](#_Toc162299918)

[5.9 Inversión de control/Inyector de dependencias 30](#_Toc162299919)

[6 Desarrollo Front-end 31](#_Toc162299920)

[6.1 Interfaz 31](#_Toc162299921)

[6.2 Firebase 31](#_Toc162299922)

[6.2.1 Usuarios 31](#_Toc162299923)

[6.2.2 Gamificación 31](#_Toc162299924)

[6.3 Inversión de control/Inyector de dependencias 31](#_Toc162299925)

[7 Tests 32](#_Toc162299926)

[7.1 Pruebas Unitarias e Integración 32](#_Toc162299927)

[7.2 Rutas (End-Points) 32](#_Toc162299928)

[8 Implantación/Despliegue 33](#_Toc162299929)

[8.1 Alojamiento Servidor 33](#_Toc162299930)

[8.2 Docker 34](#_Toc162299931)

[8.2.1 Dockerfile Backend 34](#_Toc162299932)

[8.2.2 Doker-Compose 35](#_Toc162299933)

[8.3 Android 37](#_Toc162299934)

[9 Conclusiones 38](#_Toc162299935)

[10 Bibliografía 39](#_Toc162299936)

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.

La aplicación que se propone 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.

Forma

Descripción generada automáticamente

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.

Este proceso de verificación busca garantizar el correcto funcionamiento de la aplicación y fortalecer la confianza del usuario en su uso, de ahí adquirimos el nombre “Wewiza” siendo un acrónimo rebuscado del inglés “We are wizards”,

es decir, “Somos magos” poniendo hincapié en que somos una comunidad, tanto los usuarios aportando la validación del producto como de los desarrolladores facilitando un servicio competente.

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”.

