Universitat Oberta de Catalunya

uoc.edu

Desarrollo de Aplicación móvil Android: Gestor de herramientas: Product Manager

Eric Peregrina Montfort
Grado de Ingeniería Informática
Desarrollo de Aplicaciones para dispositivos Móviles (Android)

David Escuer Latorre
Jordi Almirall López

Carles Garrigues Olivella

01/2024

Licencias alternativas (elegir alguna de las siguientes y sustituir la de la página anterior)

A) Creative Commons:

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento 3.0 España de Creative Commons

B) GNU Free Documentation License (GNU FDL)

Copyright © AÑO TU-NOMBRE.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3

C) Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Desarrollo de Aplicación Móvil Android: Gestor de herramientas*
Nombre del autor:	*Eric Peregrina Montfort*
Nombre del consultor/a:	*David Escuder Latorre* *Jordi Almirall López*
Nombre del PRA:	*Carles Garrigues Olivella*
Fecha de entrega (mm/aaaa):	01/2024
Titulación::	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*Desarrollo aplicaciones dispositivos móviles (Android)*
Idioma del trabajo:	Castellano
Palabras clave	*Android, Kotlin, MVVM, Clean Architecture, Firestore*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El desarrollo de este proyecto tiene como finalidad la creación de una aplicación móvil desarrollada en Android para gestionar el registro, alquiler y consulta de herramientas de una empresa.

El comportamiento de la aplicación dependerá de los dos roles existentes: Usuario y Administrador.

Por una parte, el perfil de usuario permite registrar el alquiler de una herramienta a partir del escaneo de su código de barras, consultar la disponibilidad de una herramienta, gestionar una reserva, consultar su historial de alquileres y abrir un parte de incidencia en caso de rotura o desgaste de una herramienta. Por otra parte, el perfil de administrador permite crear y eliminar herramientas, proyectos y zonas de trabajo así como consultar los alquileres y gestionar incidencias creadas por usuarios.

La metodología a seguir en el desarrollo de la aplicación es la de cascada, por lo que se establece el orden de la misma en la implementación del proyecto. Así pues, este trabajo está compuesto por el análisis y obtención de requisitos, el diseño centrado en el usuario, la implementación y verificación del desarrollo del software y la fase de mantenimiento, donde se realiza la entrega del proyecto.

En conclusión, se han abarcado todas las fases de desarrollo que se necesitan para crear una aplicación móvil. Para ello se han utilizado distintas tecnologías que han sido de gran ayuda en la implementación de las funcionalidades

esperadas y han aportado aspectos de organización y mejora de código que se requieren en el desarrollo de una aplicación móvil.

Abstract (in English, 250 words or less):

The development of this project has the objective of the creation of a mobile application on Android to manage the registration, rental and consultation of a company's tools.

The behavior of the application will depend on the two existing roles: User and Administrator.

On the one hand, the user profile allows you to register the rental of a tool by scanning its barcode, check the availability of a tool, manage a reservation, consult your rental history and open an incident report in case of breakage or wear of a tool. On the other hand, the administrator profile allows you to create and delete tools, projects and work areas as well as consult rentals.

The methodology to follow in the development of this project is the waterfall, which establishes its order in the implementation of the project. Therefore, this project is composed of the analysis and obtaining of requirements, the user-centered design, the implementation and verification of the software development and the maintenance phase, where the delivery of the project is carried out.

In conclusion, all the development phases needed to create a mobile application have been covered. For this, different technologies have been used that have been of great help in the implementation of the expected functions. Also they have provided aspects of organization and code improvement that are required in the development of a mobile application.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Las empresas que disponen de herramientas y dispositivos de instrumentación almacenados en el mismo espacio y que se emplean en varios proyectos al mismo tiempo suelen tener problemas de desconocimiento de su ubicación y desgaste. Los trabajadores, cuando necesitan una herramienta van a buscarla y se la llevan a su lugar de trabajo. Una vez finalizada la tarea, el trabajador tiene la obligación de devolver la herramienta al almacén para que el siguiente empleado que la necesite pueda encontrarla con facilidad. Sin embargo no siempre ocurre de ese modo, y en muchas ocasiones se desconoce la ubicación de las herramientas e incluso llegan a perderse.

De igual manera ocurre con el uso continuado y desgaste de las mismas, los trabajadores no siempre informan al personal responsable para que reponga o repare el utensilio si fuera necesario.

El desarrollo de esta propuesta de aplicación está pensado y focalizado precisamente para evitar o mitigar estos problemas, mediante un sistema de registro y control de las diferentes herramientas utilizadas en un mismo sector o departamento. Todo ello ayudará a conocer mejor los recursos de los que se dispone o de los que queremos organizar, y facilitará en gran medida las tareas a realizar por los trabajadores y empresas, reduciendo el tiempo empleado en la búsqueda y localización de las herramientas a utilizar.

Existen otras aplicaciones de búsqueda y préstamos de productos similares, como la aplicación utilizada por las bibliotecas municipales, tales como Aladi “Biblioteques XBM” donde solo se necesita un carnet de usuario y un código de barras que identifica el producto.

A continuación se muestran algunas capturas de la aplicación de préstamos bibliotecarios “Biblioteques XBM”

Ilustración 1 - Gestión de reservas

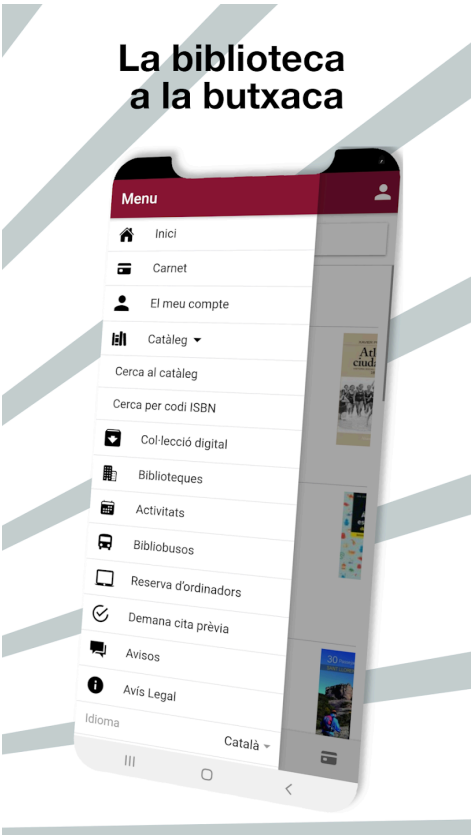


Ilustración 2 - Menú principal usuario

También existen otras aplicaciones dedicadas a la compra de herramientas como la de Leroy Merlin. Esta app incorpora características interesantes que aportan usabilidad para el usuario, como por ejemplo el botón de escanear códigos de barras a través de la cámara, que evita que el usuario tenga que introducir los números del código de forma manual.

Ilustración 3 -Aplicación Leroy Merlin

Estas aplicaciones tienen una gestión de productos robusta, sencilla y usuable, sin embargo son pocas las empresas que puedan disponer de un sistema similar. Actualmente, esta gestión de herramientas se está realizando en un documento de excel, donde los usuarios rellenan voluntariamente los datos.

DESCRIPTION	LOC IN	SETUP LOC	ID User	NAME	DATE IN	DATE OUT
Multimeter FLUKE 77 IV	LAB	OPTICAL ROOM		Jorge J.	17/12/2012	25/10/20
SET LLAVES FIJAS DURATOOL 6-18 (FALTA LA 9)	LAB	CLEAN ROOM		Jorge G.	22/01/2023	22/01/20
SET LLAVES FIJAS CHROM-VANADIUM	LAB	CLEAN ROOM		Jorge G.	22/01/2023	22/01/20
KIT DESTORNILLADORES FACOM	LAB	GREY ROOM		Eric P.	26/02/2023	26/02/20
SET DESTORNILLADORES FACOM TORX	LAB	GREY ROOM		Eric P.	26/02/2023	26/02/20
JUEGO LLAVES ALLEN DIN 911 (FALTAN 2)	LAB	GREY ROOM		Eric P.	26/02/2023	26/02/20
F.A AGILENT E3644A	LAB	OFFICES		Oscar M.	26/02/2023	
Multimeter FLUKE III	LAB	OFFICES		Carles P.	02/03/2023	04/08/20
PIE DE REY	LAB	OPTICAL ROOM		Jorge J.	06/03/2023	18/05/20
F.A AGILENT E3631A NOT WORKING	LAB	OFFICES		Juan B.	11/03/2023	11/03/20
F.A AGILENT E3631A NOT WORKING	LAB	OFFICES		Eric P.	11/03/2023	11/03/20
F.A AGILENT E3631A NOT WORKING	LAB	OFFICES		Juan B.	11/03/2023	
F.A AGILENT E3631A NOT WORKING	LAB	OFFICES		Juan B.	11/03/2023	
CORTA CABLES RS	LAB	OTHERS		Laia G.	16/07/2023	
F.A AGILENT E3633A	LAB	OTHERS		Juan B.	28/07/2023	28/07/20
F.A AGILENT E3640A	LAB	OTHERS		Juan B.	28/07/2023	08/09/20
PS Hewlett Packard E3620A	LAB	OTHERS		Juan B.	28/07/2023	
F.A AGILENT E3633A	LAB	OFFICES		Jorge G.	28/07/2023	28/07/20
Multimeter FLUKE 77 IV	LAB	OFFICES		Jorge G.	28/07/2023	25/09/20
Multimeter FLUKE III	LAB	OFFICES		Jorge G.	28/07/2023	04/08/20
PULSE GENERATOR AGILENT 81160A 330/500MHz	LAB	OFFICES		Juan B.	15/09/2023	

Ilustración 4 - Gestión de alquiler de herramientas en Excel

Como se puede observar, el sistema actual no tiene ningún sistema de identificación para las herramientas y además de que no resulta nada práctico, por lo que se ha decidido desarrollar una aplicación para cubrir estas y otras necesidades.

1.2 Objetivos del Trabajo

Una vez realizado el estudio de mercado y una comparación con el sistema actual, podemos obtener una primera idea de los requisitos funcionales (OF) y no funcionales (ONF) a implementar en nuestra aplicación:

OF1 - Un usuario se debe poder registrar en el sistema mediante su nombre, correo y contraseña en la base de datos Firebase Realtime Database.

OF2 - Un usuario se debe poder registrar mediante su cuenta de Google por medio de Firebase Authentication

OF3 - La interfaz de login y registro debe ocultar la contraseña introducida

OF4 - El administrador debe poder registrar proyectos y zonas de trabajo mediante el nombre en la base de datos de Firebase Realtime Database

OF5 - El administrador debe poder registrar herramientas mediante el proyecto al que pertenece y la zona de trabajo en la que se ubica.

OF6 - Registrar el alquiler de una herramienta en la base de datos Firebase Realtime Database a partir del número de usuario, código y nombre de la herramienta y fecha de préstamo.

OF7 - Registrar la finalización del préstamo de una herramienta en la base de datos cuando el usuario devuelva la misma.

OF8 - Actualizar el estado de una herramienta cuando la misma sea devuelta.

OF9 - La reserva de una herramienta será como máximo un usuario.

OF10 - El sistema enviará un correo electrónico al usuario suscrito a una herramienta sobre el nuevo estado de la misma mediante Firebase Cloud Messaging

OF11 - Tanto el usuario como el administrador debe poder buscar herramientas por medio de un buscador introduciendo el nombre o código de la misma.

OF12 - Informar en los resultados del buscador toda la información relevante, como por ejemplo: nombre, código, ubicación, imagen, disponibilidad y prestatario (en caso de no disponibilidad)

OF13 - El usuario debe poder notificar al administrador sobre la existencia de una herramienta desgastada o rota mediante una sección de incidencias

OF14 - El sistema enviará un correo electrónico al administrador cuando se registre una incidencia.

ONF1 - Crear una aplicación amigable, fácil de usar e intuitiva

ONF2 - Mostrar la información de forma clara e intuitiva con la ayuda de mensajes de texto

ONF3 - Implementar un sistema fiable sin errores entre transiciones de vistas

ONF4 - Asegurar que la comunicación con aplicaciones externas es buena sin errores.

ONF5 - La comunicación con la base de datos debe ser rápida.

ONF6 - Los datos modificados deben estar actualizados para todos los usuarios

ONF7 - Asegurar que los datos de usuario están protegidos

ONF8 - Implementar un sistema de identificación de herramientas y usuarios intuitiva

ONF9 - Los resultados del buscador de herramientas deben ser intuitivos

1.3 Enfoque y método seguido

El desarrollo de la aplicación se define como un producto existente que se adapta a las empresas que tengan la necesidad de instaurar un sistema de gestión de herramientas y/o productos.

La metodología a seguir en el desarrollo de este proyecto es la de cascada, puesto que la definición de requisitos, funcionalidades, tiempo y fases están bien definidas. Para ello, se expondrán todos requisitos funcionales que debe tener la aplicación, se puntuará según su coste estimado y se expondrán en una tabla basada en el modelo Kanban para dar más visibilidad a las tareas prioritarias. Por otro lado, a medida que se vaya desarrollando la aplicación, se irán aplicando los test unitarios y de integración para asegurar un buen funcionamiento y un producto funcional en todo momento.

En cuanto al entorno de programación se usará el Android Studio, ya que es el IDE oficial para programación de dispositivos móviles Android. Su uso nos aporta una serie de ventajas, como por ejemplo la compilación y su uso de Gradle, el cual automatiza la descarga de dependencias necesarias para ejecutar el proyecto, o su emulador de dispositivos para ejecutar el proyecto en un entorno simulado entre otras. El lenguaje de programación que se utilizara será Kotlin debido a la facilidad de aprendizaje, simplicidad para realizar una operación (es más conciso), y las diversas herramientas que aporta.

Para registrar datos y poderlos adquirir necesitaremos una base de datos. Se ha escogido utilizar la librería Firebase Realtime Database para esta funcionalidad, pues es una plataforma que nos ofrece diferentes herramientas multiplataforma para el desarrollo de la aplicación. Otra característica importante a resaltar, es que Firebase Realtime Database nos ofrece la posibilidad de crear una base de datos local sin necesidad de tener conexión a internet, lo que nos permite tener la aplicación operativa en todo momento.

En cuanto a las notificaciones a los usuarios que estén suscritos a una herramienta se utilizará Firebase Cloud Messaging, un servicio de mensajería multiplataforma que permite enviar mensajes de forma segura y gratuita.

Finalmente, en cuanto al aspecto de seguridad, al no tener suficiente experiencia en Android y Kotlin se tendrán que realizar varias pruebas con las herramientas que ofrece Google y Android. Por ello, con el fin de tener una mejor organización sobre las versiones desarrolladas, se trabajará con el servicio Github. De esta manera se pretende tener un producto funcional siempre en la nube y así tener una copia de seguridad ante posibles problemas.

1.4 Planificación del Trabajo

Para definir la planificación del trabajo tendremos en cuenta la cantidad de horas estimadas que el alumno puede asumir y la planificación temporal para representar las horas estimadas de cada tarea. De esta manera, obtendremos una estimación de la cantidad de trabajo a realizar en un plan de trabajo.

Dedicación Personal

Con el fin de conseguir una aproximación de las horas que el alumno deberá dedicar para realizar el proyecto primero se realizará una estimación de la dedicación personal que el alumno puede asumir. Para ello se tendrá que tener en cuenta que el alumno trabaja a jornada completa.

Así pues, la dedicación personal es la siguiente:

- L-V: 3 horas
- S-D: 5 horas

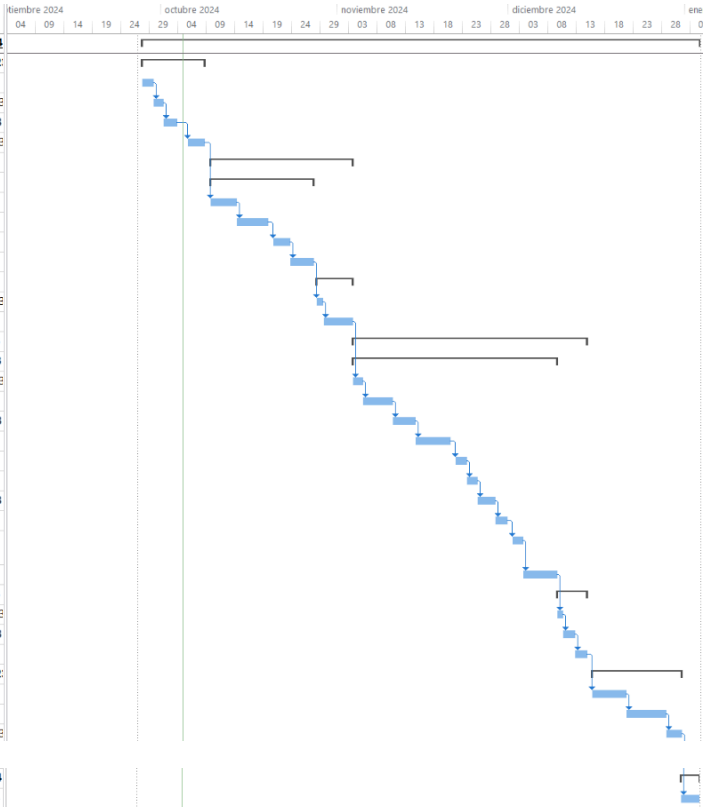
Planificación Temporal

La planificación temporal se representará mediante un diagrama de Gantt en el que podremos ver las diferentes tareas que componen cada actividad. Esto nos ayudará a tener una mejor organización y ejecución de las diferentes actividades del proyecto y tendremos mayor visibilidad del orden de ejecución de las tareas.

- Diagrama de Gantt

  Para realizar el diagrama de Gantt deberemos tener en cuenta que el inicio del proyecto se establece el día 28 de Septiembre del 2023. Para facilitar el cálculo de días necesarios según las horas introducidas, se ha activado el modo de programación automática y se ha configurado el horario acorde con la dedicación personal del estudiante.

Nombre de tarea	Duración	Comienzo	Fin
App: Gestión de herramientas	**350 horas**	**mié 27/09/23**	**mié 03/01/24**
Plan de Trabajo	**41 horas**	**mié 27/09/23**	**dom 08/10/2**
Contexto y justificación	8 horas	mié 27/09/23	vie 29/09/23
Objetivos del trabajo	8 horas	vie 29/09/23	dom 01/10/23
Enfoque y metodo elegido	9 horas	dom 01/10/23	mar 03/10/23
Planificación del trabajo	16 horas	jue 05/10/23	dom 08/10/23
Diseño y arquitectura	**89 horas**	**lun 09/10/23**	**vie 03/11/23**
Diseño del usuario	**65 horas**	**lun 09/10/23**	**vie 27/10/23**
Usuarios	16 horas	lun 09/10/23	sáb 14/10/23
Diseño conceptual	21 horas	sáb 14/10/23	jue 19/10/23
Prototipado	15 horas	vie 20/10/23	lun 23/10/23
Evaluación	13 horas	lun 23/10/23	vie 27/10/23
Diseño técnico	**24 horas**	**sáb 28/10/23**	**vie 03/11/23**
Definición de casos de uso	8 horas	sáb 28/10/23	dom 29/10/23
Diseño de la arquitectura	16 horas	dom 29/10/23	vie 03/11/23
Implementación	**148 horas**	**vie 03/11/23**	**jue 14/12/23**
Desarrollo de la app	**130 horas**	**vie 03/11/23**	**sáb 09/12/23**
Login	8 horas	vie 03/11/23	dom 05/11/23
Panel de usuario	16 horas	dom 05/11/23	vie 10/11/23
Panel de administrador	16 horas	vie 10/11/23	mar 14/11/23
Registro de herramientas	24 horas	mar 14/11/23	lun 20/11/23
Registro de proyectos	8 horas	mar 21/11/23	jue 23/11/23
Registro de localizaciones	8 horas	jue 23/11/23	sáb 25/11/23
Mis prestamos	10 horas	sáb 25/11/23	mar 28/11/23
Buscador de herramientas	8 horas	mar 28/11/23	jue 30/11/23
Información de herramienta y reserva	12 horas	vie 01/12/23	dom 03/12/23
Gestión de notificaciónes	20 horas	dom 03/12/23	sáb 09/12/23
Test	**18 horas**	**sáb 09/12/23**	**jue 14/12/23**
Pruebas unitarias	5 horas	sáb 09/12/23	dom 10/12/23
Pruebas de integración	5 horas	dom 10/12/23	mar 12/12/23
Mantenimiento	8 horas	mar 12/12/23	jue 14/12/23
Entrega final	**60 horas**	**vie 15/12/23**	**dom 31/12/2**
Mejoras	24 horas	vie 15/12/23	jue 21/12/23
Memoria	24 horas	jue 21/12/23	jue 28/12/23
Manual de usuario	12 horas	jue 28/12/23	dom 31/12/23
Defensa	**12 horas**	**dom 31/12/2**	**mié 03/01/24**
Presentación del trabajo final	12 horas	dom 31/12/23	mié 03/01/24



Como podemos observar, se necesita un total de 350 horas para el desarrollo de todo el proyecto. La defensa del TFG está programada para las fechas del 22 al 26 de Enero del 2024, por lo que nos deja un margen considerable para cualquier contratiempo.

1.5 Breve sumario de productos obtenidos

Al finalizar el proyecto, se entregarán los siguientes productos:

- Archivo de instalación de la aplicación para dispositivos Android.
- Manual de usuario
- Proyecto completo desarrollado en Android Studio
- Memoria del proyecto
- Presentación

1.6 Breve descripción de los otros capítulos de la memoria

Capítulo 1. Introducción. Se introduce el contexto del proyecto cubriendo aspectos como la necesidad y justificación para llevarlo a cabo, estudio de mercado, objetivos y requisitos del proyecto, enfoque y metodología escogida, además de la planificación del trabajo.

Capítulo 2. Diseño centrado en el usuario. Se elabora el diseño centrado en el usuario, definiendo el contexto de los mismos y los diferentes escenarios planteados. Además se define y evalúa el comportamiento del sistema con los flujos de navegación y el posterior desarrollo de prototipo.

Capítulo 3. Diseño Técnico. Definición de casos de uso, patrón de diseño y arquitectura a seguir.

Capítulo 4. Desarrollo. Se expone todo lo relativo al desarrollo de la estructura de la aplicación.

Capítulo 5. Estado actual del proyecto. Se evalúa el progreso del proyecto en función a la planificación realizada.

Capítulo 6. Pruebas. Se comprueba el correcto funcionamiento de la aplicación realizando los diferentes test definidos.

Capítulo 7. Conclusiones. Se analiza todo el desarrollo del proyecto, exponiendo tanto los puntos fuertes y débiles que se han encontrado como posibles mejoras a tener en cuenta.

Capítulo 8. Glosario. Guia de referencia para posibles términos utilizados.

Capítulo 9. Bibliografía. Fuentes de referencia utilizadas durante el desarrollo de la aplicación.

Capítulo 10. Anexo. Manual de usuario

2. Diseño centrado en el usuario y Diseño Técnico

Una vez definidos los requisitos del proyecto, empieza la etapa del diseño centrado en el usuario y el diseño técnico. Estas dos fases son de gran importancia, pues cubren aspectos que definen el comportamiento final del producto en base a la experiencia de usuario y la definición técnica de cómo implementarlo.

El diseño centrado en el usuario (DCU), se centra en obtener y cubrir todas las funcionalidades que un usuario pueda necesitar con el fin de hacerle la vida más fácil. Este apartado, es de gran importancia, pues la facilidad de uso de una aplicación se traduce en una gran experiencia de usuario y con ello crear una buena imagen de la marca.

Para llevar a cabo este paradigma, se ponen en práctica sus 3 métodos: Análisis, diseño y evaluación. En el método de análisis se realiza un estudio centrado en el uso de la aplicación a diferentes perfiles de usuarios con el fin de identificar sus objetivos y necesidades. Tras ello, empieza el método de diseño, en el que se realiza la creación y desarrollo de la arquitectura junto con el primer prototipo, para finalmente, en el método de evaluación, examinar de forma iterativa el cumplimiento de aspectos como las necesidades, y objetivos definidos por los usuarios.

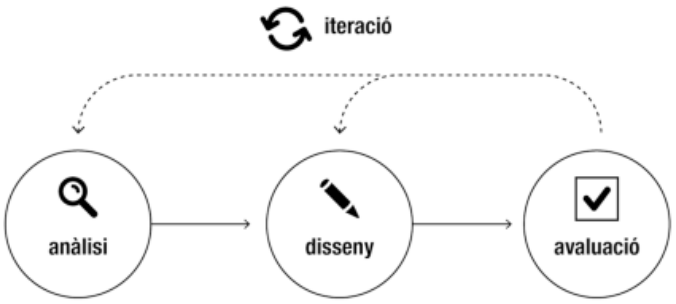


Ilustración 5 - Paradigma de diseño centrado en el usuario (DCU)

2.1 Analisis

2.1.1 Métodos de investigación

Como ya se ha mencionado anteriormente, el diseño centrado en el usuario (DCU) tiene como objetivo conocer el perfil de usuario para identificar el contexto de uso, ponerlo en contexto e identificar sus objetivos, necesidades.

Para llevar a cabo esta tarea, existen diferentes técnicas de indagación. Todas ellas son compatibles entre sí y tienen como finalidad obtener la información, objetivos y contextos de uso de los usuarios.

A continuación se describen las técnicas de investigación existentes para conocer en detalle su enfoque y contexto.

- **Observación e investigación contextual:** Dan a conocer el entorno en el que los usuarios se desarrollan y en qué condiciones lo hacen.
- **Shadowing (método de seguimiento):** Consiste en observar a los usuarios en su vida cotidiana de la manera más discreta posible. Se recogen datos de sus acciones, patrones de comportamiento, hábitos, rutinas en el contexto y cotidianidad de su vida. Se considera esta técnica apta para utilizar, pues su método se podría aplicar en el ambiente laboral del estudiante para detectar las necesidades de los

trabajadores. Con tal de respetar la intimidad de los mismos, su método de seguimiento se realizará mediante la toma de notas.

- **Método de diario:** Método en el que el mismo usuario expone la información de las actividades que se quieren investigar. Su uso puede influir en la falta de datos para la investigación, actividades laborales del usuario y el entendimiento de la información facilitada,

- **Análisis competitivo:** Consiste en realizar un estudio de mercado con el fin de detectar productos similares y recopilar información que ayude a mejorar la aplicación a desarrollar.
- **Entrevistas en profundidad, dinámicas de grupo y encuestas:** Consiste en la elaboración de un guión abierto para identificar las necesidades objetivos de los usuarios.

Una vez analizadas las diferentes técnicas de investigación, se concluye poner en práctica las siguientes técnicas en el siguiente orden:

- ➢ **Análisis competitivo**: Se definirá un primer enfoque al proyecto comparando los productos similares para tener un punto de partida y descubrir posibles requisitos nuevos.
- ➢ **Entrevistas:** Se analizará en detalle la opinión del usuario su entorno laboral
- ➢ **Shadowing:** Se ha considerado aplicar esta técnica para tener una retroalimentación de los datos adquiridos en las encuestas y entrevistas y detectar posibles necesidades no detectadas en las técnicas anteriores.

2.1.1.1 Analisis competitivo

Como ya se ha indicado anteriormente, el uso de esta técnica aportará un primer enfoque al proyecto. El estudio de mercado a realizar permite obtener un contraste del proyecto a desarrollar con otros existentes e identificar nuevas oportunidades.

Como ya vimos en el capítulo *1.1 Contexto y justificación de trabajo* se identificaron aplicaciones similares enfocadas a diferentes entornos. Este análisis permitió obtener los siguientes puntos de partida y oportunidades para el proyecto:

- La interfaz gráfica tiene que ser sencilla y amigable con el usuario
- El usuario tiene que ser capaz de alquilar un producto en pocos pasos..
- La información a mostrar de los productos tienen que tener un único formato con la información más relevante para el usuario.
- El sistema de registro que utilizan algunas aplicaciones es un poco confuso y están vinculadas a su método de registro, por lo que se considera la oportunidad de añadir un registro de usuario sencillo vinculado a la cuenta de Google.
- El sistema de inicio de sesión que utilizan algunas aplicaciones da la posibilidad de realizarlo a través de diferentes campos, como son el número de usuario o email.
- El sistema de inicio de sesión que utilizan algunas aplicaciones no da la opción de ver la contraseña introducida.
- Implementar un sistema de alertas se realiza mediante los intereses del usuario y está configurado vía email.
- La información proporcionada de algunas aplicaciones se realiza a través de otras plataformas y no desde la misma aplicación, por lo que resulta difícil de usar.
- Aplicar un sistema de lectura de códigos de barras que funcione por medio de la cámara.

2.1.1.2 Shadowing y entrevistas

Los métodos de análisis como son el shadowing y las entrevistas nos permiten obtener requisitos de los usuarios de formas distintas. Uno de los puntos en común de las dos técnicas, es la parte de la entrevista, por lo tanto en este apartado se ha optado combinar los dos métodos.

Shadowing

El uso de esta técnica nos va a permitir obtener con más detalle el conocimiento de la experiencia laboral de los empleados. Es una técnica no intrusiva, pues se trata de observar en todo momento a los empleados para detectar qué necesidades tienen sin intervenir en ningún momento.

Para aplicar este método de análisis se elige un empleado y se le pone en contexto para que tenga conocimiento del objetivo a analizar. La persona responsable de realizar esta técnica tomará notas con bolígrafo y papel sobre las actitudes del usuario.

A continuación se muestran los resultados obtenidos del shadowing:

Nombre	Carles
Puesto	Ingeniero de electrónico senior
Experiencia	El empleado realiza sus tareas habituales en una zona de trabajo destinada a un proyecto y necesita un juego de herramientas para abrir una caja y una fuente de alimentación para alimentar un circuito. Para ello, se dirige al laboratorio electrónico en busca del material. Se observa que el laboratorio dispone de un cajón donde se guardan las herramientas de forma clasificada. El empleado busca allí una herramienta y la encuentra. Para tomarla prestada, el empleado empieza a registrar el alquiler en una hoja de datos. Los datos a rellenar son: descripción de la herramienta, localización inicial, localización destino, nombre, fecha de registro y fecha de devolución. Se observa que el empleado está nervioso al tener que rellenar todos los datos manualmente. El empleado tiene prisa. Posteriormente, busca la fuente de alimentación en un armario de instrumentación y no encuentra la que necesita, por lo que decide consultar al técnico de laboratorio. Este comprueba en un documento excel para comprobar si efectivamente alguien se la ha llevado. Tras unos minutos confirma que no existe ningún registro, por lo que decide consultar a los compañeros de trabajo. Pasados 10 minutos vuelve con la fuente de alimentación y el empleado vuelve a realizar la tarea de registrarla. Una vez realizado el registro de las dos herramientas que necesitaba vuelve a su puesto de trabajo.

Una vez concluida la parte de observación, se considera realizar una entrevista breve para tener un feedback del usuario en su experiencia de usuario.

Entrevistas

La realización de las entrevistas tiene como objetivo reunir los requisitos y objetivos de los empleados de una empresa que suelen necesitar herramientas de un espacio común para desarrollar una aplicación de registro de herramientas.

Las entrevistas se llevarán a cabo a dos empleados de la empresa que suelen necesitar herramientas y productos del laboratorio. Para que el desempeño de las mismas no afecte en el desarrollo de las actividades de los trabajadores se limita el número de preguntas a nueve.

Entrevistados

Con el fin de obtener un contraste en los resultados, se han elegido a dos empleados de la empresa con diferente perfil, un técnico electrónico encargado del laboratorio y un ingeniero electrónico senior.

