UNIVERSIDAD
REY JUAN CARLOS

GRADO EN INGENIERÍA INFORMÁTICA

Curso Académico 2024/2025

Trabajo Fin de Grado

APLICACIÓN NATIVA ANDROID PARA FACILITAR AL USUARIO LA GESTIÓN DE LA COMPRA DE ALIMENTOS DEL HOGAR, RECETAS E INVITADOS.

Autor : Beatriz Esteban Alcántara

Tutor : Manuel Rubio Sanchéz

Trabajo Fin de Grado/Máster

Aplicación nativa Android para facilitar al usuario la gestión de la compra de alimentos del hogar, recetas e invitados

Autor : Beatriz Esteban Alcántara

Tutor : Manuel Rubio Sanchéz

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 20XX, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Mostolés, a de de 20XX

Dedicado a

mi familia, mis amigos y mi pareja

II

Agradecimientos

Aquí vienen los agradecimientos... Aunque esté bien acordarse de la pareja, no hay que olvidarse de dar las gracias a tu madre, que aunque a veces no lo parezca disfrutará tanto de tus logros como tú... Además, la pareja quizás no sea para siempre, pero tu madre sí.

Resumen

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Índice general

Índice de figuras

Capítulo 1

Introducción y Motivación

1.1. Mercado actual de las aplicaciones móviles

Las aplicaciones móviles han transformado por completo nuestro día a día y tienen un gran impacto en cómo nos comunicamos, cómo trabajamos, cómo nos divertimos y cómo realizamos nuestras tareas.

Entre esas tareas se encuentra realizar la compra del hogar, tarea que requiere de organización si el usuario quiere realizarla de manera óptima. Este trabajo de fin de grado se enfoca en generar una herramienta que sirva de ayuda para que los usuarios la lleven a cabo.

Según los datos de 42matters, compañía que se encarga de recopilar y ofrecer datos a diferentes empresas, a 27 de enero de 2024 hay 2 millones aplicaciones Android en Google Play y algo más de 1.900.000 aplicaciones iOS en App Store, figura 1.1. De las cuales, el porcentaje de aplicaciones que los usuarios se pueden descargar de manera gratuita es aproximadamente de 95 % en ambas tiendas de aplicaciones.

Figura 1.1: Aplicaciones gratuitas y de pago, Android vs iOS.

Fuente: `https://42matters.com/stats#available-apps-count`

En cuanto a categorías, figura 1.2 y figura 1.3, la educación lidera en Google Play, mientras que los juegos son predominantes en App Store, apareciendo el "negocio" como segunda categoría en ambos casos.

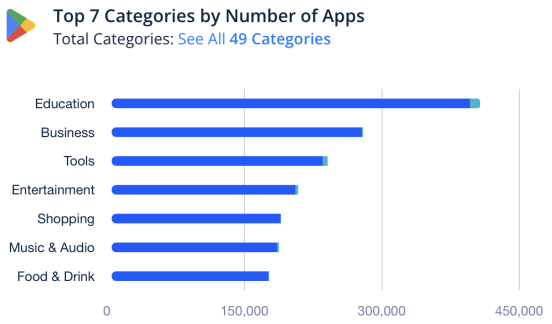

Figura 1.2: Apps por categoría Google Play

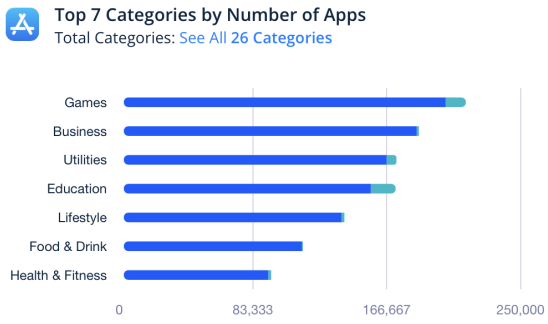

Figura 1.3: Apps por categoría App Store

Fuente: `https://42matters.com/stats#apps-by-category`

Además, se lanzan diariamente más de 2.300 nuevas aplicaciones en Google Play y unas 1.100 en App Store. Esto representa más de 90.000 aplicaciones nuevas al mes en la platafor-

ma de Android, lo que pone de manifiesto la alta competencia y dinamismo de este mercado, figura 1.4.

