UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

Aplicación Android para pedir cita previa en peluquerías

TITULACIÓN: Graduado en Ingeniería en Tecnologías de la Telecomunicación
AUTOR: Adolfo Luzardo Cabrera
TUTORAS: Dra. Ernestina Martel Jordán
Dra. Carmen Nieves Ojeda Guerra

FECHA: Julio 2014

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

Aplicación Android para pedir cita previa en peluquerías

HOJA DE FIRMAS

Alumno:

Fdo: Adolfo Luzardo Cabrera

Tutora:

Fdo: Dra. Ernestina Martel Jordán

Tutora:

Fdo: Dra. Carmen Nieves Ojeda Guerra

Fecha: Julio 2014

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

Aplicación Android para pedir cita previa en peluquerías

HOJA DE EVALUACIÓN

Calificación: _____

Presidente:

Fdo:

Vocal:

Fdo:

Secretario/a:

Fdo:

Fecha: Julio 2014

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Parte I

Introducción

INTRODUCCIÓN

0.1 Introducción y antecedentes del trabajo

En 2013 se vendieron 225.326.200 unidades de teléfonos móviles según un informe publicado por Gartner [1], una compañía norteamericana especializada en la investigación de tecnología.

Dicho informe indica que los tres sistemas operativos preferidos por los usuarios son Android, iOS [2] y Windows Phone. Android es sin duda el sistema operativo más vendido con casi el 80 por ciento de la participación accionaria.

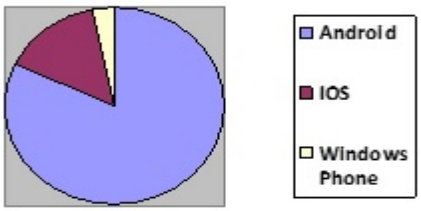

Figura 1: Gráfica con los sistemas operativos móviles más utilizados

Android es un sistema operativo pensado para teléfonos móviles, al igual que iOS y Windows Phone [3]. iOS es un sistema operativo propio para dispositivos de la compañía Apple Inc [4] que utiliza Objective-C como lenguaje de programación. Windows Phone pertenece a Microsoft y utiliza C como lenguaje de programación. En cambio, Android es una obra de Google y está basado en Linux [5], un núcleo de sistema operativo libre, gratuito y multiplataforma.

El sistema Android también permite programar aplicaciones en una variación de Java llamada Dalvik [6]. Este sistema operativo a diferencia de los anteriores es un sistema abierto que proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda, etc.) de una forma muy sencilla y utilizando un lenguaje de programación muy conocido como es Java.

Esta sencillez, junto a la existencia de herramientas de programación gratuitas, hace que una de las cosas más importantes de este sistema operativo sea la cantidad de aplicaciones disponibles.

Las aplicaciones para móviles abarcan diversos sectores. Aunque existen aplicaciones que permiten pedir cita a médicos o cita previa para renovar el DNI [7], no se han encontrado aplicaciones de cita previa en el sector de las peluquerías, sector en el que se centra el presente trabajo fin de grado.

0.2 Objetivos a conseguir

La mayoría de las peluquerías permiten pedir cita previa, pero utilizan métodos primitivos para la gestión de las mismas. Para pedir cita en dichas peluquerías necesitas llamar por teléfono o acudir presencialmente, lo que supone un trabajo añadido y un gasto de dinero tanto para el cliente como para el propio empresario.

Gracias al gran potencial de las aplicaciones móviles y al uso masivo en la población de los terminales móviles, se plantea una solución para la mejora en la gestión de las citas previas en las peluquerías. En este trabajo fin de grado se pretende realizar una aplicación móvil para que los clientes puedan pedir cita previa en la peluquería sin ningún tipo de coste, además de una base de datos online donde el empresario puede ver las citas previas.

Tomando como referencia el informe publicado por Gartner, se ha decidido que la aplicación para pedir cita en peluquerías se programará para móviles Android por ser el sistema operativo más utilizado y su lenguaje de programación nos permite realizar las funciones necesarias.

En este trabajo fin de grado se pretende crear una aplicación Android intuitiva y fácil de utilizar, compatible con la mayoría de los dispositivos Android. Como funciones principales destacan la posibilidad de pedir cita, gestionarlas y el almacenamiento de dichas citas en una base de datos online. Y como función extra la aplicación muestra datos de contacto e información de la peluquería.

0.3 Peticionario

El *Trabajo fin de grado* presentado en esta memoria ha sido elaborado a petición de la Escuela de Ingeniería de Telecomunicación y Electrónica, para la obtención del título de Graduado en Ingeniería en Tecnologías de la Telecomunicación.

0.4 Organización de la memoria

La memoria que analiza la información y desarrollo de este trabajo se divide en seis capítulos, el pliego de condiciones y el presupuesto. A continuación se comenta brevemente los capítulos incluidos:

- El capítulo 1 realiza una introducción a la tecnología propia de la programación de aplicaciones para móviles, los antecedentes del trabajo y especifica el objetivo principal del trabajo.
- El capítulo 2 amplía la información de la aplicación delimitando el sistema y buscando la funcionalidad que debe ofrecer desde la perspectiva del usuario final. Presenta los casos de uso y la interfaz de usuario de la aplicación.
- El capítulo 3 muestra las especificaciones detalladas de todos los objetos, incluyendo sus características y operaciones. Se introduce el lenguaje de programación utilizado.
- El capítulo 4 utiliza el resultado del capítulo anterior para generar el código final en el lenguaje de programación elegido.
- El capítulo 5 valida si el resultado es el que se quería, mediante una serie de pruebas sobre la interfaz de usuario y el código realizado.
- El capítulo 6 incluye una serie de conclusiones del trabajo final y las posibles mejoras del mismo.
- Por último, se adjunta el pliego de condiciones que especifica qué condiciones y requisitos ha de satisfacer la aplicación; y el presupuesto, que detalla los costes totales del software generado.

Parte II

Memoria

Capítulo **1**

MODELO DE REQUISITOS

1.1 Introducción

Un modelo, en el desarrollo de software, define cómo solucionar los problemas que aparecen en el desarrollo de una aplicación. Para desarrollar el software, existen diferentes metodologías, que definen los distintos tipos de modelos que se pueden encontrar en este proceso. Así, los modelos básicos son: **de Requisitos, de Diseño, de Implementación y de Pruebas** [10]. Asimismo, el desarrollo de software debe ser registrado a lo largo de todo el proceso, dando lugar a distintos documentos (**Documentación**). De todos los modelos indicados, en este capítulo se trataría el de requisitos, como primer modelo a definir en el desarrollo de la aplicación objeto de este trabajo fin de grado.

El **modelo de requisitos** tiene como objetivo delimitar el sistema y capturar la funcionalidad que debe ofrecer desde la perspectiva del usuario (es el contrato entre el desarrollador y el usuario final).

El modelo de requisitos consiste, básicamente, de tres modelos principales:

- **Modelo de comportamiento:** se basa directamente en el modelo de casos de uso y especifica la funcionalidad, desde el punto de vista del usuario. Tiene dos conceptos claves:
  - *Actores:* representan los distintos papeles que los usuarios pueden jugar en el sistema.
  - *Casos de uso:* representa qué pueden hacer los actores con respecto al sistema.
- **Modelo de presentación o de interfaces o de bordes:** especifica cómo interactúa el sistema con actores externos al ejecutar los casos de uso, es decir, especifica cómo se verán las interfaces gráficas y que función tiene cada una de ellas.
- **Modelo de información o del dominio del problema:** conceptualiza el sistema según los objetos que representan las entidades básicas de la aplicación.

El modelo de requisitos que se va a presentar en este capítulo, está basado en el *modelo de comportamiento* y en el *modelo de interfaces*. Asimismo, antes de realizar estos modelos, el desarrollador debe hacer una descripción detallada del problema (contrato con el usuario final).

1.2 Descripción del problema

La descripción del problema es una definición muy inicial de las necesidades que sirve como punto de partida para comprender los requisitos del sistema, es decir, debe ser una descripción de lo que se necesita y no una propuesta de solución.

En este trabajo fin de grado se pretende desarrollar una aplicación Android para pedir cita previa en peluquerías, tomando como ejemplo *Peluquerías Naranja*, una franquicia de peluquerías canarias. Esta aplicación permite, al usuario del dispositivo, reservar hora o pedir cita en la peluquería. La cita se reservará siguiendo una serie de pantallas, en las que la información introducida de forma táctil se almacenará posteriormente en una base de datos online. Asimismo, la aplicación será capaz de gestionar las citas previas disponibles permitiendo al usuario cancelar su cita previa.

La aplicación presentará una ventana principal con cuatro funciones: *Presentación de la peluquería*, en la que se describirá la peluquería brevemente; *Contacto*, donde se detallarán los datos de contacto de la empresa; *Pedir Cita*, donde se podrá pedir cita en la peluquería seleccionada, y *Gestión de citas*, donde se podrán gestionar las citas. Asimismo, también contará con un menú donde se podrá acceder a la configuración de la aplicación, ayuda, información y salir.

Una vez se ha pedido una cita con la aplicación, ésta queda almacenada en una base de datos online que la peluquería puede consultar y gestionar. La aplicación del terminal del usuario también tendrá acceso a dicha base de datos online, para poder gestionar las citas que ha realizado el usuario final.

1.3 Modelo de comportamiento

El **modelo de comportamiento** describe las diferentes formas de uso de un sistema, en el que cada uno de esas formas se conoce como caso de uso. Cada caso de uso se compone de una secuencia de eventos iniciada por el usuario. Para comprender los casos de uso del sistema, es necesario saber cómo los usuarios lo van a usar o actor (el actor no corresponde directamente con un usuario). Como se muestra en la figura 1.1, el actor y el caso de uso representan los dos elementos básicos de este modelo.

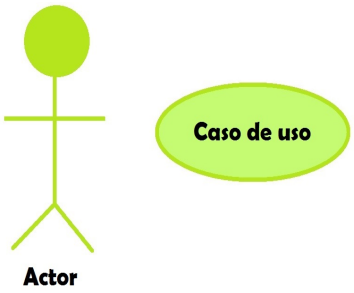

Figura 1.1: Representación de las entidades básicas para el modelo de comportamiento

1.3.1 Actores

Los **actores** [11] corresponden con el papel que un usuario puede jugar dentro de la aplicación (son entidades distintas a los usuarios). Los actores modelan cualquier entidad externa al sistema, además no están restringidos a ser personas físicas, pudiendo representar otros sistemas externos al actual. Cada uno de estos actores podrá ejecutar una o más tareas del sistema.

Los actores se identifican antes que los casos de uso, para que estos sean la herramienta principal para encontrar los casos de uso. Al definir todos los actores y los casos de uso se define la funcionalidad completa del sistema.

A la hora de definir los actores, se identifican primero aquellos que son la razón principal del sistema, conocidos como *actores primarios*, que son los que rigen la secuencia lógica de ejecución del sistema. Además existen otros actores que supervisan y mantienen el sistema, conocidos como *actores secundarios*. Estos corresponden, por lo general, a máquinas o sistemas externos.

Para especificar los actores de la aplicación, se pueden diferenciar los siguientes: el actor primario al que se le define como *Usuario*, que es el encargado de introducir los datos necesarios en la aplicación para que ésta le puedan proporcionar el servicio que proporciona, y varios actores secundarios, como son: el *Servidor de la base de datos externa*, que será el responsable de almacenar las citas previas; el *servidor Google*, que será el encargado de proporcionar los mapas, y el *GPS* que será el encargado de proporcionar la posición del usuario. En la figura 1.2 se presenta un diagrama representando al sistema como una caja cerrada y los diferentes actores como entidades externas a él.

1.3.2 Casos de uso

Después de haber definido los actores se define la funcionalidad del sistema a través de los casos de uso [11]. Cada caso de uso constituye un flujo completo de

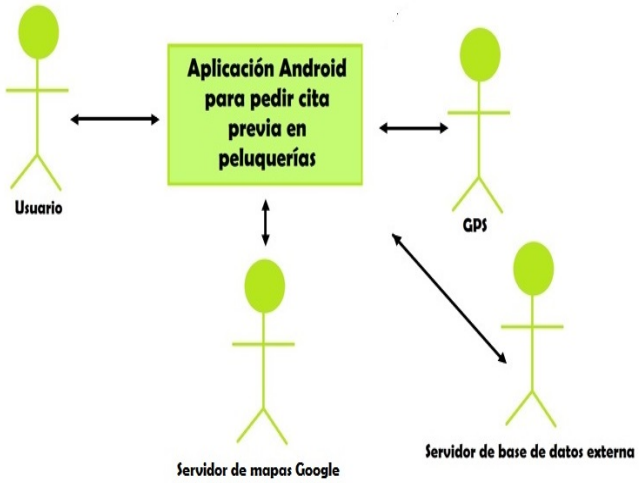

Figura 1.2: Delimitación del sistema para la *Aplicación Android para pedir cita previa en peluquerías*

eventos que muestra la interacción entre los actores y el sistema (es decir, qué van a hacer los actores). El actor primario es el encargado de empezar esta interacción y los casos de uso se muestran como respuesta al evento anterior. La ejecución del caso de uso acaba cuando algún actor genera un evento que requiere un caso de uso nuevo.

Aunque la idea es mantener el modelo de casos de uso lo más sencillo posible, existen ocasiones en los que la funcionalidad debe ser puesta en casos de uso separados o bien realizar una subdivisión del caso de uso. Existen dos enfoques para expresar variantes:

1. Si las diferencias entre los casos de uso son pequeñas, se pueden definir subflujos separados dentro de un mismo caso de uso.
2. Si las diferencias entre los casos de uso son grandes, se deben describir como casos de uso separados. Para estos casos de uso se utilizan principalmente las relaciones de inclusión, extensión y generalización.

Para la aplicación de este trabajo fin de grado la figura 1.3 refleja tres casos de uso principales (*Mostrar Peluquería*, *Pedir Cita* y *Mostrar citas*) y el resto se consideran casos de uso secundarios.

1.3.2.1. Inclusión

La **inclusión** se define como una sección de un caso de uso que es parte obligatoria del caso de uso básico. El caso de uso que se va a insertar depende del caso

de uso anterior. Como se muestra en la figura 1.3, la notación para inclusión es la etiqueta «include».

En la aplicación a desarrollar (figura 1.3), el caso de uso *Manejo de la base de datos externa* está incluido dentro de los casos de uso *Mostrar Peluquerías*, *Mostrar Horarios Disponibles*, *Mostrar citas* y *Gestión citas* porque todos ellos necesitan cargar o modificar datos de la base de datos externa. Asimismo, el caso de uso *Mostrar Mapa* también está incluido en *Mostrar Peluquerías* porque el mapa se muestra después de haber presentado la información de la peluquería seleccionada. Finalmente, los casos de uso *Mostrar peluquerías* y *Mostrar Horarios Disponibles* también están incluidos dentro del caso de uso *Pedir cita*, ya que este caso de uso se encargará de ofrecer al usuario la posibilidad de introducir datos, elegir peluquería y elegir un horario para poder posteriormente pedir cita.

1.3.2.2. Extensión

La **extensión** se utiliza para modelar secuencias de eventos opcionales de casos de uso que, al manejarse de manera independiente, pueden ser insertados o eliminados. Especifica cómo un caso de uso puede insertarse en otro para extender la función del anterior. El caso de uso donde se va a insertar la nueva funcionalidad debe ser independiente del caso de uso insertado. El caso de uso original se ejecuta hasta donde se inserta el nuevo caso de uso. Después de que este nuevo caso de uso haya terminado su función, el curso original de la secuencia continúa como si nada hubiera ocurrido.

Como se muestra en la figura 1.3, la notación para extensión es la etiqueta «extend». En esta figura se puede observar que el caso de uso *Mostrar Mapa* se extiende mediante el caso de uso *Mostrar localización usuario*, ya que una vez elegido el mapa se le da al usuario la libertad de poder ver un mapa con la peluquería más cercana a su posición. Otro ejemplo de extensión se observa en los casos de uso *Mostrar citas* y *Pedir citas* que extienden el caso de uso *Gestión de citas*, ya que en ambos se podrá realizar alguna acción sobre las citas.

1.3.2.3. Generalización

Una relación adicional entre casos de uso es la **generalización** que apoya la reutilización de los casos de uso. Mediante esta relación es necesario describir las partes similares una sola vez, en lugar de repetirlas para todos los casos de uso de comportamiento común. En la generalización hay dos tipos de casos de uso:

- A los casos de uso que se extraen se les llama casos de uso *abstractos*, ya que sirven para describir partes que son comunes a otros casos de uso (no se instancian).

- A los casos de uso que realmente son instanciados se les conoce como casos de uso *concretos*.

Las descripciones de los casos de uso abstractos se incluyen en las descripciones de los casos de uso concretos. Los casos de uso abstractos también pueden ser usados por otros casos de uso abstractos. Normalmente los comportamientos similares entre casos de uso se identifican después de describir los casos de uso, aunque en algunos casos es posible identificarlos antes. En la aplicación a desarrollar, no existen casos de uso que usen esta relación.

En la figura 1.3 se muestra el diagrama completo de los casos de uso para la aplicación a desarrollar. Los casos de uso principales son: *Mostrar Peluquerías, Pedir cita* y *Mostrar citas*. Además de éstos, los casos de uso adicionales son la inclusión o extensión de los casos de uso *Mostrar Horarios Disponibles, Gestión citas, Mostrar Mapa, Mostrar localización usuario* y *Manejo base de datos externa*.

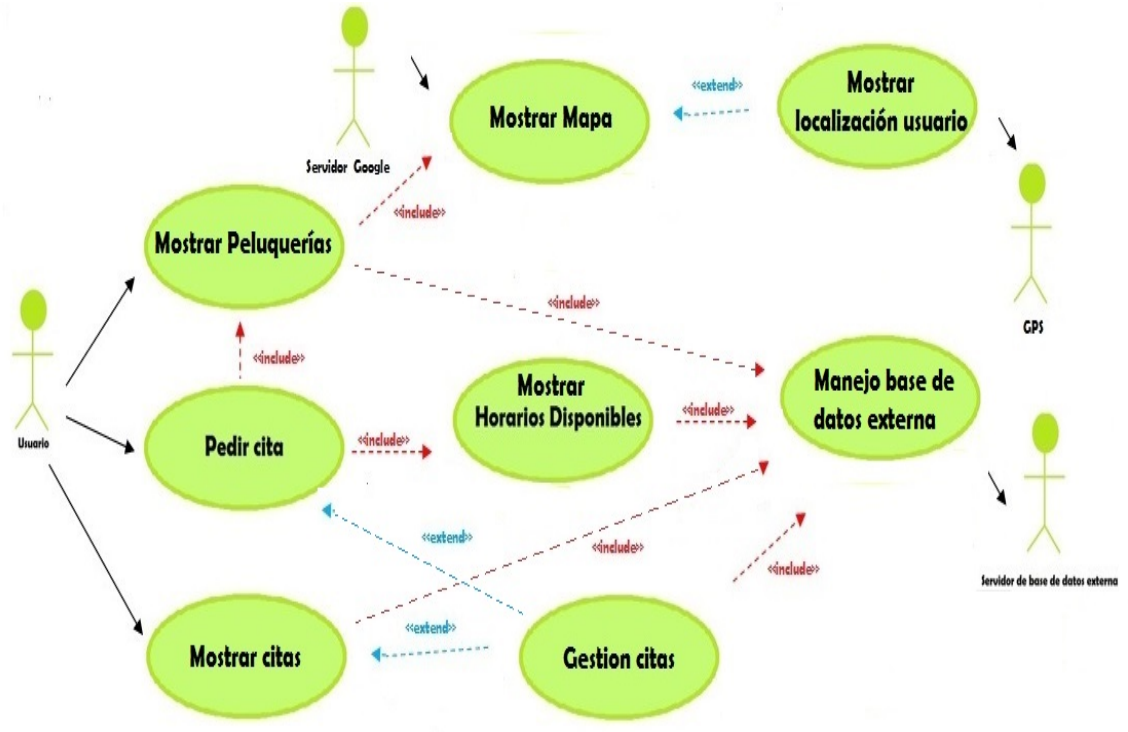

Figura 1.3: Diagrama de casos de uso para la *Aplicación Android para pedir cita previa en peluquerías*

1.4 Modelo de interfaces

El modelo de interfaces describe la presentación de información entre los actores y el sistema. Para ello, se especifica cómo se verán las interfaces de usuario al ejecutar

cada uno de los casos de uso, lo cual ayuda al usuario a visualizarlos según sean mostrados por el sistema. Cuando se diseñan las interfaces de usuario, es esencial tener a los usuarios involucrados, siendo de máxima importancia que las interfaces reflejen la visión lógica del sistema. En la sección 1.5.2 se muestran las interfaces usadas en la herramienta diseñada en este trabajo fin de grado.

