Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Sistema electrónico de bajo coste para el control de asistencia por huella dactilar

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Carles Pérez Revert

Tutor: Juan Carlos Ruiz García

2º Tutor: Roberto Capilla Lladró

Curso 2019/2020

Resumen

Cada vez es más necesario autentificar la identidad de las personas en diferentes procesos del día a día, ya sea para realizar una acción a través de una aplicación móvil bancaria o bien para registrar la asistencia en el trabajo. La autenticación biométrica cada vez está más presente en nuestra vida cotidiana. Esta tecnología se ha abaratado propiciando su uso en teléfonos, tabletas y portátiles, teniendo, además, una gran aceptación por parte de fabricantes y usuarios. No obstante, a pesar de estos avances tecnológicos, el alumnado de la UPV sigue registrando su asistencia a clase firmando en papel el parte de seguimiento de actividades docentes.

En este proyecto se ha continuado desarrollando una aplicación Android existente cuyo cometido era, y es, automatizar y agilizar el proceso de fichaje del alumnado. Esta aplicación hace uso de las herramientas que incorpora la Tarjeta Universitaria Inteligente (TUI), como son el NFC y el código QR, para obtener los datos tanto de los profesores como de los alumnos.

En este trabajo se describe el proceso de mantenimiento y evolución llevado a cabo en la aplicación. Se han corregido errores, aplicado patrones de diseño para mejorar la arquitectura de la solución, renovado por completo la IU y mejorado y añadido nuevas funcionalidades. Entre las nuevas funcionalidades destacan la autenticación biométrica del usuario (tanto de huella dactilar como de reconocimiento facial), las funciones de voz y la vinculación de la TUI al teléfono inteligente.

Palabras clave: Android, autenticación biométrica, funciones de voz, huella dactilar, NFC, QR, TUI, UPV.

Resum

Cada vegada és més necessari autentificar la identitat de les persones en diferents processos del dia a dia, ja siga per a realitzar una acció a través d'una aplicació mòbil bancària o bé per a registrar l'assistència al treball. L'autenticació biomètrica cada vegada està més present en la nostra vida quotidiana. L'abaratiment d'aquesta tecnologia ha propiciat la utilització en telèfons, tauletes i portàtils i, a més, aquesta ha tingut una gran acceptació per part de fabricants i usuaris. No obstant això, malgrat aquest avanços tecnològics, l'alumnat de la UPV continua registrant la seua assistència a classe firmant en paper el document de seguiment d'activitats docents.

En aquest projecte s'ha continuat desenvolupant una aplicació Android existent, l'objectiu de la qual era, i és, automatitzar i agilitzar el procés de fitxatge de l'alumnat. Aquesta aplicació fa ús de les ferramentes que incorpora la Targeta Universitària Intel·ligent (TUI), com són el NFC y el codi QR, per tal d'obtenir les dades tant dels professors com dels alumnes.

En aquest treball es descriu el procés de manteniment i evolució portat a cap en l'aplicació. S'han corregit errors, aplicat patrons de disseny per a millorar l'arquitectura de la solució, renovat per complet la IU i millorat i afegit noves funcionalitats. Entre les noves funcionalitats destaquen l'autenticació biomètrica (tant d'empremta dactilar com de reconeixement facial), les funcions de veu i la vinculació de la TUI al telèfon intel·ligent.

Paraules clau: Android, autenticació biomètrica, empremta dactilar, funcions de veu, NFC, QR, TUI, UPV.

Abstract

It is more and more necessary to authenticate the identity of people in everyday processes, either to perform an action through a mobile banking application or to record attendance at work. Biometric authentication is increasingly present in our daily lives. This technology has become cheaper and is being used in telephones, tablets and laptops, and is also being widely accepted by manufacturers and users. However, despite these technological advances, students at the UPV continue to record their attendance at classes by signing an attendance form.

This project has focused on the improvement of to develop an existing Android application whose purpose was, and is, to automate and speed up the process of signing up students. This application makes use of the tools incorporated by the TUI – University Smart Card – (NFC and QR code) to obtain the data of both teachers and students.

This document describes the maintenance, improvement and evolution process carried out in the application. Bugs have been corrected, architectural patterns applied, the TUI completely renewed and new functionalities improved and added. Among the new functionalities, the biometric authentication of the user (both fingerprint and facial recognition), the voice functions and the linking of the TUI to the smartphone stand out.

Keywords: Android, biometric authentication, fingerprint, NFC, QR, TUI, UPV, voice capabilities.

Dedicatoria

Este trabajo me gustaría dedicarlo a una persona muy especial: Juan Ignacio Delgado Alemany. Gracias por hacerme reír y ver la vida desde otra perspectiva en los momentos más difíciles y tristes. Eres una fuente de inspiración.

Tabla de contenido

Índice de figuras

Índice de tablas

Índice de fragmentos de código

Glosario

API	*Applications Programming Interface*
BLE	*Bluetooth Low Energy*
CP	*Clave Primaria*
CPU	*Central Processing Unit*
CRUD	*Create, Read, Update, Delete*
DNI	*Documento Nacional de Identidad*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
IU	*Interfaz de Usuario*
JVM	*Java Virtual Machine*
MVC	*Modelo Vista Controlador*
NFC	*Near Field Communication*
PDF	*Portable Document Format*
PIN	*Personal Identification Number*
P2P	*Peer-to-peer*
RF	*Requisito Funcional*
RFID	*Radio Frequency Identification*
RNF	*Requisito No Funcional*
SDK	*Software Development Kit*
SQL	*Structured Query Language*
TFG	*Trabajo Fin de Grado*
TUI	*Tarjeta Universitaria Inteligente*
UML	*Unified Modeling Language*
UPV	*Universitat Politècnica de València*
URL	*Uniform Resource Locator*
UX	*User Experience*
XML	*Extensible Markup Language*

1. Introducción

1.1. Motivación

En 1888, durante la época de la industrialización, se patentó el primer reloj de fichar. Después de la Revolución Industrial el horario laboral ya no lo marcaba el amanecer del sol, eran las sirenas de las fábricas las que indicaban el inicio y el final de la jornada de trabajo [1].

Más de un siglo más tarde los obreros siguen fichando cada día al entrar y al salir de su trabajo. No obstante, aquellos relojes han quedado en desuso. Actualmente, existen sistemas de control horario biométricos y basados en la identificación de radio frecuencias (RFID).

Los alumnos de la Universitat Politècnica de València (UPV) también registran su asistencia en cada clase a la que asisten, aunque no utilizan ninguna de las dos tecnologías citadas anteriormente. Por cada clase que se imparte se imprime un folio (incluso dos dependiendo del número de alumnos matriculados en ese grupo) y los alumnos firman en la casilla correspondiente.

1.2. Objetivo

El objetivo de este proyecto es hacer uso de tecnologías presentes en el día a día para facilitar, a alumnos y profesores, el proceso de fichaje que llevan a cabo todos los días en repetidas ocasiones.

El resultado buscado es una aplicación (de ahora en adelante también encontraremos aplicación como *app*) Android que simplifique y agilice el proceso de fichaje. Además, también deberá soportar la gestión y consulta de partes de asistencia generados previamente por el profesor.

De esta manera, mediante la tecnología RFID, los alumnos podrán fichar utilizando su tarjeta de la UPV, la cual es una de las denominadas Tarjetas Universitarias Inteligentes (TUI). Por último, la aplicación deberá tener una vista para estudiantes que permitirá fichar al alumno, lo cual ofrecerá otra alternativa para poder registrar la asistencia.

1.3. Punto de partida

Este proyecto no empieza de cero. El punto de partida es un proyecto realizado previamente por otro alumno de la UPV como TFG [2]. Las características de la solución existente y sus funcionalidades se comentarán en capítulos posteriores.

A esta aplicación le añadiremos nuevas funcionalidades, se le corregirán *bugs*[1] y se mejorarán tanto aspectos internos (refactorizaciones[2] de código) como externos (IU).

Es importante destacar lo siguiente: una **aplicación heredada** (*legacy app* o *legacy application*, en inglés) es un programa de *software* que está desactualizado u obsoleto, pudiendo resultar inestable [3]. **No obstante**, en esta memoria cuando hablemos de **heredar** o de **aplicación heredada** nos referiremos a **heredar**[3] en el sentido literal de esta palabra, es decir, recibir algo.

1.4. Estructura

En el capítulo 2 haremos un análisis del estado del registro de asistencia a clase de la UPV, hablaremos sobre el fichaje en el mundo laboral y valoraremos el estado actual de la autenticación biométrica. También hablaremos de la solución actual y la compararemos con posibles competidores.

En el capítulo 3 realizaremos un análisis extendido de la solución actual. Explicaremos su funcionamiento y qué aspectos tienen margen de mejora.

En el capítulo 4 haremos un análisis más general del problema. Valorando qué nuevas funcionalidades podemos incorporar a las ya existentes y de qué manera las podemos implementar.

En el capítulo 5 diseñaremos la solución resultante del análisis realizado y hablaremos de las tecnologías elegidas para llevar a la práctica dicho diseño.

En el capítulo 6 trataremos los diferentes aspectos del proceso de implementación. Describiremos cómo se ha implementado y argumentaremos los motivos de las diferentes decisiones que se tomarán.

En el capítulo 7 presentaremos las pruebas realizadas para comprobar que se han cumplido los requisitos y que los cambios introducidos son satisfactorios.

Finalmente, en los capítulos 8 y 9 concluiremos el proyecto. Principalmente hablaremos de la consecución de los objetivos marcados al inicio del proyecto y de las líneas futuras de desarrollo del proyecto.

[1] Un *bug* es un error que presenta una aplicación. Como es común denominarlo de esta manera, en esta memoria siempre nos referiremos a los errores como *bugs*.
[2] Una refactorización consiste en una centralización de código con ánimo de mejorar la mantenibilidad de dicho código sin cambiar el comportamiento de la aplicación.
[3] Según la Real Academia Española, heredar es recibir algo correspondiente a una situación anterior.

2. Estado del arte

2.1. Control de asistencia en la UPV

Actualmente, los alumnos de la UPV firman en la casilla correspondiente para registrar su asistencia a una clase. Normalmente, el profesor pasa el parte al alumno más cercano a él para que firme y lo pase al siguiente alumno. Este, a su vez, hará lo mismo. Este funcionamiento en cadena tiene sus inconvenientes.

Cuando una clase empieza el parte de firmas se encuentra circulando de alumno en alumno. Esto ocasiona distracciones entre los alumnos que les hace dejar de prestar de atención a lo que el profesor está explicando en ese momento.

En alguna ocasión también se ha visto un parte de asistencia firmado abandonado en alguna mesa del aula debido a que al profesor se le ha olvidado recogerlo al final de la clase.

Otras veces son los alumnos los que se acercan al final de la clase a la mesa del profesor para firmar el parte. Cuando esto sucede es normal que se produzcan colas y que, desde que firma el primer alumno hasta que lo hace el último, pasen varios minutos.

Además, es habitual que un alumno firme por otro, es decir, se producen falsos registros, lo cual es difícil de prevenir siguiendo este método.

Por último, no se debe obviar que por cada clase que se imparte se utiliza un folio (a veces dos). En la UPV cada día se imparten cientos de clases, por lo que el gasto de papel cada día es cuantioso. Esto tiene un impacto económico y ambiental.

No obstante, existe una alternativa al parte de firmas, aunque son pocos profesores los que la practican. Esta alternativa consiste en entrar en la Intranet de la UPV, seleccionar la clase que se está impartiendo y pasar lista en voz alta. Así, el profesor va marcando en un listado los alumnos que han asistido a clase. Este método no resulta nada ágil y suele llevar bastante tiempo, sobre todo si en ese grupo hay muchos estudiantes.

2.2. Aplicaciones en el ámbito educativo

Dinatia[4], iEduca[5], Teacher Aide Pro[6], Additio[7] son aplicaciones que permiten llevar un control de la asistencia. Pero también son aplicaciones que integran una gran variedad de funcionalidades porque forman parte de un ecosistema mayor. No son apps enfocadas para el uso universitario, están diseñadas sobre todo para la implantación en institutos ya que, por ejemplo, envían notificaciones a los padres. Es decir, no son aplicaciones enfocadas realmente a tener un control de la asistencia, aunque tengan integrada esta funcionalidad. Además, en este

[4] Más información en: https://www.dinantia.com/es
[5] Más información en: https://ieduca.com/videos/
[6] Más información en: http://www.teacheraidepro.com/
[7] Más información en: https://www.additioapp.com/es

tipo de sistema, el profesor tendría que pasar lista en voz alta, lo cual no mejoraría el estado actual.

2.3. Estado actual de la aplicación

Como se ha explicado en la introducción, este proyecto parte de otro anterior. La aplicación se creó para substituir los métodos actuales de control de asistencia.

Actualmente esta aplicación permite a los alumnos registrar su asistencia acercando su TUI al teléfono inteligente del profesor. Por tanto, la TUI es el elemento protagonista de esta app, ya que se utilizan diferentes recursos que esta tarjeta incorpora.

Al tratarse de una aplicación externa a la UPV, esta no puede acceder ni hacer uso de ningún tipo de dato privado de la UPV; además esta institución tampoco ofrece ninguna API para sistemas externos. Esto provoca que algunos procesos que se llevan a cabo en la aplicación sean más complejos de lo que podrían realmente ser. A pesar de ello, la aplicación consigue crear registros de asistencia de manera sencilla.

El problema de esta aplicación ahora mismo es que no incluye una forma de autenticación alternativa y/o complementaria a la TUI. Esto es un problema porque no se verifica la identidad física del alumno firmante, con lo que cualquiera que tuviera acceso a la TUI de un alumno podría firmar por él.

2.4. El fichaje en el mundo laboral

En febrero de 2019 se aprobó en el Congreso de los Diputados que a partir del día 12 de mayo de ese mismo año sería obligatorio para las empresas llevar un registro de la jornada laboral de cada trabajador. Las empresas que, por entonces, no disponían de este tipo de control tuvieron que buscar la manera de ponerse al día con la nueva legislación laboral[8].

Este hecho propició un aumento en el interés por *software* y/o *hardware* que pudiera satisfacer estas obligaciones. De este modo, podemos observar que el control de asistencia también es un tema destacado en el mundo laboral y no sólo en el ámbito universitario. Existen diferentes formas de llevar este control, desde aplicaciones móvil o web hasta dispositivos *hardware* con autenticación biométrica.

2.5. Biometría y autenticación

La autenticación biométrica es el proceso que permite confirmar que un usuario es quien dice ser (autenticación) mediante sus características biológicas y de comportamiento (biometría). "*Los rasgos de las manos y la cara, las huellas dactilares y los patrones del iris son ejemplos de características biológicas. […] Las tecnologías biométricas están listas para impregnar casi todos los aspectos de la economía y nuestra vida cotidiana*" [4]. Esta cita es del año 2005; en la

[8] En el siguiente enlace podemos consultar en detalle esta ley https://controladordepresencia.com/wp-content/uploads/2019/04/BOE-A-2019-3481.pdf.

actualidad, podemos encontrar *hardware* de autenticación biométrica en la mayoría de los teléfonos inteligentes.

El abaratamiento de esta tecnología junto a la necesidad de verificar la identidad del propietario del teléfono de forma rápida y segura ha provocado este gran aumento del sensor de huellas dactilar. Es decir, la autenticación biométrica en teléfonos inteligentes ha tenido una gran aceptación entre los desarrolladores y entre los usuarios.

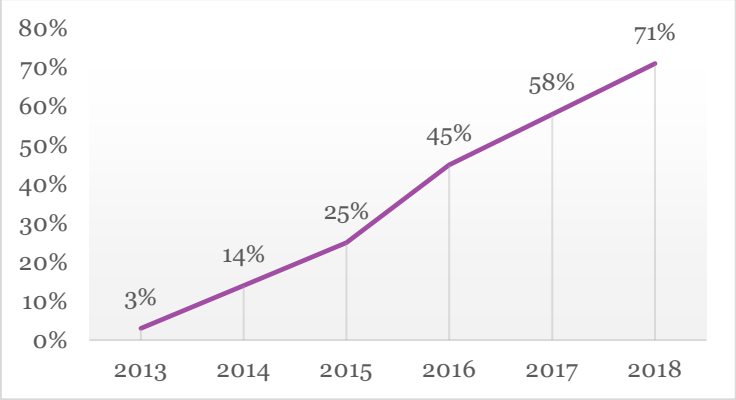

Figura 1. Penetración del sensor de huellas dactilares de los teléfonos inteligentes a nivel mundial. Fuente: Counterpoint Research – Component Tracker

El lector de huellas no sólo ha tenido éxito en los teléfonos, también lo ha tenido en los ordenadores portátiles. Por ejemplo, Windows, MacOS y Linux[9] ya soportan la autenticación a través de la huella dactilar del propietario del ordenador.

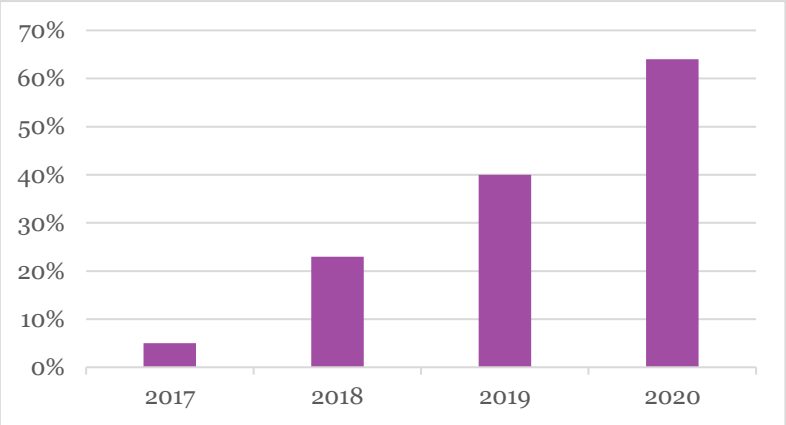

Figura 2. Penetración de la función de reconocimiento facial en los teléfonos inteligentes. Fuente: Counterpoint Research – Component Tracker

El sensor de huellas es el elemento principal de autenticación biométrica en los teléfonos, pero no es el único. Cada vez más dispositivos incorporan el reconocimiento facial, el cual ha experimentado mejorías con el tiempo. Una clara mejora es la aplicación de tecnología de detección en 3D para identificar rostros, mejorando, de esta forma, al tradicional reconocimiento de caras en 2D [5].

[9] Para saber más sobre cómo Windows (https://support.microsoft.com/en-us/help/4468253/windows-10-sign-in-options-and-privacy), Apple (https://support.apple.com/en-us/HT207054) o algunas distribuciones de Linux (https://ubuntu.com/search?q=fingerprint) gestionan la autenticación por huella dactilar puedes consultar los enlaces.

3. Análisis de la solución existente

En este capítulo analizaremos diferentes aspectos de la solución que nos va a servir como punto de partida. Asimismo, se explicará cómo funciona actualmente la aplicación.

3.1. Tecnologías

3.1.1. Android

Android es un sistema operativo desarrollado por Google cuyo núcleo está formado por Linux[10]. Las cualidades de Android más útiles resultado para este proyecto son las siguientes:

- **Portable**. Gracias a la JVM[11] se puede ejecutar en cualquier tipo de CPU ya que esta proporciona un entorno de ejecución en el que se puede ejecutar el código de bytes de Java
- **Adaptable a diferentes tipos de *hardware***. Una misma aplicación desarrollada en Android puede ser instalada en tabletas, móviles, cámaras o TV, entre otros dispositivos. Por ejemplo, en iOS tendríamos que desarrollar una aplicación diferente para cada tipo de dispositivo.
- **Servicios**. Android nos facilita el uso de servicios, como bases de datos SQLite o la localización.
- **Seguro**. A partir de la versión 6.0, una aplicación tiene que solicitar permiso al usuario antes de hacer uso de ciertos recursos (por ejemplo, la ubicación del dispositivo). Además, hereda de Linux que los programas se ejecuten de manera independiente, es decir, aislados entre sí [6].

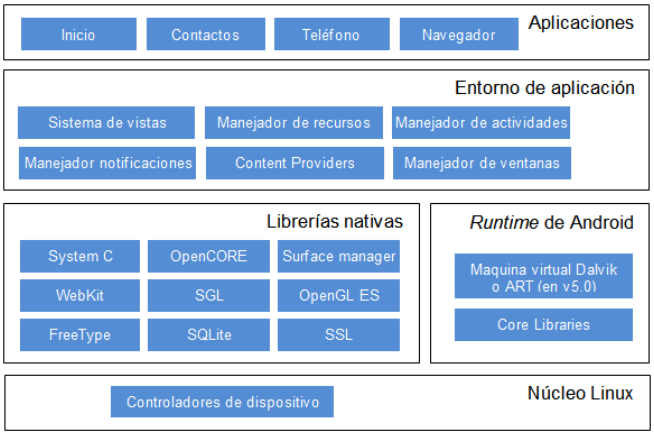

Figura 3. Arquitectura de Android.
Fuente: Curso de Android (Máster en Desarrollo de Aplicaciones Android UPV)

También es importante destacar que, actualmente, Android es el sistema operativo más utilizado por teléfono inteligentes del mundo.

[10] Más información sobre Linux: https://www.redhat.com/es/topics/linux
[11] Más información sobre la JVM: https://www.javatpoint.com/jvm-java-virtual-machine

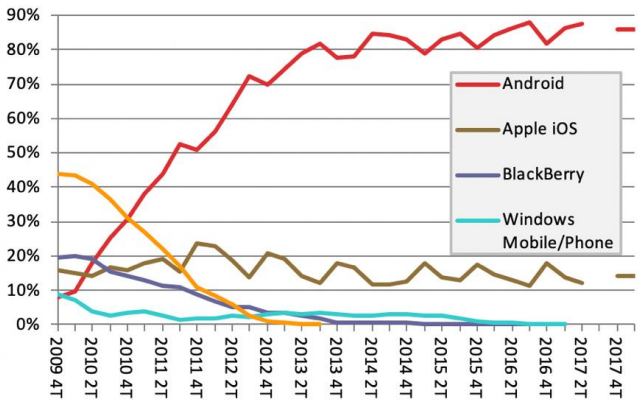

Figura 4. Porcentaje de teléfonos inteligentes vendidos en todo el mundo hasta el primer trismetre de 2018, según el sistema operativo. Fuente: Curso de Android (Máster en Desarrollo de Aplicaciones Android UPV), con datos de Gartner Group.

3.1.2. Java

Java es un lenguaje de programación de alto nivel orientado a objetos, fuertemente tipado y que se inspiró en el lenguaje C / C ++. Java fue diseñado para ser *"Write Once, Run Anywhere"* lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra [7].

3.1.3. SQLite

SQLite es una biblioteca que implementa un motor de base de datos SQL y que lee y escribe directamente en archivos de disco[12]. Además, para utilizar SQLite, no se necesita hacer ninguna instalación porque Android ya incorpora este servicio. La mayor parte de la información utilizada por la aplicación se almacena en base de datos.

3.1.4. XML

XML es un metalenguaje (no dispone de un conjunto cerrado de etiquetas) de marcas que se utiliza para almacenar información[13]. XML se utiliza para almacenar información, junto con SQLite. Además, también se utiliza para cargar información externa en el sistema.

3.1.5. TUI

La UPV utiliza la TUI para acreditar a sus miembros. Esta está equipada con diferentes tecnologías, las cuales aprovecha para extraer información del usuario.

3.1.6. NFC

La tecnología NFC surgió en 2003 y su aplicación principal fue en el sector comercial. También están surgiendo muchas aplicaciones nuevas e interesantes en otros campos de la educación y la

[12] Para más información sobre SQLite: https://www.sqlite.org/about.html
[13] Para saber más sobre XML: https://sites.google.com/site/todoxmldtd/

salud [8]. Esta tecnología se utiliza para obtener el identificador del usuario cuando este aproxima su TUI al teléfono inteligente.

Figura 5. Anverso y reverso de la TUI de la UPV. Fuente: Web de la UPV

3.1.7. Código QR

Los códigos QR son la evolución de los códigos de barras. Normalmente un código QR contiene una URL, aunque también es habitual que contenga un PDF o alguna ubicación, entre otros usos. Es sencillo identificarlos ya que son cuadrados y, a su vez, incluyen tres cuadrados en sus esquinas (superiores e inferior izquierda).

Figura 6. Ejemplo de código QR.

Cuando se escanea el código QR de la TUI desde la aplicación, esta obtiene los datos del usuario y su relación con la institución (alumno o profesor).

3.1.8. Librerías

El proyecto que hemos heredado integra diferentes librerías Android para la lectura de códigos QR y para la gestión de información HTML:

- **ZXing Android Embedded**. Es una librería de escaneo de código de barras y código QR para Android[14].
- **Jsoup**. Es una biblioteca Java para trabajar con HTML. Proporciona una API muy conveniente para obtener URLs y extraer y manipular datos[15].

