UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Guía de compra en un supermercado basada en voz

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Saugar Villar, Noelia

Tutor/a: Martínez Hinarejos, Carlos David

CURSO ACADÉMICO: 2022/2023

Resumen

Es habitual que a la hora de realizar la compra en un establecimiento no sepamos dónde se sitúa el artículo que queremos adquirir, bien por tratarse de un comercio con el que no estamos familiarizados o por cambios en la disposición de los productos.

Por esa razón, este trabajo se centra en la implementación de una aplicación móvil para dispositivos Android capaz de localizar productos en un supermercado tanto de forma manual como por voz, mostrando su ubicación sobre el plano del comercio. Los usuarios podrán escanear un código QR a la entrada del local, obteniendo acceso a la base de datos que almacena sus productos y planos. Para ello se empleará la plataforma de desarrollo de aplicaciones Android Studio, así como Firebase para la gestión de datos y la clase RecognizerIntent para el reconocimiento de voz.

En este informe se detalla el proceso seguido para el desarrollo de la aplicación, los análisis de tecnologías actuales y aplicaciones similares realizados, así como los obstáculos que se han encontrado y cómo se solucionaron.

Palabras clave: Localización, compra inteligente, reconocimiento de voz, aplicación móvil, Android, Firebase, RecognizerIntent, QR

Abstract

It is common that when shopping at a shop, we often find ourselves unaware of the location of a specific item. This could be due to unfamiliarity with the store layout or changes in products placements.

For that reason, this project focuses on the implementation of a mobile application for Android devices that can assist in locating products within a supermarket, both through manual search and voice commands, while displaying their positions on the store map. Users will be able to scan a QR code at the store entrance, gaining access to a database containing product information and store layouts. Android Studio development platform will be employed, alongside Firebase for data management and RecognizerIntent class for voice recognition.

This report outlines the development process of the app, examines current technologies and similar applications, as well as addresses encountered challenges along with their solutions.

Keywords: Localization, smart shopping, voice recognition, mobile application, Android, Firebase, RecognizerIntent, QR

Índice de contenidos

Índice de ilustraciones

Índice de tablas

1. Introducción

Hoy en día, el 97,2% de los hogares españoles tienen teléfono móvil, lo que supone más de 33 millones de usuarios de *smartphones* [27]. Teniendo en cuenta estas cifras, la integración de los teléfonos móviles en tareas cotidianas como la compra semanal, escuchar música o ir al gimnasio es una realidad para muchos.

Realizar la compra puede volverse una tarea verdaderamente peliaguda si nos encontramos en establecimientos de grandes dimensiones o novedosos para nosotros. De hecho, incluso si frecuentamos el establecimiento, los cambios en la disposición de los productos o la adquisición de un producto poco habitual pueden suponer un reto para el consumidor.

La creciente popularidad de las aplicaciones, sumada al instinto de preguntar a los dependientes por los productos que somos incapaces de localizar, hacen que la creación de una app capaz de reconocer la voz e indicar la posición de los artículos que queremos adquirir sea de especial interés.

En este capítulo se presenta la motivación que ha llevado a seleccionar este proyecto, los objetivos que pretendemos conseguir y, finalmente, la estructura del documento.

1.1. Motivación

El principal motivo por el que decidí desarrollar este proyecto es por su relación con la creación de aplicaciones Android [1]. Dado que pertenezco a la rama de computación, durante la carrera no he tenido ninguna asignatura que se enfoque en las aplicaciones o en cómo estructurarlas, y considero que, independientemente de tu rama, es una aptitud que merece la pena aprender.

Otro motivo es el hecho de que, como visitante habitual de supermercados, una aplicación capaz de localizar cualquier producto puede ahorrar mucho tiempo y motivar a explorar nuevos establecimientos en los que no sueles comprar por la falta de familiaridad o por la distribución de los productos.

1.2. Objetivos

El objetivo principal de este proyecto es implementar una aplicación móvil que pueda guiar, mediante peticiones de voz, al lugar donde se encuentra el producto de interés en un supermercado.

Suponiendo que el establecimiento permite el acceso a la localización de sus productos, este deberá tener un código QR a la entrada para poder acceder a la base de datos de situación y al plano del establecimiento. Es decir, la aplicación tendrá un lector de QR para poder mostrar los datos del comercio.

Los objetivos específicos que debemos seguir para completar el trabajo son los siguientes:

- Construir una base de datos con los planos de los supermercados y las coordenadas de los productos en plataformas como Firebase [2].
- Adquirir nociones para desarrollar aplicaciones en entornos como Android Studio [3].
- Ofrecer un buscador de productos, así como búsqueda mediante peticiones de voz.
- Analizar herramientas capaces de transcribir de audio a texto.
- Adaptar la interfaz a cualquier tipo de dimensión u orientación.
- Comprobar que la aplicación es sencilla e intuitiva para el usuario.

1.3. Estructura

El documento está formado por 7 capítulos, siendo la Introducción el primero y en el que nos encontramos actualmente. El resto se estructura en:

- Estado del arte: En este capítulo se analiza el contexto tecnológico, comenzando con un desglose sobre las plataformas y el desarrollo de aplicaciones móviles, así como el reconocimiento del habla. Finalizamos documentando *apps* con funcionalidad similar a la nuestra.
- Análisis del problema: Con este capítulo se especifican los requisitos funcionales y no funcionales de nuestra aplicación, así como los requisitos de la interfaz y los requisitos del reconocimiento del habla.
- Diseño de la solución: Se describe la arquitectura que tiene el proyecto, profundizando en la capa de presentación y de persistencia. Además, comentamos las tecnologías utilizadas a lo largo del proyecto.
- Desarrollo de la solución propuesta: En este capítulo se comentan los problemas encontrados durante el diseño del plano, la creación de la base de datos y la implementación de las funcionalidades principales de la aplicación.
- Pruebas: En este capítulo se describen las pruebas funcionales, de calidad y de usabilidad llevadas a cabo para demostrar que la aplicación cumple los requisitos.
- Conclusiones y trabajos futuros: Se reflexiona sobre si se han cumplido los objetivos propuestos al comienzo de la memoria. Se mencionan los problemas encontrados, qué se ha aprendido tanto personal como profesionalmente, la relación del trabajo realizado con los estudios cursados y posibles trabajos futuros.

Finalmente, el último apartado consta de la bibliografía, el glosario de términos y los anexos.

2. Estado del arte

Con el fin de comprender mejor la situación actual tecnológica y antes de optar por una solución potencialmente inviable, hemos decidido realizar un estudio sobre las plataformas móviles, el desarrollo de aplicaciones, el reconocimiento del habla y las aplicaciones con funcionalidades similares a la nuestra.

2.1. Plataformas móviles

A continuación, se puede observar el análisis de las dos principales plataformas móviles:

- Android
  Sistema operativo para dispositivos móviles desarrollado por Google [4] que se caracteriza por su enfoque de código abierto y su arquitectura basada en el núcleo de Linux [5]. Diseñado para brindar una plataforma flexible y accesible, Android se aplica en una amplia gama de dispositivos, desde *smartphones* y *tablets* hasta televisores.
  - Ventajas:
    - Amplia base de usuarios: Android tiene una cuota de mercado significativa y una base de usuarios extensa en todo el mundo.
    - Mayor diversidad de dispositivos: Se ejecuta en una amplia gama de dispositivos de diferentes fabricantes, lo que brinda a los usuarios más opciones.
    - Mayor libertad para personalizar: Los usuarios Android tienen más libertad para personalizar su experiencia, así como para acceder a ajustes avanzados.

  - Desafíos:
    - Fragmentación de dispositivos: Debido a la diversidad de dispositivos, los desarrolladores deben abordar la fragmentación de hardware y software, lo que puede requerir pruebas y optimizaciones adicionales.
    - Actualizaciones del sistema operativo: Las actualizaciones pueden ser más lentas de implementar debido a la participación de múltiples fabricantes y operadores.
    - Menos control de calidad: Google Play Store [6] tiene menos restricciones en comparación con la App Store [7] de iOS [8], lo que puede permitir la aparición de aplicaciones de menor calidad
- iOS
  Sistema operativo para dispositivos móviles desarrollado por Apple [9] que se distingue por su enfoque en la elegancia, la seguridad y la facilidad de uso. Es un sistema operativo cerrado, diseñado exclusivamente para los productos de Apple.
  - Ventajas:
    - Usuarios comprometidos: Los usuarios de iOS tienden a estar más comprometidos y dispuestos a gastar dinero en aplicaciones y compras dentro de la aplicación.

- Experiencia de usuario coherente: Debido al control de hardware y software de Apple, los dispositivos ofrecen una experiencia de usuario coherente y predecible.
- Actualizaciones rápidas del sistema operativo: Apple proporciona actualizaciones rápidas y frecuentes, lo que permite a los desarrolladores aprovechar las nuevas características y mejoras.
- Calidad de las aplicaciones: La App Store de iOS tiene estándares de calidad más estrictos y una revisión manual de las aplicaciones antes de su publicación.

- Desafíos:
  - Menor diversidad de dispositivos: A diferencia de Android, iOS se ejecuta en un número limitado de dispositivos fabricados por Apple, lo que puede limitar la selección para los usuarios.
  - Restricciones para personalizar: iOS tiene restricciones más estrictas en cuanto a la personalización y ajustes del sistema.
  - Costos asociados: El desarrollo para esta plataforma generalmente implica costos adicionales.
  - Menor compatibilidad con tecnologías externas: iOS puede tener restricciones en la integración con algunos servicios de terceros o tecnologías externas, lo que puede requerir soluciones alternativas.
  - Menor flexibilidad en la distribución de aplicaciones: La App Store limita la distribución de aplicaciones. Esto implica un proceso de revisión y aprobación de aplicaciones por parte de Apple.

En general, tanto iOS como Android son plataformas sólidas para el desarrollo de aplicaciones móviles y la elección entre ellas depende de factores como el mercado objetivo, los recursos disponibles y las características de la aplicación, entre otros.

Teniendo en cuenta estos factores, decidimos desarrollar la aplicación en Android ya que, además de ser la plataforma con la que más familiarizados estamos, se ejecuta en una amplia gama de dispositivos y tiene una base de usuarios extensa.

2.2. Desarrollo de aplicaciones

El desarrollo de aplicaciones es un proceso complejo que requiere el uso de estándares, entornos, lenguajes y bibliotecas adecuadas, así como la aplicación de metodologías de desarrollo eficientes. Como indicamos en el apartado 2.1, en esta sección vamos a explorar los aspectos clave del desarrollo de aplicaciones Android [28].

2.2.1 Estándares de desarrollo

Los estándares de desarrollo son reglas y pautas establecidas por la industria para garantizar la calidad, seguridad e interoperabilidad de las aplicaciones. Algunos ejemplos de estándares incluyen pautas de diseño de la interfaz de usuario, estándares de codificación, estándares de seguridad y estándares de calidad de software.

A continuación, se listan posibles criterios de calidad, agrupados en sus respectivos ámbitos, para las aplicaciones Android [29]:

- Experiencia visual: Patrones de interacción y diseño a fin de garantizar una experiencia de usuario intuitiva y coherente. Estos son algunos ejemplos:
    - La *app* debe admitir la navegación estándar del botón Atrás, sin usar avisos personalizados en pantalla.
    - Las notificaciones siguen los alineamientos de Material Design [10].
    - La *app* permite tanto orientación vertical como horizontal (si es posible).
    - La aplicación se puede cambiar al modo oscuro.

  En general, se recomienda usar los componentes de Material Design para crear la interfaz de usuario. Material Design es un sistema de diseño de código abierto de Google. Consiste en un conjunto de directrices de diseño que permite crear interfaces de usuario atractivas y coherentes. Propone colores, tipografía y elementos visuales para ofrecerle al usuario una experiencia intuitiva y consistente con la aplicación.

- Funcionalidad: La aplicación debe funcionar según lo previsto:
    - Si la reproducción de audio es una función principal, la *app* debe admitir la reproducción en segundo plano.
    - La *app* evita que se ejecuten servicios innecesariamente largos en segundo plano.
- Rendimiento y estabilidad: La aplicación debe proporcionar el rendimiento, estabilidad, compatibilidad y capacidad de respuesta esperados:
    - La *app* se carga rápidamente o proporciona comentarios en pantalla.
    - La aplicación se ejecuta en la última versión pública de Android sin problema.
- Privacidad y seguridad: Se deben administrar de forma segura los datos de usuario y su información personal:
    - La *app* transmite claramente el motivo por el que se necesitan ciertos permisos.
    - Todos los datos sensibles se almacenan en el almacenamiento interno de la *app*.
    - La aplicación declara una configuración de seguridad de red.

Debido a que el objetivo de este proyecto es el desarrollo de una aplicación Android, nuestra *app* seguirá en lo posible los estándares anteriormente mencionados.

2.2.2 Entornos de desarrollo

Los IDE, entornos de desarrollo integrado, son aplicaciones software que ayudan a los programadores a desarrollar código de manera eficiente. Aumentan la productividad de los desarrolladores al combinar capacidades como editar, crear, probar y empaquetar software en una aplicación fácil de usar.

Hay multitud de entornos de desarrollo populares para crear aplicaciones Android. Algunos de los más utilizados son [30]:

- Android Studio:

  Es el entorno de desarrollo oficial de Android, creado por Google. Está basado en IntelliJ IDEA [13] y proporciona una amplia gama de herramientas y características específicas de Android. Incluye un editor de código, un depurador, un emulador, un generador de interfaces gráficas y muchas otras herramientas.

  El lenguaje principal utilizado en Android Studio es Java [11]. Sin embargo, Google introdujo el soporte para el lenguaje Kotlin [12] como alternativa.

  Además, Android Studio ofrece un conjunto de bibliotecas y *frameworks* que facilitan el desarrollo de aplicaciones Android:

  - Android Jetpack [14]:

    Conjunto de bibliotecas que ayuda a los desarrolladores a seguir las prácticas recomendadas, reducir el código estándar y escribir código que funcione de manera coherente en los dispositivos.
    A continuación, listamos dos bibliotecas de Jetpack que ayudan a simplificar la capa de datos:
    - *LiveData*: Permite que los componentes de la interfaz (actividades o fragmentos) observen cambios en los datos. Se utiliza para actualizar automáticamente la interfaz en respuesta a cambios en los datos subyacentes.
    - *ViewModel:* Se asocia a una actividad o fragmento y mantiene los datos necesarios para la interfaz, evitando así problemas en los cambios de configuración, como rotaciones de pantalla.

  - Android App Bundle [15]:

    Es el formato recomendado por Google para publicar aplicaciones en la Play Store. Incluye todos los recursos y el código compilado de la *app*, pero delega la generación de APK y la firma a Google Play. Entre sus funciones se encuentra la reducción del tamaño de descarga y la optimización del rendimiento de la aplicación.

- Eclipse [16]:

  IDE gratuito de código abierto utilizado para crear aplicaciones en Java y otros lenguajes de programación. Las mejoras y actualizaciones más recientes le han permitido crear código en C++, Python [17], JavaScript, etc. [29].

  Proporciona un conjunto de herramientas y características para facilitar la escritura, compilación y depuración de código. Eclipse puede utilizar las bibliotecas estándar de Android, que incluyen un amplio conjunto de funciones y componentes para construir aplicaciones como actividades, fragmentos y servicios.

  Además, Eclipse ofrece complementos que facilitan el desarrollo de aplicaciones como ADT (*Android Development Tools*), herramientas específicas para el desarrollo de

aplicaciones Android, que permiten crear proyectos, diseñar interfaces, compilar, depurar y administrar recursos.

Eclipse es el mejor IDE para desarrollo Android si se necesita realizar un desarrollo significativo, pero ha dejado de ser el IDE oficial para desarrollo de aplicaciones Android, siendo reemplazado por Android Studio.

- Visual Studio [18]:

  IDE desarrollado por Microsoft [23] que se utiliza principalmente para el desarrollo de software en diferentes plataformas (Windows, Android, iOS y web).

  Para el desarrollo de aplicaciones, Visual Studio utiliza varios lenguajes de programación como C#, Visual Basic, C++ y JavaScript, siendo C# el lenguaje de programación principal y el que ofrece una amplia gama de bibliotecas y herramientas para el desarrollo de aplicaciones multiplataforma.

  En cuanto a bibliotecas y *frameworks*, se pueden utilizar las bibliotecas estándar de Android, como se mencionó en Eclipse. Además, Visual Studio ofrece soporte para Xamarin, un conjunto de bibliotecas y herramientas que permiten el desarrollo de aplicaciones móviles multiplataforma.

  En resumen, Visual Studio es una aplicación cuyo enfoque principal es el desarrollo de software multiplataforma. La mayoría de los desarrolladores lo utilizan para crear aplicaciones nativas e híbridas.

2.2.3 Metodologías de desarrollo

La metodología es un marco de trabajo (*framework*) utilizado para estructurar, planificar y controlar el proceso de desarrollo de un software. Establece un enfoque sistemático en el desarrollo informático, consiguiendo que los desarrolladores puedan trabajar juntos de forma más eficiente, o que puedan organizar sus tareas fácilmente [41].

Las metodologías ágiles [42] basan su fundamento en la adaptabilidad de los procesos de desarrollo. Son procesos incrementales, cooperativos, sencillos y adaptativos. Es decir, es una metodología que responde a los cambios más que seguir una planificación, razón por la cual hemos considerado que es la metodología apropiada para este proyecto.

Dentro de las metodologías ágiles, una de las más destacadas es Scrum [42], la cual se centra en la colaboración y la flexibilidad, adaptándose a los cambios y brindando valor constante al cliente. En esta metodología, el trabajo se divide en iteraciones cortas llamadas *sprints*, generalmente de 1 a 4 semanas, y al final de cada uno se entrega un incremento final del producto.

Otra metodología que destacar en este ámbito es Kanban [42]. Kanban se basa en un sistema de tablero visual donde se representan las tareas o actividades a realizar. Las tarjetas o notas adhesivas son las tareas, y su movimiento a través de diferentes columnas en el tablero representa el flujo de trabajo y el progreso de dichas tareas.

Una vez analizadas ambas opciones, se llegó a la conclusión de que Kanban se adapta mejor a nuestro proyecto. Esto es debido a que es una metodología con menor énfasis en la planificación y la estimación, mientras que en Scrum puede haber dificultades para estimar de manera precisa el esfuerzo y el tiempo necesarios para completar las tareas. Además, con Kanban las actualizaciones se publican en cuanto están listas, sin ninguna planificación periódica ni fechas predeterminadas, facilitando el flujo de trabajo [43].

2.3. Reconocimiento del habla

El reconocimiento del habla, también denominado ASR, es una funcionalidad que permite a un programa procesar el habla humana para obtener una transcripción escrita de la misma. Utilizando algoritmos y modelos estadísticos, los ASR analizan las señales de voz y las comparan con patrones lingüísticos y acústicos predefinidos para identificar y transcribir palabras y frases. Estos sistemas tienen aplicaciones en campos como la traducción automática, la asistencia virtual y la transcripción de audio, entre otros.

Los ASR son el medio de comunicación futuro entre humanos y máquinas, pero cuestiones como el dialecto, velocidad al hablar o el tono pueden suponer problemas para estos sistemas [31]. A continuación, vamos a detallar la estructura de los ASR, junto con las herramientas y recursos más recomendados en este campo.

2.3.1 Arquitectura

En la Figura 1 se encuentra el esquema de la estructura de un ASR.

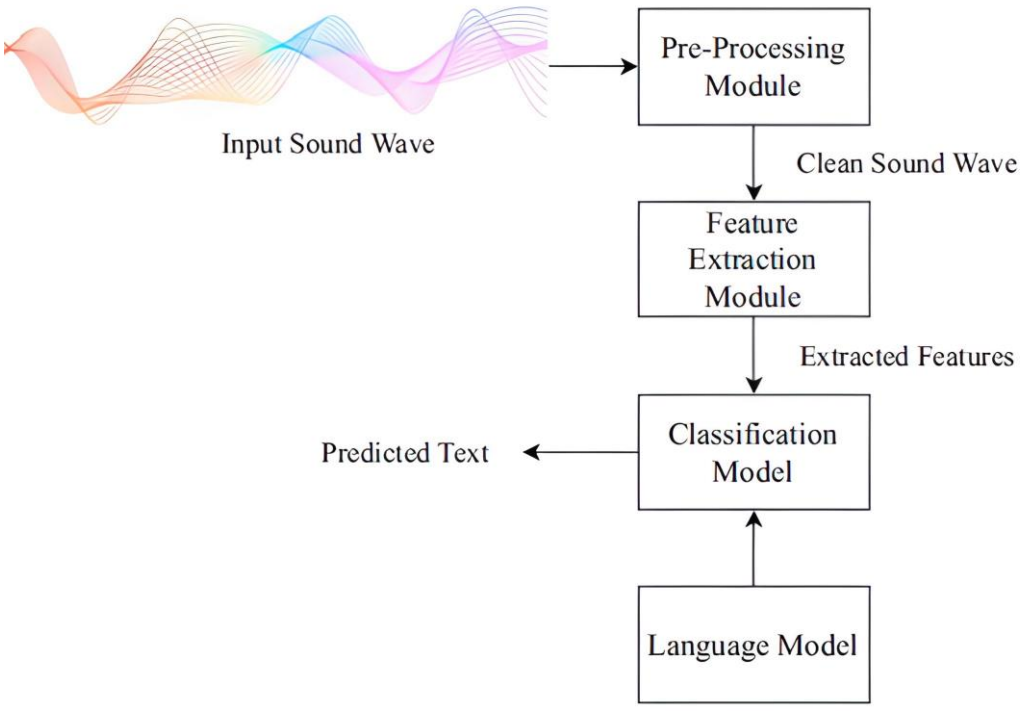

Figura 1: Estructura básica de un ASR [31]

El sistema se divide en cuatro módulos:

- Módulo de preprocesamiento (*Pre-processing module*): Se centra en reducir la relación señal-ruido, ya que la onda de sonido que reciben los ASR como entrada suele contener ruido. Se emplean métodos como el normalizado.
- Módulo de extracción de características (*Feature extraction module*): Este paso es clave para obtener un sistema eficiente. La extracción de características se realiza con métodos como LPC [44] o MFCCs [45]. Su objetivo es obtener una representación vectorial del habla, de forma que sea tratable por modelos matemáticos.
- Modelo de clasificación (*Classification model*): Predice el texto correspondiente a la señal de entrada, aprendiendo la relación entre las características extraídas del audio y su texto correspondiente. Tiene dos enfoques:
    - Generativo: Utiliza una distribución de probabilidad conjunta para predecir la salida futura a partir de los datos de entrenamiento. Algunos modelos que emplean este enfoque son HMM [46] y GMM [47].
    - Discriminativo: Calcula un modelo paramétrico a partir de un conjunto de entrenamiento de vectores de entrada y sus correspondientes vectores de salida. Es un enfoque empleado por modelos como SVM [48] y ANN [49].
- Modelo del lenguaje (*Language model*): Soluciona los posibles problemas que puedan surgir del dialecto, entonación, etc. Usa las restricciones estructurales de un idioma para predecir las probabilidades de ocurrencia de una palabra para una secuencia de palabras específica. Es decir, comprueba la probabilidad de ocurrencia de la palabra generada por el clasificador.