1.5 Actores y casos de uso de la Aplicación Android para pedir cita previa en peluquerías

En esta sección se muestra la documentación de los actores y casos de uso, junto con el diseño de las interfaces, que serán usadas como prototipo del sistema.

1.5.1 Actores

Se describen un total de cuatro actores en la aplicación: *Usuario, Servidor de base de datos externa, Servidor Google* y *GPS*.

1.5.1.1. Usuario

Representa al usuario de la aplicación, es de tipo primario y participa en todos los casos de uso de la aplicación: *Mostrar Peluquerías*, *Pedir Cita*, *Mostrar Citas*, *Mostrar Horarios Disponibles*, *Gestión citas*, *Mostrar Mapa*, *Mostrar localización usuario* y *Manejo base de datos externa*

1.5.1.2. Servidor de la base de datos Externa

Representa el servidor externo donde existe una base de datos con las citas y los horarios disponibles, es de tipo secundario y participa en el caso de uso *Manejo de base de datos externa*.

1.5.1.3. Servidor Google

Representa el servidor de mapas de Google que nos proporciona los mapas utilizados en la aplicación, es de tipo secundario y participa en el caso de uso *Mostrar Mapa*.

1.5.1.4. GPS

Representa al servicio GPS para la geo-localización del usuario, es de tipo secundario y participa en el caso de uso *Mostrar localización usuario*.

1.5.2 Casos de Uso

Se describen un total de ocho casos de uso en la aplicación: *Mostrar Peluquerías*, *Pedir cita*, *Mostrar citas*, *Mostrar Horarios Disponibles*, *Gestión citas*, *Mostrar Mapa* ,*Mostrar localización usuario* y *Manejo base de datos externa*.

1.5.2.1. Caso de uso *Mostrar Peluquerías*

- **Actores:** *Usuario*.
- **Propósito:** mostrar al usuario final información de la peluquería elegida, imágenes y la posibilidad de ver un mapa con la ubicación de la misma.
- **Precondiciones:** el usuario debe haber presionado el botón **¿Dónde estamos?**, del menú principal.
- **Flujo principal:** Se presenta al usuario la vista principal, figura 1.4 donde el usuario selecciona la opción **¿Dónde estamos?** En este punto, se inicia este caso de uso representado en la figura 1.5 (E-1).
- **Flujo secundario:** una vez dentro de este caso de uso, figura 1.5, se puede diferenciar dos flujos secundarios. El primer flujo secundario es la carga de los datos de la base de datos externa acción del caso de uso *Manejo base de datos externa* y el segundo flujo secundario es la vista de un mapa con la ubicación de la peluquería gracias al caso de uso *Mostrar Mapa* que se inicia al presionar el botón **Ver en mapa**.
- **Excepciones:**
  - E-1 (Fallo al conectar con el servidor de base de datos externa): debe de conectarse al servidor externo para cargar los datos de las peluquerías.

Figura 1.4: Vista principal propuesta de la aplicación

Figura 1.5: Vista donde se muestra la peluquería elegida

1.5.2.2. Caso de uso *Mostrar Mapa*

- **Actores:** *Usuario* y *Servidor Google*.
- **Propósito:** mostrar al usuario un mapa con la ubicación de la peluquería previamente seleccionada.

- **Precondiciones:** el usuario debe tener una peluquería seleccionada y presionar en el botón **Ver en mapa** que se encuentra en la vista de la figura 1.5.
- **Flujo principal:** se presenta al usuario la vista principal (figura 1.4), donde el usuario selecciona la opción **¿Dónde estamos?** En este punto se inicia el caso de uso *Mostrar peluquerías* y se presenta la vista de la figura 1.5. En esta vista, si el usuario selecciona el botón **Ver en mapa** se inicia el caso de uso *Mostrar mapa* (E-1), representado en la figura 1.6.
- **Flujo secundario:** si el usuario presiona el botón **Peluquería más cercana**, se cambia el mapa y se muestra la peluquería más cercana a la posición del usuario, con la ayuda del caso de uso *Mostrar localización usuario*.
- **Excepciones:**
  - E-1 (Fallo al conectar con el servidor Google): debe de conectarse al servidor Google para cargar el mapa.

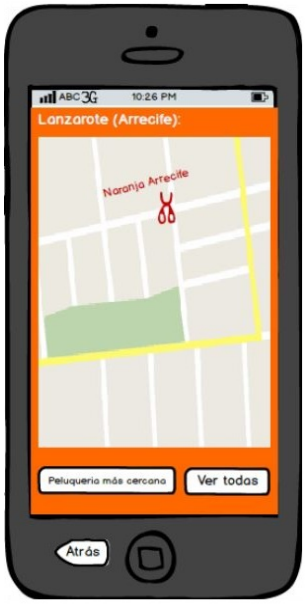

Figura 1.6: Vista del mapa con la peluquería elegida

1.5.2.3. Caso de uso *Mostrar localización usuario*

- **Actores:** *Usuario* y *GPS*.
- **Propósito:** obtener la localización geográfica del usuario.
- **Precondiciones:** el usuario debe presionar en el botón **Peluquería más cercana** que se encuentra dentro del caso de uso *Mostrar mapa* (figura 1.6).

- **Flujo principal:** se presenta al usuario la vista principal (figura 1.4) donde el usuario selecciona la opción **¿Dónde estamos?**, en este paso se inicia el caso de uso *Mostrar peluquerías* (figura 1.5). En esta vista, si el usuario selecciona el botón **Ver en mapa** se inicia el caso de uso *Mostrar mapa*, representado en la figura 1.6. Finalmente, al presionar el botón **Peluquería más cercana**, se inicia este caso de uso, de forma que se pedirá al dispositivo GPS (E-1), la localización del dispositivo del usuario.
- **Flujo secundario:** si el usuario presiona el botón **Ver todas** se muestra un mapa con todas las peluquerías.
- **Excepciones:**
  - E-1 (Fallo al conectar con el GPS): debe de conectar con el GPS para saber la geo-localización del usuario.

1.5.2.4. Caso de uso *Pedir cita*

- **Actores:** *Usuario.*
- **Propósito:** recoger los datos necesarios y pedir cita previa en la peluquería seleccionada.
- **Precondiciones:** ninguna.
- **Flujo principal:** se presenta al usuario la vista principal (figura 1.4). En esta vista, el usuario selecciona el botón **Pedir cita**, iniciando este caso de uso y apareciendo la vista de la figura 1.7, donde el usuario deberá introducir los datos solicitados (E-1).
- **Flujo secundario:** se pueden considerar dos flujos secundarios. Un flujo a través del cual se cargan las peluquerías utilizando el caso de uso *Mostrar peluquerías* y otro que carga los horarios disponibles utilizando el caso de uso *Mostrar Horarios Disponibles.*
- **Excepciones:**
  - E-1 (Fallo al introducir datos): debe introducir correctamente todos los datos (con el formato adecuado).

1.5.2.5. Caso de uso *Mostrar Horarios Disponibles*

- **Actores:** *Usuario* y *Servidor de base de datos externa.*
- **Propósito:** recoger de la base de datos externa y mostrar en pantalla los horarios disponibles en la peluquería seleccionada.

Figura 1.7: Vistas correspondientes a introducir datos para pedir cita

- **Precondiciones:** el usuario debe haber seleccionado una peluquería dentro del caso de uso *Pedir cita* y encontrarse en la ventana donde se introduce el horario.
- **Flujo principal:** se presenta al usuario la vista principal (figura 1.4). En esta vista, el usuario selecciona el botón **Pedir cita**, iniciando el caso de uso *Pedir cita* (vista de la figura 1.7). Como se puede observar en esta figura, en la última vista se debe introducir un horario que ha sido previamente cargado gracias al caso de uso *Mostrar Horarios Disponibles*.
- **Flujo secundario:** el flujo secundario de este caso de uso *Mostrar Horarios Disponibles*, es el encargado de cargar los horarios de la base de datos externa y es posible gracias a la ayuda del caso de uso *Manejo base de datos externa*.
- **Excepciones:** ninguna.

1.5.2.6. Caso de uso *Mostrar citas*

- **Actores:** *Usuario*.
- **Propósito:** mostrar una lista con las citas previas que ha pedido el usuario de la aplicación.

- **Precondiciones:** el usuario debe haber pedido previamente una cita y haber presionado el botón **Mis citas**.
- **Flujo principal:** se presenta al usuario la vista principal (figura 1.4). En esta vista, el usuario selecciona el botón **Mis citas**, iniciando este caso de uso representado en la figura 1.8.
- **Flujo secundario:** existen dos flujos secundarios, uno que carga las citas previamente solicitadas gracias a la ayuda del caso de uso *Manejo base de datos externa* y otro al que se accede presionando en una cita y que permite cancelarla, iniciando el caso de uso *Gestión Citas*.
- **Excepciones:** ninguna.

Figura 1.8: Vista con el listado de citas para la posterior gestión de las mismas

1.5.2.7. Caso de uso *Gestión citas*

- **Actores:** *Usuario*.
- **Propósito:** es el encargado de gestionar las citas, añadir o eliminar citas, de la base de datos externa.
- **Precondiciones:** ninguna.
- **Flujo principal:** se presenta al usuario la vista principal (figura 1.4). En esta vista, el usuario selecciona el botón **Mis citas**, iniciando el caso de uso *Mostrar citas*, representado en la figura 1.8. Al presionar en cualquier cita de la lista, aparece la opción de **Cancelar la cita**, como se puede observar en la

figura 1.9. Al presionar el botón **Cancelar cita**, se inicia este caso de uso que elimina la cita de la base de datos (con la ayuda del caso de uso *Manejo base de datos externa*).

Figura 1.9: Vista de la aplicación que permite cancelar una cita

También se puede acceder a este caso de uso si en la vista principal, figura 1.4, el usuario selecciona el botón **Pedir cita**. Cuando se introducen todos los datos necesarios se puede confirmar la cita tal y como se representa en la figura 1.10. Si se presiona el botón **Confirmar reserva** se inicia este caso de uso para añadir la cita a la base de datos externa.

- **Flujo secundario:** ninguno.
- **Excepciones:** ninguna.

1.5.2.8. Caso de uso *Manejo base de datos externa*

- **Actores:** *Usuario* y *Servidor de base de datos externa*.
- **Propósito:** manejar los datos de la base de datos externa, es decir, escribir, leer y borrar datos en el servidor externo.
- **Precondiciones:** se necesita interactuar con la aplicación ya sea mostrando información de una peluquería, pidiendo una cita previa o mostrando las citas previas.
- **Flujo principal:** este caso de uso tiene varios flujos principales dependiendo del uso que se le de a la aplicación:

Figura 1.10: Vista de la aplicación que permite confirmar una cita

Si se desea mostrar una peluquería: se presenta al usuario la vista principal (figura 1.4). El usuario selecciona el botón **¿Dónde estamos?**, en este paso se inicia el caso de uso *Mostrar Peluquerías*, representado en la figura 1.5. En este punto, se inicia el caso de uso *Manejo base de datos externa* para cargar la lista de peluquerías y la información de la peluquería seleccionada (E-1).

Si se desea pedir cita previa: se presenta al usuario la vista principal (figura 1.4). El usuario selecciona el botón **Pedir cita**, en este paso se inicia el caso de uso *Pedir cita*, cuya finalidad principal es pedir cita previa que se realiza iniciando el caso de uso *Manejo base de datos externa*, para escribir en la base de datos externa (E-1).

Si se desea mostrar las citas previas: se presenta al usuario la vista principal (figura 1.4). El usuario selecciona el botón **Mis citas**, en este paso se inicia el caso de uso *Mostrar citas* cuya finalidad principal es mostrar las citas previas y para ello se debe iniciar el caso de uso *Manejo base de datos externa*, para leer las citas de la base de datos externa (E-1).

- **Flujo secundario:** este caso de uso también puede ser iniciado por los casos de uso *Mostrar Horarios Disponibles* con el objetivo de leer los horarios disponibles en la base de datos externa y *Gestión citas* con la finalidad de borrar una cita de la base de datos externa (E-1).

- **Excepciones:**

  - E-1 (Fallo al conectar con la base de datos externa): debe conectar con el servidor externo para solicitar, borrar o escribir datos en la base de datos externa.

1.6 Clases e interfaces de la vista

Teniendo en cuenta que a la hora de implementar la *Aplicación Android para pedir cita previa en peluquerías* se utilizará la arquitectura, MVP (*Modelo-Vista-Presentador*), que es una variante de la arquitectura MVC, se detallan a continuación, las clases e interfaces más importantes que corresponden con la parte de la vista, explicando cada método y los parámetros de éstos, tal y como representa la figura 1.11.

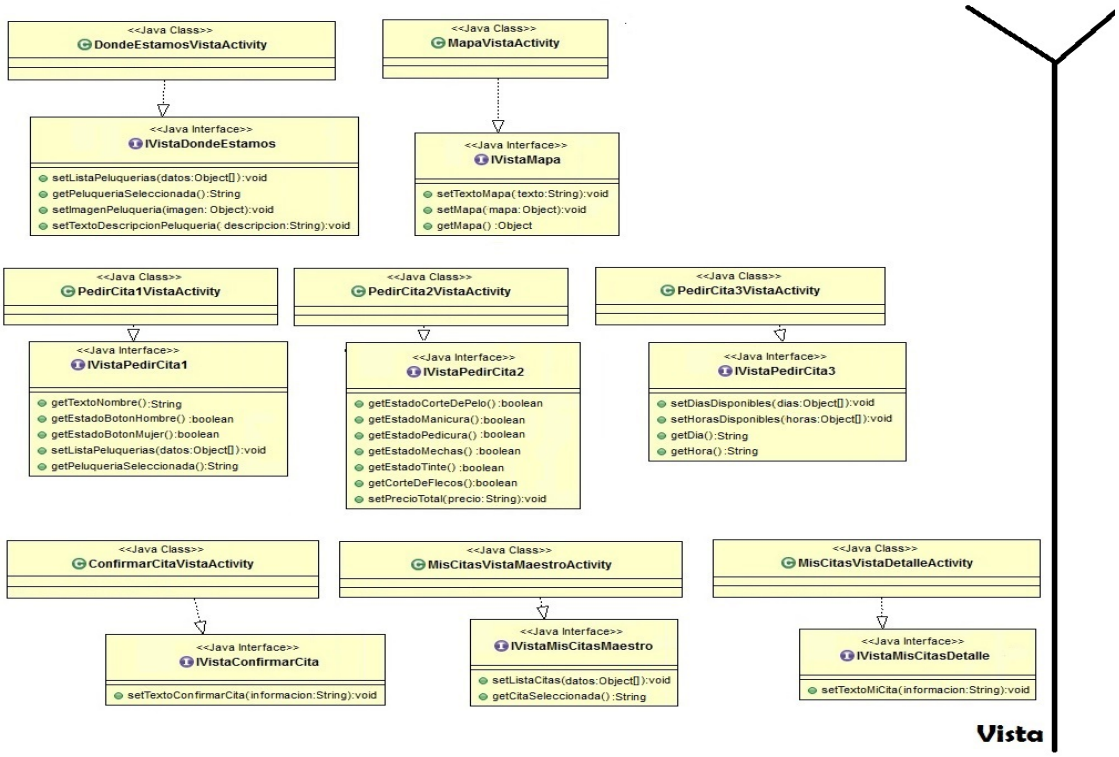

Figura 1.11: Clases de la vista de la aplicación con sus interfaces

1.6.1 Clase DondeEstamosVistaActivity

Vista correspondiente al caso de uso *Mostrar Peluquerías*, en la que el usuario debe seleccionar una peluquería y seguidamente se muestra una foto y una descripción de la peluquería seleccionada, como se observa en la figura 1.5.

Esta clase implementa la interfaz *IVistaDondeEstamos*, la cual aparece representada en el figura 1.11 con los siguientes métodos:

- **setListaPeluquerias(datos: Object[]): void**. Método que rellenará una lista con el listado de peluquerías que entra por parámetros.

- **getPeluqueriaSeleccionada(): String**. Método que obtiene la peluquería que el usuario ha seleccionado en la lista de peluquerías de la vista.
- **setImagenPeluqueria(imagen: Object): void**. Método encargado de cargar la imagen de la peluquería en la vista.
- **setTextoDescripcionPeluqueria(descripcion: String): void**. Método que actualiza la descripción de la peluquería.

1.6.2 Clase MapaVistaActivity

Vista en la cual aparece un mapa con un campo de texto en la parte superior y botones en la parte inferior, como se puede ver en la figura 1.6. La vista se mostrará cuando el usuario quiera ver un mapa con la ubicación de una peluquería.

Esta clase implementa la interfaz *IVistaMapa*, la cual aparece representada en el figura 1.11 con los siguientes métodos:

- **setTextoMapa(texto: String):void**. Método que actualizará el texto de la parte superior del mapa.
- **setMapa(mapa: Object): void**. Método encargado de actualizar el mapa en la vista.
- **getMapa(): Object**. Método que obtiene el mapa la vista.

1.6.3 Clase PedirCita1VistaActivity

Primera vista donde el usuario debe introducir datos necesarios para posteriormente generar una petición de cita previa. Algunos de los datos que se recogen en esta vista son el nombre, el sexo y la peluquería a la que desea acudir.