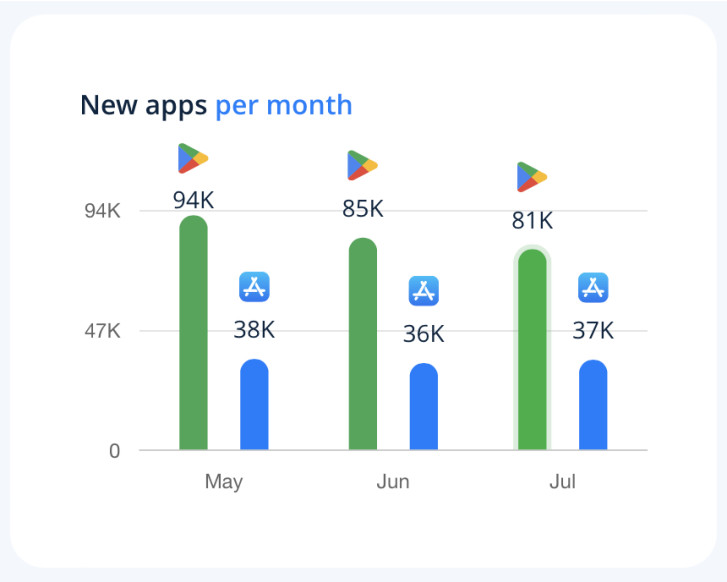

Figura 1.4: Nuevas aplicaciones por mes en cada tienda de aplicaciones (27/01/2024)

Fuente: `https://42matters.com/stats#apps-by-category`

En cuanto a los usuarios, como se puede ver en la figura 1.5, Android domina el panorama global. Según Statista (junio de 2024), su cuota de mercado alcanza el 72,15 %, frente al 27,19 % de iOS. Aunque en países como Estados Unidos e Irlanda iOS tiene más presencia, Android es el sistema operativo predominante en regiones como América Latina, África, Asia y, especialmente, España.

Figura 1.5: Mapa mundial de Android e iOS (27/01/2024)

Fuente: `https://www.statista.com`

En el caso concreto de España, que es donde se publicaría esta aplicación, el 77 % de los smartphones son Android frente a un 23 % de iOS (Statista, diciembre 2023). Este dato resulta decisivo a la hora de seleccionar la plataforma de desarrollo, ya que permite orientar el producto a la mayoría de los usuarios potenciales.

1.2. Tipos de aplicaciones móviles

El desarrollo de software para dispositivos móviles puede abordarse desde diferentes enfoques según el tipo de aplicación que se desea construir. En general, se distinguen tres tipos principales: aplicaciones nativas, aplicaciones web y aplicaciones híbridas.

Una aplicación nativa es aquella que se crea específicamente para un sistema operativo móvil, utilizando su lenguaje y herramientas oficiales. Esto permite aprovechar al máximo las capacidades del hardware del dispositivo, lo que se traduce en un mayor rendimiento y una experiencia de usuario más fluida. Por ejemplo, una aplicación desarrollada en Kotlin para Android o en Swift para iOS es considerada nativa. Este tipo de desarrollo requiere distribuir la app

a través de plataformas oficiales como Google Play o App Store y, en caso de querer abarcar varios sistemas operativos, implica desarrollar una versión distinta para cada uno.

Las aplicaciones web, por el contrario, son páginas web optimizadas para dispositivos móviles, a las que se accede desde el navegador sin necesidad de instalación. Se desarrollan con tecnologías como HTML, CSS y JavaScript, y su principal ventaja es la portabilidad entre sistemas. Sin embargo, su rendimiento es inferior al de una aplicación nativa y tienen acceso limitado a las funcionalidades del dispositivo.

Las aplicaciones híbridas combinan elementos de ambas. Básicamente, se trata de aplicaciones web que se ejecutan dentro de un contenedor nativo, lo que permite distribuirlas desde tiendas oficiales y acceder a algunas funcionalidades del hardware. No obstante, la integración entre la parte web y la parte nativa puede generar complejidades, sobre todo en términos de rendimiento y mantenimiento.

Tras analizar las características de cada enfoque, para este trabajo de fin de grado se optó por desarrollar una aplicación nativa para Android. Esta decisión responde tanto al interés por aprender en profundidad el desarrollo específico para esta plataforma, como al deseo de ofrecer una experiencia más fluida, potente y adaptada a los dispositivos que predominan en el mercado español.

1.3. Análisis del entorno

Antes de iniciar el desarrollo de la aplicación, se llevó a cabo un análisis del entorno para identificar qué soluciones digitales existen actualmente orientadas a la organización de la compra doméstica. Esta revisión permitió detectar tanto buenas prácticas como carencias que justifican este proyecto.

