
***Facultad***
***de***
***Ciencias***

# DESARROLLO DE UNA APLICACIÓN PARA DISPOSITIVOS ANDROID PARA COMERCIALES DE EMPRESA DE BAZAR Y HOSTELERÍA

## (Development of an application for Android devices for sales representatives of bazaar and catering company)

**Trabajo de Fin de Grado**
**para acceder al**

**GRADO EN INGENIERÍA INFORMÁTICA**

**Autor: Adrián García Cubas**

**Director: Juan María Rivas Concepción**

**Septiembre - 2022**

# Resumen

La digitalización de las empresas está llevando a reducir costes en todas sus áreas. Una de las áreas donde se puede automatizar y digitalizar el trabajo es la parte administrativa. Hoy en día en las empresas, todas las personas indistintamente de en que área trabaje, debería poder realizar o ayudar en otras áreas que no le corresponden.

Esto lo llamaríamos digitalización cruzada. Por ejemplo un comercial puede enviar facturas o reclamar deudas a un cliente (labor puramente administrativa) y un administrativo puede controlar, enviar o revisar presupuestos realizados. Esto es una reducción considerable del tiempo empleado por parte de la empresa ya que se ahorran todas las llamadas para consultar o aportar datos de un determinado cliente.

Este proyecto tiene como objetivo reducir el tiempo gastado en la empresa en la comunicación entre el área comercial y el área administrativa mediante una aplicación Android que permita al comercial consultar y gestionar datos de sus clientes en tiempo real y al repartidor saber y organizar sus rutas e informar al comercial en que situación están los pedidos de sus clientes.

**Palabras clave:** Aplicación móvil, Android, Digitalización de empresas.

# Abstract

The digitalization of companies is leading to cost reductions in all areas. One of the areas where work can be automated and digitized is the administrative area. Nowadays in companies, everyone, regardless of the area in which they work, should be able to perform or help in other areas that do not correspond to them.

This is what we would call cross-digitalization. For example, a salesperson can send invoices or claim debts to a client (purely administrative work) and an administrative person can control or send budgets or review them. This is a considerable reduction of the time spent by the company since all the calls to consult or provide data of a particular customer are saved.

This project aims to reduce the time spent in the company in the communication between the commercial area and the administrative area through an Android application that allows the salesperson to consult and manage customer data in real time.



**Keywords:** Mobile application, Android, digitalization of companies

# Agradecimientos

Con este trabajo se cierra la mejor etapa de mi vida, la mejor gracias a toda la gente que he conocido durante estos 4 años que me han dado un millón de experiencias nuevas y me han permitido crecer personal y academicamente.

Me ha enseñado lo que es la independencia y la responsabilidad durante mi primer año en la Universidad de Alicante, allí aprendí que puedes encontrar una familia en muy poco tiempo y que perdura con la distancia.

Agradecer a mis amigos de la Universidad de Cantabria, en especial a "10-29"por haberme acompañado durante 3 años y por tantos días de Discord.

Agradecer a *Almacenes Piscis* y en especial a los padres de mi novia por darme la oportunidad de poder realizar este TFG en su empresa y estar ahí para prestarme todo el apoyo y cafés que necesite.

Agradecer a mi familia por estar siempre a mi lado apoyándome y cuidándome.

Agradecer en especial a mi novia Andrea por aguantarme cuando ha habido momentos difíciles y por siempre estar ahí pase lo que pase.

Gracias a todos los profesores por haberme enseñado todo lo que podían.

Muchas gracias a Juan María Rivas por haber sido mi director de este TFG y haberme ayudado siempre que ha podido.

# Índice

# Índice de tablas

# Índice de figuras

# 1. Introducción

Esta sección consta de tres apartados. En el primer apartado se va a mostrar las motivaciones para la realización de este proyecto y el contexto tecnológico del mundo de las empresas para poder reducir tiempos de comunicación entre áreas. En el segundo apartado se describirán los objetivos que se buscan obtener mediante la realización de este proyecto. Por último, en el tercer apartado se mostrará la estructuración y organización de este documento.

## 1.1. Motivación y contexto tecnológico

En un mundo donde cada vez las cosas se obtienen de manera casi instantánea y donde el tiempo es más valioso que el dinero, el trabajo administrativo de una empresa debe de ser lo más rápido posible para poder ser competitivas en un entorno empresarial. Hoy en día son muchas las PYMES (*Pequeñas y Medianas Empresas*) que todavía no se han adentrado en el mundo de los SI (Sistema de Información, sirve para gestionar y almacenar los datos que componen a una empresa), y sus maneras de tratar toda la información no están siendo digitalizadas. Por lo tanto el tiempo que se tarda en procesar y/o administrar la información es exponencialmente más elevado que en una empresa digitalizada. La diferencia de tiempos entre estos tipos de empresas no es solo en la comunicación entre la empresa y el cliente, sino también entre los propios trabajadores, los cuales tienen que preguntarse unos a otros sobre ciertas cuestiones realizadas por los clientes, ya que no tienen un SI que pueda satisfacer sus preguntas.

*Almacenes Piscis* es una empresa familiar dedicada a la venta de productos de bazar y hostelería. La empresa se divide en varias áreas, una de ellas es el área comercial que es a la que se va a dedicar gran parte de de la aplicación. El problema actual radica en que cuándo un cliente solicita un documento a *Almacenes Piscis*, el comercial se tiene que poner en contacto con el área de administración, ya que a la base de datos de la empresa solo se puede acceder desde la oficina, y esperar a que se encuentre y se mande el documento solicitado al comercial para que este se lo pueda pasar al cliente. Esta cadena de sucesos es lenta comparada con las opciones que te presenta hoy en día la tecnología. Con el desarrollo de la aplicación *PiscisOutOffice* se pretende que el comercial sea capaz por si mismo de poder acceder a esos documentos desde su dispositivo móvil de una manera mucho mas rápida.

## 1.2. Objetivos

Una vez expuesto la motivación y el contexto, el objetivo de la aplicación PiscisOutOffice es el ahorro de tiempo tanto para el cliente como para los trabajadores de la oficina de la empresa, ofreciendo una respuesta rápida y eficaz a las necesidades requeridas en cada momento. Cabe recalcar que PiscisOutOffice no es una herramienta enfocada para la venta de productos, si no para la organización y la administración. Esto se lleva a cabo a través de una serie de interfaces de usuario intuitivas y sencillas de utilizar que pueda permitir la

consulta de diferentes recursos de los clientes, consulta de artículos y por último consulta, edición y creación de la propia agenda de los usuarios.

Si hablamos de los repartidores, las visitas se refieren a los lugares donde los clientes solicitan que les sea entregada la mercancia, es decir, como un reparto.

En cuanto a los comerciales, las visitas se refieren a las citas que tenga con los diferentes clientes o clientes potenciales.

Para un mayor entendimiento, se va a realizar:

- Menu de consulta de articulos en la base de datos de la empresa a través de varios parámetros de búsqueda.

- Acceso a varios parametros de los clientes tales como sus reservas, facturas y presupuestos por parte de un comercial.

- Acceso a consulta, modificación y creación de las visitas de un comercial.

- Acceso a consulta de las visitas de un repartidor.

- Creación de un servicio de geolocalización para seguimiento de las rutas trazadas por los trabajadores en horario laboral.

## 1.3. Organización y estructura del documento

Esta memoria consta de 7 secciones a través de las cuales se describirá el desarrollo de una aplicacion Android para la empresa *Almacenes Piscis*.

Los contenidos de esta memoria son los siguientes:

- **Sección 2:** se explica las tecnologías y herramientas utilizadas durante el proyecto, así como la metodología de desarrollo utilizada en el mismo.

- **Sección 3:** se recoge la especificación de requisitos demandada por la empresa, tanto los funcionales como los no funcionales. Además, se mostrarán los casos de uso de la aplicación y se detallarán en profundidad aquellos que se consideren de una complejidad superior a los demás.

- **Sección 4:** se muestran y describen los diagramas de casos de uso de la aplicación. Además, se detallarán los de mayor importancia o complejidad.

- **Sección 5:** se describe la toma de decisiones de diseño que se ha llevado a cabo previamente al comienzo de la implementación de la aplicación.

- **Sección 6:** se explica el desarrollo de la implementación de aplicación y cómo se han usado las diferentes tecnologías en el proyecto.

- **Sección 7:** se muestra y explica las pruebas que se han llevado a cabo para la comprobación del funcionamiento de la aplicación.

- **Sección 8:** se comentan las conclusiones finales tras la realización del proyecto y los próximos trabajos en referencia a la aplicación.

# 2. Tecnologías, herramientas y metodologías

Esta sección consta de tres apartados. El primer apartado y segundo se mostrarán y explicarán las tecnologías y herramientas utilizadas para la realización de este proyecto. En la tercera y última sección se detallará la metodología de desarrollo software que se ha seguido para la realización de la aplicación.

## 2.1. Tecnologías

Durante la realización del proyecto, se han utilizado las siguientes tecnologías:

### 2.1.1. Android

Android [1] es un sistema operativo móvil basado en el núcleo Linux y otros software de código abierto. Android es actualmente el sistema operativo mas utilizado en dispositivos móviles gracias a ser libre, gratuito y multiplataforma.

En cuanto a su arquitectura, Android se compone de cuatro capas, en los cuales el nivel inferior ofrece una serie de características y servicios a la capa inmediatamente superior. Las cuatro capas ordenadas de más bajo nivel más alto nivel son las siguientes:

- **Núcleo Linux:** Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

- **Librerias:** Android incluye un conjunto de librerías de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android. Algunas son: System C library (implementación biblioteca C estándar), bibliotecas de medios, bibliotecas de gráficos, 3D y SQLite, entre otras. Dentro de esta capa se encuentra el Runtime de Android.

- **Marco de trabajo de aplicaciones:** los desarrolladores tienen acceso completo a las mismas API del entorno de trabajo usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.

- **Aplicaciones:** las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros.

### 2.1.2. SQLite

SQLite [2] es una biblioteca en lenguaje C que implementa un motor de base de datos SQL pequeño, rápido, autónomo, de alta fiabilidad y con todas las funciones. Es el motor de base de datos más utilizado en el mundo. SQLite está integrado en todos los teléfonos móviles y en la mayoría de los ordenadores, y viene incluido en innumerables aplicaciones que la gente utiliza a diario.

Su formato de archivo es estable, multiplataforma y compatible con versiones anteriores. Los archivos de bases de datos SQLite se utilizan habitualmente como contenedores para transferir contenidos ricos entre sistemas y como formato de archivo de datos a largo plazo.

El código fuente de SQLite es de dominio público y su uso es gratuito para todo el mundo.

### 2.1.3. Room

La biblioteca de persistencias de Room [3] brinda una capa de abstracción para SQLite que permite acceder a la base de datos local (en el dispositivo Android) sin problemas y, al mismo tiempo, aprovechar toda su potencia.

### 2.1.4. Microsoft SQL Server

Microsoft SQL Server [4] es un sistema de gestión de base de datos relacional, desarrollado por la empresa Microsoft.

El lenguaje de desarrollo utilizado es Transact-SQL (TSQL), una implementación del estándar ANSI del lenguaje SQL, utilizado para manipular y recuperar datos (DML), crear tablas y definir relaciones entre ellas (DDL).

### 2.1.5. Git

Git [5] es un sistema de control de versiones distribuido gratuito y de código abierto diseñado para manejar todo, desde proyectos pequeños hasta proyectos muy grandes con velocidad y eficiencia.