También se integran las librerías **DroidText** (gestión de PDF) y **JavaMail** (envío de correos electrónicos), no obstante, estas presentan algunos problemas que se comentarán en puntos posteriores.

[14] El proyecto se encuentra disponible en https://github.com/journeyapps/zxing-android-embedded
[15] Su página oficial ofrece más información al respecto: https://jsoup.org/

3.2. Funcionamiento

Cuando iniciamos la aplicación, en la pantalla principal, nos aparece una especie de menú con tres opciones. La primera opción (icono de la mano con una tarjeta) está relacionada con la creación de partes de asistencia. La segunda opción (icono de la tuerca) está relacionada con la gestión de los datos de asignaturas, alumnos, profesores y grupos. La tecera (icono de un listado) nos permite buscar partes previamente generados.

Figura 7. Pantalla principal de la aplicación.

Iniciar parte. Cada vez que un profesor desea iniciar un parte, este tiene que acercar su tarjeta al teléfono. Luego selecciona la asignatura y el tipo de clase que va a impartir, normal o recuperación. Una vez completados estos pasos, los alumnos ya pueden fichar acercando su TUI al dispositivo del profesor. Finalmente, el profesor genera un PDF con los datos del parte de asistencia.

Gestión de datos. En esta parte de la aplicación se realizan las operaciones CRUD[16] sobre asignaturas, grupos, profesores y alumnos. Las asignaturas y los grupos se añaden a través de ficheros XML que deben estar en la carpeta de descargas (en el **Anexo A** se encuentra más información al respecto). Los profesores y los alumnos se añaden manualmente en cada asignatura con estos dos pasos: acercando la TUI para leer el identificador y escaneando el código QR de la TUI para obtener el resto de los datos.

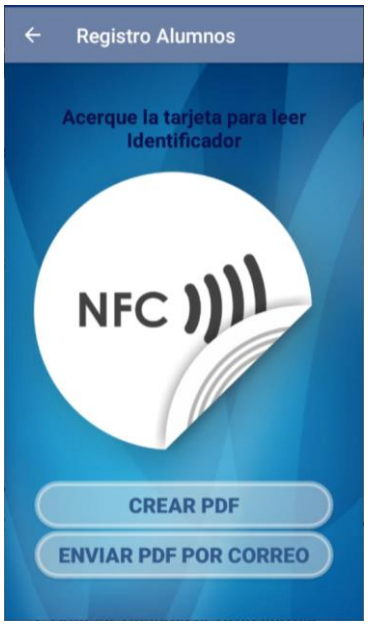

Figura 8. Pantalla en la que los alumnos registran su asistencia.

Figura 9. Pantalla de edición de datos de una asignatura.

[16] Crear, leer, actualizar y borrar. Son las funciones básicas en bases de datos.

Consulta de partes. Cuando se selecciona una opción, se muestra una pantalla con tres campos que son obligatorios rellenar para poder buscar un parte. Una vez introducidos los datos correctamente, muestra los datos del parte en cuestión. Sin embargo, el listado de los alumnos que han asistido a dicha clase no se muestra.

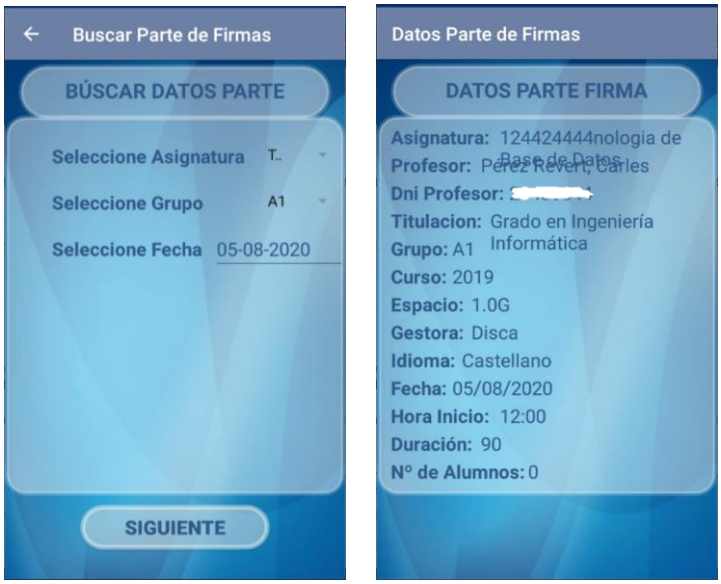

Figura 10. Pantallas relativas a la funcionalidad de consulta de partes.

3.3. Carencias en diseño y calidad

Existen unas guías de **Diseño y calidad** en Android, cuyo fin es especificar las características de calidad fundamentales que deben exhibir todas las aplicaciones[17]. La aplicación no sigue algunos de estos criterios. A continuación, podemos observar los aspectos en concreto que no se respetan junto con una descripción de lo que la app debería hacer para cumplir estos criterios.

Tabla 1. Criterios de diseño y calidad que no cumple la solución existente. junto con una descripción de lo que la app debería hacer para cumplir estos criterios.

AREA	ID	Descripción
Diseño estándar	UX-B1	La app cumple con las pautas de diseño de Android y utiliza patrones e íconos de IU comunes
Estabilidad	PS-S1	La app no falla, no impone el cierre, no se inmoviliza ni funciona de ningún otro modo anormal en ninguno de los dispositivos donde esté instalada.
Rendimiento	PS-P1	La app se carga rápidamente o le proporciona al usuario información en pantalla (un indicador de progreso o una señal similar) en caso de que demore más de dos segundos en cargarse.

[17] Esta guía se puede consultar en https://developer.android.com/docs/quality-guidelines/core-app-quality

Calidad visual	PS-V2	La app muestra texto y bloques de texto de forma aceptable. 1. La composición es aceptable en todos los factores de forma compatibles. 2. No se visualizan letras ni palabras cortadas. 3. No se visualizan ajustes automáticos de línea incorrectos en botones ni íconos. 4. Hay espacio suficiente entre el texto y los elementos que lo rodean.
Bibliotecas	SC-W1	Todas las bibliotecas, SDK y dependencias están actualizadas.
Datos	SC-D1	Todos los datos privados se guardan en el almacenamiento interno de la app.

3.4. Problemas de interfaz

La aplicación presenta graves problemas de interfaz. Android se utiliza en una amplia gama de dispositivos, por lo que una buena interfaz tiene que ser capaz de adaptarse a diferentes tamaños de pantalla. Mientras que en algunos los datos sí que se pueden leer, en otros resultan ilegibles (**Calidad Visual, PS-V2**).

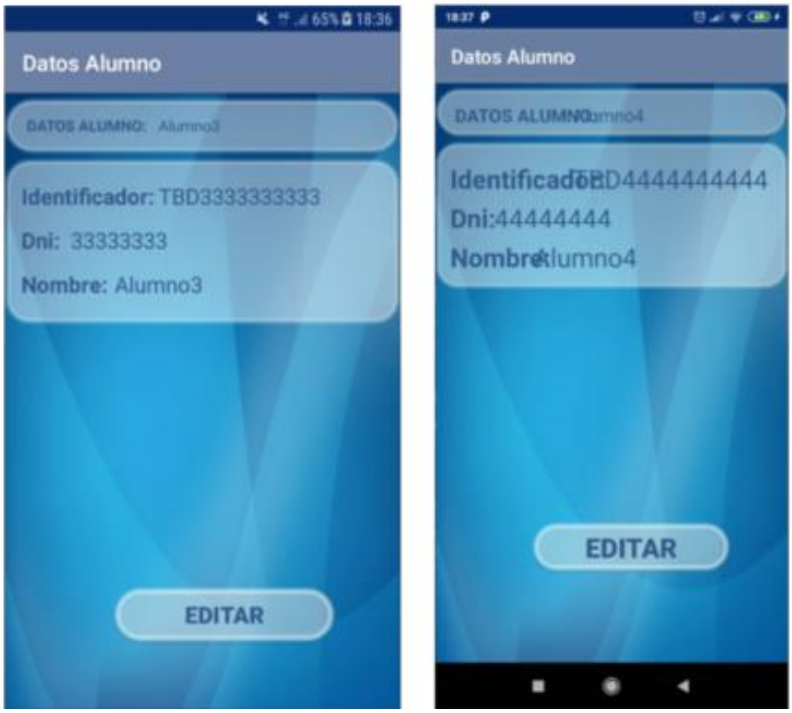

Figura 11. Comparativa de la misma pantalla de la aplicación en dos dispositivos distintos.

Además de que algunos campos se sobreponen, existen otros elementos que se muestran cortados o sin espacio entre el borde de la pantalla y estos. Estos problemas se repiten en bastantes pantallas de la aplicación, por lo que la IU será un punto importante en este proyecto.

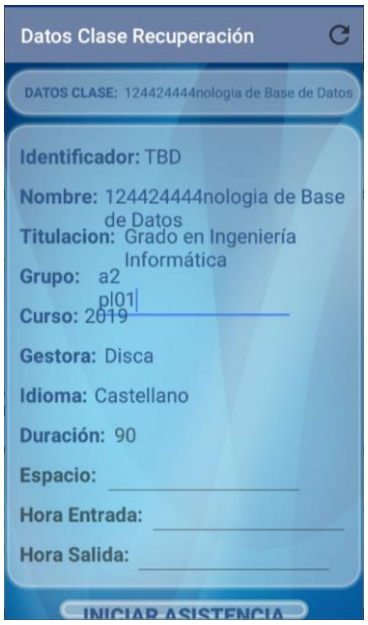

Figura 12. Pantalla en la que se sobreponen las palabras y el botón no se muestra en su totalidad.

En cuanto al **Diseño estándar (UX-B1)**, desde de la versión 5.0 de Android (API 21) se cuenta con una guía para el diseño visual de las aplicaciones. Se trata de Material Design[18]. Durante el desarrollo de esta aplicación no se tuvo en cuenta esta guía ya que, por ejemplo, no se aplican patrones UX comunes de Material como el botón de acción flotante o la navegación en la barra de app.

Figura 13. Pantalla de identificación de profesor en un teléfono Xiaomi REDMI S2.

La aplicación actualmente tiene un fondo no liso (contrariamente a las recomendaciones de Material) y cinco diferentes tonos del color azul. Esto también choca con Material Design ya que esta guía recomienda utilizar colores que destaquen los elementos más importantes, que

[18]Material Desing es un sistema adaptable de directrices, componentes y herramientas que apoyan las mejores prácticas de diseño de interfaces de usuario: https://material.io/

favorezcan la legibilidad y que, además, sean colores expresivos. Además, en esta pantalla observamos que los botones tienen el mismo aspecto que otros elementos de la interfaz.

Por todos estos motivos, realizaremos una renovación completa de la IU de la aplicación. Para ello seguiremos la guía de **Diseño y calidad** y, más en concreto, la guía de **Material Design**.

3.5. *Bugs* encontrados

En las primeras exploraciones se han detectado varios *bugs* en la aplicación. Por ejemplo, cuando iniciamos por primera vez la aplicación y seleccionamos la segunda opción del menú, nos aparece un diálogo que nos ofrece la posibilidad de cargar nuestras asignaturas desde un archivo XML que se encuentra en la carpeta de descargas. No obstante, aunque seleccionemos esta opción, no se cargan las asignaturas y el resultado es un elemento *null*.

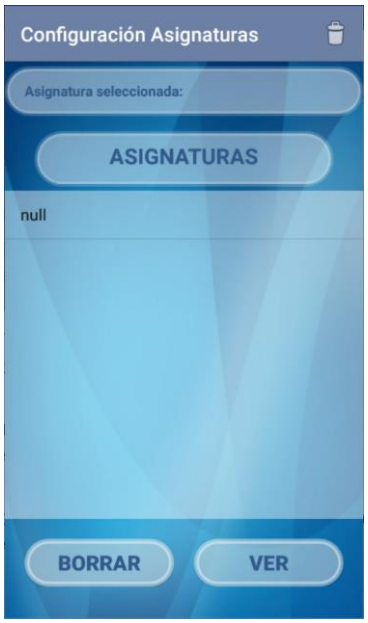

Figura 14. Listado de asignaturas después de haberse producido un error durante su carga.

En el **Anexo B** encontramos un listado con todos los *bugs* que se han encontrado en esta aplicación, junto con un identificador, una descripción y los pasos necesarios para reproducir el error.

3.6. Análisis del código fuente

Un buen diseño de *software* es importante para el buen funcionamiento a corto de plazo de una aplicación, pero resulta fundamental a medio y largo plazo.

El código incorrecto dificulta el desarrollo y el mantenimiento del *software*. El código incorrecto es, por lo tanto, un obstáculo para el desarrollador que debe continuar con el proyecto. En muchas ocasiones, se sacrifica la calidad del código a cambio de rapidez. El problema de sacrificar la calidad es que a largo plazo la productividad acaba siendo mucho menor de lo que podría ser (o directamente termina siendo nula) [9].

El código de esta aplicación presenta numerosos defectos. El más destacado es que (a pesar de lo que dice la documentación) carece de un patrón arquitectónico ya que, por ejemplo, desde cualquier clase se realizan consultas directamente a la base de datos; esto supone que cualquier mínima modificación en la base de datos pueda desembocar en cambios en múltiples clases.

Además, el código presenta *bad smells* [19], código duplicado, clases con demasiada responsabilidad o atributos que sólo se utilizan bajo circunstancias muy específicas [10]. También podemos encontrar abundantes métodos y variables que no se utilizan.

```
private int cont;
private int longui= 0;
private String[] listIdentificador = new String[1000];
private String[] listDni = new String[1000];
private String[] listNombre  = new String[1000];
private String[] listaIdentificadoresNoRepetidos  = new String[1000];
private Spinner spinner2;
private CountDownTimer timer;
private int secondsUntilFinished = 10000;
private TextView textTimer;
DataBase dataBase;

private long toReversedDec(byte[] bytes) {
    long result = 0;
    long factor = 1;
    for (int i = bytes.length - 1; i >= 0; --i) {
        long value = bytes[i] & 0xffl;
        result += value * factor;
        factor *= 256l;
    }
    return result;
}
```

Fragmento de código 1. En la clase RegistroAlumnos encontramos hasta once atributos que no se utilizan nunca.

[19] Mal olor. Son indicadores de deficiencia en el diseño software, lo cual favorece la aparición de errores y deficiencias.

4. Análisis general

En este capítulo realizaremos un análisis más general del problema, es decir, no nos centraremos en la aplicación, sino que nos centraremos en qué soluciones podemos aportar al problema del control de asistencia.

4.1. Modelo de dominio

El modelo de dominio (representado como un diagrama de clases UML[20] con los atributos más relevantes de cada entidad) nos permite reconocer y nombrar conceptos importantes y cómo se relacionan entre ellos. Este diagrama nos ayuda a familiarizarnos con el contexto del sistema con el que trabajaremos. Asimismo, nos facilitará identificar posibles mejoras que podremos implementar posteriormente en la aplicación.

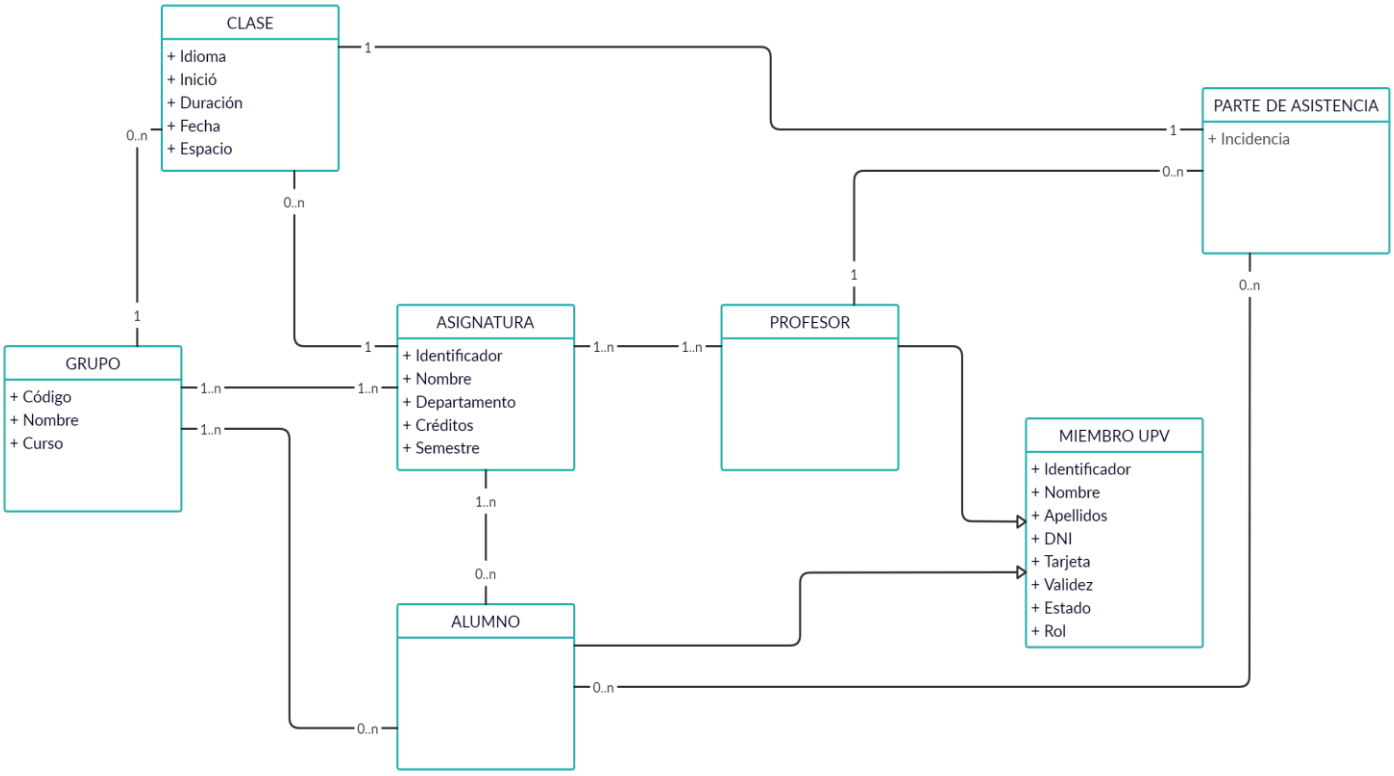

Figura 15. Modelo de dominio. Diagrama de clase UML.

[20] Lenguaje de modelado que es utilizado para representar esquemas o procesos de *software*.

4.2. Modelo de análisis

Puesto que no estamos desarrollando una aplicación de cero, el modelo de análisis resultante no ha sido fruto solamente de nuestro modelo de dominio. El modelo de análisis es el resultado de nuestro modelo de dominio adaptado al modelo de dominio creado por el anterior desarrollador y al código fuente de la aplicación heredada.

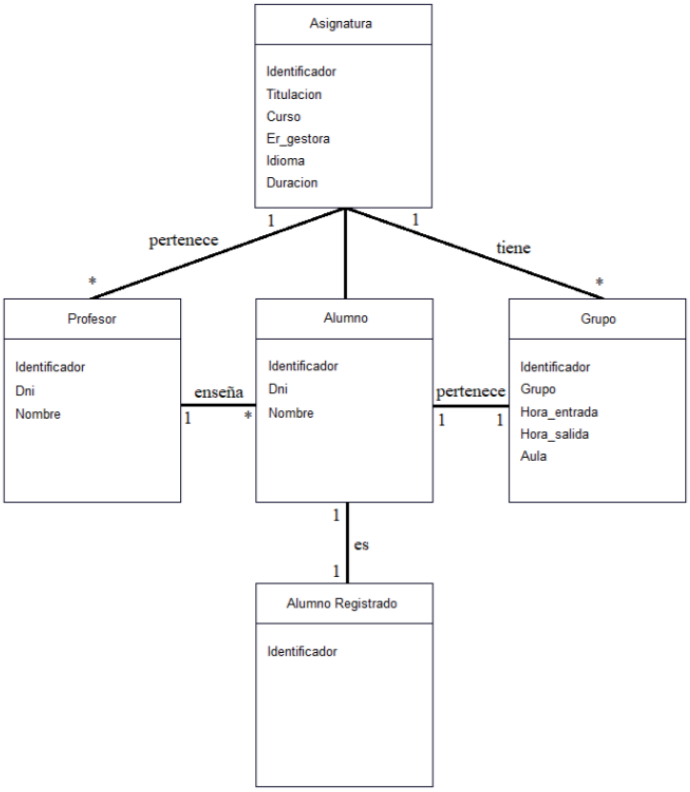

Figura 16. Modelo de dominio del sistema. Fuente: Documentación del proyecto.

Sería recomendable realizar una gran refactorización para adaptar el código a nuestro modelo de dominio ya que este se ajusta más a la realidad del problema, pero, al no tratarse del objetivo de este trabajo, simplemente nos adaptaremos a lo existente.

El modelo de análisis resultante tiene algunas diferencias significativas con respecto al modelo de dominio del primer proyecto. La entidad Alumno Registrado se ha eliminado porque en la nueva versión no será necesario almacenar esta información en la base de datos. En cambio, se ha añadido la entidad Partes de Asistencia, la cual será una de las más importantes del proyecto. Finalmente, las entidades Alumno y Profesor ahora son una especialización de la entidad Miembro de la UPV.

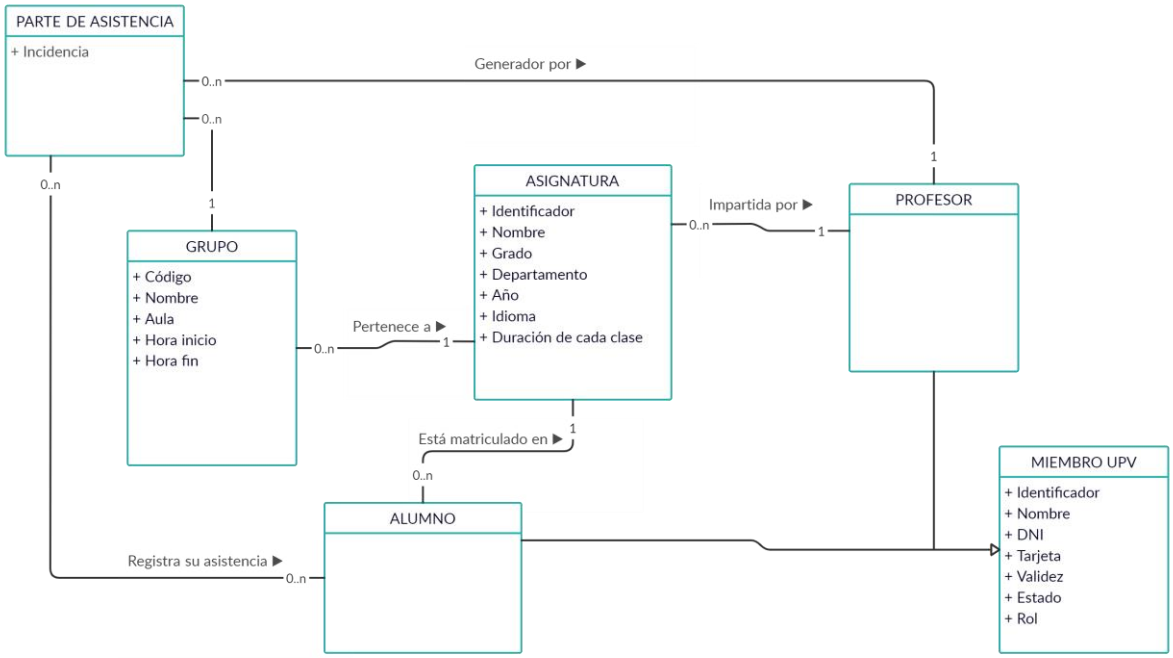

Figura 17. Modelo de análisis.

A diferencia del modelo de dominio, que no se centra en los aspectos de diseño (solamente en el contexto), el modelo de análisis sí que está dirigido al diseño de la aplicación.

4.3. Requisitos del sistema

4.3.1. Requisitos funcionales

Los requisitos funcionales (RF) están relacionados con las funcionalidades del sistema, es decir, con el "qué hace".

RF1. Un miembro de la UPV (profesor o alumno) podrá vincular su TUI a su teléfono inteligente.

RF2. Los profesores podrán generar partes utilizando su huella dactilar.

RF3. Al finalizar el proceso de creación de un parte se tendrá que generar un PDF con estos datos.

RF4. Los profesores podrán listar todos los partes que han generado.

RF5. Los profesores podrán consultar toda la información de partes antiguos.

RF6. Los profesores podrán generar archivos PDF con los datos de partes antiguos.

RF7. Los profesores podrán compartir partes antiguos.

RF8. Se mostrarán los datos personales del miembro de la UPV registrado en la app.

RF8. Los profesores podrán eliminar partes de firmas.

RF10. El sistema soportará operaciones CRUD para asignaturas.

RF11. El sistema soportará operaciones CRUD para grupos.