Esta clase implementa la interfaz *IVistaPedirCita1*, la cual aparece representada en el figura 1.11 con los siguientes métodos:

- **getTextoNombre(): String**. Método que recoge el nombre del usuario.
- **getEstadoBotonHombre(): boolean**. Método que recoge el estado del botón **Hombre** (si está seleccionado el usuario es un hombre).
- **getEstadoBotonMujer(): boolean**. Método que recoge el estado del botón **Mujer** (si está seleccionado el usuario es una mujer).
- **setListaPeluquerias(datos: Object[]): void**. Método que rellenará una lista con el listado de peluquerías que entra por parámetros.
- **getPeluqueriaSeleccionada(): String**. Método que obtiene la peluquería que el usuario ha seleccionado en la lista de peluquerías de la vista.

1.6.4 Clase PedirCita2VistaActivity

Segunda vista donde el usuario debe introducir datos necesarios para posteriormente generar una petición de cita previa. En esta vista se recogen los servicios que desea el usuario contratar en la peluquería.

Esta clase implementa la interfaz *IVistaPedirCita2*, la cual aparece representada en el figura 1.11 con los siguientes métodos:

- **getEstadoCorteDePelo(): boolean**. Método que recoge el estado del servicio corte de pelo. Si esta seleccionado el usuario quiere contratar dicho servicio.
- **getEstadoManicura(): boolean**. Método que recoge el estado del servicio manicura. Si esta seleccionado el usuario quiere contratar dicho servicio.
- **getEstadoPedicura(): boolean**. Método que recoge el estado del servicio pedicura. Si esta seleccionado el usuario quiere contratar dicho servicio.
- **getEstadoMechas(): boolean**. Método que recoge el estado del servicio mechas. Si esta seleccionado el usuario quiere contratar dicho servicio.
- **getEstadoTinte(): boolean**. Método que recoge el estado del servicio tinte. Si esta seleccionado el usuario quiere contratar dicho servicio.
- **getCorteDeFlecos(): boolean**. Método que recoge el estado del servicio corte de flecos. Si esta seleccionado el usuario quiere contratar dicho servicio.
- **setPrecioTotal(precio: String): void**. Método que actualizará el precio total según lo que el usuario haya contratado.

1.6.5 Clase PedirCita3VistaActivity

Vista donde el usuario debe elegir una fecha y una hora en la que quiere acudir a la peluquería.

Esta clase implementa la interfaz *IVistaPedirCita3* la cual aparece representada en el figura 1.11 con los siguientes métodos:

- **setDiasDisponibles(dias: Object[]): void**. Método que actualizará en la vista los días disponibles para pedir cita en la peluquería seleccionada.
- **setHorasDisponibles(horas: Object[]): void**. Método que actualizará en la vista las horas disponibles para pedir cita en la peluquería seleccionada.
- **getDia(): String**. Método que recoge el día que el usuario ha seleccionado en la vista.
- **getHora(): String**. Método que recoge la hora que el usuario ha seleccionado en la vista.

1.6.6 Clase ConfirmarCitaVistaActivity

Vista en la que aparece un texto con todos los datos recopilados de la cita previa, donde el usuario debe confirmar todos los datos que ha introducido y seguidamente se genera una petición de cita previa.

Esta clase implementa la interfaz *IVistaConfirmarCita*, la cual aparece representada en el figura 1.11 con el método:

- **setTextoConfirmarCita(informacion: String): void**. Método que actualizará en la vista el texto que aparece con todos los datos que el usuario introdujo previamente.

1.6.7 Clase MisCitasVistaMaestroActivity

Vista en la que aparece un listado con todas las citas que el usuario ha pedido previamente con dicha aplicación.

Esta clase implementa la interfaz *IVistaMisCitasMaestro*, la cual aparece representada en el figura 1.11 con los siguientes métodos:

- **setListaCitas(datos: Object[]): void**. Método que actualizará en la vista, la lista con todas las citas que el usuario ha pedido previamente.
- **getCitaSeleccionada(): String**. Método que recoge la cita que el usuario ha seleccionado.

1.6.8 Clase MisCitasVistaDetalleActivity

Vista donde aparecen los detalles de la cita seleccionada y un botón para cancelar la reserva en la parte inferior, por si desea cancelar la cita.

Esta clase implementa la interfaz *IVistaMisCitasDetalle*, la cual aparece representada en el figura 1.11 con el método:

- **setTextoMiCita(informacion: String): void**. Método que actualizará en la vista el texto que aparece con los datos de la cita que el usuario ha seleccionado previamente.

Capítulo **2**

MODELO DE DISEÑO

2.1 Arquitectura Modelo-Vista-Presentador

La arquitectura *Modelo-Vista-Presentador* (MVP) [12] separa el modelo, la presentación y las acciones basadas en la interacción con el usuario en tres clases separadas. La vista le delega a su presentador toda la responsabilidad del manejo de los eventos del usuario. El presentador se encarga de actualizar el modelo cuando surge un evento en la vista, pero también es responsable de actualizar a la vista cuando el modelo le indica que ha cambiado. Por su parte, el modelo no conoce la existencia del presentador, por lo tanto, si el modelo cambia por acción de algún otro componente que no sea el presentador, debe "disparar" un evento para que el presentador se entere. Como se puede apreciar en la figura 2.1, a la hora de implementar esta arquitectura, se identifican los siguientes componentes:

- **IVista:** es la interfaz con la que el *Presentador* se comunica con la vista.
- **Vista:** vista que implementa la interfaz *IVista* y se encarga de manejar los aspectos visuales. Mantiene una referencia a su *Presentador*, al cual le delega la responsabilidad del manejo de los eventos.
- **Presentador:** contiene la lógica para responder a los eventos y manipula el estado de la vista mediante una referencia a la interfaz *IVista*. El Presentador utiliza el modelo para saber cómo responder a los eventos y es responsable de establecer y administrar el estado de una vista.
- **Modelo:** está compuesto por los objetos que conocen y manejan los datos dentro de la aplicación.

2.2 Diagramas de secuencia

A la hora de establecer qué métodos se necesitan en el modelo y en el presentador, se debe partir del prototipo de la interfaz de usuario presentado en el capítulo *Modelo de requisitos*. Analizando los posibles usos que tiene la aplicación, se obtienen

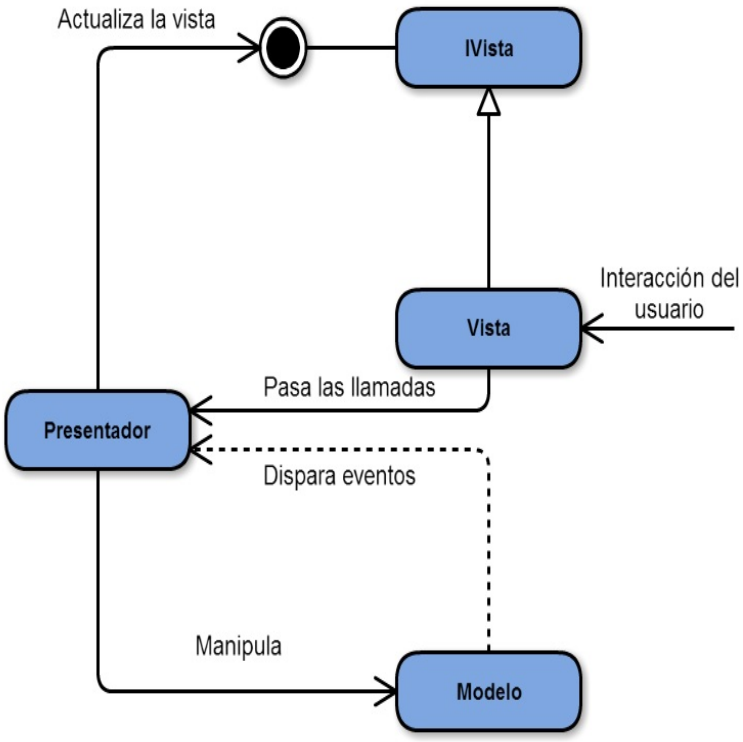

Figura 2.1: Diagrama de la arquitectura*Modelo-Vista-Presentador*

unos diagramas de secuencia determinados. Los diagramas de secuencia describen el uso de la aplicación según los eventos enviados entre los objetos de la arquitectura *MVP*. El diagrama de secuencia describe aspectos dinámicos de un sistema, a diferencia de los diagramas de clases que muestran información estática. Por esta razón, los diagramas de secuencias utilizan objetos mientras que los diagramas de clases utilizan clases como elementos básicos.

Cada objeto en el diagrama se representa con una línea vertical, que corresponde al eje temporal, donde el tiempo avanza hacia abajo. En este diagrama se muestran los eventos que ocurren en el tiempo, los cuales son enviados de un objeto a otro. El orden de los objetos no es importante. Lo importante es el orden en el que ocurren los eventos y la dependencia entre ellos, es decir, qué consecuencias tiene el envío de un evento.

Los mensajes enviados entre objetos corresponden con los métodos que hay que definir en las interfaces implementadas por la clases de esos objetos. Así, un mensaje enviado entre un objeto vista y un objeto presentador se representa mediante una flecha que va desde la línea vertical del objeto vista, hacia la línea vertical del objeto presentador, y tiene un identificador que corresponde con el nombre del método. En el diagrama de secuencia no se muestran los datos enviados o recibidos, sino que sólo se muestran los identificadores de los mensajes enviados. En los siguientes apartados se analizan detalladamente cada uno de estos mensajes y se indican los parámetros y el tipo de datos que devuelven (si es el caso).

En este apartado se presentan los diagramas de secuencia más significativos de la *Aplicación Android para pedir cita previa en peluquerías*. Así, en la figura 2.2 se muestra el flujo de mensajes que se producen entre la vista, el presentador y el modelo cuando el usuario quiere obtener información de una peluquería. En la secuencia se ha supuesto el caso de uso *Mostrar Peluquerías*.

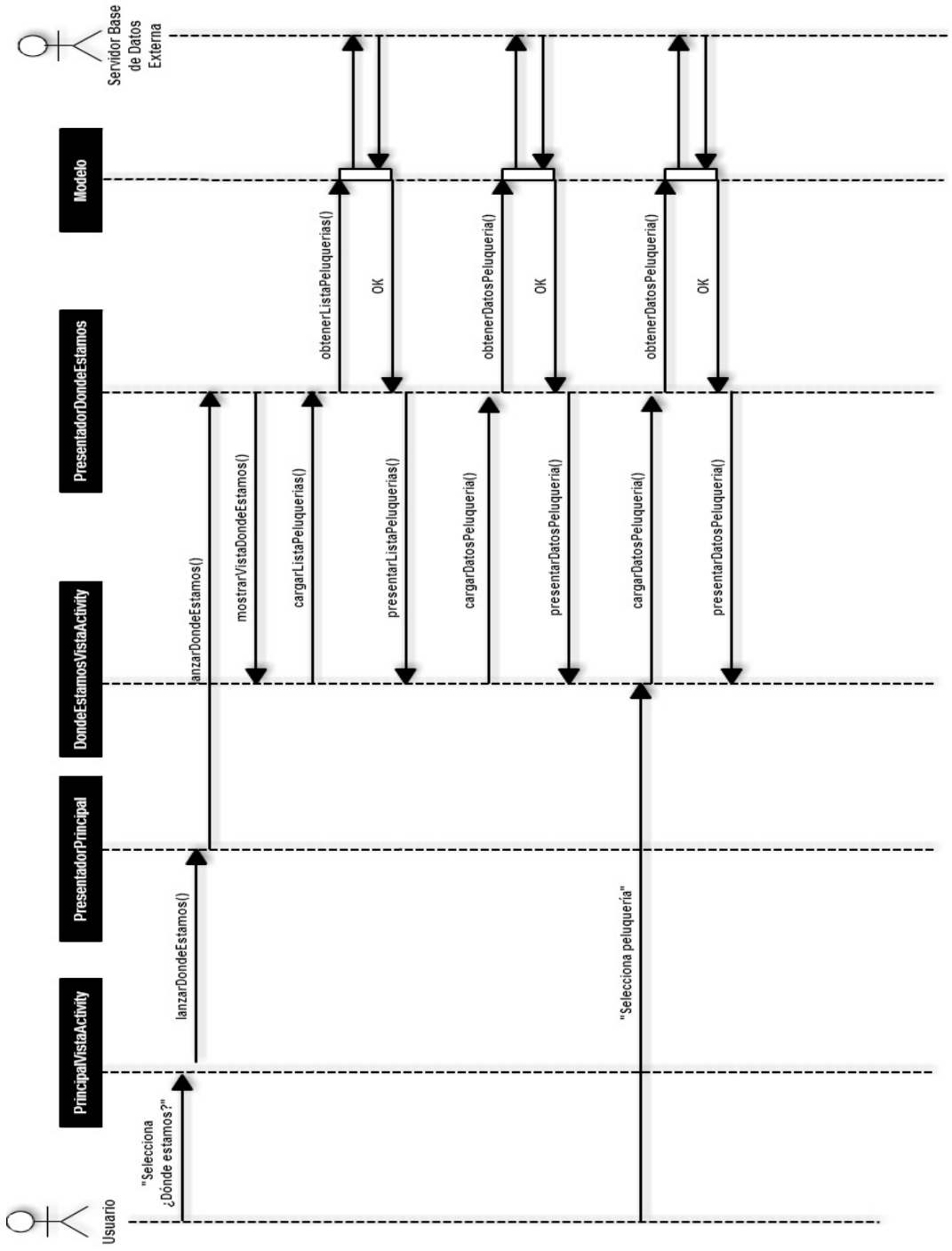

Figura 2.2: Diagrama de secuencia del caso de uso *Mostrar Peluquerías*

En la figura 2.3 se muestra el flujo de mensajes que se producen entre las vistas, presentadores y el modelo cuando el usuario, una vez elegida la peluquería, desea ver un mapa con la posición de dicha peluquería.

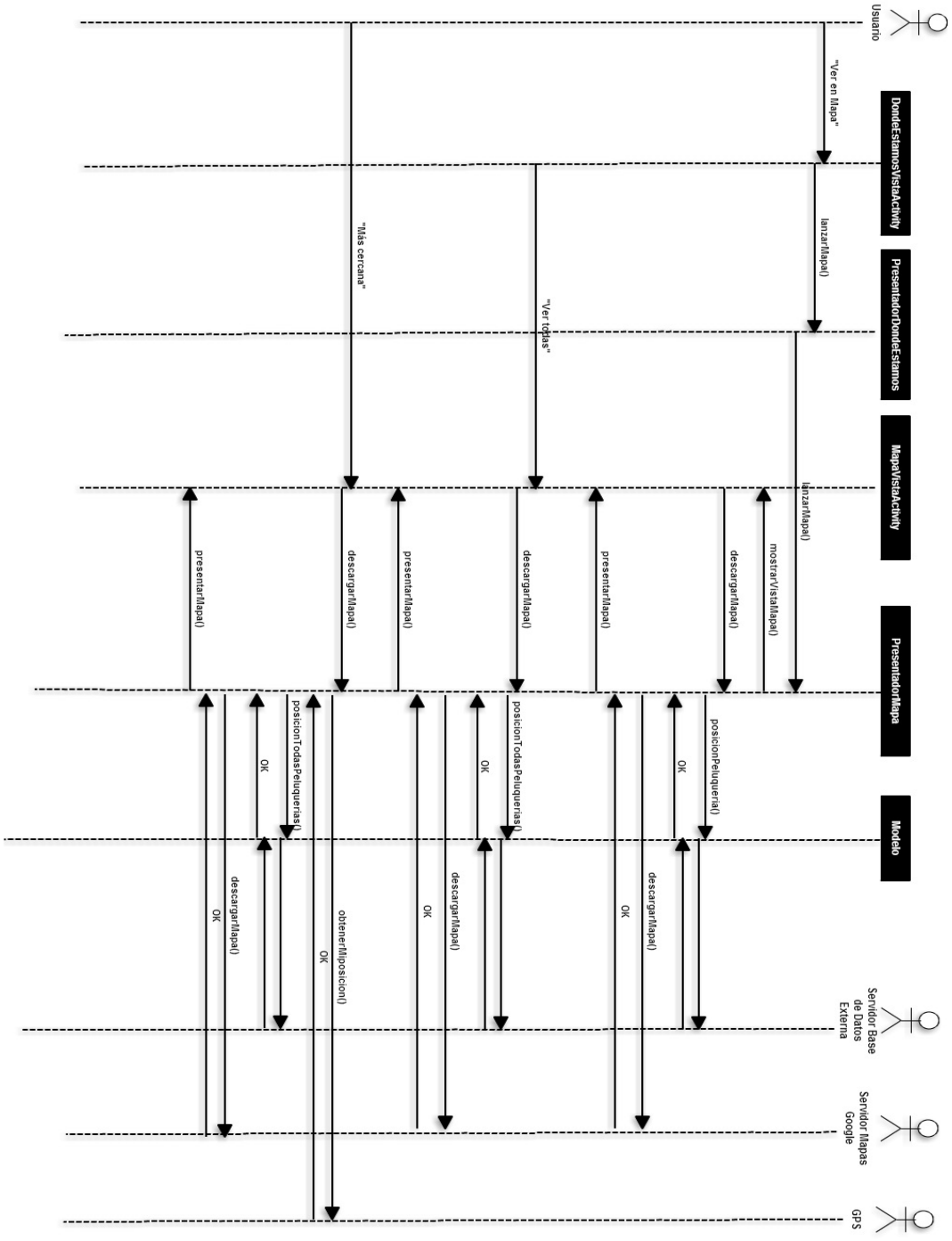

Figura 2.3: Diagrama de secuencia del caso de uso *Mostrar Mapa*

En la figura 2.4 se muestra el flujo de mensajes que se producen entre las vistas, presentadores y el modelo cuando el usuario intenta pedir una cita a través de la

aplicación. Éste es un diagrama de secuencia muy importante puesto que la finalidad principal de la aplicación es pedir cita.