2.3.2 Herramientas y recursos en línea

Actualmente, hay múltiples herramientas de reconocimiento del habla disponibles. A continuación, listamos las que hemos considerado más relevantes para el proyecto [31, 32, 63].

2.3.2.1 Kaldi

Kaldi [19] es una herramienta reconocedora del habla de código abierto escrita en C++. Su objetivo es ofrecer un código moderno y flexible, que sea fácil de entender, modificar y ampliar. Kaldi compila en los sistemas tipo Unix (Linux) y en Windows [20].

Algunas de las características principales de Kaldi son [33]:

- Diseño extensible: Los algoritmos proporcionados son genéricos.
- Licencia abierta: El código está bajo la licencia Apache v2.0, una de las licencias menos restrictivas.
- Medios completos: Se ponen a disposición medios completos para construir sistemas de reconocimiento de voz, que funcionan con bases de datos ampliamente disponibles.
- Pruebas exhaustivas: El objetivo es que casi todo el código tenga sus correspondientes rutinas de prueba.

En cuanto a la extracción de características, se busca crear características estándar de MFCC y PLP [50], estableciendo valores predeterminados, pero dejando disponibles algunas opciones para que se puedan ajustar (por ejemplo, el límite de frecuencia máxima).

Kaldi proporciona herramientas para convertir los modelos del lenguaje en el formato ARPA [51] a FSTs [52], en el que se basa su marco de trabajo. Es decir, es posible utilizar cualquier modelo de lenguaje que pueda representarse como un FST. Además, para la clasificación soporta modelos como GMM y DNN [53]. El uso de DNN permite que se acelere el procesamiento de forma significativa [32].

La principal ventaja de Kaldi frente a sus competidores, como HTK, es su código moderno, flexible y estructurado, además de la licencia abierta.

2.3.2.2 Sphinx-4

Sphinx-4 [21] es una herramienta reconocedora del habla de código abierto escrita en Java. Es altamente modular y flexible, admitiendo todo tipo de modelos acústicos basados en HMM, así como todos los tipos estándar de modelos de lenguaje. Además, como está desarrollado en Java, puede ejecutarse en cualquier sistema que admita la máquina virtual de Java (JVM).

Sphinx-4 se estructura en 3 módulos principales: el *FrontEnd*, el *Decoder* y el *Linguist*. El *FrontEnd* (Interfaz de entrada) toma la señal de entrada y la parametriza en una secuencia de características, es decir, es el módulo que extrae las características. El *Linguist* (Intérprete lingúístico) traduce cualquier tipo de modelo de lenguaje, junto con información sobre la pronunciación y la estructura, en un grafo de búsqueda. El *Decoder* (Decodificador) contiene un gestor de búsqueda que usa las características extraídas del *FrontEnd* y el grafo de búsqueda del *Linguist* para clasificar la información y ofrecer resultados [34].

Sphinx-4, a diferencia de otros sistemas, contiene un gestor de configuración (*ConfigurationManager*) que permite cargar y configurar módulos de forma dinámica durante la ejecución del sistema. Debido a esto, aunque el *FrontEnd* suele producir MFCC, se puede reconfigurar para generar PLP sin necesidad de modificar el código fuente o recompilar el sistema [34].

El modelo del lenguaje del sistema (*Linguist*) soporta gran variedad de formatos como FST, modelos de n-gramas o JSG [54]. Además, utiliza HMM para realizar la clasificación, modelando las secuencias de características y estimando las probabilidades de las palabras correspondientes.

Finalmente, aunque en un principio Sphinx-4 fue diseñado para el idioma inglés, su naturaleza modular ha permitido que se adapte a otros idiomas como el mandarín y el ruso [31].

2.3.2.3 HTK

HTK (*Hidden Markov Model Toolkit*) [22] consiste en un conjunto de herramientas para construir Modelos Ocultos de Markov (HMM). Sin embargo, HTK está diseñado

principalmente para construir herramientas de procesamiento de voz basadas en HMM [35]. El sistema está escrito en C [31] y se encuentra disponible en Linux, macOS y Windows.

HTK dispone de módulos para la conversión, preprocesado y parametrización de la información, herramientas para definir y manipular HMM, librerías para entrenar los HMM ya definidos, funciones para definir la gramática y, además, herramientas adicionales para lograr el objetivo final de obtener una transcripción del habla [36].

Entre los algoritmos disponibles para la extracción de las características encontramos MFCC y LPC, entre otros [35, 36]. En cuanto al modelo del lenguaje, las herramientas de modelado de lenguaje de HTK están diseñadas para construir y probar modelos estadísticos de n-gramas [35].

2.3.2.4 iATROS

iATROS (*improved Automatically Trainable Recognizer of Speech*) es un reconocedor tanto del habla como de la escritura a mano. El sistema está compuesto por dos módulos de preprocesamiento y extracción de características, para las señales de voz e imágenes de la escritura a mano, junto con un módulo de reconocimiento central. Todos los módulos están implementados en C [37].

Los módulos de preprocesamiento y extracción de características obtienen los vectores de características que necesita el módulo de reconocimiento. La extracción de características se realiza mediante los MFCCs en el caso del reconocimiento del habla [37].

El módulo de reconocimiento utiliza modelos HMM que, junto a las características extraídas y el modelo del lenguaje, busca la mejor hipótesis de reconocimiento. En cuanto al modelo del lenguaje, el analizador de modelos de lenguaje de iATROS admite N-gramas (en formato ARPA) y FSM [37, 55].

iATROS se puede emplear para funciones como el reconocimiento *off-line*, el reconocimiento de voz *on-line* o combinar el reconocimiento de la escritura a mano con el reconocimiento del habla [37].

2.3.2.5 RecognizerIntent

RecognizerIntent [64] es una clase de Android, implementada en el lenguaje de programación Java, que proporciona una interfaz para utilizar el servidor de reconocimiento de voz del sistema operativo. Es decir, permite usar el reconocedor del habla proporcionado por el *framework* de Android, incorporando funcionalidades de reconocimiento de voz en las aplicaciones.

A través de este reconocedor, los usuarios pueden hablar en su dispositivo y el sistema intentará convertir su discurso en texto. Puede usarse para diversas funcionalidades en la aplicación, como realizar búsquedas por voz, órdenes de voz o transcripciones de voz a texto.

Se pueden modificar parámetros como el modelo de lenguaje utilizado o los idiomas que se reconocen [63, 64]. La implementación de esta API probablemente transmitirá audio a

servidores remotos para realizar el reconocimiento de voz, lo que puede consumir una cantidad considerable de ancho de banda.

2.3.2.6 Comparativa herramientas

	Extracción características	Clasificación	Modelo del lenguaje	Lenguaje programación	Sistemas operativos
Kaldi	MFCC, PLP	GMM, DNN	FST	C++	Linux, Windows
Sphinx-4	MFCC, PLP	HMM	FST, N-gramas, …	Java	Linux, macOS, Windows
HTK	MFCC, LPC, …	HMM	N-gramas	C	Linux, macOS, Windows
iATROS	MFCC	HMM	N-gramas, FSM	C	Linux
RecognizerIntent	Depende del sistema	Depende del sistema	Se puede modificar	Java	Android

Tabla 1. Comparativa de herramientas de reconocimiento del habla

La Tabla 1 muestra la comparativa entre estas aplicaciones. Como se puede ver, Kaldi supera a todas las otras herramientas de reconocimiento del habla, brindando un conjunto completo de herramientas y algoritmos para la construcción de sistemas de reconocimiento de voz. Usa las técnicas más avanzadas (DNN, MFCC, …), habilitando los mejores resultados en poco tiempo, pero con el mayor coste computacional [32]. Sin embargo, Kaldi puede tener una curva de aprendizaje pronunciada y requerir conocimientos técnicos avanzados para su implementación adecuada.

Sphinx-4 ofrece una arquitectura modular y flexible con soporte para múltiples idiomas [31, 34]. No obstante, puede tener una precisión de reconocimiento inferior en comparación con otras herramientas.

En cuanto a HTK, es conocido por ser una herramienta ampliamente utilizada en el campo del reconocimiento del habla, ofreciendo una amplia gama de funcionalidades. En cambio, su licencia restrictiva puede limitar su uso en algunos casos [36] ya que, a diferencia de Kaldi y Sphinx-4, no es de código abierto [31].

Por otro lado, iATROS, herramienta que destaca por su capacidad para reconocer tanto señales de voz como escritura a mano, es conocido por su facilidad de uso y su enfoque en la implementación rápida de nuevas aplicaciones [37]. A pesar de tener una capacidad de reconocimiento limitada en comparación con otras herramientas, su facilidad de uso hace que destaque entre las herramientas reconocedoras del habla.

Finalmente, nos enfocamos en RecognizerIntent. A diferencia del resto de herramientas, no es una API que proporcione módulos para la extracción de características o la clasificación de voz, sino que es una forma de interactuar con servicios de reconocimiento de voz disponibles en el dispositivo. Es decir, si se busca una solución rápida y sencilla de reconocimiento de voz que se integre bien en Android, RecognizerIntent es la herramienta adecuada.

En resumen, aunque iATROS presenta gran facilidad de uso, RecognizerIntent, al ser una clase nativa de Android, requiere menos configuración y es más fácil de integrar. Además, al tratarse de una aplicación sencilla, no se necesitan las funcionalidades avanzadas de reconocimiento de voz que presenta iATROS.

2.4. Aplicaciones similares

La investigación realizada nos ha permitido encontrar algunas aplicaciones con funcionalidades similares a la nuestra:

2.4.1 LetsBuy

LetsBuy [38, 39] es una aplicación para móviles Android desarrollada por dos alumnos de la Universidad Pontificia de Salamanca (UPSA) cuyo objetivo es facilitar las compras en supermercados a través de una interfaz sencilla e intuitiva.

La aplicación permite crear listas de compra, localizar el supermercado deseado dentro de la ciudad e indicar al usuario dónde se encuentran los productos dentro de la tienda. Además, emite un sonido cuando el usuario se encuentra cerca de uno de los productos listados. Estas funcionalidades permiten reducir la dificultad para encontrar ciertos productos al realizar compras en un supermercado.

No obstante, los creadores de la aplicación no pudieron establecer convenios con supermercados reales, teniendo que desarrollarla para un supermercado “ficticio”. Además, dicha aplicación no está disponible actualmente para ningún dispositivo, pero su parecido con nuestro proyecto hizo necesaria su mención en esta sección de la memoria.

En la Figura 2 podemos observar la interfaz de esta aplicación y cómo localiza los productos mediante flechas que señalan su posición sobre el plano del supermercado.

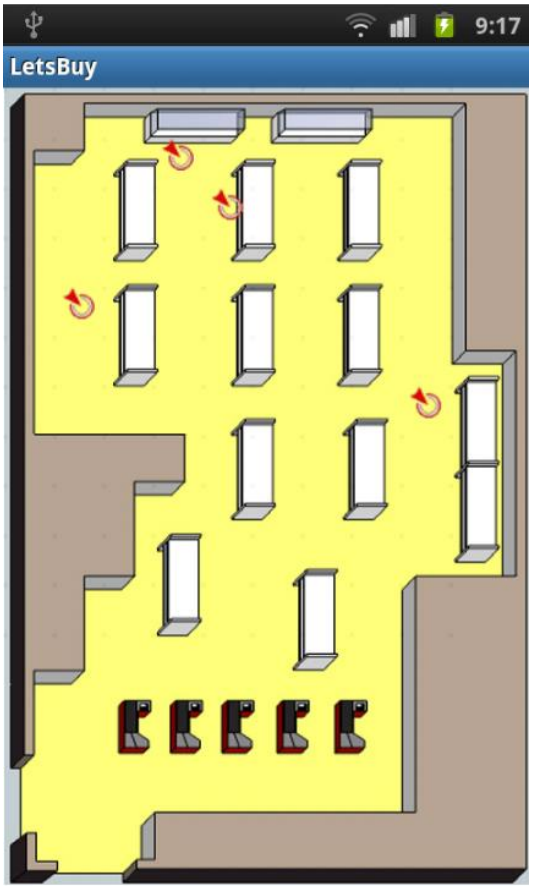

Figura 2: LetsBuy

A continuación, se muestran las características principales:

- Crear listas de compra
- Localizar productos en el supermercado
- Buscar supermercado en la ciudad
- Avisar de la proximidad a los productos seleccionados

2.4.2 Entrada de voz lista compras

Entrada de voz lista compras [24] es una aplicación móvil Android disponible en la Play Store. Permite, entre otras cosas, crear listas de compras mediante entrada de voz.

Características principales:

- Crear listas de compra: Se pueden introducir los artículos tanto por teclado como por entrada de audio. Además, podemos cambiar el nombre de las listas y reorganizar tanto los productos que tienen como las listas en sí.
- Entrada de voz: La característica más destacable de la aplicación. Permite ahorrar mucho tiempo y es el principal motivo por el que los usuarios descargan la *app*. La entrada de voz se usa para introducir productos en listas de compra.
- Compartir la lista de la compra: Una vez creada la lista, se puede compartir por redes sociales.

En la Figura 3 podemos observar la interfaz de la aplicación, así como su función de entrada de voz.

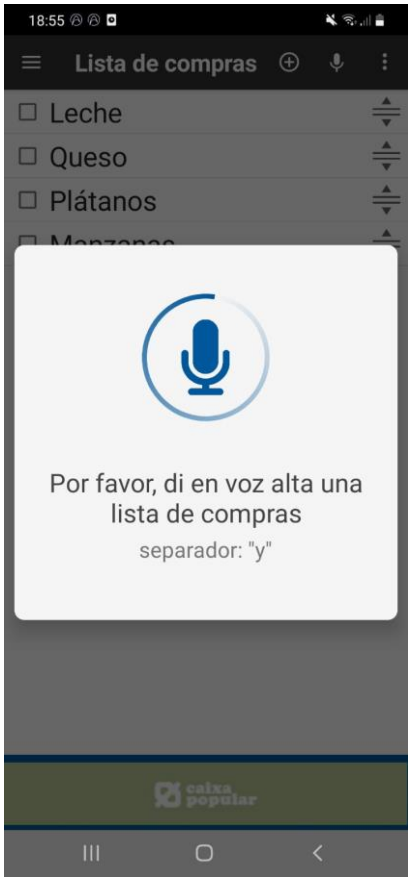

Figura 3: Entrada de voz lista compras

Esta es una aplicación que no se encuentra vinculada a ningún supermercado, a diferencia de la nuestra. Las listas se crean independientemente de si el supermercado que se quiere visitar tiene los productos o no. Decidimos mencionarla en la memoria por su función de entrada de voz, ya que es una de las características principales de nuestro proyecto.

2.4.3 Mercadona

Mercadona [25] es una aplicación móvil considerada la mejor a nivel de diseño y rendimiento [40] entre las *apps* de supermercados en España. Aunque está limitada a ciertas ciudades, nos da acceso a la compra *online*, además de permitirnos ver los productos en detalle, añadirlos a nuestra cesta y revisar compras anteriores, entre otras funciones.

La Figura 4 muestra el listado de productos disponibles en el supermercado seleccionado mediante el código postal que se debe introducir al iniciar la aplicación.

Figura 4: Mercadona

Características principales:

- Crear listas de compra: Tras introducir el código postal, se muestran los artículos y ofertas del supermercado más cercano. Podemos añadir los productos que nos interesen a un carrito virtual y realizar la compra.
- Buscar productos: Los artículos están organizados por categoría, facilitando su búsqueda. La aplicación también cuenta con una sección de bajadas de precio, novedades y productos habituales.
- Compra *online*: Si estamos dentro de los códigos postales donde el servicio está disponible, se puede hacer la compra desde donde queramos con el móvil de forma rápida e intuitiva.

Esta aplicación ofrece prestaciones similares a la nuestra, como el poder consultar los productos disponibles en el establecimiento, pero carece de búsqueda por voz y no puede localizar los productos en el supermercado.

2.4.4 Zara

Zara [26] es una aplicación móvil sobre la marca de ropa Zara. Nos permite consultar la nueva colección, las últimas tendencias y comprar *online*. Pero la función que nos interesa en este

estudio, y el motivo por el que hemos decidido incluir una aplicación sobre moda en vez de supermercados, es su capacidad para localizar una prenda en la tienda seleccionada.

Es decir, al escoger una prenda tenemos la posibilidad de pedir su disponibilidad, dejándonos seleccionar el establecimiento en el que tienen existencias y mostrando en un plano dónde está situada, como se puede ver en la Figura 5.

Figura 5: Zara

Características principales:

- Buscar productos (ropa): Organiza las prendas en función al género, edad y tipo. Se puede buscar la prenda en ese listado o escaneando el código de barras del producto.
- Compra *online*: Podemos comprar las prendas que introduzcamos al carrito desde el dispositivo móvil, sin necesidad de ir presencialmente a una tienda Zara.
- Buscar prenda en la tienda: Todas las prendas tienen un apartado que nos permite consultar su disponibilidad en la tienda. Solo tenemos que indicar la sucursal de Zara en la que queremos comprar y nos mostrará un plano del establecimiento, señalando la zona y la planta en la que se encuentra el producto, si está disponible.
- Localizar tiendas en la ciudad: En la pantalla principal de la aplicación se encuentra un menú que nos deja buscar las tiendas de la ciudad que introduzcamos, además de indicar su horario, punto de recogida, información de contacto e indicaciones sobre cómo llegar al establecimiento.

En este caso, aunque la aplicación de Zara permite localizar productos en sus establecimientos, solo señala la zona en la que se encuentra, como podemos ver en la Figura 4, mientras que nuestra aplicación funciona por coordenadas, indicando la posición exacta.

2.4.5 Comparativa entre aplicaciones

La Tabla 2 muestra una comparativa de las aplicaciones analizadas con respecto a los objetivos planteados en la Introducción.

Funcionalidad	LetsBuy	Entrada de voz lista compras	Mercadona	Zara
Sistema Operativo	Android	Android	Android/IOS	Android/IOS
Plano y coordenadas	Sí	No	No	Sí
Buscar producto	Sí	No	Sí	Sí
Entrada de voz	No	Sí	No	No

Tabla 2. Comparativa entre aplicaciones similares

LetsBuy es la *app* que más se aproxima a nuestro planteamiento, aunque carece de entrada de voz, al igual que la aplicación de Zara. Además, la *app* de entrada de voz no está vinculada con ningún supermercado, por lo que no muestra ni los productos ni el plano. Finalmente, la aplicación del Mercadona, aunque sí que muestra los artículos disponibles, no puede localizarlos en el establecimiento.

Es decir, no hay ninguna aplicación que cumpla todos los objetivos planteados para nuestro proyecto.

3. Análisis del problema

Mediante el análisis de la situación tecnológica actual (capítulo 2), se identificó la necesidad de una solución que optimice la experiencia de compra en supermercados, brindando a los clientes una herramienta eficiente y práctica para la localización de productos. El proyecto propuesto se presenta como una solución innovadora que permitirá a los usuarios encontrar productos de manera rápida y sencilla, mejorando así su experiencia de compra.

3.1. Solución propuesta

Se plantea una aplicación capaz de localizar productos tanto mediante una selección manual como mediante el reconocimiento de voz. La *app* permitirá a los usuarios escanear un código QR ubicado en el exterior del supermercado, lo que les brindará acceso al plano del establecimiento y a una lista de productos, ambos almacenados en una base de datos que solo pueden modificar los administradores.

Los usuarios podrán seleccionar productos de la lista y visualizar su ubicación en el plano del establecimiento. Además, la aplicación ofrecerá una funcionalidad de reconocimiento de voz, a través de la cual los usuarios podrán pronunciar en voz alta el nombre del producto que deseen encontrar y la aplicación lo localizará en el plano. Por lo tanto, el objetivo principal de esta aplicación es solucionar la dificultad que muchos usuarios experimentan a la hora de localizar productos específicos dentro de un supermercado, consiguiendo que los clientes puedan orientarse fácilmente por el establecimiento.

El desarrollo de la aplicación se puede dividir en varias fases:

- Análisis de requisitos: Se realizará un análisis detallado de los requisitos funcionales y no funcionales de la aplicación, así como los requisitos de la interfaz de usuario y los requisitos del reconocimiento del habla.
- Diseño de la arquitectura: Se diseñará la arquitectura de la aplicación, determinando cómo se estructurarán los componentes y cómo se realizará la comunicación con la base de datos del supermercado. Se especificarán los detalles técnicos y se mencionarán las tecnologías utilizadas.
- Desarrollo de la aplicación: Se procederá a la implementación de la aplicación, siguiendo las pautas y estándares establecidos previamente. Se desarrollarán los módulos para el escaneo del código QR, la visualización del plano, la selección de productos y el reconocimiento de voz, entre otros.
- Pruebas: Una vez completada la fase de desarrollo, se realizarán pruebas exhaustivas para identificar y corregir posibles errores o fallos de funcionamiento. Se probarán todos los escenarios posibles, verificando la precisión del reconocimiento de voz, la visualización correcta del plano y de los productos, así como la recuperación de los datos almacenados en la base de datos del supermercado.

En cuanto a la validación de la aplicación, se llevarán a cabo diferentes enfoques:

- Validación funcional: Se verificará que la *app* cumpla con todos los requisitos funcionales establecidos previamente. Se asegurará que la funcionalidad principal, como el escaneo de códigos QR, la localización de productos en el plano y el reconocimiento de voz, funcionen correctamente.
- Validación de usabilidad: Se realizarán pruebas de usabilidad con usuarios reales para evaluar la facilidad de uso, la intuición de la interfaz y la eficiencia en la búsqueda de los productos. Se recopilarán comentarios y sugerencias de los usuarios para realizar mejoras si es necesario.
- Validación del reconocimiento de voz: Se evaluará la precisión y eficacia del reconocimiento de voz mediante pruebas con diferentes usuarios y variaciones en la entonación de los productos. Se compararán los resultados obtenidos con las expectativas establecidas.

3.2. Requisitos funcionales

El listado siguiente consiste en las funcionalidades que debe ofrecer la aplicación:

- Escaneo de QR: La aplicación deberá ser capaz de escanear un código QR ubicado en la entrada del establecimiento, obteniendo así el listado con los productos y los planos.
- Lista de productos: La aplicación debe permitir a los usuarios acceder a una lista con los productos disponibles en el supermercado.
- Selección de productos: Los usuarios deben ser capaces de seleccionar productos específicos de la lista para visualizar su ubicación en el plano.
- Plano: Debe haber un plano donde se muestre la distribución del establecimiento, así como las coordenadas de los productos seleccionados.
- Reconocimiento de voz: La aplicación debe integrar un sistema de reconocimiento de voz que permita a los usuarios decir en voz alta el nombre del producto que desean localizar.
- Localización de productos: La aplicación debe ser capaz de localizar y resaltar en el plano la ubicación de los productos seleccionados por los usuarios, tanto manualmente como por voz.
- Ajustes: La aplicación debe poder cambiar entre modos de visualización de interfaz (día/noche) para adaptarse a diferentes condiciones de iluminación.