RF12. El sistema soportará operaciones CRUD para alumnos.

RF13. Los alumnos podrán fichar con su TUI o con su teléfono inteligente.

RF14. Un profesor podrá añadir incidencias/observaciones cuando esté generando un parte.

4.3.2. Requisitos no funcionales

Los requisitos no funcionales (RNF) están relacionados con las restricciones del sistema, es decir, con el "cómo lo hace".

La mayoría de los requisitos no funcionales serán tomados de la guía de **Diseño y calidad** de Android. Los requisitos de esta guía son fácilmente verificables puesto que cada requisito tiene asociado un conjunto de pruebas para asegurar que se cumplen estas condiciones.

- **Diseño visual e interacción con el usuario**

RNF1. La app cumple con las pautas de diseño de Android y utiliza patrones e íconos de IU comunes.

RNF2. Todos los diálogos se pueden descartar con el botón Atrás.

RNF3. La app admite la navegación estándar del sistema con el botón Atrás y no utiliza avisos personalizados en pantalla para el "botón Atrás".

RN4. Al presionar el botón de inicio en cualquier momento, se navega a la pantalla principal del dispositivo.

- **Estabilidad**

RNF5. La app no falla, no impone el cierre, no se inmoviliza ni funciona de ningún otro modo anormal en ninguno de los dispositivos donde esté instalada.

- **Rendimiento**

RNF6. La app se carga rápidamente o le proporciona al usuario información en pantalla (un indicador de progreso o una señal similar) en caso de que demore más de dos segundos en cargarse.

- **Compatibilidad**

RNF7. Calidad Visual: la app muestra texto y bloques de texto de forma aceptable (no se visualizan letras ni palabras cortadas y hay espacio suficiente entre el texto y los elementos que lo rodean).

RNF8. La app muestra gráficos, texto, imágenes y otros elementos de la IU sin distorsión, esfumado ni pixelado notables.

- **Seguridad**

RNF9. Todos los datos privados se guardan en el almacenamiento interno de la app.

RNF10. Todas las bibliotecas, SDK y dependencias están actualizadas.

- **Autenticidad**

RNF11. Un profesor tendrá que autenticarse para iniciar un parte y para finalizarlo.

RNF12. Un alumno tendrá que autenticarse para poder registrar su asistencia desde su teléfono inteligente.

RNF13. Para vincular la TUI al dispositivo el propietario del dispositivo tiene que autenticarse.

- **Usabilidad**

RNF14. La aplicación contará con una vista para profesores y otra diferente para alumnos.

RNF15. La app incorporará funciones de voz para agilizar diferentes procesos, como la adición de observaciones a un parte o la navegación entre pantallas.

RNF16. Para seleccionar un elemento de un listado bastará con hacer clic sobre ese elemento.

RNF17. Para eliminar un elemento de un listado bastará con hacer un clic largo sobre ese elemento.

4.4. Diagrama de casos de uso

El siguiente diagrama presenta los casos de uso de la aplicación. Este ha sido modelado con UML con el fin documentar y especificar los requisitos del sistema. El diagrama y los requisitos de los cuales partimos originalmente han sufrido varias refinaciones hasta llegar al diagrama actual.

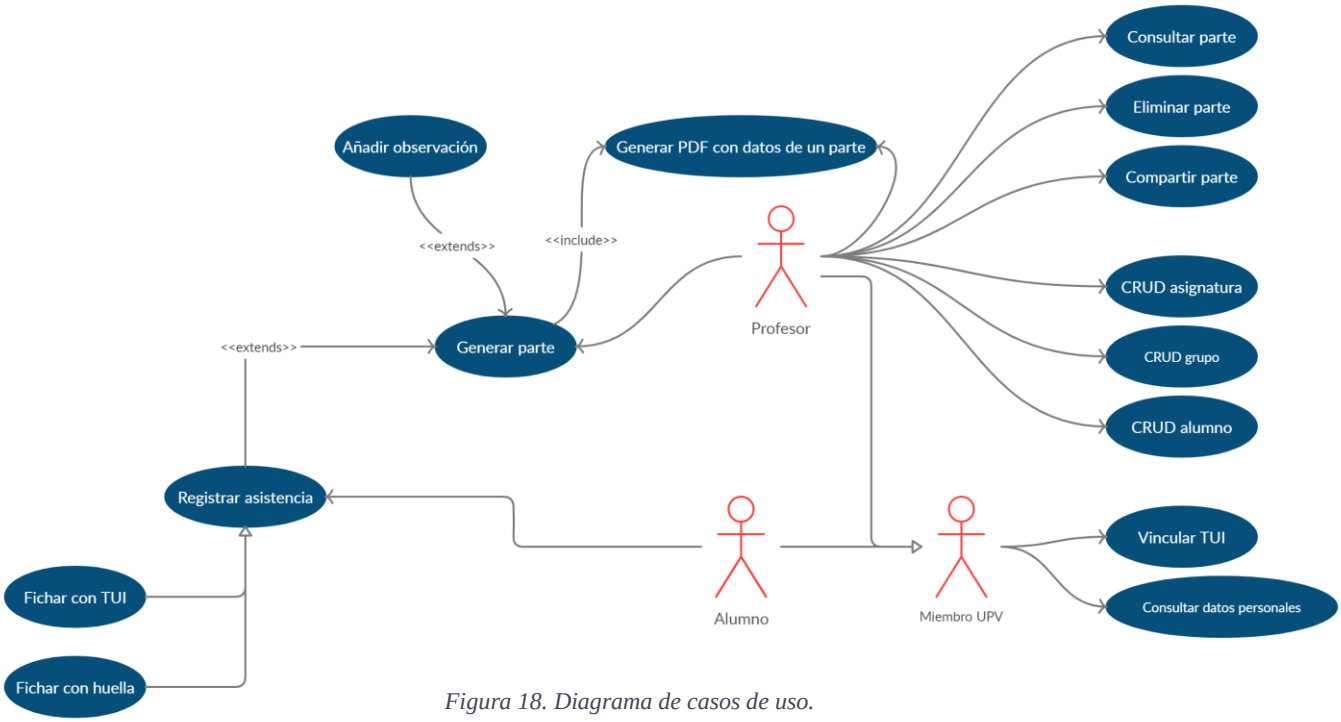

Figura 18. Diagrama de casos de uso.

5. Diseño de la solución

5.1. Diseño de la base de datos

El primer diseño de la base de datos presenta algunas deficiencias. La base de datos, pese a ser relacional, no tiene ni una sola tabla que tenga una clave foránea que haga referencia a otra tabla. Por ejemplo, un registro en la tabla Alumno utiliza su identificador (CP) para hacer referencia a otro registro de la tabla Asignatura: "Identificador de la asignatura" + "Identificador del alumno". Esto supone, entre otros muchos inconvenientes, que un alumno que esté en varias asignaturas ocupe varios registros en la misma tabla.

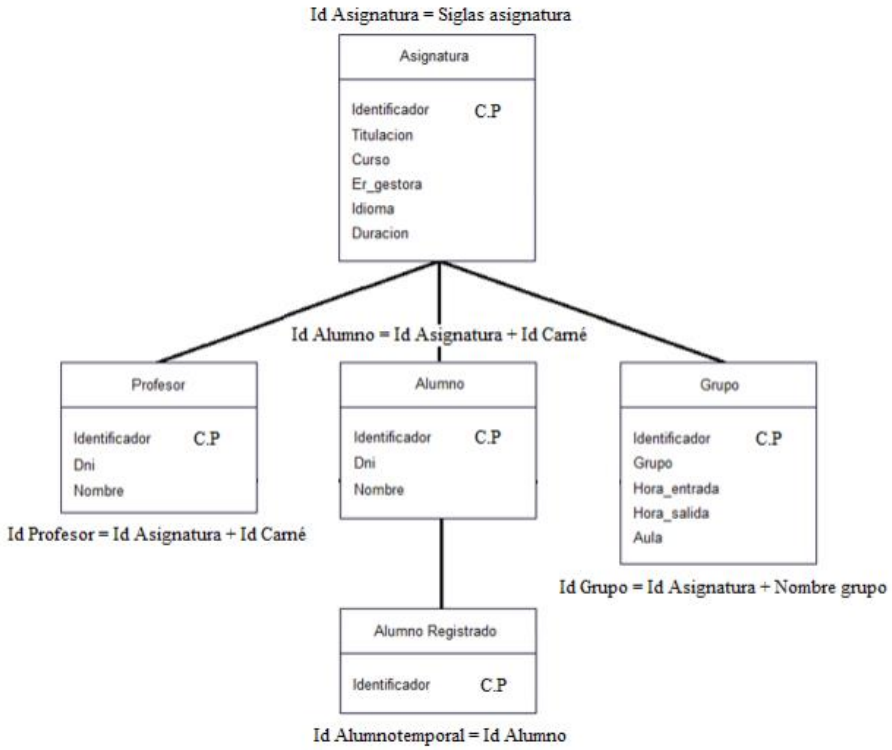

Figura 19. Primer diseño de la base de datos. Fuente: Documentación del proyecto.

La ausencia de claves foráneas supone tener mucha información duplicada, supone también un esfuerzo extra a la hora de referenciar a otros registros y, además, dificulta realizar consultas en la base de datos.

Después de nombrar sólo algunos de los problemas de no usar claves foráneas, podemos concluir que lo ideal sería diseñar una nueva base de datos, pero esto supondría un gran esfuerzo e invertir una gran cantidad de tiempo, del cual no disponemos. Asimismo, como el objetivo principal de este proyecto no es mantener este *software*, sino añadir nuevas funcionalidades, nos adaptaremos a los recursos existentes y diseñaremos la base de datos intentando surfear los inconvenientes que nos podamos encontrar.

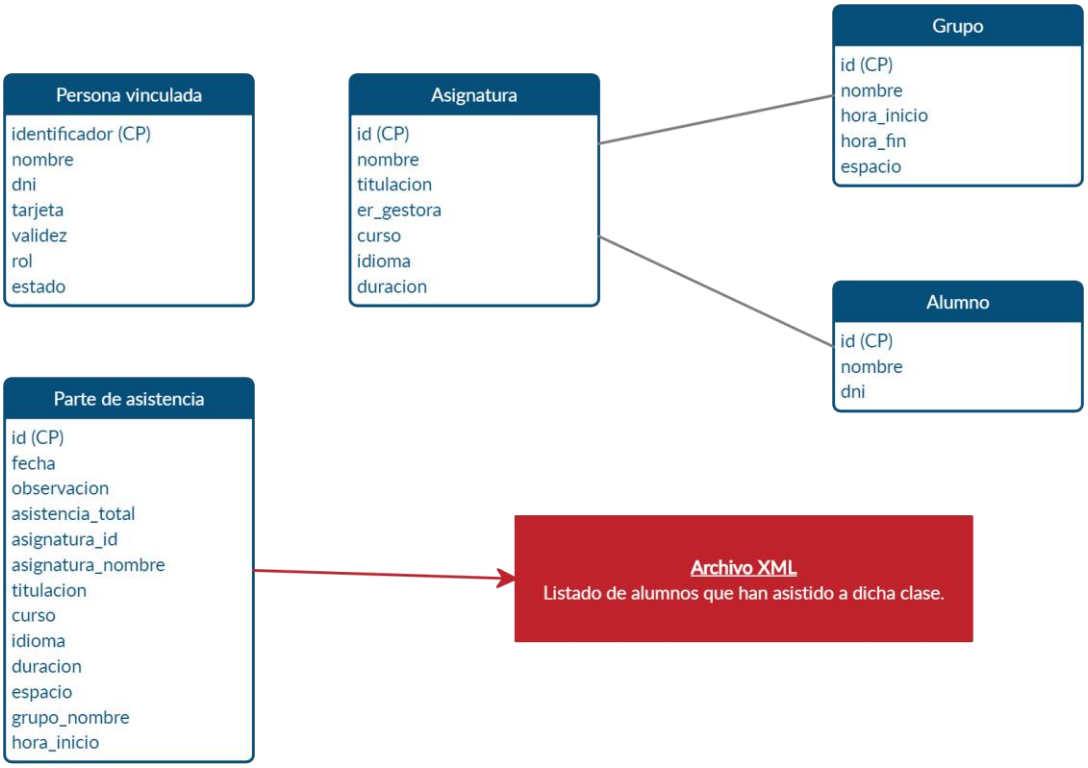

Figura 20. Diseño de la base de datos.

- **Persona vinculada**

Se trata del miembro de la UPV que ha vinculado su TUI al teléfono. En esta tabla se guardan sus datos. Como ahora el profesor puede vincular su tarjeta ya no es necesario que exista la tabla Profesor, por este motivo se ha prescindido de ella.

  - **identificador** (CP). Identificador del miembro de la UPV, este se obtiene cuando se acerca la TUI al teléfono.
  - **nombre**. Nombre y apellidos del miembro.
  - **dni**. DNI del miembro.
  - **tarjeta**. Número de la TUI.
  - **validez**. Fecha hasta la cual es válida la TUI.
  - **rol**. Si un miembro es profesor o alumno.
  - **estado**. Estado de la TUI (Activo, caducado…).

- **Asignatura**

Esta tabla contiene los datos de las diferentes asignaturas, no ha sufrido cambios. Una asignatura puede tener cero o muchos alumnos y cero o muchos grupos.

  - **id** (CP). Identificador de la asignatura, son las siglas de la asignatura.
  - **nombre**. Nombre de la asignatura.
  - **titulación**. Identificador de la titulación a la cual pertenece.
  - **curso**. Año en el que se está impartiendo la asignatura.
  - **idioma**. Idioma en el que se imparte la asignatura.
  - **duración**. Duración de cada clase en minutos.

- **Grupo**

Esta tabla contiene los grupos que imparte el profesor.

  - **id** (CP). El identificador del grupo sigue el formato: “Identificador de la asignatura”-“Nombre del grupo”. Por ejemplo, ISW-GA1.
  - **nombre**. Nombre del grupo.
  - **hora_inicio**. Hora a la que empieza la clase de ese grupo.
  - **hora_fin**. Hora a la que termina la clase de ese grupo.
  - **espacio**. Lugar, normalmente aula, en el que se imparte la clase.

- **Alumno**

Esta tabla contiene los datos de los alumnos que pertenecen a una asignatura.

  - **id** (CP). Identificador del alumno. Sigue el formato “Identificador de la asignatura” + “Identificador de la TUI del alumno”. Por ejemplo, ISW-123456.
  - **nombre**. Nombre y apellidos del alumno.
  - **dni**. DNI del alumno.

- **Parte de asistencia**

Los partes de asistencia se guardan en esta tabla. Esta tabla contiene toda la información de un parte, salvo el listado de alumnos. El listado de alumnos se guarda en un fichero XML.

  - **id** (CP). Identificador del parte.
  - **fecha**. Fecha en la que se genera el parte.
  - **observacion**. Comentarios o incidencias añadidas por el profesor.
  - **asistencia_total**. Número de alumno que han asistido a la clase.
  - **asignatura_codigo**. Siglas de la asignatura.
  - **asignatura_nombre**. Nombre de la asignatura.
  - **titulacion**. Identificador de la titulación de la asignatura.
  - **curso**. Año en el que se está impartiendo la asignatura.
  - **idioma**. Idioma en el que se ha impartido la clase.
  - **duracion**. Duración de la clase.
  - **espacio**. Espacio en el que se ha impartido la clase.
  - **grupo_nombre**. Nombre del grupo.
  - **hora_inicio**. Hora a la que ha empezado la clase.

Esta tabla no hace referencia a la asignatura, ni al grupo ni a los alumnos debido a que, si alguno de estos registros se borrara, no se podría recuperar el parte con todo su contenido correctamente. Esta no es la mejor práctica, pero, debido al estado actual de la base de datos, es la mejor solución.

- **Archivo XML: Listado de asistentes**

Por cada parte de asistencia se generará también un fichero XML con los datos de todos los alumnos que han asistido a dicha clase. Insistimos una vez más, lo ideal sería que estos datos se almacenaran en la base de datos, pero, debido al diseño, esta es la solución que nos ahorrará más tiempo.

El nombre de este archivo será el identificador del parte de asistencia. De esta forma, para rescatar el listado de alumnos sólo bastará con saber el identificador del parte y, así, sólo abriremos el fichero cuando seleccionemos el parte en cuestión.

5.2. Arquitectura del sistema

En sistema tendrá que integrar diferentes tecnologías con el fin de lograr las funcionalidades deseadas. Algunas tecnologías ya formaban parte de este, tal y como hemos comentados en puntos anteriores. En la siguiente figura podemos ver cómo se acoplarán las diferentes tecnologías y cómo interactuarán los distintos elementos entre ellos.

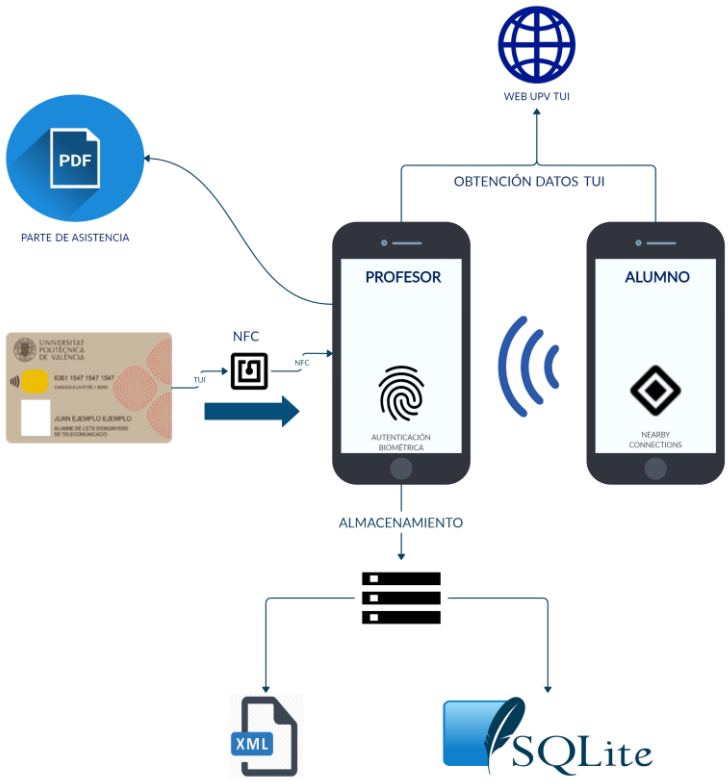

Figura 21. Arquitectura del sistema.

Algunas partes del sistema sufrirán modificaciones, como la parte de almacenamiento de datos. No sólo se usará SQLite, sino que también haremos uso de los ficheros XML para almacenar información.

5.3. Tecnologías elegidas

5.3.1. SQLite y XML

XML se utilizaba para almacenar la información básica de los partes de firmas, ahora usaremos este recurso para almacenar el listado de los alumnos que han asistido a una clase en concreto. Los datos básicos de los partes de asistencia se almacenarán en la base de datos mediante SQLite.

5.3.2. Nearby Connections API

Nearby Connections es una API de red P2P que permite a las aplicaciones descubrir, conectar e intercambiar datos con dispositivos cercanos en tiempo real, independientemente de la conectividad de la red. Esta API utiliza una combinación de puntos de acceso Bluetooth, BLE y Wifi, aprovechando las fortalezas de cada uno y sorteando sus respectivas debilidades[21].

Esta tecnología es ideal para que los alumnos puedan fichar con sus teléfonos inteligentes ya que tiene un alcance máximo de 30 metros, si no hay obstáculos de por medio, por lo que el alcance máximo real es bastante menor[22].

5.3.3. Otras APIs utilizadas

- **Android Speech-to-Text (RecognizerIntent)**. Android soporta la API de voz a texto incorporada de Google. Esta API nos permitirá incorporar a la aplicación las funciones de voz.
- **Android Biometric API**. Android ofrece autenticación biométrica, tanto huella dactilar como de cara.

5.3.4. Slidr

Slidr es una librería que nos permite añadir la funcionalidad *swipe-to-dismiss* para facilitar la navegación entre pantallas. Es decir, deslizando el dedo de derecha a izquierda volveremos a la pantalla anterior. Este gesto está incorporado en las aplicaciones de iOS y en algunas aplicaciones Android, como Telegram.

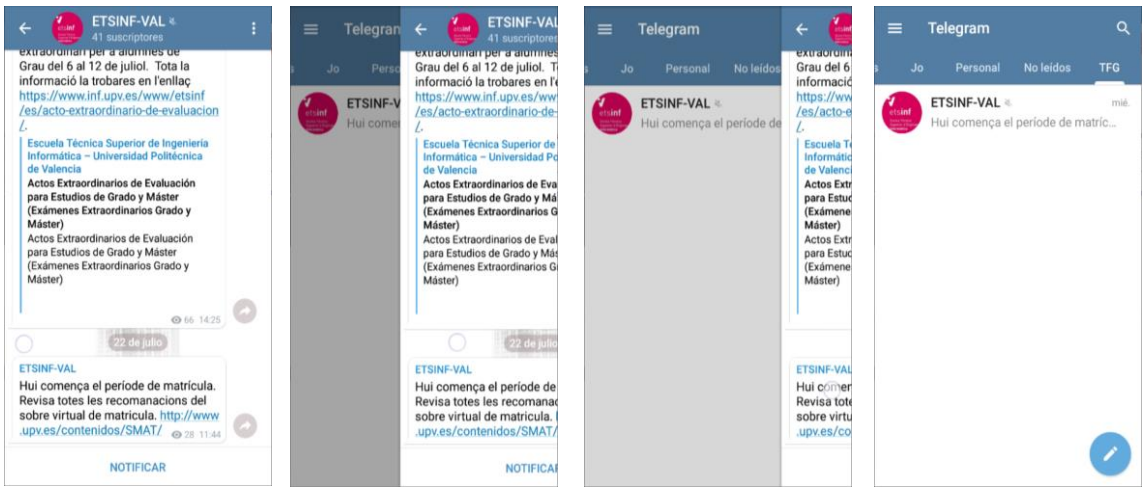

Figura 22. Funcionalidad swipe-to-dismiss en Telegram.

[21] Para saber más de Nearby: https://developers.google.com/nearby

[22] En esta conferencia sobre Nearby podemos encontrar estos detalles junto con otra información que puede resultar interesante: https://www.youtube.com/watch?v=Acdu2ZdBaZE&t=423s

5.4. Herramientas

5.4.1. Android Studio

Android Studio es el IDE oficial para el desarrollo de aplicaciones para Android, basado en IntelliJ IDEA. Además de ser una herramienta muy potente, tiene una interfaz amigable para el desarrollador.

En cuanto a Android, el nivel mínimo de API requerido para que la aplicación se ejecute es el 24. Este nivel se corresponde con la versión 7.0 de Android, también conocida como Android Nougat.

Figura 23. Logo de Android Nougat.
Fuente: Curso de Android (Máster en Desarrollo de Aplicaciones Android UPV)

5.4.2. Kotlin

Kotlin es un lenguaje de programación. Este surgió en 2011 y desde entonces se ha desarrollado continuamente. Ahora se encuentra integrado perfectamente en Android Studio por lo que hacer uso de este lenguaje no supone ninguna dificultad. Estas son algunas de las ventajas de Kotlin:

- **Expresivo y conciso**. Permite a los desarrolladores ser más expresivos y a la vez escribir menos código.
- **Seguro**. Ofrece herramientas para evitar error de tipo *NullPointerException*.
- **Familiar**. Es un lenguaje de programación de sencillo aprendizaje, especialmente para desarrolladores Java.
- **Interoperable**. Kotlin puede convivir sin ningún tipo de problema con Java en un mismo proyecto [11].

5.4.3. GitHub

El código del proyecto lo alojaremos en GitHub. GitHub es un portal web que permite el almacenamiento de recursos y que utiliza el sistema de control de versiones Git. De esta forma, si necesitamos ir a versiones anteriores del proyecto, deshacer cambios o crear diferentes ramas para cada funcionalidad lo podremos hacer con suma facilidad.

5.4.4. Proto.io

Proto.io[23] es una web que nos permite crear prototipos de alta fidelidad e interactivos. Nos ofrece una prueba gratuita de 15 días, que es más que suficiente para crear los diseños para este proyecto. Además de ser una herramienta con una interfaz bastante simple (hecho que ayuda si quien está diseñando no es un especialista en este apartado), tiene una lista de plantillas para facilitar la tarea de diseño.

5.5. Prototipado

El uso de prototipos puede ayudar tanto al cliente, para darle una idea de como será la aplicación, como al programador a la hora de implementar las diferentes pantallas. Elaborar primero una maqueta de cómo será nuestra aplicación es conveniente por diversos motivos:

- Las maquetas pueden ayudar a revelar cualquier elemento visual conflictivo mientras es sencilla su modificación.
- Empezar a codificar una vez elaborado el prototipo nos facilita la tarea de escribir un código sencillo, legible y limpio [12].