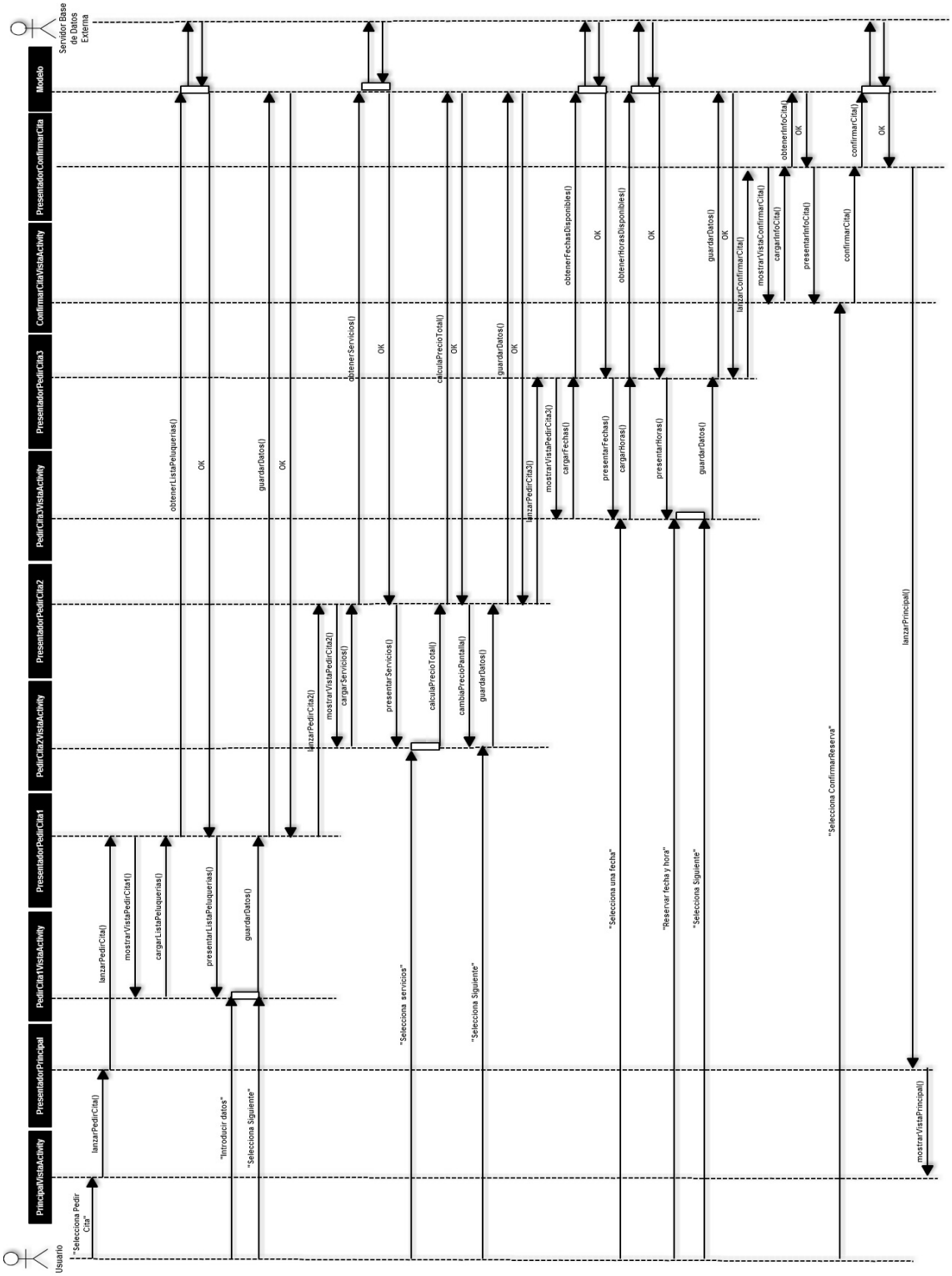

Figura 2.4: Diagrama de secuencia del caso de uso *Pedir Cita*

Por último, en la figura 2.5 se ve el flujo de mensajes del caso de uso *Mostrar Citas* que es el encargado de mostrar las citas que ha pedido el usuario mediante la aplicación.

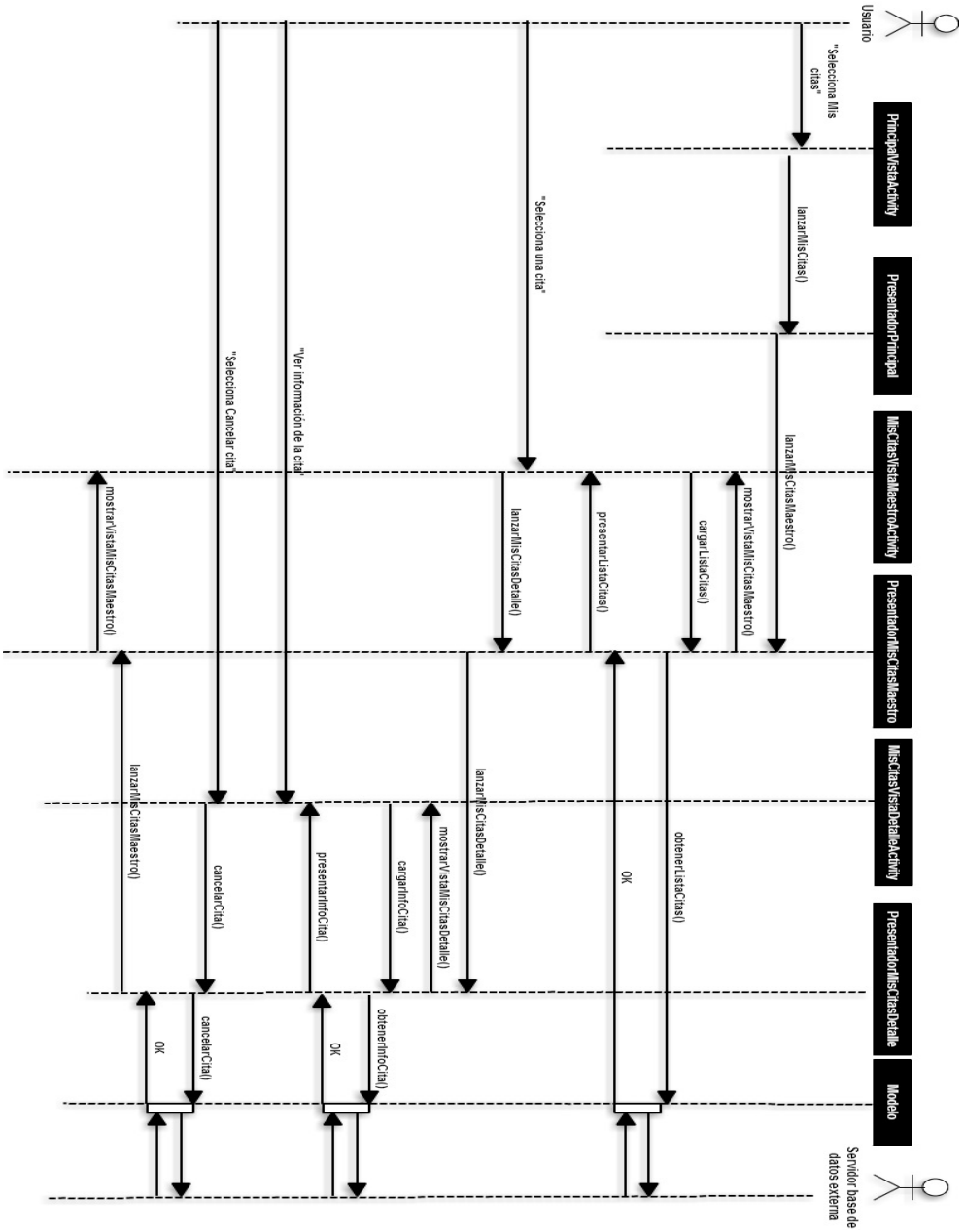

Figura 2.5: Diagrama de secuencia del caso de uso *Mostrar Citas*

2.3 **Modelo de datos**

Un modelo de datos es la descripción de la información que se utiliza en una aplicación. En el caso de la *Aplicación Android para pedir cita previa en peluquerías*,

la información se almacenará en la nube (base de datos externa) ya que ésta debe estar disponible para todos los usuarios de la aplicación. Para que la información esté disponible en cualquier momento y en cualquier lugar, debe estar almacenada en un servidor diseñado y mantenido por el desarrollador o por una entidad externa (opción escogida). Las bases de datos en la nube pueden estar basadas en *SQL* o utilizar un modelo de datos *NoSQL*. Concretamente, en este trabajo se utilizará una base de datos en la nube *NoSQL* que se caracteriza, entre otras cosas, por la ausencia de esquema, es decir, no se diseñan las tablas ni la estructura de los datos por adelantado [13]. Sin embargo, y como se conoce el tipo de información a almacenar, se puede realizar una descripción de la estructura de los datos y sus tipos. Así, para el diseño de la base de datos que utilizará la *Aplicación Android para pedir cita previa en peluquerías*, se presentan a continuación una serie de tablas en las que se definen:

- **Campo:** en donde se definirá su nombre.
- **Clave:** definirá si el campo es clave primaria o foránea (en este caso se indicará entre corchetes a qué otra tabla hace referencia en la forma *ClaveForánea [TablaExterna]*). Aunque el concepto de clave no tiene significado en una base de datos *NoSQL*, si es importante de cara a la aplicación desarrollada.
- **Tipo:** definirá el tipo del campo, y si acepta o no, que el campo esté vacío. Por defecto no existirán campos nulos, a excepción que se indique en las tablas (con la palabra *null*).

2.3.1 Diseño de la base de datos

La base de datos estará compuesta por un total de seis tablas: **Peluquerias, Horarios, Festivos, Citas, Servicios** y **CitaServicio**. A continuación se procede a definir cada tabla de la base de datos y sus campos como se puede ver en la figura 2.6.

2.3.1.1. Entidad Peluquerias

Entidad que contiene las peluquerías que permitan pedir cita a través de la *Aplicación Android para pedir cita previa en peluquerías*. Esta entidad cuenta con seis campos:

- **id_peluqueria:** es un campo de tipo *String* y clave primaria. Almacena un identificador único de la peluquería.
- **direccion:** es un campo de tipo *String* y almacena la dirección de la peluquería.

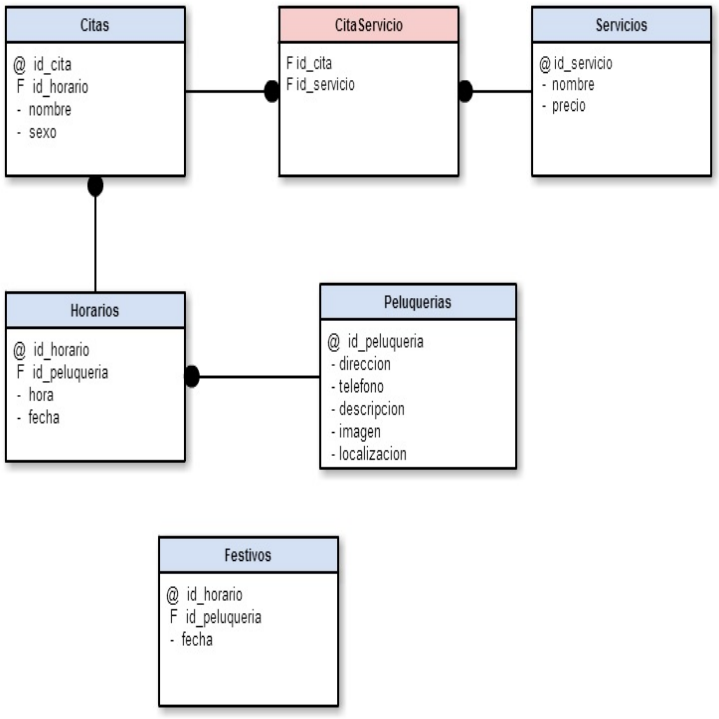

Figura 2.6: Diagrama de las tablas de la base de datos con sus campos

- **telefono:** es un campo de tipo *String* y almacena el teléfono de contacto de la peluquería.
- **descripcion:** es un campo de tipo *String* y almacena una breve descripción de la peluquería.
- **imagen:** es un campo de tipo *String* y almacena una URL donde se encuentra una imagen representativa de la peluquería.
- **localizacion:** es un campo tipo *String* y almacena las coordenadas geográficas exactas donde podemos encontrar la peluquería.

2.3.1.2. Entidad Horarios

Entidad que contiene los horarios que han reservado los usuarios de la *Aplicación Android para pedir cita previa en peluquerías*. Esta entidad cuenta con cuatro campos:

- **id_horario:** es un campo de tipo *String* y clave primaria. Almacena un identificador único para cada horario guardado.

- **id_peluqueria:** es un campo de tipo *String* y clave *Foránea [Peluquerias]*. Almacena el identificador de la peluquería en la que está ocupado el horario.
- **hora:** es un campo de tipo *String* y almacena el la hora que está ocupada.
- **fecha:** es un campo de tipo *String* y almacena la fecha que está ocupada.

2.3.1.3. Entidad Festivos

Entidad que contiene los dias festivos y días en los que no abre la peluquería. Esta entidad cuenta con tres campos:

- **id_horario:** es un campo de tipo *String* y clave primaria. Almacena un identificador único para cada horario guardado.
- **id_peluqueria:** es un campo de tipo *String* y clave *Foránea [Peluquerias]*. Almacena el identificador de la peluquería en la que está ocupado el horario.
- **fecha:** es un campo de tipo *String* y almacena la fecha festiva o en la que cierra la peluquería.

2.3.1.4. Entidad Citas

Entidad que contiene las citas que los usuarios han pedido a través de la *Aplicación Android para pedir cita previa en peluquerías*. Esta entidad cuenta con cuatro campos:

- **id_cita:** es un campo de tipo *String* y clave primaria. Almacena un identificador único para cada cita guardada.
- **id_horario:** es un campo de tipo *String* y clave *Foránea [Horarios]*. Almacena el identificador del horario que pertenece a dicha cita.
- **nombre:** es un campo de tipo *String* y almacena el nombre del usuario que ha pedido cita.
- **sexo:** es un campo de tipo *String* y almacena el sexo del usuario que ha pedido cita (este campo es necesario porque existe un peluquero para hombre y otro para mujer reservado para las peticiones a través de la aplicación, de tal forma que un hombre y una mujer pueden reservar el mismo horario).

2.3.1.5. Entidad Servicios

Entidad que contiene los servicios que se pueden contratar en la peluquería, por ejemplo: corte de pelo, tinte, corte de flecos, entre otros. Esta entidad cuenta con cuatro campos:

- **id_servicio:** es un campo de tipo *String* y clave primaria. Almacena un identificador único para cada cita guardada.
- **nombre:** es un campo de tipo *String* y almacena el nombre del servicio.
- **precio:** es un campo de tipo *String* y almacena el precio del servicio.

2.3.1.6. Entidad CitaServicio

Entidad que tiene como propósito principal relacionar cada cita con los servicios contratados. Sólo cuenta con dos campos, uno es **id_cita**, el identificador de la cita y clave *Foránea [Citas]* y el otro es **id_servicio**, el identificador del servicio y *Foránea [Servicios]*. Ambos campos son de tipo *String*.

2.3.2 Diseño de las clases e interfaces del modelo

El modelo de la *Aplicación Android para pedir cita previa en peluquerías*, como se puede ver en la figura 2.7, contará con una clase llamada *Modelo* y su interfaz *IModelo*. También contará con otras seis clases, una por cada tabla de la base de datos. En los siguientes apartados se describen cada una de las clases e interfases del modelo de la aplicación.

2.3.2.1. Clase *Modelo*

Esta clase implementa la interfaz *IModelo*, la cual define los siguientes métodos:

- **obtenerListaPeluquerias(): String**[]. Método que devuelve un vector con una lista de todas las peluquerías.
- **obtenerDescripcionPeluqueria(id_peluqueria: String): String**. Método que obtiene una descripción de una peluquería.
- **obtenerImagenPeluqueria(id_peluqueria: String): Bitmap** . Método que obtiene una imagen de una peluquería.

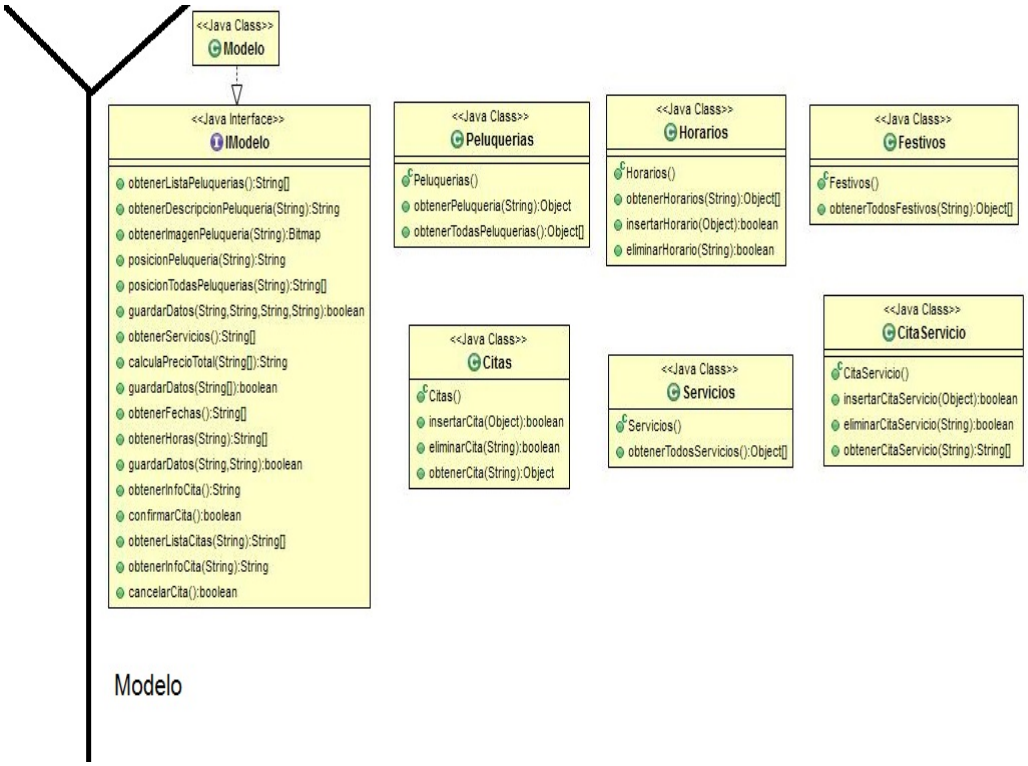

Figura 2.7: Clases e interfaces del modelo de la aplicación

- **posicionPeluqueria(id_peluqueria: String): String**. Método que obtiene la posición geográfica de una peluquería.
- **posicionTodasPeluquerias(id_peluqueria: String): String**[]. Método que obtiene un vector con la posición geográfica de todas las peluquerías.
- **guardarDatos(id_android: String, nombre: String, sexo: String, peluqueria: String): boolean** . Método encargado de guardar los datos en el modelo para posteriormente enviar al servidor.
- **guardarDatos(servicios: String[]): boolean** . Método encargado de guardar los datos en el modelo para posteriormente enviar al servidor.
- **guardarDatos(fecha: String, hora: String): boolean** . Método encargado de guardar ciertos datos en el modelo para posteriormente enviar al servidor.
- **obtenerServicios(): String**[] . Método que obtiene un vector con todos los servicios de la peluquería.
- **calculaPrecioTotal(servicios: String[]): String**. Método encargado de calcular el precio de los servicios contratados
- **obtenerFechas(): String**[]. Método encargado de obtener las fechas disponibles para poder pedir cita.

- **obtenerHoras(fecha: String): String**[]. Método encargado de obtener las horas disponibles en una fecha determinada.
- **obtenerInfoCita(): String**. Método encargado de obtener información de una cita.
- **confirmarCita(): boolean**. Método encargado de confirmar una cita, insertando dicha cita en la base de datos externa.
- **obtenerListaCitas(id_usuario: String): String**[]. Método encargado de obtener las citas que ha pedido un usuario determinado en la aplicación.
- **obtenerInfoCita(id_cita: String): String**. Método encargado de obtener información de una cita.
- **cancelarCita(): boolean**. Método encargado de cancelar una cita, eliminando dicha cita de la base de datos externa.

2.3.2.2. Clase *Peluquerias*

Esta clase tiene los siguientes métodos:

- **obtenerPeluqueria(id_peluqueria: String): Object**. Método que devuelve la peluquería de la base de datos que tiene el identificador que se le pasa por parámetros.
- **obtenerTodasPeluquerias(): Object**[]. Método que devuelve todas las peluquerías que hay guardadas en la base de datos.