Una de las aplicaciones más conocidas en este ámbito es *Bring!*. Su funcionamiento se basa en la creación de listas de la compra mediante una interfaz visual con iconos organizados por categorías (por ejemplo, frutas, lácteos, panadería). Está orientada a usuarios de cualquier perfil. Sin embargo, no permite especificar cantidades con precisión, no admite comentarios o notas personalizadas para cada producto, ni se integra con recetas o planificación semanal, lo que limita su utilidad para quienes desean gestionar la compra de forma más detallada.

Otra opción popular es *Listonic*, que apuesta por un diseño más clásico: listas de verifica-

ción con productos agrupados por secciones del supermercado. Es rápida y práctica, y ofrece sugerencias inteligentes basadas en el texto introducido por el usuario. No obstante, su interfaz resulta más básica y no incorpora funcionalidades como la planificación de menús, el control del consumo habitual o la personalización de listas según hábitos domésticos o eventos.

Ambas aplicaciones ofrecen soluciones útiles para el usuario promedio, pero presentan limitaciones cuando se busca una herramienta más flexible y adaptada a la realidad diaria de una persona o familia que necesita controlar mejor lo que compra, cuándo lo compra y por qué. Esta falta de personalización y de conexión con hábitos reales de consumo doméstico justifica el desarrollo de una alternativa más completa y adaptable.

A partir de este momento, la aplicación que se desarrolla a lo largo de este proyecto la llamaremos Pinche.

A diferencia de otras soluciones del mercado, Pinche aborda el problema de la organización de la compra doméstica desde una perspectiva práctica, personalizada y centrada en la realidad cotidiana de quienes gestionan los menús y las compras del hogar.

La aplicación se estructura en tres secciones: listas de la compra, recetas e invitados. En la sección de listas, el usuario puede crear múltiples listas adaptadas a distintas ocasiones —por ejemplo, “Lista semanal” o “Cumpleaños de María”— y añadir productos con su cantidad, e incluso el supermercado donde se prefiere comprarlos. En la sección de recetas, se pueden almacenar platos con su elaboración detallada e ingredientes para un número determinado de comensales, además de añadir fácilmente esos ingredientes a cualquier lista activa si desea realizar la receta. Por último, en la sección de invitados, se puede registrar información personalizada sobre las personas que suelen comer en casa, como sus intolerancias, preferencias y qué platos se les han preparado recientemente.

Esta estructura modular convierte a Pinche no solo en una herramienta para digitalizar listas de la compra, sino en una solución doméstica integral que facilita la planificación, optimiza el tiempo y contribuye también a evitar errores comunes como compras duplicadas, olvidos o preparación de menús inadecuados para los invitados.

Desde el punto de vista académico, el proyecto ofrece un caso completo para aplicar competencias clave del grado: desarrollo de interfaces modernas con Jetpack Compose, gestión de datos con Firebase, diseño orientado al usuario y trabajo bajo metodologías ágiles.

En definitiva, Pinche es una propuesta aplicable y escalable, que puede mejorar la calidad

de vida de los usuarios al resolver un problema doméstico frecuente mediante una herramienta accesible, inteligente y bien estructurada.

Capítulo 2

Objetivos

El objetivo principal de este proyecto es comprender y aplicar todo el proceso que conlleva la creación de una aplicación desde cero. En este caso, nos centraremos en cómo facilitar al usuario la tarea de realizar la compra doméstica, tanto de alimentos como de productos del hogar, de manera eficiente y organizada. También se tendrá en cuenta cómo gestiona las recetas y los invitados que van a comer a su hogar.

Para ello el trabajo se centrará en el desarrollo de una aplicación móvil nativa para el sistema operativo Android, denominada *Pinche*. Pinche tiene cómo objetivo facilitar la tarea expuesta previamente a los usuarios.

La forma de alcanzar este objetivo consistirá en recorrer cada una de las etapas del proceso de desarrollo, adoptando los distintos roles que forman parte de un equipo digital: experiencia de usuario (UX), producto y desarrollo. Aunque el núcleo del trabajo se centrará en la parte técnica de desarrollo, también se tendrán en cuenta aspectos clave de diseño y definición del producto.

Adoptar esta perspectiva integral permite no solo aprender a programar, sino también a tomar decisiones informadas sobre las necesidades reales de los usuarios, los problemas que enfrentan en su día a día y cómo una solución tecnológica puede aportar valor real.