Guión y resultados de la entrevista

La elaboración de las preguntas de la entrevista están planificadas para ejecutarse en dos partes. La primera parte se centra en el ámbito laboral con la intención de poner en contexto al entrevistado. La segunda parte se enfoca en el ámbito tecnológico para dar a conocer el enfoque de la aplicación y la información que los empleados esperan.

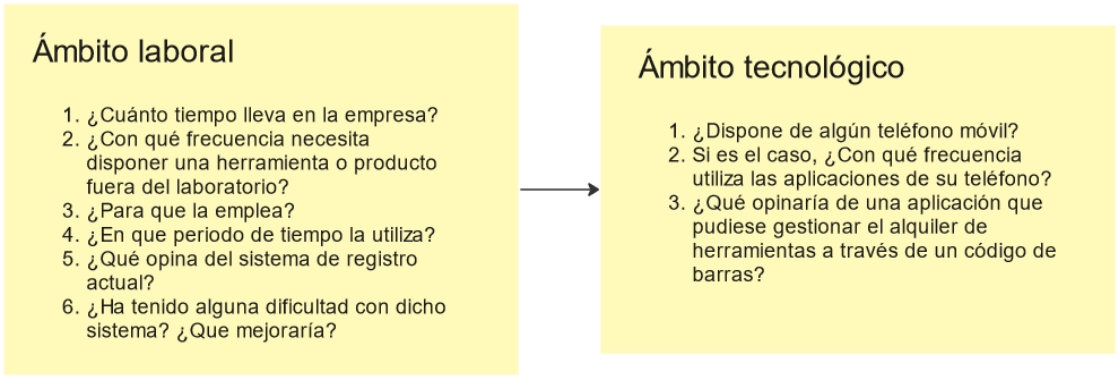
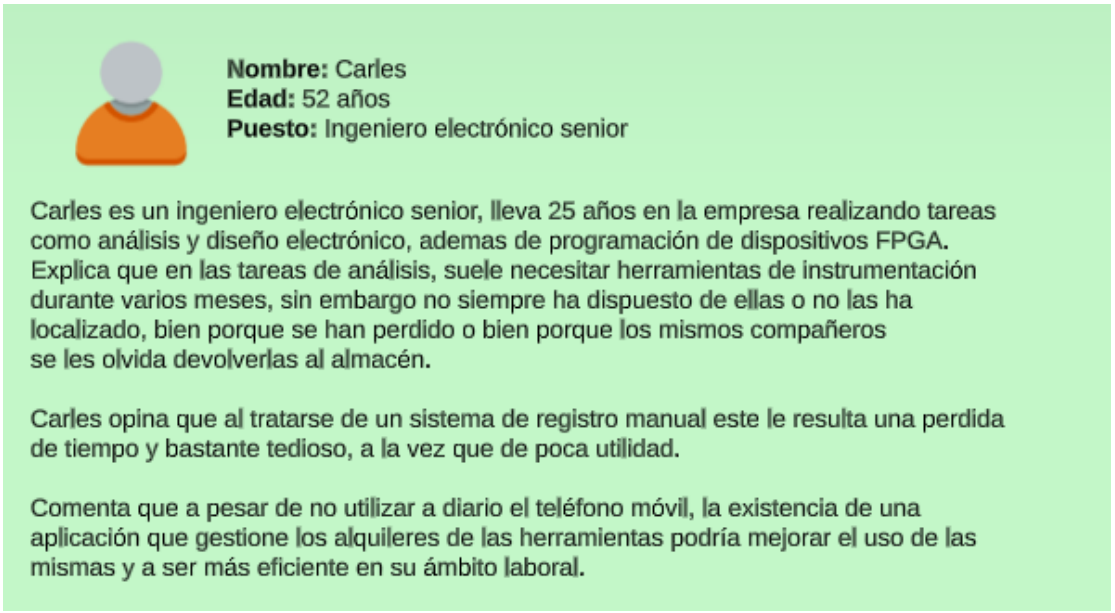


Ilustración 6 - Guión de la entrevista

A continuación, se muestra un resumen de los resultados de la entrevista:



Nombre: Carles
Edad: 52 años
Puesto: Ingeniero electrónico senior

Carles es un ingeniero electrónico senior, lleva 25 años en la empresa realizando tareas como análisis y diseño electrónico, ademas de programación de dispositivos FPGA. Explica que en las tareas de análisis, suele necesitar herramientas de instrumentación durante varios meses, sin embargo no siempre ha dispuesto de ellas o no las ha localizado, bien porque se han perdido o bien porque los mismos compañeros se les olvida devolverlas al almacén.

Carles opina que al tratarse de un sistema de registro manual este le resulta una perdida de tiempo y bastante tedioso, a la vez que de poca utilidad.

Comenta que a pesar de no utilizar a diario el teléfono móvil, la existencia de una aplicación que gestione los alquileres de las herramientas podría mejorar el uso de las mismas y a ser más eficiente en su ámbito laboral.

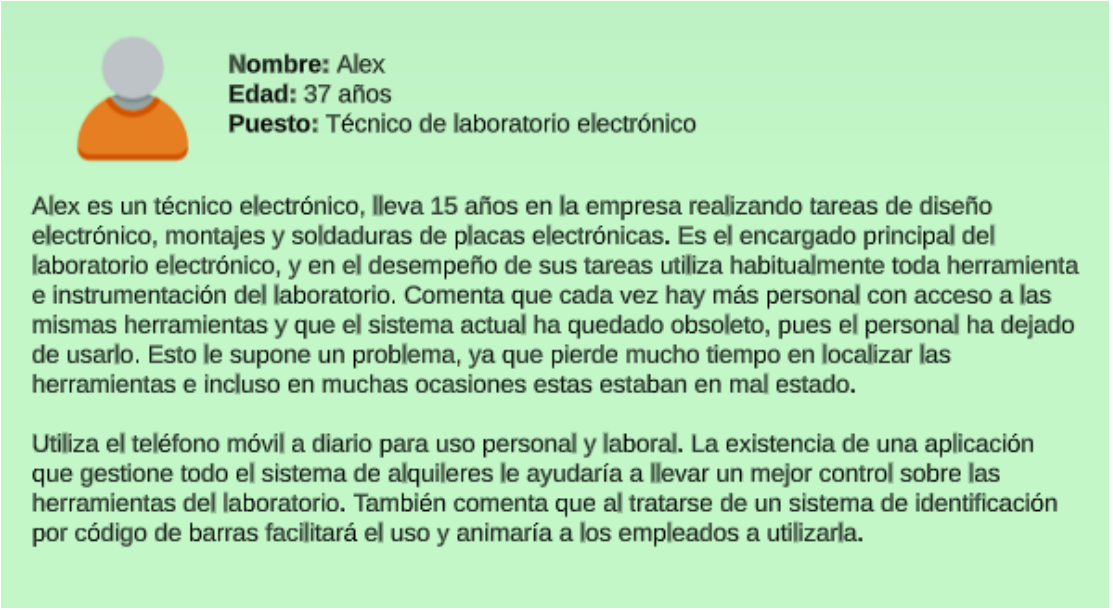


Ilustración 7 - Resumen de los resultados de la entrevista

Una vez realizada la entrevista, podemos observar que el desarrollo de la aplicación de este proyecto puede ser de gran utilidad para los empleados de esta empresa. Por otro lado, se ha identificado que el funcionamiento de la aplicación dependerá en gran parte del compromiso de uso de los usuarios y que una interfaz sencilla y fácil de utilizar puede repercutir positivamente en ello. Así pues se concluye que se debe desarrollar una aplicación atractiva, fácil de usar y debe mostrar una información clara y entendible.

2.2 Diseño conceptual

2.2.1 Definición de perfiles de usuario

Para poder analizar con más detalle las necesidades y objetivos de los usuarios se definen las fichas de los dos usuarios entrevistados que representan los dos tipos de perfiles que existen en la empresa.

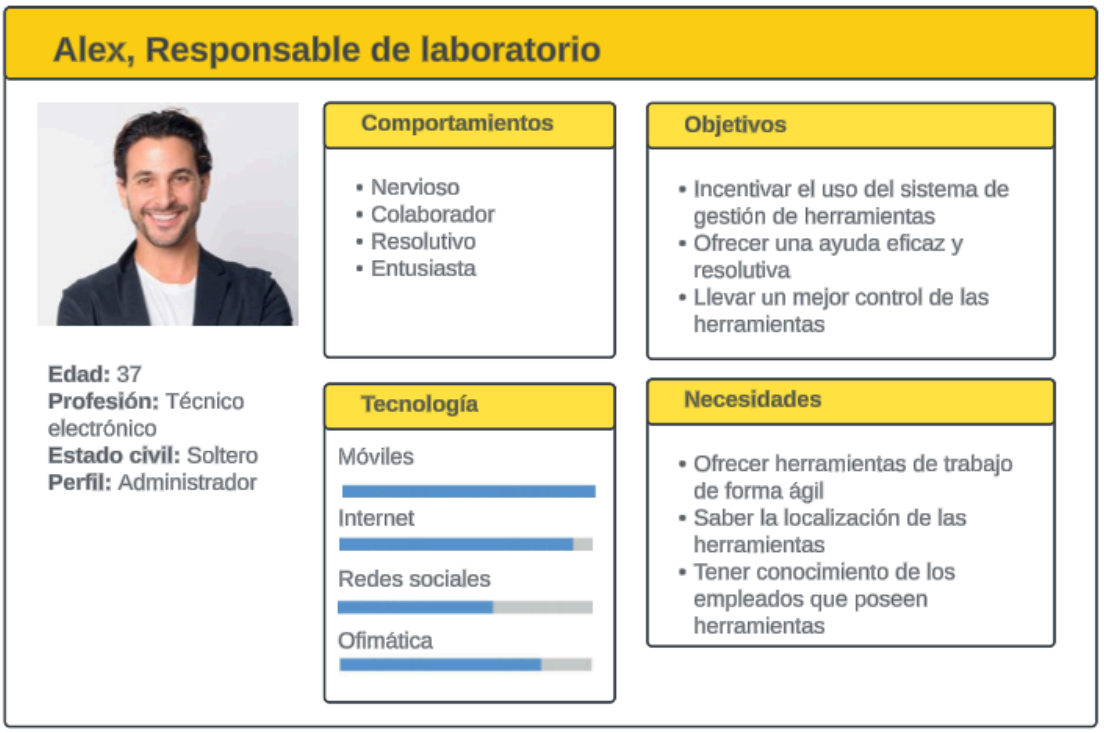


Ilustración 8 - Definición perfil Administrador

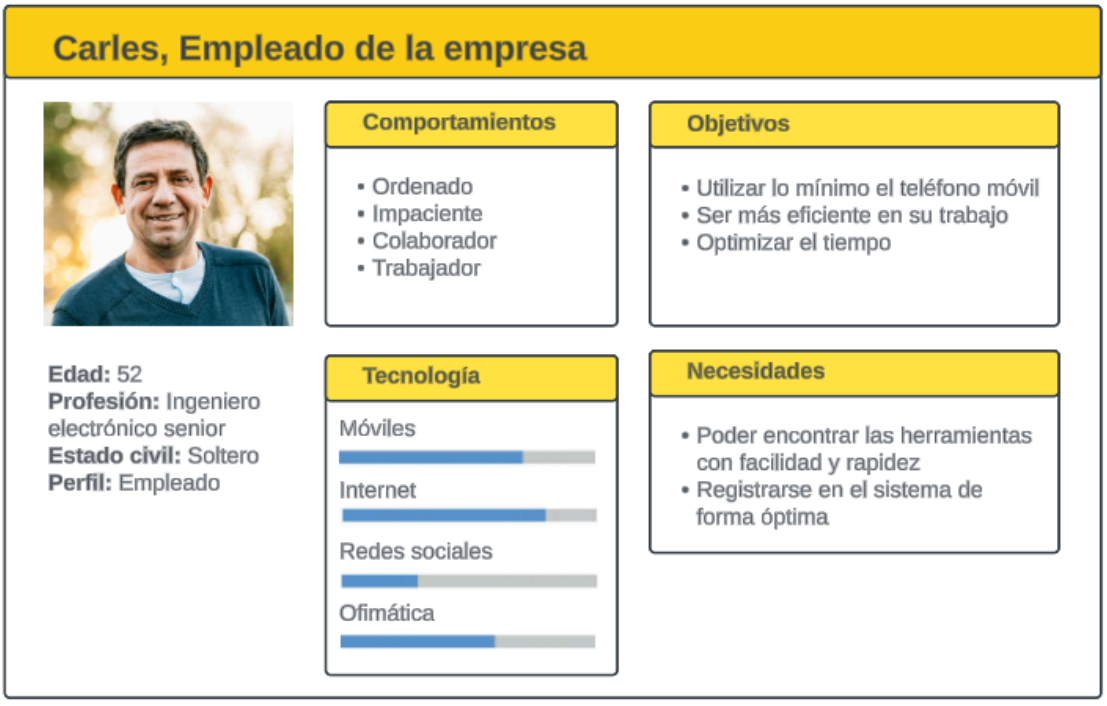
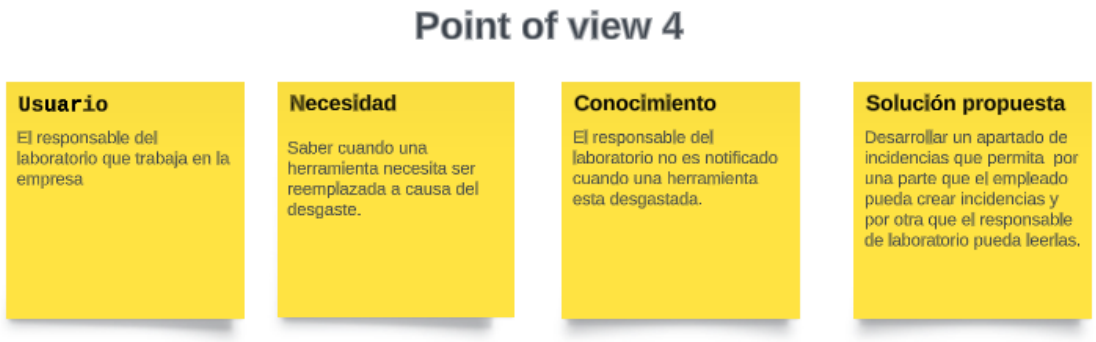


Ilustración 9 - Definición perfil Empleado

2.2.2 Problem statement

Una vez aplicados los diferentes métodos de análisis y obtenidos los requisitos se pueden identificar diferentes problemas de usuario durante la ejecución de las tareas realizadas en el apartado anterior. Para definir en detalle estos problemas se aplica una técnica llamada *point of view statement*, cuyo objetivo es realizar varias declaraciones explícitas y bien definidas enfocadas al problema que se quiere resolver.

A continuación se declaran las diferentes declaraciones que se han podido identificar :



2.2.3 Flujos de interacción

Una vez identificados los requisitos y problemas de los usuarios se representa un flujo de interacción de la aplicación a desarrollar que permitirá definir el comportamiento esperado del prototipo a realizar.

A continuación se muestra el flujo de interacción:

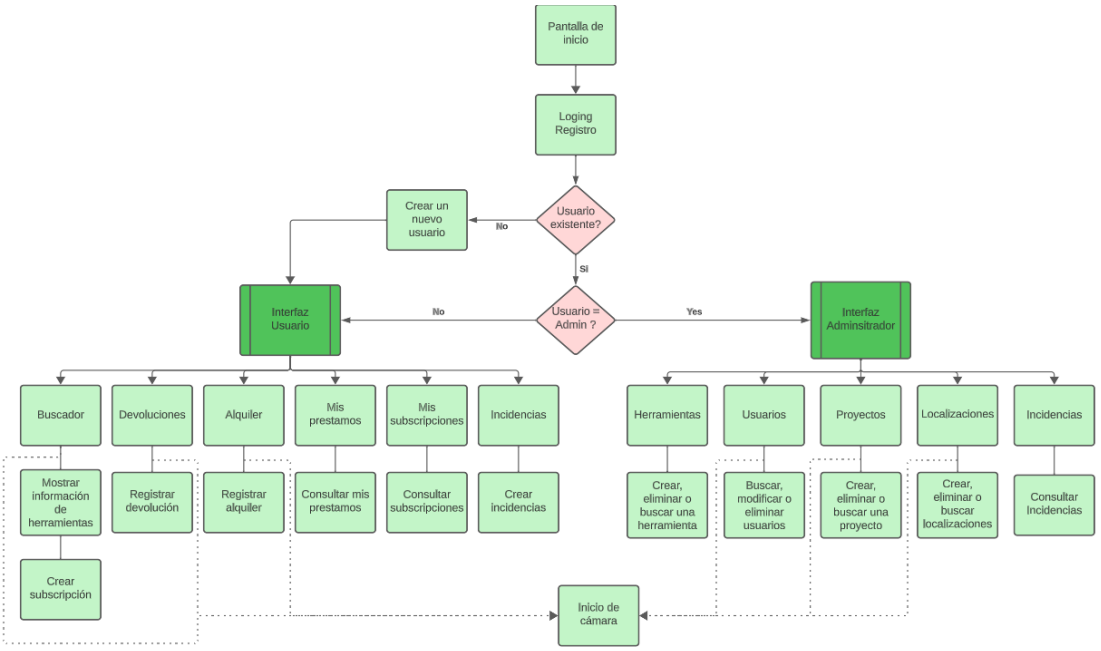


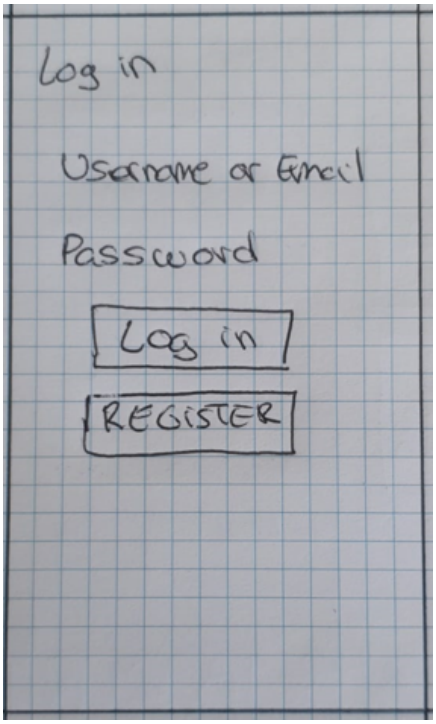
Ilustración 10 - Flujo de interacción del sistema

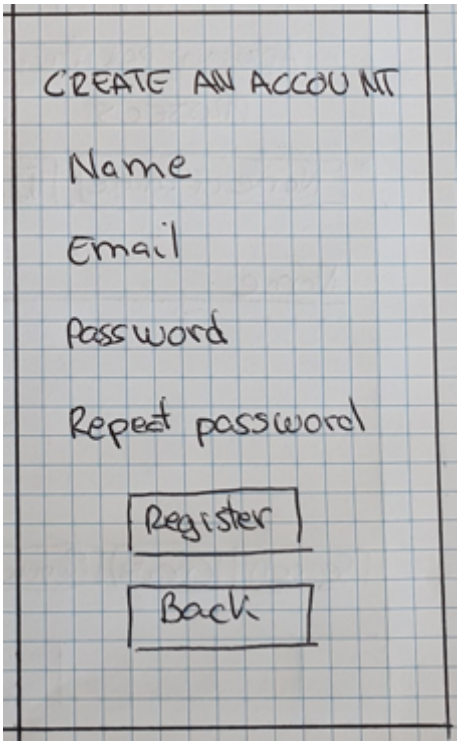
2.3 Prototipado

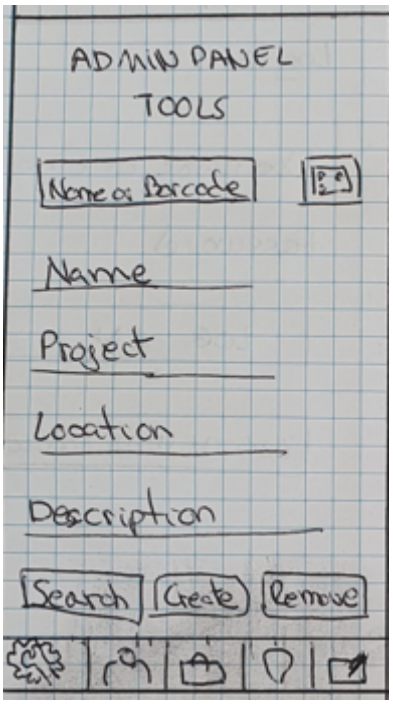
Una vez definido el comportamiento de la aplicación a desarrollar en este proyecto, se procede a realizar los primeros *sketches* para obtener una idea de la organización de las diferentes funcionalidades a implantar.

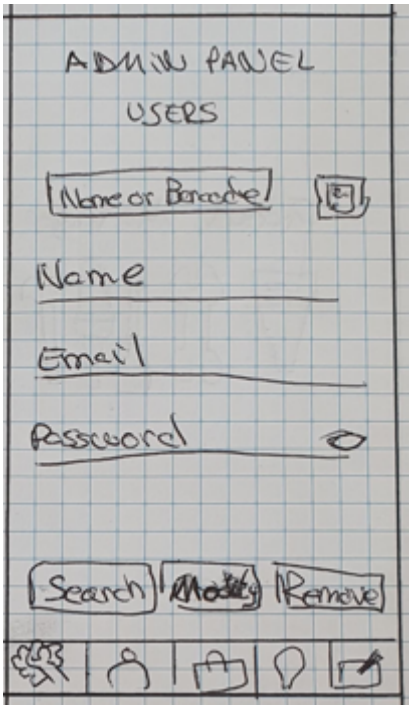
2.3.1 Sketches

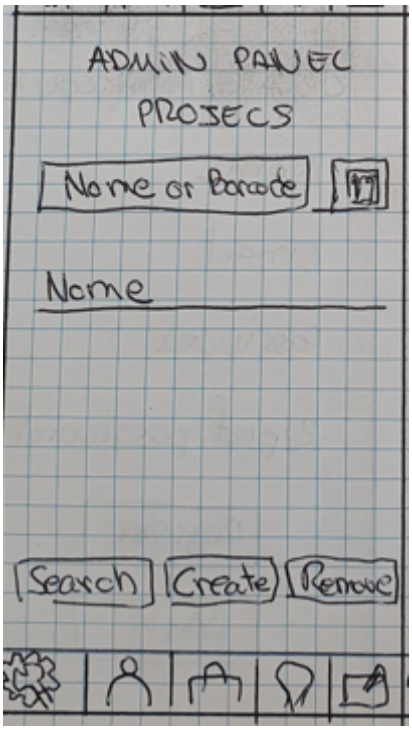
Inicio	
	La pantalla de login permite al usuario identificarse por el método convencional introduciendo el correo y contraseña o con su cuenta de Google. Esta interfaz también permite acceder a la vista de registro de usuario.

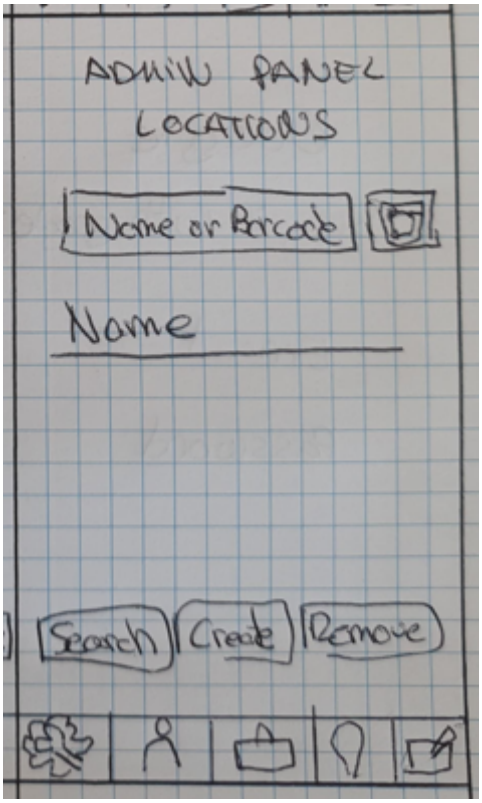
Login	
	La pantalla de login permite al usuario identificarse por el método convencional introduciendo el correo y contraseña. Esta interfaz también permite acceder a la vista de registro de usuario.

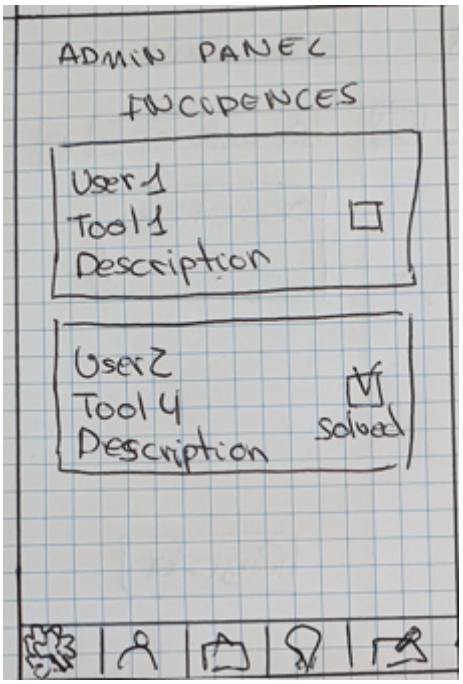
Registro de usuario	
	La pantalla de registro de usuario se basa en el método convencional de registro. El usuario podrá crearse una cuenta en el sistema introduciendo los siguientes datos: - Nombre - Correo - Contraseña - Contraseña (Repetir) Se repite el campo de contraseña para detectar errores al introducir la misma. Una vez que el usuario tenga el formulario rellenado (todos los campos son obligatorios) podrá pulsar al botón Registrar. En caso de que el usuario haya accedido por error a esta vista podrá volver atrás apretando al botón Atras.

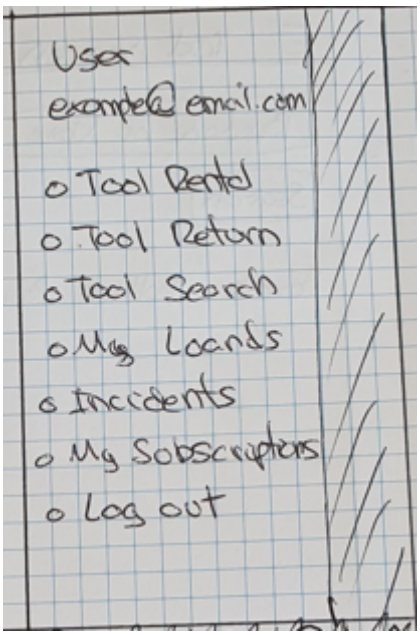
Administrador: Tools	
	En el perfil de administrador tiene varios fragmentos. El fragmento del panel de herramientas permite al administrador crear, borrar o buscar una herramienta a partir del codigo de barras de la herramienta. Si se ha rellenado todo el formulario, el administrador podrá registrar la herramienta apretando el botón Crear. Para borrar una herramienta el administrador tendrá que buscarla primero apretando el botón Buscar. Si el sistema la ha encontrado podrá eliminar la misma apretando el botón Borrar.

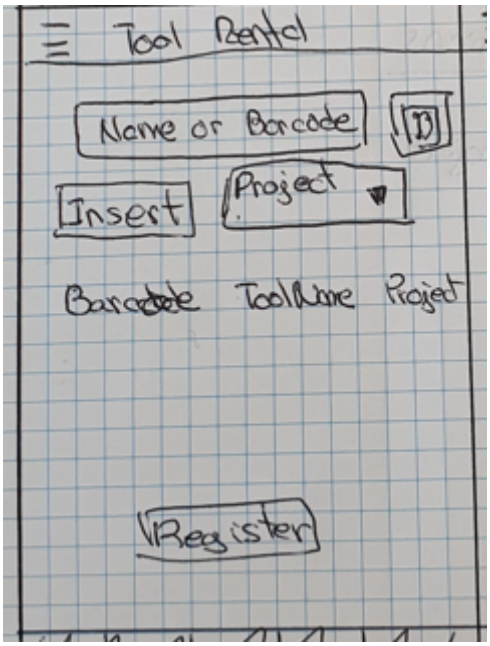
Administrador: Usuarios	
	La vista de panel de usuario permite al administrador buscar, borrar o modificar un usuario. El sistema está diseñado para buscar un usuario a partir de su código de identificación o nombre.

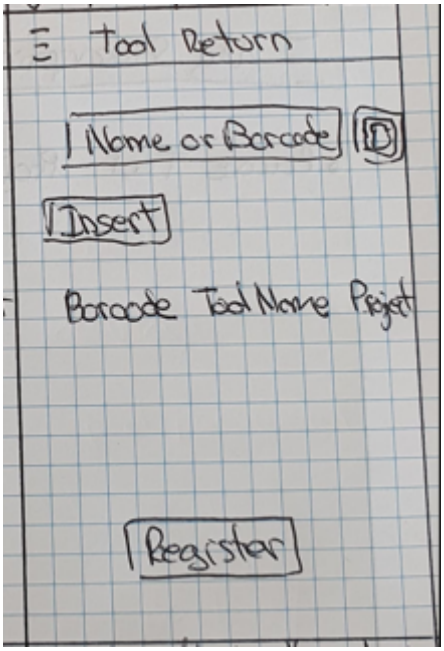
Administrador:Sección Proyecto	
	La vista de panel de proyecto permite al administrador crear, buscar y borrar un proyecto. El sistema está diseñado para buscar un proyecto a partir de su código de identificación o nombre.

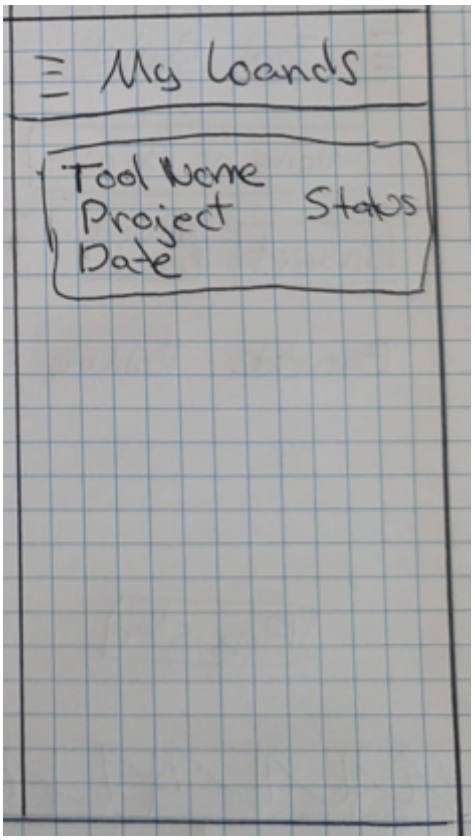
Administrador: Localizaciones	
	La vista de panel de localización permite al administrador crear, buscar y borrar una localización. El sistema está diseñado para buscar una localización a partir de su código de identificación o nombre.

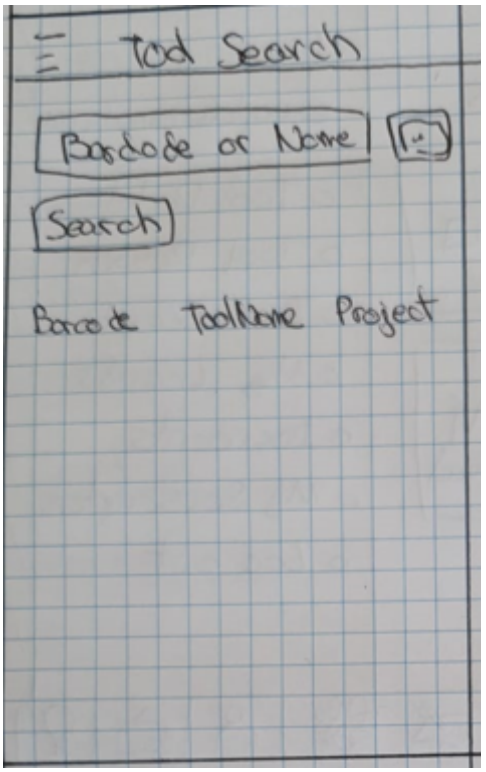
Administrador: Incidencias	
	La vista del panel de incidencias permite al administrador visualizar y gestionar todas las incidencias creadas por los usuarios.