2.3.2.3. Clase *Horarios*

Esta clase tiene los siguientes métodos:

- **obtenerHorarios(id_peluqueria: String): Object**[]. Método que devuelve los horarios de la base de datos que tienen el identificador de la peluquería que se le pasa por parámetros.
- **insertarHorario(horario: Object): boolean**. Método encargado de guardar un horario en la base de datos externa.
- **eliminarHorario(id_horario: String): boolean**. Método que elimina un horario de la base de datos externa.

2.3.2.4. Clase *Festivos*

Esta clase tiene los siguientes métodos:

- **obtenerFestivos(id_peluqueria: String): Object**[]. Método que devuelve los festivos de la base de datos que tienen el identificador de la peluquería que se le pasa por parámetros.

2.3.2.5. Clase *Citas*

Esta clase tiene los siguientes métodos:

- **insertarCita(cita: Object): boolean**. Método encargado de guardar una cita en la base de datos externa.
- **eliminarCita(id_cita: String): boolean**. Método que elimina una cita de la base de datos externa.
- **obtenerCita(id_cita: String): Object**. Método que devuelve la cita de la base de datos que tiene el identificador que se le pasa por parámetros.

2.3.2.6. Clase *Servicios*

Esta clase tiene los siguientes métodos:

- **obtenerTodosServicios(): Object**[]. Método que devuelve todos los servicios que hay guardados en la base de datos externa.

2.3.2.7. Clase *CitaServicio*

Esta clase tiene los siguientes métodos:

- **insertarCitaServicio(citaServicio: Object): boolean**. Método encargado de guardar una cita de un servicio (objeto *CitaServicio*) en la base de datos externa.
- **eliminarCitaServicio(id_cita: String): boolean**. Método que elimina una cita de un servicio (objeto *CitaServicio*) de la base de datos externa.
- **obtenerCitaServicio(id_cita: String): String**[]. Método que devuelve la cita de un servicio (objeto *CitaServicio*) de la base de datos que tiene el identificador que se le pasa por parámetros.

2.4 Diseño de las clases e interfaces del presentador

En esta sección se detallan las clases e interfaces que corresponden con la parte del presentador en el *MVP*, explicando cada método que aparece en la figura 2.8.

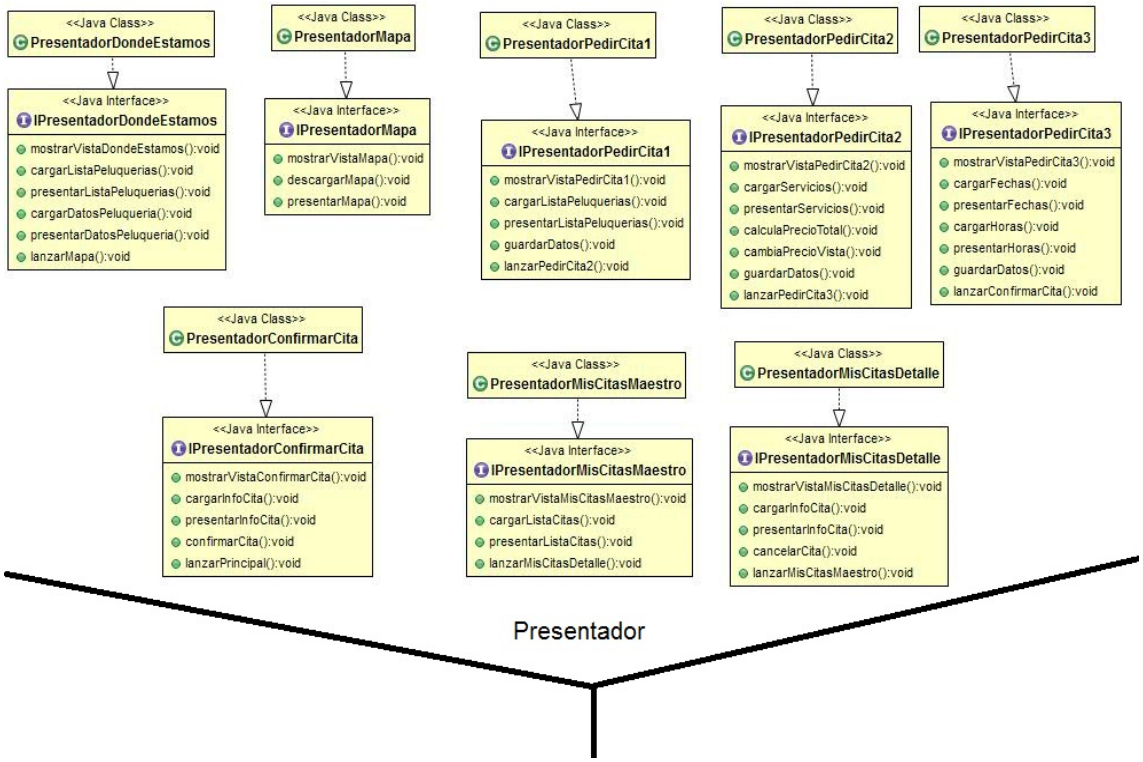

Figura 2.8: Clases del presentador de la aplicación con sus interfaces

2.4.1 Clase PresentadorDondeEstamos

Presentador correspondiente a la vista *DondeEstamosVistaActivity*, que es el encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorDondeEstamos*, la cual aparece representada en la figura 2.8 con los siguientes métodos:

- **mostrarVistaDondeEstamos(): void**. Método que mostrará la vista *DondeEstamosVistaActivity*.
- **cargarListaPeluquerias(): void**. Método que pedirá al modelo la lista de peluquerías.
- **presentarListaPeluquerias(): void**. Método presentará en la vista la lista de peluquerías previamente recogida del modelo.

- **cargarDatosPeluqueria(): void**. Método que pedirá al modelo los datos de la peluquería seleccionada en la vista.
- **presentarDatosPeluqueria(): void**. Método que presentará en la vista los datos de la peluquería previamente cargados.
- **lanzarMapa(): void**. Método que pedirá al presentador de la vista *MapaVistaActivity* que la muestre en pantalla.

2.4.2 Clase PresentadorMapa

Presentador correspondiente a la vista *MapaVistaActivity*, que es el encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorMapa*, que aparece representada en la figura 2.8 con los siguientes métodos:

- **mostrarVistaMapa(): void**. Método que mostrará la vista *MapaVistaActivity*.
- **descargarMapa(): void**. Método que construye y descarga un mapa del servidor externo de Google.
- **presentarMapa(): void**. Método que presenta el mapa previamente construido y descargado.

2.4.3 Clase PresentadorPedirCita1

Presentador correspondiente a la vista *PedirCita1VistaActivity*, que es el encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorPedirCita1*, que aparece representada en la figura 2.8 con los siguientes métodos:

- **mostrarVistaPedirCita1(): void**. Método que mostrará la vista *PedirCita1VistaActivity*.
- **cargarListaPeluquerias(): void**. Método que pedirá al modelo la lista de peluquerías.
- **presentarListaPeluquerias(): void**. Método que presentará en la vista la lista de peluquerías previamente recogida del modelo.
- **guardarDatos(): void**. Método que guarda en el modelo los datos recogidos en la vista.
- **lanzarPedirCita2(): void**. Método que pedirá al presentador de la vista *PedirCita2VistaActivity* que la muestre por pantalla.

2.4.4 Clase PresentadorPedirCita2

Presentador correspondiente a la vista *PedirCita2VistaActivity*, que es el encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorPedirCita2*, que aparece representada en la figura 2.8 con los siguientes métodos:

- **mostrarVistaPedirCita2(): void**. Método que mostrará la vista *PedirCita2VistaActivity*.
- **cargarServicios(): void**. Método que carga del modelo los servicios de la peluquería.
- **presentarServicios(): void**. Método que presenta en la vista los servicios previamente cargados.
- **calculaPrecioTotal(): void**. Método que calcula el precio según los servicios seleccionados en la vista.
- **cambiaPrecioVista(): void**. Método actualiza en la vista el precio previamente calculado.
- **guardarDatos(): void**. Método que guarda en el modelo los datos recogidos en la vista.
- **lanzarPedirCita3(): void**. Método que pedirá al presentador de la vista *PedirCita3VistaActivity* que la muestre por pantalla.

2.4.5 Clase PresentadorPedirCita3

Presentador correspondiente a la vista *PedirCita3VistaActivity*, que es el encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorPedirCita3*, que aparece representada en la figura 2.8 con los siguientes métodos:

- **mostrarVistaPedirCita3(): void**. Método que mostrará la vista *PedirCita3VistaActivity*.
- **cargarFechas(): void**. Método que carga del modelo las fechas disponibles para pedir cita.
- **presentarFechas(): void**. Método que presenta en la vista las fechas disponibles.
- **cargarHoras(): void**. Método que carga del modelo las horas disponibles según la fecha seleccionada en la vista.

- **presentarHoras(): void**. Método que presenta en la vista las horas disponibles previamente cargadas.
- **guardarDatos(): void**. Método que guarda en el modelo los datos recogidos en la vista.
- **lanzarConfirmarCita(): void**. Método que pedirá al presentador de la vista *ConfirmarCitaVistaActivity* que la muestre por pantalla.

2.4.6 Clase PresentadorConfirmarCita

Presentador correspondiente a la vista *ConfirmarCitaVistaActivity*, que es el encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorConfirmarCita*, que aparece representada en la figura 2.8 con los siguientes métodos:

- **mostrarVistaConfirmarCita(): void**. Método que mostrará la vista *ConfirmarCitaVistaActivity*.
- **cargarInfoCita(): void**. Método que carga del modelo la información de la cita.
- **presentarInfoCita(): void**. Método que presenta en la vista la información de la cita previamente cargada del modelo.
- **confirmarCita(): void**. Método que guarda en la base de datos externa la cita a través del modelo.
- **lanzarPrincipal(): void**. Método que pedirá al presentador principal que muestre la vista principal de la aplicación.

2.4.7 Clase PresentadorMisCitasMaestro

Presentador correspondiente a la vista *MisCitasVistaMaestroActivity*, que es el encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorMisCitasMaestro*, la cual aparece representada en la figura 2.8 con los siguientes métodos:

- **mostrarVistaMisCitasMaestro(): void**. Método que mostrará la vista *MisCitasVistaMaestroActivity*.
- **cargarListaCitas(): void**. Método que carga del modelo una lista con las citas que ha pedido el usuario de la aplicacion.

- **presentarListaCitas(): void**. Método que presenta en pantalla la lista con las citas previamente cargadas del modelo.
- **lanzarMisCitasDetalle(): void**. Método que pedirá al presentador de la vista *MisCitasVistaDetalleActivity* que la muestre por pantalla.
- **lanzarPrincipal(): void**. Método que pedirá al presentador principal que muestre la vista principal de la aplicación.

2.4.8 Clase PresentadorMisCitasDetalle

Presentador correspondiente a la vista *MisCitasVistaDetalleActivity*, que es el encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorMisCitasDetalle*, que aparece representada en la figura 2.8 con los siguientes métodos:

- **mostrarVistaMisCitasDetalle(): void**. Método que mostrará la vista *MisCitasVistaDetalleActivity*.
- **cargarInfoCita(): void**. Método que carga del modelo la información de la cita.
- **presentarInfoCita(): void**. Método que presenta en la vista la información de la cita previamente cargada del modelo.
- **cancelarCita(): void**. Método que borra de la base de datos externa la cita a través del modelo.
- **lanzarMisCitasMaestro(): void**. Método que pedirá al presentador de la vista *MisCitasVistaMaestroActivity* que la muestre por pantalla.

2.5 Adecuación del diseño a Android

Android es una plataforma para dispositivos móviles que contiene una pila de software donde se incluye un sistema operativo, middleware y aplicaciones básicas para el usuario. En la figura 2.9 se observan las distintas capas que componen Android. Cada una de estas capas utiliza servicios ofrecidos por las anteriores, y ofrece a su vez los suyos propios a las capas de niveles superiores [14]. Las capas se definen brevemente a continuación:

- **Aplicaciones:** contiene, tanto las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente, ya sean de terceras empresas o de su propio desarrollo. Todas estas aplicaciones utilizan los servicios, las API y bibliotecas de los niveles anteriores.

- **Framework de Aplicaciones:**. representa el conjunto de herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para Android utilizan el mismo conjunto de API y el mismo framework, representado por este nivel. Entre las API más importantes, se pueden encontrar las siguientes:
  - **Activity Manager:** conjunto de API que gestiona el ciclo de vida de las aplicaciones en Android.
  - **Content Provider:** permite a cualquier aplicación compartir sus datos con las demás aplicaciones de Android.
  - **View System:** proporciona un gran número de elementos para poder construir interfaces de usuario (GUI), como listas, mosaicos, botones, check boxes, tamaño de ventanas, control de las interfaces mediante teclado, entre otros.
  - **Location Manager:** posibilita a las aplicaciones la obtención de información de localización y posicionamiento.
- **Bibliotecas:** éstas proporcionan a Android la mayor parte de sus capacidades más características. Junto al núcleo basado en Linux, estas bibliotecas constituyen el corazón de Android. Entre las bibliotecas más importantes, se pueden encontrar las siguientes:
  - **OpenGL/SL y SGL:** Representan las bibliotecas gráficas. OpenGL/SL maneja gráficos en 3D y SGL proporciona gráficos en 2D.
  - **Librería SQLite:** creación y gestión de bases de datos relacionales.
- **Android Runtime:** al mismo nivel que las bibliotecas de Android se sitúa el entorno de ejecución. Éste lo constituyen las *Core Libraries*, que son bibliotecas con multitud de clases Java y la máquina virtual Dalvik.
- **Núcleo Linux:** Android utiliza el núcleo de Linux como una capa de abstracción para el hardware disponible en los dispositivos móviles. Esta capa contiene los *drivers* necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes. Siempre que un fabricante incluye un nuevo elemento de hardware, lo primero que se debe realizar para que pueda ser utilizado desde Android es crear las bibliotecas de control o drivers necesarios dentro de este kernel de Linux embebido en el propio Android.

2.5.1 Adecuación de la arquitectura MVP

Teniendo en cuenta la arquitectura *MVP* y con el fin de que la aplicación a desarrollar esté, a nivel de programación, lo más desacoplada posible (con el fin de

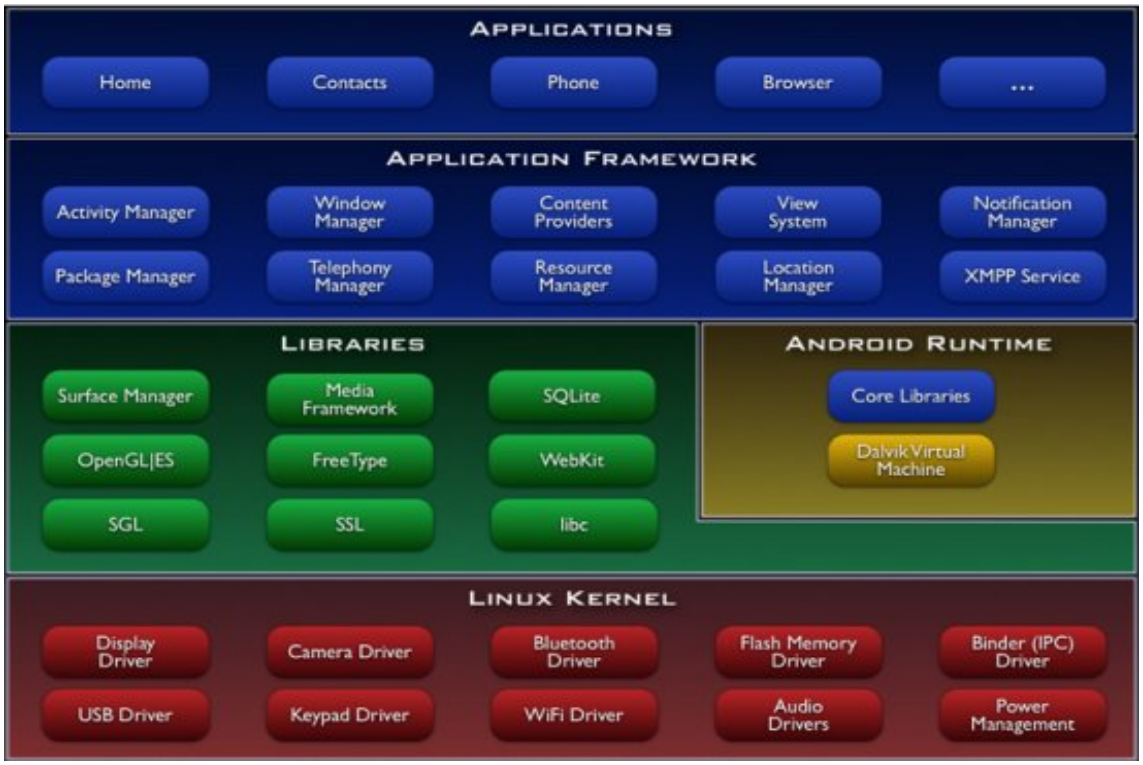

Figura 2.9: Arquitectura del sistema operativo Android

poder actualizar cualquier parte del sistema sin afectar al resto), se diseña la codificación de forma que las clases e interfaces de la *Vista*, el *Modelo* y el *Presentador* se almacenen en los paquetes vista, modelo y presentador, respectivamente. Las clases de la vista serán ventana de la interfaz de usuario, que en Android corresponden con clases que extienden de la clase Activity o de alguna de sus subclases [15]. Por otro lado, en la arquitectura *MVP* tradicional, el presentador es el punto de entrada a la aplicación y éste es el que crea a la vista y al modelo. Sin embargo, en Android, el punto de entrada a la aplicación es la vista principal y por tanto, la arquitectura *MVP*, tal y como se conoce, no puede ser implementada, por lo que hay que realizar ciertas modificaciones. La modificación más importante es el uso de una clase que conozca a todas las componentes de la aplicación y que se encargue del control de la navegación en la misma. Esta clase recibe el nombre de *AppMediador* y deriva de la clase *Application* [16] de Android (se almacena fuera de los paquetes indicados anteriormente).

Así, cuando se lanza la aplicación, se entra en la vista principal y ésta se encargará de obtener el objeto aplicación (es decir, el objeto de tipo *AppMediador*) y de indicarle a éste, quién es la vista principal. Cada vista de la aplicación, cuando sea cargada, debe indicar al objeto *AppMediador* quién es para que sus presentadores lo sepan (cuando necesiten acceder a sus vistas). Asimismo, cuando una vista tenga que delegar en su presentador para atender los eventos de usuario, le pedirá al objeto *AppMediador* que le indiqué qué objeto es su presentador. Por otro lado, el presentador de la vista principal se encargará de crear el objeto modelo y de indicar

al objeto *AppMediador* quién es este modelo, para que cualquier otro presentador pueda acceder a él.

Para el almacenamiento de la información en la *Aplicación Android para pedir cita previa en peluquerías* se va utilizar la plataforma Parse [17]. *Parse* es una plataforma de servicios online creada para facilitar la tarea de creación de un *back-end* a una aplicación móvil, entendiendo por *back-end* a los sistemas e infraestructuras necesarios para que los datos de una aplicación concreta sean accesibles desde la web.