## 2.2. Herramientas

### 2.2.1. Android Studio

Android Studio [6] es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de apps para Android y está basado en IntelliJ IDEA. Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece incluso más funciones que aumentan tu productividad cuando desarrollas apps para Android. Permite ejecutar el codigo mediante un emulador de un dispositivo android en tiempo real, plantillas para crear diseños comunes de Android y otros componentes, herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones y otros problemas y una consola de desarrollador con consejos de optimización, ayuda para la traducción y estadísticas de uso entre otras características

### 2.2.2. Microsoft SQL Server Management Studio

SQL Server Management Studio [7] es un entorno integrado para administrar cualquier infraestructura de SQL, desde SQL Server a Azure SQL Database. SSMS proporciona herramientas para configurar, supervisar y administrar instancias de SQL Server y bases de datos.

### 2.2.3. MagicDraw

MagicDraw [8] es una herramienta visual de modelado UML con soporte de colaboración en equipo. Diseñada para analistas de negocio, analistas de software, programadores e ingenieros de control de calidad. Esta herramienta de desarrollo dinámica y versátil facilita el análisis y el diseño de sistemas orientados a objetos y bases de datos.

### 2.2.4. DB Browser for SQLite

DB Browser for SQLite [9] es una herramienta de código abierto, visual, y de alta calidad para crear, diseñar y editar archivos de bases de datos compatibles con SQLite.

### 2.2.5. Github

Github [10] es una web y un servicio en la nube creado para alojar el código de los proyectos de cualquier desarrollador. Esta web utiliza el sistema de versiones Git.

### 2.2.6. Sonarcloud

Sonarcloud [11] es un servicio de análisis de código diseñado para detectar problemas de calidad de código.

## 2.3. Metodologías de desarrollo software

Una metodología de desarrollo software es un conjunto de técnicas que se utilizan para planificar el desarrollo de soluciones de software.

Para este proyecto se ha optado por la metodología de desarrollo software iterativa e incremental. Esta consiste en dividir el desarrollo del producto en varias etapas, o iteraciones, hasta llegar a una versión final. En otras palabras, se desarrolla una implementación inicial y posteriormente se trabaja sobre ella aportando nuevas funcionalidades.

Con esta metodología de trabajo se consigue retroalimentación por parte del cliente sobre una parte concreta de la funcionalidad, lo que implica que se puede modificar el trabajo realizado durante la iteración con un relativo bajo coste. Para poder tener un mejor control de las versiones del proyecto, se ha utilizado Git para llevar un correcto versionado de la aplicación y así poder deshacer cualquier cambio no deseado, o volver a una versión anterior

de la misma por cualquier motivo.

### 2.3.1. Planificación

En esta sección se va a explicar las distintas fases por las que pasó este proyecto.

En la primera fase, se tuvieron diversas reuniones con el cliente para la realización de un análisis de los requisitos que debiera de tener la aplicación, asi como una investigación sobre las tecnologías y herramientas a utilizar, y sobre el mejor diseño arquitectónico para el proyecto.
Una vez finalizada la primera fase, entramos en las iteraciones del metodo de desarrollo. Esta fase de iteraciones estaría comprendida por seis iteraciones:

- **Iteración 1:** Esta iteración se centró en el diseño e implementación de la autenticación del usuario, asi como de conectar la aplicación con el servidor SQL mediante un driver, para verificar la autenticación.

- **Iteración 2:** Esta iteración estuvo formada por la creación del menú principal del comercial, asi como del desarrollo de la funcionalidad *Buscar articulos*, que permite al comercial buscar cualquier articulo que se encuentre en la base de datos y mostrarle los datos pedidos por el cliente.

- **Iteración 3:** Esta iteración estuvo formada por la implementación de la funcionalidad *Agenda Clientes* y de las tres funcionalidades que se asocian a cada cliente. Estas son: *Mostrar sus facturas, Mostrar sus reservas* y *Mostrar sus presupuestos*. Una vez se seleccione el cliente, el comercial puede obtener los datos que necesite en ese momento.

- **Iteración 4:** Esta iteración se centró en las visitas del comercial. Se implementaron tres funcionalidades distintas: *Consultar sus visitas, Modificar las visitas y Crear una visita*.

- **Iteración 5:** Esta iteración se centró en el diseño e implementación de las funcionalidades para el repartidor. Por el momento, solo puede visualizar y modificar sus visitas.

- **Iteración 6:** En esta iteración se creó un servicio de localización para poder obtener las coordenadas de los trabajadores mientras estén en horario de trabajo y poder trazar las rutas recorridas durante la jornada laboral para su posterior análisis.

# 3. Análisis y especificación de requisitos

En este apartado se recogerán y describirán tanto los requisitos funcionales como los no funcionales, divididos en dos apartados individuales, necesarios para la creación de la aplicación.

La captura de requisitos ha sido realizada mediante varias reuniones con el jefe de la empresa.

## 3.1. Actores del sistema

Los actores del sistema son los diferentes tipos de usuario que van a hacer uso de la aplicación de diferentes maneras. En la tabla 1 se nombran y describen los actores de *PiscisOutOffice*.

| Actor | Descripción |
|---|---|
| Usuario sin registrar | Este actor lo único que puede hacer es registrarse mediante una pantalla de login. |
| Comercial | Este actor puede utilizar todas las funciones del sistema. |
| Repartidor | Este actor solo puede consultar y editar su agenda de visitas. |

Tabla 1: Actores del sistema

## 3.2. Requisitos funcionales

Un requisito funcional [12] es un requisito que describe las funciones que el software debe proporcionar.

A continuación se muestran en la tabla 2 todos los requisitos funcionales de esta aplicación:

## 3.3. Requisitos no funcionales

Un requisito no funcional [13] es una declaración de una parte de la funcionalidad requerida o un comportamiento que un sistema mostrará en condiciones específicas. En la tabla 3 se muestran todos los requisitos no funcionales de la aplicación.

| Identificador | Descripción |
|---|---|
| RF 1 | El usuario debe poder autenticarse en la aplicación. |
| RF 2 | El comercial debe poder consultar su agenda de visitas. |
| RF 3 | El comercial debe poder editar su agenda de visitas. |
| RF 4 | El comercial debe poder acceder a su agenda de clientes. |
| RF 5 | El comercial debe poder tener acceso a parámetros de facturación de sus clientes. |
| RF 6 | El comercial debe poder descargarse facturas en su dispositivo. |
| RF 7 | El comercial debe tener acceso a presupuestos ya emitidos y no confirmados. |
| RF 8 | El comercial debe poder acceder a las reservas activas de sus clientes. |
| RF 9 | El comercial debe poder acceder a todos los artículos de la base de datos. |
| RF 10 | El repartidor debe poder consultar su agenda de visitas. |
| RF 11 | El repartidor debe poder editar su agenda de visitas. |
| RF 12 | La aplicación debe poder contar con un servicio de geolocalización. |

Tabla 2: Requisitos funcionales del sistema

| Identificador | Descripción |
|---|---|
| RNF 1 | La aplicación debe cumplir con la Ley de Protección de datos "Ley orgánica 3/2018". |
| RNF 2 | La aplicación debe realizarse para dispositivos Android con la versión del sistema operativo Android 10 o superior. |
| RNF 3 | La aplicación debe ser intuitiva para poder ser utilizada de manera eficaz, eficiente y satisfactoria. |
| RNF 4 | La aplicación debe ser escalable para poder añadir funcionalidades fácilmente en un futuro si se requiriere. |

Tabla 3: Requisitos no funcionales del sistema

# 4. Casos de uso

Un diagrama de casos de uso permite poder tener una visión general del sistema, y mostrar las diferentes acciones que pueden llevar a cabo los diferentes usuarios dentro del sistema.

A continuación, se van a mostrar y describir los diferentes diagramas de casos de uso creados para un mejor entendimiento y representación de la aplicación. En la figura 1 se puede observar el diagrama de casos de uso más global, donde se puede observar las funcionalidades a más alto nivel que ofrece la aplicación, las cuales son: *Iniciar Sesión, Información Clientes, Información Articulos y Gestión Visitas.*

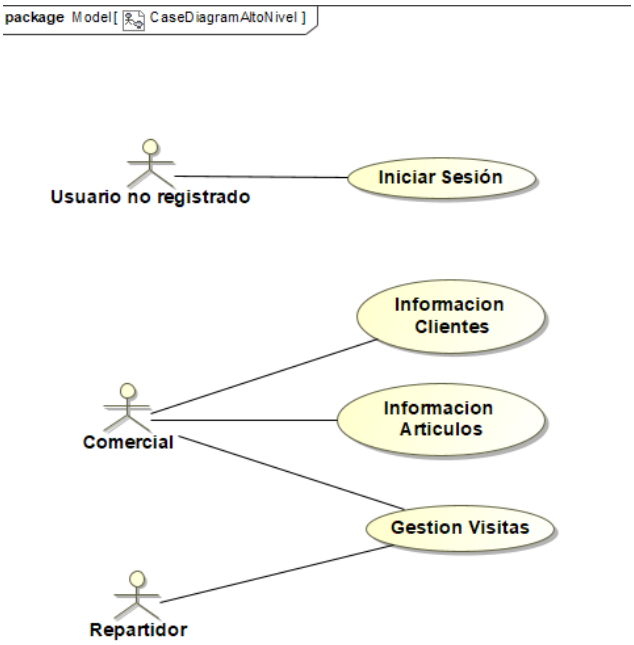


Figura 1: Diagrama de casos de uso de alto nivel

Una vez diseñado el diagrama de más alto nivel, ahora diseñamos tres diagramas más con mayor nivel de detalle en los cuales desarrollamos los casos de uso mostrados en la figura 1. Los tres diagramas pueden verse en las figuras 2, 3, 4.