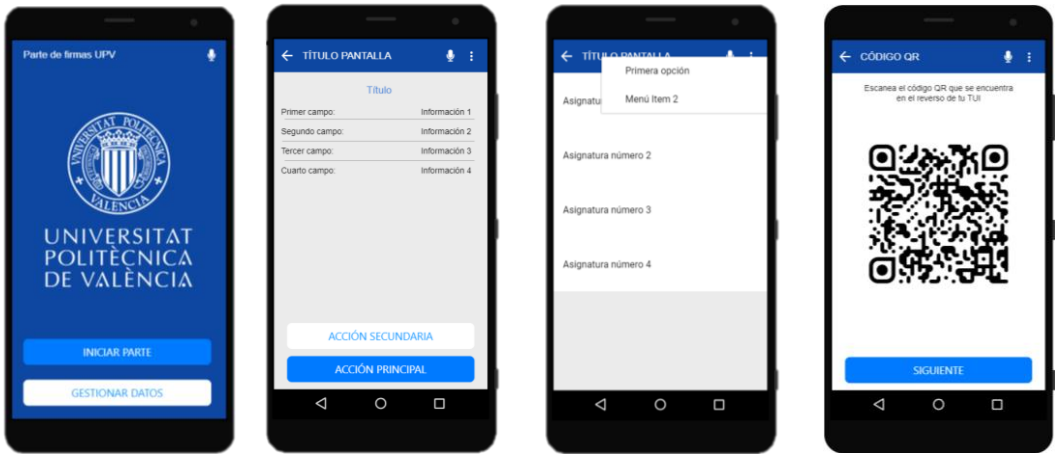

Figura 24. Prototipado de las pantallas de la aplicación.

Se han prototipado cuatro tipos distintos de pantallas. Esto es debido a que la aplicación por el momento no dispone de una gran cantidad de funcionalidades diferentes, por lo que muchas pantallas serán similares entre ellas.

Las pantallas que se han prototipado son:

- La pantalla principal de la aplicación.
- La pantalla que muestra información de un elemento. En la parte de arriba se muestra la información bien estructurada, mientras que en la parte inferior están los botones con los que gestionar dicha información.
- La pantalla con listados. Varias pantallas serán listados de asignaturas, alumnos, grupos o partes de asistencia.

[23] Para más información sobre esta herramienta: https://proto.io/

- La pantalla que muestra detalles de cómo incorporar datos a la aplicación. Por ejemplo, de cómo escanear el código QR de la TUI.

Estas pantallas tienen en común la barra superior (*toolbar*). Esta barra incorporará un micrófono para activar las funciones de voz, un menú con diferentes opciones y, salvo la pantalla principal, una flecha para poder navegar a la pantalla anterior (por razones obvias la pantalla principal no incorporará esta flecha).

5.6. Patrón arquitectónico

Como hemos comentado durante el análisis de la solución existente, la aplicación carece de patrón arquitectónico (a pesar de que en la documentación indica que sí). Esto significa que cualquier componente del sistema es el encargado de cualquier tarea.

Los patrones arquitectónicos son importantes porque asignan determinadas responsabilidades a cada componente y determinan cómo se comunican los componentes entre ellos. Sin un patrón cualquier cambio que se desea realizar en el código puede resultar fatal.

Figura 25. Esquema de una arquitectura MVC.

El patrón elegido para este proyecto es el MVC (Modelo – Vista – Controlador):

- **Modelo**. Es la capa de datos, responsable de conectar con la base de datos y, en este caso, con los ficheros XML. Representa también la lógica del negocio.
- **Vista**. Es la interfaz, lo que se muestra por pantalla.
- **Controlador**. Gestiona las peticiones de la vista. Actúa de intermediario entre el modelo y la vista.

Hemos elegido este patrón debido a que es el más sencillo de implantar teniendo en cuenta la solución actual. Implantar otro patrón supondría realizar una gran refactorización, lo cual nos modificar una gran cantidad de código y, por tanto, nos llevaría muchas horas.

6. Implementación

En este capítulo trataremos la implementación de las diferentes funcionalidades y mejoras que hemos incorporado a la aplicación. Los puntos de este capítulo estarán ordenados por orden de realización.

La corrección de *bugs* y las refactorizaciones han estado presentes en todas estas fases de programación. Si, por ejemplo, mientras se mejoraba la interfaz de un componente se podía corregir un *bug* de manera sencilla, se corregía (en el **Anexo C** podemos encontrar cómo se han solucionado todos los *bugs*).

Sin embargo, en este capítulo existen puntos dedicados a las refactorizaciones y a la solución de *bugs* porque en algunos momentos sí que hemos focalizado nuestro esfuerzo en estas actividades.

6.1. Renovación de la IU

El primer paso es renovar la IU, de esta forma, cuando desarrollemos pantallas nuevas ya tendremos una interfaz estable y uniforme.

La primera pantalla modificada ha sido la pantalla principal de la app. En el **Anexo D** encontraremos una comparativa en la que podremos observar el antes y el después de las principales pantallas de la aplicación.

Figura 26. Comparativa. El antes y el después de la pantalla principal de la aplicación.

A continuación, se ha renovado la parte de gestión de datos de la app. En esta parte, los principales cambios han sido:

- **Los listados**. Ahora un elemento de un listado se selecciona con un solo clic, mientras que, para eliminar un elemento, basta con un clic largo sobre el elemento. De esta forma mejoramos la usabilidad de la aplicación.

- **Estructuración de los datos**. Los datos se muestran de manera estructurada y sin sobreponerse entre ellos.
- **Botones flotantes**. Cuando ha sido posible se han añadido *Floating Action Buttons*. Estos son botones que se sobreponen a cualquier elemento y que se encuentran en la parte inferior derecha. De esta forma hemos reducido las responsabilidades del menú de la barra superior.

Por último, se han renovados las pantallas que conforman la funcionalidad de generación de partes de asistencia. Hemos aprovechado esta renovación para añadir más pantallas a este proceso. De esta forma, cada pantalla contendrá menos información y, consecuentemente, el usuario podrá entender mejor las indicaciones ofrecidas por la aplicación. Además, se ha implementado la pantalla para añadir observaciones al parte de asistencia y su funcionalidad.

Las pantallas de consultas de partes de asistencia ya generados no se han modificado todavía. Estas pantallas se modificarán cuando se implementen las diferentes funcionalidades relacionadas con estos partes.

Los principales cambios que han llevado a cabo en toda la aplicación son los siguientes:

- **Navegación hacia atrás**. En la barra superior se ha añadido una flecha que permite navegar a la pantalla anterior.
- **Animaciones en las transiciones**. También se han mejorado las animaciones cuando se navega entre pantallas. La aplicación ahora tiene dos animaciones, una para cuando se avanza a la siguiente pantalla y otra para cuando se vuelve a la anterior.
- **Colores más vivos y fondo liso**. La aplicación tiene un color azul más vivo y un fondo liso de color blanco, el cual mejora la legibilidad de los datos para el usuario.
- **Funcionalidad más clara**. Los botones de la pantalla principal ahora no son logos, ahora contienen también texto que permite identificar qué acción se va a realizar si se pulsan.

6.2. Vinculación de TUI al teléfono

En segundo lugar, hemos implementado la funcionalidad en la cual un miembro de la UPV puede vincular su TUI a su teléfono inteligente.

Este proceso consta de cuatro pantallas, en la primera pantalla se informa al usuario sobre los requisitos que debe cumplir su dispositivo para poder completar la vinculación de forma satisfactoria.

En las dos siguientes se recopila información del usuario a través de la tecnología NFC y del código QR. Finalmente, en la última pantalla, se muestran los datos del usuario para que este confirme la vinculación de estos datos al dispositivo.

De esta manera, ya no es necesario vincular el profesor a cada asignatura individualmente como se hacía hasta ahora. A partir de ahora, se considerará que todas las asignaturas almacenadas en el teléfono son impartidas por el usuario del dispositivo.

6.3. Autenticación biométrica

En tercer lugar, nos hemos centrado en desarrollar el objetivo principal del proyecto. La autenticación biométrica se solicita en tres puntos de la aplicación:

- Para confirmar la vinculación de la TUI al dispositivo (huella dactilar).
- Para abrir un parte de asistencia (reconocimiento facial).
- Para finalizar un parte de asistencia (huella dactilar).

Se ha establecido una alternativa a la autenticación biométrica. Así, el usuario podrá seguir completando el proceso aun habiendo algún problema. Además, de esta forma, teniendo en cuenta que existe una cantidad destacada de teléfonos que no disponen de tecnología de autenticación biométrica, estaríamos aumentando el número de potenciales usuarios de nuestra aplicación. Esta alternativa consiste en usar las credenciales de pantalla de bloqueo del teléfono (PIN, patrón o contraseña).

Como la autenticación estará presente en distintas partes de la aplicación, para no tener código duplicado, hemos creado una clase para gestionar este proceso. Se trata de la clase ***Biometry***. Esta clase (desarrollada en Kotlin) tiene tres atributos:

- **context** (del tipo *Context*[24]). Esta es una clase abstracta cuya implementación es proporcionada por el sistema Android.
- **title**. Es el título que tendrá la ventana de autenticación.
- **subtitle**. Es el subtítulo que tendrá la ventana de autenticación.

Figura 27. Ventana de aviso biométrico.

La función (en Java, método) que nos permite gestionar este proceso es ***authenticate (facial: Boolean = false, method: () -> Unit)***. Al invocar esta función el sistema nos muestra una ventana de aviso biométrico (o nos solicita introducir las credenciales de bloqueo de pantalla si el sistema no dispone de esta tecnología). Esta función cuenta con dos parámetros:

- ***facial***. Este parámetro indica el tipo de autenticación que se solicitará: reconocimiento facial o huella dactilar.
- ***method***. Es la función o líneas de código que se ejecutarán cuando la autenticación se haya realizado correctamente.

[24] Interfaz para la información global sobre un entorno de aplicación. Permite el acceso a clases y recursos específicos de la aplicación, así como a llamadas para operaciones a nivel de la aplicación como actividades de lanzamiento, emisión y recepción de intentos, etc.

```
fun authenticate(facial: Boolean = false, method: () -> Unit) {

    val biometricPrompt = BiometricPrompt(
            context as FragmentActivity, // fragmentActivity
            ContextCompat.getMainExecutor(context), // executor
            callback(method) // authenticationCallback
    )

    val promptInfo = BiometricPrompt.PromptInfo.Builder()
            .setTitle(title)
            .setSubtitle(subtitle)
            .setConfirmationRequired(!facial)
            .setDeviceCredentialAllowed(true)
            .build()

    biometricPrompt.authenticate(promptInfo)
}
```

Fragmento de código 2. Función encargada de gestionar el proceso de autenticación. Pertenece a la clase Biometry.

Internamente, la función ***authenticate*** está dividida en tres partes. En la primera parte se declara e inicializa la variable inmutable ***biometricPrompt*** del tipo ***BiometricPrompt***. Esta variable recoge lo que será la lógica de la autenticación. En la creación de esta variable se pasan tres parámetros:

- ***fragmentActivity***. Es una referencia a la actividad[25] que invoca la autenticación.
- ***executor***. Se encarga de gestionar los eventos.
- ***authenticationCallback***. Es un objeto que recibe los eventos de autenticación. La función ***callback*** es la encargada de crear este objeto.

A continuación, se define la variable ***promptInfo***. Esta variable recoge los elementos que se mostrarán en el diálogo de autenticación biométrica y el tipo de autenticación requerida (de huella dactilar o de reconocimiento facial).

- Los métodos ***setTitle*** y ***setSubtitle*** permiten establecer el título y el subtítulo del diálogo.
- Para elegir el tipo de autenticación hacemos uso de ***setConfirmationRequired***. Cuando este método recibe el valor ***true*** se solicita la huella dactilar, mientras que cuando es ***false*** se solicita la autenticación mediante reconocimiento facial.
- La sentencia ***setDeviceCredentialAllowed(true)*** es la que permite que el usuario pueda autenticarse, de manera alternativa, a través de las credenciales de la pantalla de bloqueo.

Finalmente, para mostrar el diálogo de autenticación biométrica en pantalla, se invoca el método ***authenticate*** de la clase ***BiometricPrompt***: ***biometricPrompt.authenticate(promptInfo)***.

[25] En Android, una actividad o *activity* es una entidad de la aplicación que se usa para gestionar la IU. Cada actividad se asocia con una ventana en la cual se define la IU con la que interactuará el usuario.

```
private fun callback(method: () -> Unit) : AuthenticationCallback {
    return object : BiometricPrompt.AuthenticationCallback() {

        override fun onAuthenticationError(
                errorCode: Int, errString: CharSequence
        ) {
            super.onAuthenticationError(errorCode, errString)
            toast("Error de autenticación: $errString")
        }

        override fun onAuthenticationSucceeded(
                result: BiometricPrompt.AuthenticationResult
        ) {
            super.onAuthenticationSucceeded(result)
            method()
        }

        override fun onAuthenticationFailed() {
            super.onAuthenticationFailed()
            toast("Autenticación fallida")
        }
    }
}
```

Fragmento de código 3. Método de la clase Biometry que recibe los eventos de autenticación.

La función ***callback*** recibe el parámetro ***method*** y devuelve un objeto preparado para recibir los eventos de autenticación. Esta función recoge tres supuestos:

- En caso de error, o en caso de que el usuario cancele el proceso, la aplicación mostrará al usuario en pantalla un mensaje de error y el motivo de este.
- Si la autenticación resulta fallida también se informará al usuario a través de un mensaje en pantalla.
- **Autenticación exitosa**. En este caso se ejecutará el método, o líneas de código, que esta función habrá recibido previamente como parámetro.

De esta forma, cuando el usuario realice una acción sobre el diálogo de autenticación, la aplicación podrá reaccionar adecuadamente a los diferentes supuestos.

```
Biometry biometry = new Biometry(this, "Autenticación", "Autenticación
requerida para iniciar el parte");

buttonGeneratePdf.setOnClickListener {
    biometry.authenticate  { endReport() }
}
```

Fragmento de código 4. Código escrito en Kotlin de una actividad que hace uso de la autenticación biométrica.

Por tanto, gracias a la clase ***Biometry*** podemos hacer uso del sistema de autenticación en una actividad de manera sencilla y limpia ya que con pocas líneas de código obtenemos la funcionalidad deseada.

6.4. Vista para alumnos

Para que los alumnos puedan fichar a través de su teléfono inteligente estos deben de poder usar la aplicación. Por este motivo, se ha creado una vista especial para ellos. Por el momento, puesto que la aplicación está orientada, sobre todo, a profesores, la vista de los alumnos tiene las funcionalidades justas para que estos puedan fichar. La vista para alumnos cuenta con tres pantallas:

- **Pantalla principal**. Con dos opciones, navegar a la pantalla de fichaje y navegar a la pantalla de información.
- **Pantalla de información**. La aplicación muestra los datos personales del alumno.
- **Pantalla de fichaje**. La aplicación busca partes cercanos y los muestra al alumno. Para conseguir esta funcionalidad necesitamos implementar Nearby Connections.

6.5. Implementando Nearby Connections

Para la comunicación alumno – profesor haremos uso de la API Nearby Connections. De esta forma un profesor podrá abrir un parte para que el alumno pueda fichar desde su teléfono. Las dos clases encargadas de gestionar este proceso son ***Advertise*** (profesor) y ***Disvover*** (alumno)[26].

Tabla 2. Ciclo de vida de una conexión entre un alumno y un profesor.

	ADVERTISE	**DISCOVER**
1	Inicia el anunciante	Inicia el descubridor
2	---	Encuentra un anunciante
3	---	Envía una petición de conexión al anunciante
4	Acepta la petición de conexión del descubridor	---
5	**Intercambio de información entre anunciante y descubridor**	
6	Finaliza el anunciante	Finaliza el descubridor

La actividad en la que los alumnos podían fichar acercando su TUI, a partir de ahora, tendrá también otra funcionalidad. Esta actividad tiene ahora un atributo de tipo ***Advertise***. De la misma forma, la actividad de la vista para alumnos tiene un atributo de tipo ***Discover***. Estas dos clases tienen en común los métodos ***start***, ***stop***, ***sendPayload*** y ***addObserver***.

El método ***start*** es el encargado de iniciar el anunciante o descubridor, según el caso. El método ***stop*** se encarga de detenerlo. El método ***addObserver*** se encarga de notificar cambios al usuario. Al profesor le notificará que el parte se ha abierto correctamente, mientras que al alumno le

[26] La documentación de Nearby Connections API la podemos encontrar en línea en https://developers.google.com/nearby/connections/overview. En ella está explicado el proceso de conexión entre un anunciante y un descubridor, entre otros muchos detalles.

notificará que ha descubierto un nuevo parte de asistencias. Este último método forma parte del patrón *Observer* (observador). Este patrón, como ya hemos podido ver, se utiliza para detectar eventos en tiempo de ejecución.

Por último, el método ***sendPayload*** se usa para enviar información. Desde el teléfono del alumno se enviará una dupla de valores hacia al profesor. Esta dupla consta del identificador del alumno y del identificador del teléfono (para evitar que desde un mismo teléfono se fiche más de una vez).

El teléfono del profesor recibirá esta información y la procesará. A continuación, le contestará al alumno informándole sobre la resolución de su fichaje (si se ha registrado su asistencia correctamente, si no está dado de alta en la asignatura o si ya había fichado).

En el **Anexo F** podemos encontrar una explicación más detallada del funcionamiento de Nearby en nuestra aplicación.

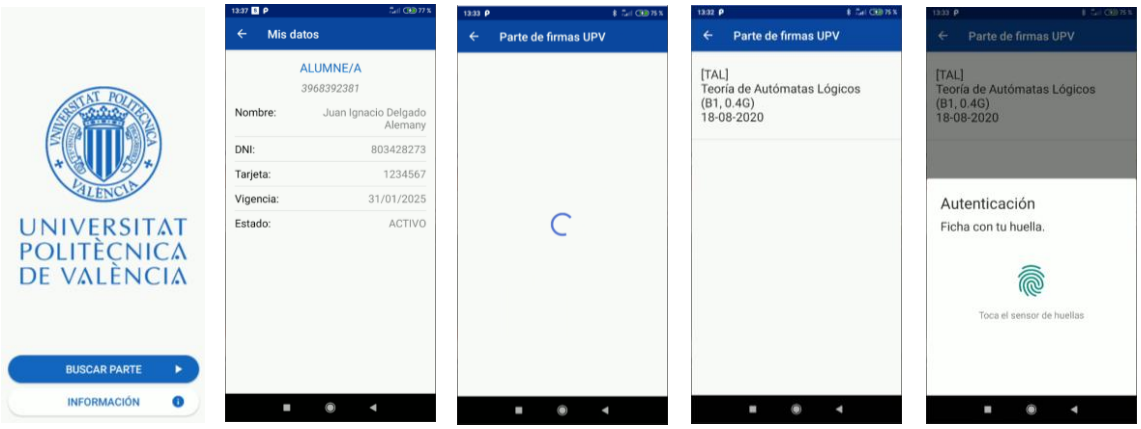

Figura 28. Diferentes pantallas de la vista para alumnos.

6.6. Gestión de los partes de asistencia

A continuación, se ha renovado por completo la parte de la aplicación sobre la cual aún no habíamos efectuado ningún cambio.

Primero hemos modificado el proceso de generación de partes de asistencia para que, cuando se vaya a crear el parte, sus datos se almacenen en la base de datos y no en un fichero XML en la carpeta de descargas del teléfono. Seguidamente, se han hechos cambios para que el listado de alumnos que han asistido a la clase no se pierda y también se pueda recuperar posteriormente. Este fichero tiene como nombre el identificador del parte y se almacena en la carpeta de la aplicación.

Después, se ha renovado la interfaz de la pantalla de consultas de partes. Antes, en esta pantalla, se mostraban tres campos que había que rellenar para encontrar un parte en concreto. Ahora se muestra un listado con todos los partes de asistencia que se han generado hasta el momento. Además, los partes se pueden filtrar por fecha, por nombre o código de la asignatura y por grupo.

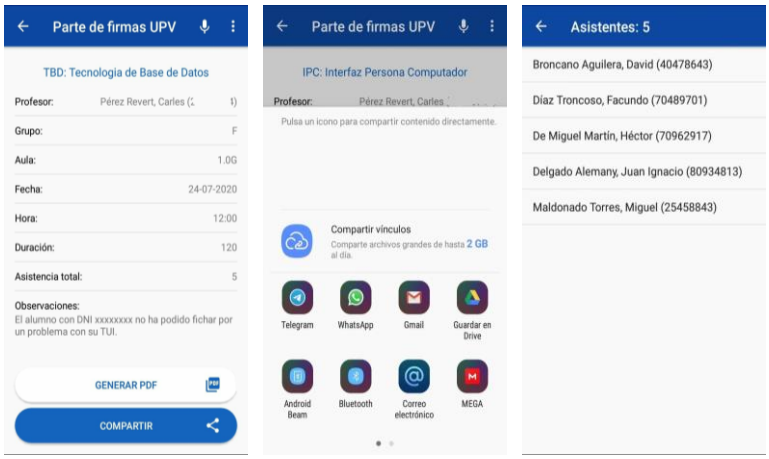

Figura 29. Pantallas que contienen información de un parte de asistencia.

Cuando seleccionamos un parte del listado accedemos a sus detalles. En esta pantalla ahora sí que podemos consultar todos sus datos. Gracias a disponer de toda su información podemos volver a generar un PDF, así como compartir el parte de asistencia a través de distintas aplicaciones.

Debido a que ahora podemos compartir un parte previamente generado, se ha suprimido la opción de enviar un parte por correo que anteriormente la aplicación ofrecía. Además, esta opción no funcionaba, cuando se intentaba ejecutar la aplicación se detenía de forma inesperada.

6.7. Funciones de voz

Las funciones de voz, en la actualidad, están orientadas principalmente a dos tareas: dar órdenes al sistema y generar texto a través del habla.

En nuestra aplicación, la generación de texto a través del habla se usa para añadir observaciones a los partes de asistencia. De esta forma, se puede añadir una observación de forma sencilla y rápida.

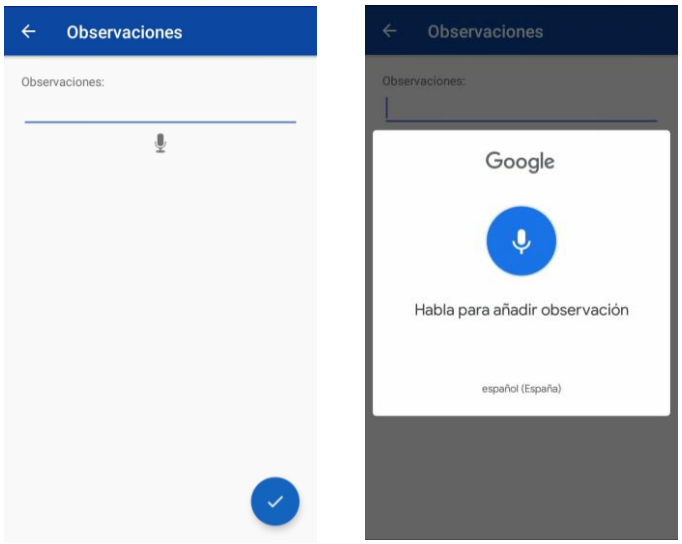

Figura 30. Funciones de voz en la pantalla de observaciones de un parte.

Para lanzar las funciones de voz es necesario llamar al método estático ***launchVoice*** de la clase ***CommandVoice***. Cuando un profesor pulse sobre el icono del micrófono de la pantalla de observaciones se ejecutarán las siguientes líneas de código.

```
speak.setOnClickListener {
    CommandVoice.launchVoice(this, "Habla para añadir una observación.")
}
```

Fragmento de código 5. Código implementado para lanzar las funciones de voz.

La aplicación también acepta órdenes por voz. Podemos usar las órdenes por voz para navegar rápidamente entre pantallas. En estos momentos, se puede navegar a cinco pantallas distintas usando la voz:

- **Principal**. Se puede navegar a la pantalla principal diciendo "Inicio" o "Pantalla principal".
- **Iniciar parte**. A través de la orden "Iniciar parte" se puede navegar a la primera pantalla del proceso de generación de partes (en la que seleccionamos la asignatura a impartir).
- **Consulta de partes**. Para ir al listado donde podemos encontrar todos los partes generados basta con decir "Consultar partes".
- **Datos del profesor**. Con la orden "Ver mis datos" navegamos a la pantalla donde se muestra la información personal del profesor.
- **Asignaturas del profesor**. Finalmente, con la orden "Ver mis asignaturas" la aplicación nos lleva al listado de asignaturas del profesor.

Para la navegación por voz, hemos creado una actividad que se encarga de gestionar estas órdenes. El texto generado por las funciones de voz se recoge en el método o función ***onActivityResult***.