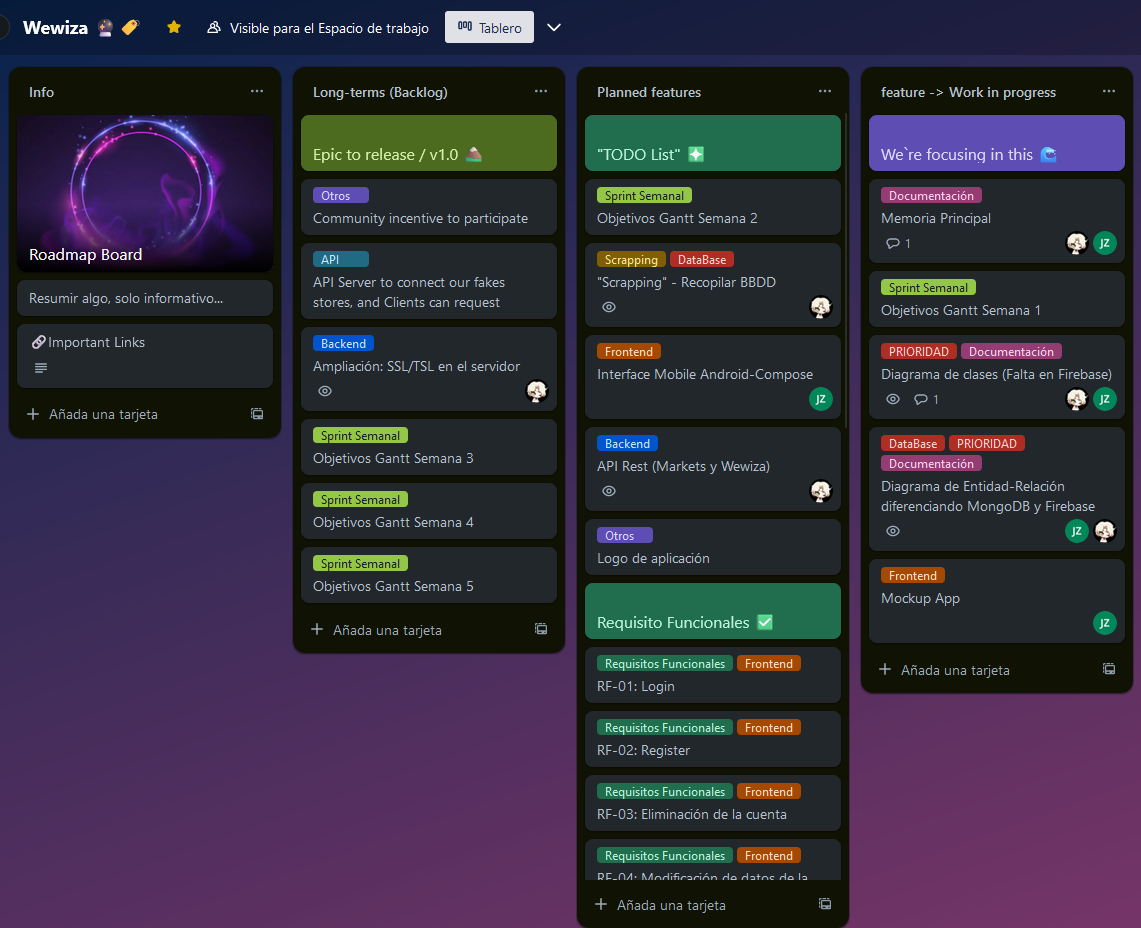
Escala de tiempo

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. PostgreSQL
      * 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.



* + 1. MySQL
    - 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.



* + 1. MariaDB
    - 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.

Logotipo, nombre de la empresa

Descripción generada automáticamente

* + 1. 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.
  1. Creación API (Back-end)
     1. Django
     + 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.
     1.  Flask
     + 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.



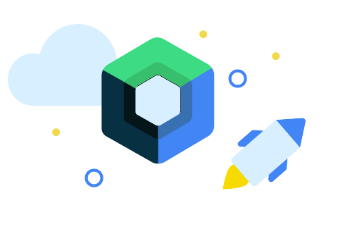
* + 1. FastAPI
    - Ventajas:
      * Alto rendimiento y eficiencia gracias a su integración con Pydantic, lo que permite la generación de código altamente optimizado.
      * Fácil creación de API RESTful con una sintaxis declarativa y tipado estático.
      * Integración sencilla con herramientas 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. 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



* + 1. Ktor
    - 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. Interfaz Android
     1. Flutter
     + 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.



* + 1. XML
    - 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. 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

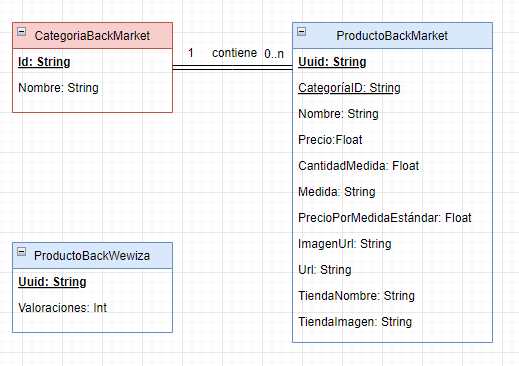
* 1. Requisitos
     1. Requisitos Funcionales
* RF-01: Acceso a la aplicación “Log in”.
* RF-02: Registro a la aplicación.
* RF-03: Eliminación de la cuenta.
* RF-04: Modificación de datos de la cuenta.
* RF-05: Buscador de productos.
* RF-06: Agregar productos a lista de la compra del usuario.
* RF-07: 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 donde haya una diferencia de precios importante en comparación con otros mercados.
* RF-14: Posibilidad de contactar con los desarrolladores.
* RF-15: Modo claro y oscuro.
  + 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.
* RNF-06: Seguridad de API REST mediante JWT.
  + 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