La motivación del proyecto radica también en aplicar y reforzar conocimientos adquiridos durante el grado, desde el diseño de interfaces hasta la integración de servicios en la nube. Pero sin permanecer como técnicos o programadores aislados que no tienen el contexto del proyecto que llevan a cabo y que entienden el por qué de las decisiones que se toman en su conjunto.

Desde ese punto de vista, el puramente técnico, se ha optado por emplear tecnologías moder-

nas del ecosistema Android: el lenguaje de programación Kotlin, el framework Jetpack Compose para la construcción de interfaces declarativas y los servicios de Firebase —específicamente Firestore como base de datos y Firebase Authentication para la gestión de usuarios—. Estas tecnologías permiten un desarrollo flexible y están alineadas con las demandas actuales del sector.

Desde la perspectiva de diseño UX se ha adoptado una combinación de la metodología Design Thinking para la definición inicial del problema y el uso de la herramienta Figma para llevar acabo un diseño final intuitivo y que sea fácilmente iterable.

Por último, desde el lado de producto y gestión del proyecto, se aplica la metodología Lean Startup para validar hipótesis y metodologías ágiles como Scrum para organizar el desarrollo en iteraciones.

El presente trabajo ofrece una oportunidad para adquirir experiencia práctica en un entorno controlado, simulando las condiciones reales del desarrollo de software en equipos multidisciplinares. Asumir diferentes roles permite entender los retos de la comunicación entre perfiles técnicos y no técnicos, la importancia de la empatía hacia el usuario final, y el valor de trabajar con un enfoque centrado en el producto y el usuario.

En resumen, el proyecto se plantea como una oportunidad para consolidar conocimientos técnicos, mejorar habilidades de diseño y comunicación, y resolver una necesidad real con impacto práctico. A través del desarrollo de esta aplicación, se busca no solo alcanzar los objetivos académicos del Trabajo de Fin de Grado, sino también generar una solución útil que podría ser utilizada por muchas personas en su vida diaria.

Capítulo 3

Tecnologías, Herramientas y Metodologías

Descripción de los lenguajes de programación, entornos de desarrollo, herramientas auxiliares, librerías de terceros, sistemas operativos, navegadores web, etc. utilizados para la realización del proyecto, así como la metodología empleada. El grado de profundidad a la hora de explicar cada tecnología dependerá de lo relevante que ha sido para el proyecto y lo conocida que es. Por ejemplo, si se usa el lenguaje de programación Java, no es necesario entrar en tanto detalle como si se usa un lenguaje mucho menos usado como Scala, por ejemplo.

Respecto a la metodología, dada la naturaleza de los proyectos, se suele describir una metodología iterativa e incremental en espiral, en la que se van sucediendo reuniones con el profesor que van definiendo el ámbito del proyecto.

Este capítulo puede tener una extensión entre 10 y 15 páginas.

3.1. Lenguaje de programación

Para implementar una aplicación nativa en Android se puede usar como lenguaje de programacion Java o Kotlin. El desarrollo de Pinche se ha realizado utilizando Kotlin.

Las principales razones por las que se ha decidido usar Kotlin para el desarrollo de esta aplicación frente a Java son:

- **Sintaxis más concisa y expresiva**: Kotlin permite reducir significativamente la cantidad de código necesario para realizar tareas comunes. Por ejemplo, la gestión de getters/setters, estructuras de datos, y operaciones sobre colecciones se realiza de forma mucho más directa y legible.

- **Seguridad frente a errores de null**: Kotlin incluye un sistema de tipos que distingue claramente entre referencias anulables y no anulables, lo que reduce la probabilidad de errores en tiempo de ejecución relacionados con punteros nulos (el clásico *NullPointerException* en Java).

- **Interoperabilidad con Java**: Kotlin es totalmente interoperable con Java, lo que permite utilizar bibliotecas existentes sin necesidad de reescribirlas. Esto es especialmente útil en Android, donde muchas APIs aún están escritas en Java.

- **Compatibilidad con herramientas modernas de Android**: Kotlin se integra de forma nativa con bibliotecas modernas como Jetpack Compose, Hilt, Coroutines o Navigation, lo que simplifica el uso de arquitecturas modernas (como MVVM) y prácticas de desarrollo actuales.

- **Soporte oficial, comunidad y futuro garantizado**: Google ha declarado que "Android is Kotlin-first", lo que significa que nuevas APIs y librerías se diseñan pensando primero en Kotlin. Esto garantiza que adoptar Kotlin no solo es una opción segura, sino también alineada con la evolución del ecosistema Android.