Empleado: Interfaz	
	La interfaz de empleado permite al usuario navegar entre las diferentes vistas. A diferencia del menú administrador, se ha decidido utilizar un menú de navegación lateral al haber más de 5 vistas.

Empleado: Alquiler herramienta	
	La vista de alquiler de herramienta permite al usuario alquilar una herramienta a partir de la lectura del código de barras de la misma. El usuario podrá introducir varias herramientas a alquilar en la tabla. Una vez que el empleado tenga todas las herramientas introducidas en la tabla podrá registrar su alquiler.

Empleado: Devolución herramienta	
	La vista de devoluciones de herramientas permite al usuario registrar la devolución de una herramienta a partir de la lectura del código de barras de la misma. El usuario podrá introducir varias herramientas en la tabla. Una vez haya introducido todas podrá registrar su devolución.

Empleado: Mis préstamos	
	La vista de mis préstamos permite al usuario tener una idea de todas las herramientas que ha devuelto y que tiene pendientes por devolver.

Empleado: Buscador	
	La vista del buscador de herramientas permite al usuario buscar una herramienta a partir de su código de barras o nombre. Este menú aportará tal información como el nombre, el usuario que la tiene y su estado (disponible o no disponible). También permitirá al usuario suscribirse a una herramienta que en ese momento no está disponible para que le llegue una notificación cuando vuelva a estarlo.

Empleado: Incidencias	
	La vista de incidencias permite al usuario crear una incidencia cuando encuentre una herramienta desgastada. El usuario deberá introducir el código de barras de la herramienta y aportar una pequeña descripción. Una vez introducidos los datos podrá registrar la incidencia pulsando el botón registrar.

Empleado: Suscripciones	
	La vista de suscripciones permite al usuario visualizar las herramientas a las que está suscrito.

2.4 Evaluación del prototipo

Una vez definidos los sketches empieza la etapa de evaluación. Esta fase tiene como objetivo comprobar que el desarrollo de las interfaces visuales realizadas en la etapa de prototipado están bien diseñadas de cara al usuario. Para ello, se ha realizado un prototipo avanzado que permite tener una mejor interacción entre usuario y aplicación.

2.4.1 Prototipo de alta fidelidad

Para ello, se realizará una primera aproximación del prototipo con la herramienta Marvel App complementado de un test de evaluación a un grupo de usuarios con el fin de detectar dificultades u obstáculos en cuanto a la navegación se refiere. El método de evaluación se realiza de forma remota a partir de un cuestionario cuyo objetivo es analizar la experiencia de usuario al evaluar el primer prototipo. El prototipo de evaluación se puede encontrar en el apartado de Anexos (Marvel App: ProductManager) o a través del siguiente link:

https://marvelapp.com/prototype/102j9a1b

A continuación se muestran las interfaces visuales desarrolladas con la herramienta Marvel App:

Ilustración 11 - Pantalla de inicio de la App

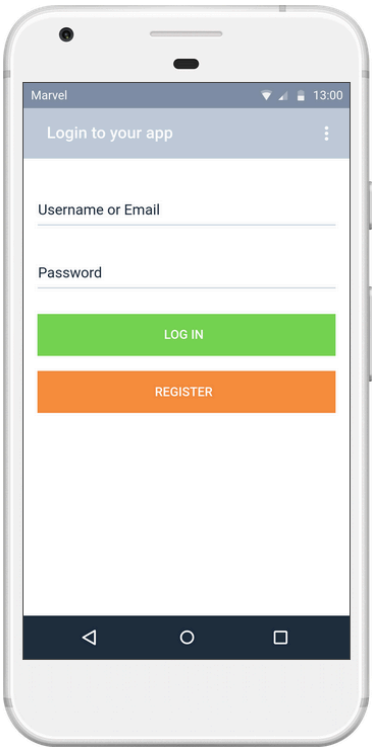


Ilustración 12 - Login

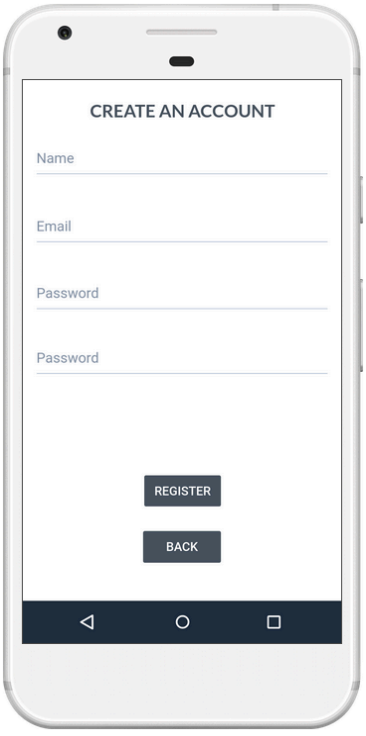


Ilustración 13 - Formulario de registro

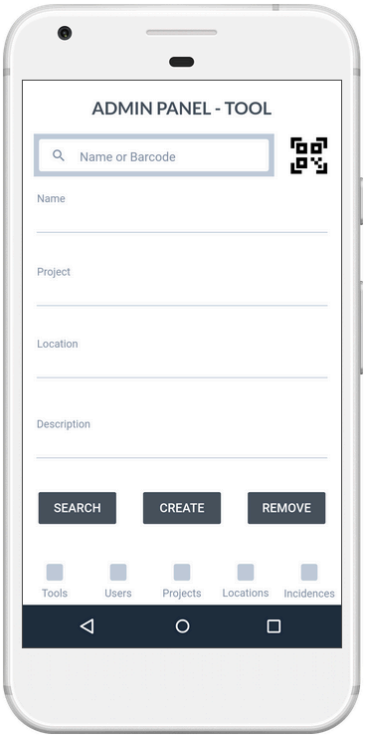


Ilustración 14 - Panel de herramientas como administrador

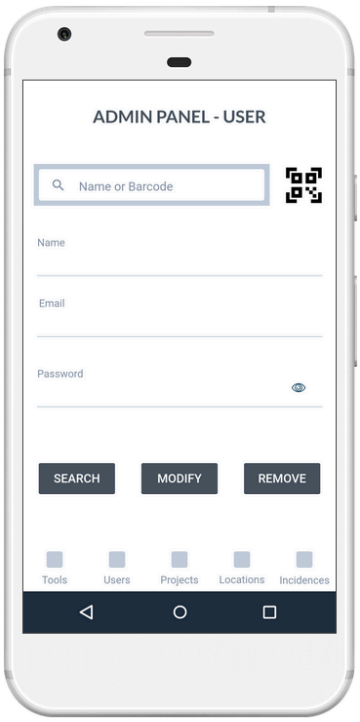


Ilustración 15 - Panel de usuario como administrador

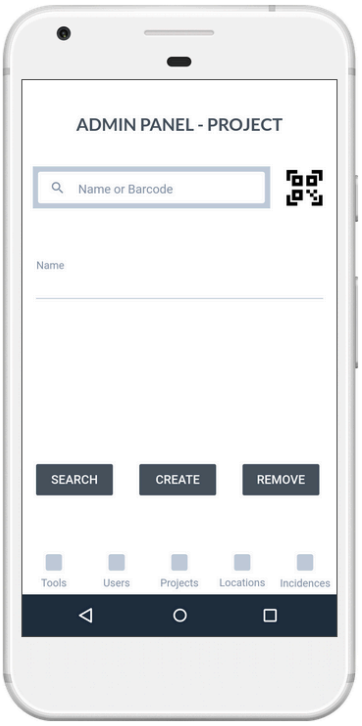


Ilustración 16 - Panel de projectos como administrador

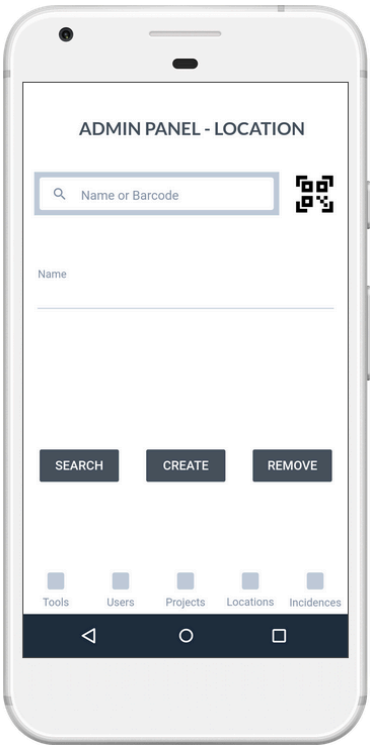


Ilustración 17 - Panel de localizaciones como administrador

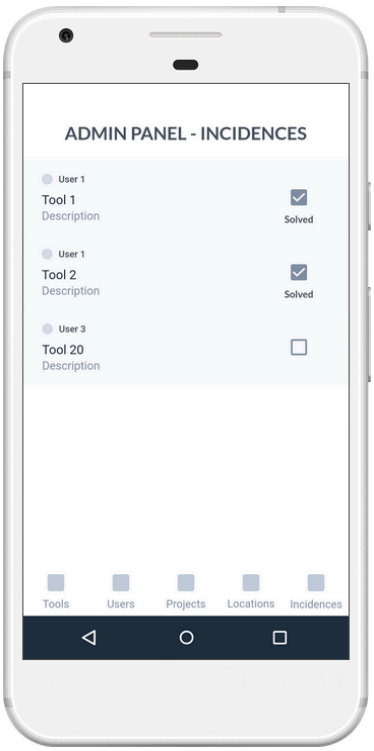


Ilustración 18 - Panel de incidencias como administrador

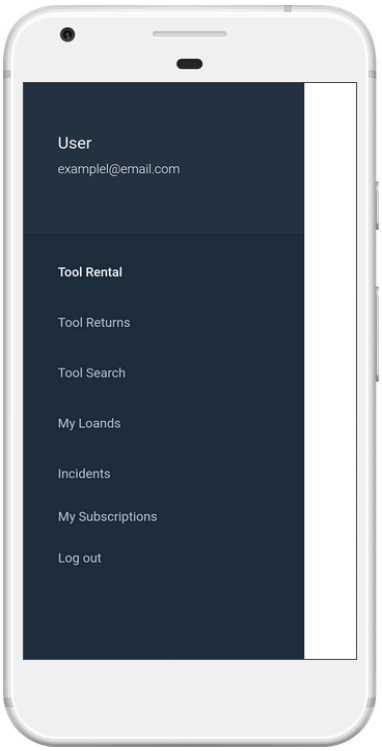


Ilustración 19 - Menú lateral de usuario

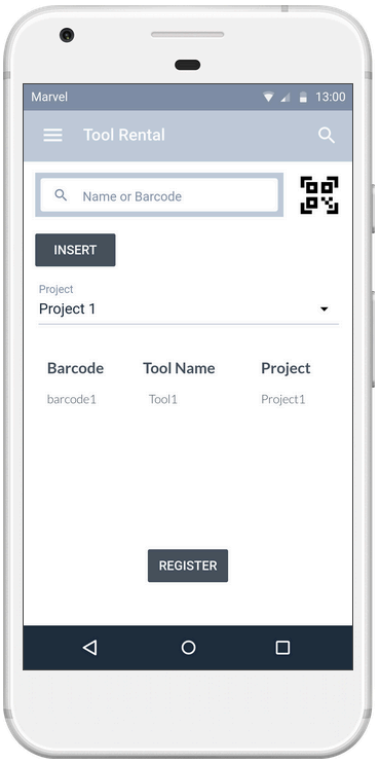


Ilustración 20 - Alquiler de herramientas como usuario

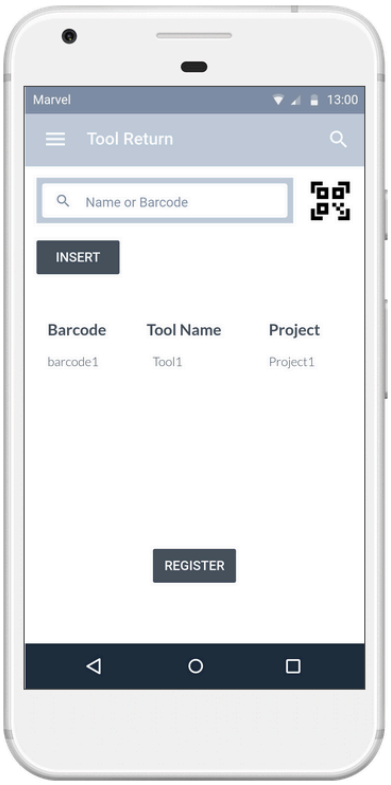


Ilustración 21 - Retorno de herramientas como usuario

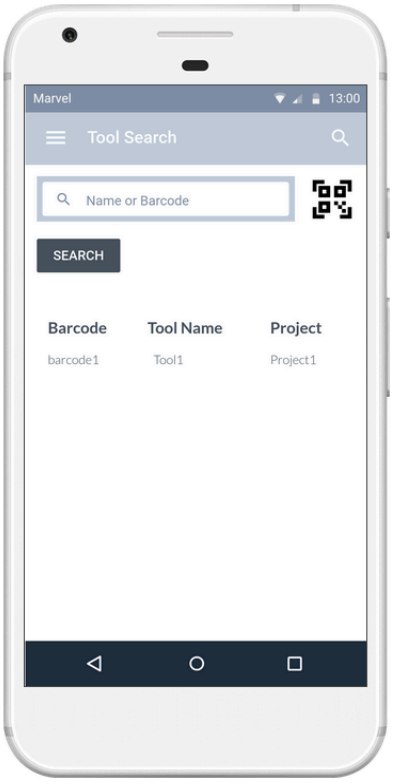


Ilustración 22- Buscador de herramientas como usuario

Ilustración 23 - Incidencias del usuario

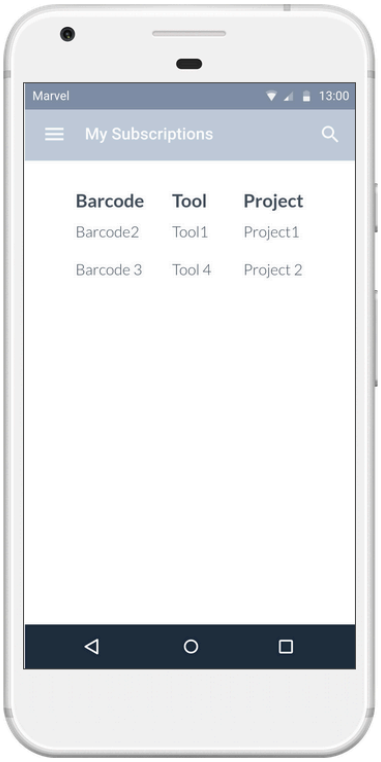


Ilustración 24 - Suscripciones del usuario

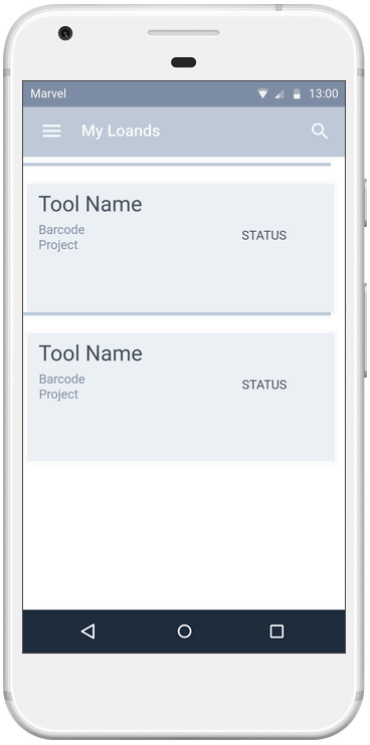


Ilustración 25 - Vista del historial de préstamos del usuario

2.4.2 Perfiles a evaluar

Tras elaborar el primer prototipo se procede a evaluar al mismo. Con el fin de obtener una mayor cantidad de información relevante y no reiterativa, se realizará un test de evaluación a cuatro usuarios. Todos ellos desempeñan el mismo perfil definido como usuario menos uno con perfil de administrador.

A continuación se definen los perfiles y sus actividades:

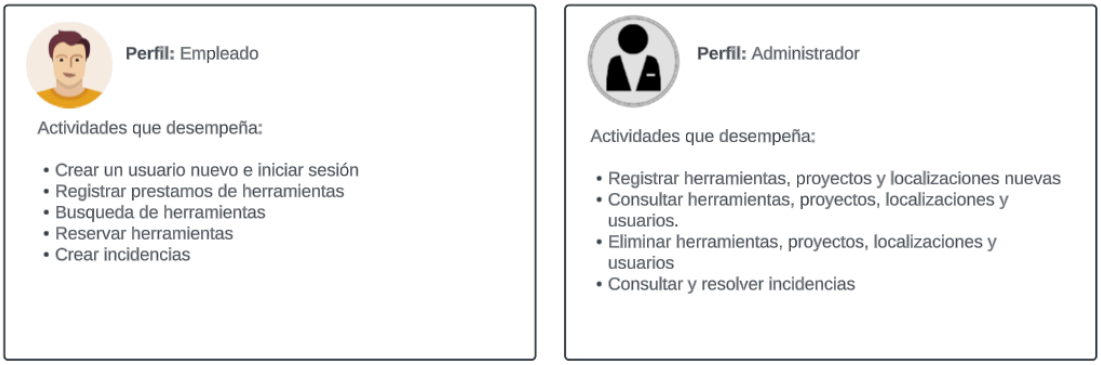


Ilustración 26 - Perfiles de los usuarios a realizar la encuesta

2.4.1 Guion

La elaboración del guión tiene como objetivo detectar la mayor cantidad de información en cuanto a la experiencia de navegación de los usuarios se refiere. Para ello, en primer lugar, se realiza el test por medio de un cuestionario online realizado por la herramienta Google Forms cuyas primeras preguntas están enfocadas a romper el hielo y para detectar información relevante para el proyecto. Posteriormente, aunque en este proyecto no se pone en práctica, se realiza un formulario de consentimiento para tener autorización para realizar grabaciones. Finalmente, se procede a analizar las respuestas para detectar posibles problemas encontrados durante la navegación. Si se desea consultar las respuestas de las preguntas se pueden encontrar en el apartado de Anexos o en el siguiente link: https://forms.gle/wD8ZnpDZo2eb4ZrEA

A continuación se muestran las preguntas del formulario:

1. ¿Qué rango de edad tiene?
2. ¿Tienes experiencia con las aplicaciones móviles?
3. ¿Utilizas habitualmente herramientas del laboratorio?
4. Evalúa del 1 al 5 el diseño de la nueva aplicación.
5. ¿Te parecen útiles todas las funcionalidades?
6. ¿Qué mejoras propondrias?
7. ¿Utilizamos esta aplicación para mejorar la gestión de herramientas

2.4.2 Conclusión y cambios a realizar

A continuación se presentan los resultados de las preguntas realizadas a los usuarios:

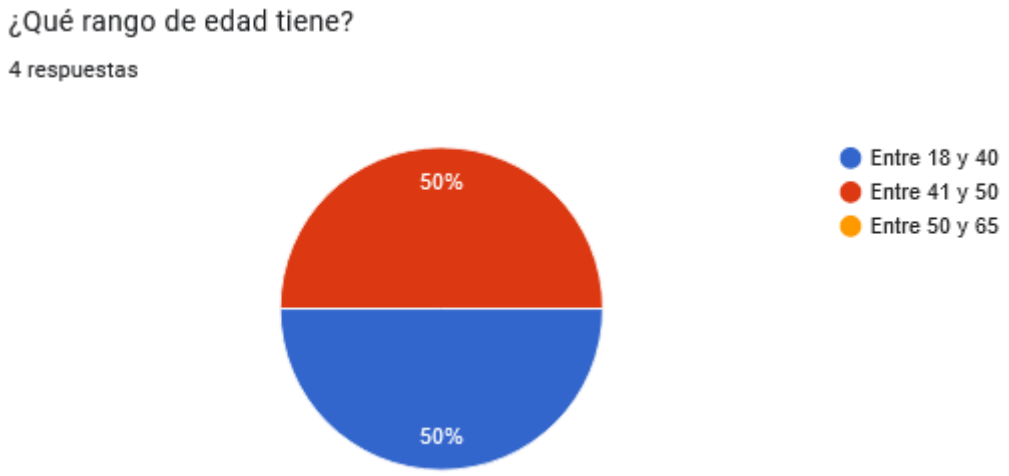


Ilustración 27 - Respuesta pregunta 1 - Rango de edad

Ilustración 28 - Respuesta pregunta 2 - Experiencia con aplicaciones

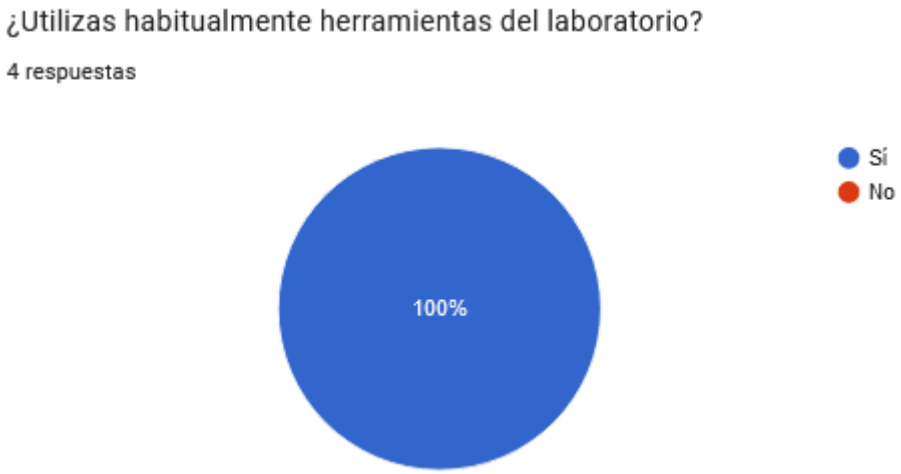


Ilustración 29 - Respuesta pregunta 3 - Uso de herramientas del laboratorio

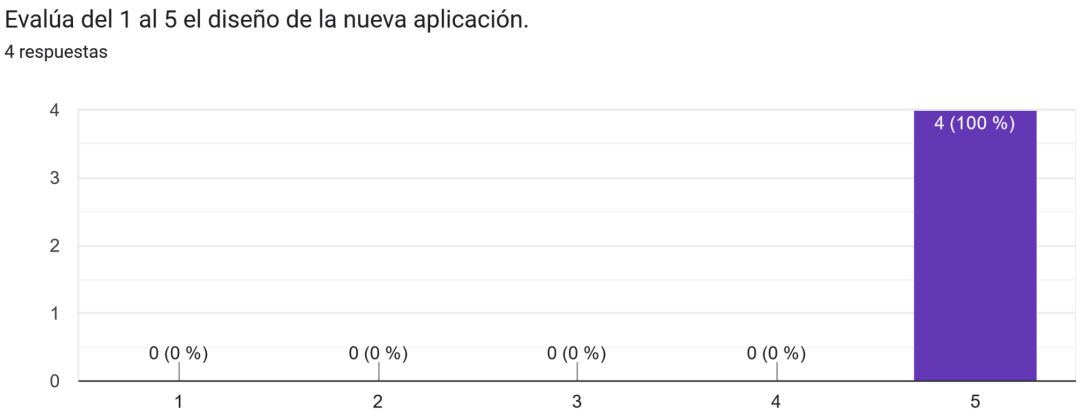


Ilustración 30 - Respuesta pregunta 4 - Evaluación diseño de la aplicación

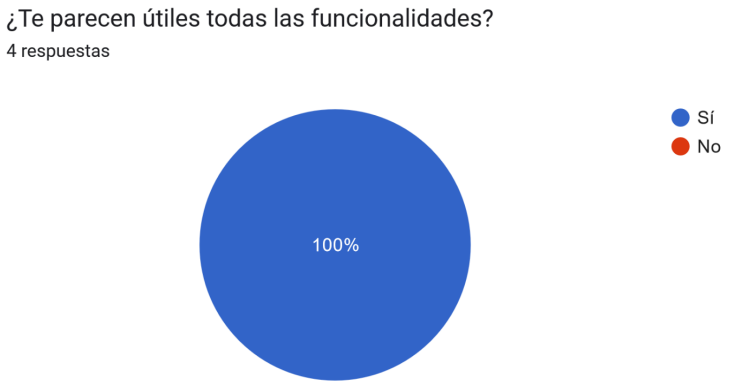


Ilustración 31- Respuesta pregunta 5- Mejoras propuestas

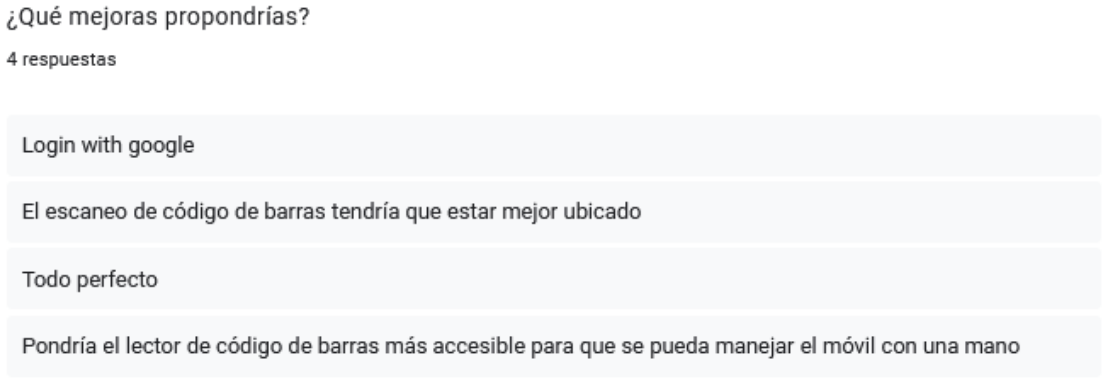


Ilustración 32 - Respuesta pregunta 6 - Uso de la nueva aplicación

Tras analizar los datos obtenidos en el cuestionario se detectan algunos problemas de usabilidad que repercuten negativamente en la navegación.

Por ello, se proponen las siguientes mejoras:

- Incorporación de sistema de registro a través de Google
- Reemplazar el botón de escanear códigos de barras por un floating button ubicado en el menú inferior para mejorar la accesibilidad de la herramienta.
- Reemplazar el menú lateral de usuario por un menú inferior para mejorar el acceso a las secciones.
- Reorganización de las opciones del menú

A continuación se muestran los cambios aplicados al prototipo:

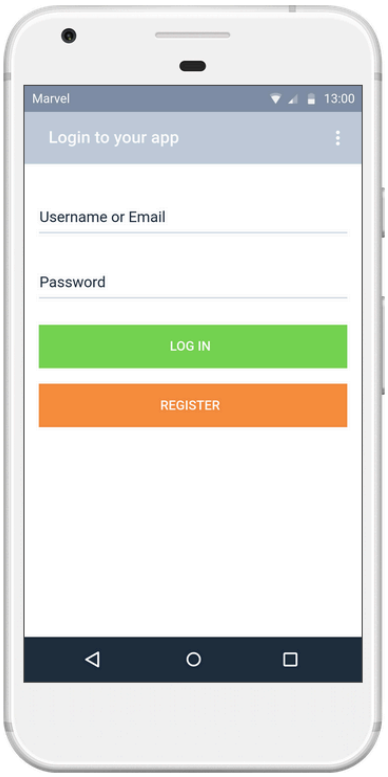


Ilustración 33 - Login sin modificar

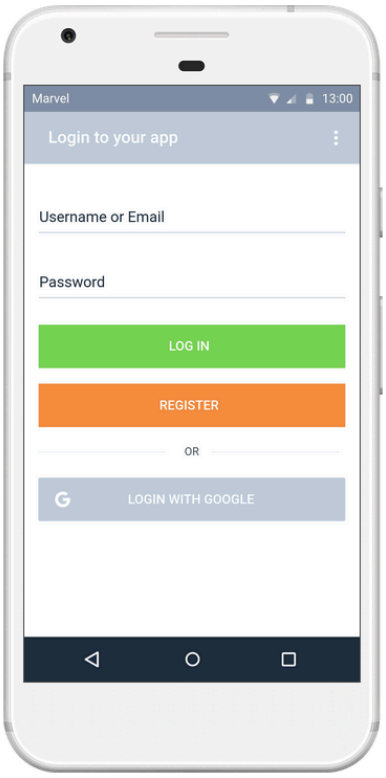


Ilustración 34 - Login modificado

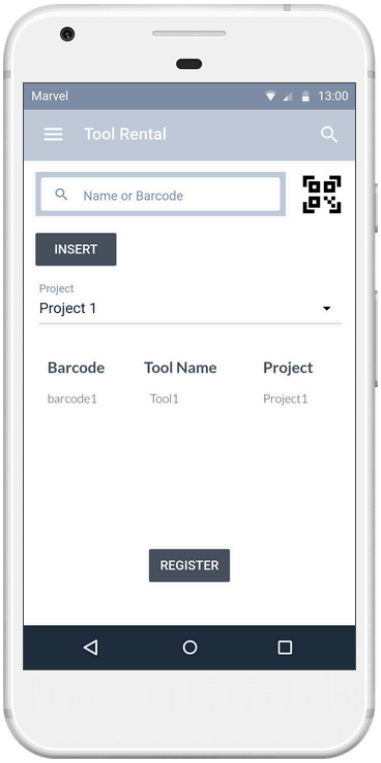


Ilustración 35 - Alquiler de herramientas como usuario

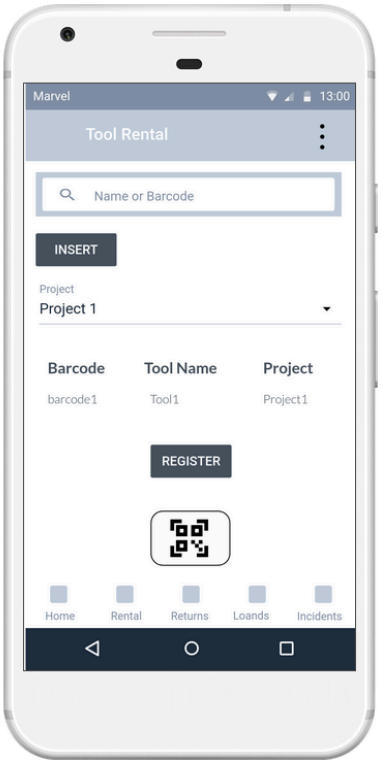


Ilustración 36 - Alquiler de herramientas modificado

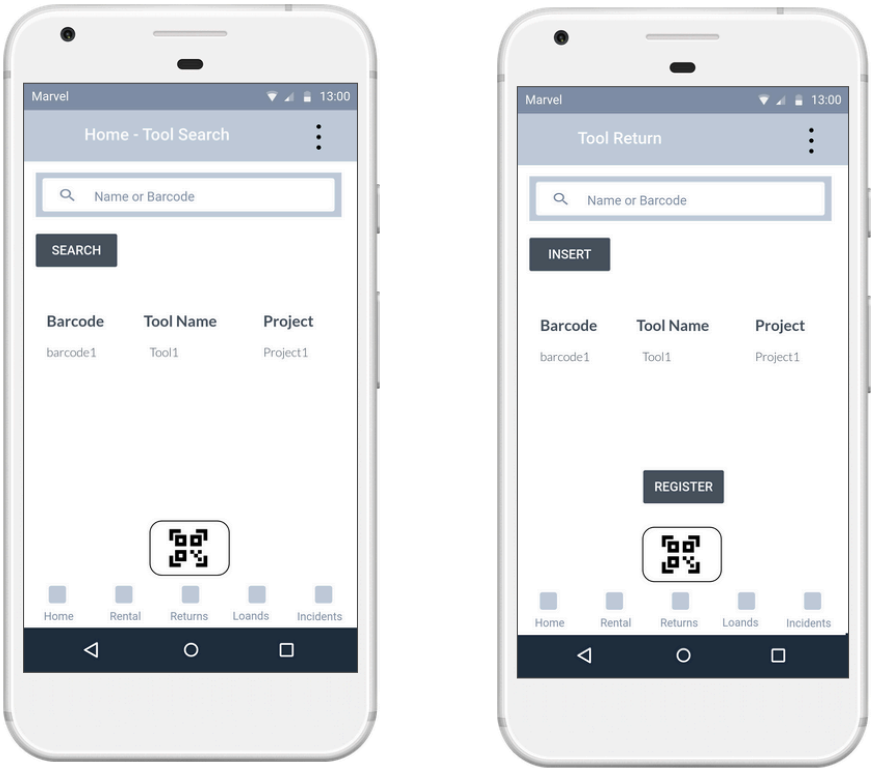


Ilustración 37 - Vista búsqueda de herramientas (home) y devoluciones modificado

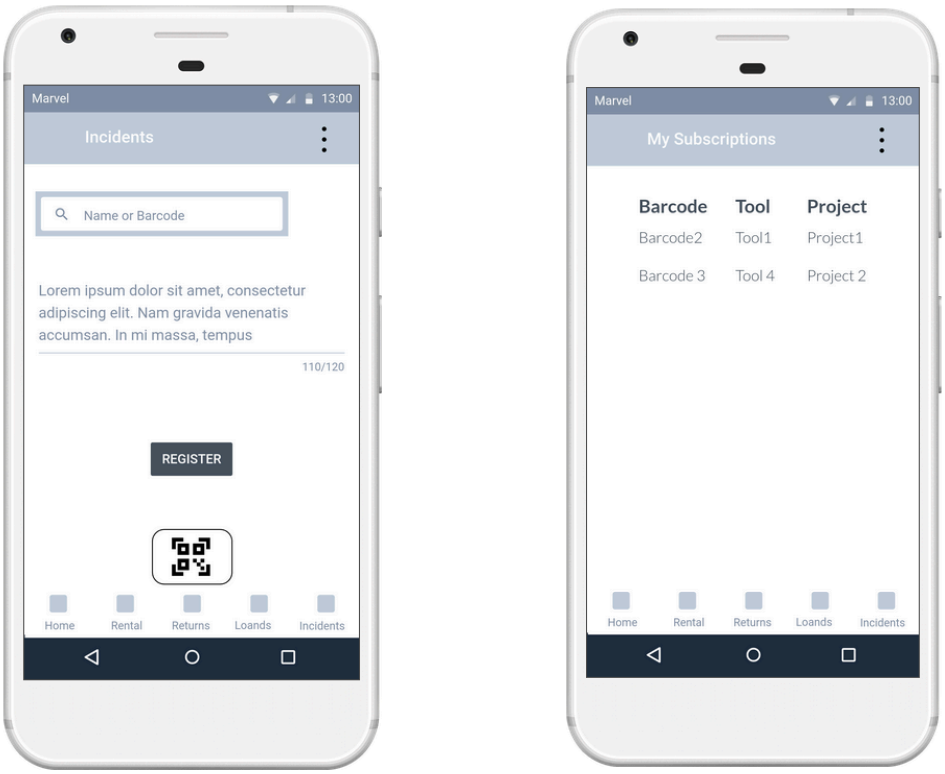


Ilustración 38 - Vista de incidencias y subscripciones modificado

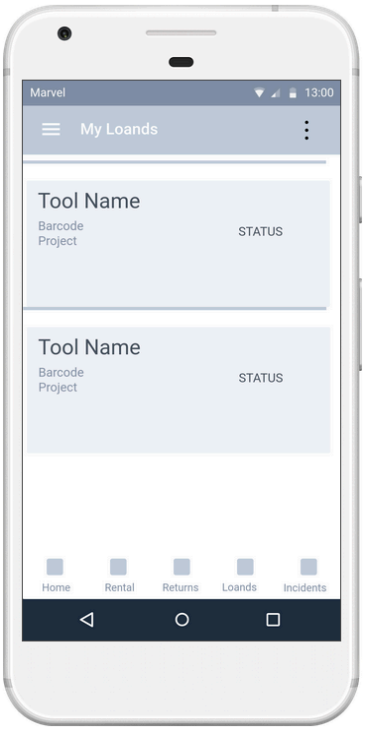


Ilustración 39 - Vista del historial de préstamos del usuario modificado

3. Diseño Técnico

3.1 Definición de los casos de uso

A continuación, se expone un diagrama de los casos de usos expuestos en el apartado *2.2.2 Casos de uso.*

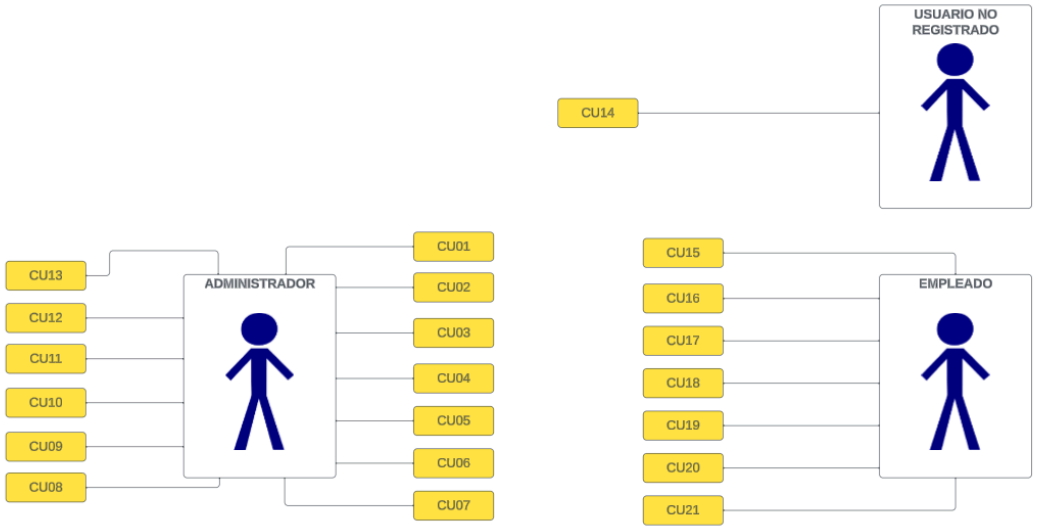


Ilustración 40 - Diagrama casos de uso

Una vez definido el diagrama se los perfiles, se analizan sus contextos de uso. Para ello, se definen diferentes casos de uso para analizar la interacción del usuario con la aplicación a desarrollar.

CU01 - Registrar herramientas	
Actor	Administrador
Ámbito	Aplicación de gestión de herramientas
Descripción	El administrador tiene que registrar una herramienta
Prioridad	Alta
Precondicion	Estar registrado a nivel de administrador
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de herramientas 3. Introducir datos 4. Pulsar en el botón crear herramienta 5. Pulsar botón registrar
Postcondicion	El sistema registrará en la base de datos la información introducida

CU02 - Borrar una herramienta	
Actor	Administrador
Ámbito	Aplicación de gestión de herramientas
Descripción	El administrador borrar una herramienta de la base de datos
Prioridad	Baja
Precondicion	Estar registrado a nivel de administrador La herramienta debe estar registrada
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de herramientas 3. Introducir código de herramienta 4. Pulsar en el botón buscar 5. Pulsar en el botón borrar herramienta
Postcondicion	El sistema eliminará la herramienta en la base de datos

CU03 - Buscar una herramienta	
Actor	Administrador
Ámbito	Aplicación de gestión de herramientas
Descripción	El administrador quiere consultar el historial de una herramienta para saber qué usuarios la han usado.
Prioridad	Media
Precondicion	Estar registrado a nivel de administrador. La herramienta debe estar registrada
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de herramientas 3. Introducir codigo de barras 4. Pulsar sobre el botón buscar herramienta.
Postcondicion	El sistema mostrará una lista de todos los usuarios que la han usado y se ordenará por fecha.

CU04 - Eliminar un usuario	
Actor	Administrador
Ámbito	Aplicación de gestión de herramientas
Descripción	El administrador quiere eliminar un usuario.
Prioridad	Baja
Precondicion	Estar registrado a nivel de administrador. El usuario debe estar registrado
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de usuarios 3. Introducir el código de barras o nombre del usuario. 4. Pulsar sobre el botón buscar 5. Pulsar botón Eliminar
Postcondicion	El sistema eliminará el usuario de la base de datos

CU05 - Modificar un usuario	
Actor	Administrador
Ámbito	Aplicación de gestión de herramientas
Descripción	El administrador quiere modificar un usuario.
Prioridad	Baja
Precondicion	Estar registrado a nivel de administrador. El usuario debe estar registrado
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de usuarios 3. Introducir el código de barras o nombre del usuario. 4. Pulsar sobre el botón buscar. 5. Modificar valores de campo 6. Pulsar sobre el botón modificar.
Postcondición	El sistema eliminará el usuario de la base de datos

CU06 - Buscar un usuario	
Actor	Administrador
Ámbito	Aplicación de gestión de herramientas
Descripción	El administrador quiere buscar un usuario.
Prioridad	Media
Precondicion	Estar registrado a nivel de administrador. El usuario debe estar registrado
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de usuarios 3. Introducir el código de barras o nombre del usuario. 4. Pulsar sobre el botón buscar.
Postcondicion	El sistema mostrará los datos de usuario en pantalla

CU07 - Crear un Proyecto	
Actor	Administrador
Ámbito	Aplicación de gestión de herramientas
Descripción	El administrador quiere crear un proyecto.
Prioridad	Alta
Precondición	Estar registrado a nivel de administrador.
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de proyectos 3. Introducir el nombre en el apartado de crear proyecto. 4. Pulsar botón Crear
Postcondición	El sistema añadirá el proyecto de la base de datos

CU08 - Eliminar un Proyecto	
Actor	Administrador
Ámbito	Aplicación de gestión de herramientas
Descripción	El administrador quiere eliminar un proyecto.
Prioridad	Baja
Precondicion	Estar registrado a nivel de administrador. El proyecto debe estar registrado
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de proyectos 3. Introducir el código de barras o nombre del proyecto. 4. Pulsar sobre el botón buscar 5. Pulsar botón Eliminar
Postcondición	El sistema eliminará el proyecto de la base de datos

CU09 - Registrar una localización	
Actor	Administrador
Ámbito	Aplicación de gestión de herramientas
Descripción	El administrador quiere crear una localización.
Prioridad	Baja
Precondicion	Estar registrado a nivel de administrador.
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de localizaciones 3. Introducir el nombre de la localización. 4. Pulsar sobre el botón añadir
Postcondición	El sistema creara una localización de la base de datos

CU10 -Busca una localización	
Actor	Administrador
Ámbito	Aplicación de gestión de herramientas
Descripción	El administrador quiere crear una localización.
Prioridad	Baja
Precondicion	Estar registrado a nivel de administrador.
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de localizaciones 3. Introducir el nombre de la localización. 4. Pulsar sobre el botón buscar
Postcondición	El sistema mostrará la localización introducida

CU11-Eliminar una localización	
Actor	Administrador
Ámbito	Aplicación de gestión de herramientas
Descripción	El administrador quiere borrar una localización.
Prioridad	Baja
Precondicion	Estar registrado a nivel de administrador.
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de localizaciones 3. Introducir el nombre de la localización. 4. Pulsar sobre el botón buscar 5. Pulsar sobre el botón eliminar
Postcondición	El sistema eliminará la localización introducida

CU12 - Consultar incidencias	
Actor	Administrador
Ámbito	Aplicación de gestión de herramientas
Descripción	El administrador quiere consultar el historial de usuario
Prioridad	Alta
Precondicion	Estar registrado a nivel de administrador.
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de incidencias 3. Pulsar sobre cualquier incidencia
Postcondición	El sistema mostrará una lista de todas las incidencias ordenadas por fecha.

CU13 - Resolver incidencias	
Actor	Administrador
Ámbito	Aplicación de gestión de herramientas
Descripción	El administrador quiere notificar al usuario que una incidencia a sido resuelta
Prioridad	Baja
Precondicion	Estar registrado a nivel de administrador.
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de incidencias 3. Pulsar sobre el botón refrescar incidencias 4. Seleccionar incidencia 5. Pulsar sobre el botón resuelta
Postcondición	El sistema eliminará la incidencia de la lista de incidencias pendientes

CU14 - Crear una cuenta nueva	
Actor	Usuario no registrado
Ámbito	Aplicación de gestión de herramientas
Descripción	El empleado se quiere registrar en el sistema
Prioridad	Alta
Precondicion	El usuario no debe existir.
Escenario principal de éxito	1. Pulsar sobre el botón Registrarse 2. Rellenar los datos del formulario 3. Pulsar sobre el botón registrar.
Postcondición	El sistema registrará los datos introducidos por el usuario en la base de datos

CU15 - Iniciar sesión	
Actor	Empleado
Ámbito	Aplicación de gestión de herramientas
Descripción	El empleado se quiere entrar en su cuenta del sistema
Prioridad	Alta
Precondicion	El usuario debe existir.
Escenario principal de éxito	1. Introducir email 2. Introducir contraseña 3. Pulsar sobre el boton login
Postcondición	El sistema mostrará la pantalla principal de usuario

CU16 - Alquilar una herramienta	
Actor	Empleado
Ámbito	Aplicación de gestión de herramientas
Descripción	El empleado se quiere registrar en el sistema
Prioridad	Alta
Precondicion	El usuario debe estar registrado. La herramienta debe estar registrada. La herramienta no debe estar alquilada.
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección alquilar herramienta 3. Introducir el código de barras 4. Introducir la zona de trabajo 5. Pulsar sobre el botón insertar 6. Pulsar sobre el botón registrar
Postcondició n	El sistema registrará los datos del alquiler en la base de datos

CU17- Devolver una herramienta	
Actor	Empleado
Ámbito	Aplicación de gestión de herramientas
Descripción	El empleado quiere registrar la devolución de una herramienta
Prioridad	Alta
Precondicion	El empleado debe estar registrado La herramienta debe estar registrada.
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de devolución de herramientas 3. Introducir el código de barras 4. Pulsar sobre el botón insertar 5. Pulsar sobre el botón registrar
Postcondición	El sistema mostrará un mensaje conforme se ha realizado con éxito la operación

CU18 - Buscar una herramienta	
Actor	Empleado
Ámbito	Aplicación de gestión de herramientas
Descripción	El empleado quiere buscar de una herramienta
Prioridad	Media
Precondicion	El empleado debe estar registrado La herramienta debe estar registrada.
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de buscador 3. Introducir el código de barras o nombre de la herramienta 4. Pulsar sobre el botón buscar
Postcondición	El sistema mostrará un mensaje conforme se ha realizado con éxito la operación

CU19 - Consultar el historial de prestamos	
Actor	Empleado
Ámbito	Aplicación de gestión de herramientas
Descripción	El empleado quiere saber las herramientas pendientes a devolver y el historial de las que ya a devuelto
Prioridad	Media
Precondicion	El empleado debe estar registrado El código de la herramienta debe existir
Garantias de exito	El sistema mostrará el historial del alquiler del empleado
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de mis préstamos
Postcondición	El sistema mostrará el historial del alquiler del empleado

CU20 - Crear una incidencia	
Actor	Empleado
Ámbito	Aplicación de gestión de herramientas
Descripción	El empleado quiere notificar al administrador sobre el desgaste de una herramienta
Prioridad	Baja
Precondicion	El empleado debe estar registrado El código de la herramienta debe existir
Escenario principal de éxito	1. Iniciar sesión 2. Ir la sección incidencias 3. Rellenar formulario de incidencia 4. Pulsar sobre el botón registrar.
Postcondición	El sistema informará al empleado el registro correcto de la incidencia.

CU21 - Consultar reservas	
Actor	Empleado
Ámbito	Aplicación de gestión de herramientas
Descripción	El empleado quiere saber las herramientas reservadas
Prioridad	Baja
Precondicion	El empleado debe estar registrado
Escenario principal de éxito	1. Iniciar sesión 2. Ir a la sección de Reservas
Postcondición	El sistema mostrará las reservas realizadas por el empleado

3.2 Diseño de la arquitectura

En este apartado trata de definir el comportamiento de la aplicación a desarrollar en este proyecto. Con ello, en primer lugar, se explica y se justifica el tipo de patrón de diseño que se aplica en el desarrollo del código. Posteriormente, se define el tipo de arquitectura a seguir además de exponer una adaptación que se adapta mejor al proyecto.

3.2.1 Patrón MVVM con DataBinding

Para el desarrollo de este proyecto se ha escogido el patrón de diseño MVVM (Model-View-ViewModel) con data binding. Este tipo de arquitectura se basa en el uso de tres componentes:

- Modelo: Encargado de la gestión de la lógica y representación de datos. Puede contener clases que gestionan la obtención y almacenamiento de datos, como bases de datos, servicios web, repositorios, etc.
- Vista: Encargado de la interfaz visual de la aplicación y transmisión de datos al ViewModel. Gestiona elementos como las actividades, fragmentos, diseños y objetos con los que el usuario interactúa.

- Vista-Modelo: Componente intermediario entre la Vista y el Modelo. Es el encargado de gestionar la lógica de la vista y adaptar los datos recibidos del modelo para la vista. Capta los eventos y acciones realizados por el usuario.

A diferencia de otros patrones como MVP o MVC, el patrón MVVM facilita la comunicación entre la capa de presentación y las clases gracias a su librería data binding. Esta librería permite cumplir uno de los objetivos SOLID, en concreto el de responsabilidad única. Gracias al databinding la capa ViewModel es totalmente independiente de la Vista, por lo que cualquier cambio realizado en cualquier interfaz de usuario no afectará a la capa de ViewModel. Sin embargo, el ViewModel contiene métodos y propiedades que la Vista puede utilizar a través de la asignación y uso patrones observadores. Con ello, se logra tener actualizada la vista en todo momento y evita el uso de callbacks, por lo que se obtiene un mejor control de eventos externos.

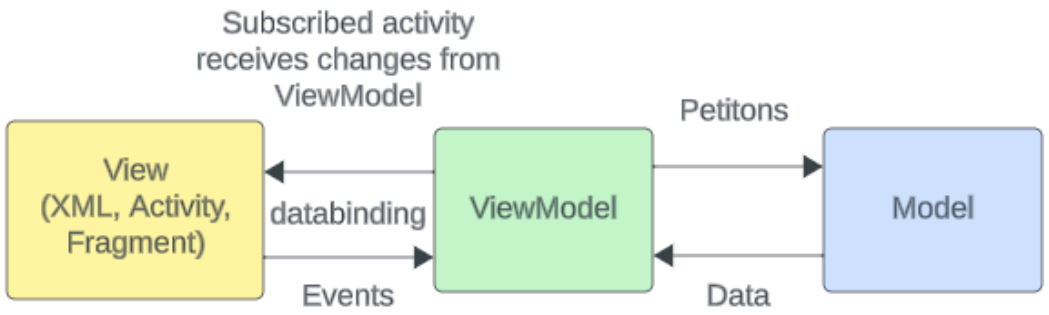


Ilustración 41 - Patrón MVVM con DataBinding

3.2.2 Arquitectura Clean

Para este proyecto, se ha creído conveniente seguir una arquitectura Clean. Este tipo de arquitectura se basa en el uso de diferentes capas que permiten tener un código mejor organizado, independiente y estable, lo que evita, entre otras cosas, tener actividades sobrecargadas.

Las ventajas que nos aporta este tipo de arquitectura son las siguientes:

- Independencia: Cada capa se encarga de realizar objetivos sin interferir en las demás.
- Estructura: El código está mejor organizado por lo que se facilita la búsqueda de funcionalidades.
- Desacoplamiento: Un cambio en una capa no afecta a las demás, lo que permite agregar o quitar diferentes tecnologías.
- Testeo: Al tener total independencia entre capas se pueden realizar test unitarios y de integración en cada una de ellas.

Tal y como se ha dicho, la arquitectura clean define una serie de capas y cada una de ellas tiene una responsabilidad. En este proyecto se ha creído conveniente realizar una variante y organizar las clases en 4 capas:

- **Dominio:** Contiene las clases que representan la lógica interna de la aplicación y la manera de representar los datos.
- **Datos:** Contiene las clases encargadas de guardar los datos y cómo acceder a ellos (bases de datos, authentication, cloud messaging)
- **Casos de Uso:** Contiene las clases que definen las operaciones que el usuario quiere realizar con la aplicación. Permite quitar responsabilidad a las actividades.
- **Capa de presentación:** Contiene las actividades, fragmentos, vistas y otros elementos con los que interactúa el usuario. Se encarga de la interfaz de usuario.

La comunicación entre capas se debe realizar de fuera hacia dentro, lo que quiere decir que por ejemplo un caso de uso no se debe comunicar con una actividad.

A continuación se muestra una imagen de la arquitectura clean aplicada a este proyecto:

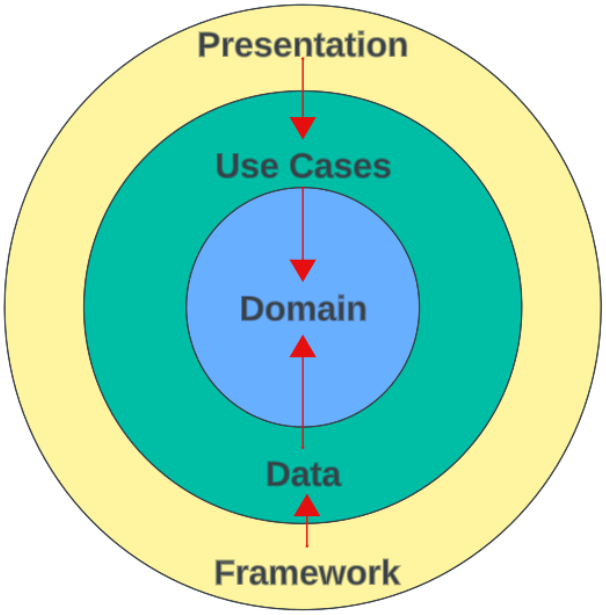


Ilustración 42 - Arquitectura Clean aplicada al proyecto

3.2.3 Estructura de la base de datos

Para la implementación de este proyecto, se utilizará Firestore como modelo de base de datos. Firestore es una base de datos NoSQL que utiliza un sistema de almacenamiento basado en el uso de documentos y colecciones. Los datos se almacenarán en los documentos identificados por su clave-valor y a su vez estos documentos se agruparán en colecciones donde se visualiza el objeto guardado.

Tras analizar las necesidades de los interesados, se cree conveniente crear un modelo de datos con 6 tipos de colecciones:

Users collection

Contiene un documento por cada usuario registrado. La clave que lo identificará será su dirección de correo electrónico.

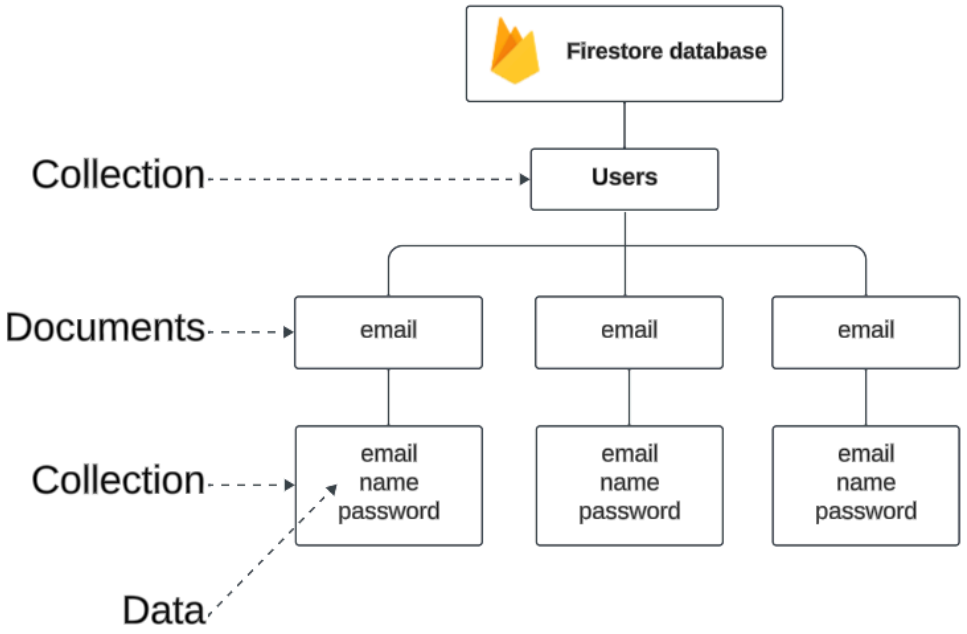


Ilustración 43 - Arquitectura Firestore Users

Projects collection

Contiene un documento por cada proyecto registrado. La clave que lo identifica será su nombre.

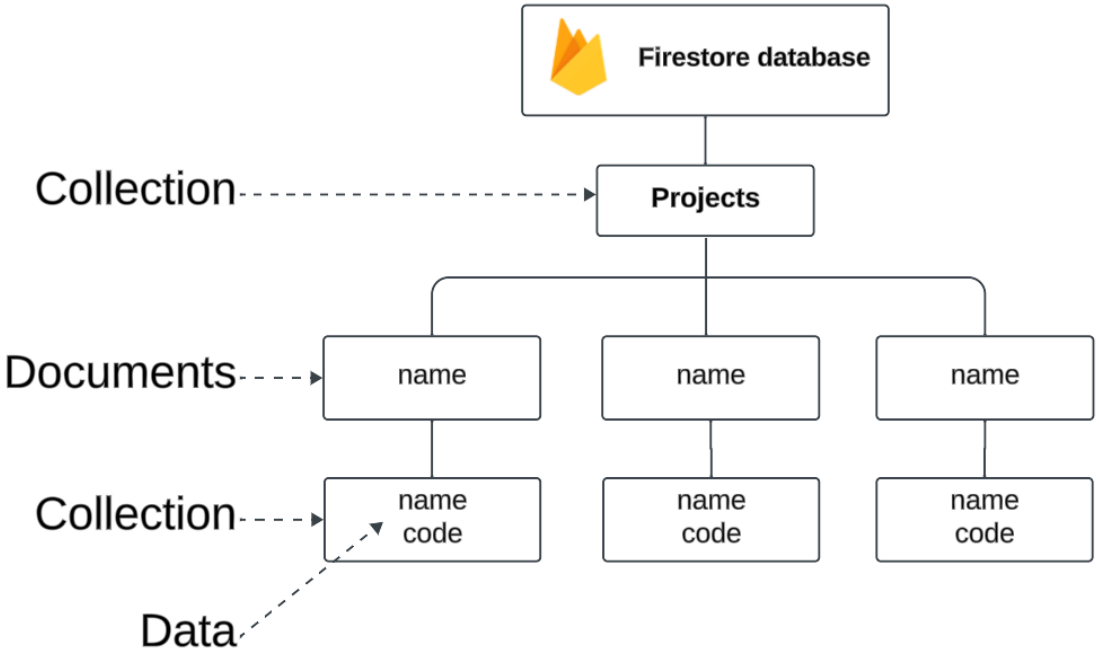


Ilustración 44- Arquitectura Firestore Projects

Locations collection

Contiene un documento por cada localización registrada. La clave que lo identifica es su nombre

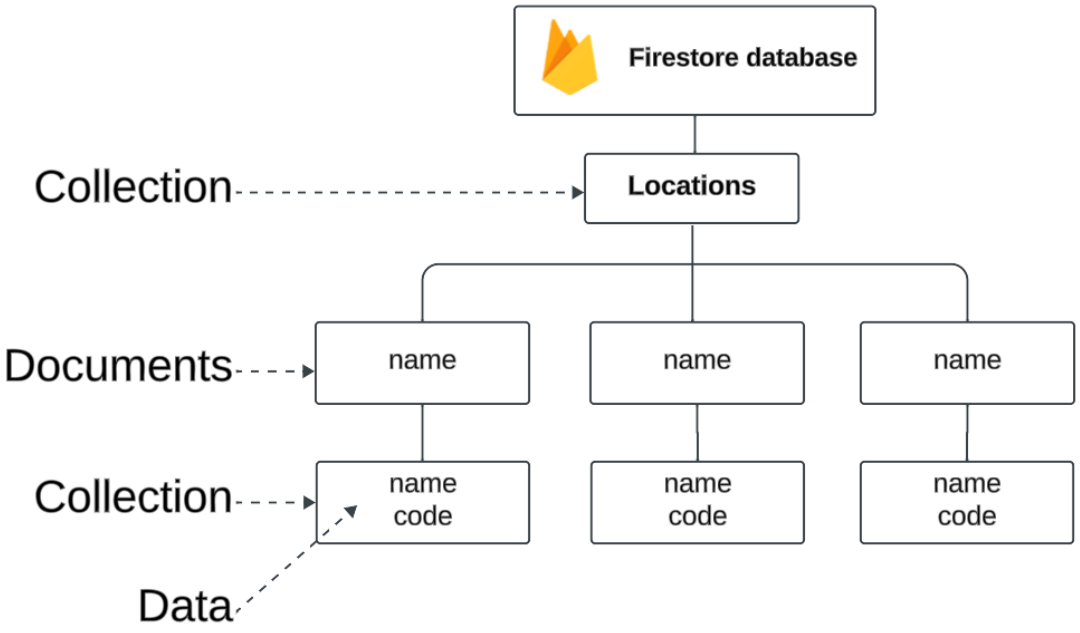


Ilustración 45- Arquitectura Firestore Locations

Tools collection:

Contiene un documento por cada herramienta registrada. La clave que lo identifica es el código de barras generado para la herramienta.