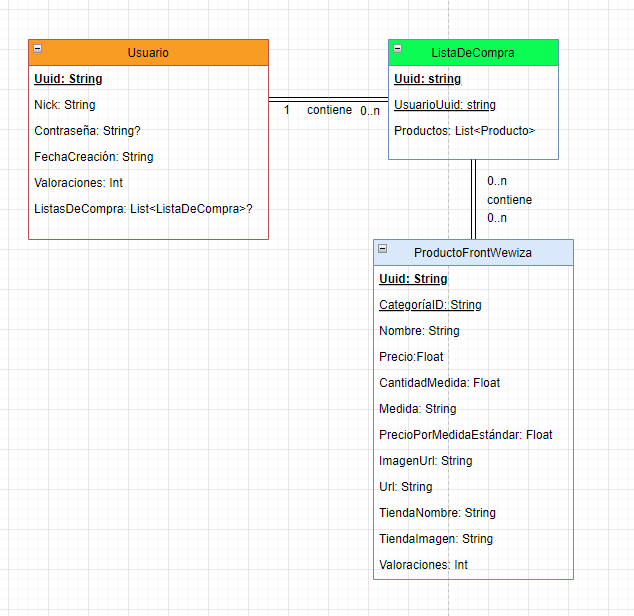
* **CategoriaBackMarket** y **ProductoBackMarket**: corresponden a las API-market, es decir el “backend” de cada mercado.