Parse ofrece una infraestructura a partir de la cual se puede empezar a desarrollar una aplicación móvil (también se pueden desarrollar otro tipos de aplicaciones), de forma gratuita dependiendo del tráfico de consultas que exista entre la aplicación y los servicios prestados por *Parse*, en este caso en particular al no superar el millón de consultas por mes no será necesario realizar ningún pago a la organización.

Gracias a la infraestructura de *Parse* y a su sistema de administración, se reduce el tiempo de desarrollo, al evitar tener que crear una base de datos externa en un servidor y realizar consultas a él directamente, o crear un servicio REST para poder hacer consultas de forma remota. *Parse* ofrece un administrador en el que se crea el modelo de datos (no utiliza base de datos SQL) y una API de alto nivel que se añade fácilmente al SDK de Android facilitando la consultas al modelo de datos.

Los tipos de datos que soporta el modelo de datos que proporciona *Parse* son los son siguientes: *String, Number, Boolean, Date, File, Geopoint, Array, Object, Pointer, Relation*. En el caso de la aplicación objeto de este trabajo, los campos de las tablas de la base de datos serán de alguno de los tipos anteriores (en el que se adapte mejor según el tipo elegido).

2.5.2 Identificación de los patrones usados

Los patrones de diseño se utilizan para resolver problemas comunes de ingeniería. Cuando se usa un patrón de diseño para resolver un problema, se adapta el patrón a las necesidades específicas de ese problema.

Android ofrece en su web un apartado de diseño [18] en el que ofrece patrones para distintas tareas. De los diferentes patrones que ofrece, en la *Aplicación Android para pedir cita previa en peluquerías* se usaran los siguientes:

- **Action Bar.** Barra superior presente en cualquier vista de la aplicación. En la que puede aparecer el nombre de la aplicación o vista en la que se encuentre, contiene botones para las acciones mas importantes y oculta en un botón las acciones menos importantes o que se utilizan en menor medida, tal y como se observa en la figura 2.10.

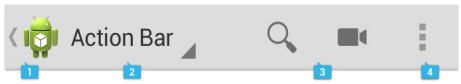

Figura 2.10: Action Bar

- **Confirmación y Reconocimiento.** En los momentos en los que se invoca alguna acción, como por ejemplo en el momento en el que el usuario selecciona el botón de salir de la aplicación, es una buena idea ofrecer al usuario la opción de confirmar la acción, por si le ha dado de forma errónea. En el caso de la *Aplicación Android para pedir cita previa en peluquerías* habrá confirmación cada vez que el usuario quiera salir de la aplicación y también habrá reconocimiento cuando el usuario, confirme o cancele cita.
- **Preferencias.** Se ofrece al usuario un lugar en su aplicación donde indica sus preferencias con la forma en la que su aplicación debe comportarse. Esto beneficia a los usuarios debido a que no es necesario que se les interrumpa con las mismas preguntas una y otra vez cuando se presentan ciertas situaciones. Los ajustes predeterminan lo que siempre va a pasar en esas situaciones.
- **Ayuda.** Aunque se debe hacer siempre una aplicación en la que el uso de la ayuda sea innecesario, siempre deber existir una apartado de ayuda en el que el usuario pueda resolver sus dudas y aprender más sobre la aplicación.

En cuanto a los patrones de diseño de software, la *Aplicación Android para pedir cita previa en peluquerías* usará:

- **Singleton:** la clase AppMediador implementa este patrón de forma que sólo existe un objeto de este tipo en la aplicación (no permiten la creación de más de un objeto de este tipo).
- **Delegado:** las vistas de la aplicación delegan el tratamiento de las acciones del usuario (por ejemplo, la selección de un determinado botón), a sus presentadores. Así, los presentadores realizarán las operaciones oportunas en nombre de sus vistas.
- **Observador:** los presentadores de la aplicación deben observar al modelo, de forma que cuando éste termine de realizar el acceso a la información, los presentadores deben saberlo. En Android, para realizar este proceso, se usan las notificaciones broadcast (o lo que es lo mismo, un objeto de tipo *BroadcastReceiver* [19]).
- **Maestro-Detalle:** existen dos vistas, la vista maestro con un listado de objetos y la vista detalle con información del objeto que se ha seleccionado previamente en la vista maestro. En la *Aplicación Android para pedir cita previa en peluquerías* este patrón se ve reflejado claramente en el caso de uso *Mostrar Citas*.

Capítulo **3**

MODELO DE IMPLEMENTACIÓN

3.1 Introducción

El *Modelo de implementación* utiliza el resultado del *Modelo de diseño* para generar el código final en el lenguaje de programación elegido [11]. Aunque el diseño de objetos es independiente del lenguaje de programación final, todos los lenguajes tienen particularidades que deben adaptarse durante el proceso de implementación. La elección del lenguaje influye en el diseño, pero el diseño no debe depender de los detalles del lenguaje, es decir, un cambio de lenguaje no puede hacer que el diseño cambie.

Asimismo, y como ya se ha comentado en capítulos anteriores, el diseño de la aplicación utiliza la arquitectura *Modelo-Vista-Presentador* (MVP) adaptada al lenguaje elegido. En esta adaptación, se incorpora una nueva clase, llamada *AppMediador* (introducida en el *Modelo de Diseño*) que va a funcionar como punto de enlace de las distintas partes de la arquitectura. Así, el *AppMediador*, que representa a la clase *Application* (aplicación) se encargará de:

- Definir los presentadores, vistas y modelo de la arquitectura y los métodos *accessor* de éstos.
- Definir la navegación en la aplicación, es decir, qué vistas (de la interfaz de usuario) se deben lanzar cuando se considere oportuno (normalmente, ante peticiones del usuario).
- Definir las constantes correspondientes a los avisos de notificación, enviadas desde el modelo cuando se haya terminado de realizar una determinada acción.
- Definir las constantes correspondientes a los datos comunicados entre vistas, o recuperados desde el modelo, entre otros, ya que en Android los datos se comunican usando pares *clave,valor*.
- Definir los métodos propios de Java para Android correspondientes al: lanzamiento de actividades, creación de servicios, registros de receptores *broadcast*, envíos de notificaciones *broadcast*, entre otros.

Es importante comentar que para que la aplicación desarrollada utilice este objeto *AppMediador*, en el archivo *AndroidManifest* hay que indicar que el nombre de la aplicación corresponde con este archivo. Para ello, hay que indicar en este archivo, dentro de la etiqueta *application*, lo siguiente:

```
<application
   android:name="rutaHastaLaClase.AppMediador"
   ...>
</aplication>
```

En la siguientes secciones se especificarán los cambios realizados en la implementación del diseño, empezando por la descripción del *AppMediador* para esta aplicación y siguiendo con las modificaciones realizadas en ciertas clases de los tres paquetes definidos: *vista*, *presentador* y *modelo*, debido a la implementación en Java para Android y a la arquitectura elegida.

3.2 Clase AppMediador

El *AppMediador* contiene una variable por cada una de las interfaces del presentador, la vista y el modelo de la aplicación *Aplicación Android para pedir cita previa en peluquerías*.

Por otro lado, se definen unas constantes de petición y notificación (*public* y *static*) que serán las claves de los *Intent* (*clave, valor*). Se utilizarán en los objetos *BroadcastReceiver* y en los métodos *sendBroadcast* para notificar cuándo ha finalizado un evento en la aplicación (por ejemplo, el modelo notifica cuándo termina de insertar datos en la base de datos).

Asimismo, se implementan:

- Los métodos de creación y eliminación de presentadores: "getPresentadorXXX" y "removePresentadorXXX" (donde *XXX* corresponde al nombre de un presentador), para cada una de las variables creadas con anterioridad y que representan a los presentadores introducidos en el *Modelo de diseño*.
- Los métodos *accessor* de las vistas definidas en el *Modelo de requisitos*: "getVistaXXX" y "setVistaXXX" (donde *XXX* corresponde al nombre de una vista). Estos métodos son utilizados para obtener o iniciar las variables de las actividades (*Activity* de Android).
- Los métodos *accessor* del modelo: "getModelo" y "setModelo".
- Los métodos de navegación.
- Los métodos de uso de Java para Android.

- El método "onCreate", que inicia todas las variables de los presentadores a *null* e indica que la clase *AppMediador* es un *singleton*.

La vista principal de la aplicación, que corresponde con la clase *PrincipalVistaActivity*, crea el objeto *AppMediador* con el método "getApplication", para que el resto de clases del proyecto puedan usarlo. Asimismo, el presentador principal, que corresponde con la clase *PresentadorPrincipal*, crea al modelo, que se encarga de manejar el almacenamiento de los datos.

3.3 Paquete vista

En el paquete vista se han implementado las clases y métodos que se observan en la imagen 3.1. A continuación se identifican las diferencias entre las clases e interfaces de la vista propuestas en el *Modelo de requisitos* y las finalmente implementadas. Así, los cambios son los siguientes:

- En todas las vistas que interactúan con la base de datos externa se han añadido los métodos:
  - **void mostrarAlerta(String titulo)**. Método que muestra en la vista una alerta con el texto que se le pasa por parámetros.
  - **void mostrarProgreso(String mensaje)**. Método que muestra una barra de progreso para indicar que la aplicación está realizando alguna operación.
  - **void eliminarProgreso()**. Método que elimina la barra de progreso de la vista.
- En Andrid se requiere que se añadan todas las clases de las vistas mediante los siguientes métodos:
  - **void onCreate(Bundle savedInstanceState)**. Método que inicia la actividad. Sólo se ejecuta una vez.
  - **void onStart()**. Método que se ejecuta justo después del *onCreate* o cuando se vuelve a la actividad. La actividad se vuelve visible para el usuario cuando se llama a este método.
  - **boolean onCreateOptionsMenu(Menu menu)**. Inicia el contenido del menú de opciones de la actividad.
  - **boolean onOptionsItemSelected(MenuItem item)**. Método que se llama cada vez que se selecciona un elemento en el menú.
  - **void onClick(View v)**. Método de la interfaz *OnClickListener* que se invoca cuando se presiona sobre un botón de una vista, para realizar las acciones oportunas. Se ha añadido este método para aquellas vistas que lo requieran.

3.3.1 Clase Clase MapaVistaActivity e interfaz IVistaMapa

Al utilizar la API de Google Maps para Android los cambios que se hacen en el presentador sobre el mapa se actualizan directamente en la vista, por ello se ha suprimido el siguiente método al quedar inutilizado:

- **setMapa(mapa: Object)**

3.3.2 Clase PedirCita1VistaActivity e interfaz IVistaPedirCita1

Se ha añadido esta vista un campo de texto en el que se introduce el teléfono móvil por cuestiones de funcionalidad. Por ello ha sido necesario añadir el siguiente método:

- **String getTextoTelefono()**. Devuelve el teléfono introducido en la vista.

3.3.3 Clase PedirCita2VistaActivity e interfaz IVistaPedirCita2

Se ha decidido que el precio de los servicios se cargue del servidor externo y en consecuencia en la vista se añaden los siguientes métodos:

- **void setPrecioCorteDePelo(String precio)**. Actualiza el precio del corte de pelo en la vista.
- **void setPrecioManicura(String precio)**. Actualiza el precio de la manicura en la vista.
- **void setPrecioPedicura(String precio)**. Actualiza el precio de la pedicura en la vista.
- **void setPrecioMechas(String precio)**. Actualiza el precio de las mechas en la vista.
- **void setPrecioTinte(String precio)**. Actualiza el precio del tinte en la vista.
- **void setPrecioCorteDeFlecos(String precio)**. Actualiza el precio del corte de flecos en la vista.

3.3.4 Clase MisCitasVistaMaestroActivity e interfaz IVistaMisCitasMaestro

Para mostrar el listado de citas del usuario es necesario identificarlo y se hace mediante número de teléfono, se ha añadido el siguiente método en la vista para recoger el teléfono del usuario:

- **String getTelefono()**. Devuelve el teléfono introducido en la vista.

3.4 Paquete presentador

En el paquete presentador se han implementado las clases y métodos que se observan en la imagen 3.2. A continuación se identifican las diferencias entre las clases e interfaces del presentador propuestas en el *Modelo de diseño* y las finalmente implementadas. Así, los cambios son los siguientes:

- En los presentadores que necesitan datos del servidor externo se ha añadido una variable privada de la clase *BroadcastReceiver* para detectar los avisos desde el modelo cuando termina de descargar, insertar o eliminar datos.

3.4.1 Clase PresentadorMapa e interfaz IPresentadorMapa

Se ha suprimido el método, **descargarMapa(): void**, porque al utilizar la API de Google Maps, se realiza este método de manera autónoma.

Se han añadido los siguientes métodos:

- **void presentarMapaTodasPeluquerias()**. Presenta un mapa con todas las peluquerías.
- **void presentarMapaPeluqueriaCercana()**. Presenta un mapa con la peluquería más cercana.

3.5 Paquete modelo

En el paquete modelo se han implementado las clases y métodos que se observan en la imagen 3.3. A continuación se identifican las diferencias entre las clases e interfaces del modelo propuestas en el *Modelo de diseño* y las finalmente implementadas. Así, se han añadido las siguientes clases:

- **DatosCita**. Clase necesaria para trabajar con bases de datos externa. Esta clase representa un objeto con los datos de la cita.
- **DatosCitaServicio**. Clase necesaria para trabajar con bases de datos externa. Esta clase representa un objeto con las relaciones de las citas con los servicios.
- **DatosHorario**. Clase necesaria para trabajar con bases de datos externa. Esta clase representa un objeto con los horarios de las citas.
- **DatosPeluqueria**. Clase necesaria para trabajar con bases de datos externa. Esta clase representa un objeto con los datos de la peluquerías.
- **DatosServicios**. Clase necesaria para trabajar con bases de datos externa. Esta clase representa un objeto con los datos de los servicios.

3.5.1 Clase Modelo e interfaz IModelo

Se modifican los métodos del *Modelo* que interactúan con el servidor externo, ahora no retornan ninguna variable, son de tipo void. Los datos del modelo ahora se envían utilizando notificaciones broadcast con objetos de tipo *BroadcastReceiver*.

Se han añadido los siguientes métodos:

- **void setCitaSeleccionada(String id_cita)**. Guarda la cita seleccionada.
- **String getCitaSeleccionada()**. Obtiene la cita seleccionada.
- **void setPeluqueriaSeleccionada(String id_peluqueria)**. Guarda la peluquería seleccionada.
- **String getPeluqueriaSeleccionada()**. Obtiene la peluquería seleccionada.
- **void setTelefono(String telefono)**. Guarda el teléfono del usuario.
- **String getTelefono()**. Obtiene el teléfono del usuario
- **void obtenerDatosPeluquerias()**. Obtiene los datos de las peluquerías guardadas en el servidor externo.