3.2. Frameworks y librerías

3.2.1. Jetpack Compose

3.2.2. Jetpack Compose

Para la construcción de la interfaz de usuario se ha utilizado **Jetpack Compose**, el moderno framework de Android para crear interfaces de forma declarativa. Esta elección responde a la necesidad de utilizar herramientas actuales, recomendadas por Google, que permiten una mayor productividad y una experiencia de desarrollo más fluida.

Durante muchos años, el desarrollo de interfaces en Android se ha basado en el uso de archivos `XML` que describen los elementos de la interfaz de forma estática, combinados con código Java o Kotlin para enlazar y actualizar esos elementos en tiempo de ejecución. Este enfoque impone una separación forzada entre lógica e interfaz, y obliga a utilizar métodos como

`findViewById()` o `ViewBinding`, lo que complica la sincronización entre los datos y la vista.

Jetpack Compose rompe con este modelo tradicional introduciendo una forma de construir la UI de manera totalmente **declarativa**, es decir, describiendo *qué* debe mostrarse en lugar de *cómo* hacerlo paso a paso. La interfaz se define directamente en Kotlin, mediante funciones composables, lo que permite una integración más natural con la lógica de negocio, facilita la reutilización de componentes y mejora la legibilidad del código.

Entre las principales ventajas de Jetpack Compose frente a XML destacan:

- **Menor complejidad**: No es necesario gestionar manualmente el enlace entre XML y código. La UI responde automáticamente a los cambios de estado.
- **Código más conciso y reutilizable**: Al trabajar con funciones de Kotlin, se pueden componer interfaces complejas a partir de pequeños componentes reutilizables.
- **Mejor integración con la arquitectura moderna**: Compose se integra de forma nativa con patrones como MVVM, flujos reactivos como StateFlow e incluso herramientas de testing específicas para UI.
- **Vista previa en tiempo real**: Android Studio permite visualizar cambios en la UI mientras se escribe el código, lo que acelera el diseño y validación de pantallas.
- **Mantenimiento más sencillo**: Al eliminar el archivo XML y mantener toda la lógica en un solo lenguaje, se reduce el esfuerzo de mantenimiento y depuración.

Por estos motivos, Jetpack Compose ha sido adoptado en este proyecto como la mejor solución para el desarrollo moderno de interfaces en Android. Su diseño orientado a la productividad, la simplicidad y la escalabilidad lo convierten en una herramienta alineada con el objetivo de seguir las mejores prácticas en este proyecto [2].

3.2.3. Jetpack Navigation

Para la gestión de la navegación entre pantallas se ha utilizado **Jetpack Navigation**, que permite definir flujos de navegación mediante un grafo centralizado, asegurando la coherencia del estado de la aplicación y facilitando el paso de argumentos entre distintas pantallas y componentes.

3.2.4. Hilt

Hilt ha sido utilizado como sistema de *inyección de dependencias*, simplificando la gestión de instancias y facilitando la sustitución de componentes durante el testeo. Su integración con el ciclo de vida de Android y su compatibilidad con Jetpack proporcionando la robustez y escalabilidad que buscamos [1].

3.3. Plataforma de backend: Firebase

El proyecto utiliza **Firebase** como plataforma en la nube, en concreto:

- **Firestore**, como base de datos NoSQL en tiempo real, para almacenar las listas, recetas e invitados [8].
- **Firebase Authentication**, para la gestión de usuarios básicos mediante email y contraseña [7]. Permitiendo también la gestión de flujos como recuperar contraseña.

La elección de Firebase responde a su integración nativa con Android, su facilidad de uso en proyectos de pequeño-medio alcance, y su compatibilidad con herramientas de emulación para pruebas.

3.4. Entorno de desarrollo

El entorno de desarrollo utilizado ha sido **Android Studio**, IDE oficial de Android, que proporciona herramientas avanzadas como el emulador, el profiler de rendimiento y soporte completo para Compose, Kotlin y Firebase. Para las pruebas de la app se han utilizado tanto emuladores como dispositivos físicos.

3.5. Control de versiones

Para la gestión del código fuente se ha utilizado **Git** como sistema de control de versiones y **GitHub** como repositorio remoto. El trabajo se ha organizado en ramas (`main`, `dev`, `feature/x`) siguiendo prácticas comunes en entornos colaborativos. Esto ha permitido mantener versiones estables y realizar integraciones progresivas de nuevas funcionalidades.

3.6. Metodologías de desarrollo

3.6.1. Design Thinking

Durante la fase inicial se aplicó el enfoque de **Design Thinking**, con especial énfasis en la definición del problema, centrada en el usuario objetivo y orientada a la acción. Su objetivo es generar soluciones de acuerdo con problemas detectados en un determinado marco de trabajo. [4] [5].