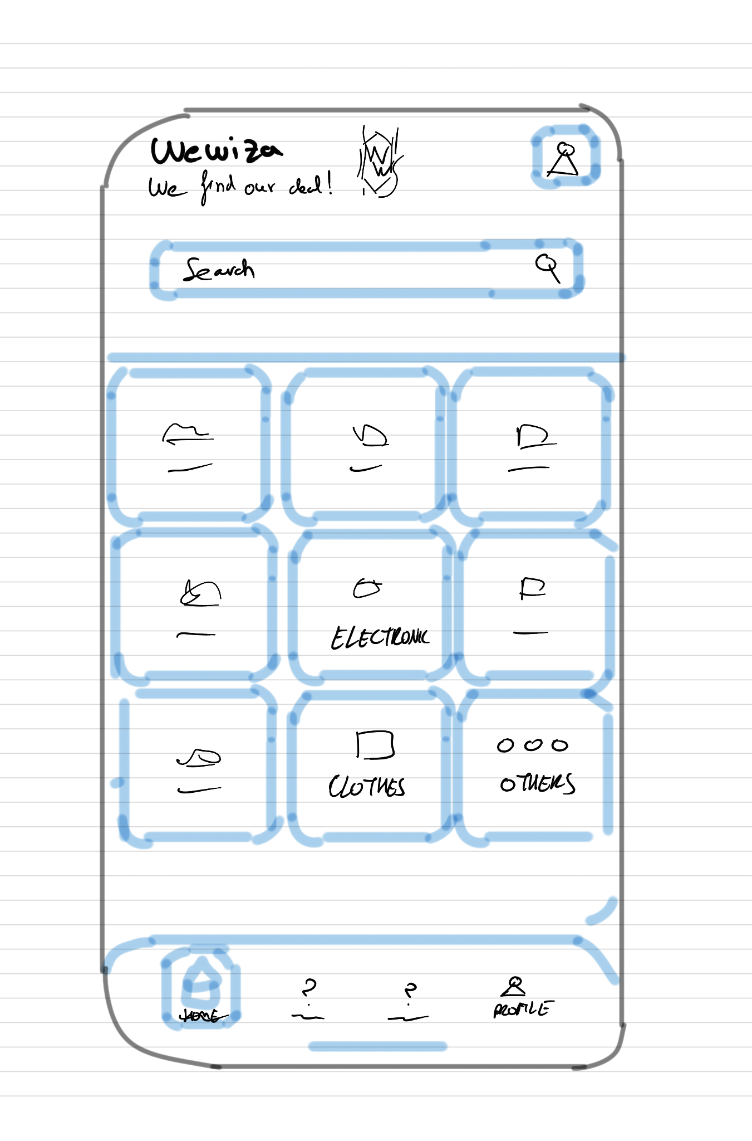
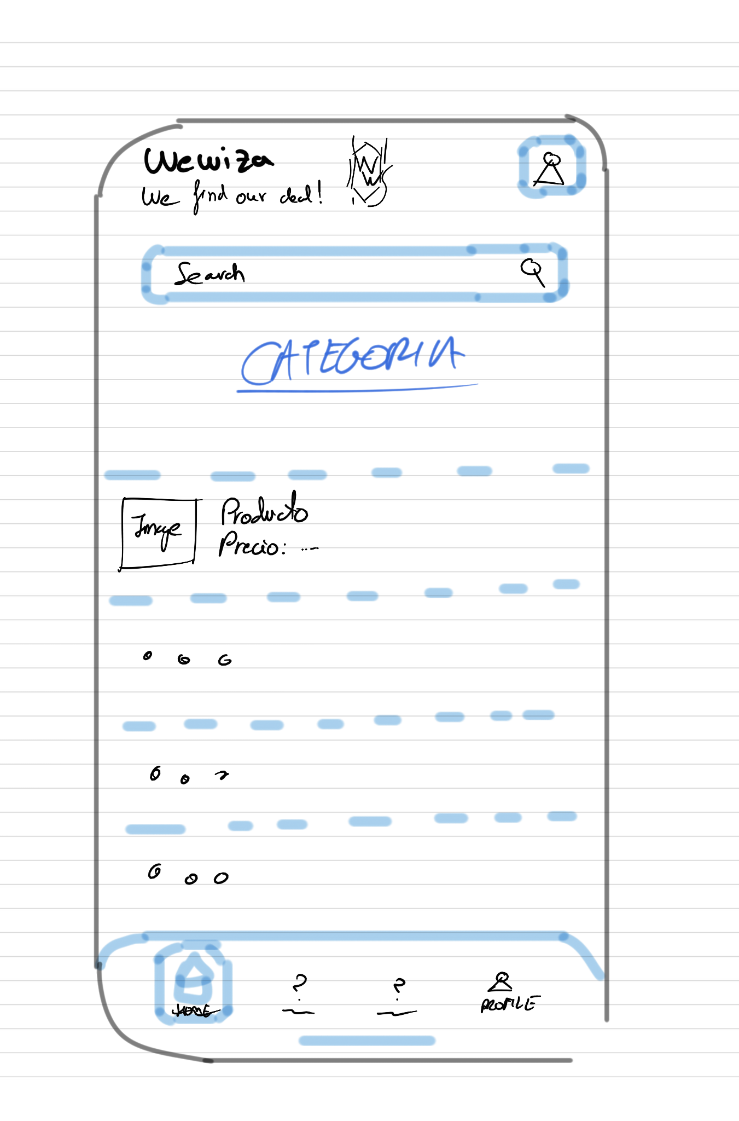
En nuestra implementación hemos definido que un producto solamente pueda tener referencia a una categoría ya que no requerimos de más, aunque una ampliación considerable sería ampliar el catálogo de categorías así dando lugar a que los productos puedan hacer referencia a una lista de categorías.

* **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.



* + 1. Frontend
* **Usuario**: siempre se almacenará un usuario independientemente del tipo de log-in.
* **ListaDeCompra**: un usuario podrá disponer de varias listas de compras.
* **ProductoFrontWewiza**: al pasar por la API-market y API-wewiza recogemos todos los datos para poderlos presentar.