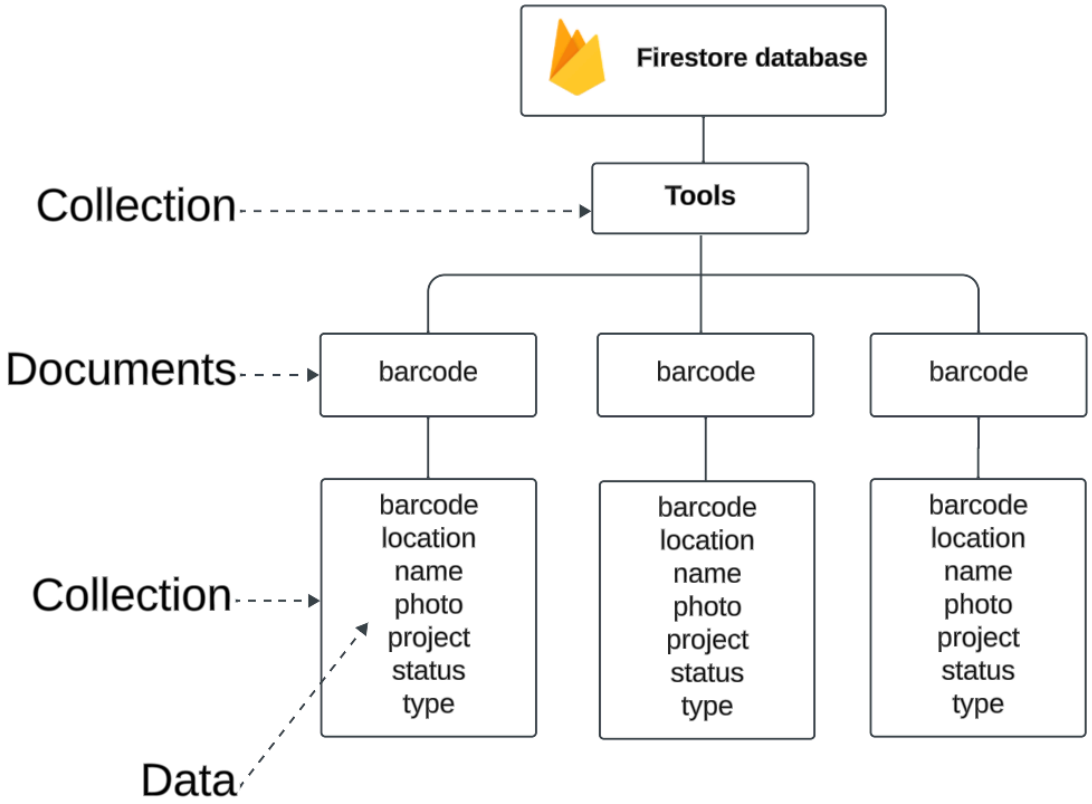


Ilustración 46- Arquitectura Firestore Tools

Rentals collection

Contiene un documento por cada usuario que haya alquilado una herramienta. La clave que lo identificará será su correo electrónico. Cada documento contendrá una colección nombrada “records” y esta contiene un documento por cada herramienta alquilada identificada por su clave de código de barras.

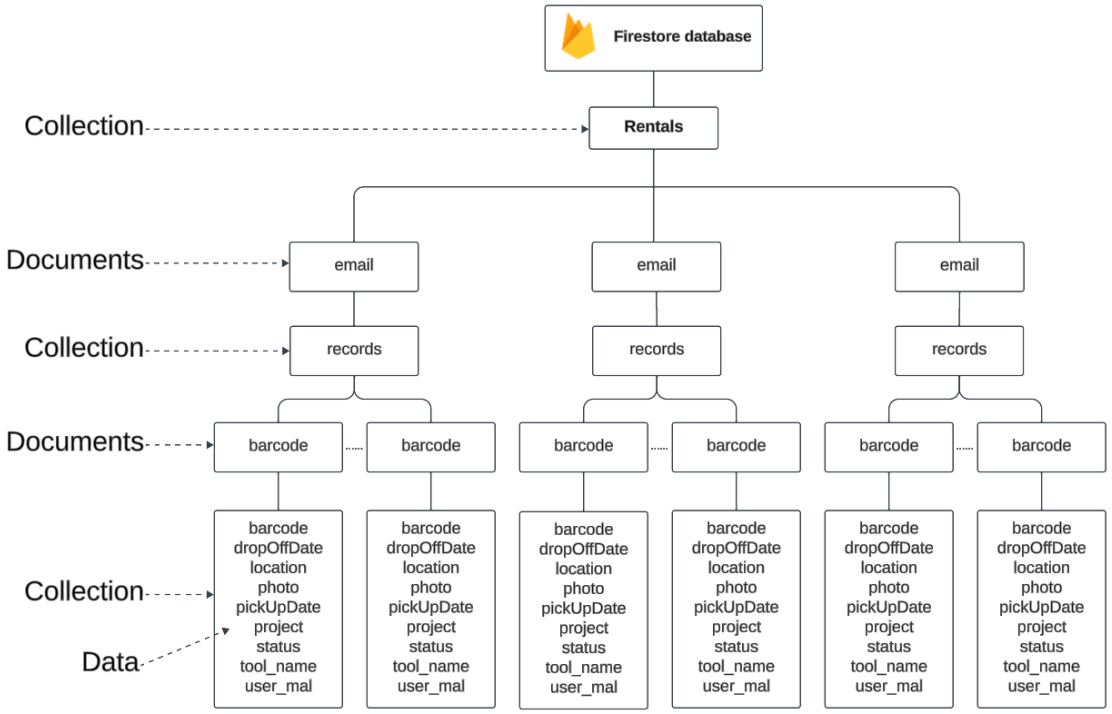


Ilustración 47- Arquitectura Firestore Rentals

Incidences collection

Contiene un documento por cada incidencia registrada. La clave que lo identifica es el id del documento.

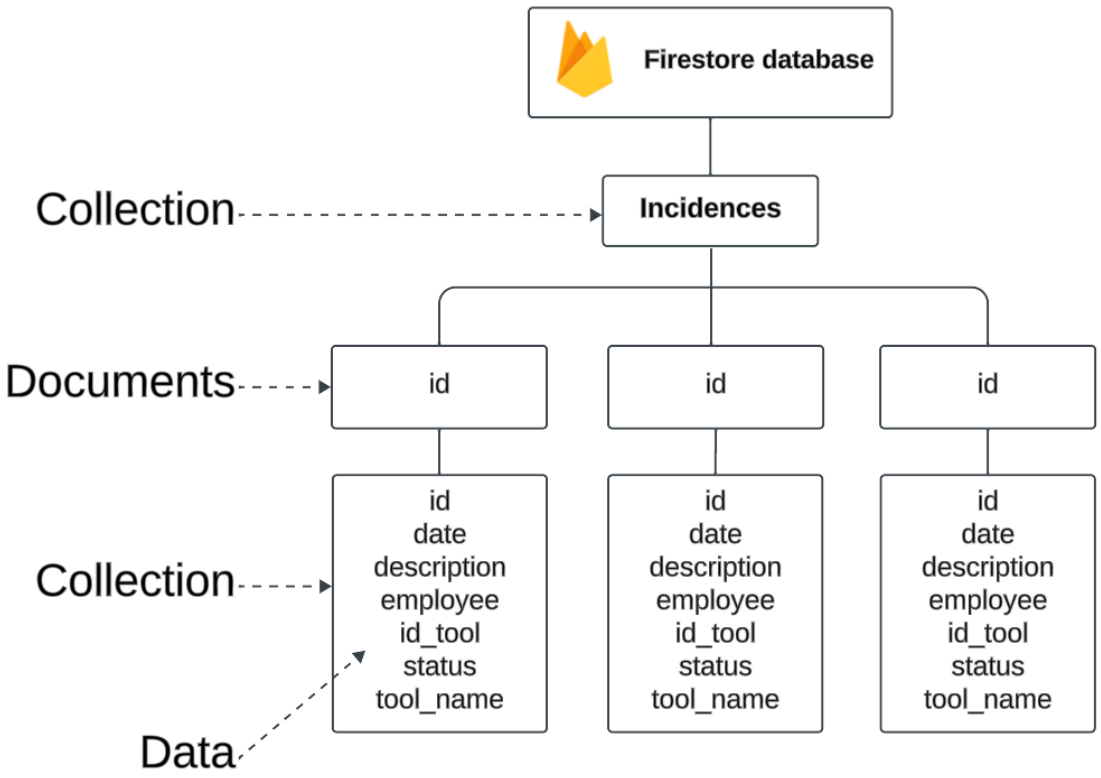


Ilustración 48- Arquitectura Firestore Incidences

3.2.4 Diseño del código de barras

Como hemos visto, cada herramienta estará identificada por un código de barras. Esto se ha decidido así para facilitar la identificación y búsqueda de herramientas mediante la lectura del código de herramienta. Por lo tanto, a continuación se define la estructura del código de barras:

ID Type	ID Project		ID Location		ID de herramienta					
X	X	X	X	X	X	X	X	X	X	X

4. Desarrollo

Tras definir analizar y obtener los requisitos del sistema por parte de los dos tipos de usuarios y realizar un el primer prototipo de proyecto se dispone a empezar la implementación de la aplicación Android. Para ello, antes de empezar a desarrollar código, se deben establecer las normas de diseño que implica la arquitectura definida en el apartado “Diseño de arquitectura”.

El uso del patrón *MVVM* y la arquitectura clean aportará al proyecto la ventaja de tener desglosadas el conjunto de responsabilidades en diferentes clases y así conseguir una baja cohesión y alto acoplamiento. Además facilitará el mantenimiento y la realización de pruebas al cumplir con los principios SOLID, como son el principio de responsabilidad única, abierto-cerrado y segregación de interfaces entre otros.
Por otro lado, al trabajar con herramientas externas como *Firebase* y por lo tanto realizar operaciones de lectura y escritura, el código está enfocado a trabajar con tareas asíncronas con el fin de ofrecer un buen rendimiento en cuanto al uso de la aplicación se refiere.

4.1 Tecnologias empleadas

4.1.1 Entorno y lenguaje de programación

La aplicación se desarrollará en el entorno de programación oficial de Android: *Android Studio*. Al tratarse de el entorno oficial de desarrollo de Android se tendrán a disposición todas las herramientas y funcionalidades propias de Android y por lo tanto facilita el desarrollo de la aplicación. La versión utilizada es Flamingo 2022.2, que incluye las actualizaciones de *IntelliJ IDEA* y aporta mejoras que facilita el desarrollo de apps.

En cuanto a la versión *SDK* utilizada, se ha creído conveniente utilizar como versión mínima la versión 24 (Android 7.0 nougat) puesto que el 96.3% de los dispositivos admiten esta versión.

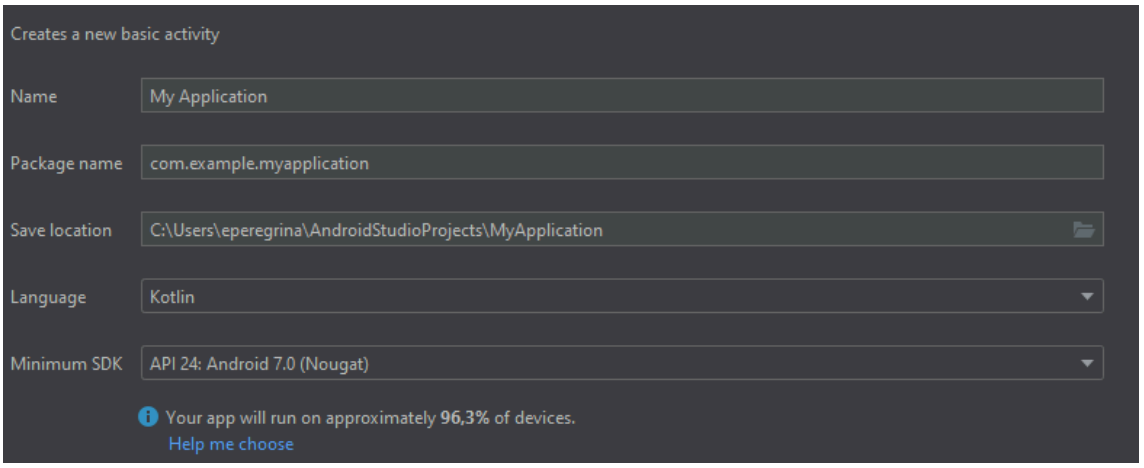


Ilustración 49- Crear nueva proyecto con versión SDK recomendada

El proyecto será desarrollado en el lenguaje de programación multiplataforma oficial de Android, ***Kotlin***, puesto que es el lenguaje predefinido para el desarrollo de aplicaciones móviles Android, tiene interoperatividad y similitud con el lenguaje Java, conocido y trabajado en otras asignaturas y reduce la cantidad de código repetitivo. Por lo tanto, el alumno tendrá como reto principal familiarizarse con un lenguaje y entorno de desarrollo nuevos.

4.1.2 Librerías y APIS

- **Firebase Auth:** Librería proporcionada por google que permite la identificación y registro de usuarios por varios métodos de autenticación, como por ejemplo email y contraseña, número de teléfono, cuenta de google, facebook y twitter entre otros.
- **Firestore Database:** Librería proporcionada por google que permite la gestión de base de datos NoSQL para almacenar y recuperar datos en formato de documentos y colecciones.
- **Glide:** Librería que permite visualizar y cargar imágenes almacenadas en internet por medio de enlaces, recursos o mapas de bits

- **Google Sign in:** Permite a los usuarios iniciar sesión con la cuenta de google por medio de sus credenciales.
- **Hilt:** Basado en la librería Dagger, permite el uso de inyección de dependencias para desarrollar código con más rapidez y facilidad.
- **Viewmodel:** Permite almacenar y gestionar datos de la interfaz gráfica de usuario y la lógica de negocio.
- **Live Data:** Permite el desarrollo de aplicaciones reactivas por medio de observadores
- **Coroutines:** Permite la lectura y escritura de código asíncrono como por ejemplo consultas y registros a la base de datos.
- **Material design**: Herramienta que permite tanto el desarrollo de interfaces de usuario como el uso de las ya creadas.
- **Navigation:** Permite la negación entre fragmentos en Android.
- **Lifecycle-extensions:** Gestiona el ciclo de vida de actividades y fragmentos de la aplicación.
- **ZXING:** Permite el uso de cámara como lector de código de barras.Para usar esta librería se habilita la aceleración de hardware en AndroidManifest.xml.
- **JUnit4:** Permite realizar las pruebas unitarias.

4.1.3 Sistema de control de versiones

Para obtener un mejor control de versiones implementadas y tener una copia de seguridad del proyecto en caso de algún error ajeno al estudiante, se utiliza el software de control de versiones ***Github***. La estrategia a seguir será la de GitHub Flow, puesto que al ser un proyecto en con un único desarrollador no se necesitan gestionar muchas versiones.

GitHub Flow

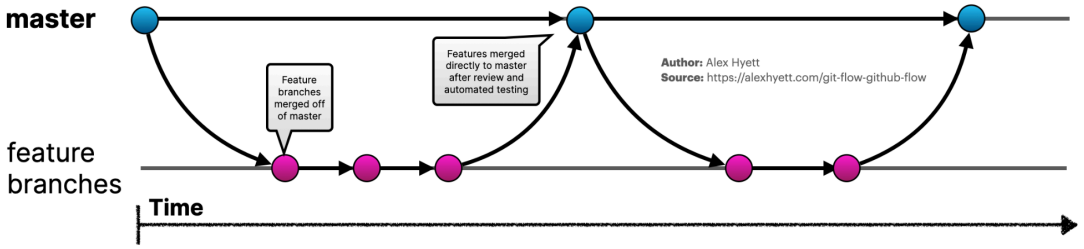


Ilustración 50- Arquitectura GitHubFlow

Por lo tanto, el estudiante trabajará sobre dos ramas: master y develop.

- **Master:** Rama principal en la que se tendrá en todo momento una versión funcional.
- **Develop**: Rama destinada a subir versiones de código funcionales que obtengan pequeños avances sobre algún avance especializado.

El enlace al repositorio del proyecto se define a continuación:

https://github.com/ericpm090/ProductManager

4.1.4 Firestore Database

Como ya se ha comentado en los puntos anteriores, se ha creído conveniente utilizar la base de datos *Firestore Database* como herramienta de gestión de base de datos. Los motivos de su elección son los siguientes

- Soporte que ofrece en modo *offline*, lo que permite a la aplicación ejecutar operaciones de lectura y escritura de datos de manera local cuando no hay conexión a Internet y posteriormente sincronizarlos cuando se vuelva a restablecer la conexión.
- Integración con *Firebase Auth* para gestionar el acceso de datos con los usuarios.
- Consulta de datos en tiempo real. Los datos se ven reflejados instantáneamente en tiempo real a todos los clientes conectados.
- Uso gratuito para aplicaciones con pocos datos.

- Escalabilidad horizontal: Maneja grandes volúmenes de datos y los distribuye en bloques.
- Flexibilidad: Permite agregar nuevos cambios o modificar los mismos sin tener que realizar cambios en las filas existentes.

Para poder poder realizar operaciones de lectura y escritura en la base de datos se han tenido que configurar las reglas que la definen:

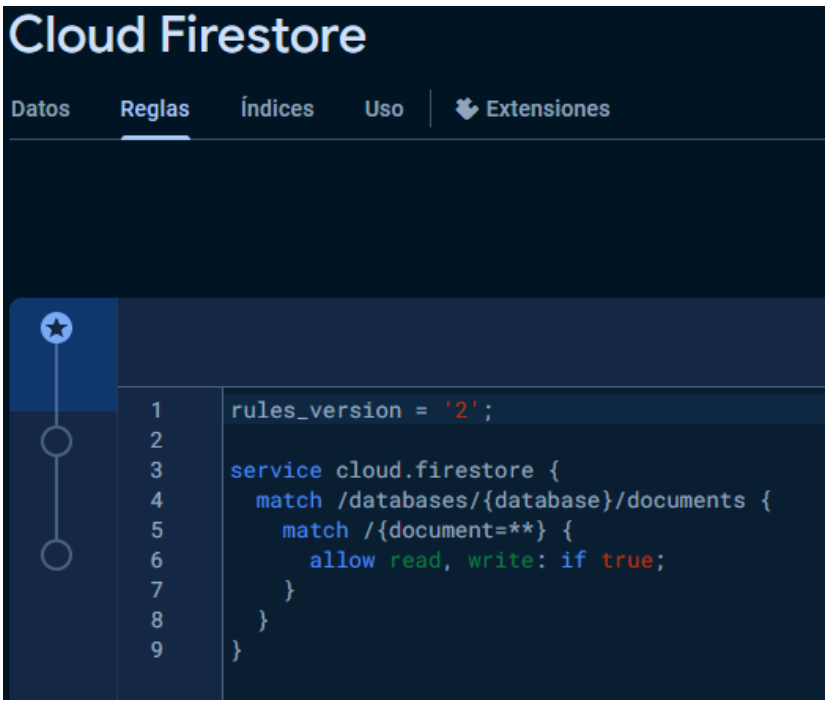


Ilustración 51- Configuración de reglas Cloud Firestore

Una vez configuradas las reglas, se procede a definir las diferentes colecciones creadas en Firestore Database así como su arquitectura:

- Locations: Colección para almacenar las localizaciones creadas por el administrador. Se define como clave primaria el nombre del proyecto para facilitar el uso de funcionalidades de búsqueda y obtención de datos.

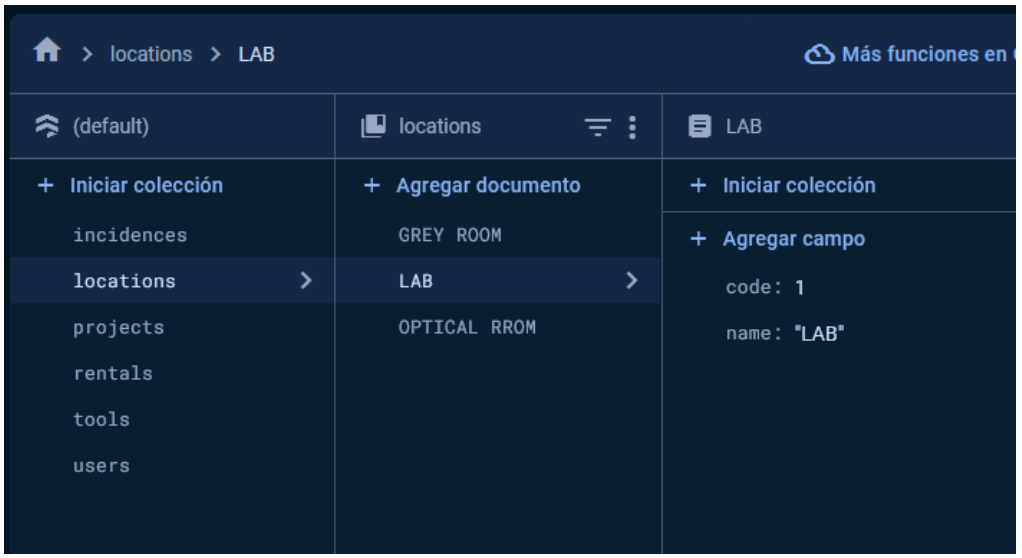


Ilustración 52 - Colección locations

- Projects: Colección para almacenar los proyectos creados por el administrador. Se define como clave primaria el nombre del proyecto para facilitar el uso de funcionalidades de búsqueda y obtención de datos.

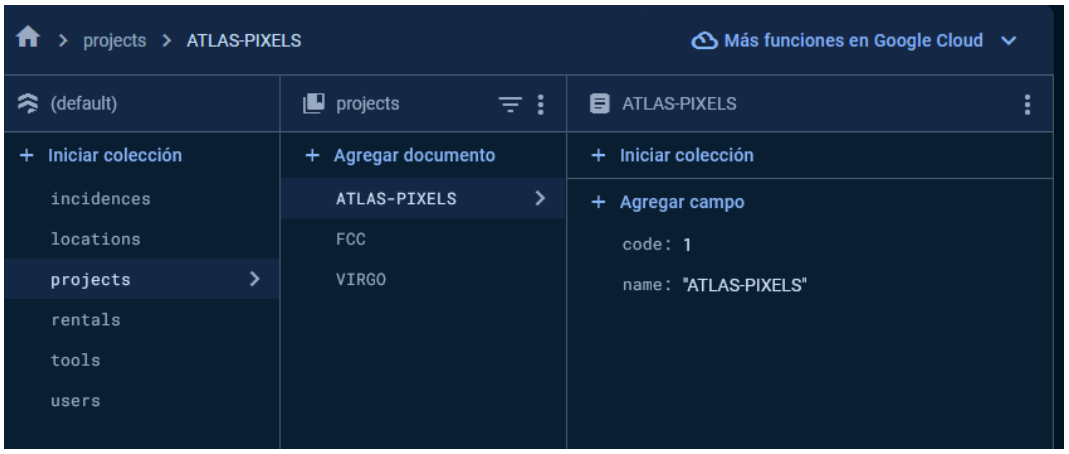


Ilustración 53- Colección projects

- Rentals: Colección para almacenar los correos de los usuarios que han registrado algún alquiler de herramienta. Esta colección almacena un documento que tiene como clave primaria el correo de usuario. El documento almacena una subcolección llamada “records” que almacena el código de las herramientas como documentos. Cada documento contiene un objeto tipo RentalTool.

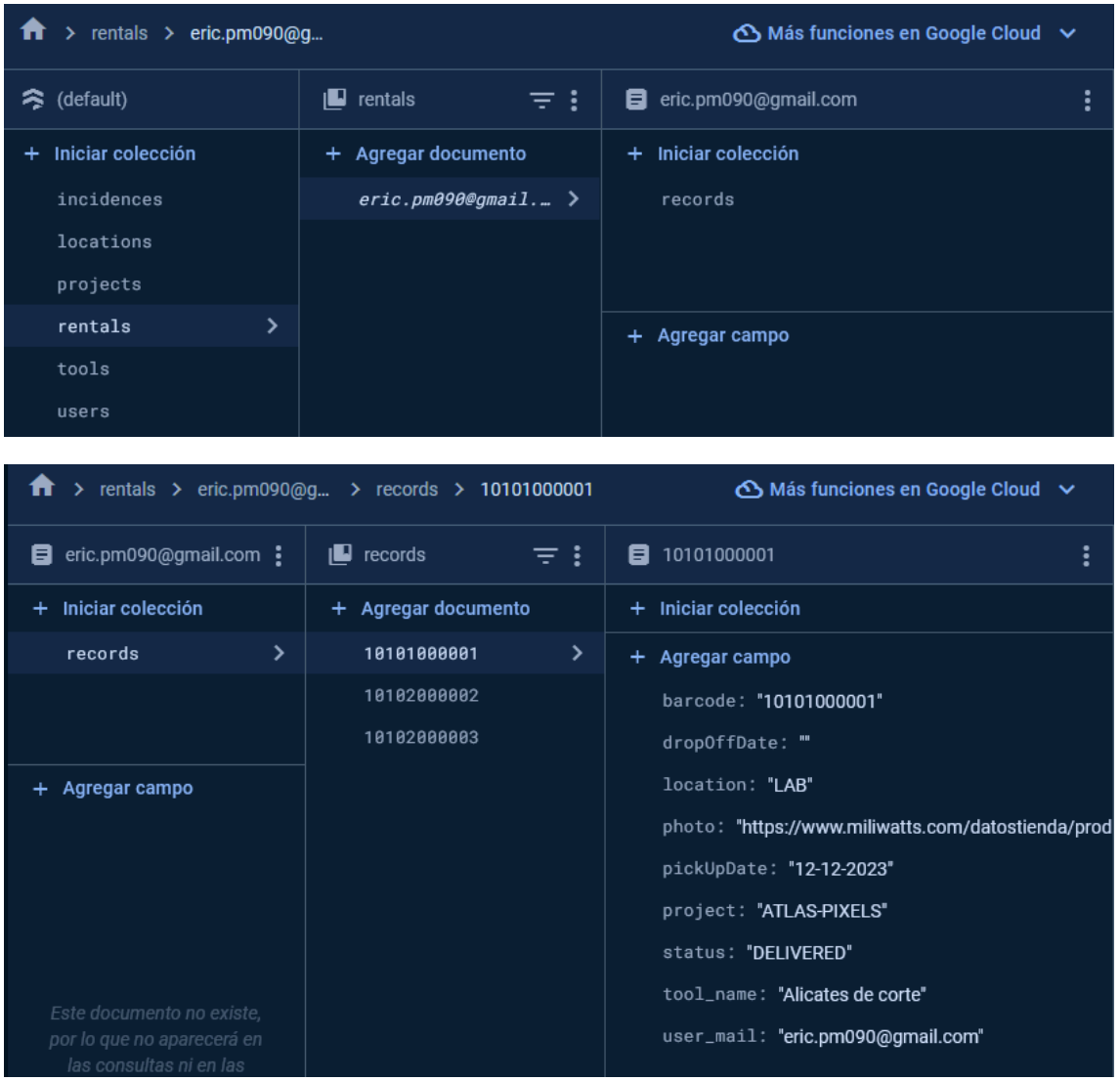


Ilustración 54- Colección rentals

- Tools: Colección para almacenar las herramientas creadas por el administrador. Se define como clave primaria el código de herramienta, el cual está compuesto por el código de localización, proyecto, tipo de herramienta y número de herramienta.

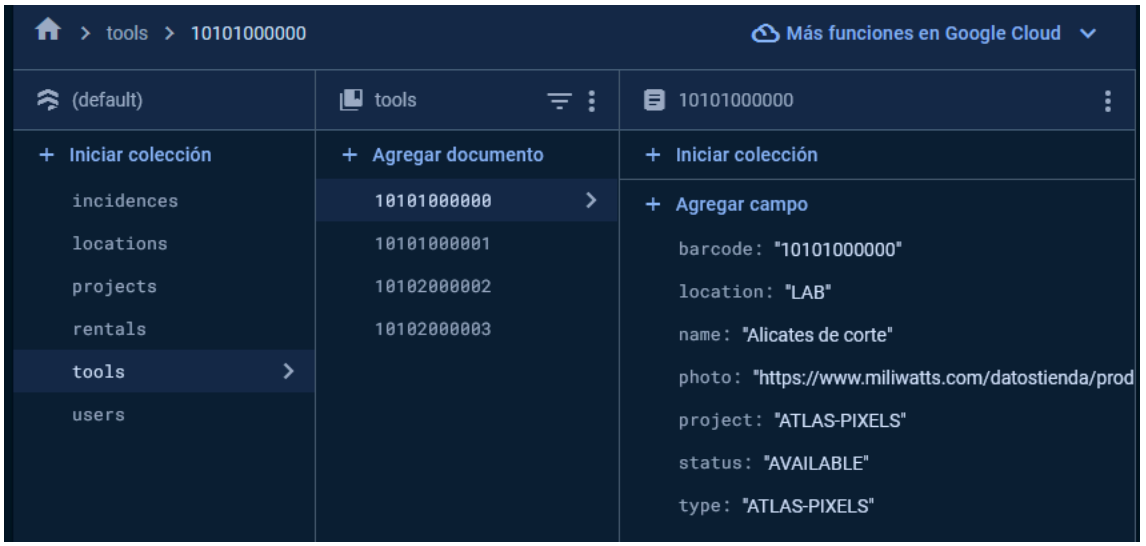


Ilustración 55- Colección tools

- Users: Colección para almacenar los usuarios registrados en el sistema, ya sea mediante el registro convencional como por Google Auth.

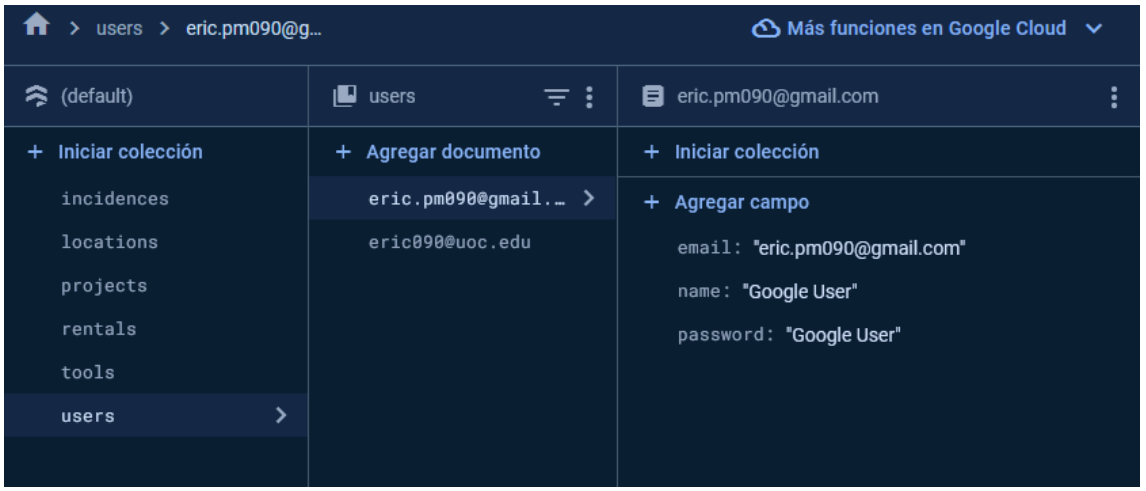


Ilustración 56- Colección users

- Incidences: Colección para almacenar las incidencias creadas por los usuarios y gestionadas por los administradores.

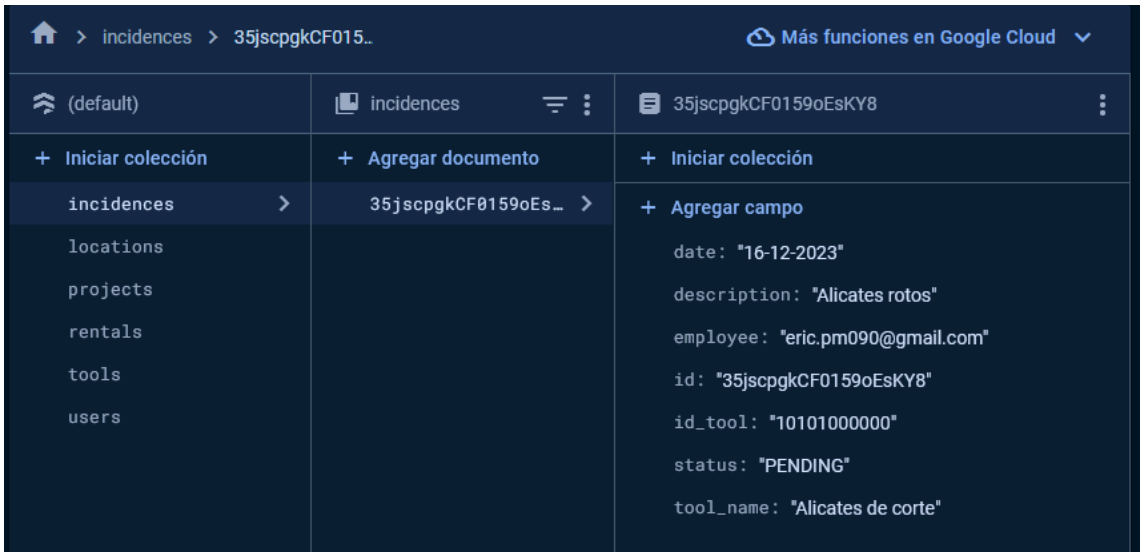


Ilustración 57- Colección incidences

4.1.5 Firebase Authentication

Para gestionar el registro y la identificación de usuarios se ha utilizado el sistema de registro que incorpora Firebase, puesto que es un servicio fácil de utilizar al no tener que desarrollar ninguna infraestructura desde cero y también por su gestión de registro de usuarios. Además, se ha aprovechado el servicio que ofrece para registrarse con diferentes proveedores, como Google, para facilitar la tarea de registro de usuario.