3.3. Requisitos no funcionales

A continuación, se muestran los requisitos no funcionales de la aplicación:

- Seguridad: Se tiene que garantizar la integridad de la información almacenada en la base de datos. Para eso debemos asegurarnos de que los únicos capaces de editar los datos de un supermercado (productos y planos) sean los administradores.
- Usabilidad: La interfaz debe ser intuitiva, fácil de usar y comprensible para usuarios de diferentes niveles de habilidad.

- Disponibilidad: La aplicación debe estar disponible para su uso en todo momento, con un tiempo de inactividad mínimo planificado para mantenimiento y actualizaciones.
- Adaptabilidad: La aplicación debe poder adaptarse a diferentes tamaños de dispositivos móviles y a diferentes orientaciones para garantizar una experiencia de usuario consistente.
- Internacionalización: La aplicación debe poder adaptarse a diferentes idiomas, siendo estos en un principio el inglés y el español.

3.4. Requisitos de la interfaz

La lista siguiente consiste en los requisitos de la interfaz:

- Interfaz intuitiva: La interfaz debe ser fácil de usar y comprensible para los usuarios, con un diseño claro y una navegación sencilla.
- Visualización del plano: El plano del supermercado debe mostrarse con una representación clara de las secciones, las entradas y los pasillos.
- Lista de productos: La lista debe presentarse clara y organizada, con la posibilidad de desplazarse y seleccionar elementos fácilmente.
- Localización de los productos: La interfaz debe mostrar la posición del producto en el plano de forma clara.

3.5. Requisitos del reconocimiento del habla

Los requisitos del reconocimiento del habla son los siguientes:

- Precisión del reconocimiento: El sistema de reconocimiento de voz debe tener una alta precisión para interpretar correctamente las palabras pronunciadas por los usuarios.
- Tiempo de respuesta: La aplicación debe tener una respuesta rápida al transcribir de voz a texto y localizar los productos en el plano.
- Manejo de errores: La aplicación debe ser capaz de manejar situaciones en las que no se reconozcan correctamente las palabras emitidas y proporcionar retroalimentación adecuada al usuario.

4. Diseño de la solución

Este capítulo se centra en el diseño de la solución propuesta. Comienza con la arquitectura, identificando los componentes de la aplicación y cómo interactúan. A continuación, se centra en la capa de presentación y en la capa de persistencia, comentando el diseño inicial de la interfaz y la gestión de datos para los servidores externos. Finaliza con las tecnologías utilizadas para el diseño del proyecto.

4.1. Arquitectura del sistema

La arquitectura general del sistema se ha diseñado siguiendo un enfoque cliente-servidor, en la cual los clientes hacen peticiones al servidor y éste les contesta con la información deseada. La arquitectura propuesta consta de los siguientes componentes principales:

- Cliente: La aplicación móvil actúa como la interfaz principal para los usuarios. Se ha implementado utilizando el entorno de desarrollo Android Studio, el IDE oficial para el desarrollo de aplicaciones para Android. La aplicación se encarga de la interacción con los usuarios, la presentación del plano y los productos del supermercado, el escaneo de códigos QR, el uso de RecognizerIntent, y la comunicación con la base de datos.

- Base de datos (Servidor): La aplicación se conecta a dos servicios ajenos, ambos utilizados a través de Firebase [2]:
    - Firestore Database: Sistema de base de datos en tiempo real, donde vamos a introducir los productos del supermercado junto con sus coordenadas en el plano.
    - Storage: Sistema de almacenamiento de imágenes y archivos, utilizado para almacenar los planos del supermercado.

- Comunicación: La comunicación entre cliente y servidor se realiza a través de la red utilizando API y servicios proporcionados por Firebase. El cliente envía y recibe datos de la base datos mediante el uso de bibliotecas específicas de Firebase.

- Reconocimiento de voz: RecognizerIntent es un componente del cliente que permite el reconocimiento de voz en el dispositivo del usuario. Cuando el usuario solicita la localización de los productos mediante audio, RecognizerIntent convierte la voz en texto, obteniendo así el nombre del producto a localizar.

En la Figura 6 podemos observar una representación de la arquitectura del sistema.

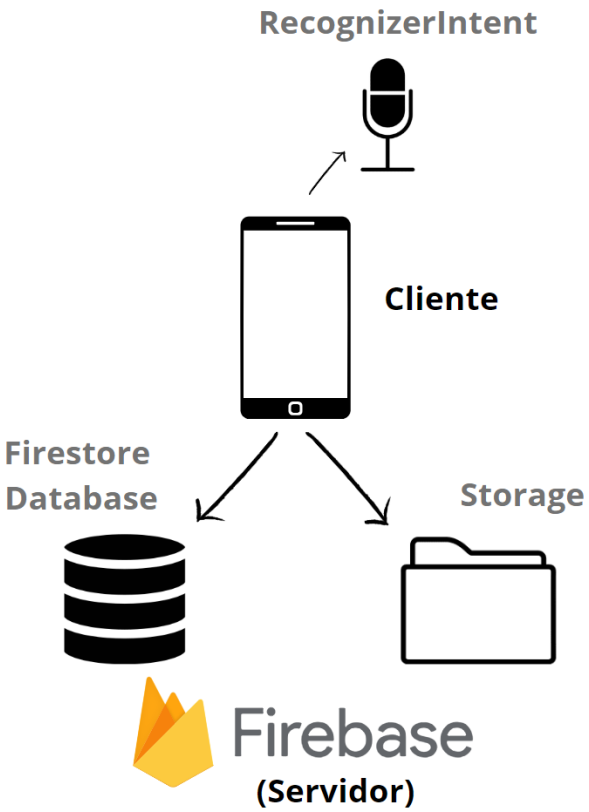

Figura 6: Arquitectura de la aplicación

La arquitectura de la aplicación consiste en un modelo de tres capas. Este enfoque de arquitectura se basa en la separación de las diferentes funcionalidades del sistema en tres capas distintas, cada una con un conjunto de tareas específicas: la capa de presentación, la capa de lógica de negocio y la capa de persistencia.

- Capa de presentación: Capa responsable de interactuar con los usuarios y proporcionar una interfaz intuitiva y receptiva. Esta capa es la interfaz de la aplicación, donde se expone la información y las funcionalidades que se pueden emplear.
- Capa de lógica de negocio: Capa que contiene la lógica y las reglas de negocio de la aplicación. Es la responsable de procesar las solicitudes del usuario, manipular los datos y realizar las operaciones necesarias para cumplir los requisitos funcionales de la aplicación. Es decir, son las funciones y servicios que usa la *app*.
- Capa de persistencia: Capa de almacenamiento de datos, responsable de la gestión y persistencia de los datos utilizados por la aplicación. Almacena y recupera la información necesaria para la funcionalidad de la aplicación, como las coordenadas de los productos y los planos del supermercado.

4.2. Capa de presentación

En este apartado nos centraremos en la creación y visualización de los bocetos de la interfaz, mostrando la disposición de los elementos, la navegación y las interacciones. Los bocetos nos ayudarán a comprender cómo se verá y funcionará la interfaz de la aplicación, permitiéndonos obtener una idea clara de la experiencia del usuario. Todos los bocetos se realizaron con

Wondershare Mockitt [57], una herramienta de gestión de proyectos desarrollada por Wondershare [58] que permite diseñar prototipos.

4.2.1 Escáner QR

La Figura 7 representa la pantalla de escaneo de códigos QR de la *app*. Es la primera pantalla a la que se accede al iniciar la aplicación, ya que el código QR es necesario para la descarga del plano y los productos del supermercado.

Para escanear un código basta con pulsar el botón "Escanear QR" y se activará la cámara del dispositivo. Una vez detectado el QR, se saldrá de la función de cámara y regresaremos a la pantalla del principio.

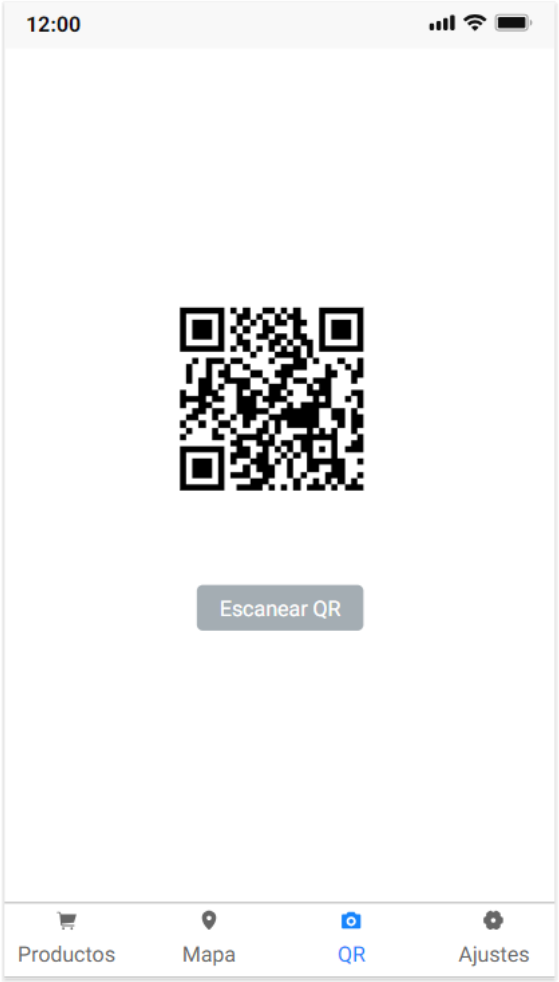

Figura 7: Boceto escáner QR

4.2.2 Plano supermercado

La Figura 8 consiste en la pantalla encargada de mostrar el plano del supermercado. Al iniciar la aplicación se verá una pantalla en blanco, pero tras escanear un código QR válido mostrará la imagen del plano del establecimiento obtenida desde la base de datos.

Esta es la pantalla en la que se visualizará la localización de todos los productos, ya sean seleccionados manualmente (Figura 9) o por audio. Para poder buscar un producto por audio, basta con pulsar el botón con el icono que se ve en la Figura 8

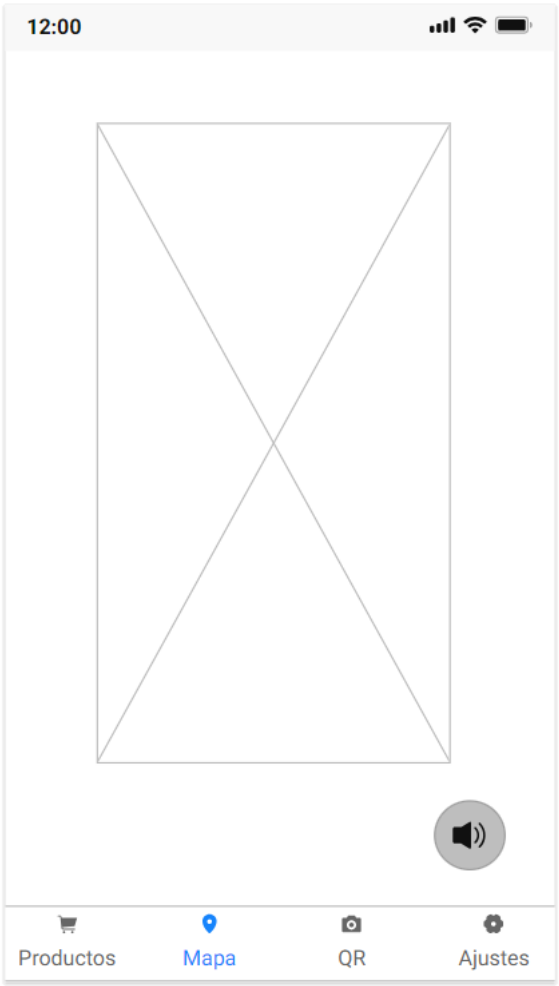

Figura 8: Boceto plano supermercado

4.2.3 Listado de productos

La Figura 9 muestra un listado con todos los productos disponibles en el supermercado. Al igual que en la pantalla con el plano del supermercado, ésta se mostrará vacía hasta que se escanee un QR válido.

En caso de pulsar uno de los botones que se encuentran al lado de los productos, se activará un mensaje de confirmación sobre la búsqueda del producto seleccionado. Si se pulsa el botón "Aceptar", la aplicación se desplazará a la pantalla con el plano del establecimiento (Figura 8), indicando la ubicación del producto seleccionado.

Figura 9: Boceto listado de productos

4.2.4 Ajustes

La Figura 10 representa la pantalla de ajustes de la aplicación, es decir, la pantalla desde la que se puede cambiar la visualización de la pantalla al modo nocturno y viceversa. Para modificar la visualización basta con pulsar el *switch* de la Figura 10.

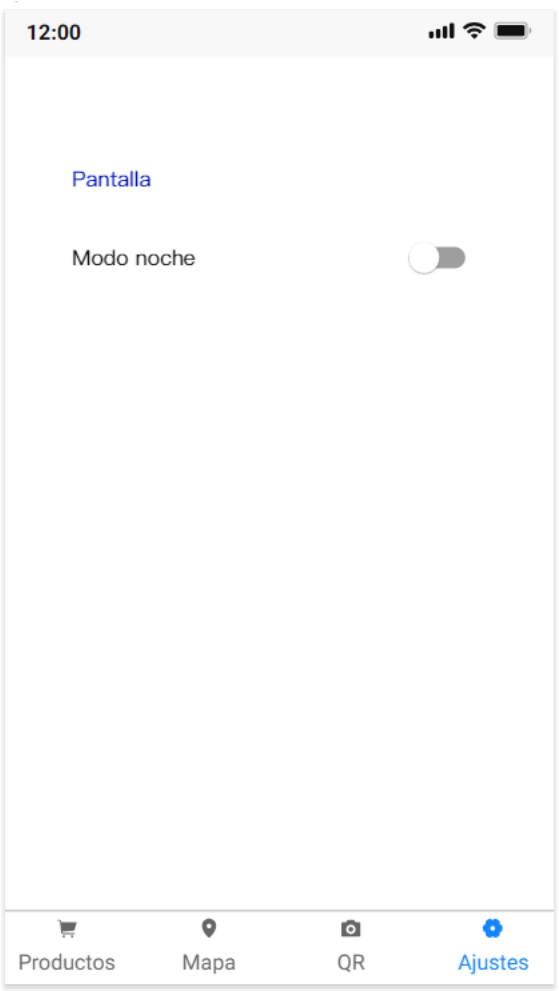

Figura 10: Boceto ajustes

4.3. Capa de persistencia

En este apartado, se mostrará el modelo de datos que vamos a utilizar para almacenar la información.

Como ya se informó en apartados anteriores, Firebase Storage se utilizará solo para almacenar los planos del supermercado. Dado que la aplicación se limita a extraer estos archivos y estos no tienen ningún parámetro destacable, no vamos a profundizar en ellos.

En cuanto a Firestore Database, la usaremos para almacenar los productos del supermercado junto con sus características. Como podemos ver en la Figura 11, el producto está formado por 6 parámetros:

- id: Nombre del producto, sin mayúsculas ni tildes. Será el parámetro que se use a la hora de localizar productos mediante órdenes de voz.
- nom_producto: Nombre del producto tal y como debe aparecer en la interfaz de usuario.
- cord1: Coordenada horizontal del producto en el plano, para la orientación vertical del dispositivo. Todas las coordenadas son porcentajes.

- cord2: Coordenada vertical del producto en el plano, para la orientación vertical del dispositivo.
- cord11: Coordenada horizontal del producto en el plano, para la orientación apaisada del dispositivo.
- cord22: Coordenada vertical del producto en el plano, para la orientación apaisada del dispositivo.

Figura 11: Esquema productos

La tabla que se puede observar a la derecha en Figura 11 es un ejemplo de la representación de un producto en Firestore Database.

Los productos y planos de un establecimiento estarán todos guardados en colecciones con el mismo nombre (el nombre del establecimiento). Al escanear el QR a la entrada del supermercado se obtendrá el nombre de la colección a la que debe acceder la aplicación, garantizando así que los únicos capaces de modificar los datos sean los administradores y que los usuarios puedan acceder automáticamente a la información.

Además, la colección del establecimiento en Firestore Database se dividirá en tantos documentos como idiomas habilitados tenga la aplicación. En un principio habrá dos documentos, uno con los productos en español y otro con los productos traducidos al inglés.

4.4. Tecnología utilizada

En este apartado se listan las tecnologías utilizadas a lo largo del proyecto. Las dividiremos en tecnologías y *frameworks* utilizados para crear la aplicación, para gestionar la base de datos y para control de versiones.

4.4.1 Tecnología para desarrollo de aplicaciones móviles

Tecnologías utilizadas para crear nuestra aplicación Android.

- Android Studio:

  Android Studio [3] es un entorno de desarrollo integrado desarrollado por Google enfocado en la creación de aplicaciones móviles para dispositivos Android. Proporciona un conjunto completo de herramientas y recursos para el desarrollo de aplicaciones, incluyendo un editor de código.

  Android Studio es el entorno de desarrollo que hemos empleado en este proyecto. Su elección se basa en varios factores fundamentales. En primer lugar, nos permite una integración fluida con herramientas como GitHub y Firebase. Además, es el entorno de desarrollo oficial para la plataforma Android y fue el entorno de desarrollo empleado en DADM (Desarrollo de Aplicaciones para Dispositivos Móviles), una de las asignaturas de la carrera.

- Kotlin:

  Kotlin [12] es un lenguaje de programación moderno y conciso desarrollado con el objetivo de ofrecer una alternativa más segura, expresiva y fácil de usar a otros lenguajes de programación, especialmente en el contexto de desarrollo de aplicaciones para Android.

  Ha ganado popularidad debido a su integración perfecta con Android Studio y su amplio soporte por parte de Google. Desde mayo de 2017, Kotlin se ha convertido en el lenguaje oficial para el desarrollo de aplicaciones Android, motivo por el cual decidimos usarlo como lenguaje principal para el desarrollo de nuestra aplicación.

- Magicplan:

  Magicplan [60] es una aplicación dirigida a profesionales de diferentes campos, como remodelación, restauración, arquitectura y diseño de interiores. Ofrece tecnologías de realidad aumentada e inteligencia artificial para mapear rápidamente habitaciones y crear planos de planta. Actualmente se encuentra disponible para dispositivos Android y Apple.

  Los usuarios de la aplicación pueden dibujar manualmente los planos de planta, como podemos ver en la Figura 12, así como cambiar la visualización del plano de 2D a 3D, entre otras muchas funciones. En nuestro caso, magicplan se usará para el diseño del plano de un supermercado ficticio (Figura 12).

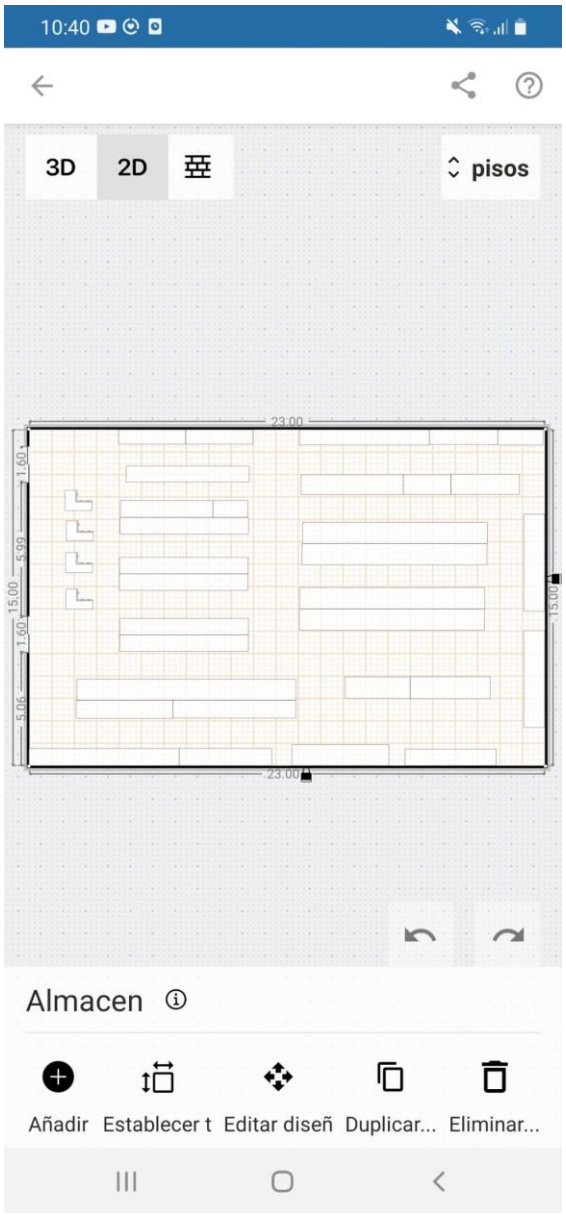

Figura 12: magicplan

- Wondershare Mockitt:

  Wondershare Mockitt es una herramienta de diseño y prototipado de interfaces de usuario basada en la nube. Permite crear rápidamente bocetos interactivos, prototipos y *wireframes* para aplicaciones web y móviles.

  Esta herramienta se ha usado principalmente para diseñar los bocetos iniciales de la interfaz.

- Canva:

  Canva [65] es una herramienta de diseño gráfico amigable para principiantes fácilmente accesible tanto en computadoras como teléfonos móviles.

  Sus usuarios pueden guardar, organizar y compartir sus diseños, además de personalizar plantillas, buscar elementos, agregar texto y cargar imágenes, entre otras funciones.

Canva se ha empleado para el diseño de muchas de las figuras incluidas en este documento.

- XML:

  XML (*Extensible Markup Language*) [61] es un lenguaje utilizado ampliamente para definir la estructura y el diseño de las interfaces de usuario de las aplicaciones Android.

  XML se utiliza en Android Studio para crear archivos de diseño que describen la disposición y apariencia de los elementos de la interfaz de usuario.

- Java:

  Java [11] es un lenguaje de programación orientado a objetos, versátil y de alto nivel [56] que es ampliamente utilizado en aplicaciones móviles, de escritorio y web, así como videojuegos y conexiones a bases de datos.

  Su popularidad se debe a su compatibilidad multiplataforma, facilidad de aprendizaje, naturaleza de código abierto, seguridad y sólido apoyo de la comunidad. Además, al ser un lenguaje orientado a objetos, ofrece estructuras claras y permite la reutilización de código.

  Java se usa para implementar ciertos aspectos de la aplicación, como el escaneo de QR y el reconocimiento de voz.

4.4.2 Tecnología para bases de datos

La tecnología utilizada para almacenar y gestionar los datos de la aplicación es Firebase.

Firebase es una plataforma de desarrollo de aplicaciones que ofrece una amplia gama de servicios y herramientas para facilitar el desarrollo de aplicaciones web y móviles.

Algunos de los principales servicios de Firebase, y los que vamos a usar en nuestro proyecto, son Firestore Database y Storage.

Firestore Database es una base de datos de tiempo real que permite almacenar y sincronizar datos entre los clientes y el servidor. Ofrece un modelo de datos jerárquico con colecciones y documentos y almacena los datos en formato JSON [62]. Los datos almacenados se pueden acceder y manipular fácilmente mediante consultas.