3.6.2. Lean Startup

3.6.3. Lean Startup

Lean Startup es una metodología que ayuda al desarrollo de productos o servicios de manera ágil, reduciendo los riesgos, promoviendo el aprendizaje y disminuyendo el tiempo de lanzamiento de dicho servicio o producto. Todo esto contribuye también a reducir los gastos y riesgos, al situar al cliente real en el centro de todas las decisiones de desarrollo [10].

Los principios fundamentales de Lean Startup son:

- **Producto Mínimo Viable (MVP)**: Se definen unas hipótesis a confirmar y se genera una versión básica del producto que cubra o simule la funcionalidad necesaria para comprobar dichas hipótesis y redirigir la definición del producto según lo que funciona y lo que no.
- **Construir-Medir-Aprender**: Se construye el MVP (producto mínimo viable), se mide su desempeño y aceptación por parte del usuario, se recopilan datos y se decide en base a ellos. Este ciclo se repite durante toda la definición del producto.
- **Experimentación continua**: Se evalúa constantemente el producto o servicio, de manera que no hay un plan fijo de acción ni se comprometen recursos desde el principio.
- **Iteraciones rápidas**: La generación de MVPs debe ser ágil, demostrando flexibilidad a la hora de aplicar los cambios necesarios tras la última iteración.
- **Validación de hipótesis**: Se utilizan métricas y datos que ayudan a vislumbrar si una idea de negocio tiene mercado o no, evitando el gasto innecesario de recursos y tiempo.

3.6.4. Agile

La metodología **Agile** se centra en la flexibilidad, la colaboración y la entrega incremental de valor al cliente durante el desarrollo de un proyecto. Con esta metodología se pretende generar una mayor adaptabilidad frente a los cambios que surgen durante el desarrollo, reduciendo los tiempos de entrega y mejorando la calidad del producto, con lo que se consigue que el cliente final esté más satisfecho.

Agile se basa en el *Manifiesto Ágil*, publicado en 2001, que consta de cuatro valores principales y doce principios.

Valores del Manifiesto Ágil:

- Individuos e interacciones frente a procesos y herramientas.
- Software funcional sobre documentación extensiva.
- Colaboración con el cliente frente a negociación de contratos.
- Responder ágilmente al cambio frente a seguir un plan rígido.

Principios:

- Satisfacer al cliente mediante la entrega rápida y continua de software funcional.
- Aceptar los cambios, los requisitos por cumplir pueden ser incluidos en cualquier etapa del desarrollo.
- Entregar frecuentemente un producto funcional, en ciclos cortos de tiempo.
- Colaboración constante con el cliente para asegurar que se cubren todas sus necesidades.
- Motivar a los equipos de trabajo para fomentar su implicación y confianza.
- Comunicación cara a cara como medio más eficiente y efectivo.
- Valorar el software funcional como principal medida de progreso.
- Sostenibilidad mediante una velocidad constante y sostenible para que el ritmo al que trabaja el equipo sea accesible.

- Atención a la excelencia técnica y al buen diseño.
- Simplicidad, eliminando lo innecesario o lo que no aporta valor real.
- Equipos autoorganizados con capacidad de decisión.
- Revisión y ajuste constantes para mejorar la efectividad.

Algunas metodologías que surgen del enfoque Agile son **Scrum**, **Kanban** y **Extreme Programming (XP)**.

Scrum. Es una metodología de trabajo que permite el manejo de proyectos complejos permitiendo que los equipos trabajen de manera iterativa e incremental. El trabajo se organiza en **Sprints**, que son ciclos cortos de tiempo que tienen como objetivo un incremento funcional del producto que el equipo está desarrollando.[11]

Entre sus principales elementos dentro de la metodología Scrum encontramos roles calves como Product Owner, Scrum Master y el equipo de desarrollo. El **Product Owner** es el responsable de definir cuáles son las prioridades que maximizan el valor del producto. El **Scrum Master** es el encargado de facilitar el proceso Scrum y asegurarse que se aplica correctamente eliminando posibles dificultades que le pueden surgir al equipo con esta metodología. Y, por último, el **equipo de desarrollo** es el encargado de implementar el producto.

Dentro de esta metodología también encontramos elementos o herramientas clave que facilitan su aplicación: **Product Backlog**, que consiste en una lista priorizada de todas las tareas del proyecto, **Sprint Backlog**, que abarca todas las tareas a realizar en un sprint, y el **incremento** que sería el producto funcional que entrega el equipo de desarrollo al final de cada sprint.