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (
            requestCode == CommandVoice.COMMAND_VOICE_CODE
            && resultCode == RESULT_OK && null != data
    ) {
        val array = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS)
        val string = array[0]
        val cl = commandVoiceData.processData(string)

        if (cl != null) {
            val intent = Intent(this, cl)
            startActivity(intent)
        } else {
            toast("Lo siento, no te he entendido... \uD83D\uDE25")
        }
    }

    finish()
}
```

Fragmento de código 6. Función en la que se recoge el texto generado por la función de voz.

Con la sentencia `data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS)` obtenemos un vector, el cual contiene el texto generado en su primera posición. En la clase ***CommandVoiceData*** figuran las distintas acciones que puede realizar la aplicación; por este motivo hacemos uso de su método ***processData***. Este método devuelve la clase (actividad) a la que posteriormente navegará la aplicación. Sin embargo, si no se detecta ninguna coincidencia, la aplicación hará saber al usuario que no ha dado ninguna orden reconocible por el sistema.

La aplicación no sólo soporta los comandos por voz para navegar entre pantallas, sino que también permite la realización de acciones, como generar archivos PDF o compartir un parte de asistencia. Este tipo de órdenes están presentes en dos pantallas de la aplicación. El funcionamiento de estos comandos es similar al de las órdenes de navegación.

- **Pantalla de detalles de un parte de asistencia**. Esta es la pantalla en la que se muestran los datos de un parte de asistencia previamente generado. En ella se soportan estos tres comandos:
    - **"Generar PDF"**. Para generar un PDF con los datos del parte.
    - **"Compartir"**. Para compartir un parte de asistencia con otra aplicación.
    - **"Ver asistencia"**. Para ver el listado de alumnos que han asistido a clase.
- **En la última pantalla del proceso de generación de partes**. Esta es la pantalla en la que se muestran los detalles del parte que se está generando en ese momento y en la cual se pueden añadir observaciones al mismo.
    - **"Añadir observación"**. Para añadir observaciones al parte de asistencia.
    - **"Ver asistencia"**. Para ver el listado de alumnos que han asistido a clase.
    - **"Finalizar parte"**. Para cerrar el parte y generar un archivo PDF con los datos recopilados.

Hemos podido observar que al dar una orden oral a un teléfono inteligente es fácil que, por diferentes motivos, se produzca alguna leve modificación cuando esta se traduce a texto. Por ejemplo, si un profesor dice "Ver mis asignaturas" y el sistema lo entiende como "Ver asignaturas", esta no corresponderá literalmente con la orden preestablecida y, por lo tanto, no realizará la tarea. Para evitar que esto ocurra, hacemos uso del algoritmo de distancia de Jaro Winkler.

El algoritmo de distancia de Jaro Winkler compara dos cadenas de caracteres y devuelve un valor entre 0 y 1. El valor 1 significa que las cadenas son exactamente iguales, mientras que el valor 0 indica que no guardan ninguna similitud entre ellas [13]. Hemos establecido que la comparación entre la orden preestablecida y la entendida por el sistema deben tener un 75% (valor 0.75) de similitud, como mínimo, para que dicha orden se ejecute.

Como hemos comentado anteriormente, la clase ***CommandVoiceData*** contiene las distintas acciones que se pueden realizar y el método ***processData*** que devuelve la actividad (pantalla) a la que se navegará. Esta clase tiene dos atributos:

- ***MIN_MATCH***. Es un atributo estático e inmutable cuyo valor es 0.75. Este atributo establece el grado de similitud que deben tener dos cadenas.
- ***actions***. Este atributo (de tipo ***Map***) almacena tanto las órdenes que se pueden dar como la clase asociada de cada orden. ***Map<Key,Value>*** es una interfaz que nos permite representar una estructura de datos en la que sus elementos son pares clave-valor. Hemos implementado esta interfaz a través de la clase ***HashMap***.

```
public Class<?> processData(String data) {
    String key = data.replace(" ", "").toLowerCase();
    Class<?> cl = actions.get(key);

    if (cl == null) cl = getMoreSimilar(key);

    return cl;
}
```

Fragmento de código 7. Método processData de la clase CommandVoiceData.

En el método ***processData*** primero se realiza una búsqueda en la estructura de datos usando métodos propios de la clase ***HashMap***. El método ***get*** de ***HashMap*** busca un elemento de la estructura de datos que tenga una clave que coincida exactamente con la orden dada por el usuario. En caso de no encontrar ninguna coincidencia se recurre al algoritmo de distancia.

```
private Class<?> getMoreSimilar(String key) {
    Class<?> cl = null;
    double match = MIN_MATCH; // 0.75

    double res;

    for (Map.Entry<String, Class<?>> entry : actions.entrySet()) {
        res = compareDistance(key, entry.getKey());

        if (res > match) {
            match = res;
            cl = actions.get( entry.getKey() );
        }
    }

    return match > MIN_MATCH ? cl : null;
}


private double compareDistance(String s1, String s2) {
    JaroWinklerSimilarity jw = new JaroWinklerSimilarity();
    return jw.apply(s1, s2);
}
```

Fragmento de código 8. Métodos getMoreSImilar y compareDistance de la clase CommandVoice.

En el método ***getMoreSimilar*** se recorre la estructura del atributo ***actions*** comparando la orden dada por el usuario con cada una de las claves. Este método devolverá el valor del elemento, es decir, una clase, cuya clave tenga un mayor porcentaje de similitud con la orden. En cambio, si ningún resultado supera el mínimo establecido (0.75) el valor devuelto será ***null***.

6.8. Gestos en Android

La última funcionalidad que hemos añadido ayuda a mejorar la experiencia de usuario cuando usa nuestra aplicación. Esta funcionalidad consiste en volver a la pantalla anterior de una manera cómoda y rápida: deslizando el dedo de izquierda a derecha por la pantalla.

Este gesto está presente en aplicaciones como Telegram y en dispositivos cuyo sistema operativo es iOS. Esto significa, que este gesto, asociado a esta funcionalidad, cuenta con una gran aceptación por parte de los usuarios. Por esta misma razón se ha incorporado este gesto a la aplicación.

Para dotar a la aplicación de esta funcionalidad, se ha añadido en el método ***onCreate*** de cada actividad la siguiente sentencia: `Slidr.attach(this)`.

Sin embargo, esta no es la única sentencia que hemos añadido para implementar de manera satisfactoria esta funcionalidad. Una vez añadida la sentencia en las actividades correspondientes, la interfaz se mostraba sin fondo, es decir, el fondo era transparente. Para solucionar esto, en la vista de cada actividad hemos añadido la siguiente línea de código: `android:background="@color/background_material_light"`.

Por último, al estilo de la aplicación definido previamente en el archivo ***styles.xml*** le hemos añadido dos nuevos ítems. Estos dos ítems, ***Android:windowIsTranslucent*** y ***Android:windowBackground***, son los que permiten que se pueda ver la pantalla anterior debajo de la actual cuando el usuario desplaza el dedo de izquierda a derecha de la pantalla.

Figura 31. Funcionalidad swipe-to-dismiss en la que se ve la pantalla actual a la derecha y la anterior a la izquierda.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/background_material_light"
    tools:context=".presentation.teacher.management.ManagementActivity">
```

Fragmento de código 9. Fragmento de código de la vista de una actividad que implementa la funcionalidad swipe-to-dismiss.

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">

    <!-- ...Código omitido... -->

    <!-- Swipe back -->
    <item name="android:windowIsTranslucent">true</item>
    <item name="android:windowBackground">@android:color/transparent</item>
</style>
```

Fragmento de código 10. Código XML que permite definir el estilo de la aplicación.

6.9. Últimos retoques

Una vez desarrolladas todas las tareas planificadas nos hemos centrado en corregir los pocos *bugs* que se habían quedado sin solucionar. Los otros se han podido resolver perfectamente durante el desarrollo de las nuevas funcionalidades. Una gran parte de los *bugs* estaban ocasionados por sentencias SQL defectuosas, es decir, lo que realmente se solicitaba a la base de datos no era lo que realmente se requería.

Finalmente, después de corregir todos los *bugs*, hemos invertido tiempo en explorar el código fuente con el fin de seguir realizando refactorizaciones para corregir posibles defectos del código y con el fin de dejar un código más legible, mejor estructurado y que, consecuentemente, sea más fácil de mantener y de seguir desarrollando posteriormente. De la misma manera, atributos que se utilizaban bajo circunstancias muy específicas se han convertido en variables locales del método correspondiente.

Primero, hemos eliminado una gran cantidad de variables, atributos y métodos que no se usaban en ninguna parte. Anrdoid Studio tiene una herramienta que nos indica cuando un elemento no se utiliza y se puede eliminar sin problemas.

En segundo lugar, hemos eliminado las librerías que ya no nos eran útiles, estas son JavaMail y DroidText. También hemos actualizado las bibliotecas que sí que utilizamos en la aplicación.

En tercer lugar, nos hemos hecho cargo del código duplicado. Por ejemplo, la tecnología NFC se utiliza en varios puntos de la aplicación y en todos ellos se repetía el mismo código. De la misma forma, en varias actividades podíamos encontrar consultas SQL que, además, estaban duplicadas en varias clases. Para solucionar esto, creamos un método común y conseguimos eliminar el código duplicado. Así, en el futuro, para realizar una modificación simplemente será necesario cambiar el código en mismo lugar.

Como hemos comentado, había clases que realizaban consultas directamente en la base de datos. También existía una actividad con unas 40 líneas de código cuya función era generar un PDF. Es decir, existían clases con demasiada responsabilidad. En el caso de la generación de archivos PDF, se creó una clase encargada de generarlos, abrirlos y compartirlos. En cambio, en el caso de las consultas, se han añadido nuevos métodos en la capa de modelo para evitar que se acceda a la base de datos desde la capa intermedia.

Se han realizado también otras refactorizaciones, pero las principales son la que acabamos de nombrar. Algunas otras refactorizaciones han estado centradas en evitar métodos con muchas líneas de código y en renombrar métodos, variables y atributos.

7. Pruebas

Se han realizado diferentes tipos de pruebas para comprobar ciertos aspectos de la solución. Los aspectos en cuestión son: la renovada interfaz que hemos implementado a lo largo del proyecto y los requisitos no funcionales.

7.1. Validación de la nueva interfaz

Los cambios de interfaz en las aplicaciones web o móvil suelen generar una gran variedad de opiniones entre los usuarios, causando, en ocasiones, un fuerte rechazo. Para evaluar la nueva interfaz y garantizar que los cambios hechos se pueden considerar como mejoras hemos creado una encuesta a través de la herramienta Formularios de Google. Esta encuesta se ha publicado y difundido en distintas redes sociales por lo que el perfil del encuestado es el de una persona joven familiarizada con el uso de teléfonos inteligentes. Han participado un total de 64 personas.

En la descripción de la encuesta se ha informado al encuestado sobre qué hace la aplicación y cuál es el objetivo de su creación: permitir a los profesores de la UPV pasar lista en sus clases con su teléfono inteligente y generar partes de asistencia.

Al encuestado le hemos solicitado que compare las dos versiones de la aplicación. En cada pregunta hemos incidido en una característica y él ha elegido cuál de las dos versiones cree la representa mejor.

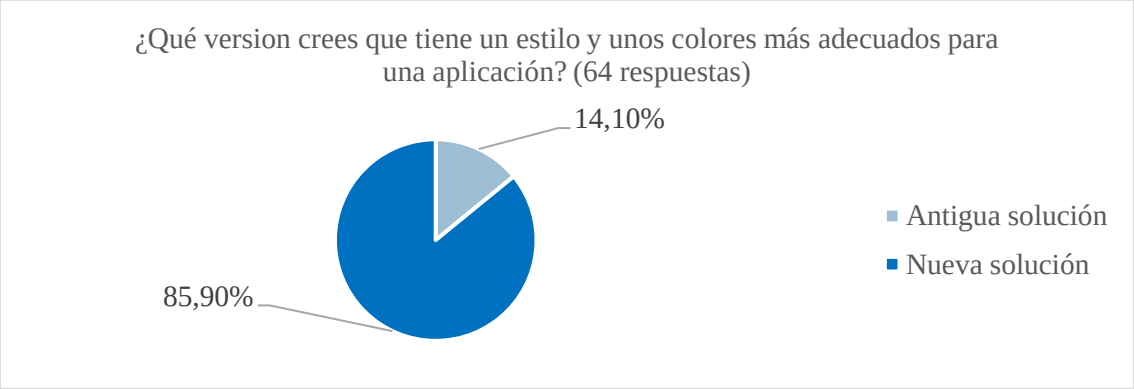

Figura 32. Resultado de la encuesta: estilo y colores de la IU.

La mayoría de la gente encuestada (85,90%) piensa que la nueva solución tiene elementos que han podido reconocer en otras aplicaciones. Creen, por lo tanto, que el estilo y los colores utilizados se adaptan mejor a los estándares de Android. De la misma manera, una gran cantidad de respuestas (79,80%) nos muestra que la actual versión describe de mejor manera para el usuario qué acciones puede realizar en cada momento.

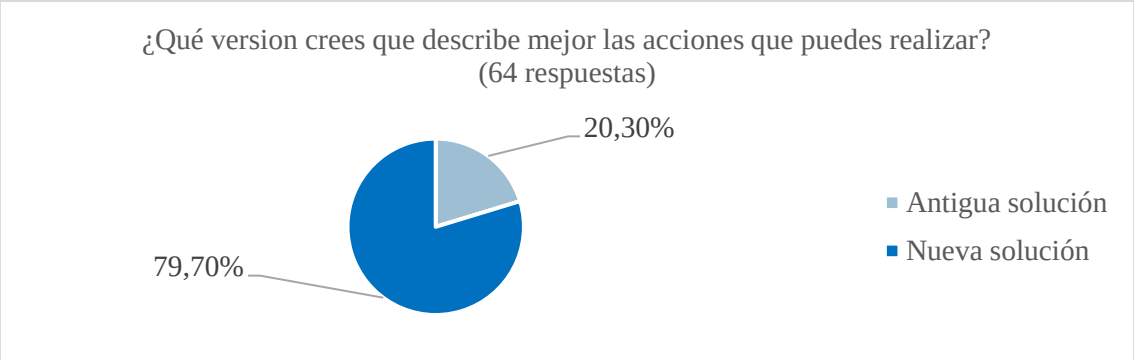

Figura 33. Resultado de la encuesta: IU intuitiva.

También nos interesaba saber si habíamos mejorado la forma en la que se muestra la información. Para ello, le hemos pedido al usuario que comparase una pantalla de edición de datos de la antigua solución con una pantalla de edición de datos de la nueva versión. Igualmente, le hemos pedido que haga lo mismo con pantallas de lectura de datos.

Los resultados han sido manifiestamente favorables puesto que un 100% de los encuestados cree que la pantalla de lectura de datos es mejor ahora, mientras que un 98,40% cree que la pantalla de edición ofrece una mejor visualización de la información. Además, un 84,40% de los votantes piensan que las nuevas pantallas tienen una mejor distribución de los elementos que la componen.

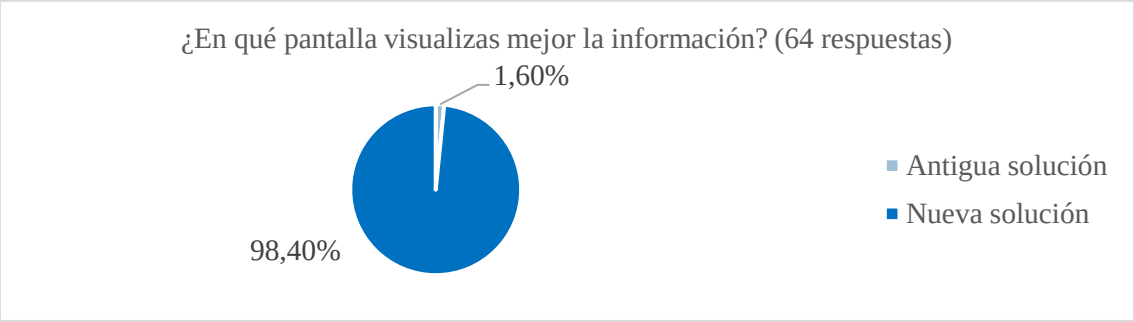

Figura 34. Resultado de la encuesta: visualización de los datos.

Finalmente, después de enseñar a los encuestados una variedad de capturas de pantalla de cada solución, se les ha preguntado cuál de las dos versiones les gusta más y el por qué. Nuevamente, la diferencia entre las dos opciones ha sido realmente significativa. De las 64 personas que han respondido, sólo 2 han votado que prefieren la anterior versión, mientras que 62 han elegido la nueva solución (96,90%). Los tres motivos más repetidos han sido “más limpia”, “más intuitiva” y “más sencilla”.

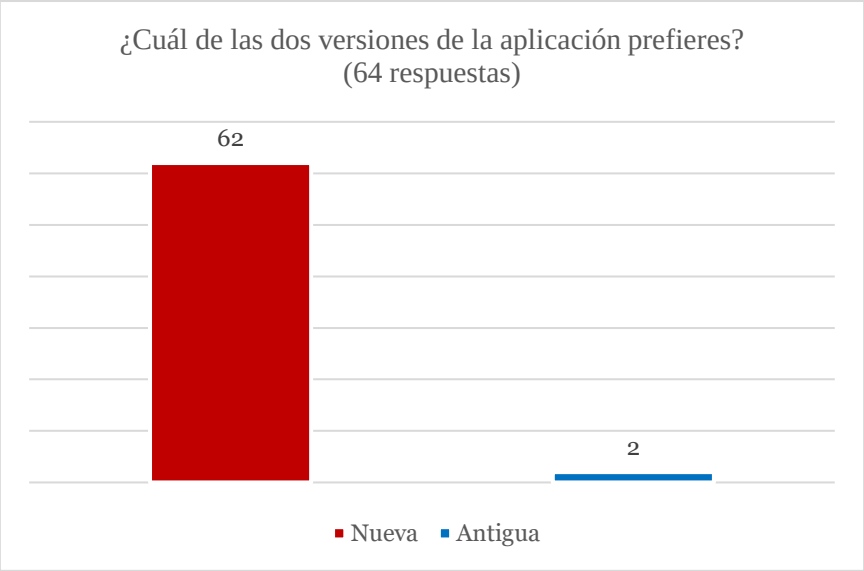

Figura 35. Resultado de la encuesta: preferencias de los encuestados.

La renovación de la interfaz ha sido un éxito rotundo ya que ha tenido una gran aceptación. En cada una de las preguntas la nueva versión se ha impuesto por una gran mayoría a la antigua. Por todo esto, definitivamente, podemos considerar que la renovación de la interfaz es una mejora que hemos aplicado a esta solución.

7.2. Verificación de los requisitos no funcionales

Para evaluar los aspectos de calidad de nuestra aplicación, es decir, para comprobar que nuestra aplicación cumple con los requisitos no funcionales, hemos seguidos las pruebas asociadas que tiene cada requisito de la guía de **Diseño y calidad** de Android.

RNF1 **La app cumple con las pautas de diseño de Android y utiliza patrones e íconos de IU comunes.**

Pruebas: Navega a todas las partes de la app, es decir, a todas las pantallas, los diálogos, las configuraciones y a todos los flujos de usuarios.

Desde cada una de las pantallas de la app, pasa a otra que se esté ejecutando y luego regresa a la app que estás probando con el conmutador de Apps recientes.

Presiona el botón de encendido para colocar el dispositivo en modo de suspensión y luego presiona nuevamente el botón de encendido para reactivar la pantalla.

Resultado: Satisfactorio ✔

RNF2 **Todos los diálogos se pueden descartar con el botón Atrás.**

Pruebas: Desde cada uno de los diálogos de la app, presiona el botón Atrás.

Resultado: Satisfactorio ✔

RNF3 **La app admite la navegación estándar del sistema con el botón Atrás y no utiliza avisos personalizados en pantalla para el "botón Atrás".**

Pruebas: Desde cada una de las pantallas de la app, presiona el botón Atrás.

Resultado No ha pasado la prueba ✕

Motivo: En la pantalla principal el botón Atrás no hace nada.

Solución: Cuando en la pantalla principal se pulsa el botón Atrás se sale de la app.

FINAL: Resultado satisfactorio ✔

RNF4 **Al presionar el botón de inicio en cualquier momento, se navega a la pantalla principal del dispositivo.**

Pruebas: Desde cada una de las pantallas de la app, presiona la tecla de Inicio del dispositivo y luego reinicia la app desde la pantalla Todas las apps.

Resultado: Satisfactorio ✔

RNF5 **La app no falla, no impone el cierre, no se inmoviliza ni funciona de ningún otro modo anormal en ninguno de los dispositivos donde esté instalada.**

Pruebas: Navega a todas las partes de la app, es decir, a todas las pantallas, los diálogos, las configuraciones y a todos los flujos de usuarios.

Resultado: Satisfactorio ✔

RNF6 **La app se carga rápidamente o le proporciona al usuario información en pantalla (un indicador de progreso o una señal similar) en caso de que demore más de dos segundos en cargarse.**

Pruebas: Navega a todas las partes de la app, es decir, a todas las pantallas, los diálogos, las configuraciones y a todos los flujos de usuarios, y comprueba que ningún proceso tarda más de dos segundos en cargarse o, en su lugar, se muestra algún aviso en pantalla.

Resultado: No ha pasado la prueba ✕

Motivo: Cuando se escanea el código QR de la TUI en alguna ocasión se tarda más de dos segundos en recuperar los datos (por motivos ajenos a la aplicación) del miembro de la UPV.

Solución: Se ha añadido una ventana que avisa que la información se está cargando.

FINAL: Resultado satisfactorio ✔

RNF7 **Calidad Visual: la app muestra texto y bloques de texto de forma aceptable (no se visualizan letras ni palabras cortadas y hay espacio suficiente entre el texto y los elementos que lo rodean).**

Pruebas: Navega a todas las partes de la app, es decir, a todas las pantallas, los diálogos, las configuraciones y a todos los flujos de usuarios, y comprueba que no se muestran elementos cortados ni sobrepuestos.

Resultado: Satisfactorio ✔

RNF8 **La app muestra gráficos, texto, imágenes y otros elementos de la IU sin distorsión, esfumado ni pixelado notables.**

Pruebas: Navega a todas las partes de la app, es decir, a todas las pantallas, los diálogos, las configuraciones y a todos los flujos de usuarios, y comprueba que los elementos se muestran nítidos.

Resultado: Satisfactorio ✔

RNF9 **Todos los datos privados se guardan en el almacenamiento interno de la app.**

Pruebas: Revisa todos los datos guardados en el almacenamiento externo.

Resultado: Satisfactorio ✔

RNF10 **Todas las bibliotecas, SDK y dependencias están actualizadas.**

Pruebas: Consulta el archivo *build.gradle (app)* con el IDE Android Studio y comprueba que este no muestra ningún aviso sobre que hay nuevas versiones disponibles ni similares.

Resultado: Satisfactorio ✔

RNF11 **Un profesor tendrá que autenticarse para iniciar un parte y para finalizarlo.**

Pruebas: Sigue el proceso de generación de un parte de asistencia y comprueba que el profesor no puede completar el proceso sin su autenticación.

Resultado: Satisfactorio ✔

RNF12 **Un alumno tendrá que autenticarse para poder registrar su asistencia desde su teléfono inteligente.**

Pruebas: Cuando haya un parte disponible intenta fichar y comprueba que sin la autenticación no es posible completar el proceso.

Resultado: Satisfactorio ✔

RNF13 **Para vincular la TUI al dispositivo el propietario del dispositivo tiene que autenticarse.**

Pruebas: Inicia el proceso de vinculación de TUI al teléfono y comprueba que el sistema solicita la autenticación al usuario del teléfono para completar este proceso.

Resultado: Satisfactorio ✔

RNF14 **La aplicación contará con una vista para profesores y otra diferente para alumnos.**

Pruebas: Inicia el proceso de vinculación de TUI al teléfono y comprueba que el sistema solicita la autenticación al usuario del teléfono para completar este proceso.

Resultado: Satisfactorio ✔

RNF15 **La app incorporará funciones de voz para agilizar diferentes procesos, como la adición de observaciones a un parte o la navegación entre pantallas.**

Pruebas: Navega por diferentes pantallas de la aplicación y comprueba que admite órdenes por voz.

Resultado: Satisfactorio ✔

RNF16 **Para seleccionar un elemento de un listado bastará con hacer clic sobre ese elemento.**

Pruebas: Navega a todas las listas de la aplicación y comprueba que haciendo clic sobre un elemento navegas a su pantalla de detalles (si la hay).

Resultado: Satisfactorio ✔

RNF17 **Para eliminar un elemento de un listado bastará con hacer un clic largo sobre ese elemento.**

Pruebas: Navega a todas las listas de la aplicación y comprueba que seleccionando un elemento con un clic largo la aplicación muestra un aviso de eliminación.

Resultado: Satisfactorio ✔

8. Conclusiones

Una vez finalizado el desarrollo de este proyecto nos vamos a centrar en comentar algunos aspectos de este. Hablaremos de la consecución de los objetivos iniciales de este proyecto, de los problemas surgidos y cómo se han intentado solucionar, de los conocimientos adquiridos y de la relación del proyecto con los estudios cursados.

El funcionamiento y los diferentes procesos que se ejecutan de la aplicación los podemos encontrar explicados en el **Anexo E**.

8.1. Consecución de los objetivos

El objetivo principal de este proyecto era el de proveer de mecanismos para la autenticación biométrica a una solución ya existente cuyo fin era el de dotar al profesorado de la UPV de un nuevo mecanismo para pasar lista en sus clases. No obstante, durante el análisis del proyecto surgieron varios objetivos nuevos. El hecho de partir de un proyecto ya propició que actividades relacionadas con el mantenimiento y la evolución del *software* formaran parte de los objetivos del proyecto.

Los objetivos iniciales han podido ser cumplidos ya que la aplicación cuenta con un sistema de autenticación, tanto biométrica como de credenciales de pantalla de bloqueo, así como se ha renovado totalmente la IU de forma satisfactoria.

La vista para alumnos y que estos pudiesen fichar con su teléfono inteligente fue un objetivo añadido, extra, podríamos decir. Debido a la gran utilización de los teléfonos inteligentes hoy en día y a la gran aceptación de los sistemas biométricos se valoró positivamente probar con esta nueva funcionalidad.

Además, hemos conseguido solucionar todos los *bugs* encontrados en la aplicación. Igualmente, hemos dado los primeros pasos para dotar a la arquitectura *software* de un patrón arquitectónico y que, de esta forma, el sistema sea más fácil de mantener y de seguir desarrollando (como hemos comentado en repetidas ocasiones).

8.2. Conocimientos adquiridos

Nunca había desarrollado una aplicación para Android. Por este motivo, antes de empezar el proyecto hice un curso de iniciación para conocer las características de Android y así tener una base aceptable de cara a empezar el desarrollo de esta aplicación[27].

Aprovechando que tenía que desarrollar una aplicación Android, me propuse aprender un nuevo lenguaje: Kotlin. De esta manera, no me estaría limitando simplemente a hacer uso de

[27] El curso en cuestión se puede encontrar en http://www.androidcurso.com/ y en https://courses.edx.org/courses/course-v1:UPValenciaX+AIP201x+2T2019/course/.

conocimientos previos, sino que debería esforzarme por aprender cosas nuevas que me pudieran servir en mi futuro profesional.

Una vez familiarizado con Android, Android Studio y Kotlin, desarrollé una simple y pequeña aplicación para almacenar listas de la compra[28]. Terminada esta app, ya estaba preparado para afrontar el proyecto.

8.3. Cómo se han afrontado los problemas

Los principales problemas que hemos encontrado en este proyecto han venido propiciados por el hecho de que teníamos que partir de una solución existente. El diseño original de la base de datos, los defectos en el código fuente, la ausencia de patrón arquitectónico, entre otros, han ocasionado problemas a la hora de desarrollar nuevas funcionalidades. Esto provocó que tuviésemos que invertir más tiempo del que podríamos haber dedicado a desarrollar el proyecto.

Otro problema ha estado relacionado con el *hardware*. Para probar la aplicación necesitábamos dispositivos que estuviesen equipados con sensor de huellas dactilar, lector NFC y *software* de reconocimiento facial. El emulador de Android Studio cuenta con el sensor de huellas, pero no con las otras dos características. También contábamos con un teléfono *Xiaomi REDMI S2*, pero este no disponía ni de reconocimiento facial ni de lector NFC. El otro dispositivo con el que hemos contado es un *Samsung Galaxy SM-J510FN 2016*, pero este no cuenta con ningún método de autenticación biométrica.

Para usar la aplicación en un teléfono sin lector NFC desarrollamos una especie de puente en el código fuente para que, de forma artificial o simulada, se produzca esta lectura (obviamente este código nunca ha estado en una versión final). En cambio, la única solución para probar el reconocimiento facial era encontrar algún dispositivo Android que dispusiese de esta tecnología. Por suerte, conseguimos probar el reconocimiento facial en un *Huawei Honor Play* que un conocido nos facilitó durante unas horas.

El último problema importante con el que nos hemos enfrontado ha sido el de poder probar la tecnología Nearby Connections. Esta tecnología permite que dos o más dispositivos, cercanos entre sí, interactúen para poder realizar el fichaje en clase. Este proyecto se ha desarrollado bajo un contexto de pandemia mundial, por lo que no hemos podido probar esta funcionalidad en un espacio de la UPV en el que se impartan clases. No obstante, hemos intentado emular estos espacios en una casa, es decir, hemos probado a conectar entre profesor y alumno a varias distancias (a seis metros, a dos o incluso a 15 metros). También hemos intentado representar un alumno que intenta conectarse desde fuera de clase, para hacer esto hemos colocado los dos teléfonos en habitaciones distintas. El resultado fue que no se podía fichar en habitaciones distintas, pero las paredes de una casa pueden tener una grosor distinta que las paredes de las aulas de la UPV, por lo que lo ideal sería probarlo en un aula.

[28]Puedes encontrar el código de esta aplicación en https://github.com/perezrevert04/ShoppingList.

8.4. Relación con los estudios cursados

En este proyecto hemos aplicado diferentes conocimientos adquiridos en el grado. Por ejemplo, Java es el lenguaje troncal del grado, el más utilizado a lo largo de la carrera. La aplicación estaba desarrollada en Java, por lo que, a pesar de haber programado principalmente en Kotlin, también hemos escrito muchas líneas en Java. También hemos escrito y modificado sentencias SQL. SQL en el grado se aprende en la asignatura Base de Datos y Sistemas de Información.

En varias asignaturas hemos tratado los casos de uso, utilizado UML para crear diagramas de clase así como hemos elaborado modelos de dominio, pero en la asignatura en la que más detalle se ha impartido ha sido Análisis y Especificación de Requisitos. En esta asignatura también hemos tratado los RF y los RNF de un sistema. Podríamos decir que estos conocimientos han jugado un papel importante en este proyecto en la fase de análisis.

Para el diseño de la IU nos ha ayudado la asignatura Interfaces Persona Computados, en la cual se adquieren nociones sobre cómo desarrollar interfaces que sean amigables e intuitivas para el usuario. En este punto también nos han sido de gran ayuda las prácticas de empresa. Durante estas prácticas pude desarrollar interfaces para teléfonos inteligentes, las cuales fueron prototipadas previamente por un diseñador gráfico. Por estos motivos contábamos con una buena base para poder renovar la IU de la solución.

La asignatura Diseño de *Software* nos ha servido a la hora de aplicar patrones de diseño (como el patrón *Observer*) y el patrón arquitectónico (MVC). En esta asignatura se incide en elaborar código de calidad mediante el uso de buenas prácticas. Para ello se habla de libro como Clean Code, el cual hemos aplicado en este proyecto.

Por último, al trabajar sobre un proyecto previo, hemos podido aplicar conocimientos adquiridos en la asignatura Mantenimiento y Evolución del *Software*.

9. Trabajos futuros

La solución actual es totalmente funcional, ahora mismo se podría utilizar para generar partes de asistencia en una clase normal. No obstante, la aplicación puede y debe continuar desarrollándose puesto que está en sus primeras fases de vida.

9.1. Posibles mejoras

Existen diferentes formas de mejorar un sistema. Se puede mejorar ofreciendo nuevas funcionalidades a los usuarios, mejorando el código fuente o mejorando la eficiencia y rendimiento de la aplicación.

9.1.1. Refactorizaciones

El primer cambio que se debería realizar quizá debería estar relacionado con el formato del PDF del parte de asistencia. Aunque los datos están completos, el archivo que genera actualmente la aplicación tiene diferencias significativas respecto al parte de asistencia actual de la UPV.

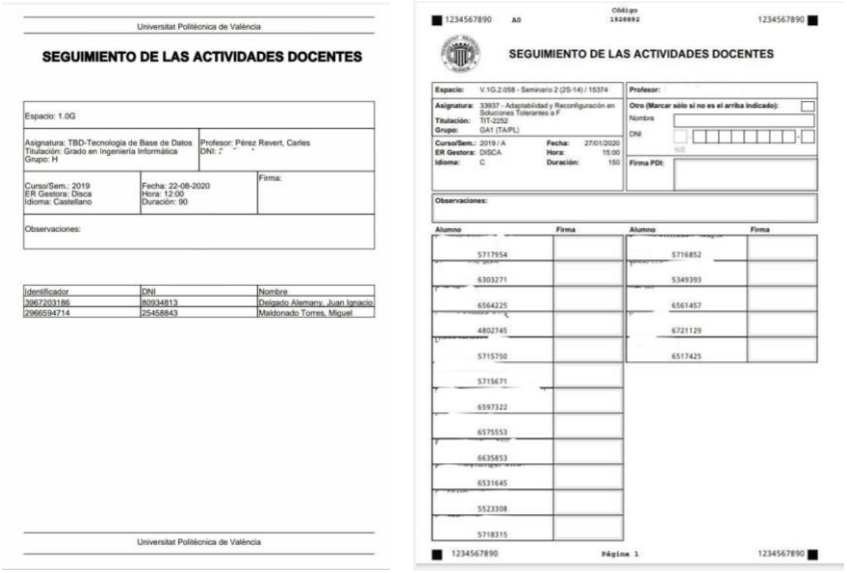

Figura 36. Comparativa de los partes de asistencia.

Aunque se recomienda que las refactorizaciones de código se efectúen en tiempo de programación, dado su estado actual sería recomendable dedicar un tiempo a seguir mejorando el código. A pesar de haber dado los primeros pasos para brindar al *software* de una arquitectura, en algunas partes de la aplicación no se han corregido estos errores. Por ejemplo, en algunas clases que corresponden a la función de controlador sigue habiendo consultas directas a la base de datos que no se han podido corregir por falta de tiempo.

También sería conveniente rediseñar la estructura de la base de datos ya que, como hemos comentado en capítulos anteriores, no utiliza la relaciones entre diferentes tablas; actualmente se hace de forma manual.

9.1.2. Nuevas funcionalidades

Dado que la vista para alumnos no era un objetivo prioritario de este proyecto, esta se limita a unas funciones básicas. Sería interesante de cara al futuro seguir desarrollando esta parte de la aplicación o incluso crear una aplicación independiente para alumnos. Igual que los profesores pueden consultar los partes que han generado, los alumnos deberían poder ver sus registros de asistencia.

Otra funcionalidad que sería útil es que el usuario pudiese consultar sus horarios, ya sea profesor o alumno. Realmente, la aplicación ofrece una gran cantidad de líneas de desarrollo posibles, pero en la mayoría de estas líneas es necesario que la UPV se implique en el proyecto.

Por falta de tiempo no se le han podido añadir idiomas a la aplicación. Pero teniendo en cuenta que la UPV es una universidad valenciana, la app debería contar con una versión en valenciano. De la misma manera, podría contar con otros idiomas, como inglés, francés o alemán ya que en la UPV hay alumnos de Erasmus.

9.2. Implicación de la UPV en el proyecto

La implicación de la UPV podría favorecer tanto el uso de la aplicación como impulsar su desarrollo. Actualmente la aplicación no puede acceder a ningún tipo de dato que no sea el del usuario de la TUI. Si esta aplicación dispusiera de acceso a alguna API de la UPV obtendría múltiples beneficios, puesto que nos permitiría simplificar procesos, como el proceso de agregado de alumnos ya que a través de los datos del profesor se podrían obtener los datos de todas sus asignaturas y alumnos.

Además, se podría añadir una pantalla de inicio de sesión ya que contaríamos con medios para verificar que el usuario es un miembro de la UPV. De esta forma, tendríamos distintos métodos para autenticar al usuario: autenticación biométrica, TUI y contraseña o PIN de la UPV.

La implicación de la UPV también significaría una gran reducción de código. Esto es debido a que algunos procesos se podrían eliminar de la aplicación ya que a partir de ese momento se ejecutarían en los servidores.

La persistencia de los datos también estaría asegurada. Los datos dejarían de almacenarse exclusivamente en el dispositivo del usuario para almacenarse en los servidores. Así, el usuario podría consultar todos sus registros aun cambiando de teléfono o de dispositivo.

La aplicación se podría difundir por diferentes medios pertenecientes a la UPV. De esta forma se daría a conocer la aplicación a alumnos y profesores, animándolos a utilizar la aplicación

Finalmente, teniendo en cuenta lo acontecido en los primeros meses de 2020, con una pandemia y las escuelas, institutos y universidades realizando clases online, esta app podría facilitar el fichaje de alumnos incluso cuando las clases no son presenciales. Con algunos retoques esta app podría solucionar los problemas de fichaje cuando la asistencia se produce de manera remota.

Bibliografía

1. **Wisconsin Historical Society.** Wisconsin Historical Society Website. *Workman's Time Recorder*. [En línea] 30 de Agosto de 2007. [Citado el: 20 de Julio de 2020.] https://www.wisconsinhistory.org/Records/Article/CS2674.

2. **Ponce Martínez, Adrián.** Sistema de bajo coste para la generación automática de partes de firmas utilizando el carné de la UPV. *Riunet UPV.* [En línea] 3 de Junio de 2020. [Citado el: 8 de Agosto de 2020.] http://hdl.handle.net/10251/145466.

3. **Rouse, Margaret y Mell, Emily.** TechTarget. *Lecagy application.* [En línea] Abril de 2017. [Citado el: 21 de Julio de 2020.] https://searchitoperations.techtarget.com/definition/legacy-application.

4. **Podio, Fernando Luís.** *Biometrics: global challenges and customer needs.* Ginebra : Central Secretariat of ISO (International Organization for Standarization), 2005. págs. 12-13. ISSN 1729-8709.

5. **Naiya, Pavel.** More than one billion smartphones to feature facial recognition in 2020. *Counterpoint research Web site.* [En línea] Counterpoint, 7 de Febrero de 2018. [Citado el: 4 de Agosto de 2020.] https://www.counterpointresearch.com/one-billion-smartphones-feature-face-recognition-2020/.

6. **Tomás Gironés, Jesús.** *El gran libro de Android.* s.l. : Marcombo, 2017. ISBN: 978-84-267-2256-0.

7. **Flanagan, David.** *Java in a nutshell.* Sebastopol : O'Reilly, 2005. ISBN: 0-596-00773-6.

8. **Singh, Neeraj Kumar.** *Near-field Communication (NFC).* s.l. : Information Technology and Libraries, 2020.

9. **Martin, Robert Cecil.** *Clean Code: A Handbook of Agile Software Craftsmanship.* Stoughton, Massachusetts : Prentice Hall, 2008. ISBN 0-13-235088-2.

10. **Fowler, Martin.** *Refactoring: Improving the Design of Existing Code.* Reading, Massachusetts : Addison–Wesley, 2014. ISBN 0-201-48567-2.

11. **Samuel, Stephen y Bocutiu, Stefan.** *Programming Kotlin.* Birmingham : Packt Publishing, 2017. ISBN 978-1-78712-636-7.

12. **Cao, Jerry.** UxPin. *Why Designers Shouldn't Neglect Mockups.* [En línea] 18 de Marzo de 2015. [Citado el: 22 de Julio de 2020.] https://www.uxpin.com/studio/blog/designers-shouldnt-neglect-mockups/#:~:text=The%20main%20role%20of%20mockups,atmosphere%20created%20from%20its%20appearance.&text=As%20such%2C%20mockups%20have%20an,more%20digestible%20to%20clients%20stakeholders..

13. **Keil, Jan Martin**. Efficient Bounded Jaro-Winkler Similarity Based Search. 2019. 10.18420/btw2019-13.

Anexo A. Formato de los archivos XML

El archivo XML del cual se cargan los datos de las asignaturas debe tener el siguiente formato:

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
    <Asignaturas>

        <!-- N es el número total de asignaturas -->
        <NumeroAsignaturas>N</NumeroAsignaturas>

        <AsignaturaX>
            <Identificador>ID</Identificador>
            <Nombre>Nombre de la asignatura</Nombre>
            <Titulacion>Grado en Ingeniería Informática</Titulacion>
            <Curso>2019/2020</Curso>
            <Gestora>DSIC</Gestora>
            <Idioma>Valencià</Idioma>
            <Duracion>90</Duracion>
        </AsignaturaX>

        <!-- Más grupos si los hubiera -->
        ...
    </Asignaturas>
</resources>
```