Usaremos esta base de datos para almacenar los productos del supermercado, los cuales solo podrán ser modificados por los administradores y se podrá acceder a ellos tras escanear un código QR válido.

Firebase Storage es un servicio de almacenamiento en la nube que permite a los desarrolladores almacenar y compartir archivos, como imágenes y vídeos. En nuestro caso, almacenaremos los planos del supermercado. Al igual que con Firestore Database, los únicos capaces de modificar los datos en Storage son los administradores.

4.4.3 Tecnología para control de versiones

La Herramienta usada durante el desarrollo de la aplicación como sistema de control de versiones es GitHub.

GitHub [59] es una plataforma de desarrollo colaborativo basada en la nube que permite a los equipos de desarrollo trabajar juntos en proyectos de software. Proporciona un sistema de control de versiones distribuido, es decir, permite a los desarrolladores rastrear los cambios en el código fuente a lo largo del tiempo.

Dicho sistema de control de versiones es el motivo por el que decidimos utilizar la plataforma para el proyecto. En caso de algún error, se puede volver a la versión previa del código almacenada en GitHub.

5. Desarrollo de la solución propuesta

El desarrollo de la solución propuesta ha sido llevado a cabo siguiendo rigurosamente las pautas y estándares establecidos en el capítulo 4. En este capítulo se comentarán los problemas identificados, las decisiones adoptadas y las dificultades encontradas durante el proceso de desarrollo de la aplicación. Se presentará la interfaz final y se describirá su funcionalidad, así como las particularidades de algunas de las implementaciones realizadas.

Comenzaremos con el diseño del plano y de la base de datos, para finalizar con una explicación detallada de la implementación de funcionalidades clave, como el escaneo de QR o el reconocimiento de voz.

5.1. Diseño del plano

El primer paso realizado durante la implementación de la aplicación consistió en el diseño del plano del supermercado. Dado que, como se explicó en el capítulo 2, no fue posible establecer convenio con ningún supermercado para obtener acceso a sus planos y productos, se optó por crear un supermercado ficticio.

En este proceso, nos enfrentamos a la mayor dificultad al intentar encontrar una página web o aplicación gratuita capaz de diseñar planos de supermercado realistas. Tras una exhaustiva búsqueda, escogimos la aplicación magicplan. En la Figura 13 podemos ver el plano diseñado utilizando las herramientas proporcionadas por magicplan. A la izquierda, se encuentra el plano original en modo de visualización 2D, mientras que, a la derecha, se muestra la versión 3D del mismo plano. Para mejorar la calidad del plano 3D, se empleó una IA llamada Zyro [66].

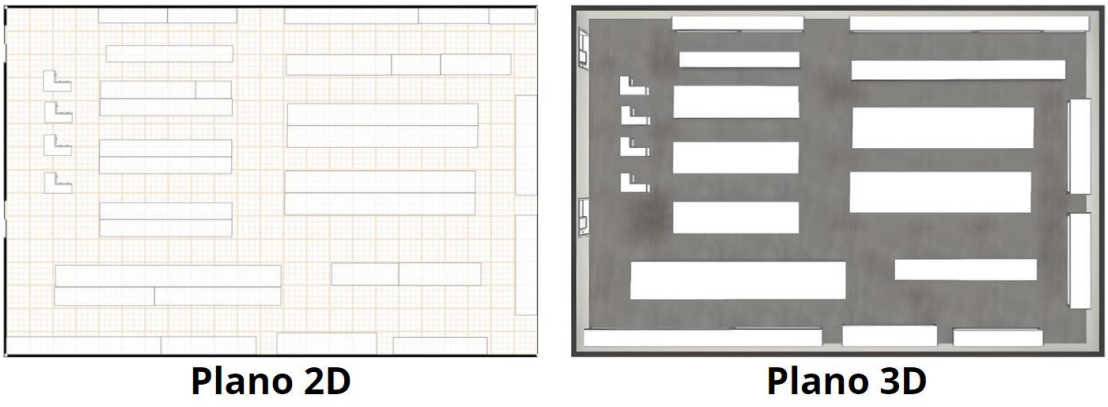

Figura 13: Planos iniciales del supermercado

En la etapa final del diseño del plano, se tomó la decisión de procesar el plano 3D mediante un editor de imágenes para agregar color. Este enfoque permitió diferenciar visualmente los

estantes con los productos, aportando una mayor claridad y distinción entre ellos. Además, la adición de color contribuyó a infundir más vitalidad y atractivo al diseño global del plano.

En la Figura 14 se presentan las dos versiones finales del plano, una para la orientación vertical del dispositivo y otra para la apaisada.

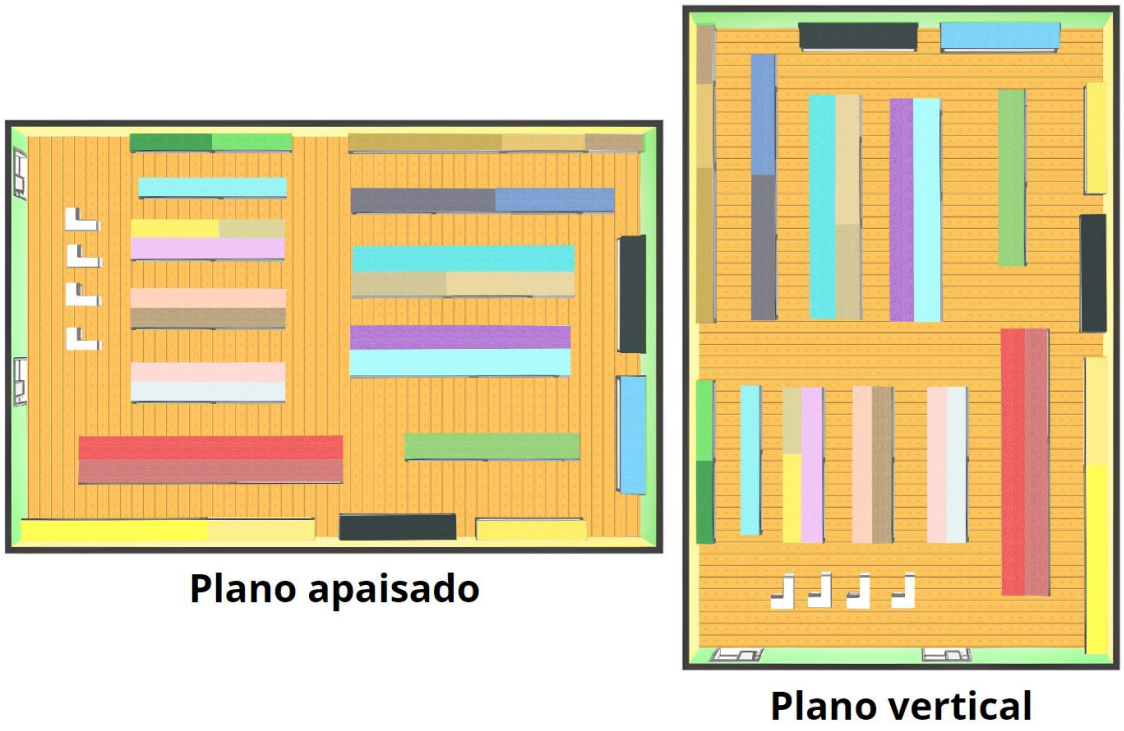

Figura 14: Planos finales del supermercado

5.2. Diseño de la base de datos

El siguiente paso en el desarrollo del proyecto consiste en la creación de una base de datos capaz de almacenar los productos y los planos del supermercado ficticio, y de acceder a ellos mediante el escaneo de un código QR.

Tras una extensa búsqueda, decidimos usar Firebase como base de datos, como se explicó en apartados anteriores. La lógica para el acceso a la base de datos es la siguiente: al vincular Android Studio, plataforma que usamos para desarrollar la aplicación, con Firebase, basta con obtener el nombre de la colección o carpeta donde tenemos almacenados los datos para poder extraerlos.

Tras una breve búsqueda, decidimos usar la página web QR Code Generator [67] para la creación del código QR que nos permitirá el acceso a la base de datos. Este QR, que podemos observar en la Figura 15, consiste en un *String* con el nombre del establecimiento (en nuestro caso, "*market*"), que a su vez será el nombre de las colecciones y carpetas en las que tendremos almacenados los datos del supermercado.

Figura 15: Código QR del supermercado

Las imágenes con los planos se almacenaron manualmente en Firebase Storage, como se puede ver en la Figura 16. Ambos están dentro de la carpeta "*market*", nombre que obtendremos al escanear el código QR.

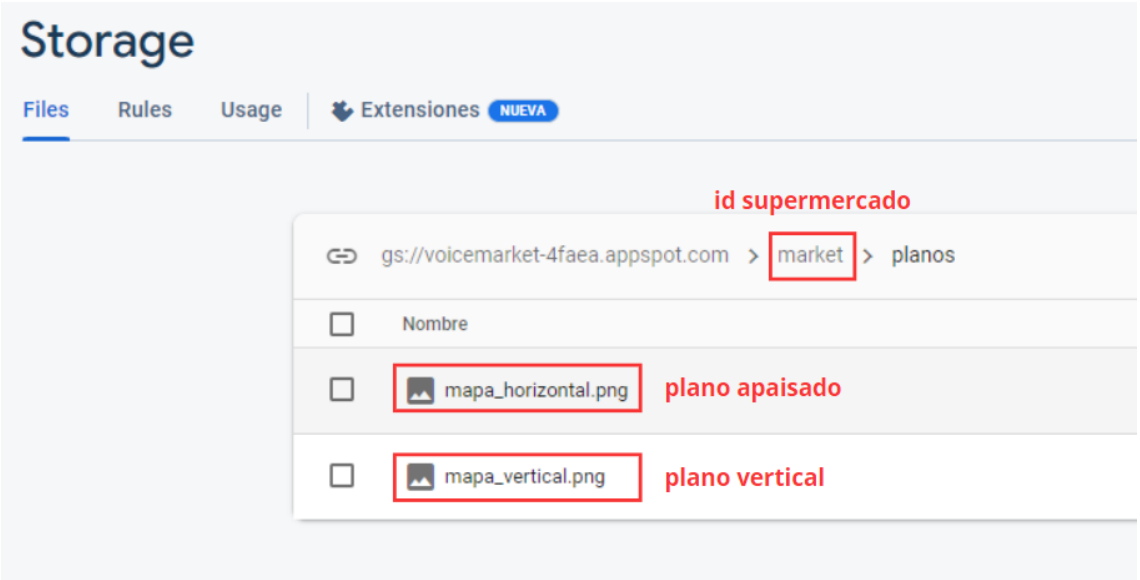

Figura 16: Firebase Storage

En cambio, los productos y sus características, descritas en el Capítulo 4, se introducirán a través de Android Studio en Firestore Database. Como podemos ver en la Figura 17, la colección tiene el mismo nombre que la carpeta de Storage y que el QR, pero en este caso se divide en dos documentos. El documento "en" son los productos traducidos al inglés, asegurando así la internacionalización de la *app*, mientras que el documento "es" son los productos en español. A la hora de extraer los datos, ambas listas de productos se almacenarán en variables diferentes, cambiando los productos mostrados en la interfaz en función del idioma predeterminado del dispositivo.

Figura 17: Colección e idiomas en Firestore Database

Finalmente, en la Figura 18 se observan los atributos de los productos, en este caso de un producto del documento "es". Los productos en el documento "en" tienen las mismas coordenadas, modificando solo el id y el nombre del producto.

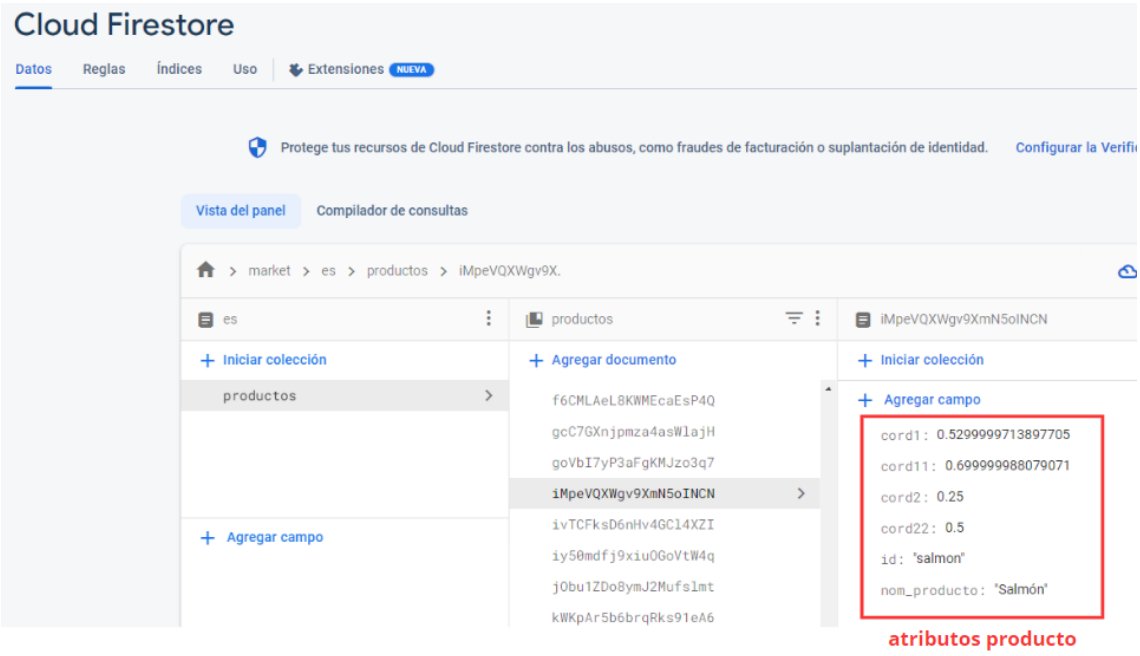

Figura 18: Atributos de los productos en Firestore Database

5.3. Implementación del escaneo de QR

Una vez implementado el servidor (bases de datos), nos enfocamos en la interfaz y el desarrollo de la aplicación. En primer lugar, estructuramos y organizamos la interfaz de usuario (UI) tal y como se puede ver en la Figura 19.

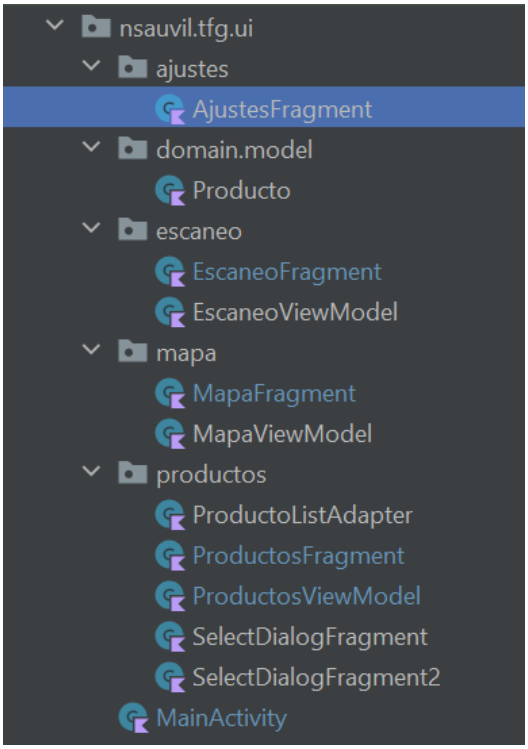

Figura 19: Estructura de carpetas

En este apartado nos centraremos en la pantalla de escaneo de códigos QR. Es decir, vamos a desarrollar la clase `EscaneoFragment` de la Figura 19.

La interfaz final de esta pantalla se verá como la Figura 20, con un `bottomNavigationView` que permite al usuario navegar entre todos los fragmentos (Ajustes, Productos, Mapa y QR). Para poder activar la cámara y escanear el código QR, basta con pulsar el botón “Escanear QR”. Para informar a los usuarios de que la cámara ha detectado un código, el mensaje cambiará por “Escanear otro QR”.

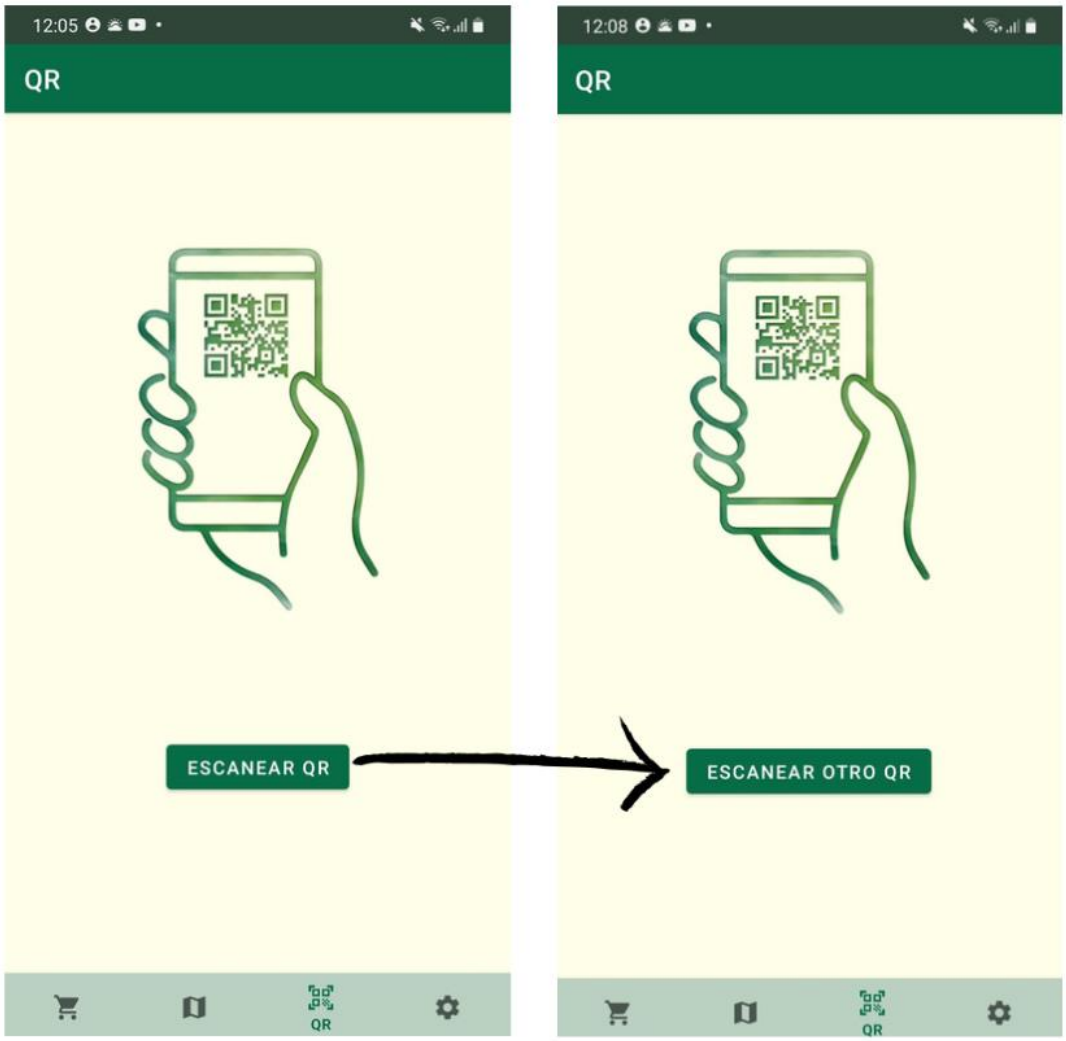

Figura 20: Interfaz final para el escáner de QR

La búsqueda de una API capaz de escanear y reconocer códigos QR supuso un gran reto, ya que es una de las funcionalidades principales de la aplicación. Pero tras una extensa búsqueda logramos la implementación de la funcionalidad de escaneo de códigos QR mediante el uso de la biblioteca ZXing.

ZXing (*Zebra Crossing*) [68] es una API de código abierto ampliamente utilizada para integrar el procesamiento de códigos QR en aplicaciones, aunque también ofrece soporte para otros tipos de códigos de barras. Está basada en Java y permite que la *app* pueda leer e interpretar los QR.

Como se puede observar en la Figura 21, primero creamos una instancia de la clase `IntentIntegrator` (clase que proviene de la biblioteca ZXing), ajustando partes del código como el formato de código de barras deseado (`IntentIntegrator.QR_CODE`), el mensaje que se mostrará al usuario durante el proceso de escaneo, la orientación de la cámara y la desactivación del sonido de escaneo. Finalmente, se inicia el proceso de escaneo con `initiateScan()`.

```
private fun startQRScanner() {
    val integrator = IntentIntegrator.forSupportFragment( fragment: this)
    integrator.setDesiredBarcodeFormats(IntentIntegrator.QR_CODE)
    integrator.setPrompt( "Escanear QR")
    integrator.setCameraId(0) // Usar la cámara trasera del dispositivo
    integrator.setBeepEnabled(false) // Desactiva el sonido de escaneo
    integrator.initiateScan()
}
```

Figura 21: Código para activar la cámara

La clase `onActivityResult` es la encargada de manejar el resultado obtenido por el escáner de códigos QR. Es decir, en esta clase se accede a las bases de datos, obteniendo la lista con los productos y los planos del supermercado.

En primer lugar, se vuelve a usar la biblioteca ZXing para analizar el resultado del escaneo. Si el resultado es no nulo, se obtiene una instancia de FirebaseFirestore, la base de datos en la que están almacenados los productos, y se intenta acceder a la colección con el nombre `qrCodeResult`. Si dicha colección no existe, se mostrará el mensaje de error "Código QR no válido", mientras que si `qrCodeResult` coincide con la colección que estamos buscando (es decir, si el código escaneado tiene el nombre del establecimiento), la aplicación accederá a los documentos con los productos tanto en inglés como en español.

En cuanto a los planos, se crea una instancia de FirebaseStorage, al igual que con FirebaseFirestore, y se accede a las carpetas con las imágenes.

Para la gestión de los productos hemos creado la clase de datos que se observa en la Figura 22. Los documentos extraídos de Firestore Database se convertirán en objetos de tipo Producto.

```
data class Producto(val id:String, val nom_producto:String, val cord1:Float, val cord2:Float, val cord11:Float, val cord22:Float){
    nsauvil
    constructor(): this( id: "", nom_producto: "", cord1: 0.0f, cord2: 0.0f, cord11: 0.0f, cord22: 0.0f)
}
```

Figura 22: Clase de datos Producto

Es decir, una vez obtenidos los documentos de Database, se recorrerán de uno en uno, convirtiéndose en objetos de tipo Producto y almacenándose en una lista que, tras convertirse todos los documentos en Productos, se asignará a la interfaz. Los productos del supermercado se mostrarán en el fragmento Productos, ordenados alfabéticamente.

En el caso de las imágenes, ambas (apaisada y vertical) obtendrán su URL de descarga para poder cargar los planos en los `ImageView` del fragmento Mapa.