Para la gestión del tiempo dentro de un sprint, Scrum recomienda realizar cuatro eventos: Sprint Planning, Daily Scrum, Sprint Review y Sprint Retrospective. En el **Sprint Planning** el objetivo será planificar qué tareas se van a desarrollar este sprint, teniendo en cuenta su dificultad y el tiempo que se estima que se va a tardar en desarrollarlas. Las tareas deben ser lo más atómicas posibles. Diariamente el progreso se coordinará en la sesión de **Daily Scrum**, será el momento de poner sobre la mesa posibles bloqueos que el equipo se haya encontrado o los avances que ha ido realizando. Una vez finalizado el sprint tendremos el **Sprint Review** donde se revisará que cantidad de trabajo ha sido realizado y por último una sesión en la que

el equipo pueda reflexionar sobre cómo mejorar para el próximo Sprint que será la sesión de **Sprint Retrospective**.

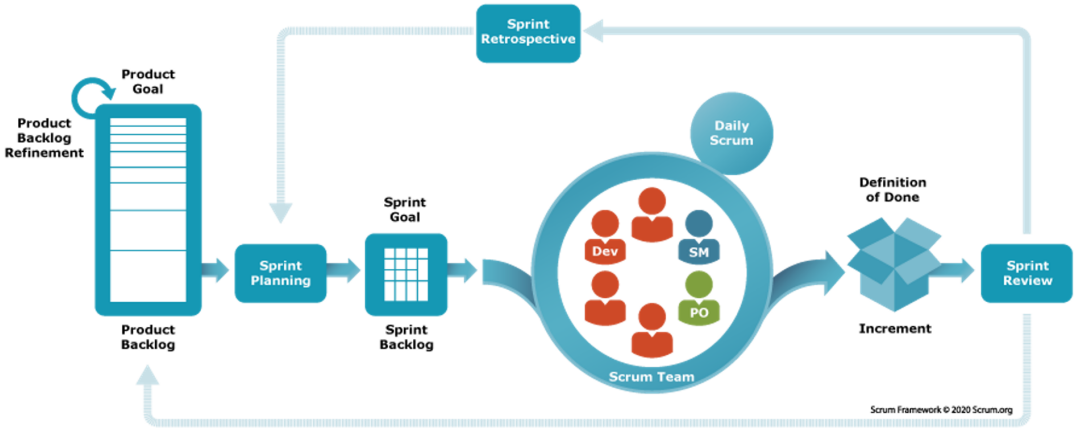

Figura 3.1: Esquema de funcionamiento de Scrum.

Fuente: `https://www.scrum.org/learning-series/what-is-scrum/`

Kanban. Esta metodología utiliza un tablero visual (*Tablero Kanban*) para representar el flujo de trabajo. Las tareas se representan como tarjetas que se mueven por columnas como *Por hacer*, *En progreso* y *Terminado*. Se establece un límite de trabajo en curso (WIP: Work In Progress, trabajo en progreso) para evitar sobrecargas. Kanban permite detectar cuellos de botella, optimizar procesos y mejorar de forma continua la eficiencia del equipo.

Extreme Programming (XP). XP enfatiza la mejora continua y la satisfacción del cliente. Reduce riesgos mediante prácticas técnicas rigurosas:

1. Desarrollo iterativo con entregas frecuentes de software funcional.
2. Pruebas constantes (TDD: Test-Driven Development, desarrollo guiado por tests).
3. Programación en pareja. Dos desarrolladores trabajan simultáneamente en la misma tarea con el propósito de reducir errores y mejorar el aprendizaje del equipo.
4. Integración continua.
5. Código simple y funcional.

6. Retroalimentación rápida del cliente. El cliente está presente de manera frecuente en el proceso de desarrollo y proporciona su punto de vista para ajustar el desarrollo a sus necesidades

3.7. Diseño de la interfaz

La interfaz ha sido diseñada siguiendo principios de **UX/UI centrados en la simplicidad, accesibilidad y claridad visual**. Se han creado prototipos en **Figma**, permitiendo iterar sobre la estructura de navegación, la jerarquía visual y la organización de las secciones: listas, recetas e invitados.