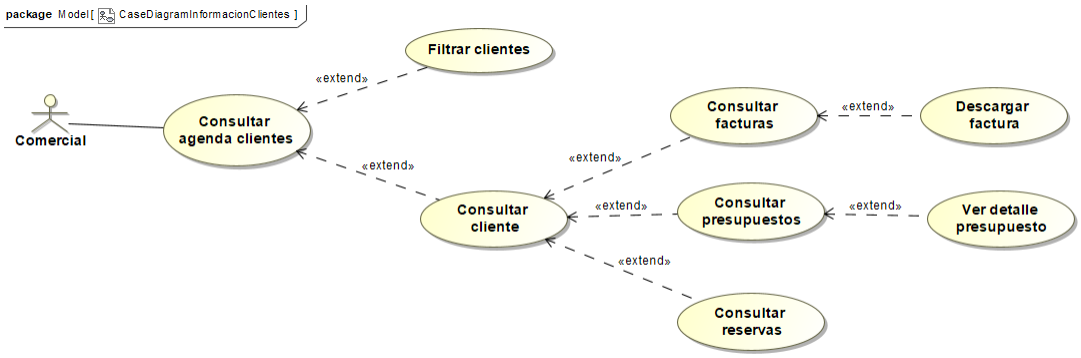


Figura 2: Diagrama de casos de uso de Informacion Clientes

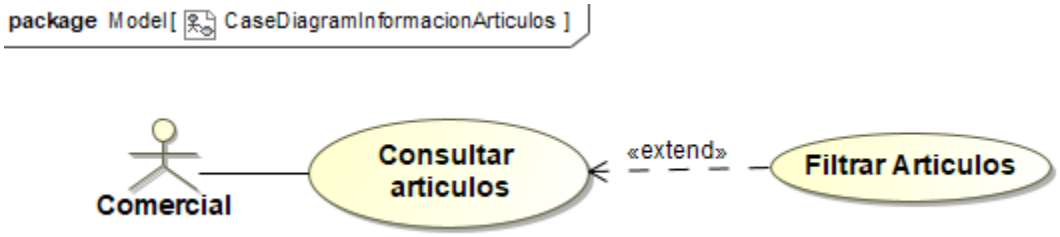


Figura 3: Diagrama de casos de uso de Informacion Articulos

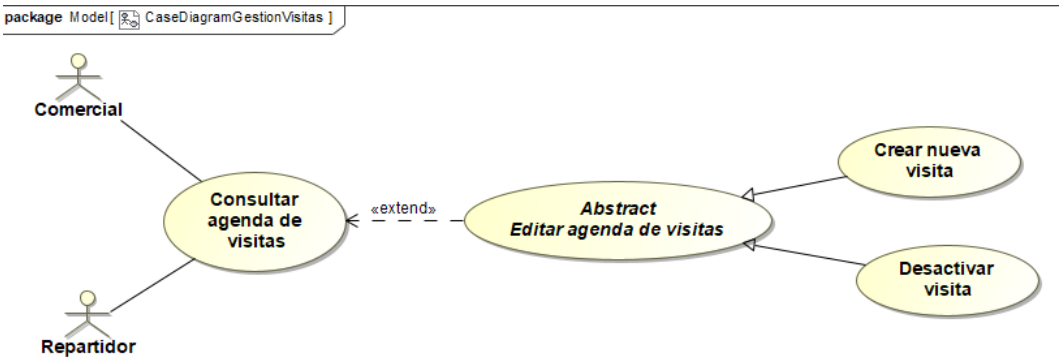


Figura 4: Diagrama de casos de uso de Gestion Visitas

## 4.1. Especificación de casos de usos

Una vez realizados todos los diagramas de casos de uso, se pretende dar aun más nivel de detalle y ofrecer una especificación detallada para cada caso de uso. Previamente vamos a definir que es una especificación detallada de un caso de uso. Esta es una descripción de cómo los distintos actores interactúan con la aplicación. Con esto se logra considerar todas las posibles situaciones que se puedan dar, tanto las habituales, como las inesperadas o las de error. Esta especificación puede realizarse utilizando diversas técnicas que van desde una do-

cumentación textual (normalmente utilizando plantillas), hasta diagramas de diversos tipos (secuencia, estados, actividades). En este caso se va a optar por una documentación textual.

UML no tiene ninguna propuesta para esta especificación, por lo que se va a utilizar una plantilla basada en las propuestas por Arlow [14] y por Cockburn [15]. La plantilla se muestra en la tabla 4

| | **Significado** |
|---|---|
| Nombre | Nombre unico que identifica el caso de uso |
| Descripción | Párrafo con el objetivo del caso de uso |
| Actores Primarios | Inicia el caso de uso |
| Actores secundarios | Participa en el caso de uso una vez iniciado |
| Evento de activacion | Evento que lanza el caso de uso |
| Precondiciones | Restricciones que ha de cumplir el estado del sistema para que el caso de uso pueda ejecutarse |
| Flujo principal | Pasos del escenario de ejecucción normal del caso de uso, el escenario de éxito |
| Postcondiciones | Estado del sistema al finalizar el caso de uso |
| Flujos alternativos | Pasos de los escenarios alternativos de ejecución, que suelen corresponder a escenarios de error |

Tabla 4: Ejemplo de plantilla de especificación de casos de uso

Una vez mostrada la plantilla que se va a seguir, se mostrará la especificación de los casos de uso que requieran de una mayor documentación, que generalmente suelen coincidir con los de una mayor complejidad. En la tabla 5 se detalla el caso de uso *Iniciar Sesion* y en la tabla 6 se detalla el caso de uso *Crear Visita*.

| | **Significado** |
|---|---|
| Nombre | CU1 |
| Descripción | El usuario inicia sesión en el sistema |
| Actores Primarios | Usuario no registrado |
| Actores secundarios | |
| Evento de activacion | El usuario inicia la aplicación |
| Precondiciones | |
| Flujo principal | 3.1 El usuario introduce su dni y contraseña.<br>3.2 El usuario pulsa el botón *Login*.<br>3.3 El sistema lleva al usuario a su respectiva pantalla. |
| Postcondiciones | |
| Flujos alternativos | 3.1 El sistema muestra un mensaje de error si el dni o la contraseña son incorrectas.<br>3.2 El sistema muestra un mensaje de error si no hay conexion a internet.<br>3.3 El sistema muestra un mensaje de error si no hay conexion con la base de datos. |

Tabla 5: Especificacion para el caso de uso: Iniciar Sesion

| | **Significado** |
|---|---|
| Nombre | CU2 |
| Descripción | El comercial crea una visita |
| Actores Primarios | Comercial |
| Actores secundarios | |
| Evento de activacion | El usuario pulsa el botón *Crear Visita* |
| Precondiciones | El usuario esta en la lista de visitas |
| Flujo principal | 1. El sistema muestra un formulario al comercial.<br>2. El comercial rellena los campos *Usuario* y *Contraseña*.<br>3. El comercial pulsa el botón *Crear visita*. |
| Postcondiciones | |
| Flujos alternativos | 3.1 El sistema muestra un mensaje de error. |

Tabla 6: Especificacion para el caso de uso: Crear visita

# 5. Diseño

En esta sección se va a describir la toma de decisiones que se ha llevado para poder desarrollar este sistema software de manera que pueda satisfacer los requisitos funcionales y no funcionales que se especificaron al principio del proyecto durante la fase de analisis. Se van a describir dos tipos de diseño: el diseño arquitectonico o de alto nivel y el diseño detallado o de bajo nivel.

## 5.1. Diseño Arquitectónico

El diseño aquitectonico de un sistema software describe a alto nivel los componentes, tanto hardware como software, y la estructura del mismo, es decir, la relacion entre los distintos subsistemas y componentes y como se relacionan entre ellos.

### 5.1.1. Diseño del sistema

A la hora de elegir qué diseño arquitectónico utilizar para el proyecto, no se tiene porque empezar de cero, sino que hay patrones y modelos que ya se han utilizado y probado ofreciendo distintas ventajas y desventajas.

La elección de un correcto diseño arquitectónico a la hora de realizar un proyecto software tiene un gran impacto en el resultado final del mismo, ya que permite analizar y verificar que se cumplen los requisitos funcionales, predecir y controlar el grado de satisfacción de los requisitos no funcionales desde una etapa temprana de desarrollo y permite una escalabilidad más fácil.

Tras un proceso de estudio, la elección del diseño del sistema viene restringido por la necesidad de que la aplicación cliente sea Android, que se debe de conectar a un servidor ya existente. Por tanto, se ha decidido usar una arquitectura en dos capas, mostrado en la figura 5 también llamada cliente/servidor. Este diseño consiste en separar la aplicación en dos componentes lógicos y que cada uno de ellos cumpla una función determinada.

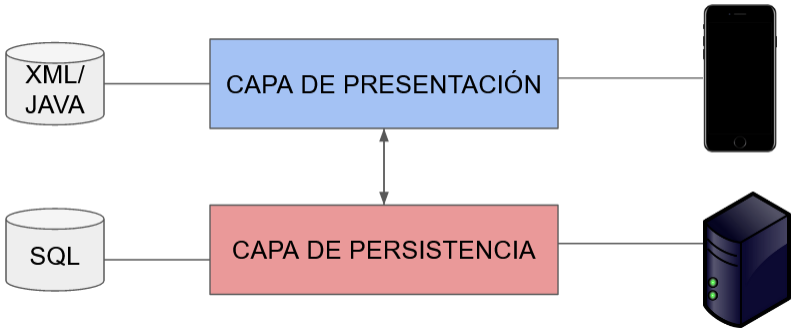


Figura 5: Diseño arquitectonico de 2 capas

- **Capa de presentación:** donde el usuario interactúa con la aplicación. Su principal objetivo es mostrar información al usuario y recoger la que introduzca, es decir, se encarga de recoger los eventos del usuario, procesarlos mediante una serie de reglas y operaciones definidas y presentarle los resultados. Si se requiriesen datos que no tuviese la aplicación, se encargaría de pedírselos a la capa de persistencia.

- **Capa de persistencia:** almacena y gestiona los datos de la aplicación. En este proyecto la capa de persistencia esta hecha en SQL y se utiliza el gestor de bases de datos SQL Server.

### 5.1.2. Diseño de la aplicación móvil

Para el diseño de la aplicación móvil se ha elegido el patrón de diseño *Modelo-Vista-Presentador*, el cual es una derivación del patrón *Modelo-Vista-Controlador*. Este patrón es muy utilizado para la construcción de interfaces de usuario.

Consiste en separar el código en tres partes para darle una responsabilidad a cada una de ellas. Como su nombre indica las tres partes son: Modelo, Vista y Presentador. Gracias a esta separación por responsabilidades, este patrón favorece la mantenibilidad, escalabilidad y reusabilidad de la aplicación. Es ampliamente utilizado ya que permite separar la vista o diseño de la interfaz de la aplicación de la lógica de la presentación.

Al igual que con el diseño del sistema, se va a explicar este patrón en mayor profundidad para una mayor comprensión del mismo con el apoyo de la figura 6:

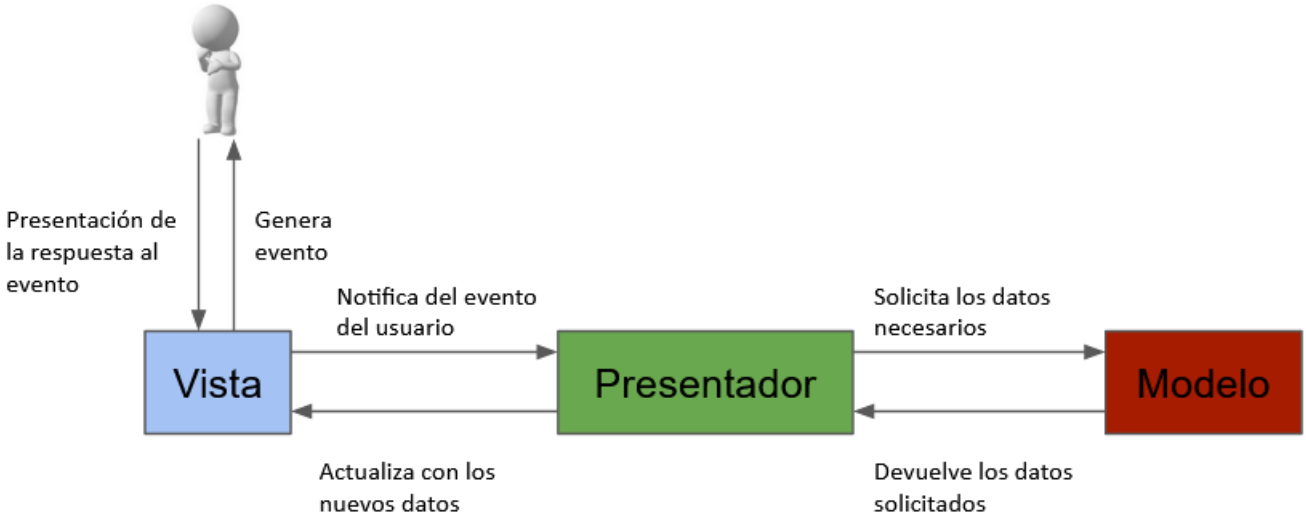


Figura 6: Patrón MVP

- **Modelo:** capa que define los datos a mostrar por la aplicación o sobre los que puede actuar el usuario. El modelo es la representación de los datos.