La carga de las imágenes obtenidas fue otro de los puntos problemáticos durante el desarrollo del proyecto. Se probó con bibliotecas como Glide [69] y Picasso [70], pero leves problemas a la hora de su implementación nos llevaron a la aplicación de Coil.

La biblioteca Coil se usa para la carga y visualización de los planos [71]. Coil es una librería de carga de imágenes que ofrece una solución eficiente y ligera para la manipulación de imágenes en aplicaciones Android, aprovechando las características de Kotlin y otras bibliotecas modernas. Permite la creación de instancias personalizadas del `ImageLoader` para ejecutar las solicitudes de imágenes de manera eficiente.

El código de la Figura 23 aplica esta biblioteca. Comienza creando un objeto `ImageRequest` con el `Builder` de Coil, configurando la solicitud de carga de imágenes al indicar el recurso a cargar (`data(it)`). El destino donde se mostrará la imagen cargada se determina con `target(binding.map1)`, siendo `map1` el `ImageView` del fragmento Mapa en su orientación vertical. Una vez configurados todos los parámetros de la solicitud, se finaliza llamando al `build()`.

La función termina al encolar la solicitud de carga (`enqueue(request)`) y habilitar la visibilidad tanto del `ImageView` como del botón de reconocimiento del habla.

```
viewModel2.imagenV.observe(viewLifecycleOwner) { it: String! //imagen escaneada co
    if (!it.isNullOrEmpty() && binding.map1 != null) {
        val request = ImageRequest.Builder(requireContext())
            .data(it)
            .target(binding.map1!!)
            .build()
        val disposable = Coil.imageLoader(requireContext()).enqueue(request)
        binding.map1?.visibility = View.VISIBLE
        binding.floatingActionButton.visibility = View.VISIBLE
    }
}
```

Figura 23: Código para descargar las imágenes

5.4. Implementación de la selección manual de productos

Este apartado se centra en la pantalla de selección manual de productos, es decir, en el fragmento Productos. Desarrollaremos los aspectos más importantes de las clases del paquete `productos` mostrado en la Figura 19.

La interfaz final de este fragmento se puede observar en la Figura 24, donde los productos se muestran en un `RecyclerView` formado por `MaterialCardViews`, cada una con un `MaterialTextView` para el nombre del producto y un `MaterialButton` para el botón asociado al producto. `RecyclerView` es un componente de interfaz de usuario que se usa para mostrar listas o conjuntos de datos dinámicos de manera eficiente, mientras que `MaterialCardView` muestra el contenido en forma de tarjetas.

Al pulsar sobre los botones se activa un `DialogFragment` para que los usuarios confirmen si desean localizar el producto seleccionado y acceder al mapa del establecimiento.

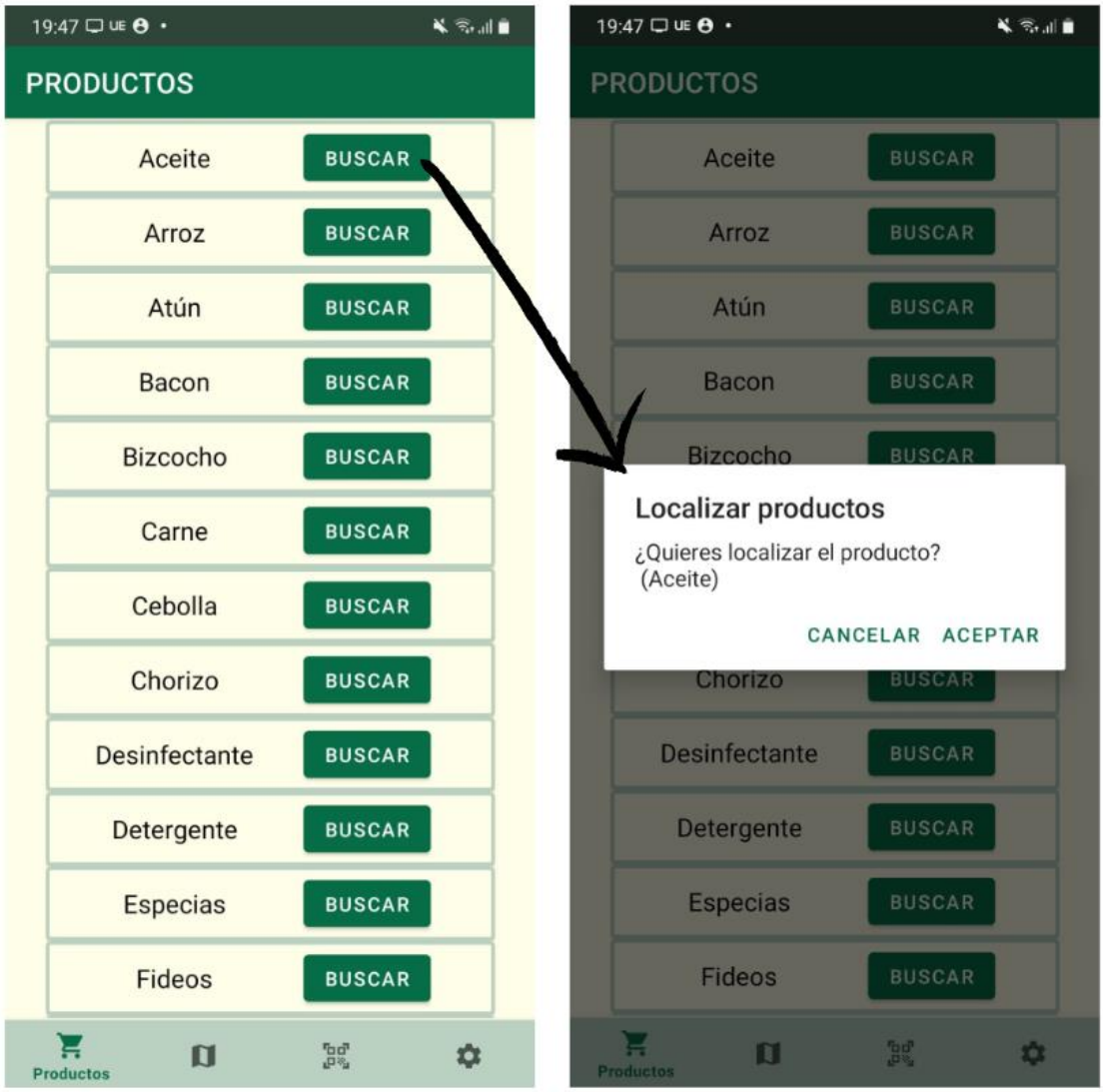

Figura 24: Interfaz final para la selección de productos

Una de las grandes dificultades de este apartado fue conseguir pasar el producto seleccionado al `DialogFragment`, ya que no se pulsa sobre el producto sino sobre el botón que se encuentra a su lado. Se solucionó mediante la implementación del control de la actividad del botón en `ProductoListAdapter`, en vez de en `ProductosFragment`.

Como se recomendó en DADM, una asignatura sobre aplicaciones móviles cursada en la carrera, se implementó un `ViewHolder` en la clase `ProductoListAdapter`. Un `ViewHolder` es una clase que representa una única vista de elemento dentro de un `RecyclerView`, optimizando el rendimiento al reutilizar las vistas en lugar de crear nuevas cada vez que se muestra un elemento en la lista.

En la función `bind` se realiza la vinculación de datos entre el objeto Producto y las vistas del `ViewHolder`. Al `TextView` se le asigna el parámetro `nom_producto` y se gestiona la activación de los `DialogFragment` al pulsar los botones.

Una vez activado el `DialogFragment`, si se pulsa el botón “Aceptar”, el producto seleccionado se transfiere a `ProductosViewModel` para gestionar su localización desde `MapaFragment`. Tras pasar el producto a la otra clase, se navega hasta el fragmento Mapa, donde se mostrará el producto localizado sobre el plano.

5.5. Implementación de la búsqueda de productos

Este apartado desarrolla la que seguramente es la funcionalidad principal de la aplicación, la localización de productos. En este caso, la complejidad deriva de cómo adaptar las coordenadas de los productos tanto para dispositivos de varios tamaños como para ambas orientaciones, la vertical y la apaisada, ya que en apaisado usamos el plano horizontal y, por lo tanto, las coordenadas cambian.

Los productos se pueden buscar de dos formas distintas: manualmente o por audio. Los productos seleccionados manualmente se obtienen a través del `DialogFragment`, como explicamos en la sección 5.4, mientras que los emitidos por audio se consiguen mediante una transcripción de audio a texto, la cual detallaremos en la sección 5.6. Lo importante es que ambos realizan el mismo proceso para obtener las coordenadas en el plano.

La lógica para obtener las coordenadas y que éstas se adapten al tamaño del dispositivo es la siguiente: en vez de determinar una posición fija para el producto en el plano, se le asignan dos porcentajes, uno para el eje x (“`cord1`”) y otro para el eje y (“`cord2`”). Cada vez que se localiza un producto, lo que hacemos es multiplicar el porcentaje por el alto y ancho del plano, sumando la posición en la que se encuentra la esquina superior izquierda de la imagen. De esta forma, incluso sin saber cuánto mide el plano, las coordenadas se adaptarán en tiempo de ejecución.

En la Figura 25 podemos ver el código descrito, además de una comprobación adicional (`binding.map1?.let`) para asegurarnos de que el mapa no es nulo, caso en el que el acceso a las dimensiones del plano daría error. Todos estos cálculos se realizan en `MapaFragment` (Figura 19).

```
private fun calculateCoordinatesV(prod:Producto): Pair<Float, Float> { //
    binding.map1?.let { mapView ->
        if (mapView.width > 0 && mapView.height > 0) {
            val mapWidth = mapView.width.toFloat()
            val mapHeight = mapView.height.toFloat()
            val mapStartX = mapView.left.toFloat()
            val mapStartY = mapView.top.toFloat()
            val locationXPercentage = prod.cord1
            val locationYPercentage = prod.cord2
            val locationX = mapStartX + locationXPercentage * mapWidth
            val locationY = mapStartY + locationYPercentage * mapHeight
            return Pair(locationX, locationY)
        }
    }
    return Pair(prod.cord1, prod.cord2)
}
```

Figura 25: Código para calcular las coordenadas

Una vez obtenidas las coordenadas, el puntero (`marca1`) se posiciona mediante una translación al origen de coordenadas, seguida de otra translación a la posición calculada en `calculateCoordinates`. El código descrito se encuentra en la Figura 26.

```
private fun moveImageToLocation(producto: Producto) {
    val coordenadasV = calculateCoordinatesV(producto)
    binding.marca1?.let { marca1 ->  //orientación vertic
        marca1.translationX = 0.0f   //asegurarnos de que
        marca1.translationY = 0.0f
        marca1.translationX = coordenadasV.first
        marca1.translationY = coordenadasV.second
        marca1.visibility = View.VISIBLE
    } //comprueba que no haya null
}
```

Figura 26: Código para posicionar el puntero

La Figura 27 muestra cómo se verá la interfaz final del fragmento Mapa. A la derecha se resaltan las coordenadas del producto “Detergente”, como ejemplo de la localización de productos.

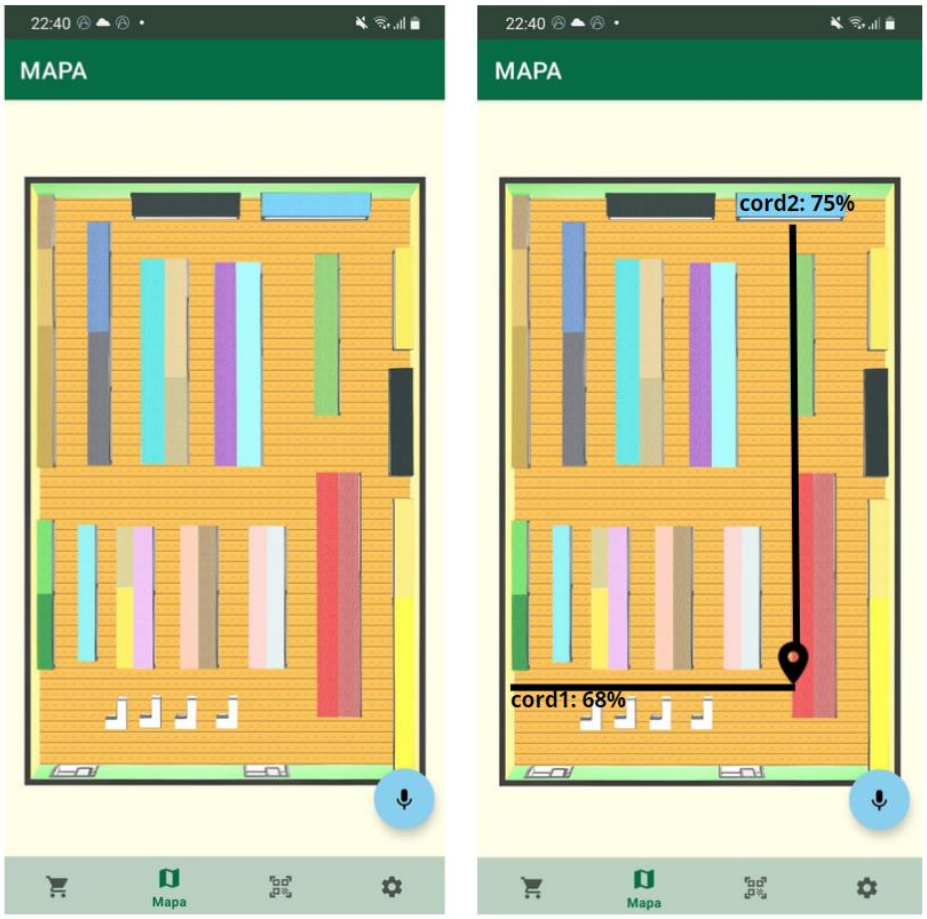

Figura 27: Interfaz final para la localización de productos

Finalmente, para la orientación apaisada se realizará el mismo cálculo que en las Figuras 25 y 26, pero las coordenadas (`cord11` y `cord22`) serán distintas. No se realizó ningún cálculo o escala para obtener esta diferencia, sino que se seleccionaron de tal forma que ambos punteros señalen a la misma zona, como se puede ver en la Figura 28.

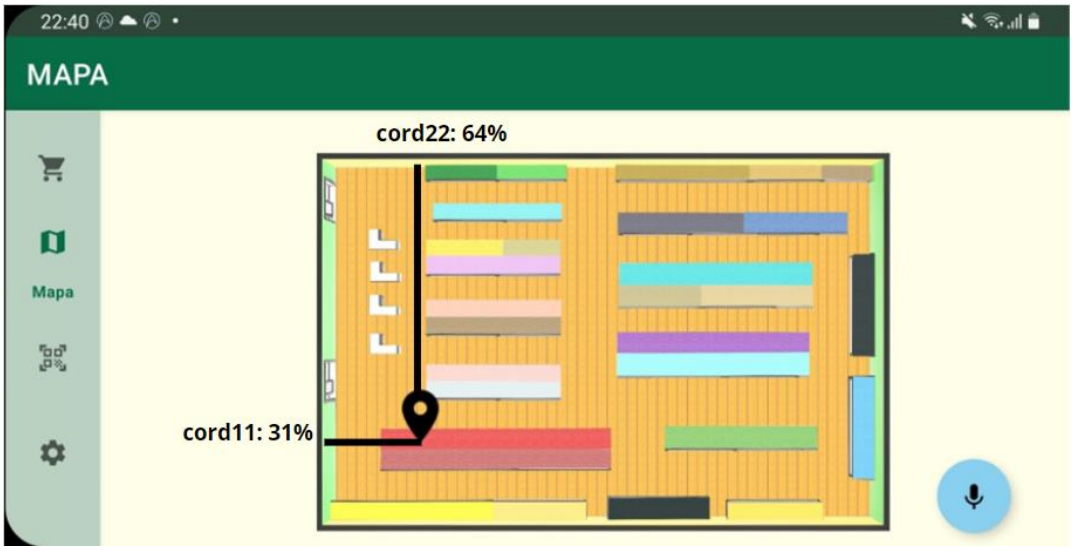

Figura 28: Interfaz final para la localización de productos en apaisado

5.6. Implementación del reconocimiento del habla

Finalmente, llegamos a la función que más conflictos ha generado a lo largo del proyecto, el reconocimiento del habla. En un principio, el objetivo era almacenar el audio y transmitirlo a un servidor remoto que contaba con la herramienta iATROS y el sistema operativo Linux, pero restricciones a la hora de guardar el audio en el almacenamiento interno del dispositivo, así como complicaciones en la conexión HTTP con el servidor, dieron lugar al descarte de ese planteamiento y a la búsqueda de alternativas.

Tras otra extenuante búsqueda se halló el RecognizerIntent que, como indicamos en el capítulo 2, es una clase de Android implementada en Java que proporciona una interfaz para utilizar el servidor de reconocimiento de voz del sistema operativo. El reconocimiento de voz se implementa, al igual que la localización de productos, en el `MapaFragment`.

Como podemos ver en la Figura 29, al pulsar el botón de audio se crea un `Intent` con la acción `ACTION_RECOGNIZE_SPEECH`, indicando así que se desea iniciar la actividad de reconocimiento de voz. En el código también configuramos parámetros extras como el modelo del lenguaje, que en este caso no tiene restricciones (`LANGUAGE_MODEL_FREE_FORM`) y el mensaje que se mostrará en pantalla mientras se capta el audio. Finalmente, se inicia la actividad con `startActivityForResult()`.

```
recordButton = view.findViewById(R.id.floatingActionButton)
recordButton.setOnClickListener{ it: View!
    val intent = Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH) //iniciar actividad reconocimiento d
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
    intent.putExtra(RecognizerIntent.EXTRA_PROMPT, getString(R.string.audioRecord))
    startActivityForResult(intent, RECOGNIZER_CODE)
}
```

Figura 29: Código para implementar el reconocimiento de voz

Una vez obtenidos los resultados del reconocedor de voz, el `resultCode` debería ser `RESULT_OK` si la actividad se ejecutó correctamente. Si eso es cierto, y los datos recogidos no son nulos, se obtiene el resultado de la transcripción de voz como una lista de cadenas y se almacena la primera (índice 0).

Para identificar el producto que se corresponde con el audio transcrito, primero se lleva a cabo un procesamiento sobre `pal`, la palabra obtenida por el reconocedor. Nos aseguramos de que esté en minúsculas y de que no tenga tildes, para poder localizar el producto seleccionado mediante un bucle que recorrerá todos los productos obtenidos de la base de datos hasta encontrar uno cuyo `id` coincida con la palabra procesada. En caso de no encontrar ningún producto, se emitirá el mensaje de error “El producto no se encuentra en el supermercado”.

Este mensaje de error se usa para minimizar una de las desventajas de RecognizerIntent, el hecho de que no puede restringir fácilmente el modelo de lenguaje para que solo reconozca los productos seleccionados.

Además, los productos se encuentran almacenados en dos listas distintas, una por cada idioma reconocido por la aplicación. Se llevó a cabo una identificación previa del lenguaje predeterminado del dispositivo con el fin de iterar únicamente sobre la lista de productos correspondiente a dicho idioma. En otras palabras, el reconocedor de voz es capaz de reconocer palabras en el idioma predeterminado, lo cual representa uno de los fundamentos que nos llevaron a escoger RecognizerIntent para este proyecto.

Para finalizar, la Figura 30 refleja la interfaz final para la localización de productos mediante el reconocedor del habla. La pantalla izquierda muestra el mensaje e icono que surgen tras pulsar el botón del audio. En cuanto a la imagen derecha, una vez reconocida la palabra, se activará un `DialogFragment` similar al mostrado en la Figura 24.

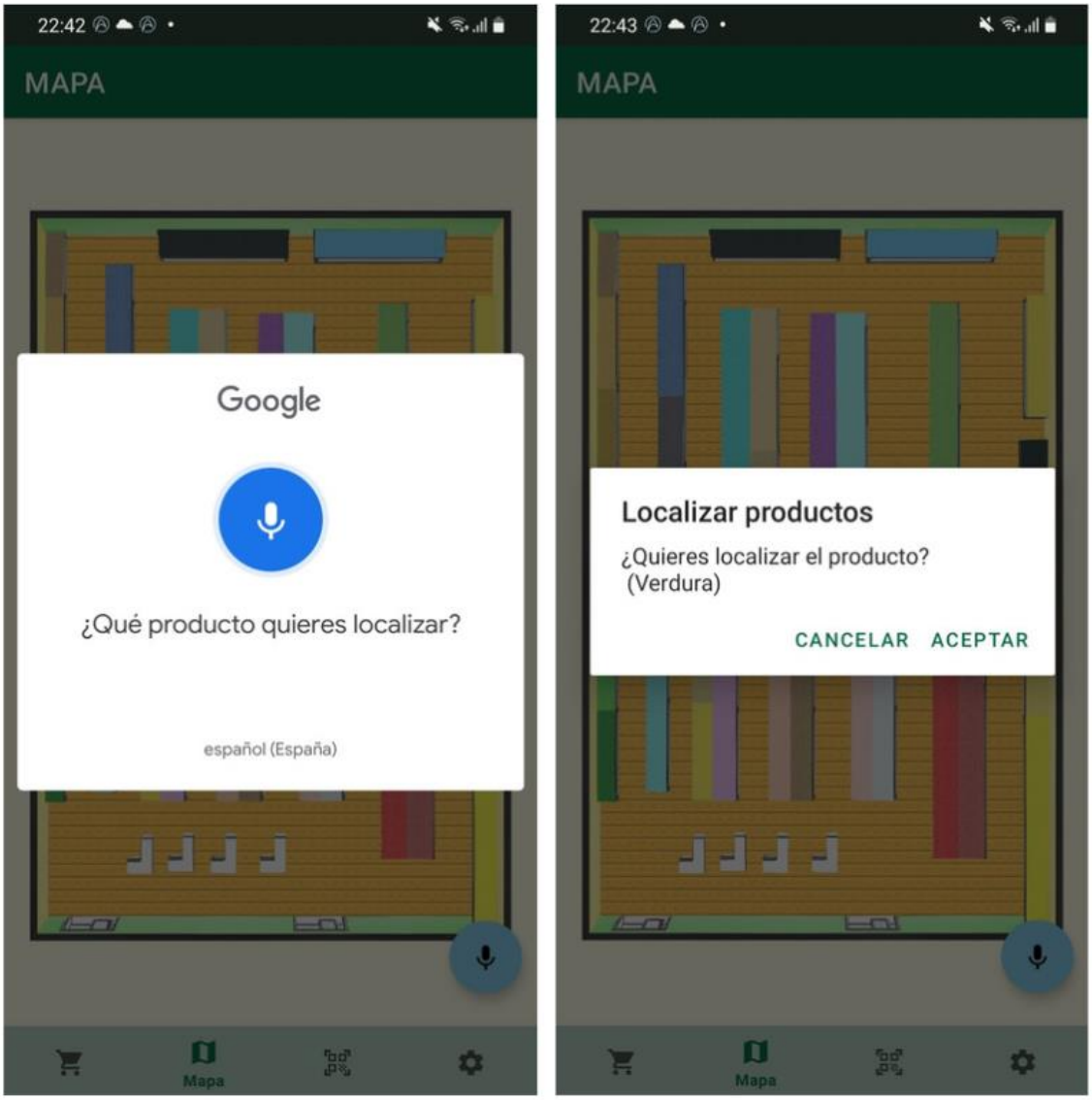

Figura 30: Interfaz final para la localización de productos por audio

5.7. Cambio de visualización, sistema de control de versiones, estilo e internacionalización

Este apartado lo hemos dedicado a comentar otros aspectos de la aplicación, como el fragmento de Ajustes, GitHub como sistema de control de versiones, las modificaciones realizadas para adaptar la *app* al estilo buscado y cómo se implementó la internacionalización.