Figura 3.1: Clases del paquete vista

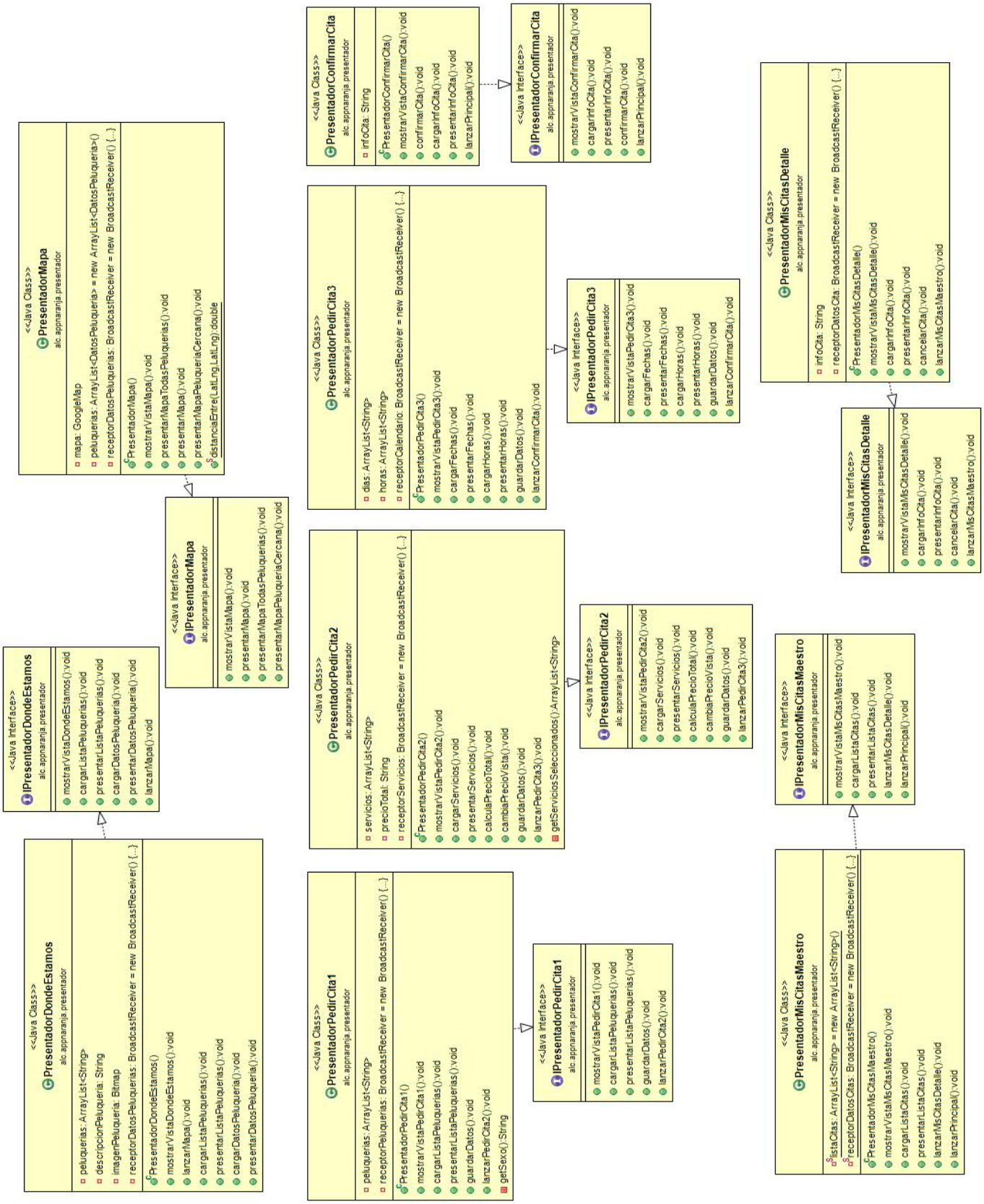

Figura 3.2: Clases del paquete presentador

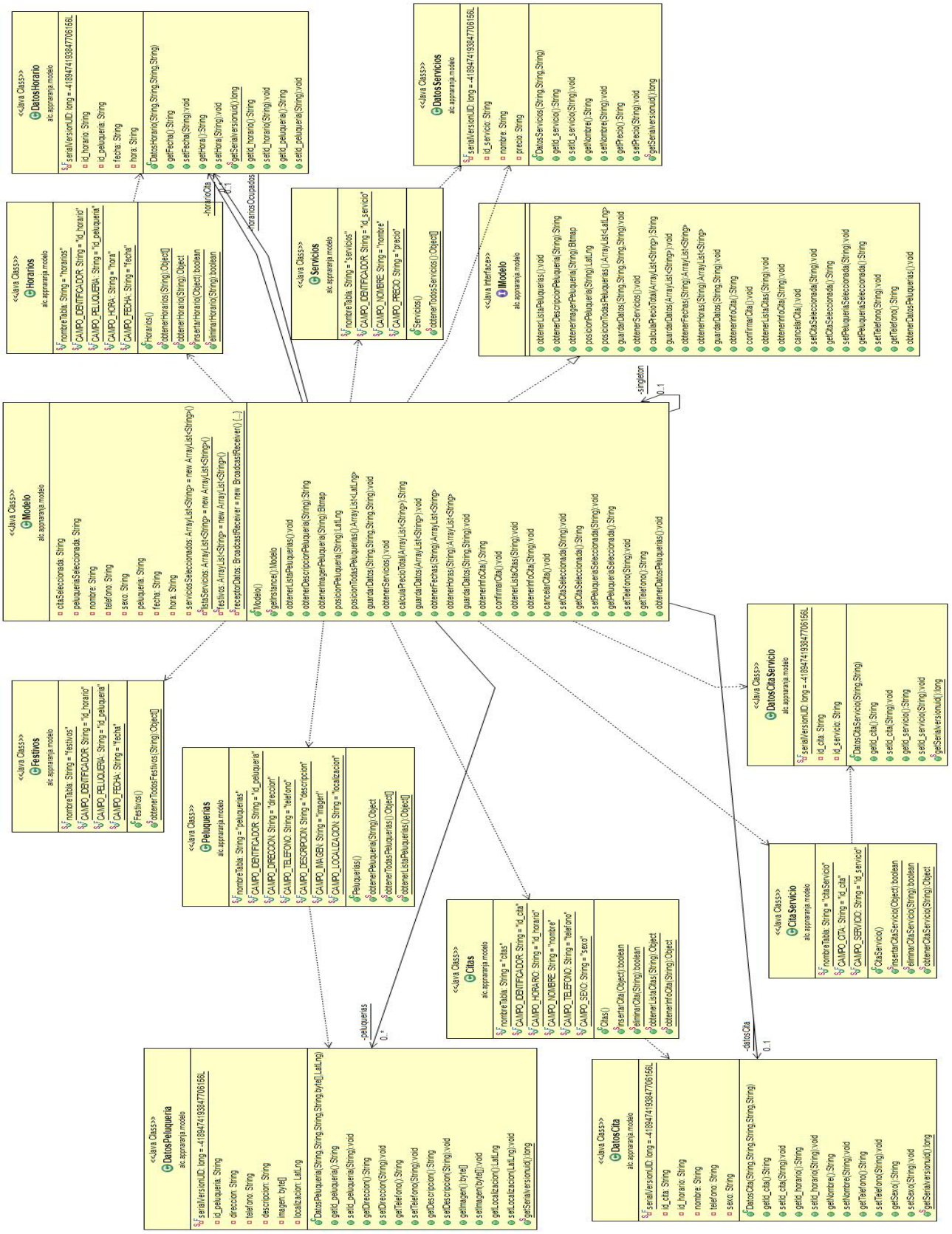

Figura 3.3: Clases del paquete modelo

Capítulo **4**

MODELO DE PRUEBAS

4.1 Test de usabilidad de la interfaz de usuario

El propósito del test de usabilidad es descubrir en qué grado la interfaz de la *Aplicación Android para pedir cita previa en peluquerías* cumple con su propósito y satisface los objetivos en cuanto a efectividad, eficiencia y satisfacción. El test es una observación de cómo interactúan los usuarios representativos con la interfaz.

La realización de este test de usabilidad está motivado por la necesidad de detectar debilidades en el producto y su nivel de utilidad, es decir, revelar si el usuario encuentra provecho, beneficio y muestra interés.

El test lo realizarán personas habituadas al uso de aplicaciones móviles.

4.1.1 Explicación del producto

La *Aplicación Android para pedir cita previa en peluquerías* facilita al usuario la labor de pedir cita en su peluquería. Éste no tiene que realizar ninguna llamada, sólo tiene que abrir la aplicación, introducir sus datos, elegir fecha y la aplicación enviará directamente la petición de cita a su peluquería.

Sus características principales son:

- Diseño claro y simple.
- Ubicación de peluquerías en un mapa.
- Petición de citas introduciendo datos de la cita (fecha, peluquería, servicios a realizar, entre otros).

4.1.2 Objetivos del producto

La *Aplicación Android para pedir cita previa en peluquerías* tiene como objetivo principal pedir cita en la peluquería pero también tiene una pequeña descripción de la peluquería, datos de contacto y visualización de las peluquerías en un mapa.

La aplicación pretende ayudar al usuario en los siguientes aspectos:

- Pedir cita.
- Ponerte en contacto con la peluquería.
- Localizar las peluquerías en el mapa.

Las mejoras que aporta son:

- Agilidad a la hora de pedir cita.
- Ahorro de tiempo por parte del usuario.
- Ahorro de dinero por parte del usuario (servicio totalmente gratuito).
- Publicidad para la peluquería.
- Innovación y nuevas tecnologías en la peluquería.

4.1.3 Usuarios y participantes

El producto está destinado a todo tipo de usuarios porque las peluquerías son utilizadas por todo tipo de personas tanto jóvenes como personas de media edad y personas mayores.

El tipo de usuario que realizará el test de usabilidad no debe ser experto, pero sí estar familiarizado con el uso de aplicaciones móviles, y ser cliente habitual de alguna peluquería, para ser capaces de sacar el máximo provecho al test.

4.1.3.1. Usuarios finales del producto

El perfil del usuario final a quien va dirigido el producto es una persona mayor a 15 años, de cualquier género, que acude al menos una vez al mes a la peluquería.

4.1.3.2. Criterios de selección del grupo de test

Es muy importante que los participantes sean lo más representativos posible ya que, normalmente, no hay posibilidad de trabajar con grandes muestras para hacer las pruebas. Por esta razón se seguirán los siguientes criterios para seleccionar los usuarios que realizarán los test:

- Hispanohablante con edad del cliente objetivo (>15 años).
- Debe disponer de un terminal Android.
- Lleva utilizando un smartphone o similar más de 1 año.
- Alguna vez se ha interesado o descargado aplicaciones móviles.
- Acude al menos una vez al mes a la peluquería.

4.1.4 Diseño del proceso de test

En los siguientes apartados se explica cómo se realizará el proceso de test, herramientas necesarias, tareas del usuario, cómo se obtendrán las evidencias, entre otros.

4.1.4.1. Logística

La evaluación se llevará a cabo en un ambiente relajado y sin ruidos durante el periodo del test, el cual tendrá una duración de 15 minutos. La ubicación ideal para esta tarea es el domicilio del usuario, pero no siempre será posible, por lo que se optarán por lugares donde el usuario se sienta cómodo.

Los tests se realizarán los días 18, 20 y 21 de Marzo de 2014. El análisis de los datos se llevará a cabo al finalizar la recopilación de los mismos, y se obtendrán los resultados el 21 de Marzo de 2014.

Antes de iniciar la evaluación, se le dará al usuario una breve descripción sobre el propósito de la aplicación y se le explicará las tareas (no las acciones) que debe realizar de forma sencilla y clara. Una vez finalizada la evaluación, se entregará al usuario un cuestionario de usabilidad para obtener las evidencias. En dicho cuestionario se recopilarán datos de usabilidad, utilidad y estética de la aplicación.

4.1.4.2. Instrumentación

El material que necesitará el usuario para realizar el test será un ordenador o tableta que disponga de un software capaz de leer archivos PDF. En este terminal, se abrirá el archivo *NaranjaAppPrototipo.pdf*, que corresponde con el archivo generado por la herramienta *Balsamiq Mockups* [20]. El usuario realizará las tareas oportunas en este archivo interactivo, mientras el evaluador apunta todos los datos necesarios para la evaluación.

4.1.4.3. Tareas a realizar

A continuación se muestran las tareas a realizar por el usuario, en la que deberá apuntar el tiempo que tarda en realizarla y las observaciones que éste crea oportunas:

- **Tarea nº1:** busca información de *Peluquerías Naranja*. Acciones:
  1. Abrir el documento *NaranjaAppPrototipo.pdf*.
  2. En la primera pantalla te aparecerá un móvil, abrir la aplicación *NaranjaApp* (representada en rojo con unas tijeras).
  3. Entrar en la sección *Peluquerías Naranja*.
  4. Tomar una captura de pantalla y nombrarla "TAREA1.jpg".
  5. Contestar a esta pregunta en observaciones: **¿En qué año y dónde se inauguró el primer salón Naranja?**
- **Tarea nº2:** ver en mapa la *Peluquería Naranja* de Lanzarote (Arrecife). Acciones:
  1. Abrir el documento *NaranjaAppPrototipo.pdf*.
  2. En la primera pantalla te aparecerá un móvil, abrir la aplicación *NaranjaApp* (representada en rojo con unas tijeras).
  3. Entrar en la sección Contacto.
  4. Presionar en *Ver en Mapa*.
  5. Tomar una captura de pantalla y nombrarla "TAREA2.jpg".
- **Tarea nº3:** pide cita con la aplicación. Acciones:
  1. Abrir el documento *NaranjaAppPrototipo.pdf*.
  2. En la primera pantalla te aparecerá un móvil, abrir la aplicación *NaranjaApp* (representada en rojo con unas tijeras).
  3. Entrar en la sección *Pedir Hora*.

4. Navegar por las distintas pantallas introduciendo los datos (en el pdf no se pueden introducir datos pero deja los que salen por defecto). Deja de pasar pantallas antes de presionar en *Confirmar Reserva*.
5. Tomar una captura de pantalla y nombrarla "TAREA3.jpg".

- **Tarea nº4:** accede a la configuración. Acciones:
  1. Abrir el documento *NaranjaAppPrototipo.pdf*.
  2. En la primera pantalla te aparecerá un móvil, abrir la aplicación *NaranjaApp* (representada en rojo con unas tijeras).
  3. Presionar en el botón de configuración.
  4. Tomar una captura de pantalla y nombrarla "TAREA4.jpg".
- **Tarea nº5:** salir de la aplicación. Acciones:
  1. Abrir el documento *NaranjaAppPrototipo.pdf*.
  2. En la primera pantalla te aparecerá un móvil, abrir la aplicación *NaranjaApp* (representada en rojo con unas tijeras).
  3. Entrar en la sección *Salir*.
  4. Tomar una captura de pantalla y nombrarla "TAREA5.jpg".

4.1.4.4. Indicadores de éxito

- **Tarea nº1:** se habrá encontrado la información satisfactoriamente si en pantalla aparece un párrafo que comienza así "Inauguramos nuestro primer salón Naranja en Lanzarote a principios de 2011..."
- **Tarea nº2:** se habrá visto el mapa con la peluquería de Lanzarote (Arrecife) si en pantalla aparece un mapa callejero con una tijera roja con el nombre *Naranja Arrecife*.
- **Tarea nº3:** se habrá pedido cita correctamente si aparece una pantalla en la que aparecen los datos de la cita y se puede confirmar cita.
- **Tarea nº4:** cuando haya entrado en configuración debe aparecer una pantalla para poder cambiar el idioma, el fondo, activar el GPS, activar el WIFI, entre otros.
- **Tarea nº5:** se habrá salido de la aplicación cuando aparezca en pantalla el texto "Gracias por utilizar nuestra aplicación..."

4.1.4.5. Respuesta de los usuarios a los indicadores

Los tres usuarios que han realizado el test, han completado con éxito y en el tiempo establecido todas las tareas exceptuando el participante 1, que no ha podido realizar la tarea nº4.

4.1.4.6. Resumen de tipo de errores encontrados

En general los usuarios no han encontrado errores en el prototipo utilizado, aunque han expresado algunos aspectos negativos que se detallan en la siguiente sección.

4.1.4.7. Resumen del tipo de aspectos negativos expresados

- **Tarea nº1:**
  - Letras muy pequeñas que se confunden con el color del fondo.
  - Se hace pesado leer tanto texto sin ninguna imagen.
- **Tarea nº2:** Nombre de sección *Contacto* poco intuitivo. Se recomienda **¿Dónde Estamos?**
- **Tarea nº3:**
  - Nombre de sección *Pedir Hora* poco intuitivo. Se recomienda **Pedir Cita**.
  - Una vez confirmada la cita no hay ninguna sección con las citas pedidas. Se recomienda un apartado **Mis Citas**.
- **Tarea nº4:** Icono configuración no visible, se confunde con el fondo.
- **Tarea nº5:** Botón *Salir* muy grande, muchos usuarios salen de la aplicación sin querer.

4.1.5 Conclusiones finales

Tras realizar el test a los usuarios y analizar los resultados obtenidos, se han detectado carencias en el diseño y en las funciones de la aplicación.

La modificación en el diseño del prototipo contempla lo siguiente:

- Quitar el botón *Configuración* porque se confunde con el fondo.
- Añadir un menú con las opciones (*Configuración*, *Ayuda*, *Info* y *Salir*).
- Añadir la función *Mis Citas* para gestionar las citas pedidas.
- Cambiar la sección *Contacto* por la de *¿Dónde estamos?* Puesto que en esta sección no sólo se puede poner en contacto con la peluquería elegida, sino que también se puede ver un mapa con su ubicación.

- Cambiar la sección *Pedir Hora* por *Pedir Cita*, ya que es algo más intuitivo.

Se ha modificado el prototipo inicial *NaranjaAppPrototipo.pdf* y se ha generado uno nuevo llamado *NaranjaAppPrototipoCorregido.pdf* con todas las modificaciones que se han estimado oportunas. Ambos archivos, así como los test realizados a los usuarios, se encuentran en la documentación de este Trabajo fin de grado.

4.2 Test de verificación del modelo de datos

A continuación se detallan los tests realizados al código de la aplicación. El objetivo principal de este test es comprobar el funcionamiento de la aplicación de forma automática, para lo cual se han desarrollado dos tests en los que se controlan las partes del código que interactúan con el modelo (al ser este el más susceptible a errores al depender de la nube).

Para realizar los tests se utilizan las herramientas que ya integran el SDK de Android, que permiten ejecutar las pruebas en un emulador o directamente en el dispositivo físico [21].

4.2.1 Test pedir cita

En este test se pretende comprobar que al pedir una cita, se queda guardada en la base de datos externa. Para ello se ha realizado un test en el que se inserta una cita en la tabla *Citas* con los siguientes datos:

- **Nombre:** Adolfo
- **Teléfono:** 636175744
- **Sexo:** Hombre
- **Peluquería:** Lanzarote
- **Fecha:** 23/03/2015
- **Hora:** 10:00

Para comprobar que se ha escrito la cita en la base de datos externa, se obtiene la lista de citas del usuario (con su número de teléfono).Acto seguido comprobamos que la cita que se acaba de introducir se encuentra en la lista descargada, en caso afirmativo se procede a comprobar si los datos son correctos.

Para comprobar si los datos de la cita son correctos se comparan los datos que se introdujeron en la cita con los datos que se descargan de la base de datos externa.

Este test se realizó en un tiempo de 6.780 segundos y resultó satisfactorio, por lo que se asume que la implementación es correcta.

4.2.2 **Test de cargar lista peluquerías**

En este test se pretende descargar la lista de peluquerías de la base de datos externa y comprobar que se descarga correctamente.

En la base de datos externa existe una tabla con el siguiente listado de peluquerías:

- La Minilla
- Lanzarote
- Las Canteras
- Siete Palmas
- Tamaraceite
- Telde
- Triana
- Vecindario

Para comprobar si el listado es correcto, el test se encarga de descargar el listado y comprobar que existe la peluquería **Lanzarote**.

El test se ha realizado correctamente en un tiempo de 1.534 segundos.

BIBLIOGRAFÍA

[1] *Estudio de la firma norteamericana Gartner.*
Disponible en: http://www.gartner.com/newsroom/id/2573415 Consultada Marzo 2014.

[2] *IOS Dev Center.*
Disponible en: https://developer.apple.com/devcenter/ios/index.action Consultada Marzo 2014.

[3] *Características Windows Phone.*
Disponible en: http://www.windowsphone.com/es-es/features Consultada Marzo 2014.

[4] *Apple Inc.*
Disponible en: https://www.apple.com/es/ Consultada Marzo 2014.

[5] Petersen, Richard.
Linux: manual de referencia (6a. ed.).

[6] *Dalvik Technical Information.*
Disponible en: http://source.android.com/devices/tech/dalvik/ Consultada Marzo 2014.

[7] *Aplicaciones cita previa en el Play Store.*
Disponible en: https://play.google.com/store/ Consultada Marzo 2014.

[8] *Entorno de programación Eclipse Android.*
Disponible en: http://developer.android.com/sdk/index.html Consultada Marzo 2014.

[9] *Información Android para desarrolladores.*
Disponible en: http://developer.android.com/index.html Consultada Marzo 2014.

[10] *Objectory, por Ivar Jacobson.*
Disponible en: http://metodologiasoo.wikispaces.com/Objectory,+por+Ivar+Jacobson Consultada Junio 2014.

[11] Alfredo Weitzenfield.
Ingeniería de software orientada a objeto con UML, Java e Internet.
Editorial Thomson.

[12] *Patrón MVP.*
Disponible en: http://www.imaginanet.com/blog/patron-mvp.html
Consultada Junio 2014.

[13] *Bases de datos NoSQL.*
Disponible en: http://www.slideshare.net/dipina/nosql-introduccin-a-las-bases-de-datos-no-estructuradas
Consultada Junio 2014.

[14] *Programación en dispositivos móviles portables - Android.*
Disponible en: https://sites.google.com/site/swcuc3m/home/android
Consultada Junio 2014.

[15] *Clase Activity de Android.*
Disponible en: http://developer.android.com/reference/android/app/Activity.html
Consultada Junio 2014.

[16] *Clase Application de Android.*
Disponible en: http://developer.android.com/reference/android/app/Application.html
Consultada Junio 2014.

[17] *Parse.com.*
Disponible en: https://parse.com/
Consultada Junio 2014.

[18] *Patrones de diseño Android.*
Disponible en: https://developer.android.com/intl/es/design/patterns/
Consultada Junio 2014.

[19] *Clase BroadcastReceiver de Android.*
Disponible en: http://developer.android.com/reference/android/content/BroadcastReceiver.html
Consultada Junio 2014.

[20] *Sitio Web de Balsamiq Mockups.*
Disponible en: http://balsamiq.com/products/mockups/
Consultada Junio 2014.

[21] *Test Android.*
Disponible en: https://developer.android.com/tools/testing/index.html
Consultada Junio 2014.

Parte III

Pliego de condiciones

PLIEGO DE CONDICIONES

Introducción

A continuación se especificarán los requisitos técnicos que permitirá la ejecución de la aplicación en terminales móviles con Android. Para ello, se establecerán las diferentes condiciones referentes al hardware y al software, así como la instalación de la aplicación. Por último, se expondrá la licencia de uso del software.

Requisitos

El hardware requerido para el correcto funcionamiento de la aplicación consiste en un terminal *Android* con la versión 2.3.3 o superior de su sistema operativo, GPS (deseable) y conexión a Internet.