- **Vista:** es una capa que muestra los datos que el usuario ha pedido y se encarga de comunicar las peticiones del usuario al presentador. La vista no procesa nada que no sea la propia vista de la interfaz.

- **Presentador:** capa intermedia entre el modelo y la vista. Recoge de la vista los eventos y peticiones del usuario, y mediante los datos propocionados por el modelo procesa esas peticiones y le devuelve a la vista la informacion requerida actualizada.

Una vez elegido el patrón arquitectonico que se va a seguir durante el desarrollo del proyecto software, vamos a realizar un diagrama de componentes para modelar la arquitectura de la aplicación. Un diagrama de componentes permite modelar a alto nivel los diferentes componentes de los que se compone la aplicación, la forma en la que estos se organizan y las dependencias entre ellos.

A continuación, se muestra el diagrama de componentes en la figura 7 que corresponde con esta aplicación. Este diagrama sirve para representar las relaciones existentes entre cada unidad de la aplicación. En este diagrama se muestran las relaciones entre las vistas, los presentadores y el acceso a la base de datos. Por motivos de visualización se muestran solo 9 vistas y presentadores aunque la aplicación tenga en total 14.

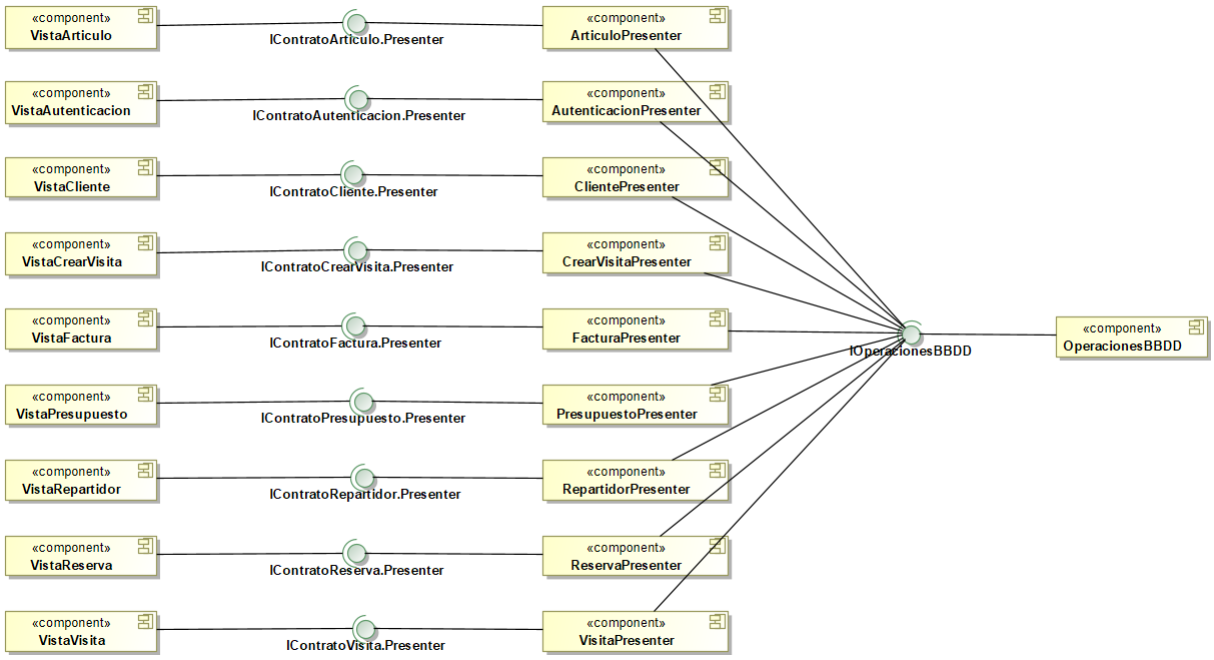


Figura 7: Diagrama de componentes de PiscisOutOffice

### 5.1.3. Diseño de despliegue

Por último, en cuanto al diseño arquitectonico, se va a mostrar el diagrama de despliegue de esta aplicación. Con este diagrama conseguimos detallar como se relacionan físicamente los componentes software y hardware en el sistema.

En la figura 8, se puede ver el diagrama de despliegue correspondiente al sistema del proyecto. Primeramente se puede observar cómo la aplicación PiscisOutOffice se empaqueta en un fichero .apk, que es el formato que utiliza el sistema operativo Android para sus aplicaciones. Este .apk se despliega y tiene acceso a la librería Room, la cual hace de *caché* con los datos que otorga la capa de persistencia. La aplicación se va a conectar con el servidor de hosting de la empresa donde se encuentra la base de datos SQL sobre la cual hará consultas, modificaciones y eliminaciones de datos.

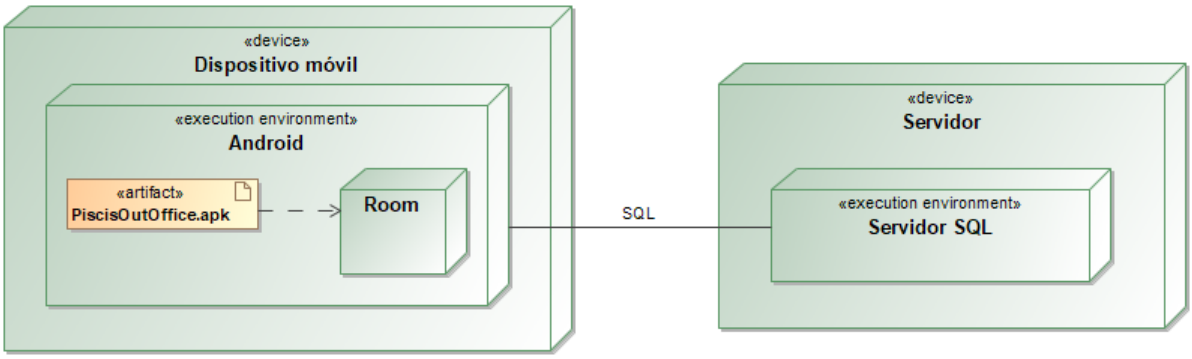


Figura 8: Diagrama de despliegue de PiscisOutOffice

## 5.2. Diseño detallado

Una vez finalizado con el diseño arquitectónico, se va a proceder a profundizar en las versiones preeliminares de los componentes del sistema hasta llegar a las clases finales del mismo.

Con este proceso se pretende detallar las funcionalidades y su comportamiento de manera que el proyecto puede empezar a implementarse.

### 5.2.1. Diagrama de clases

La figura 9 muestra el diagrama de clases correspondiente a la aplicación *PiscisOutOffice*, su objetivo es representar de manera gráfica y con detalle el conjunto de clases del sistema y la interacción entre las mismas. Como se puede ver en la figura, las clases se diferencias en dos grupos distintos: la parte de empleados con sus visitas y la parte de los clientes. De los empleados se guardan en la aplicación temporalmente sus datos personales y las visitas que tienen asociadas. Por la parte de los clientes, se guardan temporalmente datos de contacto y sus facturas, presupuestos y reservas de artículos.

### 5.2.2. Almacenamiento de los datos en la aplicación móvil

Para no tener que estar accediendo a la base de datos constantemente durante la utilización de la aplicación, se ha creado una base de datos dentro del dispositivo Android utilizando SQLite mediante la librería Room. Esto supone una mejora en el rendimiento de la aplicación, ya que una conexión a la base de datos remota supone muchos recursos. Otro de los objetivos de la creación de esta base de datos local es evitar los posibles problemas de memoria que podría tener lugar si los datos que recogemos de la base de datos remota se alojasen en estructuras de datos propias de Java.

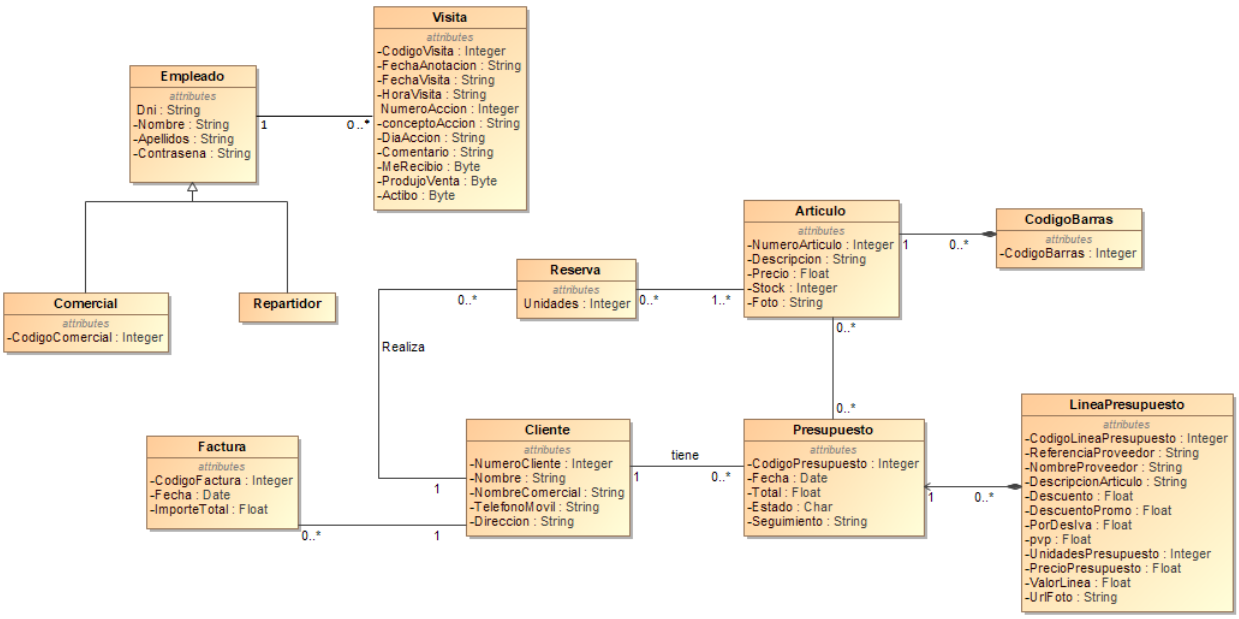


Figura 9: Diagrama de clases de PiscisOutOffice

# 6. Implementación

Una vez concluidas las fases de analisis y diseño, se desarrollará la fase correspondiente a la implementación. En esta sección se explicará el desarrollo del sistema y cómo se han utilizado las diferentes tecnologías teniendo en cuenta las decisiones que se llevaron a a cabo en la fase de diseño.

## 6.1. Estructura del proyecto

Como se mostró al principio de este informe, la aplicación móvil ha sido desarrollada utilizando el entorno de desarrollo *Android Studio*, entorno oficial para el desarrollo de aplicaciones Android por parte de Google.

Antes de mostrar la division por paquetes creada para poder respetar los patrones de diseño, es importante destacar que Android Studio de por si crea una serie de carpetas (figura 10) en donde se almacenan diferentes tipos de archivo con diferentes funcionalidades que se van a explicar a continuación.

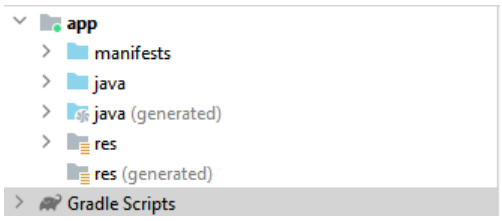


Figura 10: Estructura de paquetes de PiscisOutOffice

Con el apoyo en la figura anterior, se va a proceder a explicar la funcionalidad de estas carpetas:

- **Manifest:** este paquete solamente contiene el archivo AndroidManifest.xml. Este es un archivo con la información imprescindible para que se pueda ejecutar la aplicación. En este archivo se debe nombrar el identificador único de la aplicación, sus diferentes componentes, los permisos que se van a pedir al usuario para su correcto funcionamiento, los servicios del sistema que se van a utilizar y la clase por la que debe de empezar a ejecutarse la aplicación.