Comenzando con el fragmento Ajustes, nos limitamos a crear un nuevo archivo XML de preferencias con un *switch* para activar el modo de visualización nocturno. Una vez creado, solo falta modificar `AjustesFragment` para que extienda de `PreferenceFragmentCompat`, activando así la funcionalidad de preferencias en esa pantalla. La Figura 31 muestra algunos ejemplos de cómo se ve la aplicación con el modo nocturno activado.

El modo nocturno se consideró como una funcionalidad necesaria para esta aplicación debido a las ventajas que conlleva aplicarlo, como la reducción de la fatiga visual y la accesibilidad para usuarios con ciertas condiciones visuales (sensibilidad a la luz y astigmatismo, por ejemplo).

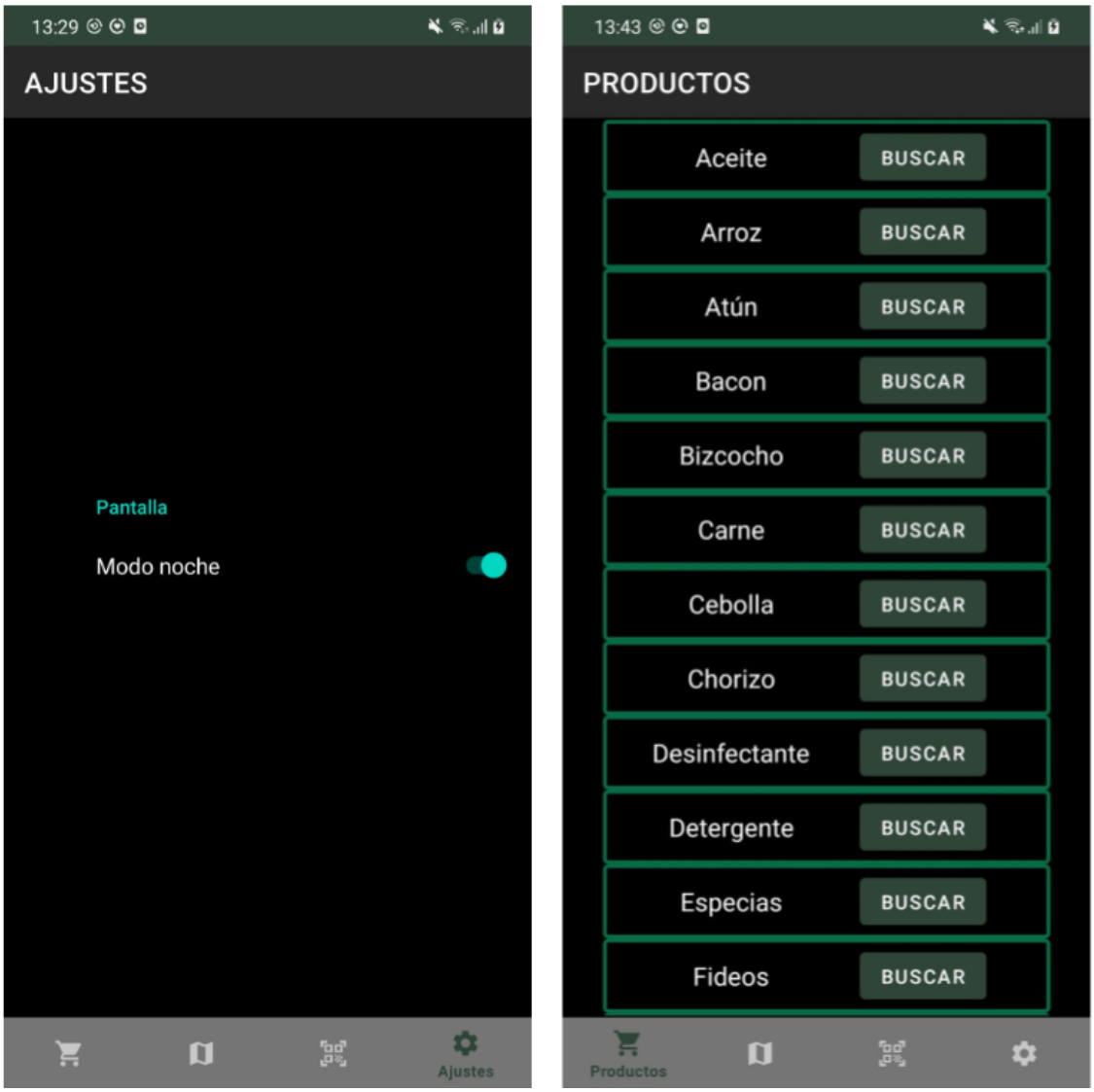

Figura 31: Interfaz final para el modo noche

En cuanto al estilo de la aplicación, para escoger una paleta de colores agradable y apropiada para nuestra *app* se consultaron varias páginas web sobre diseño gráfico, así como páginas para obtener el código hexadecimal de los colores que se querían aplicar. El aspecto de elementos como los botones, el fondo de pantalla o la barra de navegación se modificó mediante la creación de estilos personalizados para cada objeto. También se crearon tanto una pantalla Splash (pantalla de carga), la cual se muestra brevemente al iniciar la aplicación, como un logo para el icono de la *app*.

En cuanto al sistema de control de versiones, usamos GitHub, como se indicó en el capítulo 4. El código completo del proyecto se encuentra en el siguiente repositorio: GitHub

Finalmente, para asegurar la internacionalización de la *app*, es decir, para asegurarnos de que se adapta al inglés y al español, verificamos que todas las cadenas de texto mostradas en la interfaz tuviesen una traducción directa al inglés. En la Figura 32 se ven las carpetas donde se encuentran las cadenas traducidas en ambos idiomas. Para otros elementos como los productos, ya se explicó en apartados anteriores que la base de datos almacena dos carpetas, una para cada idioma. La Figura 32 muestra cómo, en `ProductosFragment`, se modifican los productos mostrados en la interfaz en función del lenguaje seleccionado en la configuración del dispositivo.

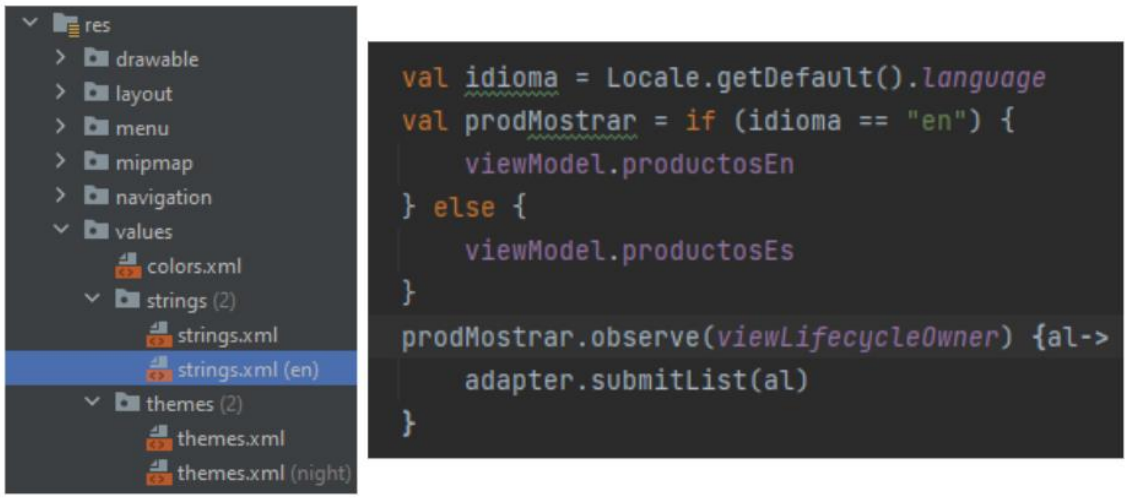

Figura 32: Código que adapta los productos según el idioma seleccionado

6. Pruebas

En este capítulo se describen las pruebas llevadas a cabo para demostrar que la aplicación cumple los requisitos mencionados en el capítulo 3, funcionando según lo establecido. Hemos realizado tres tipos de pruebas: pruebas funcionales, de calidad y de usabilidad.

6.1. Pruebas funcionales

Las pruebas funcionales son una parte esencial del proceso de desarrollo de software. Se enfocan en verificar que una aplicación o sistema cumpla con los requisitos y funcionalidades establecidos en su diseño. Estas pruebas se centran en evaluar el funcionamiento general de la aplicación desde la perspectiva del usuario, es decir, se evalúa cómo se comporta la *app* en situaciones del mundo real y si es capaz de realizar las tareas para las cuales fue diseñada.

Requisitos	En qué consiste	¿Se cumple?
Escaneo de QR	La aplicación deberá ser capaz de escanear un QR, obteniendo el listado con los productos y los planos	Sí
Lista de productos	La aplicación debe permitir a los usuarios acceder a una lista con los productos disponibles	Sí
Selección de productos	Los usuarios deben ser capaces de seleccionar productos específicos de la lista para visualizar su ubicación en el plano	Sí
Plano	Debe haber un plano donde se muestre la distribución del establecimiento, así como las coordenadas de los productos seleccionados	Sí
Reconocimiento de voz	La aplicación debe integrar un sistema de reconocimiento de voz	Sí
Localización de productos	La aplicación debe ser capaz de localizar productos tanto manualmente como por voz	Sí
Ajustes	La aplicación debe poder cambiar entre modos de visualización de interfaz (día/noche)	Sí

Tabla 3. Validación de los requisitos funcionales

Para la validación de los requisitos funcionales del capítulo 3, hemos conectado un dispositivo físico a Android Studio mediante cable con el administrador de dispositivos de la plataforma. Una vez conectado, hemos instalado la aplicación y probado su funcionamiento.

Basándonos en los resultados recopilados en la Tabla 3, la cual fue redactada a partir de los resultados obtenidos mediante la conexión con el dispositivo físico a Android Studio, podemos concluir de manera satisfactoria que la aplicación ejecuta sus funcionalidades principales sin inconvenientes.

6.2. Pruebas de calidad

Las pruebas de calidad son un conjunto de actividades y procesos realizados para evaluar la calidad del software y asegurar que cumpla con los requisitos y estándares establecidos. Estas pruebas tienen como objetivo identificar y corregir posibles errores, defectos o problemas en la aplicación, verificando que funcione correctamente, sea segura, eficiente y cumpla las expectativas de los usuarios.

Para este tipo de pruebas se suelen verificar los criterios de calidad establecidos por *Android Developers* [29], pero para centrarnos en los criterios que aplican a nuestro proyecto, nos vamos a basar en los requisitos no funcionales, los requisitos de la interfaz y los requisitos del reconocimiento del habla establecidos en el capítulo 3, ya que la mayoría provienen de los criterios de calidad definidos por *Android Developers*.

Requisitos	**En qué consiste**	**¿Se cumple?**
Seguridad	Se tiene que garantizar la integridad de la información almacenada en la base de datos	Sí
Disponibilidad	La aplicación debe estar disponible para su uso en todo momento	Sí
Adaptabilidad 1	La aplicación debe poder adaptarse a diferentes tamaños de dispositivos móviles	Sí
Adaptabilidad 2	La aplicación debe poder adaptarse a diferentes orientaciones del dispositivo	Sí
Internacionalización	La aplicación debe poder adaptarse a diferentes idiomas	Sí (inglés y español)
Interfaz intuitiva	La interfaz debe ser fácil de usar y comprensible para los usuarios, con un diseño claro y una navegación sencilla	Sí
Visualización del plano	El plano debe mostrarse con una representación clara de las secciones, las entradas y los pasillos	Sí
Lista de productos	La lista debe presentarse clara y organizada, con la posibilidad de desplazarse y seleccionar elementos fácilmente	Sí
Localización de los productos	La interfaz debe mostrar la posición del producto en el plano de forma clara	Sí
Precisión del reconocimiento	El sistema de reconocimiento de voz debe tener una alta precisión	Sí
Tiempo de respuesta	La aplicación debe tener una respuesta rápida al transcribir de voz a texto y localizar los productos en el plano	Sí
Manejo de errores	La aplicación debe ser capaz de proporcionar retroalimentación adecuada al usuario en caso de error	Sí

Tabla 4. Validación de los requisitos no funcionales, de la interfaz y del reconocimiento del habla

Como se puede ver en la tabla 4, todos los requisitos establecidos en el capítulo 3 han sido aplicados de manera efectiva en nuestra aplicación. Los requisitos no funcionales, tales como la internacionalización y la orientación de la pantalla, fueron tenidos en cuenta desde las etapas iniciales del desarrollo de la aplicación, así como la retroalimentación en caso de errores.

En lo referente a otros requisitos más subjetivos, como el diseño y la usabilidad, serán sometidos a pruebas con diversos usuarios con el fin de obtener conclusiones más precisas y acertadas.

6.3. Pruebas de usabilidad

Las pruebas de usabilidad consisten en evaluar la eficacia, eficiencia y satisfacción del usuario al interactuar con la aplicación en un contexto de uso específico. Estas pruebas son esenciales para proporcionar una experiencia de usuario óptima, lo que es fundamental para el éxito de una aplicación móvil, sobre todo para una enfocada en usuarios de diversas tipologías, como la nuestra.

Para llevar a cabo esta prueba, se contó con la participación de cinco usuarios cuyas edades fluctúan entre los 20 y los 80 años. Estos usuarios siguieron el guion del Anexo 1: Cuestionario de usabilidad, garantizando así la exploración de todas las funcionalidades de la aplicación. Posteriormente, una vez realizada la prueba, completaron un formulario diseñado mediante la herramienta Google Forms [72]. El contenido de dicho formulario se encuentra disponible para su consulta en el Anexo 1.

El cuestionario consiste en una escala de puntuación del 1 al 5, donde el 1 indica “totalmente en desacuerdo” y el 5 “totalmente de acuerdo”. Esta escala se utilizará en el eje horizontal de las gráficas que presentan los resultados (Figuras 33 a 44), mientras que el eje vertical representa el número de usuarios que seleccionaron cada opción. Además, se incorporaron dos preguntas (Figuras 42 y 43) destinadas a recabar la opinión de los usuarios en caso de experimentar algún tipo de dificultad durante el desarrollo de la prueba.

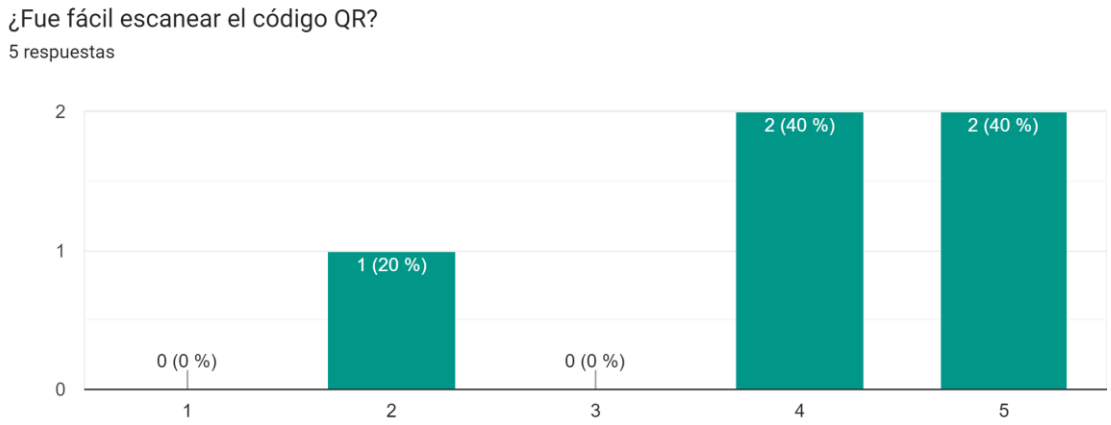

Figura 33: Resultados de la primera pregunta del cuestionario

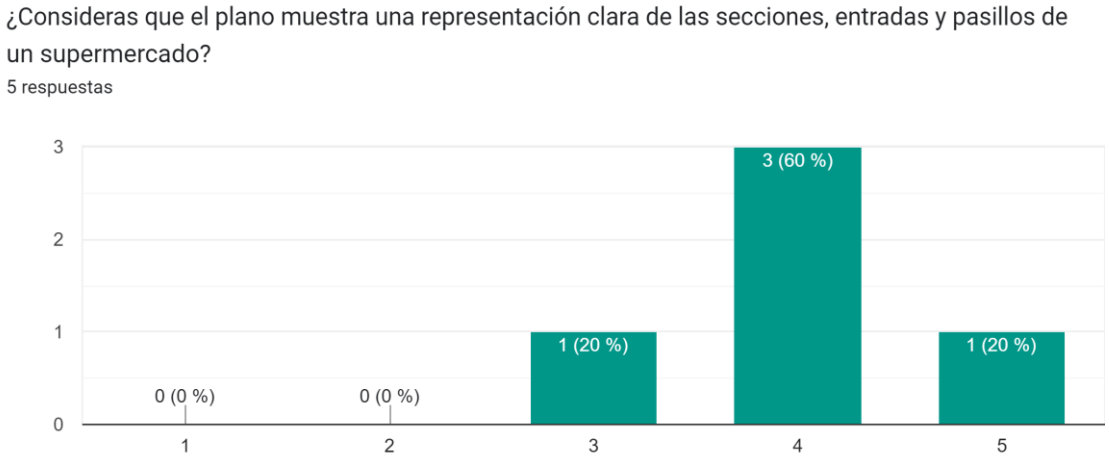

Figura 34: Resultados de la segunda pregunta del cuestionario

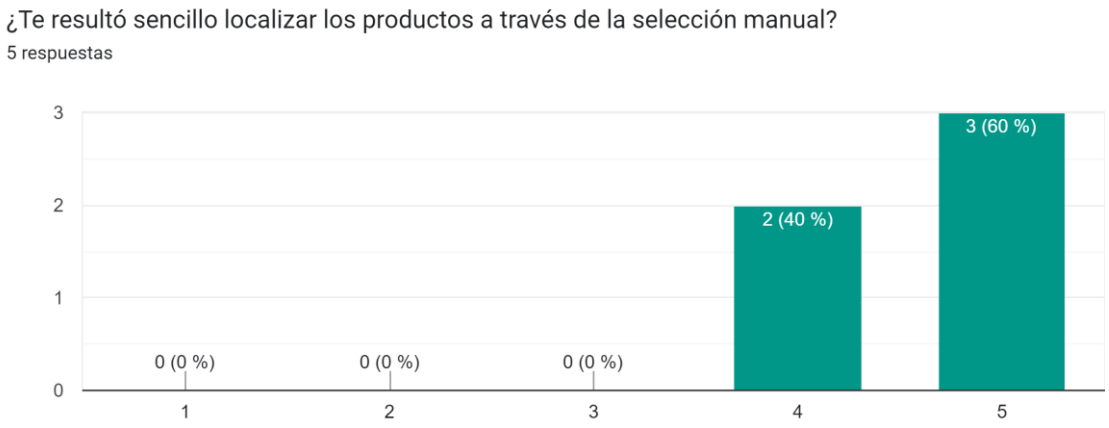

Figura 35: Resultados de la tercera pregunta del cuestionario

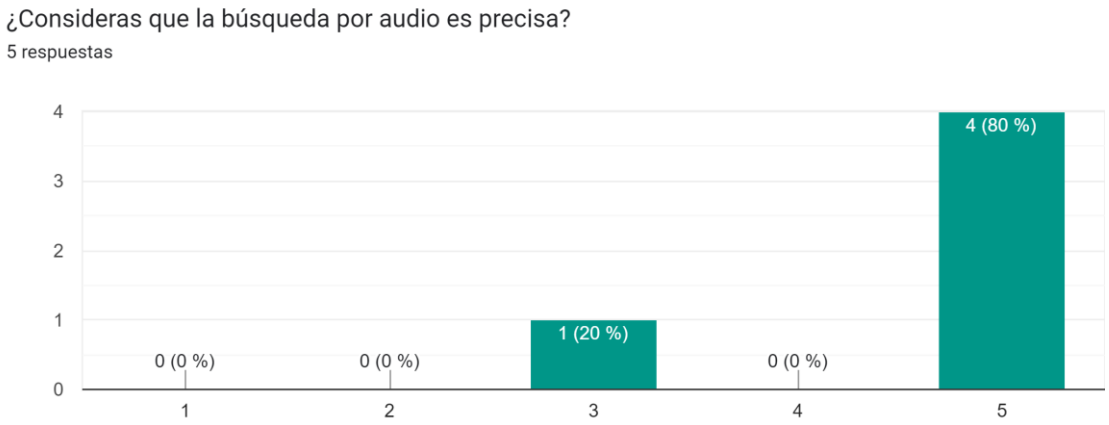

Figura 36: Resultados de la cuarta pregunta del cuestionario

Como se puede observar en las Figuras 33, 34, 35 y 36, parece que los objetivos principales de la aplicación (búsqueda por audio, manual y el escaneo de QR) han sido alcanzados satisfactoriamente. Sin embargo, se observa que los usuarios no familiarizados con los códigos QR, especialmente los ancianos, enfrentaron ciertos desafíos al tratar de enfocarlos correctamente. Asimismo, para este grupo de usuarios, se presentaron algunas dificultades menores al utilizar la función de búsqueda por audio, como hablar antes de tiempo o cometer errores al volver a pulsar el micrófono, entre otros aspectos.