* 1. Diagrama Entidad-Relación (Base de Datos)
     1. Backend MongoDB:
     2. Frontend Firebase:
  2. Mockup
  3. Tipos de fuentes de texto
* Títulos de grado 1 (H1): Mermaid1001
* Títulos de grado 2 (H2): TT Neue Bold, DemilBold o Medium
* 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

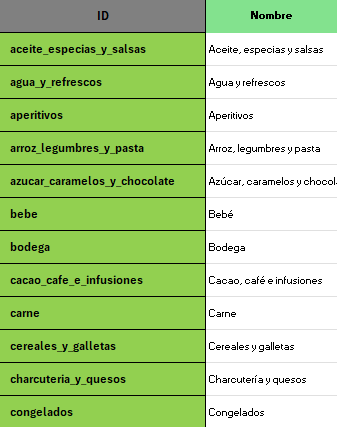
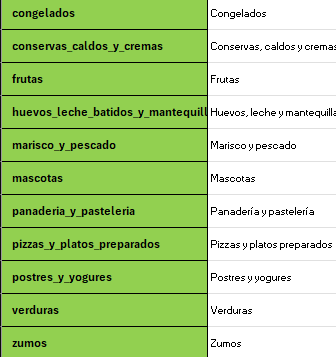
Descripción generada automáticamente

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:



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:

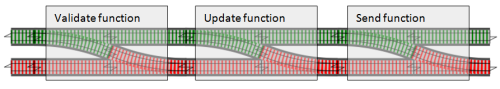
Texto

Descripción generada automáticamente

* 1. Repositorios
     1. ProductosMarkets
     2. ProductosWewiza
  2. Servicios
     1. Scrapping
     2. Procesamiento de productos
  3. Endpoints
  4. Implementación basada en eventos (Reactividad)

Esto sería lo de los likes/valoraciones de cada producto… y es probable que sea buena idea en el back, hacerlo todo asíncrono? Por el tema de no saturar el servidor y delegamos las peticiones a las capacidades del procesador del servidor…

* 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
     2. JWT
  2. Inversión de control/Inyector de dependencias

**D**ependency Injector o DInject???

1. Desarrollo Front-end
   1. Interfaz
   2. Firebase
      1. Usuarios
      2. Gamificación
   3. Inversión de control/Inyector de dependencias

Koin=???

1. Tests
   1. Pruebas Unitarias e Integración

Pytes y Pytest-mock

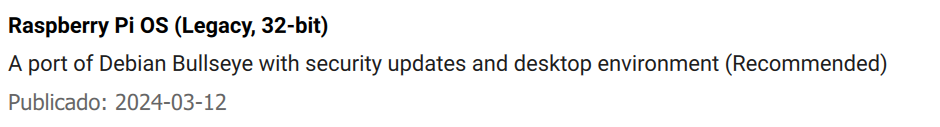
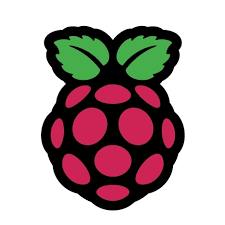
* 1. Rutas (End-Points)

Thunder-Client, Postman, etc…

1. Implantación/Despliegue
   1. Alojamiento Servidor

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

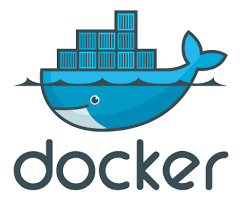
Está ejecutando una distribución Debian como sistema operativo, lo que proporciona un entorno estable y bien soportado para mi aplicación



* **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.

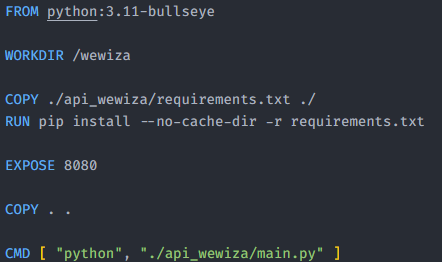
* **Base de Datos:** se implementó una base de datos MongoDB acorde a la arquitectura ARM del procesador de la Raspberry Pi 3, mediante la imagen de Docker: “apcheamitru/arm32v7-mongo”.
* **UFW:** facilitó la seguridad para disponer de los puertos abiertos deseados y configurados.
* **SSH**: nos facilitó 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:

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

Texto

Descripción generada automáticamente

* 1. Android

Intellij o AndroidStudio nos facilita la distribución por APK…

1. Conclusiones

1. Bibliografía