- **Java:** este paquete, como su nombre indica, contiene los archivos de código fuente de Java. El contenido de este paquete será detallado más adelante con el apoyo de una figura de su contenido para este proyecto.

- **Res:** este paquete contiene los recursos que se van a utilizar durante el proyecto. Contiene 4 paquetes que se encargan de recoger diferentes tipos de archivos:

    - **Drawable:** contiene las diferentes imágenes que utiliza la aplicación. También contiene archivos XML para diferentes recursos visuales.
    - **Layout:** contiene los archivos XML que describen las vistas en la interfaz gráfica.
    - **Mipmap:** Contiene los iconos que utiliza la aplicación.
    - **Values:** contiene diferentes archivos XML para la definición de strings o de colores
- **Gradle Scripts:** Contiene todos los scripts necesarios para la correcta compilación y funcionamiento de la aplicación. En estos scripts se define la versión de Android requerida, las dependencias del proyecto, los repositorios a los que acceder para las dependencias, opciones de compilación, ubicación de librerias etc.

En la figura 11 se muestra la división por paquetes que se ha creado para poder respetar los patrones de diseño dentro del paquete Java de la figura anterior, y se explicarán con detalle la función que desarrollan dentro de la aplicación.

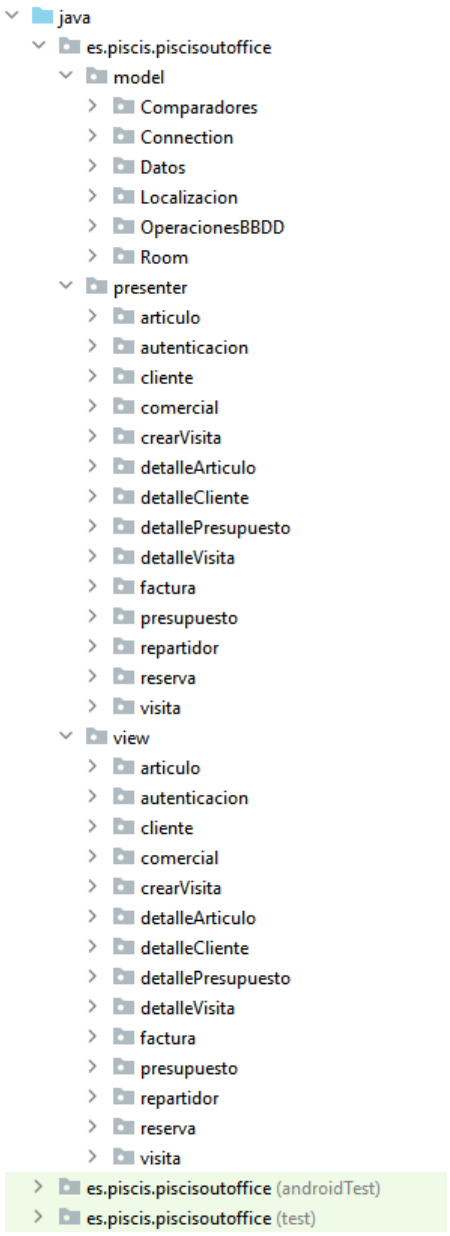


Figura 11: Estructura de paquetes Java de PiscisOutOffice

Como se puede apreciar en la figura, se ha dividido el codigo java en los siguientes paquetes:

- **Model:** este paquete contiene seis subpaquetes:
  - **Comparadores:** Aquí se encuentran los diferentes comparadores creados para ordenar las visitas segun su fecha de visita y los clientes por orden alfabético.
  - **Conexión:** Aquí se encuentra la clase destinada a establecer la conexión con la base de datos de la empresa, en donde se persiste la información.
  - **Datos:** Aquí se encuentran una serie de variables para facilitar la lógica de la aplicación.
  - **Localizacion:** Aquí se encuentra la lógica de la geolocalización del dispositivo.
  - **OperacionesBBDD:** Aquí se encuentra las operaciones que se realizan sobre la base de datos, es decir, las consultas, inserciones y modificaciones que se deben realizar en la base de datos.
  - **Room:** Aquí encontramos las entidades, DAOs (Data Access Objects, es un objeto que proporciona una interfaz abstracta a algún tipo de base de datos) y la base de datos interna del dispositivo.
- **Presenter:** este paquete contiene las clases que se encargan de intermediar entre los paquetes *View* y *Model*. Gestiona la lógica de la presentación de la información, actualizando la vista con los datos que aporta el modelo.
- **View:** se encuentran las clases encargadas de mostrar toda la información requerida por pantalla al usuario. Solo contiene lógica de vista, toda la demás lógica se la delega al presenter.

Los paquetes de la base anterior son la base del patrón MVP, en la figura 12 se muestran los metodos de la clase AutenticacionPresenter.java que se encargan de realizar toda la lógica y las operaciones necesarias que requiere la vista para la autenticación. Este presenter se encarga de comprobar si hay conexion a internet y a la base de datos. Una vez comprobada la conexion, llama a un método del modelo (buscarTrabajador() que se encuentra en el paquete OperacionesBBDD listado anteriormente) para buscar al trabajador en la base de datos remota, después, lo introduce en la base de datos local del dispositivo y llama al método que corresponda de la vista, es decir, si el trabajador encontrado es un comercial, se llama al método de la vista que abre la vista del comercial y, si el trabajador encontrado es un repartidor, abre la vista del repartidor.

```
public class AutenticacionPresenter implements IContratoAutenticacion.Presenter  {

    // ROOM
    IComercialDAO comercialDAO;
    IRepartidorDAO repartidorDAO;

    private final IContratoAutenticacion.View vista;
    private final IOperacionesBBDD operacionesBBDD;

    public AutenticacionPresenter(IContratoAutenticacion.View vista, BaseDeDatos db) {...}

    @Override
    public void onLoginClicked(String dni, String contrasena) {...}

    @Override
    public void comprobarConexiones() {...}

    private void comprobarConexionInternet() {...}

    private void comprobarConexionBBDD() {...}
}
```

Figura 12: AutenticacionPresenter

En la figura 13 se muestra los métodos de la vista, los cuales se encargan de abrir otras vistas.

Las figuras muestran los métodos de la clase MenuAutenticacionActivity.java, esta vista es la primera que se ve cuando se ejecuta la aplicación, por lo cual tiene una importancia superior a otras. Esta vista se encarga de comprobar si se tienen los permisos necesarios para poder iniciar el servicio de localización, en caso afirmativo lo inicia y en caso negativo le pide los permisos al usuario. La vista, al extender la clase *AppCompatActivity*, la cual es la clase base para todas las activities, permite acceder a parametros del sistema. Con el acceso a estos parametros se puede comprobar si el dispositivo tiene conexión a internet. Si no la tiene, el método onConexionInternetError() muestra una alerta al usuario indicandole que no tiene conexión a internet, y ofrece la opción de actualizar para cuando se haya retomado la conexion. Lo mismo ocurre con el método onConexionBBDDError(), el cual comprueba la conexión con la base de datos remota. Por último, se encarga de abrir la vista dependiendo del trabajador que este usando la aplicación.

Como se ha podido ver, la vista se encarga de comprobar parámetros del sistema, de abrir otras vista y de mostrar alertas visuales. El presenter por su parte se encarga de gestionar las peticiones de la vista y de actualizarla con los datos obtenidos del modelo. Por último el modelo se encarga de consultar la base de datos remota y transformar los resultados en objetos Java sobre los que pueda trabajar los presenters.

```
public class MenuAutenticacionActivity extends AppCompatActivity implements
        IContratoAutenticacion.View, View.OnClickListener {

    @SuppressLint("WrongThread")
    @Override
    protected void onCreate(Bundle savedInstanceState) {...}


    @Override
    public void onClick(View view) {...}

    @Override
    public void onComercialAutenticado() {...}

    @Override
    public void onRepartidorAutenticado() {...}

    @Override
    public void onAutenticacionError() {...}

    @Override
    public void onRolError() {...}

    /**
     * Metodo que saca un Alert Dialog cuando no hay conexión a internet
     */
    public void onConexionInternetError() {...}

    /**
     * Metodo que saca un Alert Dialog cuando no hay conexión con la BBDD
     */
    public void onConexionBBDDError() {...}

    /**
     * Metodo que comprueba si el teléfono móvil tiene conexión a Internet
     * @return true si hay conexión, false si no hay conexión
     */
    public boolean hasInternetConnection() {...}
```

Figura 13: MenuAutenticacionActivity

## 6.2. Servicios

En esta sección se explicará el servicio de localización que se ha creado para poder registrar la localización del dispositivo. Esto corresponde al RF12.

Lo primero de todo se tienen que solicitar los permisos necesarios para que la aplicación pueda utilizar el sistema de geolocalización del móvil y para que el servicio se ejecute en segundo plano. En concreto se han pedido los permisos de: *ACCESS_BACKGROUND_LOCATION, ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION y FOREGROUND_SERVICE*, que corresponden respectivamente al acceso a la localización en segundo plano, acceso a la localizacion precisa, acceso a la localización aproximada y permiso para ejecutar servicios en primer plano.

Estos permisos hay que declararlos en el archivo AndroidManifest.xml como se indica en la figura 14.

```
<!-- Permiso para acceder a la localizacion en segundo plano -->
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
<!-- Permiso para poder obtener una localizacion precisa -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<!-- Permiso para poder obtener una localizacion aproximada -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<!-- Permiso para poder crear un servicio -->
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
```

Figura 14: Declaracion de servicios en el archivo AndroidManifest.xml

El servicio de localización se tiene que activar según se inicia la aplicación, por lo que debemos de llamar al servicio en la primera pantalla que se ejecute. Como se puede ver en la figura 15, primero se comprueba si se tiene el permiso para poder acceder a la localización exacta, si no se tiene, se llama a un método para solicitar los permisos al usuario. Si los permisos ya se tienen, se llama al método *empezarServicioLocalizacion()*, el cual, lanza el servicio de localización.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_menu_autenticacion);

    if (ContextCompat.checkSelfPermission(getApplicationContext(), Manifest.permission.ACCESS_FINE_LOCATION) !=
            PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: MenuAutenticacionActivity.this,
                new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, REQUEST_CODE_LOCATION_PERMISSION);
    } else {
        empezarServicioLocalizacion();
    }
```

Figura 15: Metodo onCreate de la pantalla de inicio de la aplicación

Todas las vistas o activities, tienen un método llamado *onCreate*, el cual es llamado en la construcción de la actividad. La figura 15 corresponde al método *onCreate()* de la clase MenuAutenticacionActivity.java, esta clase es la primera en ejecutarse cuando se inicia la aplicación, por lo tanto este método *onCreate* es el primer método en ejecutarse de toda la aplicación. En este se comprueba si la aplicación tiene el permiso para acceder a la localización precisa, si lo tiene se llama al método *empezarServicioLocalizacion()*, el cual comprueba si esta en ejecución el servicio ya, y de lo contrario lo inicializa. Si el servicio no esta iniciado, se crea un Intent (objeto que representa una acción que se realizará) con la clase creada para obtener la localización, y se inicia como un servicio.

En la figura 16 se muestra el método que se encarga de ejecutar el servicio de localización en segundo plano. En este método se crea un gestor de notificaciones, a través del cual se avisa al usuario mediante una notificación textual y sonora de que está corriendo en su dispotivo un servicio de geolocalización. Además, se crea la petición de la ubicación del dispositivo al sistema, se establece el intervalo de la petición de la localización, es decir, el tiempo que tiene que transcurrir entre una petición de la localización y otra, y se establece esta petición de alta prioridad dentro del sistema.