Fragmento de código AA 1. Ejemplo de XML del cual se pueden extraer los datos de las asignaturas.

La duración es la duración de cada clase de esa asignatura, en minutos. La gestora es el departamento encargado de dicha asignatura.

El fichero del cual se cargan los datos de las asignaturas sigue un formato similar:

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
    <Grupos>
        <!-- N es el número total de grupos -->
        <NumeroDeGrupos>N</NumeroDeGrupos>

        <!-- X es el número del grupo -->
        <GrupoX>
            <!--
            El identificador del grupo tiene el formato:
            "Id asignatura"-"Nombre del grupo"
            -->
            <Identificador>Id-Grupo</Identificador>
            <grupo>Grupo</grupo>
            <hEntrada>13:00</hEntrada>
            <hSalida>14:00</hSalida>
            <Aula>0.4G</Aula>
        </GrupoX>

        <!-- Más grupos si los hubiera -->
```

```
        ...
    </Grupos>
</resources>
```

Fragmento de código AA 2. Ejemplo de XML del cual se pueden extraer los datos de los grupos.

El identificador del grupo debe seguir la estructura indicada en el comentario del anterior fragmento para poder vincular el grupo a su asignatura correctamente. La etiqueta grupo indica el nombre del grupo.

Anexo B. Listado de *bugs*

B-1: Error en la carga de asignaturas XML

Cuando se inicia por primera vez la app y vamos directamente a la parte de gestión de asignaturas, cuando intentamos cargar los datos de estas, se produce un error y sólo muestra un listado con el elemento *null*.

Paso 1: Iniciar app por primera vez.

Paso 2: Seleccionar segunda opción del menú.

Paso 3: En el aviso seleccionar la opción “Cargar asignaturas propias”.

B-2: Error al compartir parte

Cuando se intenta enviar un parte por correo la aplicación se detiene de manera inesperada.

Paso 1: En la actividad principal seleccionar la primera opción (iniciar parte).

Paso 2: Leer el identificador de la TUI.

Paso 3: Pulsar “Siguiente”.

Paso 4: Seleccionar una asignatura del listado.

Paso 5: Pulsar “Asistencia Extraoficial”.

Paso 6: Pulsar “Iniciar Asistencia”.

Paso 7: Pulsar “Enviar PDF por correo”.

Paso 8: Rellenar el campo con una dirección de correo electrónico.

Paso 9: Pulsar enviar.

B-3: Problema al añadir alumno.

Al añadir un nuevo alumno a una asignatura la app no vuelve a la pantalla anterior automáticamente.

Paso 1: Seleccionar la segunda opción de la actividad principal.

Paso 2: Seleccionar una asignatura y pulsar “Ver”.

Paso 3: Seleccionar “Ver alumnos asignatura”.

Paso 4: Seleccionar la opción “Añadir alumno” de la ventana de diálogo.

Paso 5: Escanear el código QR de la TUI.

Paso 6: Leer el identificador de la TUI.

Paso 7: Pulsar “Añadir alumno”.

B-4: Problema al añadir profesor.

Al añadir un nuevo profesor a una asignatura la app no vuelve a la pantalla anterior automáticamente.

Paso 1: Seleccionar la segunda opción de la actividad principal.

Paso 2: Seleccionar una asignatura y pulsar “Ver”.

Paso 3: Seleccionar “Ver profesores asignatura”.

Paso 4: Seleccionar la opción “Añadir profesor” de la ventana de diálogo.

Paso 5: Escanear el código QR de la TUI.

Paso 6: Leer el identificador de la TUI.

Paso 7: Pulsar “Añadir profesor”.

B-5: Desincronización de datos.

Al editar los datos de un alumno, los datos no se muestran actualizados.

Paso 1: Seleccionar la segunda opción de la actividad principal.

Paso 2: Seleccionar una asignatura y pulsar “Ver”.

Paso 3: Seleccionar “Ver alumnos asignatura”.

Paso 4: Seleccionar un alumno y pulsar “Ver”.

Paso 5: Pulsar “Editar”.

Paso 6: Editar un campo y pulsar “Finalizar”.

B-6: Error en la edición de alumnos.

Al editar los datos de un alumno, volver a la pantalla de información de un alumno y volver a pulsar editar se muestra una pantalla sin información.

Al editar los datos de un alumno, los datos no se muestran actualizados.

Paso 1: Seleccionar la segunda opción de la actividad principal.

Paso 2: Seleccionar una asignatura y pulsar “Ver”.

Paso 3: Seleccionar “Ver alumnos asignatura”.

Paso 4: Seleccionar un alumno y pulsar “Ver”.

Paso 5: Pulsar “Editar”.

Paso 6: Editar un campo y pulsar “Finalizar”.

Paso 7: Pulsar “Editar”.

B-7: Grupo no relacionado con asignatura.

Al seleccionar una asignatura para iniciar un parte, el sistema carga el grupo que corresponde a la hora actual, aunque ese grupo no tenga ninguna relación con la asignatura. Por ejemplo, si son las 17:10, el sistema cargará el grupo con hora de inicio 17:00 y hora de fin 18:00, independientemente de la relación con dicha asignatura.

Paso 1: En la actividad principal seleccionar la primera opción (iniciar parte).

Paso 2: Leer el identificador de la TUI.

Paso 3: Pulsar “Siguiente”.

Paso 4: Seleccionar una asignatura del listado.

Paso 5: Pulsar “Asistencia Normal”.

B-8: Error al cargar el grupo

Al seleccionar una asignatura para iniciar un parte, sólo se selecciona correctamente el grupo si empieza a en punto (por ejemplo, 17:00) y dura una hora (siguiendo el ejemplo, termina a las 18:00). Si no se cumplen estas dos condiciones, el grupo nunca se selecciona.

Paso 1: En la actividad principal seleccionar la primera opción (iniciar parte).

Paso 2: Leer el identificador de la TUI.

Paso 3: Pulsar “Siguiente”.

Paso 4: Seleccionar una asignatura del listado.

Paso 5: Pulsar “Asistencia Normal”.

Anexo C. Resolución de *bugs*

En este anexo se explicará de qué manera se han resuelto los *bugs* descritos en el **Anexo B**.

B-1: Error en la carga de asignaturas XML

Este error se debía a que en ningún momento durante este proceso se solicitaba al usuario permiso para acceder al almacenamiento del dispositivo. De esta forma, al no tener permiso, la app no podía cargar el fichero XML de la carpeta Descargas.

Ahora se solicitan todos los permisos en la pantalla principal la primera vez que el usuario inicia la aplicación.

B-2: Error al compartir parte

No se ha detectado el motivo de este error, aunque tampoco ha sido necesario. Esta funcionalidad ha sido eliminada debido a que ahora ya no es necesaria.

Para compartir un parte ahora el usuario lo busca en el listado de Consulta de partes. En la pantalla de detalle ahora se puede compartir un parte a través de diferentes aplicaciones.

B-3: Problema al añadir alumno.

Cuando se pulsaba el botón para confirmar la adición del alumno no se ejecutaba ningún tipo de código que permitiese la vuelta atrás.

Se han añadido unas líneas de código para que al añadir un nuevo alumno la aplicación vuelva a la pantalla anterior automáticamente.

B-4: Problema al añadir profesor.

La causa y la solución es la misma que la del *bug* **B-3: Problema al añadir alumno**.

B-5: Desincronización de datos.

El error era debido a un error en la consulta a la base de datos. La sentencia SQL no buscaba al alumno por su identificador sino por su DNI. Cuando se editaba el campo DNI del alumno, al volver atrás no se encontraba al alumno y los datos no se actualizaban.

Ahora se busca al alumno a través de su identificador, por lo que los datos ya se muestran siempre actualizados.

B-6: Error en la edición de alumnos.

La causa de este error es el mismo que el del **B-5: Desincronización de datos**. Además, al resolver el error anterior también se ha resuelto este.

B-7: Grupo no relacionado con asignatura.

Este error se debe también a un error en la sentencia SQL que se usa para consultar a la base de datos. Ahora en la sentencia no se incluye ninguna referencia a la asignatura, por lo que si el grupo seleccionado pertenece a la asignatura será simplemente casualidad.

Ahora en la sentencia se ha insertado el identificador de la asignatura para que en la búsqueda se tenga en cuenta.

B-8: Error al cargar el grupo

Una vez más, el error está relacionado con una sentencia SQL defectuosa. El proceso de selección de grupo era el siguiente:

1. El sistema mira la hora actual y la selecciona en su totalidad, es decir, con hora y minutos.
2. El sistema selecciona la hora en punto inmediatamente anterior.
3. El sistema comprueba que la hora de inicio del grupo está entre estas dos horas.

Por ejemplo, si son las 17:24, el sistema coge esta hora. A continuación, la hora en punto anterior, que serían las 17:00. De esta manera, si una clase empieza a las 16:45 ya no se seleccionaría, por lo que se estaría produciendo un error.