3.8. Herramientas de testeo

Durante el desarrollo se ha planificado el uso de diferentes tipos de pruebas con herramientas específicas según el nivel:

Tipo de Test	**Herramientas**	**Descripción**
Unitarios	JUnit4, MockK	Pruebas de lógica de negocio y mo pendencias.
UI (Compose)	`ui-test-junit4`	Verificación de componentes de int
Integración	Firebase Emulator Suite, Fake Repositories + Hilt	Pruebas aisladas de servicios y fluj
End-to-End (E2E)	Espresso, Firebase Emulator Suite	Automatización de flujos de usuari
Cobertura	Jacoco	Generación de informes de cobertu

Cuadro 3.1: Resumen de herramientas de testeo utilizadas

Estas herramientas permiten asegurar que la aplicación cumple sus requisitos funcionales y no funcionales, y que es robusta frente a cambios [3, 9, 6].

Capítulo 4

Descripción informática

Descripción del proyecto realizado. Después de unos párrafos introductorios el capítulo se divide en subcapítulos. (de 25 a 35 páginas)

4.1. Requisitos

Descripción detallada de las funcionalidades que tendría que implementar la aplicación (pues se asume que los requisitos se escriben antes de empezar el desarrollo). Pueden tener forma de historias de usuario o bien ser una lista de requisitos funcionales y no funcionales.

4.2. Arquitectura y Análisis

Descripción de los aspectos de alto nivel de la aplicación. Diagramas de clases de análisis, diagramas de clases de diseño, etc. Se debe incluir la suficiente información para que el lector pueda entender la estructura de alto nivel del software desarrollado. Se pueden incluir diagramas de casos de uso si se considera útil.

4.3. Diseño e Implementación

Descripción de algún aspecto relevante de la implementación que quiera mencionarse. Por ejemplo se podría incluir alguno de los siguientes aspectos:

- Algoritmo complejo que se haya tenido que desarrollar.

- Integración entre librerías problemática.
- Resolución de algún bug que haya sido especialmente problemático.
- Focalizar en alguna parte del desarrollo y describirla en más detalle.

En esta sección se pueden incluir fragmentos de código fuente. También se pueden incluir algunas métricas del proyecto (número de clases, líneas de código, etc.). También se puede incluir la evolución del repositorio de GitHub (gráfico de commits por día).

4.4. Pruebas

En esta sección se describen las pruebas automáticas que han sido implementadas para el proyecto. Sobre los tests, conviene indicar la cobertura del código. Si no se han implementado pruebas automáticas, deberían haberse implementado y describirse aquí o tener una buena justificación de por qué no se han implementado.

Capítulo 5

Conclusiones y trabajos futuros

Reflexión sobre el trabajo realizado, qué objetivos se han cumplido y qué aspectos quedan pendientes para una futura ampliación del proyecto. Además, se deben incluir unas conclusiones personales indicando lo que ha supuesto para el alumno la realización del trabajo. Entre 2 y 4 páginas.

Bibliografía

[1] Android Developers. Dependency injection with hilt. https://developer.android.com/training/dependency-injection/hilt-android, 2024. Consultado el 10 de diciembre de 2024.

[2] Android Developers. Jetpack compose. https://developer.android.com/jetpack/compose, 2024. Consultado el 10 de diciembre de 2024.

[3] Android Developers. Test apps on android. https://developer.android.com/training/testing, 2024. Consultado el 10 de diciembre de 2024.

[4] Design Thinking España. ¿qué es design thinking? https://designthinkingespaÃśa.com, 2024. Consultado el 10 de diciembre de 2024.

[5] Design Thinking España. ¿qué es design thinking? https://designthinkingespa%C3%B1a.com/#:~:text=El%20Design%20Thinking%20es%20una,muy%20poco%20tiempo%20soluciones%20innovadoras., 2024. Consultado el 10 de diciembre de 2024.

[6] EclEmma. Jacoco java code coverage library. https://www.eclemma.org/jacoco. Consultado el 10 de diciembre de 2024.

[7] Google Firebase. Firebase authentication. https://firebase.google.com/docs/auth, 2024. Consultado el 10 de diciembre de 2024.

[8] Google Firebase. Firebase firestore documentation. https://firebase.google.com/docs/firestore, 2024. Consultado el 10 de diciembre de 2024.

[9] Google Firebase. Test your app with firebase emulator suite. `https://firebase.google.com/docs/emulator-suite`, 2024. Consultado el 10 de diciembre de 2024.

[10] E. Ries. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Publishing Group, 2011.

[11] Scrum.org. What is scrum. `https://www.scrum.org/learning-series/what-is-scrum/`, 2024. Consultado el 10 de diciembre de 2024.

Apéndice A

Manual de usuario