```
private void empezarServicioLocalizacion() {
    String idCanal = "canal_notificacion_localizacion";
    NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    Intent resultIntent = new Intent();
    PendingIntent pendingIntent = PendingIntent.getActivity(
            getApplicationContext(),
            requestCode: 0,
            resultIntent,
            PendingIntent.FLAG_UPDATE_CURRENT
    );
    NotificationCompat.Builder builder = new NotificationCompat.Builder(
            getApplicationContext(), idCanal
    );

    builder.setSmallIcon(R.mipmap.ic_launcher);
    builder.setContentTitle("Servicio de localizacion");
    builder.setDefaults(NotificationCompat.DEFAULT_ALL);
    builder.setContentText("Corriendo");
    builder.setContentIntent(pendingIntent);
    builder.setAutoCancel(false);
    builder.setPriority(NotificationCompat.PRIORITY_MAX);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        if (notificationManager != null && notificationManager.getNotificationChannel(idCanal) == null) {
            NotificationChannel notificationChannel = new NotificationChannel(
                    idCanal, name: "Servicio de Localizacion", NotificationManager.IMPORTANCE_HIGH
            );
            notificationChannel.setDescription("Este canal esta usado por el servicio de localizacion");
            notificationManager.createNotificationChannel(notificationChannel);
        }
    }

    LocationRequest locationRequest = new LocationRequest();
    locationRequest.setInterval(1000);
    locationRequest.setFastestInterval(500);
    locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

    LocationServices.getFusedLocationProviderClient( context: this)
            .requestLocationUpdates(locationRequest, locationCallback, Looper.getMainLooper());
    startForeground(ID_SERVICIO_LOCALIZACION, builder.build());
```

Figura 16: Metodo que permite ejecutar el servicio de localización

## 6.3. Interfaz gráfica

En esta sección se mostrará cómo se realiza la creación de una interfaz gráfica y se explicarán alguna de las pantallas principales de la aplicación en base a figuras.

Para que el usuario pueda interactuar con la aplicación de manera correcta, la aplicación

debe de estar dotada de pantallas con las cuales pueda realizar ciertos eventos. En Android estas pantallas se denominan actividades. El contenido de estas actividades se describen mediante XML, ya que Android Studio proporciona un editor visual de estos archivos XML, donde se muestra una previsualización de la actividad. Cada actividad es independiente de las demás, por lo tanto se hace responsbale de gestionar los diferentes elementos visuales que se quieran colocar sobre ella y de recoger los eventos que se quiera ofrecer a los usuarios. Todo elemento gráfico debe de estar dentro de un *Layout*, que son elementos que dotan a la pantalla de unas ciertas reglas sobre las que se pueden colocar los elementos gráficos. Hay distintos tipos de Layouts dependiendo de cómo se quiera que se distribuyan los elementos en la pantalla.

En la figura 17 se muestra el codigo XML de la pantalla de autenticación para poder entender visualmente como funcionan las actividades.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".view.autenticacion.MenuAutenticacionActivity">

    <ImageView
        android:id="@+id/logo_imagen"
        android:layout_width="300dp"
        android:layout_height="200dp"
        android:layout_marginTop="44dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/logo_piscis_png"
        tools:ignore="ImageContrastCheck" />

    <TextView
        android:id="@+id/tv_iniciar_sesion"
        android:layout_width="250dp"
        android:layout_height="50dp"
        android:layout_marginTop="40dp"
        android:background="@drawable/textview_border_red"
        android:gravity="center"
        android:text="Iniciar sesión"
        android:textAppearance="@style/TextAppearance.AppCompat.Body1"
        android:textSize="24sp"
        app:circularflow_defaultRadius="50dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.500"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/logo_imagen" />

    <EditText
        android:id="@+id/et_usuario"
        android:layout_width="250dp"
        android:hint="Usuario"
        android:inputType="text"
        android:textAlignment="center"
        android:textAppearance="@style/TextAppearance.AppCompat.Body1"
        android:textSize="24sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/tv_iniciar_sesion" />

    <EditText
        android:id="@+id/et_password"
        android:layout_width="250dp"
        android:layout_height="50dp"
        android:layout_marginTop="48dp"
        android:background="@drawable/gradiant_red"
        android:ems="10"
        android:gravity="center"
        android:hint="Contraseña"
        android:inputType="numberPassword"
        android:textAlignment="center"
        android:textSize="24sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/et_usuario" />

    <Button
        android:id="@+id/btn_login"
        android:layout_width="250dp"
        android:layout_height="50dp"
        android:layout_marginTop="44dp"
        android:text="Login"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/et_password" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Figura 17: Fichero XMl de la actividad MenuAutenticacionActivity

En la figura se puede observar como lo primero que definimos es el layout, en este caso se ha obtado por un ConstraintLayout el cual coloca automáticamente los elementos de acuerdo a una serie de restricciones que se describen en el fichero xml. Para esta actividad (mostrada en la figura 18) se ha usado un Button, un ImageView, que permite mostrar una imagen, un TextView, que permite mostrar una cadena de texto no editable y dos EditText, que es un cuadro de texto donde el usuario tendrá que introducir su usuario y contraseña para poder autenticarse en la aplicación.

### 6.3.1. Menu autenticación

Lo primero que se encuentra el usuario al abrir la aplicación es esta pantalla de autenticación donde tendrá que introducir sus credenciales. Si las credenciales son correctas, dependiendo si se trata de un comercial o de un repartidor se le llevará a una vista u otra. En la interfaz se puede observar el logo de la empresa arriba, una cadena de texto que indica que se encuentra en la pantalla de iniciar sesion, dos cuadros de texto donde tendrá que introducir las credenciales, y por último un botón que contiene un evento para procesar la autenticación. La pantalla de la figura 18 corresponde con el archivo XML mostrado en la figura 17.

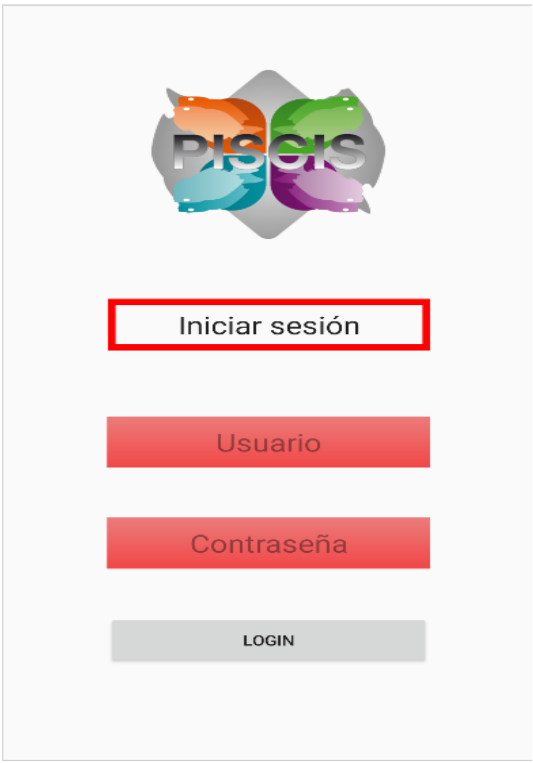


Figura 18: Menu de autenticacion de la aplicación PiscisOutOffice

### 6.3.2. Menu Comercial

Una vez el comercial se ha autenticado, se le muestra un menú con las tres opciones que puede realizar, las cuales son:

- **Agenda clientes:** Este botón dirige al comercial a su agenda de clientes.
- **Agenda visitas:** Este botón dirige al comercial a su agenda de visitas activas.
- **Buscar articulo:** Este botón dirige al comercial a una pantalla con todos los artículos de la empresa.

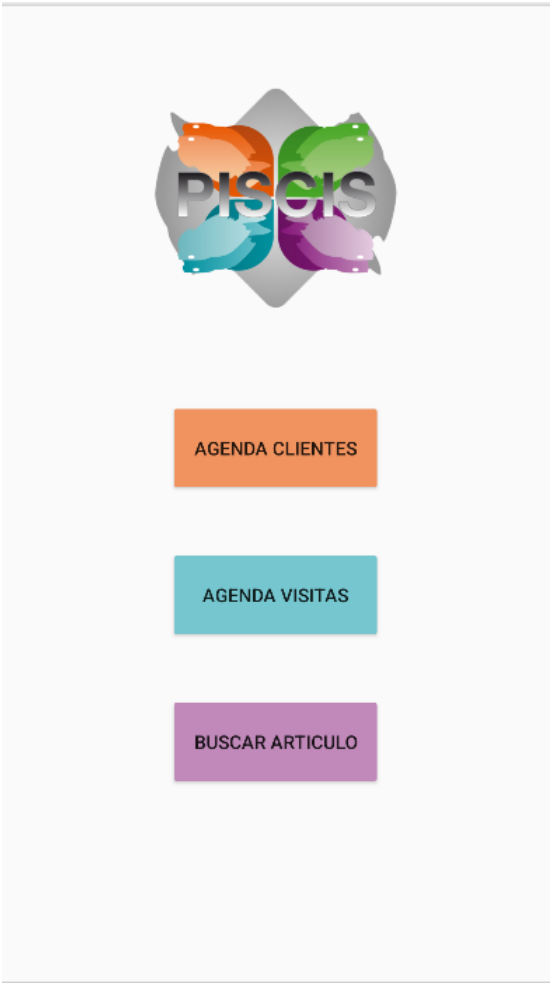


Figura 19: Menu del comercial de la aplicación PiscisOutOffice

# 7. Pruebas

En esta sección se describen las diferentes pruebas que se han llevado a cabo durante el proyecto con el objetivo de probar el funcionamiento correcto del software desarrollado. Estas pruebas permiten poder detectar fallos si los hubiera y así poder corregirlos.

Durante este proyecto se han definido y realizado diferentes tipos de prueba a fin de poder cubrir el máximo código posible y minimizar los posibles fallos.

Esta fase del proyecto es de gran importancia, ya que permite mejorar la calidad del producto, detectando posibles problemas de manera sistemática.

## 7.1. Pruebas unitarias

Las pruebas unitarias permiten comprobar el correcto funcionamiento de las unidades individuales más pequeñas del código software desarrollado. En estas pruebas cada unidad es aislada del resto, de esta forma se asegura que por separado todas las unidades funcionan.

En este proyecto, las pruebas unitarias se han desarrollado haciendo uso del framework JUnit, el cúal permite la automatización de las pruebas en Java, y las librerias Mockito, que te permite la creación de Mocks (objeto que simula el comportamiento de la unidad a la que sustituye para poder aislar la unidad en prueba) y Roboelectric, que es un marco que permite que las pruebas se ejecuten en un entorno simulado, para así no tener la sobrecarga del emulador Android.

Por motivos de espacio, se va a mostrar el plan de prueba y las pruebas realizadas para la clase ClientePresenter.

Un plan de pruebas es un documento técnico en el que se especifican las diferentes posibles situaciones que puede ocurrir con la unidad en prueba y su comportamiento esperado en esas situaciones.

Antes de pasar con las pruebas en sí, explicar que en la clase de los test se han creado dos métodos de configuración, uno que se ejecute antes de cada test, identificado con la anotación de Junit *@Before*, y otro que se ejecuta después de cada test, identificado con la anotación de Junit *@After*. Estos dos métodos se pueden ver en la figura 20