Para solucionar esto, hemos cambiado el proceso:

1. El sistema coge la hora de inicio y de fin de la clase.
2. El sistema comprueba que la hora actual esté entre estas dos horas.

De esta forma solucionamos el problema ya que se seleccionará correctamente el grupo. Si no existe un grupo que cumpla está condición se dejarán los campos vacíos para que el profesor los rellene.

Anexo D. Comparativa IU

El objetivo de este anexo es comparar la interfaz de la nueva versión con la antigua y comentar si se ha mejorado alguna funcionalidad, pero ignorando si se han añadido nuevas funcionalidades. Las pantallas de nueva creación que se han creado para contener nuevas funcionalidades no se mostrarán en este anexo.

La pantalla principal tenía antes tres opciones; ahora dos, debido a que estas opciones se han fragmentado en dos pantallas.

En cuanto a los botones, ya no son simplemente logos, ahora también tiene texto para facilitar describir mejor al usuario qué acciones puede realizar. El fondo de las pantallas ahora es liso, por lo que también facilita la lectura al usuario. Igualmente, se han cambiado los colores de la app por unos más vivos.

Figura AD 1. Comparativa pantalla principal.

A la pantalla principal se le ha añadido también el logo de la UPV. De esta forma, la aplicación tiene un aspecto más corporativo.

Cuando se quiere iniciar un nuevo parte el sistema solicita seleccionar una asignatura. Antes el profesor tenía que seleccionar la asignatura y luego hacer clic en el tipo de asistencia que quería generar. Ahora con un solo clic ya puede seleccionar la asignatura y es el sistema el que se encarga de comprobar si se trata de una asistencia normal o de recuperación.

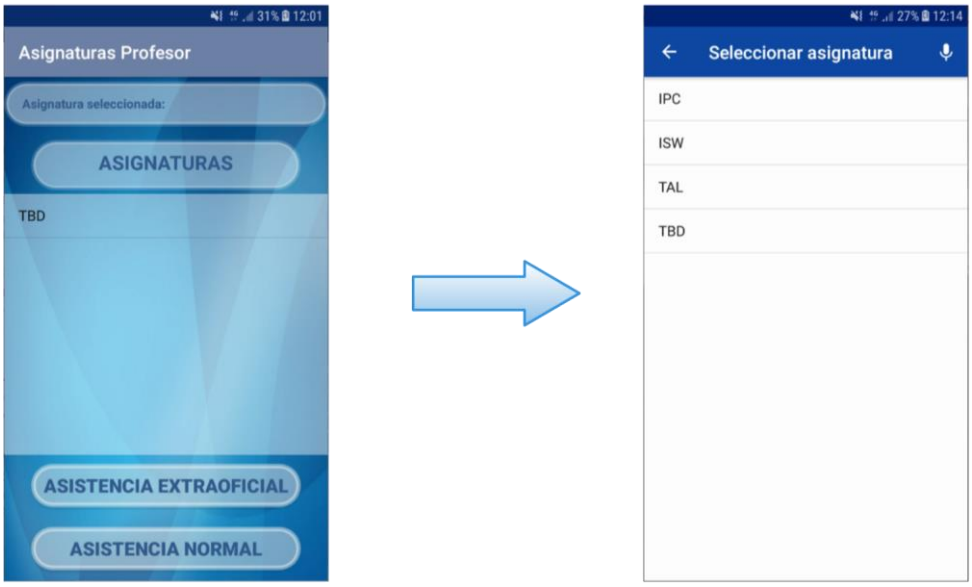

Figura AD 2. Comparativa de la pantalla de selección de asignatura.

Además, todos los listados de la aplicación se han renovado. Con un simple clic se puede seleccionar el elemento deseado. Para eliminar un elemento basta con hacer un clic largo sobre él (siempre que se un objeto que se pueda borrar).

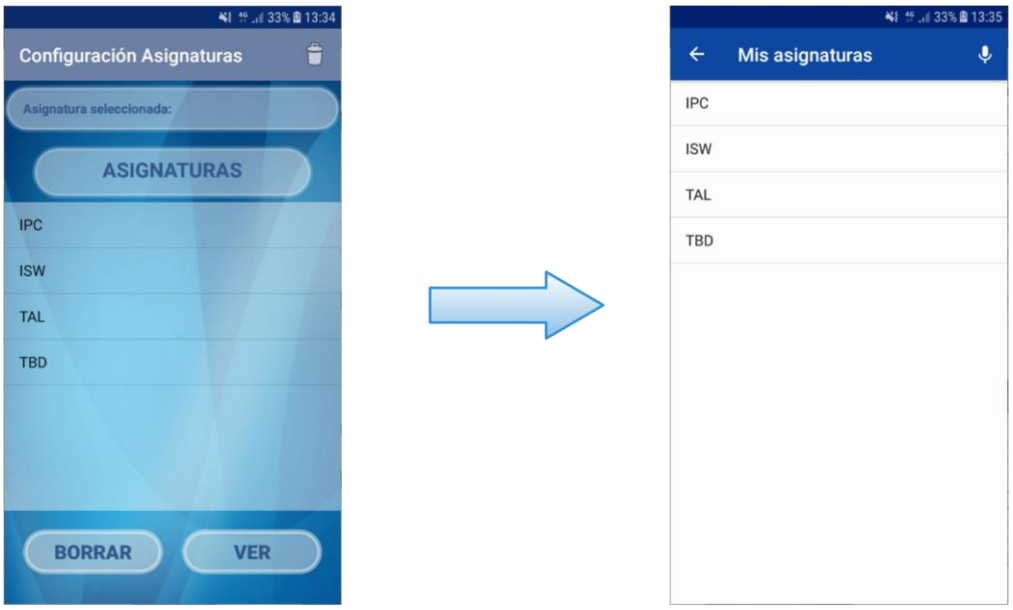

Figura AD 3. Comparativa de los listados.

Con este cambio ahorramos clics al usuario y le permitimos tener una experiencia más fluida.

En el proceso de generación de un nuevo parte, una vez el profesor ha seleccionado la asignatura, se muestra una pantalla con los detalles de la asignatura y del grupo. Antes esta pantalla era defectuosa ya que no se adaptaba a la pantalla del dispositivo.

Aprovechando esta renovación hemos aprovechado para permitir que el profesor pueda editar los datos relativos a esta clase, como la hora de inicio o su duración.

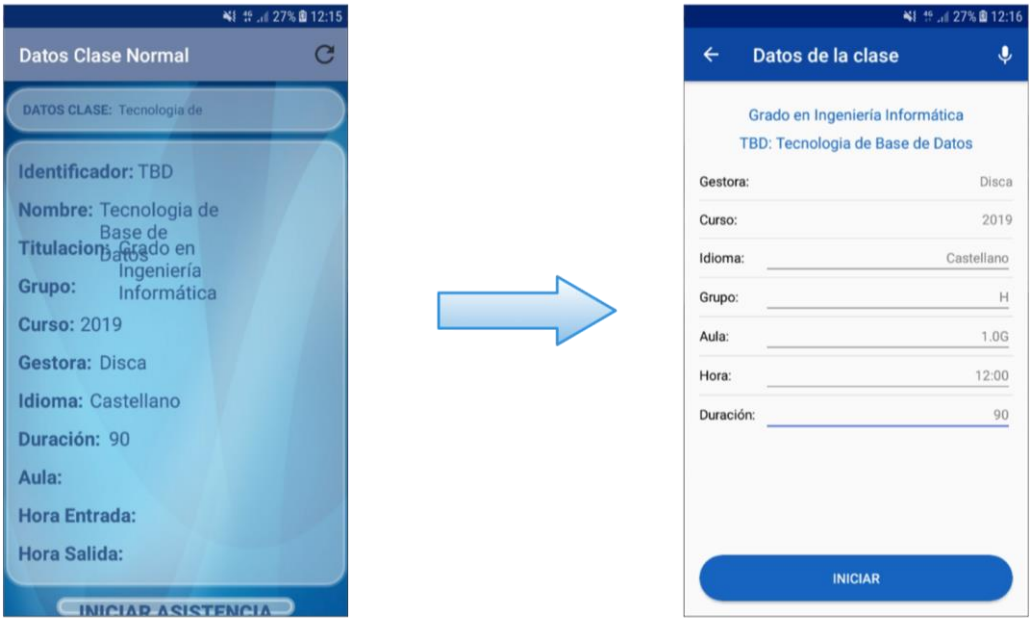

Figura AD 4. Comparativa de la pantalla de detalle de una clase.

La pantalla que permite a los alumnos fichar y al profesor generar el PDF se ha dividido en dos pantallas. Ahora la primera pantalla se limita a recibir los datos de los alumnos, mientras que la segunda pantalla es la que permite la gestión del parte.

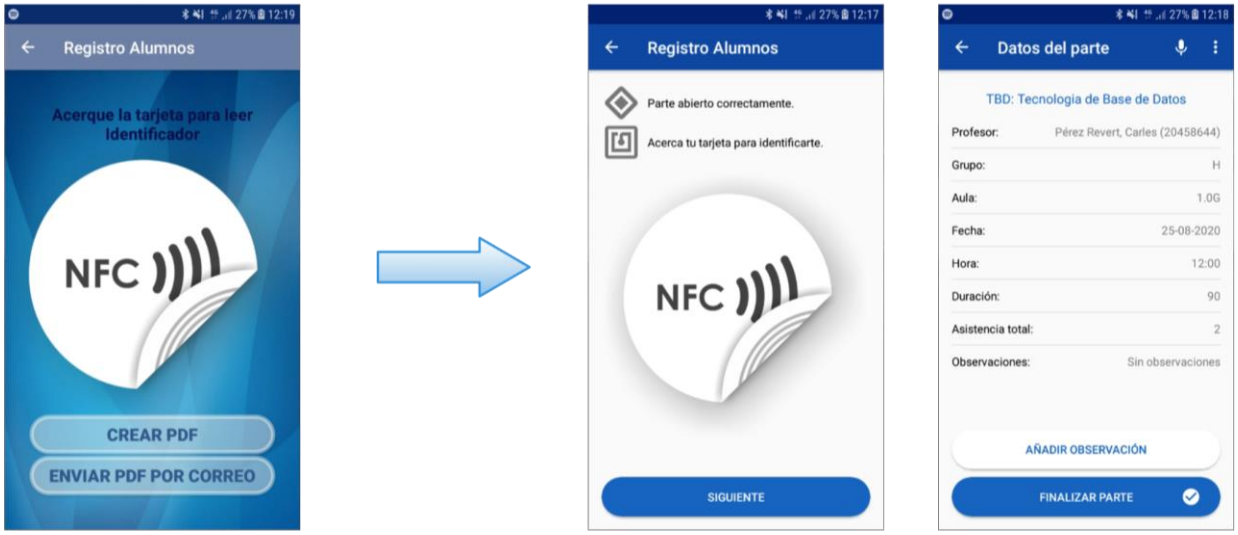

Figura AD 5. Comparativa de la pantalla de generación de partes de asistencia.

En cuanto a la gestión de asignaturas, antes no se mostraban los datos de la asignatura, simplemente se mostraba un menú. Se han unificado varias pantallas para ahorrar pasos al usuario. En una misma pantalla se muestra la información de la asignatura y se le permite al usuario realizar diferentes acciones, como editar la asignatura o gestionar los alumnos y grupos de la asignatura en cuestión.

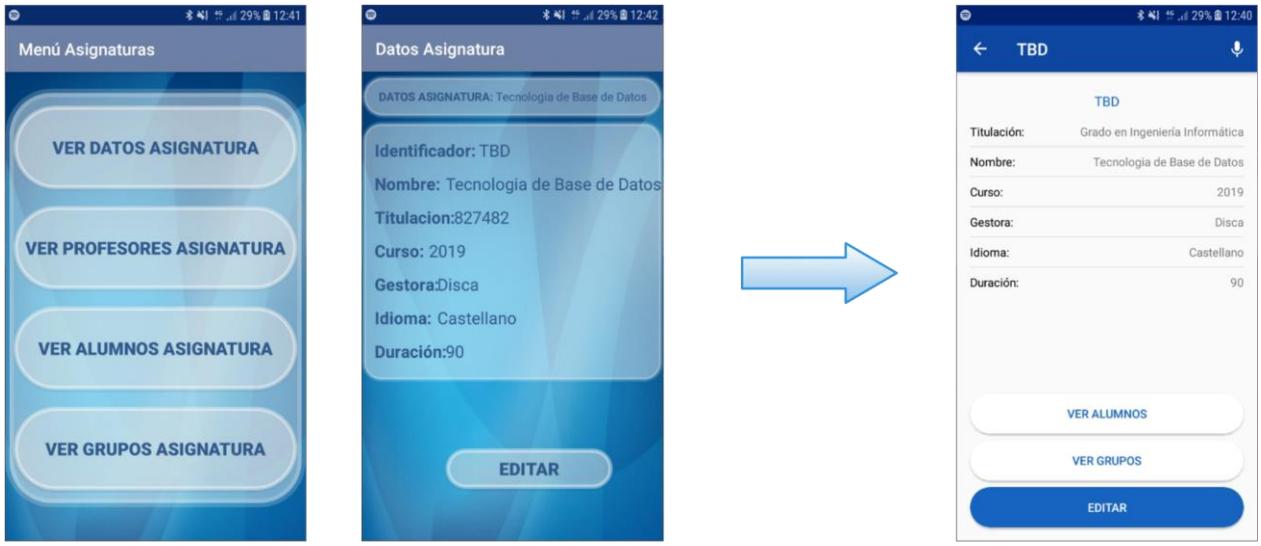

Figura AD 6. Comparativa de la pantalla de detalles de una asignatura.

La pantalla de edición de una asignatura también ha sufrido modificaciones. Ahora los datos se muestran de manera ordenada y estructurada, sin sobreponerse entre ellos. Todas las pantallas de edición de la aplicación muestran el mismo patrón.

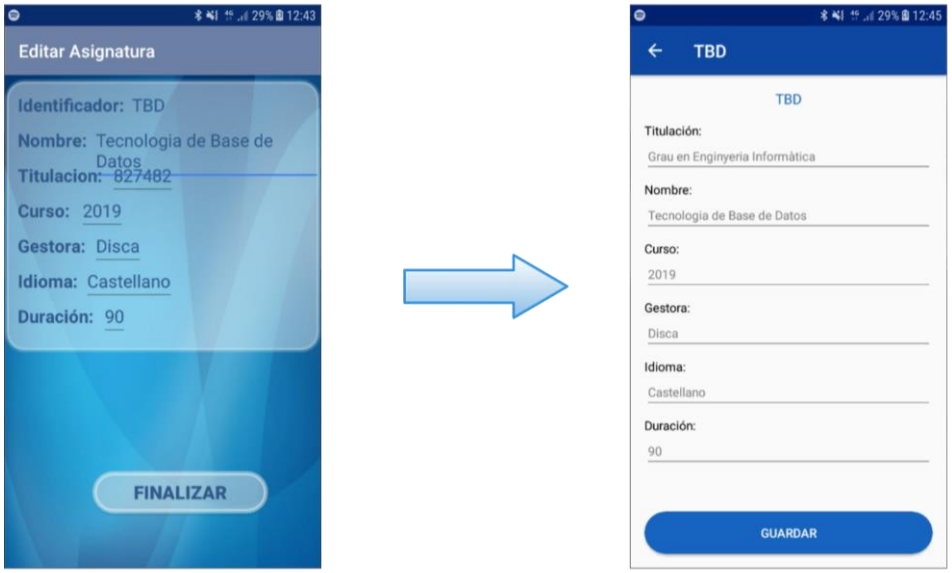

Figura AD 7. Comparativa de la pantalla de edición de una asignatura.

Cuando ha sido posible hemos añadido botones flotantes. De esta forma ahorramos clics al usuario y le mostramos de forma más clara las acciones que puede realizar.

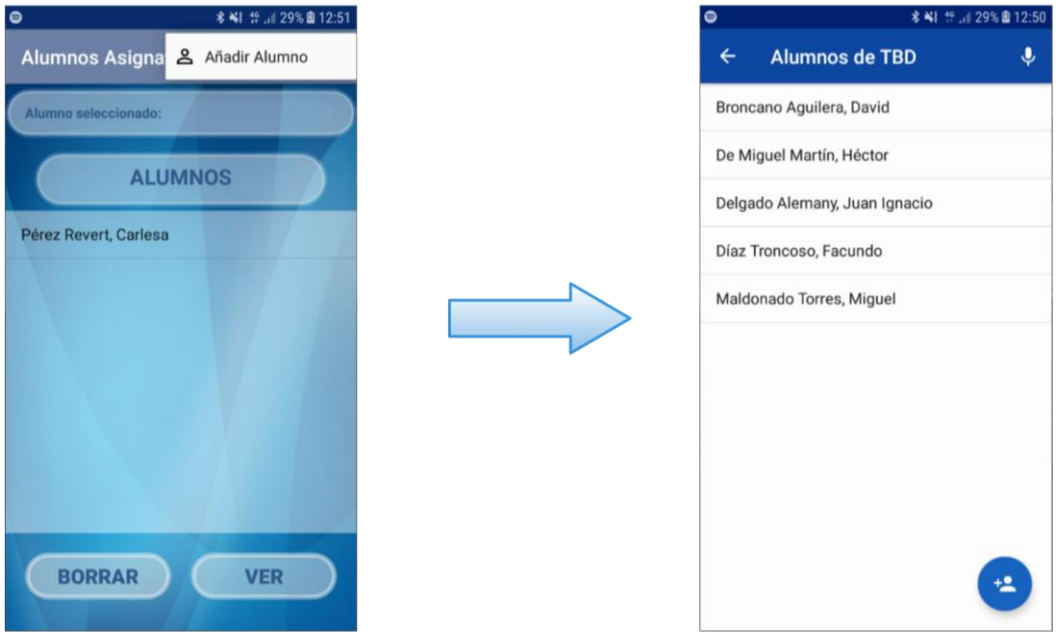

Figura AD 8. Comparativa de la lista de alumnos de una asignatura.

Para añadir un nuevo alumno a una asignatura se mostraba una pantalla con dos secciones: una para escanear el código QR y la otra para leer el identificador de la TUI. Esta pantalla presentaba botones similares a otros elementos de la interfaz.

La nueva pantalla tiene descripciones que le indican al usuario qué dos acciones debe realizar. Además, cuando realiza una acción la aplicación la marca como realizada.

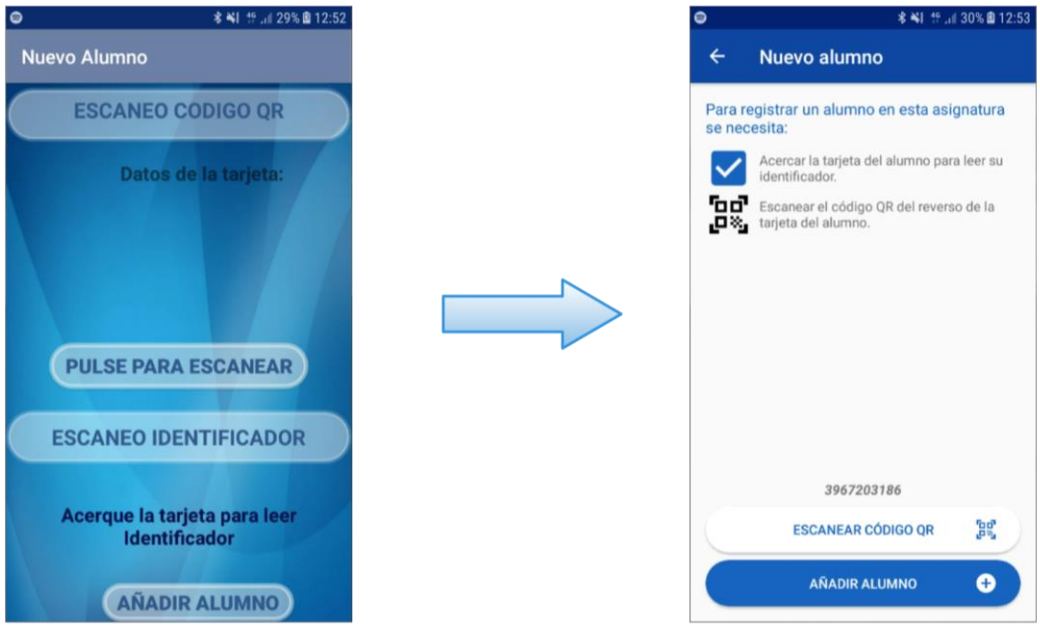

Figura AD 9. Comparativa de la pantalla de nuevo alumno.

Por último, las pantallas de Consulta de partes de asistencia también se han renovado. Antes se mostraba un filtro con el que buscar un parte en concreto. Esto se ha substituido por un listado con todos los partes generados. No obstante, se ha añadido un filtro que se puede abrir a través del botón flotante.

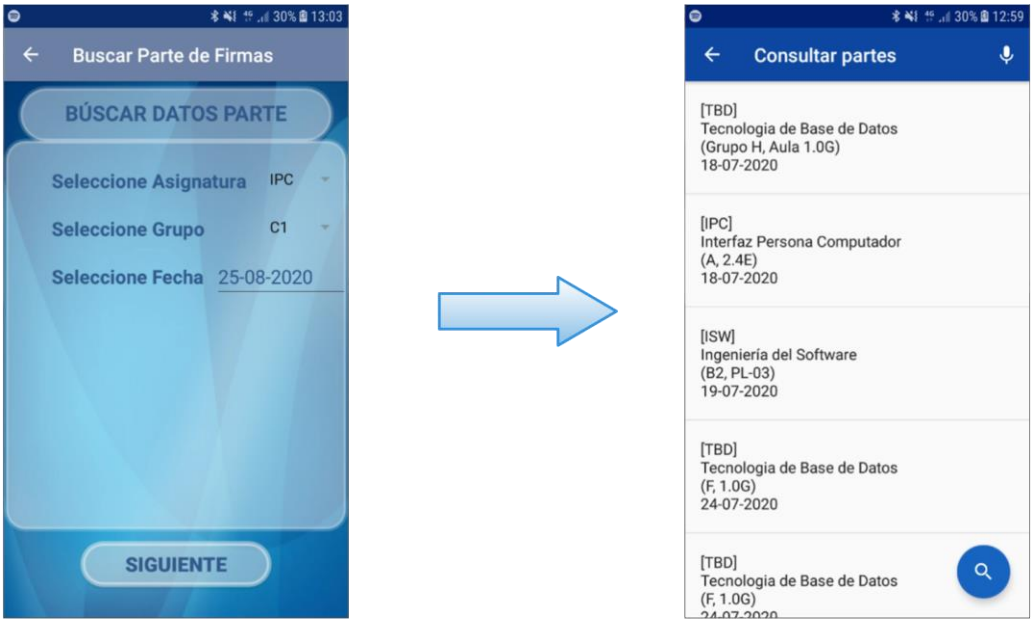

Figura AD 10. Comparativa de la pantalla de consulta de partes.

La pantalla de detalle de un parte de asistencia ahora muestra la información de manera estructurada. Además, esta pantalla contiene la funcionalidad de compartir partes.

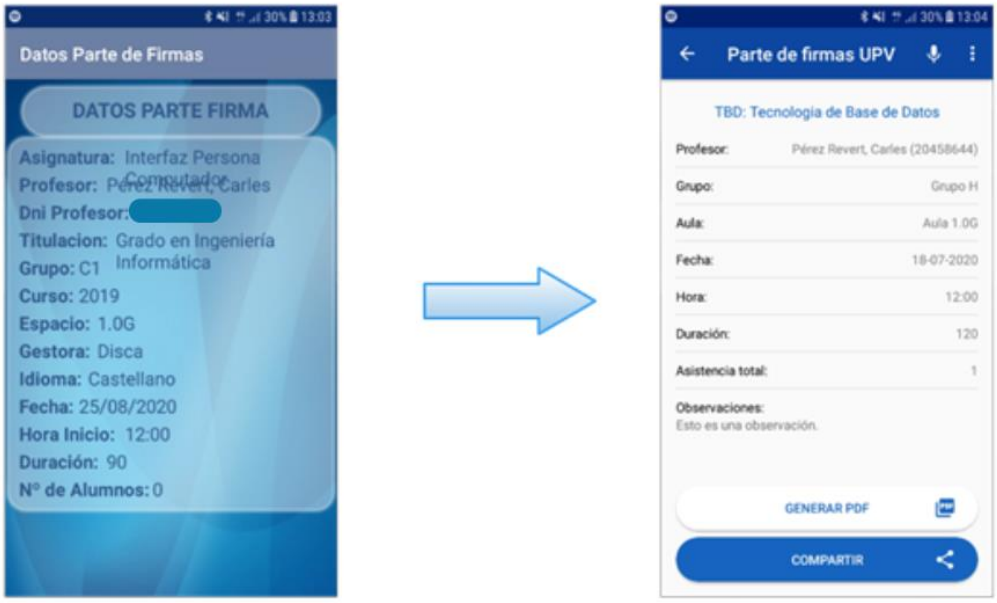

Figura AD 11. Comparativa de la pantalla de detalles de un parte de asistencia.

Anexo E. Funcionamiento de la app

En este anexo se explican los diferentes flujos de la aplicación. Los flujos de la vista para alumnos y el flujo de vinculación de la TUI al teléfono se mostrarán a través de un esquema con capturas de pantalla de la aplicación. En cambio, el flujo de la vista de profesor se hará con un esquema sin capturas de pantalla.

El flujo de vinculación de la TUI al teléfono es bastante básico ya que en cada pantalla se realiza una acción.

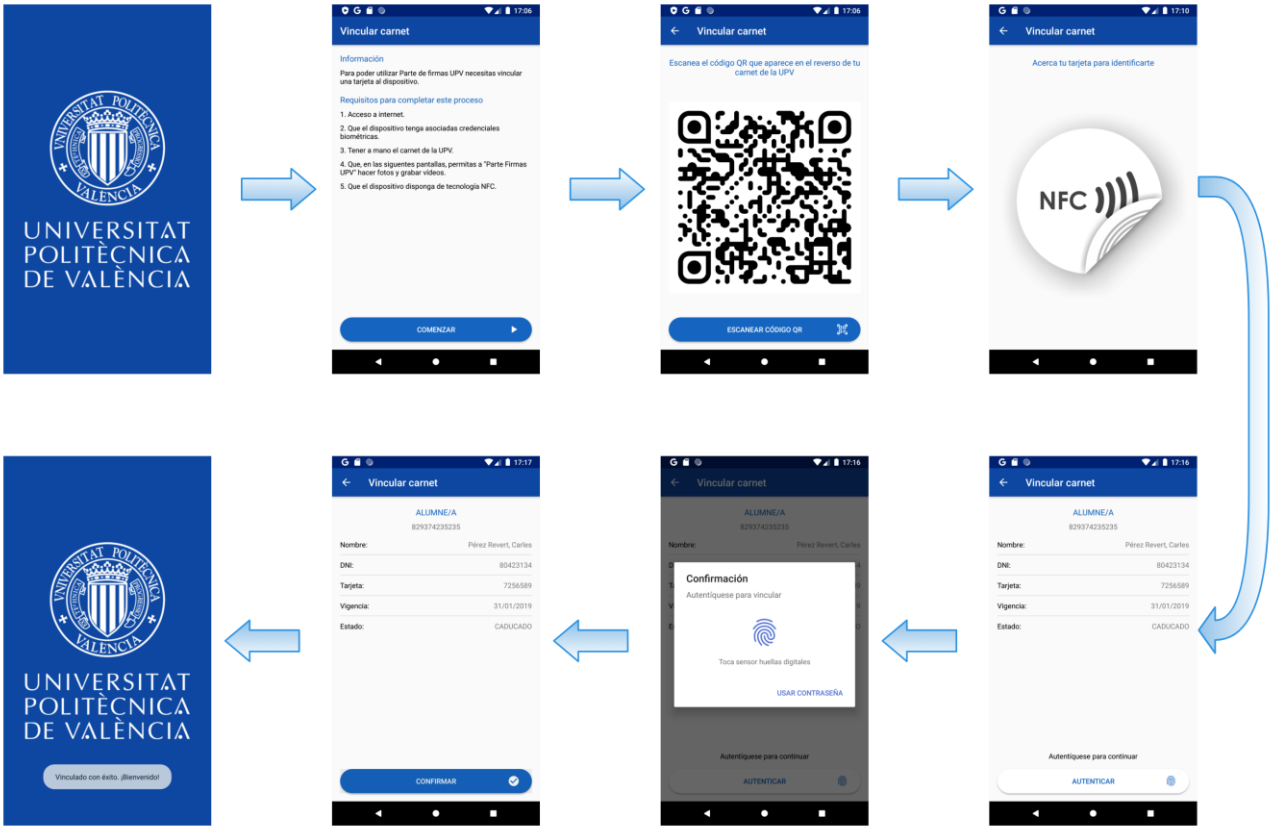

Figura AE 1. Proceso de vinculación de la TUI.

Una vez se inicia la aplicación, esta le muestra los requisitos necesarios para poder completar el proceso. Los siguientes pasos serán:

1. Escanear el código QR de la ´TUI.
2. Leer el identificador del alumno a través de la tecnología NFC (acercando la TUI al teléfono).
3. El usuario se autenticará mediante la autenticación biométrica o mediante las credenciales de pantalla de bloqueo.
4. El usuario confirmará los datos y la aplicación vinculará la TUI.

Cuando el usuario termina este proceso el sistema le redirige a la vista correspondiente.

La vista para alumnos tiene pocas funcionalidades actualmente. Desde la pantalla principal se puede navegar a la pantalla de datos del alumno y a la pantalla de fichaje.

1. Para fichar el alumno selecciona la opción “Buscar partes” en la pantalla principal.
2. La aplicación buscará partes de asistencia cercanos que estén abiertos.
3. El alumno seleccionará la asignatura en la que desea registrar su asistencia.
4. A continuación, se autenticará y su asistencia quedará registrada.

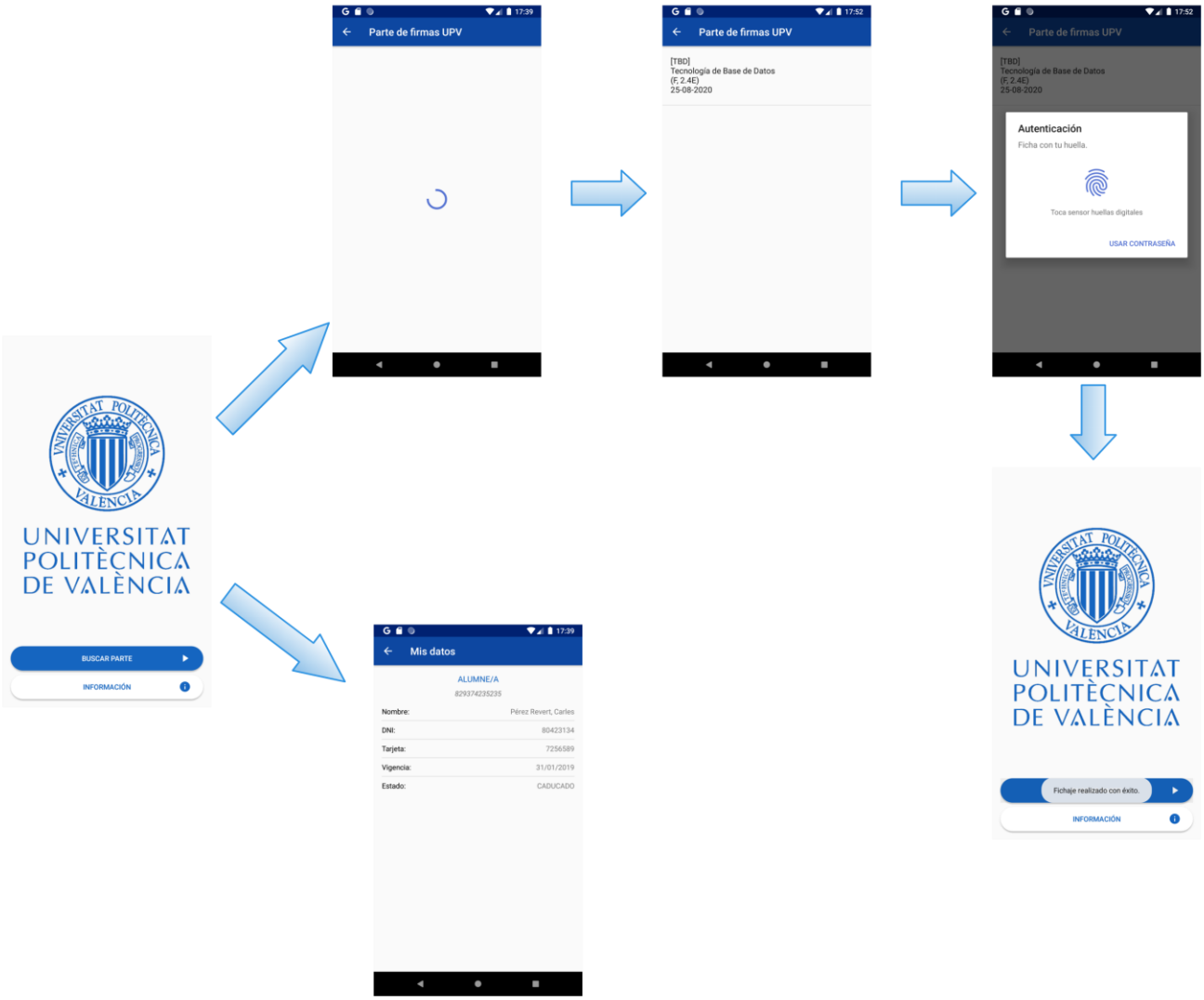

Figura AE 2. Procesos de la vista para alumnos.

La vista para profesor es más compleja. Se ha elaborado un esquema mostrando los diferentes flujos de esta vista. Las entidades a las que se puede navegar mediante los comandos por voz están marcadas con un micrófono, mientras que en los procesos en los que el sistema solicita la autenticación están marcados con una huella dactilar.

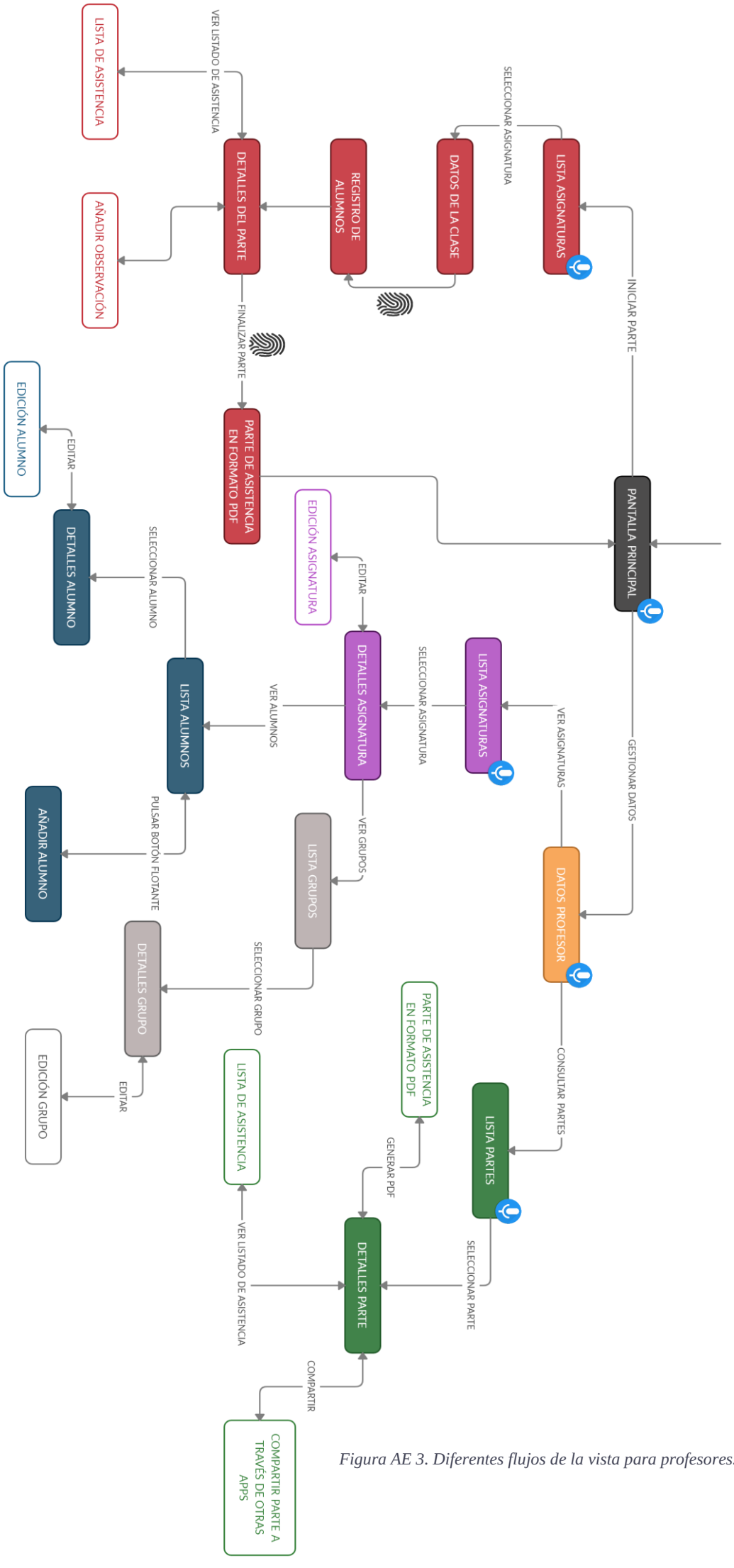

Figura AE 3. Diferentes flujos de la vista para profesores.

Anexo F. Nearby Connections API

La actividad responsable de recoger los fichajes de los alumnos es ***TeacherSignActivity***, mientras que la actividad encargada de que un alumno pueda fichar desde su teléfono es ***StudentSignActivity***. La primera clase pertenece a la vista para profesores, en cambio, la segunda pertenece a la vista para alumnos.

La clase ***TeacherSignActivity*** tiene un atributo de tipo ***Advertise***; por su parte, ***StudentSignActivity*** lo tiene de tipo ***Discover***. Tanto ***Advertise*** (publicador) como ***Discover*** (descubridor o buscador) son las clases principales de este proceso de intercambio de información entre profesores y alumnos.