Los distintos fabricantes de terminales móviles *Android* especifican, a través de las hojas de características, la version del sistema operativo soportado. Asimismo, el terminal también lo indica accediendo a: *Ajustes* → *Información del teléfono* → *Versión de Android*.

Condiciones de instalación

Para la instalación de la aplicación desarrollada, el archivo a instalar se encuentra en la entrada *Código* del menú del disco compacto adjunto a esta memoria.

Así, se procede a copiar el archivo *AppNaranja.apk* en el terminal móvil. Si dispone de memoria externa (*sdcard* o memoria SD), copia el archivo en una ubicación cualquiera (se recomienda la carpeta *Aplicaciones* o similar). En el caso que no disponga de memoria externa se copia directamente en la memoria interna del terminal móvil.

La copia puede llevarse a cabo mediante USB, *Bluetooth* o *Wi-Fi*. Si bien, para el primero de ellos se realiza la copia directamente mediante el propio explorador de archivos del ordenador, en el caso del *Bluetooth* y *Wi-Fi* es necesario un gestor de ficheros para la transferencia.

Asimismo, es necesario tener activada la opción de poder instalar aplicaciones que no se hallan descargado desde el *Google Play*. Para ello hay que activar la opción *Orígenes desconocidos* que se encuentra en *ajustes* → *Seguridad* → *Orígenes desconocidos*.

Una vez copiado el archivo *apk* en la memoria del terminal móvil, se procede a la instalación. Para ello es necesario que el usuario haya descargado e instalado previamente, a través del *Google Play* de *Android*, cualquier explorador de archivos, tal como *Astro administrador de archivos*, *Root Explorer*, entre otros. Así, se ejecuta el explorador de archivos en el terminal móvil y se entra en la carpeta donde se transfirió el archivo *AppNaranja.apk*. Al seleccionar el archivo debe salir un menú contextual en el que se sugieren varias opciones. Entre ellas, *Instalar*. Si se elige *Instalar*, aparecerá una ventana que explica al usuario los permisos que concede a la aplicación. Si el usuario está de acuerdo, se elige *Aceptar*, con lo que la instalación queda realizada.

Una vez instalada la aplicación, se debe acceder al listado de aplicaciones del terminal, donde la aplicación objeto de este trabajo fin de grado vendrá representada por un icono característico (figura 1). Mediante la selección del icono será ejecutada la aplicación.

Figura 1: Icono de la aplicación *AppNaranja*

Para que la aplicación funcione correctamente debe tener conectado el terminal móvil a Internet. Dicha conexión se puede establecer mediante Wi-fi o mediante tarifa de datos. La conexión a Internet es necesaria porque la aplicación carga e inserta información de una base de datos externa alojada en Parse.com.

Licencia del programa

Normas generales

Esta aplicación es propiedad de la Universidad de Las Palmas de Gran Canaria y su uso debe estar sujeto a los términos y condiciones establecidas en esta licencia del programa, aceptándose todas las cláusulas. El uso de esta aplicación ha de estar bajo la expresa autorización del autor o de la tutora del proyecto o de la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria.

Derechos de autor

La aplicación y la documentación están protegidos por la ley de Propiedad Intelectual aplicable, así como por las disposiciones de los tratados internacionales. En consecuencia, el usuario podrá usar copia de esta aplicación, así como del código fuente de programación y de la documentación, siempre bajo la autorización del autor o de la tutora del proyecto o de la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria.

Garantía

Se garantiza el correcto funcionamiento de la aplicación en el momento de la instalación de acuerdo con las especificaciones vistas con anterioridad. La instalación de la aplicación no acarrea la aparición de defectos en los dispositivos que cumplan con las especificaciones técnicas.

Con la única excepción de lo expresamente expuesto en el párrafo anterior, la aplicación ha sido desarrollada sin garantías de ninguna clase. El autor no asegura, garantiza o realiza ninguna declaración respecto al uso o los resultados derivados de la utilización del programa o de la documentación. Tampoco se garantiza que la operación del programa sea interrumpida o sin errores. En ningún caso serán el autor, tutora o la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria responsables de los perjuicios directos, indirectos, incidentales, ejemplarios o consiguientes gastos, lucro cesante, pérdida de ahorros, interrupción de negocios, pérdida de información comercial o de negocio o cualquier otra pérdida que resulte del uso o de la incapacidad de usar la aplicación o la documentación. El usuario conoce y acepta que los derechos de licencia reflejan esta asignación de riesgo como el resto de cláusulas y restricciones.

Otras consideraciones

La fiabilidad de operación de la aplicación puede estar afectada por factores adversos, a los que se denominan “fallas del sistema”. En estos se incluyen errores en el funcionamiento del hardware del dispositivo, sistema operativo o entorno del mismo, compiladores o software de desarrollo usado para realizar la aplicación, software externo para el funcionamiento de la misma, errores de instalación, problemas de compatibilidad del software y hardware, fallos o funcionamiento incorrectos de equipos de control, fallas por uso, errores por parte del usuario de la aplicación o problemas en el acceso de Internet. En el supuesto de que cualquier disposición de esta licencia sea declarada total o parcialmente invalida, la cláusula afectada será modificada convenientemente de manera que sea ejecutable y una vez modificada, plenamente eficaz, permaneciendo el resto de este contrato en plena vigencia.

Esta licencia se regirá por las leyes de España. El usuario o licenciatario acepta la jurisdicción exclusiva de los tribunales de este país en relación con cualquier disputa que pudiera derivarse de la presente licencia.

Parte IV

Presupuesto

PRESUPUESTO

Introducción

En este capítulo se estimarán los gatos generados por el trabajo de fin de grado presentado en esta memoria. El presupuesto presentado se divide en las siguientes partes:

- Tarifa de honorarios por tiempo empleado.
- Amortización de los equipos empleados.
  - Amortización del material hardware.
  - Amortización del material software.
- Coste de acceso a Internet.
- Redacción de la documentación.
- Derechos de visado.
- Gastos de tramitación y envío.

Tarifa de honorarios por tiempo empleado

Este concepto contabiliza los gastos correspondientes a la mano de obra. Para realizar este cálculo, se propone la siguiente fórmula:

$$H = (14,48 \times Hn) + (20,27 \times He)$$

Siendo:

- **H**: honorarios.
- **Hn**: honorarios en jornada laboral normal.
- **He**: honorarios fuera de la jornada laboral normal.

En el trabajo fin de grado presentado en esta memoria, se ha empleado un periodo de aproximadamente **3** meses y **3** semanas de trabajo, trabajando alrededor de **5** horas diarias, en horario laboral normal.

Distribución de la temporización del trabajo fin de grado

El trabajo fin de grado se divide en cuatro etapas bien diferenciadas, que se pasan a definir:

Documentación

Esta primera etapa comprende el tiempo empleado en la recopilación de la información necesaria para alcanzar los conocimientos necesarios para la realización de la aplicación. De igual manera, esta etapa comprende el tiempo necesario para la valoración de las ventajas y desventajas de los lenguajes de programación disponibles para la realización del trabajo fin de grado, así como el proceso didáctico de documentación. Asimismo, una vez escogido el lenguaje de programación Java para Android, se ha realizado un estudio exhaustivo del mismo así como de las herramientas necesarias para su empleo.

Otros aspectos de documentación que se han tenido en cuenta, son los siguientes:

- El entorno de diseño lógico de documentos LaTeX.
- El entorno de diseño de diagramas **Cacoo.com**.
- El desarrollo de bases de datos utilizando **Parse.com**.
- El entorno de desarrollo **Eclipse**.
- El lenguaje de programación **Java**.
- El **SDK** de Android (*Software development kit*).
- El **ADT** de Android (*Android Development Tools*).
- El plugin **ObjectAid** de Eclipse para diagramas UML.

Definición

Este apartado está dedicado a la definición funcional y constitutiva de cada una de las partes del trabajo fin de grado. A medida que se desarrolle esta parte debe distinguirse la actividad que desempeña cada parte y la interrelación de las mismas.

Análisis

En esta etapa se establecen los módulos definitivos de los que dispondrá la aplicación, determinando sus funcionalidades. Este es un proceso puramente descriptivo, con la finalidad de sentar las bases lógicas sobre las que construir la aplicación. Se detallan los objetivos de cada módulo de manera que puedan ser agrupados en módulos de nivel superior, estableciendo una visión global del sistema.

Codificación

En esta etapa se realiza el desarrollo de la aplicación, es decir, se traduce lo establecido en el análisis a código, concretizando desde el concepto lógico a la variable funcional. Durante esta etapa se depura y verifica el código.

Asimismo, y a la vez que se han llevado a cabo las etapas anteriores, se ha ido elaborando la memoria final explicativa del proceso. Este último punto se contempla en la sección *Redacción del trabajo fin de grado*.

Cálculo de tarifa de honorarios por tiempo empleado

En la tabla 1 se detalla el tiempo empleado para cada una de las etapas anteriormente descritas y se realiza el cálculo de los honorarios.

Etapa	**Horas laborales**	**Honorarios**
Documentación	100	1.448 €
Definición	65	941,2 €
Análisis	70	1013,6 €
Codificación	140	2.027,2 €
Total		**5.430** €

Tabla 1: Honorarios por tiempo empleado

Por lo que sumadas cada una de las etapas, el cálculo de honorarios por tiempo empleado asciende a CINCO MIL CUATROCIENTOS TREINTA EUROS.

Amortización de los equipos empleados

Dentro de este concepto se considera tanto la amortización del hardware como del software empleado en la realización del trabajo fin de grado presentado. De este modo, se estipula el coste de amortización para un período de 3 años, utilizando un sistema de amortización lineal o constante. En este sistema, se supone que el inmovilizado material se deprecia de forma constante a lo largo de su vida útil. La cuota de amortización anual se calcula haciendo uso de la siguiente fórmula:

$$C = \frac{Vad - V}{N}$$

Donde:

- **C**: cuota de amortización anual.
- **Vad**: valor de la adquisición.
- **V**: valor residual.
- **N**: número de años de vida útil de la adquisición.

Siendo el valor residual el valor teórico que se supone tendrá el elemento en cuestión después de su vida útil, teniendo en cuenta los índices de depreciación actual. En el caso del hardware y del software son 3 años (al 33 % de depreciación máximo por año).

Amortización del material hardware

Debido a que el trabajo fin de grado se ha elaborado en un periodo inferior a 3 años, que es el periodo en que se calcula la amortización de material hardware, se realizará una amortización equiparable al período de duración del mismo. Según esto, se obtienen los gastos expuestos en la tabla 2.

Por lo tanto, el coste total de hardware asciende a la cantidad de CIENTO TREINTA Y CINCO EUROS Y CUARENTA Y UN CÉNTIMOS DE EURO.

Amortización del material software

El coste total del software utilizado es la suma de las herramientas software empleadas para la realización del trabajo de fin de grado (tabla 3). De esta manera se describe cada uno de las herramientas utilizadas y el coste supuesto para su utilización usando la fórmula de amortización anteriormente citada.

Por tanto, el coste total de software asciende a DIEZ EUROS Y VEINTE Y NUEVE CÉNTIMOS DE EURO.

Descripción	**Valor de adquisición**	**Tiempo de uso**	**Coste anual**	**Total**
Ordenador Ultrabook Samsung Series 5 Ultra, Intel Core i3 , 4 Gb de RAM	800,00 €	3,75 meses	266,67 €	83,33 €
Teléfono móvil Samsung S3	400,00 €	3,7 meses	133,33 €	41,66 €
Impresora Samsung ML2160	100,00 €	3,7 meses	33,33 €	10,42 €
Total	1.300,00 €	-	433,33	**135,41** €

Tabla 2: Precios y costes de amortización del hardware

Descripción	**Valor de adquisición**	**Tiempo de uso**	**Coste anual**	**Total**
Windows 8.1	0,00 €	3,7 meses	0,00 €	0,00 €
Eclipse	0,00 €	3,7 meses	0,00 €	0,00 €
SDK y ADT Android	0,00 €	3,7 meses	0,00 €	0,00 €
Parse.com (base de datos)	0,00 €	3,7 meses	0,00 €	0,00 €
WinEdt (entorno de LaTeX)	36,89 €	3,7 meses	12,30 €	3,84 €
Balsamiq Mockups (para desarrollo de muestras de la aplicación en formato pdf)	62 €	3,7 meses	20.67 €	6,45 €
Cacoo.com (diagramas e ilustraciones)	0,00 €	3,7 meses	0,00 €	0,00 €
Total	98,89 €	-	32,97 €	**10.29** €

Tabla 3: Precios y costes de amortización del software

Coste de acceso a Internet

Para la conexión a Internet se dispone de una solución ADSL a 10Mbps cuyo coste mensual es de 35 euros. Debido a que se ha usado dicha conexión durante cada una de las etapas de creación del mismo (3,7 meses), el coste de acceso a Internet se traduce a un total de CIENTO TREINTA Y UN EUROS Y VEINTE Y CINCO CÉNTIMOS DE EURO.

Redacción de la documentación

Para calcular el valor monetario de la redacción del trabajo fin de grado se aplicará la fórmula siguiente:

$$R = 0,05 \times P \tag{1}$$

Siendo:

- **R**: coste de la redacción.
- **P**: presupuesto.

El valor de P se obtiene sumando los costes de las secciones anteriores tal y como muestra la tabla 4.

Concepto	**Coste**
Tarifa de honorarios por tiempo empleado	5.430 €
Amortización del material hardware	135,41 €
Amortización del material software	10,29 €
Coste de acceso a Internet	131,25 €
Total	**5.706,95 €**

Tabla 4: Precios y costes de la ejecución del trabajo fin de grado más la amortización y acceso a Internet

De este modo, se obtiene que:

$$R = 0,05 \times 5.706,95 = 285,35 \tag{2}$$

Al coste de redacción obtenido hasta el momento se le deben añadir otros gastos, quedando el importe final de redacción del trabajo fin de grado como se describe en la tabla 5. Por lo tanto, el coste final de redacción del trabajo fin de grado asciende a un total de TRESCIENTOS CATORCE EUROS Y CINCUENTA CÉNTIMOS DE EURO.

Derechos de visado

El COITT establece que los derechos de visado para las aplicaciones telemáticas en el año 2012 se calculan de acuerdo con la siguiente ecuación:

$$V = 0,0035 \times P \times C \tag{3}$$

Siendo:

Concepto	Coste
Redacción del trabajo fin de grado	285,35 €
Papel de impresión	5 €
Tinta de impresión negra(0.05 €)	4,35 €
Tinta de impresión color(0.55 €)	12,65 €
Encuadernación	6,00 €
CDs de 700MB	1,15 €
Total	**314,50 €**

Tabla 5: Coste final de redacción del trabajo fin de grado

- **V**: coste del visado.
- **P**: presupuesto.
- **C**: coeficiente reductor en función del presupuesto.

El valor del coeficiente se obtiene de la tabla 6, mientras que el valor de P es el valor total del presupuesto, que se pasa a calcular en la tabla 7.

Coste del presupuesto (€)	Factor de Correlación (C)
Hasta 30.050	1
Exceso de 30.050 hasta 60.101	0.9
Exceso de 60.101 hasta 90.151	0.8
Exceso de 90.151 hasta 120.202	0.7
...	

Tabla 6: Tabla de coeficientes para el cálculo del visado

Concepto	Coste
Coste del tiempo empleado en la ejecución, amortización y acceso a Internet	5.706,95 €
Redacción del trabajo fin de grado	314,50 €
Total	**6.021,45 €**

Tabla 7: Cálculo total P para el cálculo del visado (Presupuesto base)

Por lo tanto, el valor del visado será de:

$$V = 0,0035 \times 6.021,45 \times 1 = 21,08 \tag{4}$$

Los costes de derechos de visado del trabajo fin de grado ascienden a un total de VEINTE Y UN EUROS CON OCHO CÉNTIMOS DE EURO.

Gastos de tramitación y envío

Los gastos de tramitación y envío según la tarifa asciende a SEIS EUROS por cada documento visado de forma telemática.

Presupuesto antes de impuestos

Sumando todos los conceptos calculados hasta el momento, se obtiene el presupuesto, sin incluir los impuestos, que se muestra en la tabla 8.

Concepto	**Coste**
Presupuesto base	6.021,45 €
Derechos de visado	21,08 €
Gastos de tramitación y envío	6,00 €
Total	**6.048,53** €

Tabla 8: Presupuesto total sin impuestos

El presupuesto calculado, antes de incluir los impuestos, asciende a SEIS MIL CUARENTA Y OCHO EUROS CON CINCUENTA Y TRES CÉNTIMOS DE EURO.

Presupuesto incluyendo impuestos

Al presupuesto calculado anteriormente hay que incluirle un 7 % de IGIC obteniendo el coste del presupuesto final (tabla 9).

Concepto	**Coste**
Total (sin IGIC)	6.048,53 €
IGIC (7 %)	423,40€
Total	**6.471,93** €

Tabla 9: Presupuesto total

Por tanto, el presupuesto total, incluyendo impuestos, asciende a la cantidad de SEIS MIL CUATROCIENTOS SETENTA Y UN EUROS CON NOVENTA Y TRES CÉNTIMOS DE EURO.

Las Palmas de Gran Canaria, a 05 de Julio de 2014

Fdo: Adolfo Luzardo Cabrera

Parte V

Conclusiones y mejoras

Apéndice **A**

CONCLUSIONES Y MEJORAS

En este capítulo se detallarán los objetivos que se pretenden conseguir con la elaboración de este TFG, así como algunas mejoras que se podrán aplicar a la *Aplicación Android para pedir cita previa en peluquerías*.

A.1 Conclusiones

Mediante la realización de este TFG se pretende hacer más sencillo y rápido el proceso de pedir cita en una peluquería. La mayoría de las peluquerías permiten pedir cita previa, pero utilizan métodos tradicinales para la gestión de las mismas. Para pedir cita en dichas peluquerías necesitas llamar por teléfono o acudir presencialmente, lo que supone un trabajo añadido y un gasto de dinero tanto para el cliente como para el propio empresario.

La principal idea es aprovechar el gran potencial de las aplicaciones móviles y el uso masivo en la población de los Smartphones, para mejorar la gestión de las citas previas en las peluquerías y permitir al usuario pedir cita mediante una aplicación.

De esta forma se evitaría cualquier tipo de coste y espera en el proceso de pedir cita previa en una peluquería. Además la peluquería dispone de un sistema de gestión de citas transparente y organizado con todas las citas de sus clientes.

A.2 Mejoras

Implementación de una nueva aplicación servidor para el empresario y trabajadores de la peluquería. Con la aplicación servidor se podrá gestionar festivos, gestionar citas de los clientes, añadir nuevas peluquerías, añadir nuevos servicios o cambiar los precios, añadir ofertas o avisos publicitarios que sean visibles en la aplicación cliente *AppNaranja*.

Mejoras en la aplicación cliente *AppNaranja* implementada en este TFG:

- Notificaciones nativas de Android, se avisará mediante la barra de notificaciones cuando se produzcan los siguientes eventos:
  - Cita cancelada
  - Nueva peluquería
  - Cambios de precios en servicios
  - Ofertas o avisos publicitarios
- Implementación de un calendario donde se pueda elegir la fecha y la hora.
- Integrar la aplicación con las redes sociales (Twitter, Facebook y Google +).