```java
/**
 * Obtenemos la conexión con la base de datos y con el semáforo esperamos a que se obtenga
 */
@Before
public void setUp() {
    StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();
    StrictMode.setThreadPolicy(policy);
    FactoriaDeConexiones.obtenerConexion();
    lock.arriveAndAwaitAdvance();
    createDb();
    DatosTrabajador.setCodigoComercial(15);
}

/**
 * Despues de cada test cerramos la base de datos
 * @throws IOException
 */
@After
public void closeDb() throws IOException {
    bd.clearAllTables();
    bd.close();
}
```

Figura 20: Metodos de configuracion de los test unitarios

Como se indica en los comentario de la figura, estos métodos se encargan de establecer la conexión con la base de datos SQL y crear la base de datos Room antes de cada ejecución del test, y de limpiar y cerrar la base de datos Room al acabar cada ejecución de test. Una vez explicado esto, pasamos con los planes de prueba.

Método **onClientePulsado(int clienteIndex)** de la clase ClientePresenter.java

Casos de prueba válidos:

| **Identificador** | **Entrada** | **Resultado** |
|---|---|---|
| UT.1a | 0 → se corresponde con el primer cliente de la lista | Se abre la lista detallada del primer cliente de la lista |

Tabla 7: Casos válidos del método onClientePulsado(int clienteIndex)

Casos de prueba no válidos:

| Identificador | Entrada | Resultado |
|---|---|---|
| UT.1b | -1 → esta fuera de los límites de la lista | Se lanza la excepción *IndexOutOfBounds* |
| UT.1c | Tamaño de la lista + 1 → esta fuera de los límites de la lista | Se lanza la excepción *IndexOutOfBounds* |

Tabla 8: Casos no válidos del método onClientePulsado(int clienteIndex)

Cómo se puede ver en las tablas 7 y 8, se van a probar tres casos de prueba distintos, los cuales se muestran en la imagen 21

```
@Test
public void onClientePulsadoTest() {
    sut = new ClientePresenter(mockView, bd);

    // Ut.1a Se comprueba que al pulsar en un cliente se llama abrirDetalleCliente() con el cliente correcto
    sut.onClientePulsado( clienteIndex: 0);
    verify(mockView).abrirDetalleCliente(captor.capture());
    assertEquals(captor.getValue().getNombre(), clientesDAO.getAllClientes().get(0).getNombre());

    // UT.1b Se comprueba que al introducir un indice x < 0 se lanza la excepcion indexOutOfBound
    try {
        sut.onClientePulsado( clienteIndex: -1);
        fail("No se ha lanzado la excepción");
    } catch (IndexOutOfBoundsException e) {
        assertTrue( condition: true);
    }

    // UT.1c Se comprueba que al introducir un indice x > listSize, se lanza la excepcion indexOutOfBound
    try {
        sut.onClientePulsado(clientesDAO.getAllClientes().size());
        fail("No se ha lanzado la excepción");
    } catch (IndexOutOfBoundsException e) {
        assertTrue( condition: true);
    }
}
```

Figura 21: Metodo onClientePulsadoTest()

Método **onFiltroBusquedaClientesTiempoReal(String busqueda)** de la clase ClientePresenter.java

Casos de prueba válidos:

| **Identificador** | **Entrada** | **Resultado** |
|---|---|---|
| UT.2a | x → ningun cliente contiene la x | la lista debe salir vacía |
| UT.2b | ortega → un cliente contiene ortega | la lista debe contener 1 cliente |
| UT.2c | a → dos clientes contienen la letra a | la lista debe contener 2 clientes |

Tabla 9: Casos válidos del método onFiltroBusquedaClientesTiempoReal(String busqueda)

Casos de prueba no válidos:

| **Identificador** | **Entrada** | **Resultado** |
|---|---|---|
| UT.2d | null | Se lanza la excepción *NullPointerException* |

Tabla 10: Casos no válidos del método onFiltroBusquedaClientesTiempoReal(String busqueda)

Cómo se puede ver en las tablas 9 y 10, se van a probar cuatro casos de prueba distintos, los cuales se muestran en la figura 22.

```java
@Test
public void onFiltroBusquedaClientesTiempoReal() {
    sut = new ClientePresenter(mockView, bd);

    String ningunFiltrado = "x";
    String unoFiltrado = "ortega";
    String ambosFiltrado = "a";


    /*...*/
    sut.onFiltroBusquedaClientesTiempoReal(ningunFiltrado);
    assertEquals( expected: 0, clientesDAO.getAllClientes().size());

    /*...*/
    sut.onFiltroBusquedaClientesTiempoReal(unoFiltrado);
    assertEquals( expected: 1, clientesDAO.getAllClientes().size());

    /*...*/
    sut.onFiltroBusquedaClientesTiempoReal(ambosFiltrado);
    assertEquals( expected: 2, clientesDAO.getAllClientes().size());