La integración de firebase authentication en el proyecto es totalmente compatible con otros servicios, como Firestore database, ya nombrado en el punto anterior. En este caso, el registro de un usuario implica el guardado de sus datos en la base de datos.

En cuanto a los servicios de identificación de usuarios, se ha decidido utilizar el método convencional de correo y contraseña en el que se comprueba si el correo y contraseña tienen formatos correctos. Además también se habilita la identificación por cuenta de google para facilitar el sistema de registro de usuarios:

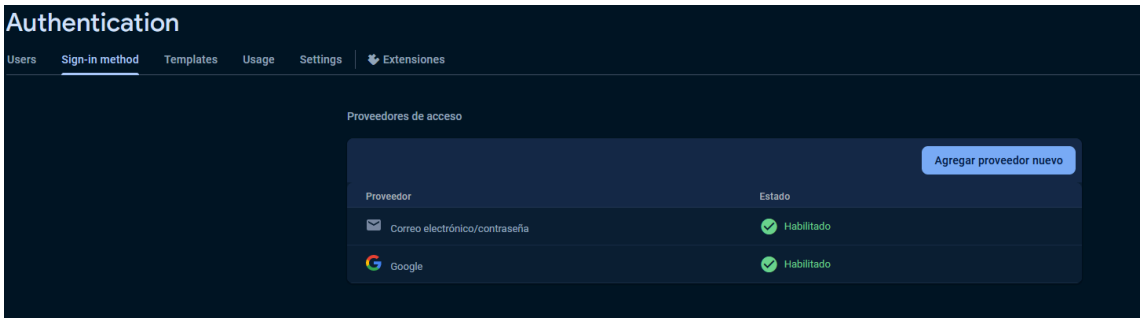


Ilustración 58- Métodos de Firebase Authentication habilitados

A continuación se muestra el registro de un usuario identificado por el servicio Google:

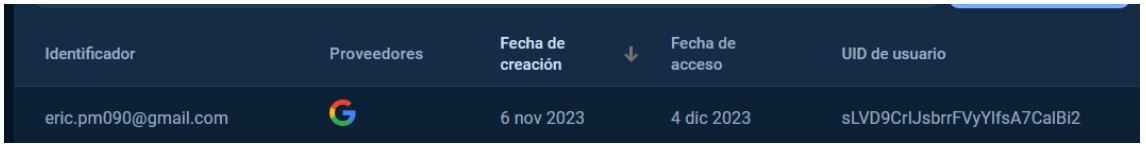


Ilustración 59- Ejemplo cuenta google registrada

4.2 Patrones SOLID

Durante el desarrollo de la aplicación se ha considerado seguir, en la medida posible, los principios de diseño SOLID.

- **Single Responsibility**: Una clase “debería tener una, y solo una, razón para cambiar”. Por lo tanto una clase debe ser responsable únicamente de sus responsabilidades y no de otras que no tengan nada que ver.
- **Open/Close**: Una clase debe estar abierta para la extensión pero cerrada para la modificación, por lo que las clases se deben diseñar para poder agregar nuevas funcionalidades sin tener que modificar el código existente.
- **Liskov** Substitution Las intendencias de una clase derivada debe poder sustituirse por instancias de la clase base sin afectar la corrección del programa. Esto permite reemplazar objetos por instancias de sus subtipos sin alterar el funcionamiento del sistema.

- **Interface Segregation**: Promueve que una clase debe utilizar una interfaz pequeña con los métodos que necesite implementar y no una interfaz grande y compleja que no se necesite.
- **Dependency Inversión**: Las clases de alto nivel no deben depender de las de bajo nivel. Para ello, las clases tanto de alto como de bajo nivel tienen que depender de abstracciones .

La implementación de estos patrones aportan al proyecto el desarrollo de un software de calidad, con un código robusto, limpio y flexible, por lo que permite realizar cambios sin necesidad de modificar prácticamente nada. También nos aporta tener un proyecto bien estructurado y escalable para añadir nuevas funcionalidades en un futuro de manera ágil.

Es posible que debido al corto tiempo de plazo de entrega no se lleguen a aplicar todos los principios, sin embargo se tratará de aplicar el principio de responsabilidad única.

4.3 Desarrollo de la aplicación

En este apartado se exponen los aspectos más relevantes del desarrollo de la aplicación, abarcando aspectos como la estructura interna del proyecto, en el que se explican las diferentes clases funcionalidades implementadas, la navegación de la interfaz de usuario así como la estructura de la base de datos y autentificación.

4.3.1 Estructura del proyecto

Como ya se ha explicado en apartados anteriores, se ha utilizado el entorno de desarrollo Android Studio, por lo que al crear un nuevo proyecto, el IDE provee un esqueleto básico generado por defecto. Sin embargo, como también se ha explicado en el apartado de diseño de la arquitectura, para el desarrollo de este proyecto se ha creído conveniente utilizar un patrón de diseño MVVM basado en la arquitectura Clean. La implementación de este tipo de arquitectura y patrón, aportan al proyecto una mejor organización de clases

ordenadas por su responsabilidad y capa de uso en cuanto a la arquitectura se refiere.

Por una parte, el patrón de diseño MVVM nos permitirá separar el modelo, la vista y el modelo de vista, por lo tanto la estructura de este patrón se define en estos ficheros:

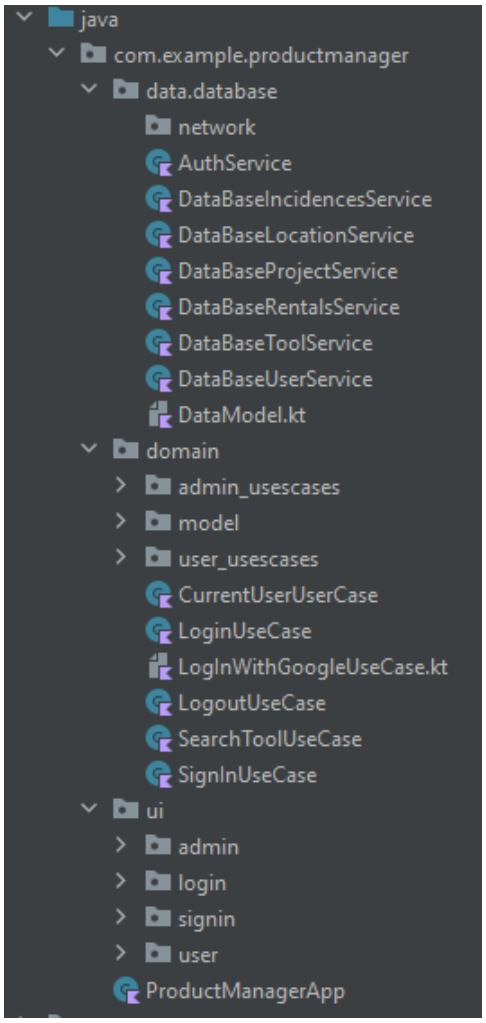
- **Vista (view):** Fichero con extensión ".xml" que se encarga de la representación de la interfaz gráfica del usuario.
- **Fragmento/Actividad:** Clase responsable de captar los eventos y definir el comportamiento de la vista.
- **Modelo Vista**: Clase que actúa como intermediario entre la vista y el modelo. Recibe los eventos captados por el fragmento o actividad, se comunica con el modelo para recibir información y la otorga de nuevo a la actividad o fragmento.
- **Modelo**: Representa la capa de datos y lógica de negocio de la aplicación. Se encarga de interactuar con bases de datos, consultas web, almacenar entidades y otras funciones.

Como veremos a proximamente, el uso del patrón MVVM implica utilizar el componente *LiveData* que se utiliza para mantener los datos observados. Esto permite tener observado un dato almacenado en el viewmodel para obtener los cambios y reflejarlos automáticamente en la interfaz de usuario por medio del data binding, encargado de enlazar los componentes de la interfaz de usuario con el modelo vista.

Por otro lado, teniendo en cuenta el uso de la arquitectura clean se estructura el proyecto en dos directorios:

- /java, se encuentran los directorios que representan las capas de la arquitectura.
- / res: Se almacenan todos los recursos utilizados por el directorio /java y por la propia aplicación.

Por lo tanto, a continuación se define la estructura del proyecto:



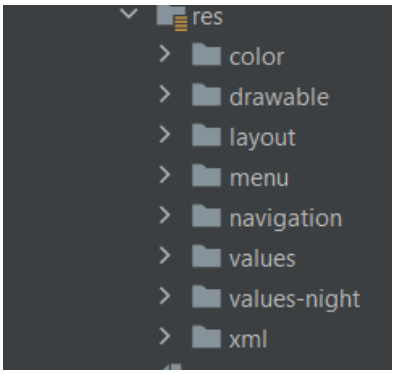


ilustración 60- Estructura del proyecto

Directorio: /java/com.example.productmanager:

- **domain (Datos + Casos de uso)**: Almacena la lógica de la aplicación, así como entidades, y servicios. Es habitual que la capa de dominio y la de uso de datos se representen de forma separada pero trabajen en conjunto, puesto que al final la capa de dominio necesita obtener los datos de la capa de datos.
- **data.Database(Datos)** : Almacena las clases encargadas de la gestión de base de datos.

- **ui (Presentación)**: Contiene las diferentes actividades y fragmentos de la aplicación. En ella podremos encontrar los archivos separados por tipo:
  - admin: Contiene las actividades y fragmentos que gestionan la interacción con las vistas de administrador.
  - user: Contiene actividades y fragmentos que gestionan la interacción con las vistas de usuario.
  - login: Contiene la actividad y *viewmodel* que gestiona la vista de la pantalla *login*.
  - singin: Contiene la actividad y *viewmodel* que gestiona la vista de la pantalla *signin*.

Directorio: /res

- **color:** Contiene los valores de los colores en formato hexadecimal.
- **drawable:** Contiene todos los archivos utilizados en las interfaces visuales, así como iconos o imágenes.
- **layout:** Contiene todas las interfaces visuales utilizadas en la aplicación y que están asociadas a una actividad, fragmento o pantalla para un recyclerview.
- **menu:** Espacio dedicado a almacenar los layouts que definen la estructura del menú de administrador y usuario, como son los menús inferiores y superiores.
- **navigation:** Espacio dedicado para almacenar datos de el menu de administrador y usuario.
- **values:** Contiene toda la información general de la aplicación, como son los textos, estilos, dimensiones y colores
- **xml:** Contiene el archivo de animación utilizado en el inicio de la aplicación. [Pendiente de implementación]

Además de los archivos y recursos que observados en las imágenes anteriores, en el proyecto también se encuentran otros archivos de gran importancia:

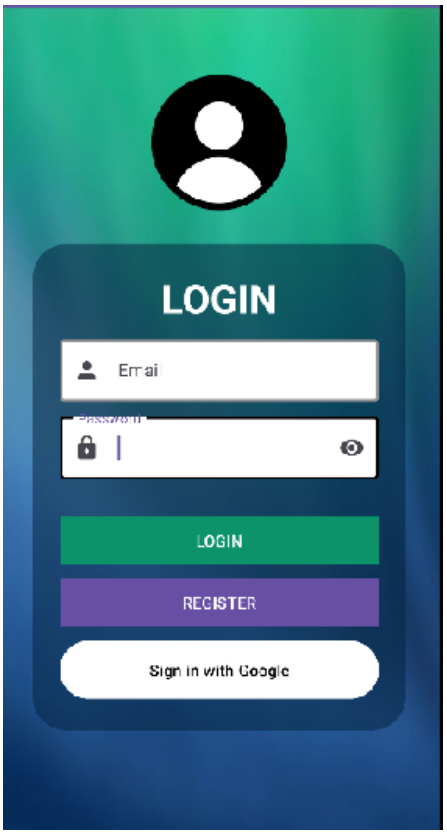
- **AndroidManifest.xml**: Archivo que define todas las actividades de la aplicación, servicios utilizados, actividad principal a ejecutar y otras configuraciones del sistema.
- **build.gradle**: Define toda la información de ejecución de la aplicación, así como los plugins, versión de SDK, buildTypes, opciones de compilación y dependencias entre otros

4.3.2 Interfaces de usuario

Este apartado tiene como objetivo describir el comportamiento de las diferentes interfaces visuales de la aplicación así como de las funcionalidades de las actividades y fragmentos así como los aspectos técnicos de cada pantalla.

Un aspecto importante a nombrar en cuanto al desarrollo de interfaces de usuario es que se ha tratado de implementar las interfaces gráficas según su responsabilidad, ya que en este caso se ha desarrollado una única aplicación que contiene dos tipos de comportamiento para dos tipos de usuarios.

4.3.2.1 Inicio de sesión

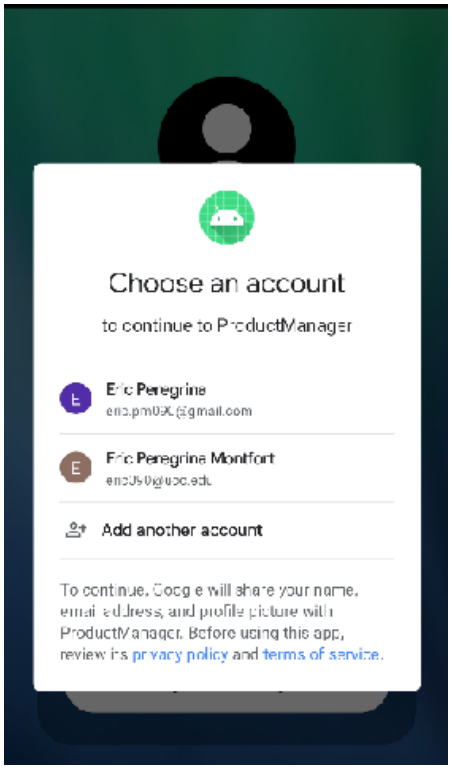


La activity LoginActivity se encarga de gestionar la autoidentificación de un usuario siempre que este exista. Este se realiza de dos formas: Identificación tradicional mediante correo y contraseña e identificación por medio de cuenta Google.

Los datos obtenidos en el método de identificación por correo y contraseña se realizan por medio de los objetos TextInputEditText proveídos del Material Design. En ellos, se han añadido funcionalidades extras, como el borrado del

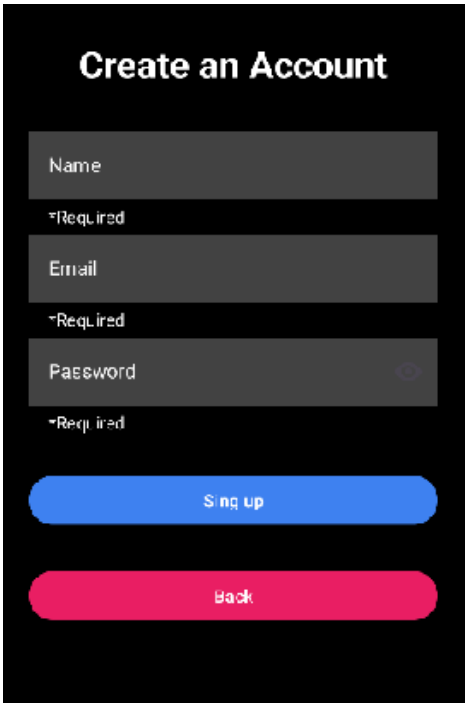
texto en el caso del correo y la ocultación de caracteres en el caso de la contraseña.

Si los datos ingresados son correctos y cumplen el formato adecuado, al pulsar el botón LOGIN se realizará una llamada a Firebase Autentificación para realizar la autentificación en el servició de signInWithEmailAndPassword. En el caso de que la identificación sea exitosa se iniciara la actividad UserActivity y el el usuario ingresara en el panel de usuario . Si por el contrario son incorrectos se informa al usuario mediante un Toast que el usuario o contraseña son incorrectos.



En cuanto a la identificación mediante cuenta Google se realiza una llamada al servicio createAccountWithCredential para iniciar una gestión externa que obtiene y verifica las credenciales de la cuenta de usuario indicadas. Si la identificación se realiza de forma exitosa se guarda el correo electrónico obtenido en Firestore database de la colección users. Posteriormente se iniciará la actividad UserActivity y el usuario ingresara en el panel de usuario.

4.3.2.2 Registro de usuario



La activity LogInActivity se encarga de obtener el nombre, correo y contraseña ingresados por el usuario en los objetos TextInputEditText. Al obtener los datos el ViewModel comprobará que el formato es correcto. Si los datos son correctos se realizará una llamada a Firestore database para guardar los datos dentro de la colección users. En caso contrario se informará al usuario por medio de un Toast que la contraseña o el correo son incorrectos.

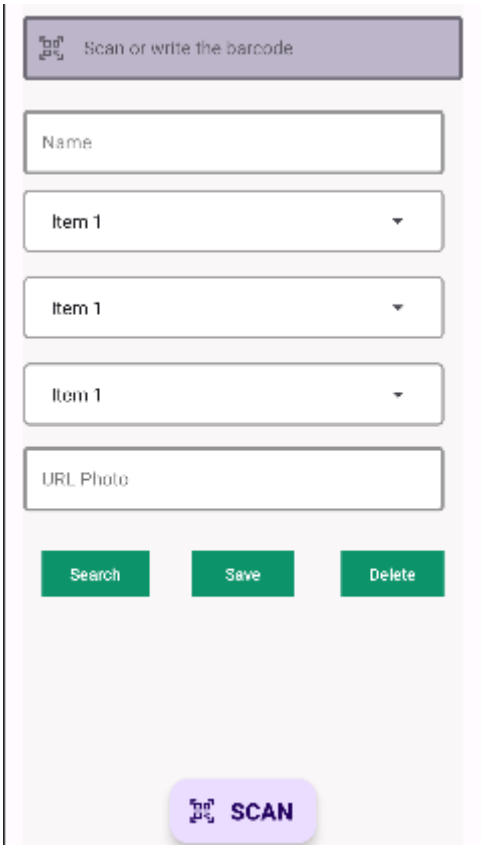
4.3.2.3 Admin - BottomNavigationView & AppBarLayout

La actividad AdminActivity cuenta con un menú inferior para ir navegando entre las diferentes pantallas habilitadas. Para ello se ha utilizado el objeto BottomNavigationView en el que se habilita la navegación en 5 fragmentos: Tools, Users, Projects, Locations, Incidences.

Se ha utilizado también el objeto AppBarLayout para informar en todo momento al administrador en que menú se encuentra y habilitar un submenú para cerrar sesión.

Las transiciones entre fragmentos se realizan mediante el método setOnItemSelectedListener asociado al BottomNavigationView, el cual se realiza una llamada al método replaceFragment y se introduce el fragmento obtenido y título como argumento.

4.3.2.4 Admin - Tools



El fragmento ToolsFragment es el encargado de buscar, crear y borrar herramientas. En él podemos encontrar objetos TextInputEditText para introducir el nombre y un enlace web para obtener una imagen mediante la liberia Glide.

Se utilizan también objetos Spinner para obtener un contenedor desplegable que informe de los diferentes proyectos, localizaciones y tipos de herramientas disponibles.

La información desplegada en los objetos Spinner se obtienen mediante varias llamadas a Firestore para obtener listas mutables de los diferentes tipos de objetos guardados.

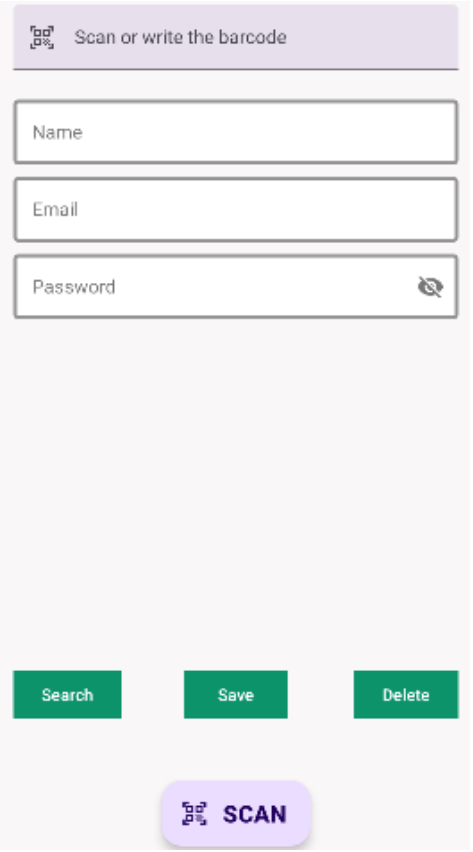
El botón buscar genera el evento de obtener la información introducida en el TextInputEditText de barcode. La información obtenida se consultará la clase en la base de datos mediante el método *get* y se gestionan los posibles eventos. Si no se encuentra ninguna herramienta asociada al código introducido se informará mediante el helpertext del objeto TextInputEditText de barcode. Por el contrario, se inicializan los diferentes objetos del formulario con la información recibida.

El botón guardar genera el evento para guardar toda la información introducida. El guardado de datos se realiza mediante una llamada al método *set* de firestore database para guardar el objeto en la colección tools.

El botón borrar tiene el mismo comportamiento que el botón buscar, pero a diferencia de él una vez obtenido el objeto, se realiza otra llamada al método *delete* para borrar el documento obtenido.

Para facilitar las tareas de búsqueda y borrado de herramientas se ha introducido un FloatingActionButton que acciona la cámara del dispositivo móvil para leer el código de la herramienta y ponerlo en el campo de código de herramientas.

4.3.2.5 Admin - Users



El fragmento UsersFragment es el encargado de obtener los datos de la interfaz y crear, buscar o borrar usuarios. Tiene el mismo comportamiento que el fragmento ToolsFragment, por lo que se considera no explicar los mismos detalles.

4.3.2.6 Admin - Projects

El fragmento ProjectsFragment es el encargado de obtener los datos de la interfaz y crear, buscar o borrar proyectos mediante su nombre o código. Tiene el mismo comportamiento que el fragmento ToolsFragment, por lo que se considera no explicar los mismos detalles

4.3.2.7 Admin - Locations

El fragmento LocationsFragment es el encargado de obtener los datos de la interfaz y crear, buscar o borrar localizaciones mediante su nombre o código. Tiene el mismo comportamiento que el fragmento ToolsFragment, por lo que se se considera no explicar los mismos detalles

4.3.2.8 Admin - Incidences

El fragmento IncidencesFragment es el encargado de mostrar al administrador las incidencias creadas por los usuarios.

Para realizar esta operación se necesita utilizar un botón que realiza una llamada a firestore database y obtiene una lista mutable.
La representación de los datos obtenidos se realiza mediante un objeto *recyclerview*, el cual es el encargado de representar cada objeto de la lista en la interfaz de usuario.

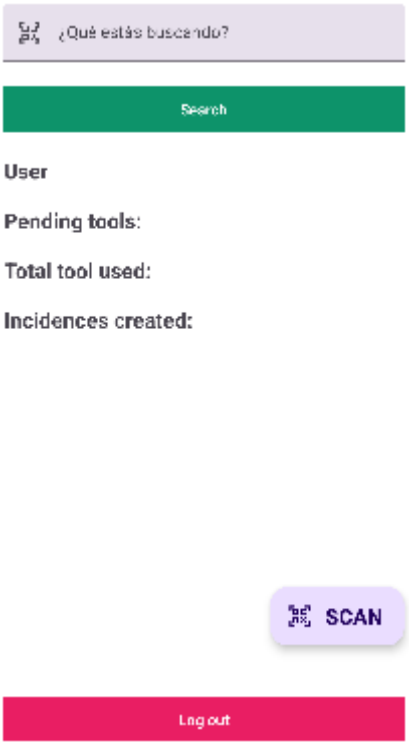
4.3.2.8 User - BottomNavigationView & AppBarLayout

La actividad UserActivity cuenta con un menú inferior para ir navegando entre las diferentes pantallas habilitadas. Para ello se ha utilizado el objeto BottomNavigationView en el que se habilita la navegación en 5 fragmentos: Home, Rentals, Returns, Loands, Incidents

De la misma manera que la actividad AdminActivity, se ha utilizado también el objeto AppBarLayout para informar en todo momento al usuario en qué menú se encuentra y habilitar un submenú para cerar sesión.

Las transiciones entre fragmentos se realizan mediante el método setOnItemSelectedListener asociado al BottomNavigationView, el cual se realiza una llamada al método replaceFragment y se introduce el fragmento obtenido y título como argumento.

4.3.2.9 User - Home



El fragmento HomeFragment es el encargado de facilitar al usuario sus datos de interés, así como el correo electrónico, herramientas pendientes de devolución, el total de herramientas alquiladas y número de incidencias creadas.

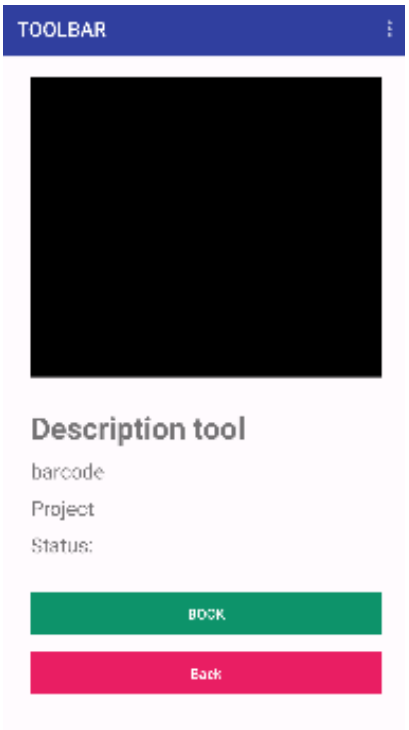
Además se habilita la búsqueda de herramientas mediante un objeto TextInputEditText y un botón de búsqueda, el cual acciona la búsqueda de la herramienta en la base de datos por medio de su código. Si la

herramienta se encuentra en la base de datos se iniciará la actividad ToolScreenActivity. Si por el contrario no existe se informará al usuario del error mediante el objeto helperText de TextInputEditText.

En cuanto a la lectura del código de barras se ha puesto un botón flotante que acciona la cámara y realiza la lectura del código de barras de la herramienta.

La actividad también cuenta con un botón de cerrar sesión para que el usuario pueda salir de la interfaz de usuario.

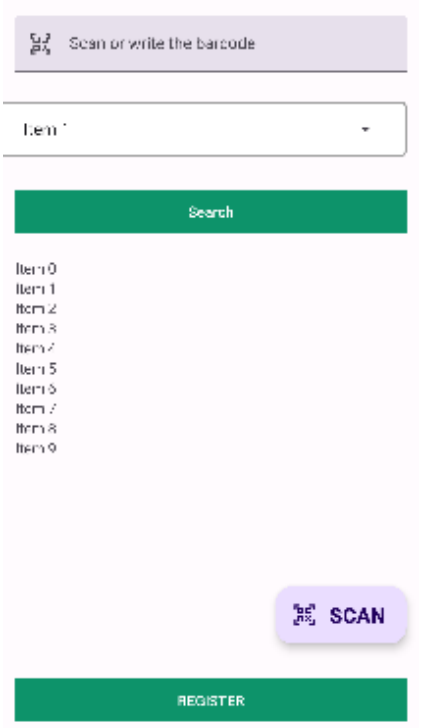
4.3.2.10 User - ToolScreen



La actividad ToolScrenActivity se encarga de representar la información recibida por el fragmento HomeFragment. La información a visualizar es la imagen, descripción, código, proyecto y estado de la herramienta.

Esta actividad se ha realizado para tener un acceso directo a la información de la herramienta y alquiler de la misma.

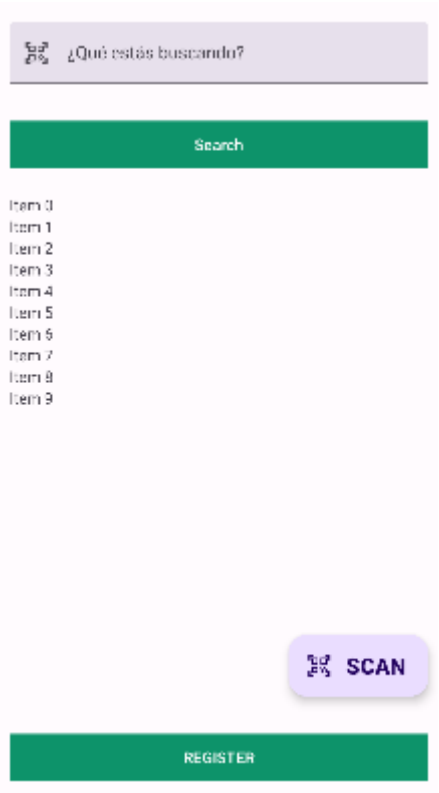
4.3.2.11 User - Rentals



El fragmento RentalFragment es el encargado de gestionar el alquiler de herramientas de un usuario. La gestión se realiza mediante la lectura del código de barras obtenido en el campo de código de barras y la información obtenida en el Spinner de localizaciones.

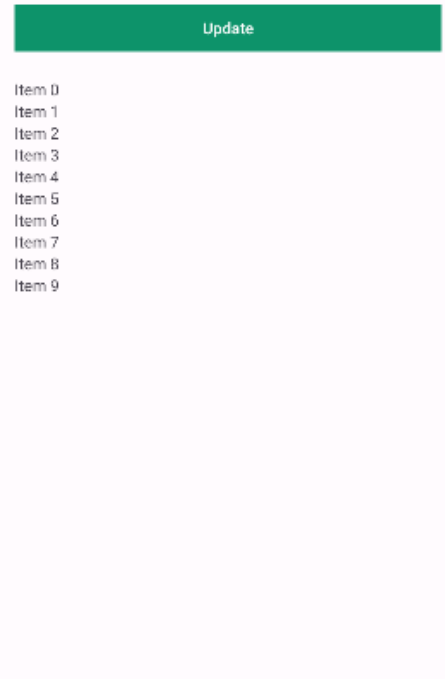
Un usuario podrá alquilar una herramienta si el código introducido es correcto y el campo de Spinner no está vació. Para ello, el usuario deberá introducir el código de las herramientas a utilizar y pulsar sobre el botón de búsqueda. Si la herramienta existe y está disponible será almacenada en la lista utilizada por un recyclerview y se mostrará en la pantalla. Por el contrario se muestra un error por medio de un Toast. Una vez finalizada la búsqueda el usuario deberá registrar la lista de herramientas pulsando sobre el botón registrar. Las operaciones de búsqueda y registro se realizan mediante las operaciones de set y get de firebase en la coleccion *rentals*, en el que se gestionan objetos tipo RentalTool. Cuando se registra un objeto RentalTool en la base de datos se procede a cambiar el estado de la herramienta a “NOT AVAILABLE”. De la misma, el estado del objeto creado tipo RentalTool asociado a un usuario cambia a “PENDING”.