```
public Advertise(
        Context context,
        String nickname,
        String serviceId,
        PayloadCallback payloadCallback
) {
    this.context = context;
    this.nickname = nickname;
    this.serviceId = serviceId;
    this.payloadCallback = payloadCallback;
}
```

Fragmento de código AF 1. Constructor de la clase Advertise.

El constructor de la clase ***Advertise*** recibe cuatro parámetros:

- `Context context`. Es una referencia a la actividad que hace uso de esta clase.
- `String nickname`. Es el texto que se mostrará en el teléfono del alumno.
- `String serviceId`. Este valor debe identificar a nuestra aplicación de manera única.
- `PayloadCallback payloadCallback`. Es el objeto que recibirá la información enviada por el alumno.

La clase ***Advertise*** también implementa el método ***start***, el cual se encarga de iniciar el publicador y hacer que el parte esté visible para los alumnos. Dentro del método ***start***, el atributo ***mConnectionsClient*** es el que se encarga de lanzar el publicador. En este proceso se selecciona la estrategia. La estrategia seguida en este proyecto es la ***P2P_CLUSTER***; esto significa que la relación de profesores a alumnos será de *n* a *m*, siendo *n* y *m* números naturales (incluyendo el cero). El atributo ***connectionLifecycleCallback*** es el que recibe los eventos relacionados con el ciclo de vida del publicador.

```
public void start() {
    mConnectionsClient = Nearby.getConnectionsClient(context);

    AdvertisingOptions advertisingOptions = new AdvertisingOptions.Builder().setStrategy(
            NearbyInterface.STRATEGY
    ).build();
```

```
    mConnectionsClient
            .startAdvertising(
                    nickname, serviceId, connectionLifecycleCallback, advertisingOptions
            )
            .addOnSuccessListener( (Void unused) -> {
                notifyObservers();
                Log.d(NearbyInterface.LOG, "Anunciante iniciado..." );
            })
            .addOnFailureListener( (Exception e) -> {
                Log.d(NearbyInterface.LOG, "Se ha producido un error...\n" + e.getMessage());
            });
}
```

Fragmento de código AF 2. Método start de la clase Advertise.

Cuando el publicador se inicia con éxito, haciendo uso del patrón *Observer* se comunica a la actividad que el parte se ha abierto correctamente. Así, logramos mostrar por pantalla esta información. Previamente, desde el método ***onResume*** de la actividad ***TeacherSignActivity***, habremos llamado al método ***start***.

```
report = (Report) getIntent().getSerializableExtra("ReportObject");
subject = report.getSubject();

String nickname = report.toString();
advertise = new Advertise(
        this,
        nickname,
        getApplicationContext().getPackageName(),
        payloadCallback
);

ProgressBar progressBar = findViewById(R.id.progressBar2);
TextView nearbyStatus = findViewById(R.id.textView16);
advertise.addObserver( () -> {
    nearbyStatus.setText("Parte abierto correctamente.");
    progressBar.setVisibility(View.INVISIBLE);
});
```

Fragmento de código AF 3. Código del método onCreate de la clase TeacherSignActivity.

Además, una vez el publicador se ha iniciado con éxito, este se encuentra preparado para establecer conexiones. Cuando reciba alguna petición de conexión la aceptará automáticamente.

```
private final ConnectionLifecycleCallback connectionLifecycleCallback = new
ConnectionLifecycleCallback() {
    @Override
    public void onConnectionInitiated(String endpointId, ConnectionInfo connectionInfo) {
        // Automatically accept the connection on both sides.
        Log.d(NearbyInterface.LOG, "Aceptando conexión con el cliente...");
        mConnectionsClient.acceptConnection(endpointId, payloadCallback);
    }

    /* Código omitido */
}
```

Fragmento de código AF 4. Atributo que recibe los eventos de clico de vida de conexión del publicador de la clase Advertise.

La actividad ***StudentSignActivity*** funciona de manera similar a ***TeacherSignActivity***. Esta actividad tiene un atributo de tipo ***Discover***, como hemos comentado anteriormente. El constructor de la clase ***Discover*** recibe los mismos parámetros que la clase ***Advertise***. Sin embargo, en este constructor se inicializan los atributos ***nConnectionsClient*** (se encarga de lanzar el descubridor) y ***map***, (almacena los partes de asistencia detectados por el descubridor). Esta clase también tiene un método ***start***; funciona igual que su homónimo de la clase ***Advertise***.

```
public Discover(
        Context context,
        String nickname,
        String serviceId,
        PayloadCallback payloadCallback
) {
    this.nickname = nickname;
    this.serviceId = serviceId;
    this.payloadCallback = payloadCallback;

    mConnectionsClient = Nearby.getConnectionsClient( context );
    map = new HashMap<>();
}
```

Fragmento de código AF 5. Constructor de la clase Discover.

El atributo que recibe los eventos de descubrimiento es ***endpointDiscoveryCallback***. En el momento que el descubridor encuentra un publicador le envía una petición de conexión. El publicador aceptará esta conexión y enviará una petición de conexión de vuelta al descubridor. El descubridor también aceptará la conexión automáticamente y, entonces, estarán preparados para empezar a intercambiar información.

```
private final EndpointDiscoveryCallback endpointDiscoveryCallback = new
EndpointDiscoveryCallback() {
    @Override
    public void onEndpointFound( String endpointId, DiscoveredEndpointInfo info) {
        Log.d(NearbyInterface.LOG, "Se ha encontrado un endpoint: " + endpointId);

        // An endpoint was found. We request a connection to it.
        mConnectionsClient.requestConnection(
                nickname,
                endpointId,
                connectionLifecycleCallback
        ).addOnSuccessListener(
                (Void unused) -> {
                    // We successfully requested a connection.
                    // Now both sides must accept before the connection is established.
                    Log.d(NearbyInterface.LOG, "Se ha enviado una petición de conexión...");
                }
        ).addOnFailureListener(
                (Exception e) -> {
                    Log.d(
                        NearbyInterface.LOG,
                        "Se ha producido un error...\n" + e.getMessage() + " " + e.hashCode()
                    );
```

```
                    if (
                        e.getMessage().equals("8003: STATUS_ALREADY_CONNECTED_TO_ENDPOINT")
                        && !map.containsKey(endpointId)
                    ) {
                        map.put(endpointId, info.getEndpointName());
                        notifyObservers(map);
                    }

                });
    }

    @Override
    public void onEndpointLost(@NonNull String endpointId) {
        // A previously discovered endpoint has gone away.
        Log.d(NearbyInterface.LOG, "Se ha perdido la conexión con: " + endpointId);
    }
};
```

Fragmento de código AF 6. Atributo que recibe los eventos de descubrimiento del buscador de la clase Discover.

La clase ***Discover*** también implementa el patrón *Observer*, pero esta lo usa para informar al alumno que ha encontrado (y conectado con éxito) un parte de asistencia. Así, se añadirá la información del parte encontrado al listado que se muestra por pantalla al alumno. Igualmente, si se pierde una conexión con el profesor, la aplicación también informará al alumno y se eliminará el parte de la pantalla.

```
private final ConnectionLifecycleCallback connectionLifecycleCallback =
        new ConnectionLifecycleCallback() {

            String endpointName;

            @Override
            public void onConnectionInitiated(
                    String endpointId,
                    ConnectionInfo info
            ) {
                // Automatically accept the connection on both sides.
                Log.d(NearbyInterface.LOG, "Aceptando conexión con el servidor...");
                endpointName = info.getEndpointName();
                mConnectionsClient.acceptConnection(endpointId, payloadCallback);
            }

            @Override
            public void onConnectionResult(
                    String endpointId,
                    ConnectionResolution result
            ) {
                switch (result.getStatus().getStatusCode()) {
                    case ConnectionsStatusCodes.STATUS_OK:
                        Log.d(NearbyInterface.LOG, "¡SE HA CONECTADO!");
                        map.put(endpointId, endpointName);
                        notifyObservers(map);
                        break;
                    /* Código omitido */
                }
            }
```

```
            @Override
            public void onDisconnected(@NotNull String endpointId) {
                map.remove(endpointId);
                Log.d(NearbyInterface.LOG, "Desconectado...");
                notifyObservers(map);
            }
        };
```

Fragmento de código AF 7. Atributo que recibe los eventos de ciclo de vida de conexión del descubridor de la clase Discover.

El proceso de intercambio de datos lo inicia el alumno. Desde su teléfono se envía una matriz de bytes que contiene:

- el identificador de la conexión con el profesor (para que Nearby sepa a quién está dirigida la información),
- el identificador del alumno (para que quede registrada su asistencia)
- y un valor que identifica el teléfono del alumno (para evitar que desde un mismo dispositivo fichen dos alumnos diferentes).

```
@Override
public void sendPayload(String endpointId, byte[] bytes) {
    mConnectionsClient.sendPayload(endpointId, Payload.fromBytes(bytes));
    Log.d(NearbyInterface.LOG, "Sending payload");
}
```

Fragmento de código AF 8. Método sendPayload de la clase Advertise.

Tanto en la clase ***Discover*** como en la clase ***Advertise*** se sobrescribe el método ***sendPayload***. Este método es el encargado de transmitir información hasta el otro punto. En cambio, esta información la reciben los atributos llamados ***payloadCallback***. Este atributo lo contienen las dos actividades que participan en este proceso, y en ambas recibe el mismo nombre.

Cuando el teléfono del profesor recibe una petición de un alumno para registrar su asistencia, se comprueban estos tres aspectos:

1. Que el alumno está registrado en la asignatura en cuestión.
2. Que el alumno no ha fichado anteriormente.
3. Que el dispositivo, desde el que se envía la petición, no ha realizado ninguna otra anteriormente.

Si estos tres aspectos se cumplen, se comunicará al alumno que su fichaje ha sido registrado con éxito.

```
private val payloadCallback = object : PayloadCallback() {
    override fun onPayloadReceived(endpointId: String, payload: Payload) {
        // This always gets the full data of the payload.
        // Will be null if it's not a BYTES payload.
        // You can check the payload type with payload.getType().
        val receivedBytes = payload.asBytes()
        val msg = String(receivedBytes!!, StandardCharsets.UTF_8)
        toast(msg)
        if (NearbyCode.SUCCESS.msg == msg) {
            val intent = Intent(applicationContext, MainStudentActivity::class.java)
```

```
            startActivity(intent)
            discover.stop()
            finish()
        }
    }

    override fun onPayloadTransferUpdate(endpointId: String, payloadTransferUpdate:
PayloadTransferUpdate) {
        // Bytes payloads are sent as a single chunk,
        // so you'll receive a SUCCESS update immediately
        // after the call to onPayloadReceived().
    }
}
```

Fragmento de código AF 9. Atributo payloadCallback de la actividad SignStudentActivity.

Cuando el proceso llegue a su fin, publicador y descubridor pondrán fin a su labor. Para ello, desde las respectivas actividades se llamará al método ***stop***. Tanto el publicador como el descubridor implementan este método.

```
@Override
public void stop() {
    map.clear();
    mConnectionsClient.stopAllEndpoints();
    mConnectionsClient.stopDiscovery();
    Log.d(NearbyInterface.LOG, "Stop discovery");
}
```

Fragmento de código AF 10. Método stop de la clase Discover.