    /*...*/
    try {
        sut.onFiltroBusquedaClientesTiempoReal(null);
        fail("No se ha lanzado la excepción");
    } catch (NullPointerException e) {
        assertTrue( condition: true);
    }
}
```

Figura 22: Metodo onFiltroBusquedaClientesTiempoReal()

## 7.2. Pruebas de integración

Una vez se han realizado las pruebas unitarias y comprobado el funcionamiento de cada unidad individual del software desarrollado, se ha procedido a realizar las pruebas de integración. Estas pruebas consisten en comprobar el funcionamiento entre las diferentes unidades funcionales. A parte de esta diferencia, con las pruebas de integración no se sustituye ninguno de los componentes en prueba, si no que se utilizan las unidades reales. Para esto, se simulan las interacciones del usuario y se captura la pantalla de la aplicación en el emulador android ofrecido por Android Studio.

Estas pruebas se realizan de manera incremental, primeramente se prueban las unidades que menos dependencias tienen, y se acaban con las unidades que mas dependencias tengan. Mediante esta metodología al comprobar primero las que menos dependencias tienen, se puede minimizar el riesgo de fallo por programación de las unidades que mas dependencias tienen.

Se van a mostrar los casos de prueba diseñados para las pruebas de integración de la clase MenuAutenticacionActivity, ya que se consideran representativas del resto de pruebas de integración realizadas. Se corresponden a las tablas 11 y 12.

Método **onClick()** de la clase MenuAutenticacionActivity.java

Casos de prueba válidos:

| **Identificador** | **Entrada** | **Resultado** |
|---|---|---|
| IT.1a | DNI: '72262441Y', Contraseña: '1000' | Se autentica en la aplicación |

Tabla 11: Casos válidos del método onClick()

Casos de prueba no válidos:

| **Identificador** | **Entrada** | **Resultado** |
|---|---|---|
| IT.1b | DNI: '45236521G', Contraseña: '5698' | Se muestra un pop-up diciendo que el dni o contraseña son incorrectos |
| IT.1c | DNI: '72262441Y', Contraseña: '5698' | Se muestra un pop-up diciendo que el dni o contraseña son incorrectos |
| IT.1d | DNI: '45236521G', Contraseña: '1000' | Se muestra un pop-up diciendo que el dni o contraseña son incorrectos |

Tabla 12: Casos no válidos del método onClientePulsado(int clienteIndex)

## 7.3. Pruebas de sistema

Las pruebas de sistema son las encargadas de comprobar el correcto funcionamiento de la aplicación en todo su conjunto y así poder verificar si se han cumplido los requisitos definidos al principio del proyecto.

Estas prueban han sido realizadas en un dispositivo Android real, para simular un uso de la aplicación por los empleados de la empresa.

Una vez comprobada la funcionalidad en el dispositivo, se han llevado a cabo pruebas de usabilidad y de rendimiento para comprobar si la aplicación es apta para su uso.

### 7.3.1. Pruebas de usabilidad

En este apartado se ha verificado el cumplimiento del requisito no funcional RNF03, es decir, que la aplicación sea intuitiva al usuario y así conseguir una experiencia satisfactoria y poder utilizarla de manera eficaz. Para realizar esta prueba se le ha dado a un trabajador de la empresa la aplicación para que la probase y el resultado ha sido favorable.

### 7.3.2. Pruebas de rendimiento

Con esta prueba se ha comprobado el rendimiento de la aplicación en el dispositivo a la hora de hacer tareas pesadas, como por ejemplo la conexión con la base de datos, la búsqueda de articulos en la base de datos (hay mas de 50.000) y la obtención de la geolocalización del dispositivo. El resultado ha sido correcto.

## 7.4. Pruebas de aceptación

Las pruebas de aceptación son realizadas por el cliente con el objetivo de que verifique el funcionamiento de la aplicación y de que se han cumplido los requisitos que se definieron durante el inicio del proyecto. Estas pruebas han sido realizadas por el jefe de la empresa *AlmacenesPiscis*. El resultado de estas pruebas ha sido favorable.

## 7.5. Pruebas de calidad

Las pruebas de calidad comprueban que el código cumple unos ciertos estándares. En este proyecto se han llevado a cabo mediante el uso de la herramienta sonarcloud. En la figura 23 se puede comprobar visualmente los estándares comprobados.

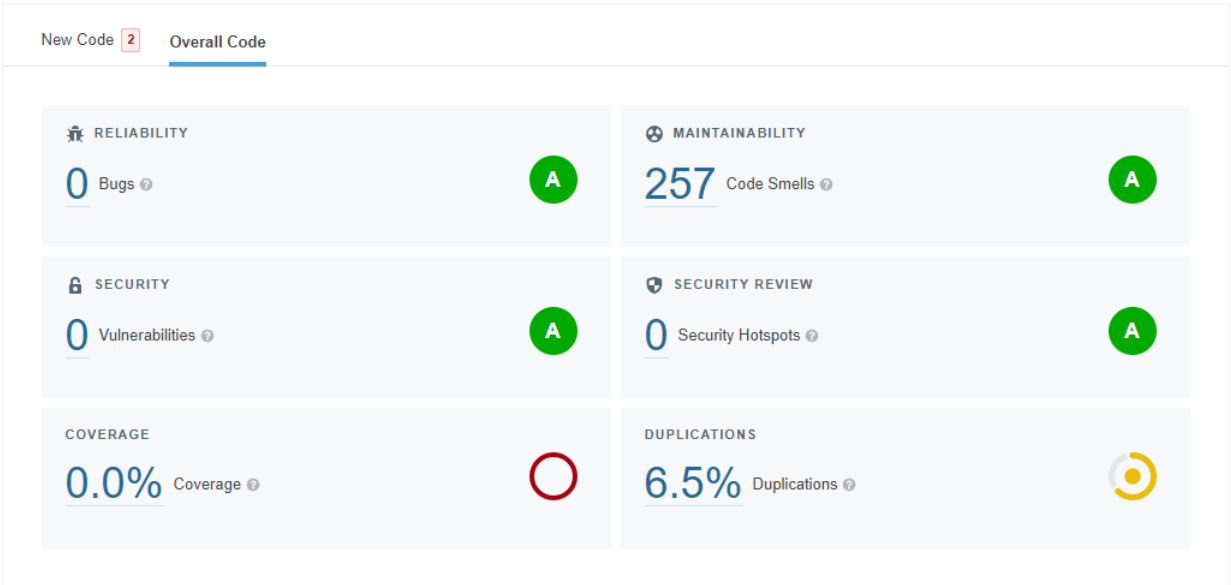


Figura 23: Análisis de código realizado por Sonarcloud

En el apartado de fiabilidad (Reliability), se cuenta la cantidad de errores que pueden romper el código, es decir, provocar un fallo no controlado que haga que la aplicación falle. En el apartado de mantenibilidad como su nombre indica se muestra la cantidad de código difícil de mantener. Los siguientes 2 apartados pertenecen a la seguridad de la aplicación. Estos 4 primeros apartado cumplen con una A los estándares de calidad propuestos por Sonarcloud.

El siguiente apartado es el coverage, que comprueba las líneas de código que son cubiertas por los test realizados. En este caso esta métrica no se ha podido obtener debido a problemas de compatibilidad con el plugin de analizar la cobertura.

Por último se mide la cantidad de líneas de código repetidas durante el código, en total se repiten un 6,5 %, lo cual es un porcentaje bajo de toda la aplicación.

En resumen, las pruebas de calidad son superadas con éxito.

# 8. Conclusiones y trabajos futuros

En esta sección se va a destacar las principales conclusiones que se obtienen tras la realización de este trabajo, y los trabajos que se van a realizar en el futuro.

Como objetivo principal, la aplicación debe de ofrecer una serie de funciones administrativas para ahorrar el tiempo que supone la comunicación entre distintas áreas de la empresa. Este objetivo se ha cumplido, así como el de crear un servicio de geolocalización para poder obtener las rutas que han hecho los comerciales y repartidores de la empresa.

La tecnología que se ha utilizado ha sido Android Studio con Java, tecnología con la cual he trabajado durante el primer cuatrimestre del último año, por lo que tenía unas nociones básicas de uso.

El primer problema que encontré fue la comunicación entre la aplicación y la base de datos remota de la empresa, ya que el driver no se podía utilizar sobre Android Studio sin la importación de una librería determinada. La solución fue descargar la librería en local y copiarla en el sistema de ficheros de Android Studio para que la pueda localizar.

Otro de los problemas fue la persistencia de los datos necesarios de manera local para así tener un mejor rendimiento al no ser necesario la conexión a la base de datos remota todo el rato. Al principio se persistian mediante estructuras de datos propias de Java, pero ocasionalmente generaba problemas de memoria, por lo que se optó por la creación de una base de datos interna dentro del dispositivo Android. Para la creación de esta base de datos se usó la tecnología Room, la cual nunca había usado, por lo que tuve problemas a la hora de guardar los datos ya que Android Studio en su documentación ofrecía varías maneras distintas para poder guardar los datos de manera asíncrona sin la necesidad de utilizar el thread principal, las cuales no funcionaban para la aplicación. Al final debido al pequeño volumen de los datos almacenados de manera local, se utiliza el thread principal para la persistencia de los datos.

Respecto a los trabajos futuros se van a seguir implementando funcionalidades para la aplicación tanto para los comerciales como para los repartidores con el fin de evitar la comunicación por completo entre estas áreas para la petición de documentos relativos al cliente. Además se pretende mantener y crear mas funcionalidades para la web de la empresa *Almacenes Piscis*, lo que supondrá un nuevo reto ya que nunca he trabajado con tecnologías web.

Este proyecto me ha permitido poner en práctica cosas estudiadas durante la carrera y que todavía no había podido ver en un entorno real empresarial. He podido entrar en el mundo empresarial y poder conocer mejor cómo va a ser el entorno de mi futuro. También me ha ayudado a aprender a gestionar satisfactoriamente las situaciones de posible estrés en las que me encontraba problemas técnicos que en principio parecían de difícil solución.

# 9. Bibliografía

## Referencias


[1] Wikipedia. *Android — Wikipedia, La enciclopedia libre*. URL: https://es.wikipedia.org/w/index.php.

[2] sqlite.org. *What Is SQLite?* URL: https://www.sqlite.org/index.html.

[3] Android. *Room*. URL: https://developer.android.com/jetpack/androidx/releases/room?hl=es_419.

[4] Wikipedia. *Microsoft SQL Server — Wikipedia, La enciclopedia libre*. URL: https://es.wikipedia.org/wiki/Microsoft_SQL_Server.

[5] git scm. URL: https://www.git-scm.com/.

[6] Android. *Android Studio*. URL: https://developer.android.com/studio/intro.

[7] Microsoft. *¿Qué es SQL Server Management Studio (SSMS)?* URL: https://docs.microsoft.com/es-es/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16.

[8] Wikipedia. *MagicDraw — Wikipedia, La enciclopedia libre*. URL: https://en.wikipedia.org/wiki/MagicDraw.

[9] sqlitebrowser. *What it is?* URL: https://sqlitebrowser.org/.

[10] gihub. URL: https://github.com/.

[11] Sonarcloud. *What is SonarCloud?* URL: https://sonarcloud.io/.

[12] Bourque y Dupuis. *Guide to the Software Engineering Body of Knowledge*. 2004.

[13] Karl Wiegers y Joy Beatty. *Software requirements*. Pearson Education, 2013.

[14] Jim Arlow e Ila Neustadt. *UML 2 and the unified process: practical object-oriented analysis and design*. Pearson Education, 2005.

[15] Alistair Cockburn. *Writing effective use cases*. Pearson Education India, 2001.

# Anexo

En este anexo se muestran los mockups realizados con el cliente al inicio del proyecto, en los que se describen los ciclos de vida de la interfaz de usuario mostrada a los trabajadores.

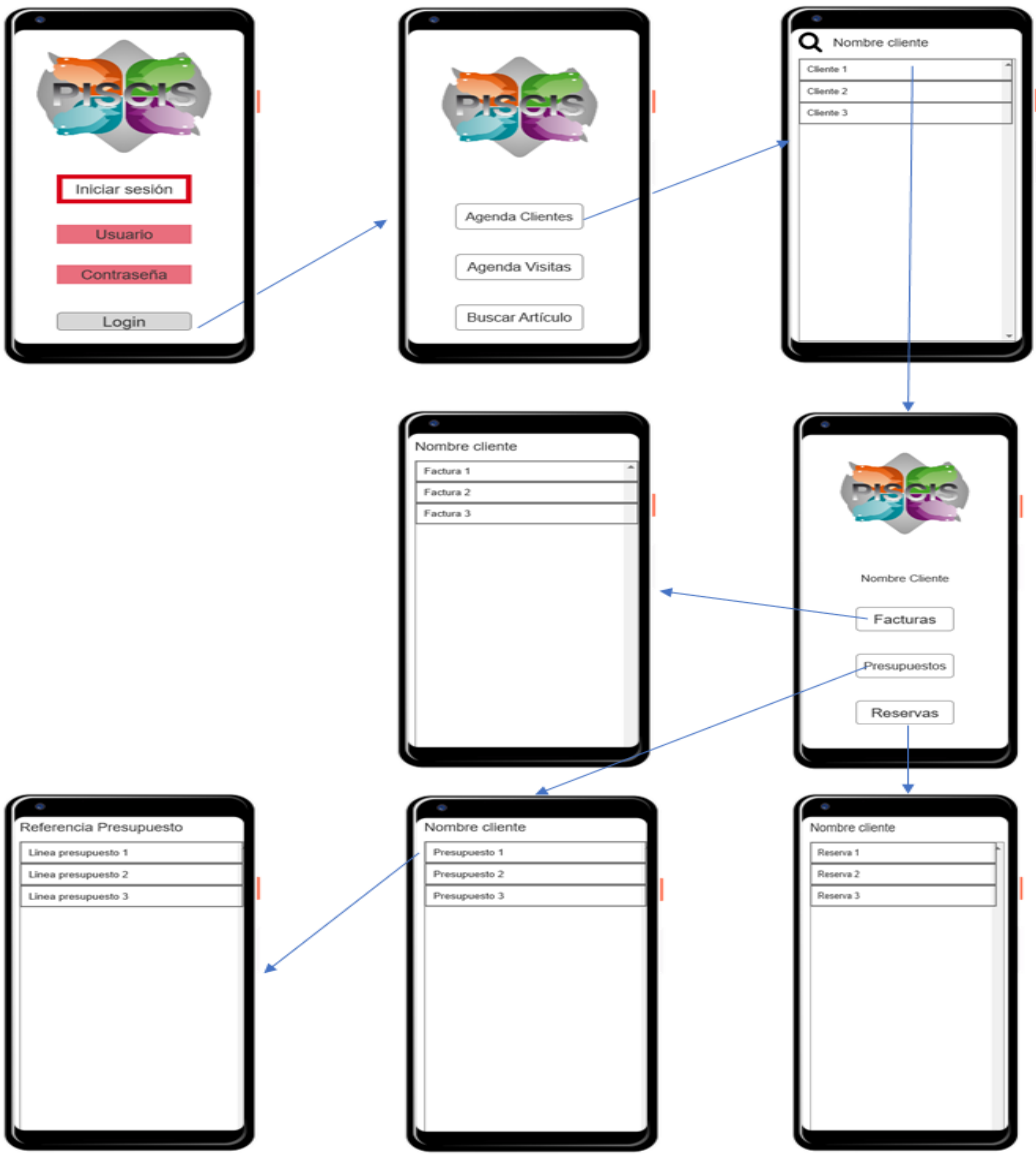


Figura 24: Ciclo de vida de las funciones de un comercial sobre los clientes


Figura 25: Ciclo de vida de las funciones de un comercial sobre las visitas

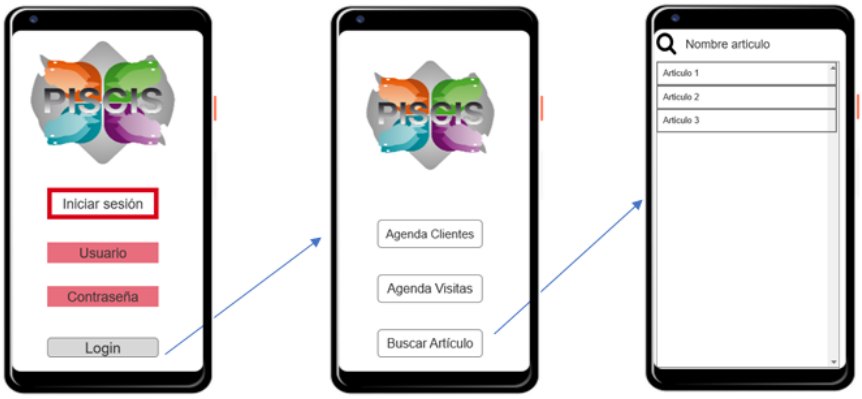


Figura 26: Ciclo de vida de las funciones de un comercial sobre los artículos

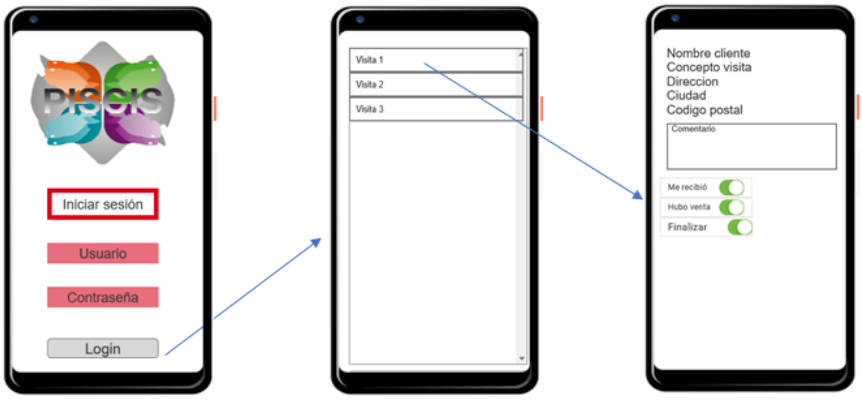


Figura 27: Ciclo de vida de las funciones de un repartidor