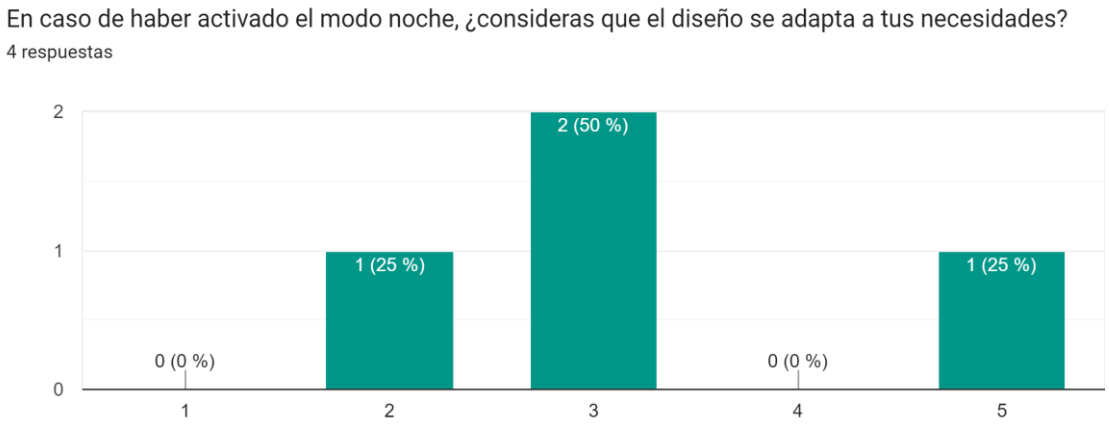

Figura 37: Resultados de la quinta pregunta del cuestionario

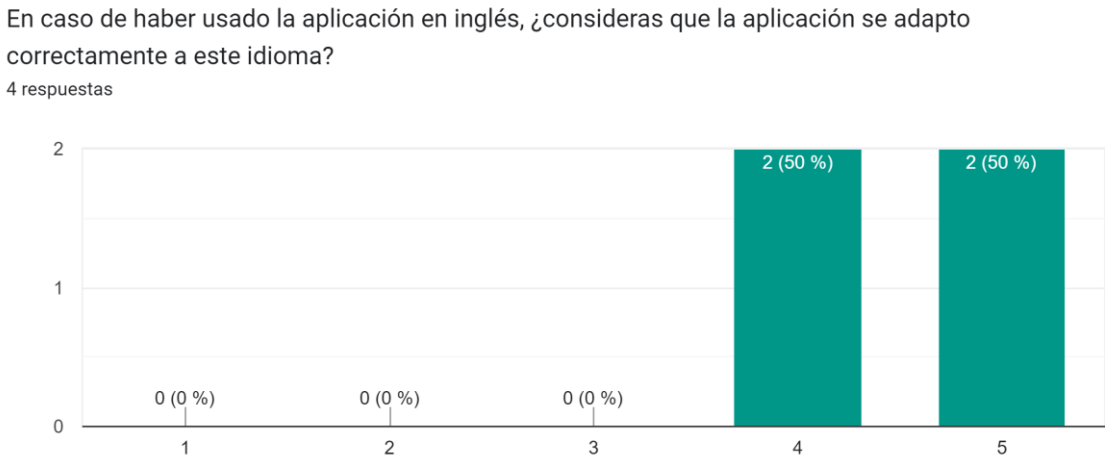

Figura 38: Resultados de la sexta pregunta del cuestionario

Con respecto a los resultados presentados en las Figuras 37 y 38, inicialmente ninguno de los usuarios encontró necesario activar el modo nocturno, y aquellos que lo hicieron fue debido a que estaba especificado en el guion del cuestionario. En cambio, el manejo de idiomas en la aplicación parece funcionar de manera adecuada.

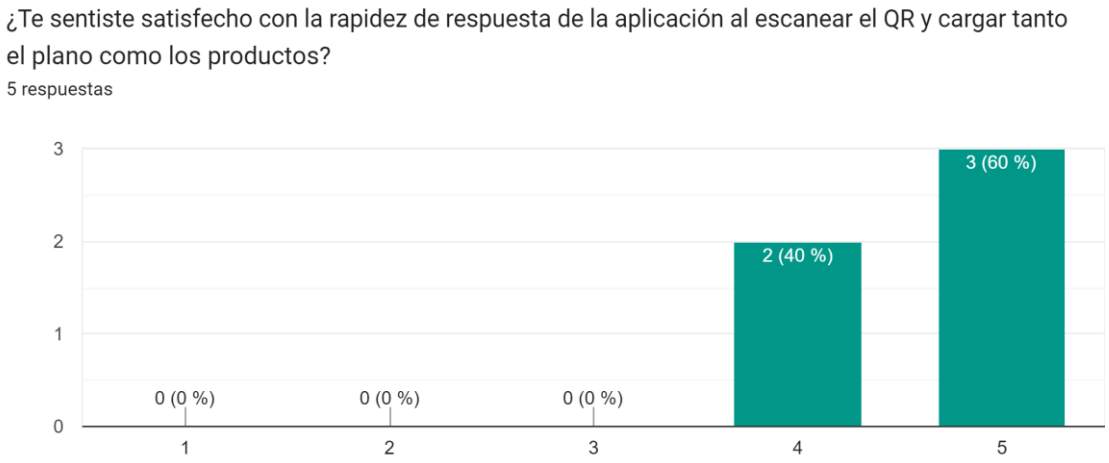

Figura 39: Resultados de la séptima pregunta del cuestionario

Como se puede observar en la Figura 39, parece que tanto el acceso a la base de datos como la carga de los planos en la interfaz no presenta problemas.

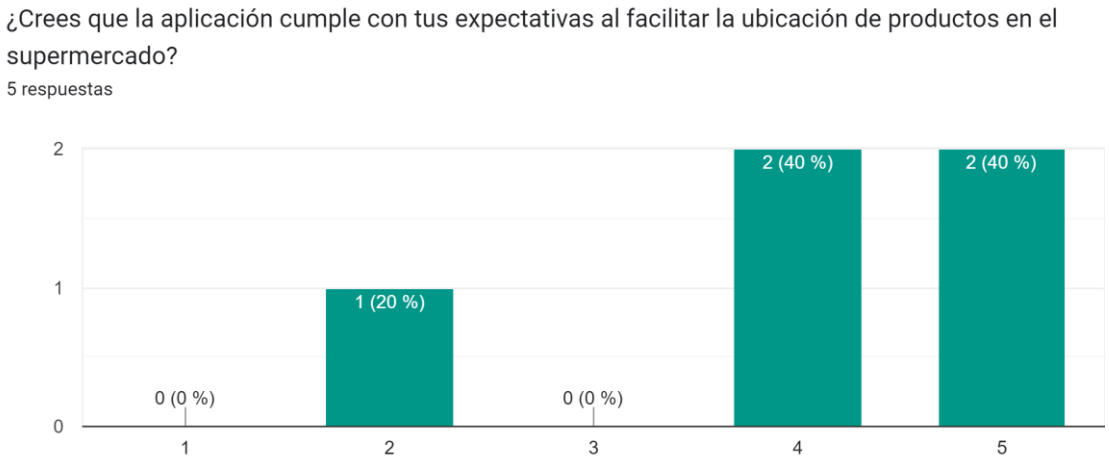

Figura 40: Resultados de la octava pregunta del cuestionario

La Figura 40 presenta resultados de notable interés. En este caso, uno de los usuarios no se mostró conforme con el funcionamiento de la función de localización de productos y sugirió la inclusión de otro cursor que indicara la posición del cliente dentro del establecimiento, con el objetivo de facilitar la orientación. Sin embargo, es importante señalar que en nuestro caso nunca se tuvo en cuenta esta posibilidad, dado que estamos trabajando con la premisa de que este supermercado es ficticio y, por lo tanto, no se encuentra habilitada la localización en tiempo real.

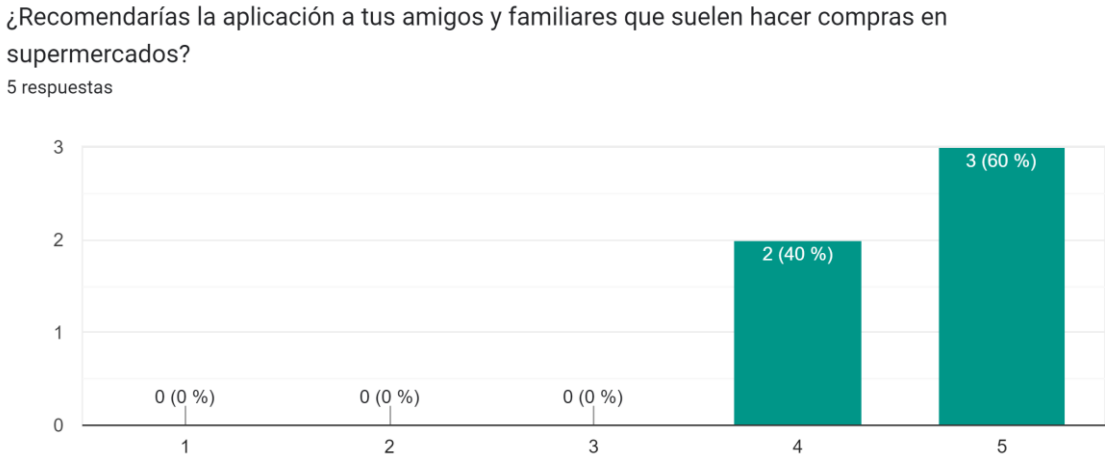

Figura 41: Resultados de la novena pregunta del cuestionario

La Figura 41 se asegura de que el objetivo de la aplicación es alcanzado. Es decir, comprueba que los usuarios consideran la aplicación útil y están dispuestos a compartirla con conocidos que podrían estar interesados en sus funciones.

Figura 42: Resultados de la décima pregunta del cuestionario

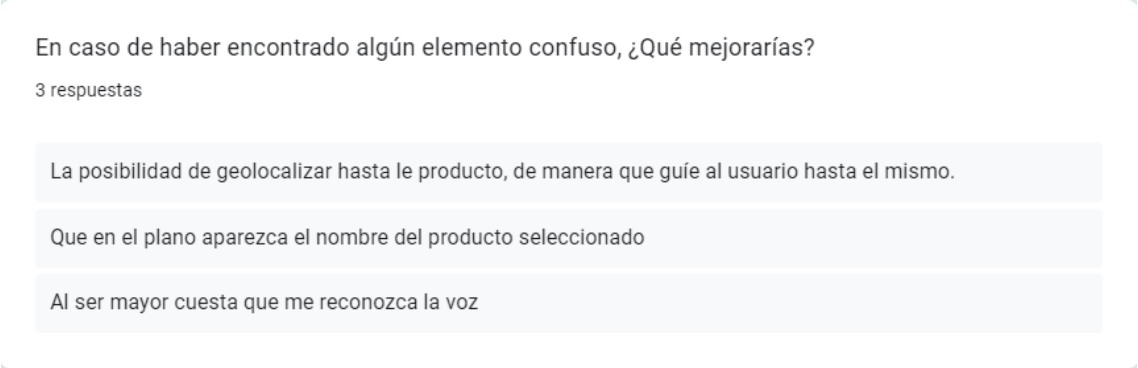

Figura 43: Resultados de la undécima pregunta del cuestionario

La Figura 43 recopila algunos de los comentarios y sugerencias propuestas por los usuarios (tanto por aquellos que no seleccionaron ningún aspecto como difícil de usar en la Figura 42 como los que lo hicieron) para realizar mejoras en la aplicación. Como se mencionó en figuras anteriores, la propuesta de llevar a cabo una geolocalización precisa hasta el producto resulta complicada; sin embargo, la sugerencia de mostrar el nombre del producto podría ser aplicable.

En nuestro caso, hemos optado por mostrar el nombre del producto en el mensaje emergente que aparece justo antes de localizarlo en el plano (por si los usuarios olvidan el elemento que solicitaron o lo que dijeron por audio). No obstante, es plausible añadir un texto adicional en el mapa con el nombre del producto seleccionado.

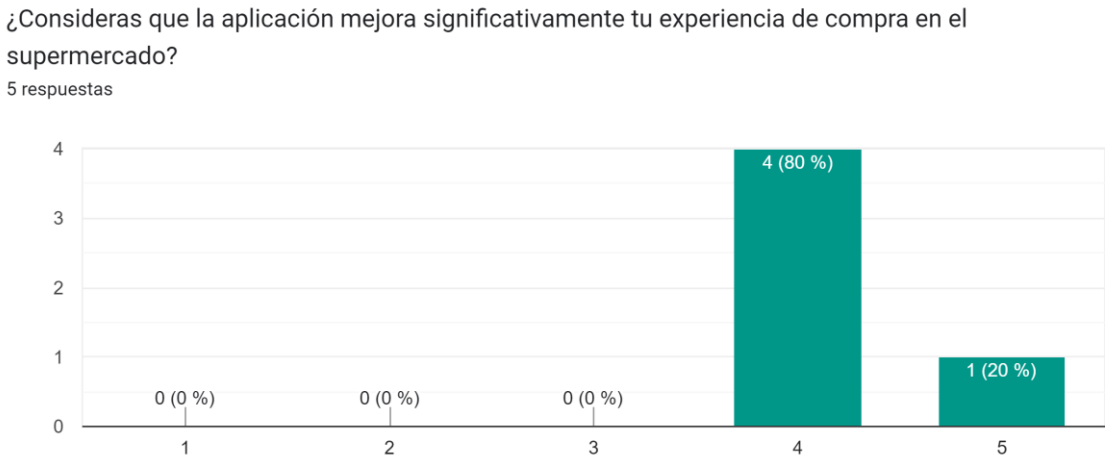

Figura 44: Resultados de la duodécima pregunta del cuestionario

En conclusión, la aplicación puede suponer un desafío para ciertos perfiles de usuarios; sin embargo, en términos generales, se puede apreciar que presenta un diseño intuitivo y eficiente (Figura 44). Todas las mejoras propuestas por los usuarios que participaron en esta prueba serán tomadas en consideración para su implementación en futuras versiones de la aplicación.

Asimismo, sería altamente beneficioso considerar la incorporación de una muestra de población considerablemente más amplia en las evaluaciones de usabilidad destinadas a futuras versiones de la *app*. La muestra utilizada en este proyecto, compuesta por únicamente cinco individuos, restringe las conclusiones extraídas por el cuestionario.

7. Conclusiones y trabajos futuros

En este capítulo se reflexiona sobre si se han cumplido los objetivos propuestos en el apartado 1.2 de la memoria, mencionando los problemas encontrados y cómo se solucionaron, qué se ha aprendido con este proyecto tanto profesional como personalmente, la relación del trabajo realizado con los estudios cursados y, finalmente, trabajos futuros.

7.1. Objetivos cumplidos

En primer lugar, el objetivo principal consistía en desarrollar una aplicación móvil capaz de guiar, mediante peticiones de voz, hacia la ubicación del producto de interés en un supermercado. Tal como se mostró en el capítulo 5, este objetivo ha sido alcanzado exitosamente, no solo mediante la implementación de la búsqueda por voz sino también mediante la incorporación de la búsqueda manual.

En segundo lugar, se buscaba construir una base de datos con los planos del supermercado y las coordenadas de los productos, razón por la cual se investigó sobre la herramienta Firebase, llegando a usar tanto Firebase Storage como Firestore Database para almacenar los datos. Asimismo, se ha implementado un lector de códigos QR con el fin de mostrar los datos del supermercado ficticio, así como un QR que nos permite acceder a la base de datos.

En tercer lugar, se adquirieron nociones sobre Android Studio y el desarrollo de aplicaciones, empleando los conocimientos adquiridos en la asignatura optativa DADM (Desarrollo de Aplicaciones para Dispositivos Móviles) y los aprendidos por cuenta propia.

A su vez, y conforme al análisis detallado en el apartado 2.3, se investigaron herramientas capaces de transcribir de manera precisa el contenido de archivos de audio a texto. A pesar de que en un principio se seleccionó la herramienta iATROS para la implementación de esta funcionalidad, diversos problemas surgieron en cuanto a la recepción y envío de información, lo cual dio lugar a su reemplazo por RecognizerIntent. Además, desde el inicio del desarrollo de la aplicación, se tuvo en cuenta la adaptación de la interfaz para que pudiera ser visualizada adecuadamente en distintas dimensiones y orientaciones de pantalla. Por ejemplo, las coordenadas de los productos son calculadas en función del tamaño del plano, y ciertos elementos de la interfaz se ajustan automáticamente según la orientación del dispositivo; tal es el caso de la barra de navegación que se sitúa en la parte inferior en modo vertical y en un lateral en modo apaisado.

Finalmente, para comprobar que la aplicación es sencilla e intuitiva, se realizaron las pruebas de usabilidad del apartado 6.3, llegando a la conclusión de que, en efecto, se cumple este objetivo final. En conclusión, se puede decir que la aplicación desarrollada ha cumplido todos los objetivos propuestos.

7.2. Aprendizaje e imprevistos

Las mayores dificultades surgieron al comienzo del proyecto, principalmente a la hora de seleccionar las tecnologías necesarias para la implementación de las funcionalidades principales. Esto se debe a no estar familiarizada ni con el desarrollo de aplicaciones móviles ni con la gestión de bases de datos, así como la gestión de redes. El diseño de elementos gráficos como el plano o el logo también supuso un reto al inicio.

Sin embargo, una vez establecida la estructura base de la aplicación y encontrada la plataforma en la que se implementó la base de datos, las dificultades se redujeron considerablemente. Además, tras considerar que el tiempo necesario para familiarizarse con la gestión de redes era demasiado extenso (la herramienta iATROS se encontraba en otro servidor y sistema operativo) decidimos cambiar por otra herramienta más accesible.

Desde una perspectiva personal, este proyecto me ha ayudado a adquirir conocimientos significativos, abarcando áreas como el desarrollo de aplicaciones móviles, la creación y administración de bases de datos y el funcionamiento de los sistemas de reconocimiento del habla. También se ha aprendido a identificar soluciones alternas en respuesta a dificultades, junto con la gestión eficiente del tiempo dedicado a dichas tareas.

A nivel profesional, he aprendido que nuestra profesión exige una dedicación perpetua al aprendizaje autodidacta. Un profesional competente en el campo de la informática no puede limitarse a los conocimientos adquiridos en su rama; al contrario, debe estar dispuesto a tener que profundizar su formación de manera independiente o con la ayuda de otros profesionales.

7.3. Relación del trabajo desarrollado con los estudios cursados

El desarrollo de este proyecto no habría sido posible sin los conocimientos adquiridos durante la carrera de Ingeniería informática. El primer y segundo año aportaron las bases de programación y organización necesarias para el desarrollo de cualquier proyecto en el ámbito informático. En cuanto a conocimientos específicos, se ha consultado material sobre las siguientes asignaturas:

- DADM (Desarrollo de Aplicaciones para Dispositivos Móviles): La asignatura que más aportó al proyecto fue la que instruyó en la creación de aplicaciones desde cero. Adicionalmente, durante uno de los proyectos grupales de la asignatura, un compañero de equipo de la rama de Software empleó la plataforma Firebase, exponiendo de esa manera su multitud de usos y su fácil integración con Android Studio.
- APR (Aprendizaje Automático): Asignatura cursada en cuarto (rama Computación) que combina la IA con el reconocimiento de formas. Nos enseñó la estructura y el funcionamiento de los sistemas de aprendizaje automático como las redes neuronales profundas. Estos conocimientos fueron clave para entender el funcionamiento de los sistemas reconocedores de voz.
- AIN (Agentes Inteligentes): Asignatura cursada en tercero (rama Computación). Al igual que con APR, nos ayudó a comprender mejor el funcionamiento de los sistemas inteligentes, así como el uso de modelos probabilísticos en sus procesos.

- BDA (Bases de Datos y Sistemas de Información) Asignatura cursada en tercero. Nos aportó la base sobre el funcionamiento de una base de datos y cómo se realizan las consultas a la misma.
- GPR (Gestión de Proyectos): Asignatura cursada en tercero. Nos ayudó a comprender la estructura de los proyectos informáticos, sobre todo en el ámbito del reparto de tareas y la gestión del tiempo. Fue la asignatura en la que usamos la herramienta Trello [73], la cual aplica la metodología Kanban.

Es decir, pese a que las materias cursadas en la rama de Computación desempeñaron un papel fundamental en la comprensión de los sistemas de reconocimiento de voz, es preciso resaltar que la asignatura de Desarrollo de Aplicaciones para Dispositivos Móviles (DADM) constituyó el factor determinante para la realización de este proyecto. Gran parte de los conocimientos adquiridos durante el desarrollo de la aplicación nos ayudaron a profundizar en asignaturas vistas al comienzo de la carrera, complementando así los conocimientos sobre IA que ya se tenían en la rama de Computación.

7.4. Trabajos futuros

La aplicación desarrollada sigue abierta a posibles cambios, como los solicitados en las pruebas de usabilidad del apartado 6.3. A continuación, se listan algunas de esas posibles mejoras:

- Indicar la posición del cliente: Fue una de las mejoras propuestas durante las pruebas de usabilidad. Actualmente es inviable, pero en el caso de que un establecimiento optara por usar esta aplicación y ofrecer acceso a sus productos, esta funcionalidad podría mejorar exponencialmente la experiencia del usuario. Además, se contempla la posibilidad de introducir una retroalimentación táctil sutil, como una leve vibración, al alcanzar el producto deseado.

- Mostrar el nombre del producto en el cursor: Otra de las mejoras propuestas durante las pruebas de usabilidad. No se implementó debido a que existían otros elementos con prioridades superiores; sin embargo, es una consideración que podría ser tomada en cuenta para versiones futuras.

- Mejorar la visualización del plano: Sería factible la incorporación de componentes tales como la opción de aplicar *zoom* al plano, con el propósito de enriquecer la experiencia del usuario.

- Mostrar el contenido de los estantes: Una mejora bastante innovadora que, lamentablemente, no fue considerada debido a su dificultad de aplicación. Consiste en presentar los productos que se encuentran en cada sección o estante mediante la selección de la zona correspondiente en el plano. Este enfoque tiene la intención de acortar significativamente el tiempo destinado a la búsqueda, especialmente para aquellos usuarios que desean evitar la búsqueda individualizada de los productos.

- Proporcionar información sobre los productos: Funcionalidad que podría mejorar la experiencia del usuario al permitirle consultar información adicional sobre los

productos buscados, como los alérgenos. Debido a nuestro reducido catálogo de productos, no se consideró necesaria su implementación.

- Multiplataforma: Actualmente, la aplicación solo es funcional en dispositivos Android. Para futuras versiones se puede considerar su expansión a otras plataformas tales como iOS.

- Evaluación mediante una población más extensa: El uso de una muestra de población más extensa en las pruebas de usabilidad podría facilitar la detección de un mayor número de aspectos susceptibles de mejora.

En conclusión, aunque el proyecto cumple con los objetivos planteados, se puede seguir desarrollando hasta obtener una aplicación más completa y con funcionalidades más avanzadas.