4.3.2.12 User - Returns



El fragmento de ReturnsFragment es el encargado de gestionar las devoluciones realizadas por un usuario. El comportamiento de este fragmento es el mismo que el de RentalFragment con la diferencia que cuando un usuario registra la devolución de una herramienta se cambia el estado de la misma a “AVAILABLE” así como el estado del objeto tipo RentalTool del usuario a “DELIVERED”.

4.3.2.13 User - Loands



El fragmento LoandsFragment es el encargado de representar las herramientas pendientes de devolución de un usuario. Para ello, el usuario debe pulsar sobre el botón Update. Al pulsar sobre él, se realiza una búsqueda en la colección “rentals” de Firestore para obtener una lista de objetos RentalTool con el estado “PENDING”. El resultado se muestra en un objeto recyclerview encargado de representar los datos de la lista obtenida.

4.3.2.14 User - Incidents

El fragmento IncidentsFragment es el encargado de gestionar las incidencias creadas por el usuario.

Para crear una incidencia, un usuario debe introducir el código de barras de herramientas y la descripción del incidente. Para registrar la incidencia se debe pulsar sobre el botón REGISTRAR. Si el código de la herramienta es erróneo se avisará mediante un objeto Toast del error.
Las incidencias creadas se guardaran en la colección incidences de firestore.

4.3.3 Operaciones del sistema

Una vez definidas las interfaces de usuario, se describen las diferentes operaciones del sistema implementadas a lo largo del desarrollo de la aplicación.

Actor	Caso de uso	Comando	Descripción
Administrador	Modificar un usuario	saveUser()	Modifica un usuario
Administrador	Buscar un usuario	getUser()	Obtiene un usuario
Administrador	Crear herramientas	saveTool()	Guarda una herramienta en la base de datos

Administrador / Employee	Buscar herramientas por código de barras	getToolByBarcode()	Busca una herramienta a partir del código de barras indicado.
Administrador	Cambiar el estado de una herramienta	changeStatus()	Modifica el estado de una herramienta
Administrador	Saber el estado de una herramienta	isAvailable()	Obtiene el estado de una herramienta
Administrador / Empleado	Obtener las localizaciones creadas	getLocations()	Obtiene un listado de las localizaciones creadas.
Administrador	Obtener las incidencias de los usuarios	getIncidences()	Obtiene las incidencias creadas de todos los usuarios
Administrador	Solucionar una incidencia	solveIncidence()	Modifica el estado de una incidencia con el estado SOLVED
Administrador	Notificar a los usuarios sobre una incidencia solucionada	notify()	Notifica al usuario que ha creado la incidencia que el estado de la misma está

			resuelta.
Empleado	Obtener las herramientas en alquiler	getAllPendintTools()	Obtiene un listado de las herramientas a devolver.
Empleado	Obtener el número de histórico de las herramientas utilizadas	getRentalHistory()	Obtiene un listado de las herramientas que ha utilizado el usuario.
Empleado	Obtener el número de incidencias creadas	getIncidencesByEmal()	Obtiene un listado de las incidencias que ha creado el usuario.
Empleado	Crear una incidencia de una herramienta	createIncidence()	Crea una incidencia en la base de datos.
Empleado	Registrar el alquiler de una herramienta	saveRentalTool()	Guarda el alquiler de una herramienta en la base de datos.
Empleado	Obtener la información de una herramienta alquilada	findPendingTool()	Obtiene una herramienta en alquiler.
Empleado	Registrar la devolución de una herramienta	deliveryRentalTools()	Registra el retorno de una herramienta. y

			cambia el estado de la misma a DELIVERED

4.3.3 Servicios del sistema

Para obtener una mejor organización de las operaciones del sistema, se ha creído conveniente organizarlas en diferentes servicios.

- **User service:** Gestiona los usuarios. Contiene las operaciones de guardado y obtención de usuarios.
- **Tool Service:** Gestiona las herramientas. Contiene las operaciones de guardado, búsqueda, cambio de estado y consulta de estado de las herramientas.
- **Location Service:** Gestiona las localizaciones. Contiene las operaciones de crear, buscar y obtener localizaciones.
- **RentalService**: Gestiona el alquiler de herramientas. Contiene las operaciones de búsqueda de herramientas pendientes, registro de alquileres y obtención de historial de herramientas alquiladas.
- **NotificationService:** Gestiona las notificaciones a los usuarios.Contiene la operación de notificar al usuario cuando una incidencia ha sido solucionada.

A continuación se muestra un resumen de las operaciones implementadas en relación a los servicios identificados:

Servicio	Operaciones
User Service	• saveUser() • getUser()
Tool Service	• saveTool() • getToolByBarcode()

	• changeStatus() • isAvailable()
RentalTool Service	• findPendingTool() • deliveryRentalTools() • rentalTools() • saveRentalTool() • getAllPendingTools() • getRentalHistory()
Notification Service	• notify()
Location Service	• getLocations() • createLocation() • saveLocation() • searchLocation() • deleteLocation()
Incidences Service	• createIncidence() • saveIncidence() • getIncidences() • solveIncidence() • updateIncidence() • getIncidencesByEmail()

5. Estado actual del proyecto

El apartado de pruebas no se ha podido realizar a tiempo pues la planificación planteada al principio del proyecto se ha visto afectada.

27		Gestión de notificaciónes	20 horas	dom 03/12/23	sáb 09/12/23	26
28		**Test**	**18 horas**	**sáb 09/12/23**	**jue 14/12/23**	
29		Pruebas unitarias	5 horas	sáb 09/12/23	dom 10/12/23	27
30		Pruebas de integración	5 horas	dom 10/12/23	mar 12/12/23	29
31		Mantenimiento	8 horas	mar 12/12/23	jue 14/12/23	30
32		**Entrega final**	**60 horas**	**vie 15/12/23**	**dom 31/12/2**	
33		Mejoras	24 horas	vie 15/12/23	jue 21/12/23	31
34		Memoria	24 horas	jue 21/12/23	jue 28/12/23	33
35		Manual de usuario	12 horas	jue 28/12/23	dom 31/12/23	34
36		**Defensa**	**12 horas**	**dom 31/12/2**	**mié 03/01/24**	
37		Presentación del trabajo final	12 horas	dom 31/12/23	mié 03/01/24	35

Como se puede observar, el proyecto debería estar en la fase de la entrega final, sin embargo, a pesar de haber desarrollado la aplicación al 100% faltan acabar algunos detalles, como mejora de las interfaces visuales, revisión de código, detección y correción de bugs y realizar la fase de pruebas

Esto es debido a que la estimación de número de horas previsto para desarrollar la aplicación ha sido insuficiente, pues el alumno ha tenido que invertir casi el doble de horas para aprender un nuevo lenguaje y conceptos nuevos del mundo de la programación móvil.

Por lo tanto, se cree oportuno realizar una modificación en la planificación del proyecto y aumentar el número de horas empleadas hasta la entrega final. Las horas a aumentar serán un total de 46 horas semanales en vez de las 25 horas planificadas anteriormente. Para ello, se emplearán 6 horas de lunes a viernes y 8 los fines de semana. Esto supondrá añadir 21 horas más al proyecto dedicado al apartado de mejoras, pruebas unitarias y de integración.

A continuación se muestra la nueva planificación en lo que queda de fases de proyecto:

		Test	**28 horas**	**mar 19/12/23**	**mar 26/12/23**	
		Pruebas unitarias	10 horas	mar 19/12/23	vie 22/12/23	
		Pruebas de integración	10 horas	vie 22/12/23	dom 24/12/23	29
		Mantenimiento	8 horas	dom 24/12/23	mar 26/12/23	30
		Entrega final	**67 horas**	**mié 27/12/23**	**dom 14/01/2**	
		Mejoras	31 horas	mié 27/12/23	jue 04/01/24	31
		Memoria	24 horas	vie 05/01/24	jue 11/01/24	33
		Manual de usuario	12 horas	jue 11/01/24	dom 14/01/24	34
		Defensa	**12 horas**	**dom 14/01/2**	**jue 18/01/24**	
		Presentación del trabajo final	12 horas	dom 14/01/24	jue 18/01/24	35

6. Pruebas

Una vez finalizado el desarrollo de la aplicación móvil, se inicia el apartado de pruebas. Este apartado, es de gran importancia, pues nos confirmará que nuestro código sea de calidad y funcional. Con ello, lograremos reducir costes, optimizar el rendimiento, evaluar la usabilidad y garantizar la compatibilidad entre otras cosas.

Para lograr todos estos aspectos, en este proyecto se ha considerado realizar las pruebas tanto en un dispositivo como en un emulador. De esta manera, además de comprobar el funcionamiento en el emulador, podemos evaluar la aplicación en un entorno real.

6.1 Pruebas unitarias

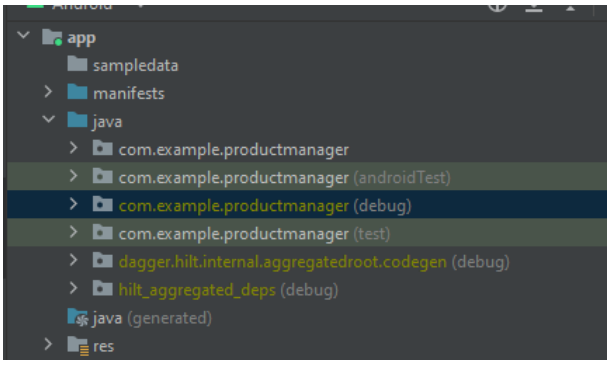
Las pruebas unitarias permiten comprobar que todas las funcionalidades desarrolladas cumplen con su propósito y enfoque específico. Esto resulta de gran utilidad cuando se modifica o se crea una mejora, pues se podrán detectar posibles anomalías.

Para evitar eventos externos y comprobar su correcto funcionamiento se ejecutan de forma aislada y únicamente se basan en la ejecución de casos de usos que cubren todos los aspectos de la aplicación.

6.1.1 Directorios

En el esqueleto del proyecto, encontramos dos directorios para guardar las pruebas:

- **AndroidTest:** Directorio en el que se almacenan pruebas que utilizan el framework de Android.
- **Test**: Directorio en el que se almacenan pruebas que no utilizan el framework de Android.



En nuestro caso, al tratarse de métodos unitarios en los que no se utilizara el emulador, las clases de prueba se almacenarán en el directorio **Test**.

6.1.2 Definición de las pruebas

Las pruebas unitarias implementadas a implementar en este proyecto son las definidas en los casos de uso e implementadas en los servicios:

Administrador

1. Crear herramientas a partir de su nombre, proyecto, localización, tipo y url de imagen.
2. Buscar herramientas por código de barras
3. Borrar herramientas
4. Consultar el historial de herramientas
5. Buscar usuarios por correo
6. Modificar usuarios
7. Borrar usuarios
8. Crear proyectos a partir de su nombre
9. Buscar proyectos por nombre

10. Borrar proyectos
11. Crear localizaciones a partir de su nombre
12. Buscar localizaciones por nombre
13. Borrar localizaciones
14. Consultar incidencias
15. Resolver incidencias

Usuario no registrado:

16. Registro de usuario

Empleado:

17. Consultar información de usuario: Herramientas pendientes a devolver, número total de herramientas utilizadas, incidencias creadas.
18. Buscar una herramienta por su código de barras
19. Alquilar una herramienta
20. Devolver una herramienta
21. Crear incidencias

Para poder realizar las pruebas unitarias utilizaremos el objeto Mock para simular las clases y así poder acceder a los métodos que nos interese testear.

Como se ha comentado en el apartado anterior, se ha considerado agrupar las funcionalidades en servicios de sistema para organizar los casos de uso por tipo. Al aplicar la arquitectura clean, estos servicios se ubican en la capa de modelo con el fin de cumplir con uno de los principios SOLID, el principio de responsabilidad única.

A continuación se exponen las pruebas unitarias de los diferentes servicios del proyecto:

ToolServiceTest: Implementa las pruebas de crear, buscar y borrar herramientas. Las pruebas realizadas se describen a continuación:

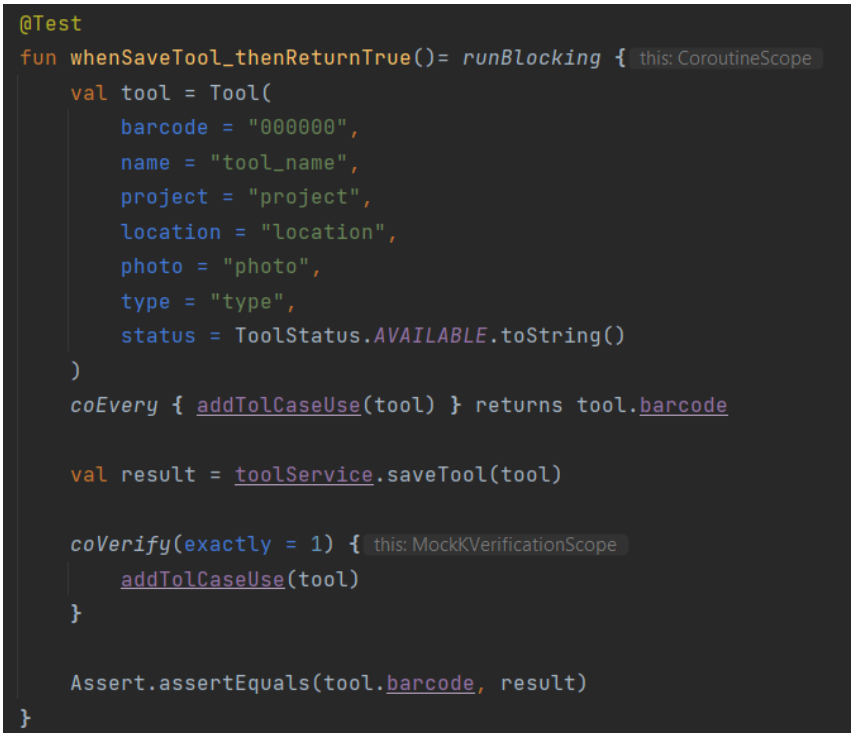
- **whenSearchExistenTool_thenReturnTool**: Comprueba que el método getToolByBarcode devuelve una objeto Tool existente.

```
@Test
fun whenSearchExistentTool_thenReturnTool()= runBlocking { this: CoroutineScope
    val tool = Tool(
        barcode = "000000",
        name = "tool_name",
        project = "project",
        location = "location",
        photo = "photo",
        type = "type",
        status = ToolStatus.AVAILABLE.toString()
    )

    coEvery { searchToolUsecase(tool.barcode) } returns tool

    val result = toolService.getToolbyBarcode(tool.barcode)

    coVerify(exactly = 1) { this: MockKVerificationScope
        searchToolUsecase(tool.barcode)
    }

    Assert.assertEquals(tool, result)
}
```

- **whenSearchNonExistentTool_thenReturnNull**: Comprueba que el metodo getToolByBarcode devuelve un valor null cuando se busca una herramienta inexistente.

```
@Test
fun whenSearchNonExistentTool_thenReturnNull()= runBlocking { this: CoroutineScope
    val barcode = ""
    coEvery { searchToolUsecase(barcode) } returns null

    val result = toolService.getToolbyBarcode(barcode)

    coVerify(exactly = 1) { this: MockKVerificationScope
        searchToolUsecase(barcode)
    }

    Assert.assertEquals( expected: null, result)
}
```

- **whenSaveTool_thenReturnTrue**: Comprueba que el metodo saveTool devuelve una herramienta cuando esta se guarda.



```
@Test
fun whenSaveTool_thenReturnTrue()= runBlocking { this: CoroutineScope
    val tool = Tool(
        barcode = "000000",
        name = "tool_name",
        project = "project",
        location = "location",
        photo = "photo",
        type = "type",
        status = ToolStatus.AVAILABLE.toString()
    )
    coEvery { addTolCaseUse(tool) } returns tool.barcode

    val result = toolService.saveTool(tool)

    coVerify(exactly = 1) { this: MockKVerificationScope
        addTolCaseUse(tool)
    }

    Assert.assertEquals(tool.barcode, result)
}
```

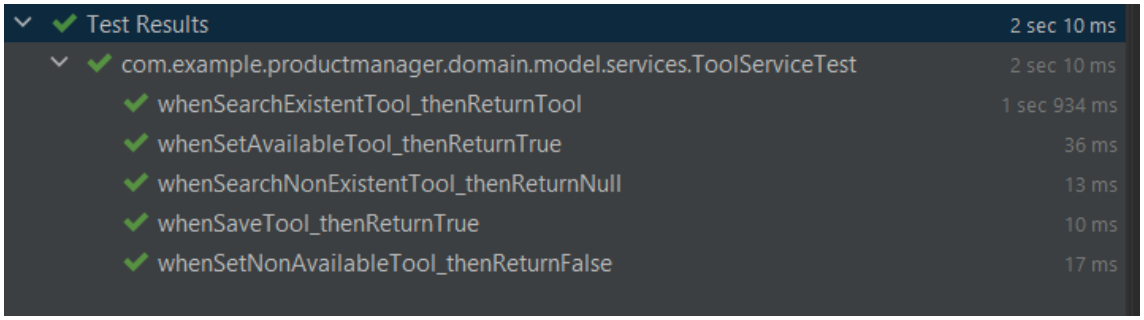
- **whenSetNonAvailableTool_thenReturnFalse**: Comprueba que el método changeStatus devuelve el estado “NOT AVAILABLE” cuando se establece una herramienta como no disponible.

```
@Test
fun whenSetNonAvailableTool_thenReturnFalse()= runBlocking { this: CoroutineScope
    val tool = Tool(
        barcode = "000000",
        name = "tool_name",
        project = "project",
        location = "location",
        photo = "photo",
        type = "type",
        status = ToolStatus.AVAILABLE.toString()
    )
    //Given
    coEvery { searchToolUsecase(tool.barcode) } returns tool
    coEvery { addTolCaseUse(tool) } returns tool.barcode

    //When
    toolService.changeStatus(tool.barcode)
    val result = toolService.isAvailable(tool)

    coVerify(exactly = 1) { this: MockKVerificationScope
        searchToolUsecase(tool.barcode)
    }
    coVerify(exactly = 1) { this: MockKVerificationScope
        addTolCaseUse(tool)
    }
    Assert.assertEquals( expected: false, result)
}
```

- **whenSetAvailableTool_thenReturnTrue**: Comprueba que el metodo changeStatus devuelve el estado “AVAILABLE” cuando se establece una herramienta como disponible

```
@Test
fun whenSetAvailableTool_thenReturnTrue()= runBlocking { this: CoroutineScope
    val tool = Tool(
        barcode = "000000",
        name = "tool_name",
        project = "project",
        location = "location",
        photo = "photo",
        type = "type",
        status = ToolStatus.NOT_AVAILABLE.toString()
    )
    //Given
    coEvery { searchToolUsecase(tool.barcode) } returns tool
    coEvery { addTolCaseUse(tool) } returns tool.barcode
    //When
    toolService.changeStatus(tool.barcode)
    val result = toolService.isAvailable(tool)

    coVerify(exactly = 1) { this: MockKVerificationScope
        searchToolUsecase(tool.barcode)
    }
    coVerify(exactly = 1) { this: MockKVerificationScope
        addTolCaseUse(tool)
    }
    Assert.assertEquals( expected: true, result)
}
```

A continuación se muestran los resultados de los test unitarios de ToolService:



ProjectServiceTest: Implementa las pruebas de crear, buscar y borrar proyectos.

- **whenSearchInvalidProject_thenReturnNull:** Comprueba que cuando se busca un proyecto no existente devuelve un valor nulo.

```
@Test
fun whenSearchInvalidProject_thenReturnNull()= runBlocking { this: CoroutineScope


    coEvery { searchProjectUseCase( name: "") } returns null

    val result = projectService.get("")

    coVerify(exactly = 1) { searchProjectUseCase( name: "") }

    assertEquals( expected: null,result)
}
```

- **whenSearchProject_thenReturnProject:** Comprueba que cuando se busca un proyecto existente se devuelve el objeto Project correspondiente.

```
@Test
fun whenSearchProject_thenReturnProject()= runBlocking { this: CoroutineScope
    val project = Project(
        code = 1L,
        name = "Project"
    )
    coEvery { searchProjectUseCase(project.name) } returns project

    val result = projectService.get(project.name)

    coVerify(exactly = 1) { searchProjectUseCase(project.name) }

    assertEquals(project,result)

}
```

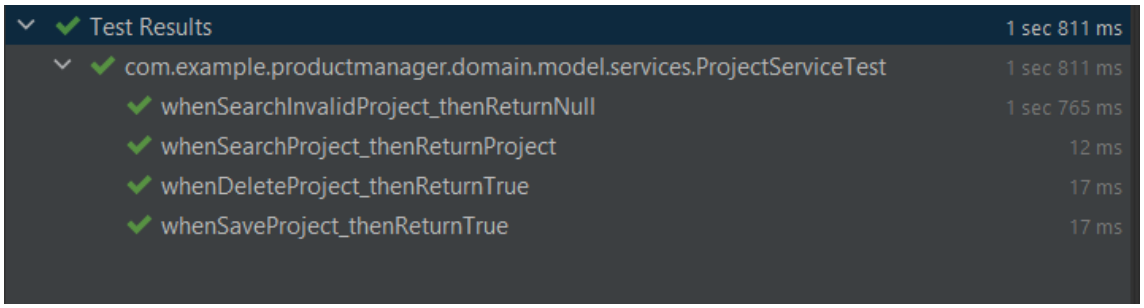
- **whenDeleteProject_thenReturnTrue:** Comprueba que cuando se borra un proyecto exitosamente se devuelve un valor booleano true.

```
@Test
fun whenDeleteProject_thenReturnTrue()= runBlocking { this: CoroutineScope
    coEvery { deleteProjectUseCase( name: "Project") } returns true

    val result = projectService.delete( name: "Project")

    coVerify(exactly = 1) { deleteProjectUseCase( name: "Project") }

    assertEquals( expected: true,result)
}
```

- **whenSaveProject_thenReturnTrue:** Comprueba que cuando se guarda un proyecto de forma exitosa devuelve un valor booleano true.

```
@Test
fun whenSaveProject_thenReturnTrue()= runBlocking { this: CoroutineScope
    coEvery { addProjectUseCase( name: "Project") } returns true

    val result = projectService.save( name: "Project")

    coVerify(exactly = 1) { addProjectUseCase( name: "Project") }

    assertEquals( expected: true,result)
}
```

A continuación se muestran los resultados de los test unitarios ProjectService:

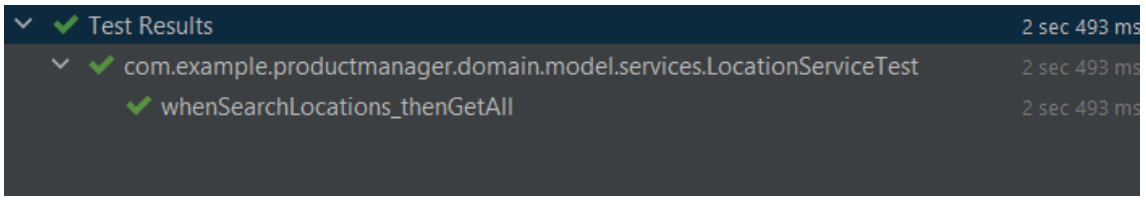


LocationServiceTest: Implementa las pruebas de crear, buscar y borrar localizaciones.

- **whenSearchLocations_thenGetAll:** Comprueba que el método getLocations devuelve una lista de localizaciones.

```
@Test
fun whenSearchLocations_thenGetAll()= runBlocking { this: CoroutineScope
    val location1 = Location(
        code: 1L,  name: "0001"
    )
    val location2 = Location(
        code: 2L,  name: "0002"
    )
    val location3 = Location(
        code: 3L,  name: "0003"
    )
    val locations = mutableListOf(location1,location2,location3)

    coEvery { getAllLocationsUseCase() } returns locations

    val result = locationService.getLocations()
    coVerify(exactly = 1){getAllLocationsUseCase()  }

    Assert.assertEquals(locations.size, result.size)

}
```

A continuación se muestran los resultados de los test unitarios de LocationService:



UserServiceTest: Implementa las pruebas de crear, buscar y borrar usuarios.

- **whenCreateUser_thenGetUser:** Comprueba que al crear un usuario se obtiene un objeto booleano de valor true.

```
@Test
fun whenCreateUser_thenGetUser() = runBlocking { this: CoroutineScope
    val employee = Employee(
        name = "User",
        email = "user@gmail.com",
        password = "1234"
    )
    //Given
    //coEvery para corutinas
    coEvery { this: MockKMatcherScope
        //userService.getUser(employee.email)} returns employee
        modifyUserUseCase(employee)
    } returns true

    //When
    // userService.getUser(employee.email)
    val result = userService.saveUser(employee)
    //Then
    //verificamos que se llama una vez
    coVerify(exactly = 1) { this: MockKVerificationScope
        modifyUserUseCase(employee)
    }
    //verificamos que obtenemos los resultados esperados
    assertEquals( expected: true, result)
}
```

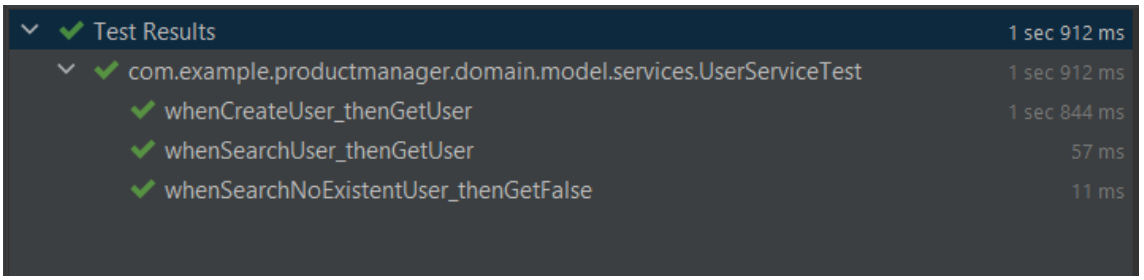
- **whenSearchUser_thenGetUser:** Comprueba que al buscar un usuario se obtiene el objeto Employee buscado.

```
@Test
fun whenSearchUser_thenGetUser() = runBlocking { this: CoroutineScope

    val employee = Employee(
        name = "User",
        email = "user@gmail.com",
        password = "1234"
    )
    //Given
    coEvery { searchUseUseCase(employee.email) } returns employee

    //When
    val result = userService.getUser(employee.email)
    coVerify(exactly = 1) { this: MockKVerificationScope
        searchUseUseCase(employee.email)
    }

    assertEquals(employee, result)
}
```

- **whenSearchNotExistentUser_thenGetFalse:** Comprueba que al buscar un usuario no existente se devuelve un objeto booleano con valor false.

```
@Test
fun whenSearchNoExistentUser_thenGetFalse() = runBlocking { this: CoroutineScope
    val email = "noexistentuser@gmail.com"
    //Given
    coEvery { searchUseUseCase(email) } returns null

    //When
    val result = userService.getUser(email)
    coVerify(exactly = 1) { this: MockKVerificationScope
        searchUseUseCase(email)
    }

    assertEquals( expected: null, result)
}
```

A continuación se muestran los resultados de los test unitarios de UserService:



RentalToolServiceTest: Implementa las pruebas relacionadas con el alquiler de herramientas.

- **wheenDeliveryTools_thenGetEmptyListTools:** Comprueba que cuando un usuario devuelve una lista de herramientas este no tenga herramientas pendientes a devolver.

```
@Test
fun whenDeliveryTools_thenGetEmptyListTools()= runBlocking { this: CoroutineScope
    val email = "user@gmail.com"
    val rentalTool1 = RentalTool(
        userMail = email,
        barcode = "000000",
        toolName = "tool_name",
        project = "project",
        location = "location",
        pickUpDate = "21/12/2023",
        dropOffDate = "",
        status = RentalToolStatus.PENDING.toString(),
        photo = "photo"
    )
    val rentalTool2 = RentalTool(
        userMail = email,
        barcode = "000001",
        toolName = "tool_name",
        project = "project",
        location = "location",
        pickUpDate = "21/12/2023",
        dropOffDate = "",
        status = RentalToolStatus.PENDING.toString(),
        photo = "photo"
    )
```

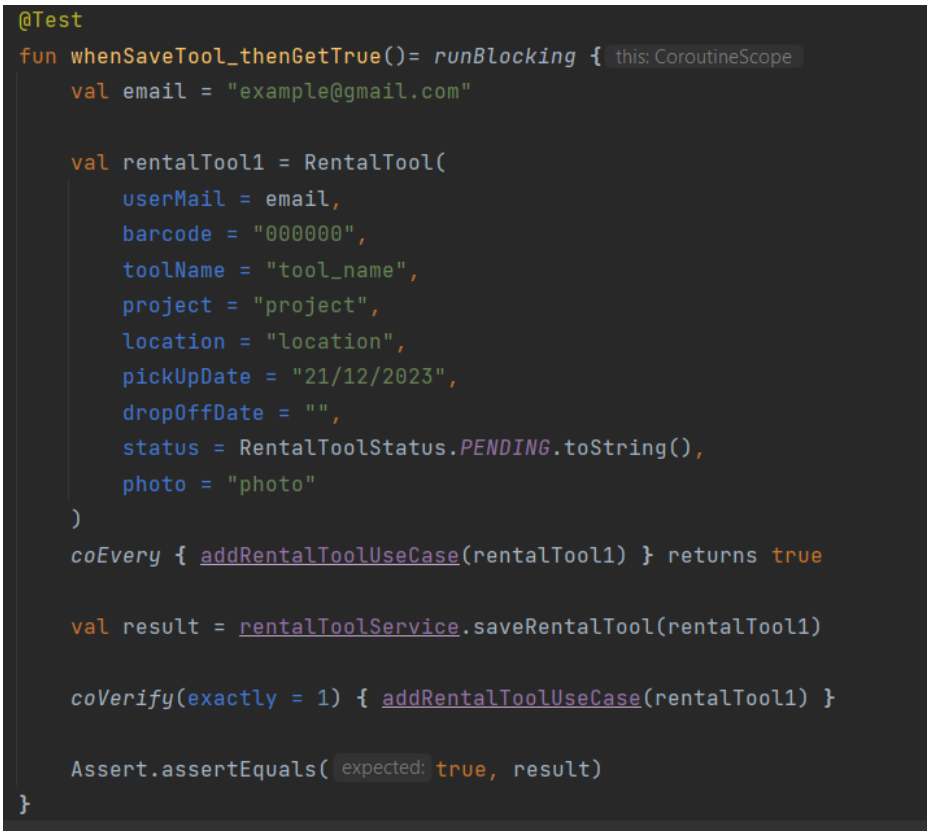
```
val rentalTool3 = RentalTool(
    userMail = email,
    barcode = "000002",
    toolName = "tool_name",
    project = "project",
    location = "location",
    pickUpDate = "21/12/2023",
    dropOffDate = "",
    status = RentalToolStatus.PENDING.toString(),
    photo = "photo"
)
val rentalTools = mutableListOf(rentalTool1,rentalTool2,rentalTool3)

coEvery { addRentalToolUseCase(any()) } returns true
coEvery { toolService.changeStatus(any()) } coAnswers {}
coEvery { getRentalHistoryUseCase(email) } returns rentalTools


rentalToolService.deliveryRentalTools(rentalTools)
val res = rentalToolService.getAllPendingTools(email)

rentalTools.forEach { coVerify(exactly = 1) { this: MockKVerificationScope
    addRentalToolUseCase(it)
}}
```

```
    rentalTools.forEach { coVerify(exactly = 1) { this: MockKVerificationScope
        toolService.changeStatus(it.barcode)
    }}

    Assert.assertEquals( expected: 0, res.size)
}
```

- **whenUserNotHaveRentalHistoryTools_thenGiveNullList**: Comprueba que cuando un usuario no tiene herramientas pendientes a devolver se obtiene una lista vacía.

```
@Test
fun whenUserNotHaveRentalHistoryTools_thenGiveNullList()= runBlocking { this:
    val email = "example@gmail.com"

    coEvery { getRentalHistoryUseCase(email) } returns mutableListOf()

    val result = rentalToolService.getRentalHistory(email)

    coVerify(exactly = 1) { getRentalHistoryUseCase(email) }

    Assert.assertEquals( expected: 0, result.size)

}
```

- **whenSaveTool_thenGetTrue**: Comprueba que cuando se guarda correctamente un registro de alquiler se devuelve un objeto booleano con valor true.



```
@Test
fun whenSaveTool_thenGetTrue()= runBlocking { this: CoroutineScope
    val email = "example@gmail.com"

    val rentalTool1 = RentalTool(
        userMail = email,
        barcode = "000000",
        toolName = "tool_name",
        project = "project",
        location = "location",
        pickUpDate = "21/12/2023",
        dropOffDate = "",
        status = RentalToolStatus.PENDING.toString(),
        photo = "photo"
    )
    coEvery { addRentalToolUseCase(rentalTool1) } returns true

    val result = rentalToolService.saveRentalTool(rentalTool1)

    coVerify(exactly = 1) { addRentalToolUseCase(rentalTool1) }

    Assert.assertEquals( expected: true, result)
}
```

- **whenToolisNotUsed_thenGetEmtpyHistoryList:** Comprueba que cuando una herramienta no se ha utilizado se devuelve una historial vacío.