Bibliografía

[1] Android. Disponible en: https://www.android.com/intl/es_es

[2] Firebase. Disponible en: https://firebase.google.com/?hl=es-419

[3] Android Studio. Disponible en: https://developer.android.com/studio

[4] Google. Disponible en: https://www.google.com/

[5] Linux. Disponible en: https://www.kernel.org/

[6] Play Store. Disponible en: https://play.google.com/store/games?hl=es_419&gl=US

[7] App Store. Disponible en: https://www.apple.com/es/app-store/

[8] iOS. Disponible en: https://www.apple.com/es/ios/ios-16

[9] Apple. Disponible en: https://www.apple.com/

[10] Material Design. Disponible en: https://m3.material.io/

[11] Java. Disponible en: https://www.java.com/es/

[12] Kotlin. Disponible en: https://kotlinlang.org/

[13] IntelliJ IDEA. Disponible en: https://www.jetbrains.com/idea/

[14] Android Jetpack. Disponible en: https://developer.android.com/jetpack?hl=es-419

[15] Android App Bundle. Disponible en: https://developer.android.com/guide/app-bundle?hl=es-419

[16] Eclipse. Disponible en: https://www.eclipse.org/downloads/

[17] Python. Disponible en: https://www.python.org/

[18] Visual Studio. Disponible en: https://visualstudio.microsoft.com/es/#vs-section

[19] Kaldi. Disponible en: https://github.com/kaldi-asr/kaldi

[20] Microsoft Windows. Disponible en: https://www.microsoft.com/es-es/windows?r=1

[21] Sphinx-4. Disponible en: https://github.com/cmusphinx/sphinx4

[22] HTK. Disponible en: https://htk.eng.cam.ac.uk/

[23] Microsoft. Disponible en: https://www.microsoft.com/es-es

[24] Play Store. (2013). Entrada de voz lista compras [Aplicación móvil]. Versión 5.8.06. Android. Disponible en: https://play.google.com/store/apps/details?id=com.tksolution.einkaufszettelmitspracheingabe&hl=es&gl=US

[25] Play Store. (2018). Mercadona [Aplicación móvil]. Versión 207. Android. Disponible en: https://play.google.com/store/apps/details?id=es.mercadona.tienda&hl=es&gl=US

[26] Play Store. (2012). Zara [Aplicación móvil]. Versión 12.21.0. Android. Disponible en: https://play.google.com/store/apps/details?id=com.inditex.zara&hl=es&gl=US

[27] FERNÁNDEZ, Rosa. Tema: Consumo y uso de smartphones en España. *Statista* [en línea]. 11 de febrero de 2023 [consultado el 25 de mayo de 2023]. Disponible en: https://es.statista.com/temas/4086/consumo-y-uso-de-smartphones-en-espana/#topicOverview

[28] *Desarrolladores de Android*. Android Developers. Disponible en: https://developer.android.com/?hl=es-419

[29] Calidad básica de las apps. *Android Developers* [en línea]. 17 de mayo de 2021 [consultado el 20 de junio de 2023]. Disponible en: https://developer.android.com/docs/quality-guidelines/core-app-quality?hl=es-419#listing

[30] ¿Hay algún IDE para Android 2023?. *CISIN* [en línea] [consultado el 21 de junio de 2023]. Disponible en: https://www.cisin.com/coffee-break/es/technology/is-there-any-ide-for-android.html

[31] MALIK, Mishaim, et al. Automatic speech recognition: a survey. *Multimedia Tools and Applications*, 2021, vol. 80, p. 9411-9457

[32] GAIDA, Christian, et al. Comparing open-source speech recognition toolkits. En *11th International Workshop on Natural Language Processing and Cognitive Science*. 2014

[33] POVEY, Daniel, et al. The Kaldi speech recognition toolkit. En *IEEE 2011 workshop on automatic speech recognition and understanding*. IEE Signal Processing Society, 2011

[34] WALKER, Willie, et al. Sphinx-4: A flexible open source framework for speech recognition. Informe técnico SMLI TR-2004-139, 2004

[35] YOUNG, Steve, et al. The HTK book. *Cambridge university engineering department*, 2002, vol. 3, no 175, p. 12

[36] CARRILLO AQUILAR, Roberto. Diseño y manipulación de modelos ocultos de Markov, utilizando herramientas HTK: una tutoría. *Ingeniare. Revista chilena de ingeniería*, 2007, vol. 15, no 1, p. 18-26

[37] LUJÁN-MARES, Míriam, et al. iATROS: A speech and handwritting recognition system. *V Jornadas en Tecnologías del Habla (VJTH'2008)*, 2008, p. 75-78

[38] Alumnos de la UPSA desarrollan una aplicación para agilizar las compras en supermercados. *DiCYT* [en línea]. 2 de julio de 2012 [consultado el 27 de mayo de 2023]. Disponible en: https://www.dicyt.com/noticias/alumnos-de-la-upsa-desarrollan-una-aplicacion-para-agilizar-las-compras-en-supermercados

[39] Una aplicación para móviles agiliza las compras en supermercados. *SINC* [en línea]. 5 de julio de 2012 [consultado el 2 de junio de 2023]. Disponible en: https://www.agenciasinc.es/Noticias/Una-aplicacion-para-moviles-agiliza-las-compras-en-supermercados

[40] Aguilar, R. Qué supermercado tiene la mejor app: comparativa a fondo. *Xataka Móvil* [en línea]. 19 de junio de 2021 [consultado el 3 de junio de 2023]. Disponible en: https://www.xatakamovil.com/aplicaciones/que-supermercado-tiene-mejor-app-comparativa-a-fondo

[41] *Las 5 mejores Metodologías de Desarrollo de Software*. (2022, 27 de octubre). GooApps. Disponible en: https://gooapps.es/2022/10/27/las-5-mejores-metodologias-de-desarrollo-de-software/

[42] MAIDA, Esteban Gabriel; PACIENZIA, Julián. Metodologías de desarrollo de software. 2015.

[43] *Kanban frente a scrum*. Atlassian. Disponible en: https://www.atlassian.com/es/agile/kanban/kanban-vs-scrum

[44] BRADBURY, Jeremy. Linear predictive coding. *Mc G. Hill*, 2000

[45] HASAN, Md Rashidul, et al. Speaker identification using mel frequency cepstral coefficients. *variations*, 2004, vol. 1, no 4, p. 565-568

[46] EDDY, Sean R. Hidden Markov models. *Current opinion in structural biology*, 1996, vol. 6, no 3, p. 361-365

[47] REYNOLDS, Douglas A., et al. Gaussian mixture models. *Encyclopedia of biometrics*, 2009, vol. 741, no 659-663

[48] MEYER, David; WIEN, F. T. Support vector machines. *The Interface to libsvm in package e1071*, 2015, vol. 28, no 20, p. 597

[49] ZOU, Jinming; HAN, Yi; SO, Sung-Sau. Overview of artificial neural networks. *Artificial neural networks: methods and applications*, 2009, p. 14-22

[50] ABDELRAHMAN, Yusri T.; SAEED, Rashid A; EL-TAHIR, Amel. Multiple Physical Layer Pipes performance for DVB-T2. En *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*. IEE, 2017. P. 1-7

[51] CHURCH, Kenneth W.; RAU, Lisa F. Commercial applications of natural language processing. *Communications of the ACM*, 1995, vol. 38, no 11, p. 71-79

[52] MOHRI, Mehryar. Weighted finite-state transducer algorithms. an overview. *Formal Languages and Applications*, 2004, p. 551-563

[53] CICHY, Radoslaw M.; KAISER, Daniel. Deep neural networks as scientific models. *Trends in cognitive sciences*, 2019, vol. 23, no 4, p. 305-317

[54] LÁZARO-GREDILLA, Miguel, et al. Beyond imitation: Zero-shot task transfer on robots by learning concepts as cognitive programs. *Science Robotics*, 2019, vol. 4, no 26, p. eaav3150

[55] KORNAI, András. Extended finite state models of language. *Natural Language Engineering*, 1996, vol. 2, no 4, p. 287-290

[56] GOSLING, James. *The Java language specification*. Addison-Wesley Professional, 2000

[57] Wondershare Mockitt. Disponible en: https://mockitt.wondershare.com/brand

[58] Wondershare. Disponible en: https://www.wondershare.es/

[59] GitHub. Disponible en: https://github.com/

[60] Play Store. (2013). magicplan [Aplicación móvil]. Versión 2023.7.0. Android. Disponible en: https://play.google.com/store/apps/details?id=com.sensopia.magicplan&hl=es_419

[61] SOUCHON, Nathalie; VANDERDONCKT, Jean. A review of XML-compliant user interface description languages. En *International Workshop on Design, Specification, and Verification of Interactive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 377-391

[62] JSON. Disponible en: https://www.json.org/json-es.html

[63] PATEL, Pritesh N.; PATEL, Jigisha K.; VIRPARIA, Paresh V. Generating Selected Query from Spoken Words on Android Smart Phone. En *International Journal of Emerging Trends & Technology in Computer Science*, 2013, vol. 2, no 3, p. 91-94

[64] RecognizerIntent. *Android Developers* [en línea]. [consultado el 22 de julio de 2023]. Disponible en: https://developer.android.com/reference/android/speech/RecognizerIntent

[65] Canva. Disponible en: https://www.canva.com/

[66] Zyro. Disponible en: https://zyro.com/es/herramientas/upscaler-de-imagenes

[67] QR Code Generator. Disponible en: https://es.qr-code-generator.com/

[68] ZXing. Disponible en: https://github.com/zxing/zxing

[69] Glide. Disponible en: https://github.com/bumptech/glide

[70] Picasso. Disponible en: https://github.com/square/picasso

[71] Coil. Disponible en: https://coil-kt.github.io/coil/

[72] Google Forms. Disponible en: https://docs.google.com/forms

[73] Trello. Disponible en: https://trello.com/es

Glosario

ANN (*Artificial Neural Network*): Red neuronal artificial. Modelo computacional inspirado en la estructura y funcionamiento del cerebro humano. Consiste en una red de nodos interconectados, llamados neuronas artificiales, que procesan información de entrada y generan una salida.

API (*Application Programming Interfaces*): Interfaz de programación de aplicaciones.

APK (*Android Application Package*): Archivo que contiene una aplicación que necesita ser instalada en un dispositivo. Facilita el proceso de instalar una aplicación, haciendo que sea más rápido.

ARPA (*Advanced Research Projects Agency*): Formato de archivo comúnmente utilizado para representar modelos de lenguaje en el campo del procesamiento de lenguaje natural. Fue desarrollado por la Agencia de Proyectos de Investigación Avanzada (ARPA).

ASR (*Automatic Speech Recognition*): Reconocimiento automático del habla.

DNN (*Deep Neural Network*): Una red neuronal profunda es un tipo de modelo de aprendizaje automático basado en redes neuronales artificiales con múltiples capas ocultas.

***Framework*:** Conjunto de herramientas, bibliotecas y componentes predefinidos que proporcionan una estructura y funcionalidades comunes para facilitar el desarrollo de aplicaciones. Se trata de una plataforma de software que proporciona a los desarrolladores una serie de funciones y características listas para usar, así como una arquitectura bien definida, evitando que se tenga que desarrollar todo desde cero. Angular, de JavaScript, es un ejemplo de *framework.*

FSM (*Finite-State Models*): Los modelos de estado finito son una representación matemática y computacional de sistemas que pueden estar en diferentes estados y cambiar de uno a otro en respuesta a eventos o entradas. En el contexto del reconocimiento de voz, se usan para modelar gramáticas y lenguajes. En la etapa de decodificación realizan la búsqueda eficiente de hipótesis y encuentran la mejor secuencia de palabras que coincida con la señal de entrada.

FST (*Finite-State Transducer*): Un transductor de estados finitos es un autómata finito con dos cintas, una de entrada y otra de salida. Pueden representar modelos de lenguaje, diccionarios de pronunciación y reglas de transformación fonética, lo que permite realizar conversiones y manipulaciones sofisticadas de secuencias de palabras y fonemas.

GMM (*Gaussian Mixture Models*): Algoritmo de mezclas gaussianas. Modelo probabilístico en el que se considera que las observaciones siguen una distribución probabilística formada por la combinación de múltiples distribuciones normales.

HMM (*Hidden Markov Model*): Modelo oculto de Markov. Modelo estadístico cuyo objetivo es determinar los parámetros desconocidos a partir de parámetros observables.

HTTP (*HyperText Transfer Protocol*): El Protocolo de Transferencia de Hipertexto es un protocolo de comunicación ampliamente utilizado en la web para transferir información entre un cliente (navegador) y un servidor.

IA: Inteligencia Artificial

IDE (*Integrated Development Environment*): Entorno de Desarrollo integrado.

JSON (*JavaScript Object Notation*): Formato ligero de intercambio de datos, fácil de leer y escribir para los humanos y simple de interpretar para las máquinas.

JSG (*Joint Symbol Grammar*): Formato de archivo utilizado para representar gramáticas. JSG permite describir reglas gramaticales para definir la estructura y el reconocimiento de un lenguaje.

LPC (*Linear Predictive Coding*): Codificación predictiva lineal. Se usa partiendo de la idea de que la voz puede modelarse como una combinación lineal de p muestras anteriores más una señal de error.

MFCC (*Mel-Frequency Cepstral Coefficients*): Coeficientes Cepstrales en las Frecuencias de Mel. Son coeficientes para la representación del habla basados en la percepción auditiva humana.

N-gramas (*N-Gram*): Técnica estadística usada para modelar la probabilidad de una secuencia de palabras o caracteres en un texto. Un n-grama es una secuencia de n elementos consecutivos.

PLP (*Physical Layer Pipe*): Enfoque alternativo a las características MFCC. Son un conjunto de características acústicas utilizadas en el reconocimiento de voz que se derivan de la señal de audio mediante un procesamiento en el dominio temporal y espectral, teniendo en cuenta las propiedades del sistema auditivo humano.

SVM (*Support Vector Machines*): Máquinas de vectores soporte. Algoritmo de aprendizaje supervisado cuyo objetivo es encontrar un hiperplano que separe de la mejor forma posible dos clases diferentes de puntos de datos.

Anexo 1: Cuestionario de usabilidad

En este anexo se incluye el contenido completo del formulario realizado para las pruebas de usabilidad de la aplicación.

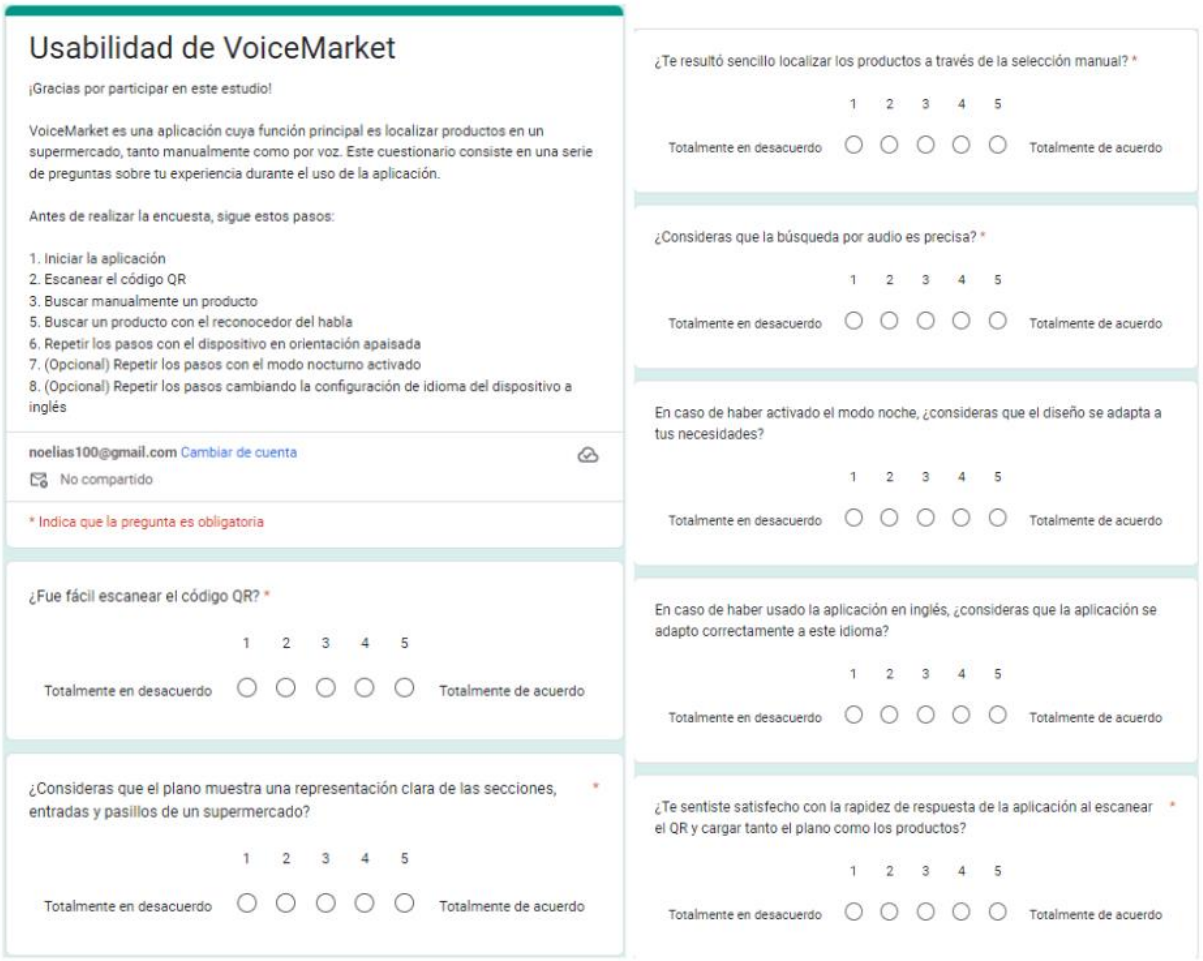

Figura 45: Primera parte del cuestionario de usabilidad

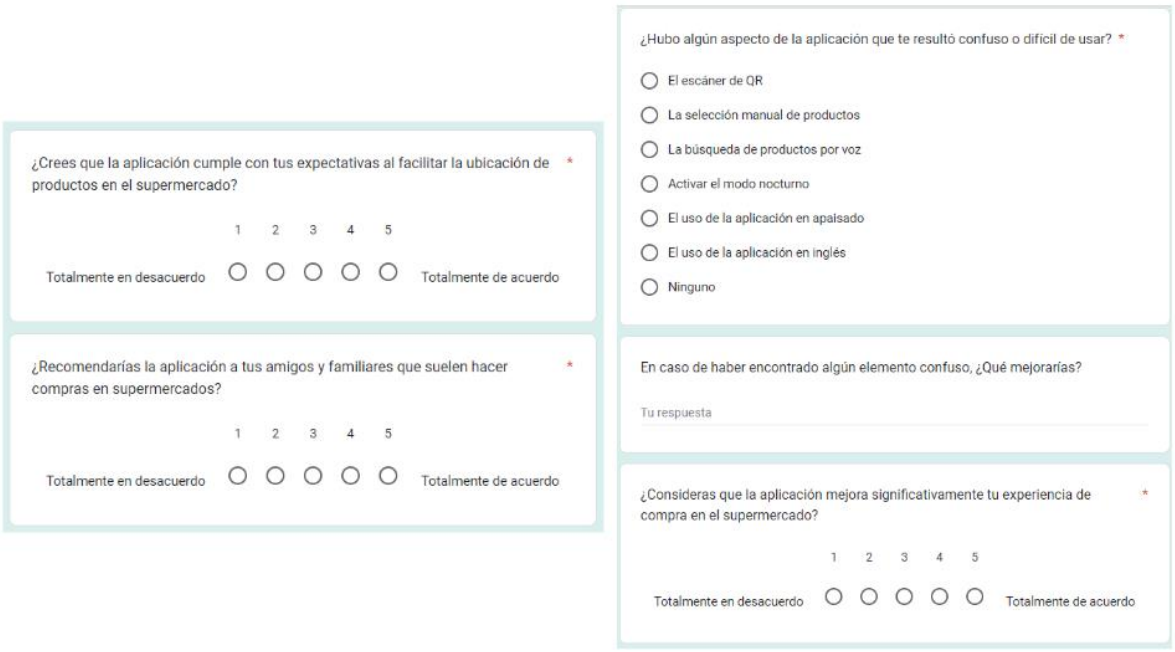

Figura 46: Segunda parte del cuestionario de usabilidad

Anexo 2: Objetivos de Desarrollo Sostenible (ODS)

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**				**X**
ODS 4. **Educación de calidad.**				**X**
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**				**X**
ODS 8. **Trabajo decente y crecimiento económico.**				**X**
ODS 9. **Industria, innovación e infraestructuras.**	**X**			
ODS 10. **Reducción de las desigualdades**.				**X**
ODS 11. **Ciudades y comunidades sostenibles**.	**X**			
ODS 12. **Producción y consumo responsables.**	**X**			
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**	**X**			

Tabla 5: Objetivos de desarrollo sostenible relacionados con el proyecto

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Los Objetivos de Desarrollo Sostenible (ODS), también llamados Objetivos Globales, fueron adoptados por las Naciones Unidas en 2015 como un llamamiento universal para poner fin a la pobreza, proteger el planeta y garantizar que para el 2030 todas las personas disfruten de paz y prosperidad. Es decir, son un conjunto de 17 metas globales que buscan abordar una amplia gama de desafíos socioeconómicos, ambientales y de desarrollo que enfrenta el mundo en la actualidad.

Todo trabajo o proyecto realizado actualmente puede estar vinculado de alguna manera a estos objetivos, aportando así un valor añadido. En lo que respecta a la aplicación desarrollada mediante este proyecto, esta guarda relación con cuatro de los ODS debido a que se trata de una aplicación orientada a mejorar la experiencia de compra en supermercados.

A continuación, se van a listar los ODS que se encuentran presentes en mayor o menor medida, justificando su implicación:

- ODS 9. Industria, innovación e infraestructura:

  La aplicación móvil desarrollada incorpora tecnologías de vanguardia como los códigos QR y el reconocimiento de voz para mejorar la experiencia de compra en supermercados. Al proporcionar una herramienta que permite la búsqueda tanto manual como por voz de productos dentro de un supermercado, la aplicación se posiciona como pionera en la implementación de soluciones avanzadas para la mejora de la experiencia del consumidor. Además, esta innovación es un reflejo directo de la evolución tecnológica en la industria minorista.

- ODS 11. Ciudades y Comunidades Sostenibles:

  La capacidad de la aplicación para ayudar a los consumidores a localizar productos específicos en los supermercados tiene implicaciones significativas para la gestión de espacios urbanos. Al disminuir el tiempo dedicado a la búsqueda de productos, es posible reducir la congestión en los establecimientos, contribuyendo a una dinámica más fluida en estos entornos. Este efecto puede promover una mayor eficiencia en la distribución y el uso del espacio, fomentando una experiencia más agradable y sostenible para los consumidores.

- ODS 12. Producción y consumo responsables:

  Una característica distintiva de la aplicación, como ya se mencionó en el ODS 11, es su capacidad para acortar el tiempo de búsqueda de productos, lo que a su vez reduce la necesidad de recorrer repetidamente los pasillos del supermercado, evitando pasar por secciones en las que no se encuentra ninguno de los productos que deseamos comprar. Esta funcionalidad podría disminuir el desperdicio de recursos y optimizar el proceso de compra. Al ser una herramienta que promueve un consumo más eficiente y consciente, la aplicación respalda el objetivo de producción y consumo responsables.

- ODS 17. Alianzas para lograr objetivos:

  Este es el objetivo más fomentado por nuestra aplicación. Aunque actualmente el supermercado mostrado en la *app* es ficticio, la perspectiva de colaboración entre supermercados y desarrolladores de aplicaciones móviles para mejorar la experiencia del consumidor destaca el potencial de las asociaciones público-privadas. Esta aplicación es un ejemplo tangible de cómo las alianzas estratégicas pueden contribuir a la consecución de metas compartidas en términos de sostenibilidad.

En conclusión, el trabajo realizado se identifica con varios de los ODS, apoyando la innovación tecnológica, la sostenibilidad, el consumo responsable y las alianzas en el sector público-privado