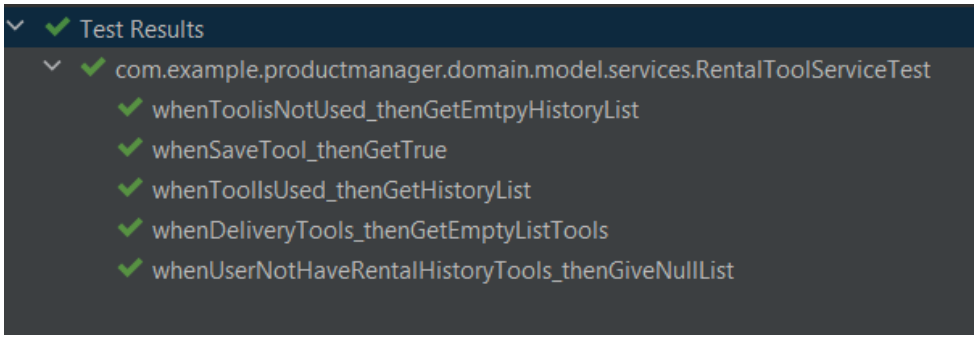
```
@Test
fun whenToolisNotUsed_thenGetEmtpyHistoryList() = runBlocking { this: CoroutineScope
    val userList = mutableListOf(
        Employee( name: "User1",  email: "user1@gmail.com",  password: "123456"),
        Employee( name: "User2",  email: "user2@gmail.com",  password: "123456")

    )


    coEvery { getAllUsersUseCase.invoke() } returns userList
    coEvery { getRentalHistoryUseCase(any()) } returns mutableListOf()

    val result = rentalToolService.getRentalToolHistory( barcode: "10101000000")

    coVerify { getRentalHistoryUseCase(any()) }

    assertEquals( expected: 0, result.size)
}
```

- **whenToolIsUsed_thenGetHistoryList**: Comprueba que cuando una herramienta ha sido utilizada por usuarios se devuelve una lista con el tamaño de los usuarios que la han utilizado.

```
@Test
fun whenToolIsUsed_thenGetHistoryList() = runBlocking { this: CoroutineScope
    val userList = mutableListOf(
        Employee( name: "User1",  email: "user1@gmail.com",  password: "123456"),
        Employee( name: "User2",  email: "user2@gmail.com",  password: "123456")
    )
    val rentalHistoryUser1 = mutableListOf(
        RentalTool(
            userMail = "user1@gmail.com",
            barcode = "10101000000",
            toolName = "tool_name",
            project = "project",
            location = "location",
            pickUpDate = "21/12/2023",
            dropOffDate = "22/12/2023",
            status = RentalToolStatus.DELIVERED.toString(),
            photo = "photo"
        )
    )
```

```
    val rentalHistoryUser2 = mutableListOf(
        RentalTool(
            userMail = "user2@gmail.com",
            barcode = "10101000000",
            toolName = "tool_name",
            project = "project",
            location = "location",
            pickUpDate = "21/12/2023",
            dropOffDate = "",
            status = RentalToolStatus.PENDING.toString(),
            photo = "photo"
        )
    )
    coEvery { getAllUsersUseCase.invoke() } returns userList
    coEvery { getRentalHistoryUseCase( email: "user1@gmail.com") } returns rentalHistoryUser1
    coEvery { getRentalHistoryUseCase( email: "user2@gmail.com") } returns rentalHistoryUser2

    val result = rentalToolService.getRentalToolHistory( barcode: "10101000000")

    coVerify(exactly = 1) { getAllUsersUseCase.invoke() }

    userList.forEach { it: Employee
        coVerify(exactly = 1) { getRentalHistoryUseCase(it.email) }
    }
    assertEquals( expected: 2, result.size)
}
```

A continuación se muestran los resultados de los test unitarios RentalToolService:



IncidencesServiceTest: Implementa las pruebas de crear y resolver incidencias

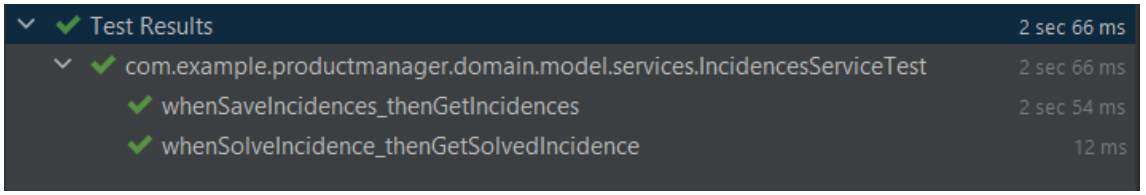
- **whenSaveIncidences_thenGetIncidences:** Comprueba que cuando se crea una incidencia se devuelve un objeto boolean con valor true.

```kotlin
@Test
fun whenSaveIncidences_thenGetIncidences() = runBlocking { this: CoroutineScop
    val incidence = Incidence(
        id="0",
        employee = "employee",
        date = "21/12/23",
        id_tool = "00001",
        tool_name = "tool",
        description = "description",
        status = IncidenceStatus.PENDING.toString()
    )
    coEvery { addIncidenceUseCase(incidence) } returns true

    val result = incidencesService.saveIncidence(incidence)

    coVerify { addIncidenceUseCase(incidence) }

    assertEquals( expected: true, result)
}
```

- **whenSolveIncidence_thenGetSolveIncidence:** Comprueba que cuando el administrador registra una incidencia como solventada esta se registre de forma correcta devolviendo un objeto booleano con valor true.

```kotlin
@Test
fun whenSolveIncidence_thenGetSolvedIncidence() = runBlocking { this: CoroutineScope
    val incidence = Incidence(
        id="0",
        employee = "employee",
        date = "21/12/23",
        id_tool = "00001",
        tool_name = "tool",
        description = "description",
        status = IncidenceStatus.PENDING.toString()
    )

    coEvery { updateIncidenceUseCase(incidence) } returns true

    val result = incidencesService.solveIncidence(incidence)

    coVerify { updateIncidenceUseCase(incidence) }

    assertEquals( expected: true, result)
}
```

A continuación se muestran los resultados de los test unitarios de IncidencesService:



Una vez realizadas las pruebas unitarias, se pueden dar por correctos los casos de uso planificados para este proyecto:

- ☑ CU01 - Registrar herramientas
- ☑ CU02 - Borrar una herramienta
- ☑ CU03 - Buscar una herramienta
- ☑ CU04 - Eliminar un usuario
- ☑ CU05 - Modificar un usuario
- ☑ CU06 - Buscar un usuario
- ☑ CU07 - Crear un Proyecto
- ☑ CU08 - Eliminar un Proyecto
- ☑ CU09 - Registrar una localización
- ☑ CU10 - Busca una localización
- ☑ CU11 - Eliminar una localización
- ☑ CU12 - Consultar incidencias
- ☑ CU13 - Resolver incidencias
- ☑ CU14 - Crear una cuenta nueva
- ☑ CU15 - Iniciar sesión
- ☑ CU16 - Alquilar una herramienta
- ☑ CU17 - Devolver una herramienta
- ☑ CU18 - Buscar una herramienta
- ☑ CU19 - Consultar el historial de préstamos
- ☑ CU20 - Crear una incidencia
- ☑ CU21 - Consultar reservas

6.2 Pruebas de instrumentación

A pesar de que en este proyecto no se han realizado las pruebas de integración, se ha creído conveniente detallar el funcionamiento de las pruebas de instrumentación realizadas con el framework Espresso.

Las pruebas unitarias se realizan con el entorno de ejecución de android para evaluar las interfaces de usuario así como su comportamiento. A diferencia de las pruebas unitarias, las pruebas de instrumentación no son muy comunes de realizar, pues suponen un mayor coste de tiempo y recursos

Las pruebas realizadas con espresso, permiten realizar test sencillos gracias al uso de los *matchers, viewActions* y *ViewAssertions.* Con ello, se puede comprobar toda la funcionalidad de la interfaz de usuario, como por ejemplo comparar un texto con el valor introducido en un TextEdit, seleccionar elementos de una lista y accionar botones entre otras cosas.

6.3 Pruebas de rendimiento

Las pruebas de rendimiento tienen como objetivo medir la capacidad de respuesta, estabilidad y uso de recursos de la aplicación en las distintas condiciones que puede encontrarse el dispositivo móvil. Este tipo de pruebas aportan a la aplicación una mejora de asignación de recursos y garantiza que la aplicación se ejecuta sin problemas en los dispositivos y redes.

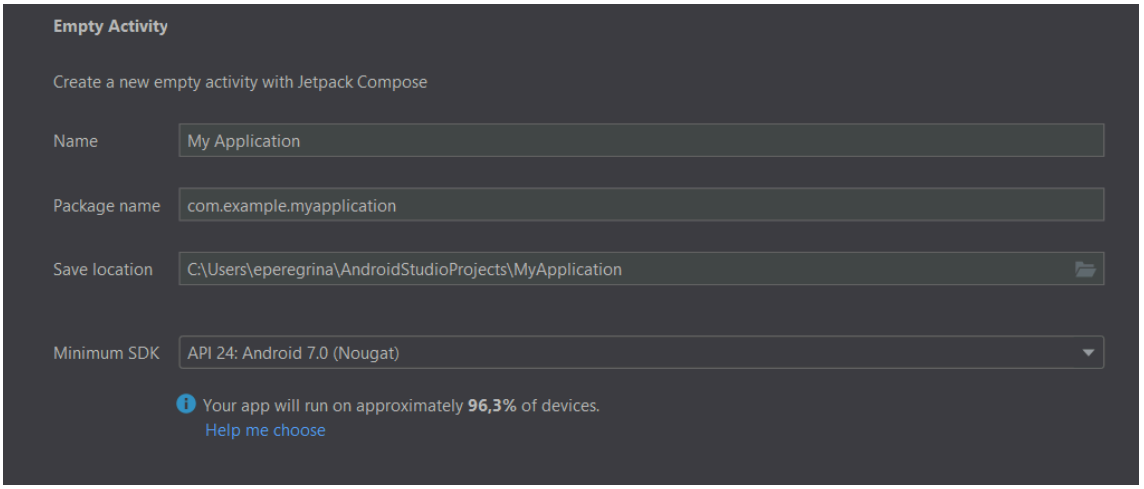
Puesto que esta aplicación ha sido desarrollada para cubrir las necesidades de un sector específico , no se espera un uso intensivo en cuanto a rendimiento. La mayoría de operaciones de consultas y registros se realizan por los empleados a través del servicio Firestore ubicado en la nube. El administrador, una vez que haya registrado todas las herramientas, ubicaciones y proyectos solamente gestiona las incidencias, por lo que se reduce el número de operaciones.

Por lo tanto, se considera no realizar las pruebas de rendimiento en esta versión de aplicación.

6.4 Pruebas de compatibilidad

Esta aplicación ha sido probada en el dispositivo móvil Samsung Galaxy S6 (Nougat, Android 7.0 API 24) y Samsung Galaxy S10e (API 35) y con el emulador Pixel 3a API30.

A pesar de las recomendaciones indicadas por entorno de programación para elegir la versión mínima SDK API nivel 24, en algunos casos se ha tenido que cambiar la versión mínima del SDK al nivel 26, pues hay algunos métodos que requieren una versión superior. Tras ello, se ha comprobado todas las funcionalidades tanto en el dispositivos virtuales como en fisicos y se han corregido algunos errores de coroutinas que no se habían tenido en cuenta al probar en el emulador.
A continuación se muestra una captura de pantalla de la recomendación mínima SDK de Android Studio en el momento de crear un proyecto:



7. Conclusiones

El desarrollo de la aplicación ProductManager ha sido un reto en todos los aspectos, puesto que se han utilizado herramientas desconocidas, como Kotlin, Firebase, patrón de diseño MVVM, Arquitectura clean, manejo de coroutinas e inyección de dependencias. Sin embargo, algunas tecnologías

implementadas, como el lenguaje oficial de Android, Kotlin, tuvo una repercusión menor al tener similitudes con el lenguaje aprendido en el grado: Java.

A pesar de las dificultades encontradas, se ha conseguido cumplir con todos los objetivos establecidos al principio del proyecto. Con ello, se ha desarrollado una aplicación totalmente funcional a falta de algunas mejoras en algunas interfaces de usuario.

En cuanto a la planificación del proyecto, se propuso seguir una metodología en cascada, puesto que los objetivos estaban claramente definidos. Pese a ello, el tiempo propuesto fue muy optimista, ya que otros factores como el desconocimiento e inexperiencia de algunas tecnologías, supuso un aumento de horas estimadas en la parte de implementación.

Finalmente, decir que el desarrollo de este proyecto ha sido un tema de gran interés y se pretende seguir investigando para futuras mejoras, puesto que han habido conceptos que no se han cubierto, como por ejemplo las pruebas de instrumentación o la gestión de errores. También con más tiempo y experiencia la organización de las interfaces de usuario se podrían mejorar y se podría aumentar la seguridad de los accesos a métodos de administrador, ya que en desde la misma aplicación se puede acceder a los dos perfiles. Este problema se podría solucionar con añadiendo condiciones en los métodos o también separando en dos aplicaciones los paneles de administrador y la de usuario.

8. Glosario

Firebase: Plataforma de desarrollo multiplataforma alojada en la nube que ofrece un conjunto de herramientas para crear y sincronizar proyectos.
Android: Sistema operativo desarrollado por Google que se utiliza en los dispositivos móviles.

Kotlin: Lenguaje de programación de código abierto desarrollado por JetBrains que se utiliza para programar aplicaciones Android.

IDE: Entorno de desarrollo integrado que permite desarrollar código de software de manera eficiente

SQL: Lenguaje que programación se utiliza para administrar bases de datos.

matchers: Herramienta de espresso que ayuda a identificar y actuar sobre elementos de la interfaz de usuario.
viewActions: Herramienta de espresso que proporciona metodos a realizar sobre elementos de la interfaz de usuario.
ViewAssertions: Herramienta de espresso para realizar aserciones en los estados de las interfaces de usuario.

9. Bibliografía

app - Biblioteca Virtual
https://bibliotecavirtual.diba.cat/es/app
[Consultada el 01/10/2023]

myur! - ¡La app para alquilar cosas!
https://www.myur.app/

[Consultada el 01/10/2023]

Firebase | Google's Mobile and Web App Development Platform
https://firebase.google.com/?hl=es
[Consultada el 04/10/2023]

Diseño Centrado en el Usuario para dispositivos móviles
https://xwiki.recursos.uoc.edu/wiki/matm1202es/
[Consultada el 14/10/2023]

Herramienta de diseño esquemático Lucidchart
https://lucid.app/lucidchart/
[Consultada el 20/10/2023]

Herramienta de diseño de prototipo Marvel App
https://marvelapp.com/project/6871280
[Consultada el 24/10/2023]

Arquitecturas Android en 2020
https://devexpert.io/arquitecturas-android/
[Consultada el 26/10/2023]

Clean Architecture | Deloitte España | Tecnología
https://www2.deloitte.com/es/es/pages/technology/articles/clean-architecture.html
[Consultada el 27/10/2023]

Herramientas para desarrolladores de aplicaciones para dispositivos Android
https://developer.android.com/?hl=es-419
[Consultada el 15/10/23]

MVVM Architecture components: Una guia para los amantes de MVP
https://devexpert.io/mvvm-vs-mvp/
[Consultada el 17/10/23]

Cloud Firestore

https://firebase.google.com/?hl=es-419
[Consultada el 17/10/23]

Principios SOLID: Qué son, cuáles, y qué beneficios aporta usarlos
https://devexpert.io/principios-solid/
[Consultada el 27/10/23]

Coroutines con Kotlin — Introducción
https://medium.com/kotlin-en-android/coroutines-con-kotlin-introducci%C3%B3n-a68f5eeee6a8
[Consultada el 29/10/23]

Using Glide with Kotlin
https://medium.com/@vlonjatgashi/using-glide-with-kotlin-5e345b557547
[Consultada el 29/11/23]

Glide repository github
https://github.com/bumptech/glide
[Consultada el 29/11/23]

Testing en Android - Test unitarios
https://cursokotlin.com/testing-en-android-test-unitarios/
[Consultada el 25/12/23]

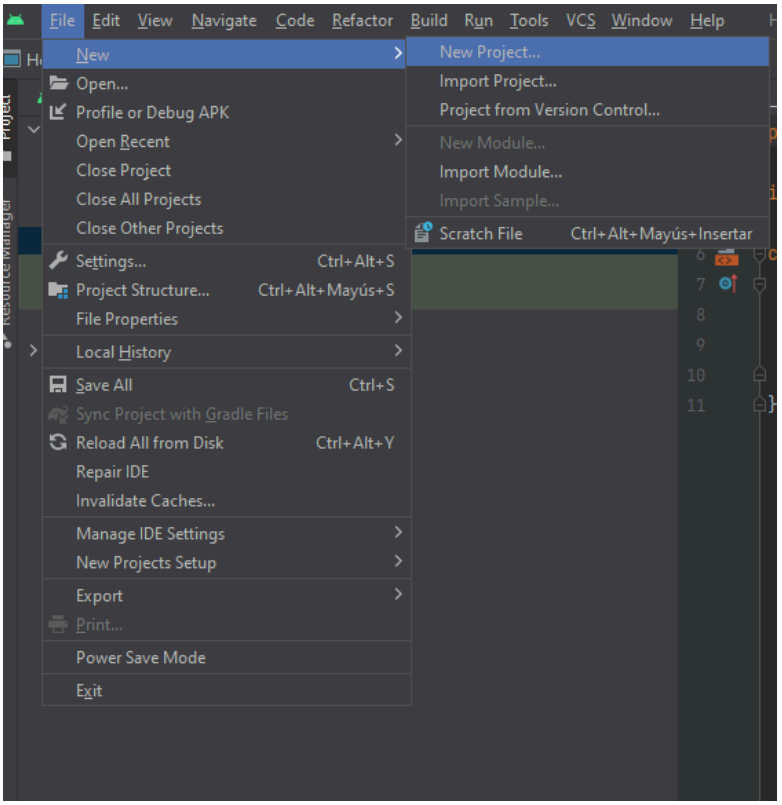
Testing en Android: cómo hacer test unitarios
https://www.paradigmadigital.com/dev/testing-android-tests-unitarios/
[Consultada el 25/12/23]

10. Anexos

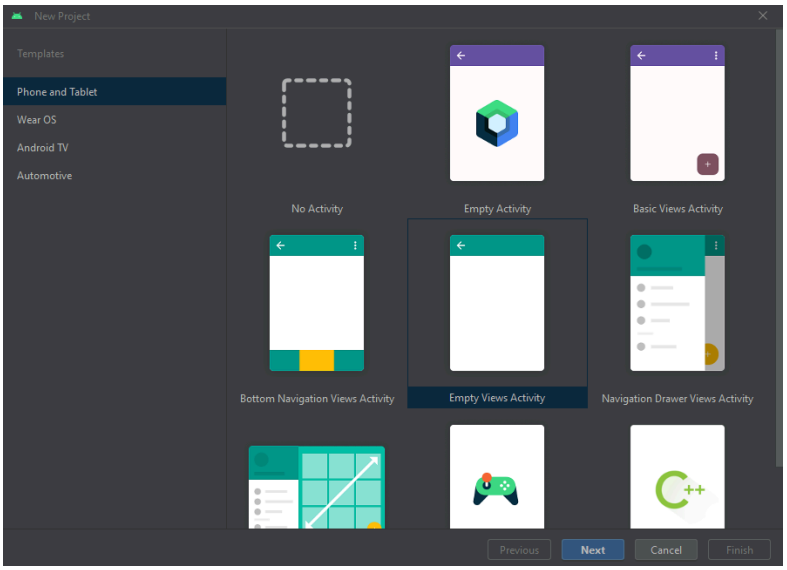
Anexo I. PEC1 - Plan de trabajo. Producto 2 Aplicación

En el segundo apartado de la primera entrega nos piden una aplicación tipo “Hola mundo”. Para ello necesitaremos tener instalado el entorno de programación Android Studio.

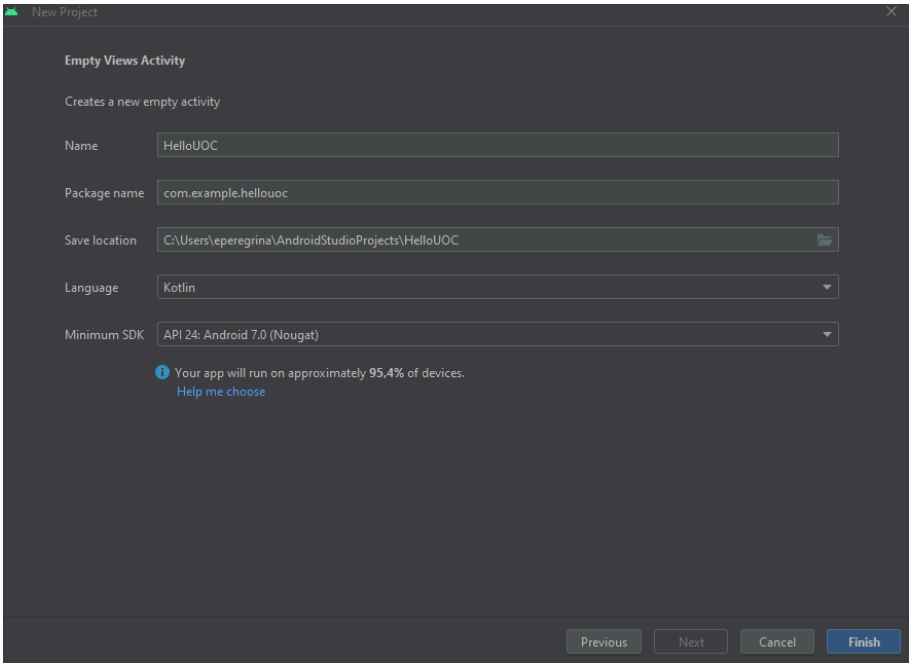
Una vez instalado el IDE, crearemos un nuevo proyecto haciendo click en el primer apartado llamado “File” y “New Project”, ubicado en el menú superior.



Posteriormente seleccionaremos el tipo de vista a crear. En nuestro caso para este ejemplo elegiremos una “Empty Views Activity”.



A continuación, definiremos el nombre del proyecto, el lenguaje de programación en Kotlin y estableceremos la versión mínima del SDK en 7.0, pues nos indica que el 95.4% de los dispositivos usan esta versión.



Una vez creado el proyecto, podremos ver su jerarquía en el menú lateral izquierdo. De momento, solo definiremos las siguientes carpetas:

- com.example.hellouoc.MainActivity.kt: Actividad principal donde se definira el comportamiento de la vista.
- drawable: Carpeta donde se almacenan las imagenes e iconos que se usen en la aplicación.

- layout: Interfaces gráficas de la aplicación.
- Gradle Script: Encontraremos el build.gradle (Module:app). En este archivo se definen las dependencias de librerías que se utilicen en el desarrollo de la app.

El primer paso para crear la aplicación “Hello UOC” será diseñar una vista. Para ello nos dirigiremos a la carpeta layout y abriremos el archivo “activity_main.xml”.

Como esta aplicación no tendrá muchos componentes, definiremos el comportamiento del diseño gráfico en forma lineal utilizando un LinearLayout. Lo configuraremos para que organice los objetos de forma vertical.

A continuación, introduciremos algunos objetos para darle forma al proyecto:

- EditText: Introduciremos una caja de texto para que la aplicación pueda leer lo que el usuario haya puesto.
- Button: Botón que nos permitirá iniciar el evento definido.
- TextView: Texto que mostrará lo que hayamos introducido en el EditText.

Es importante nombrar que cada objeto tiene definida una ID. Esto nos ayudará posteriormente cuando definamos el comportamiento de cada uno de ellos.

Realizada la interfaz gráfica, pasamos a definir la gestión de los eventos en las actividades, y para ello abrimos la actividad MainActivity.

Como hemos dicho, la actividad va a definir el comportamiento de los eventos. En esta entrega se ha decidido, que además de hacer el mensaje Hello world por pantalla, también se va a implementar un asistente de voz que lea en voz alta el texto introducido por el usuario. Para ello, deberemos heredar de la clase TextToSpeech e implementar sus métodos, en este caso solo el onInit(status: Int).

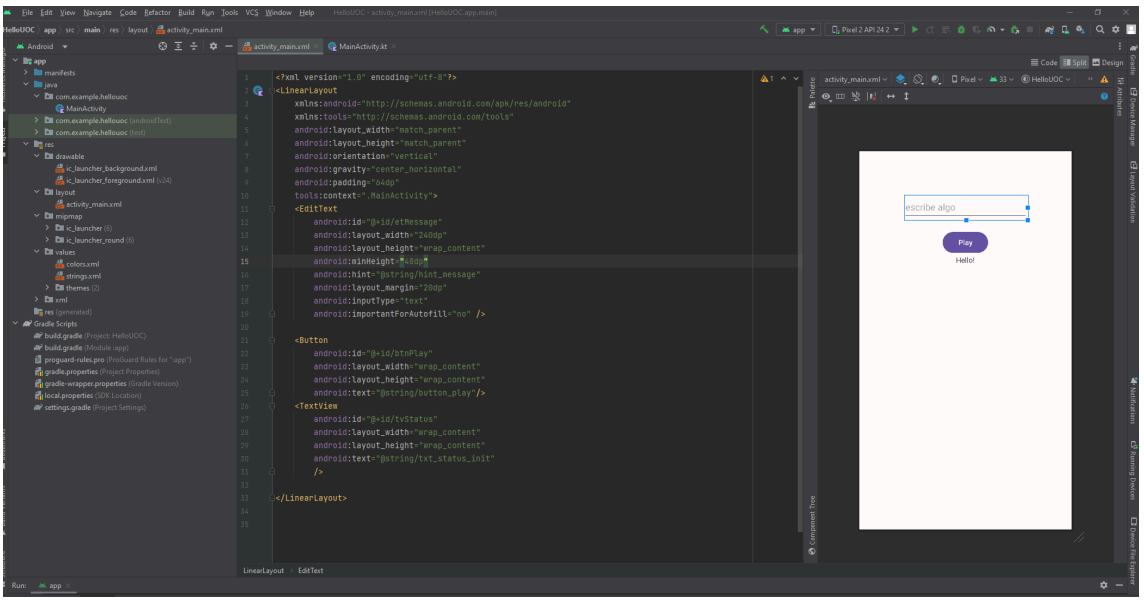
En el método onInit, se realizan las comprobaciones de compatibilidad con el dispositivo para ver si puede realizar dichas funcionalidades. Por lo tanto, si el dispositivo no es compatible observaremos que el texto indica “No disponible). En caso de que el dispositivo sea compatible, se realizan las configuraciones del lenguaje en el que va a hablar el/la asistente.

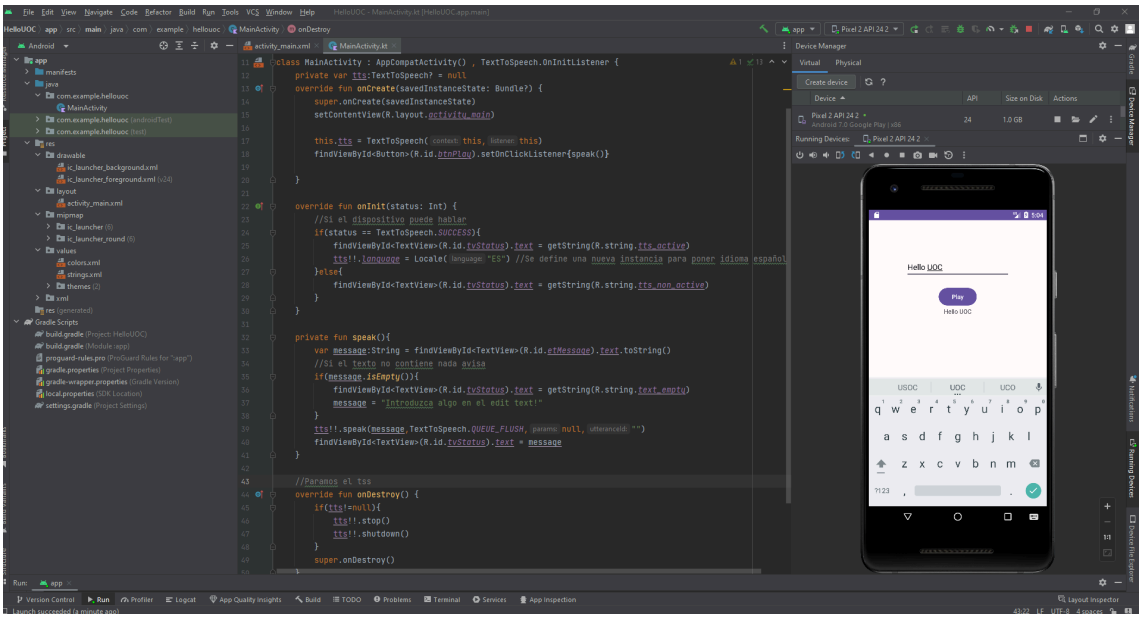
Posteriormente, se creará una instancia de la clase TextToSpeech para poder acceder a sus métodos más adelante y se creará un listener para el botón de Play. De esta manera, si el usuario pulsa el botón, se creará un evento y se llamará al método introducido, en este caso el método speak().

Este método obtendrá el texto introducido por el usuario y, si no está vacío, cambiará el texto de la pantalla y reproducirá el mensaje introducido.

Finalmente, disminuir el consumo de recursos, sobre escribiremos el método onDestroy para parar el asistente de voz

A continuación se muestran las capturas del código y de la aplicación en funcionamiento:





Anexo II. Cuestionario de evaluación del prototipo e interfaz de navegación con Marvel App

Cuestionario de Evaluación:
https://forms.gle/wD8ZnpDZo2eb4ZrEA

Interfaz de navegación
https://marvelapp.com/prototype/102j9a1b

: