



UNIVERSIDAD DE GRANADA

MEMORIA TÉCNICA MÓVIL

Asignatura:

NUEVOS PARADIGMAS DE INTERACCIÓN

Grupo 09:

CARMEN AZORÍN MARTÍ, MARÍA CRIBILLÉS PÉREZ,

ANA GRACIANI DONAIRE, JAIME MARTÍNEZ BRAVO,

JONAS RIEMANN

Índice

1. Introducción	3
2. Descripción general de la aplicación	3
2.1. Funcionalidades principales	3
2.1.1. Docencia con sensor NFC	5
2.1.2. Localización	11
2.1.3. Trámites	11
2.1.4. Mis pagos con sensor cámara para escanear QR	12
2.1.5. Comedor	20
2.1.6. Profesorado	21
2.1.7. OyeETSIIT	25
2.1.8. Gestión del login	27
2.1.9. Soporte multilingüe	30
3. Sensor NFC	31
4. Sensor multitáctil	34
4.1. Diseño e Implementación Técnica	34
4.2. Ventajas de la Funcionalidad	35
4.3. Pruebas y Validación	35
4.4. Conclusión	35
5. Uso del Sensor GPS	36
5.1. Funcionalidad General	36
5.2. Implementación Técnica	36
5.3. Ventajas del Enfoque Implementado	37
6. Navegación por Voz en el Menú Principal	39
6.1. Funcionamiento General	39
6.2. Implementación Técnica	39
6.3. Accesibilidad y Usabilidad	40
6.4. Conclusión	40
7. Reparto de trabajo	40

Nuevos Paradigmas de Interacción

Memoria técnica móvil

1. Introducción

La aplicación desarrollada tiene el propósito de ayudar a cualquier miembro de la Universidad de Granada a resolver sus dudas desde su propio dispositivo móvil.

En concreto, está enfocada en usuarios que sean alumnos de la Escuela Técnica Superior de Ingenierías Informática y Telecomunicaciones.

Para desarrollar la aplicación se ha utilizado Android Studio, junto con GitHub para poder gestionar las distintas versiones.

El alcance del proyecto es el de un prototipo: está dividido en varios submenús según la tarea a realizar, donde todos son funcionales, pero sólo para casos de usos concretos.

2. Descripción general de la aplicación

La aplicación ha sido desarrollada para estudiantes y personal de la ETSIIT con el propósito de facilitar el acceso a información útil sobre la escuela y sus servicios. No está destinada a realizar gestiones complejas, sino a ofrecer ayuda rápida y accesible, por ejemplo, para localizar puntos de interés, consultar horarios, realizar pequeños pagos o resolver dudas frecuentes.

2.1. Funcionalidades principales

La aplicación cuenta con un menú principal que organiza las funcionalidades de manera clara y accesible. Este menú, mostrado en la figura, está compuesto por varios submenús que permiten a los usuarios acceder rápidamente a las distintas secciones de la aplicación.

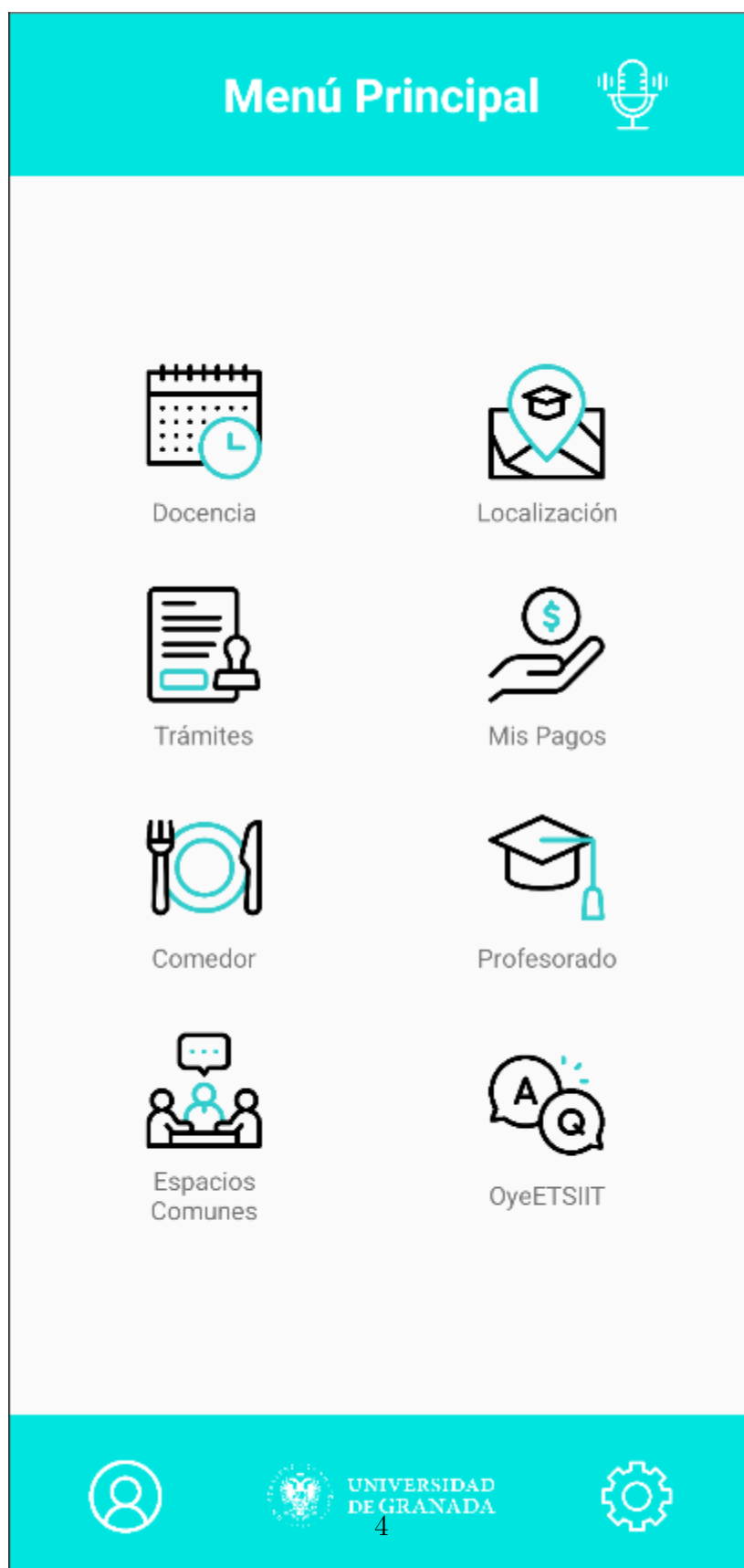


Figura 1: Menú principal de la aplicación.

Cada submenú está representado por un icono y un texto descriptivo. Las funcionalidades principales incluyen el acceso a información como el calendario académico, horarios de tutorías y servicios como el comedor. También permite localizar puntos de interés dentro de la escuela mediante un mapa interactivo y ofrece la posibilidad de interactuar con un chatbot para resolver dudas comunes. Además, incluye una barra superior con un botón de micrófono que activa la navegación por voz, diseñada para mejorar la accesibilidad.

El diseño del menú prioriza la simplicidad y la facilidad de uso, permitiendo que los usuarios encuentren rápidamente la información o funcionalidad que necesitan en su día a día dentro de la escuela.

2.1.1. Docencia con sensor NFC

Contiene información sobre horarios de asignaturas (esto variará en función del usuario, por lo que requiere autenticación), así como el calendario académico y de exámenes del curso actual. Por tanto, tenemos tres opciones:

- **Consulta de horarios:** donde tenemos los horarios disponibles gracias al sensor NFC.
- **Calendario académico:** con el calendario correspondiente.
- **Calendario de exámenes:** con la fecha de cada examen.

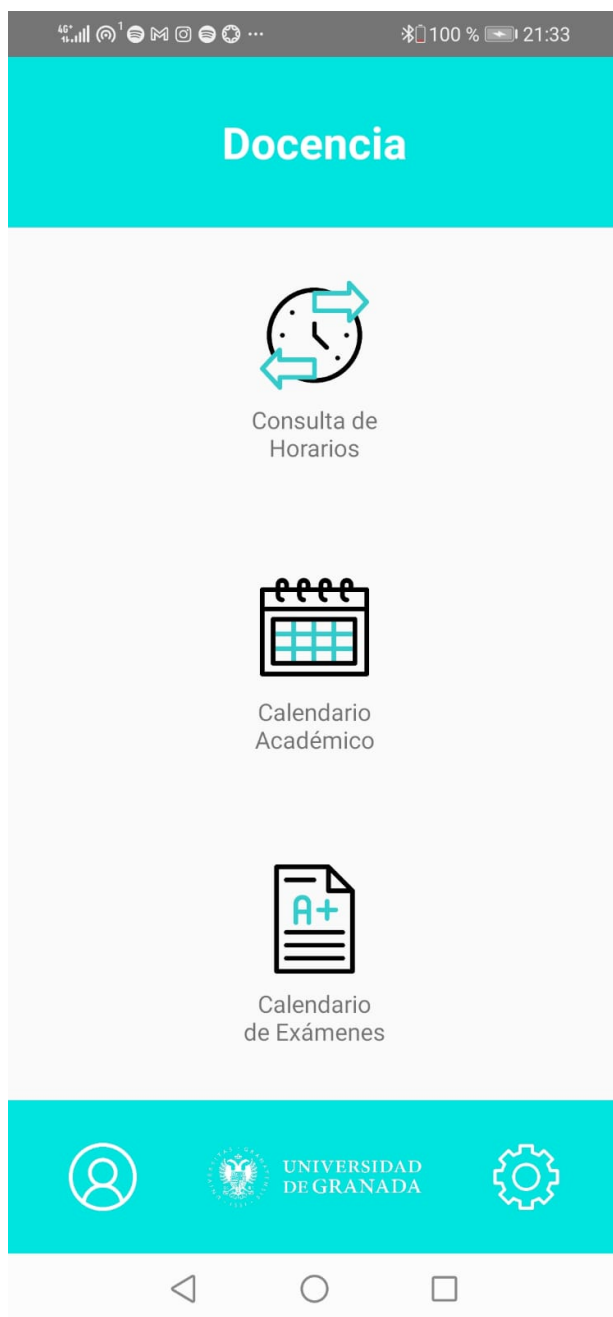


Figura 2: Submenú Docencia

Para poder implementar el sensor de NFC hemos utilizado los siguientes archivos:

- **Diseño de las vistas:** en la carpeta /res/layout hemos implementado nfc_loaded_classroom_layout.xml y nfc_loading_layout.xml

- **Código con la implementación del NFC:** script NFCActivity dentro de la carpeta: /kotlin+java/com.e

Cuando nos metemos en el submenú Docencia, en el apartado de Consulta de horarios, hemos implementado el sensor NFC para que nos indique en cada clase que horario hay. Nos sale la primera pantalla (nfc_loading_layout.xml):



Figura 3: Pantalla inicial para escanear NFC

En general, tenemos pensado que en cada clase haya un chip NFC, donde al detectarlo, nos dice que clases hay actualmente y que clases hay en la siguientes horas. Nosotros lo hemos implementando de una manera reducida: con nuestras tarjetas como si fuesen los chips de las diferentes clases ya que es lo que teníamos a disposición. Cuando pasamos una tarjeta que tenemos almacenada por su ID (como si fuese una clase), nos dirá el horario correspondiente. Tenemos implementadas tres tarjetas como si fuesen tres clases diferentes. Además, si pasas una tarjeta no registrada, te da aviso de que no está contemplada ese ID del NFC. Una vez detectada una tarjeta válida, dependiendo del horario y del día de la semana en que estemos saldrá las horas de la clase correspondiente (nfc_loaded_classroom_layout.xml):

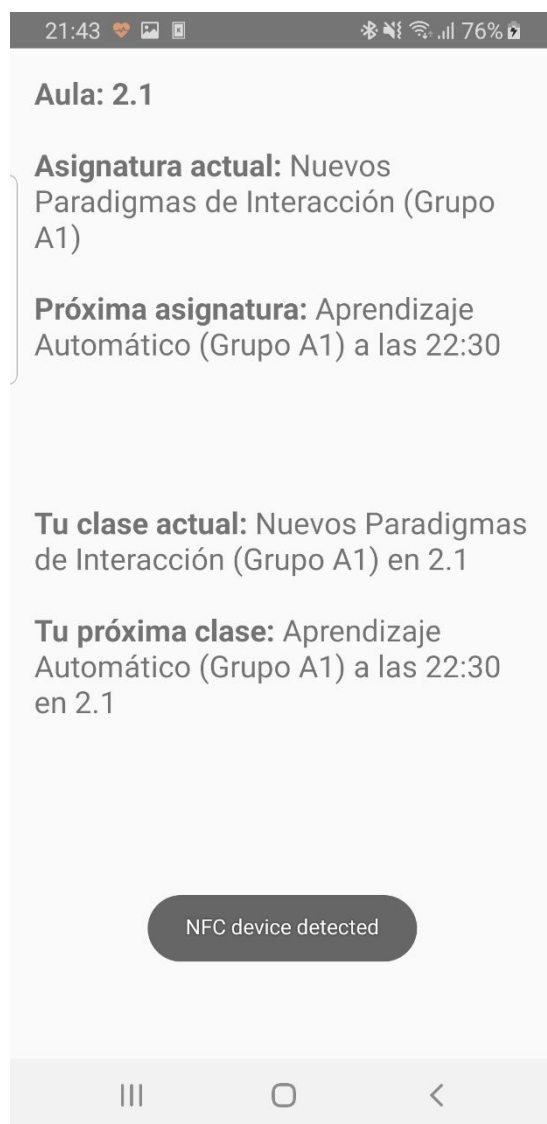


Figura 4: Consulta del horario de la clase

En cuanto al script con la funcionalidad del NFC tenemos el NFCActivity:

Primero, importamos las librerías necesarias para trabajar con NFC, Text-to-Speech (TTS), y otras funciones de Android. Por ejemplo: NfcAdapter y Tag para trabajar con dispositivos NFC, TextToSpeech para convertir texto a voz y JSONObject para procesar datos en formato JSON.

Las propiedades principales son:

- nfcAdapter: es el adaptador NFC que se utiliza para interactuar con etiquetas NFC.
- textToSpeech: controla el sistema de texto a voz.

En la función onCreate() cargamos el horario de clases desde un archivo JSON que lo tenemos guardado en la carpeta res/raw. A continuación, se inicializa el adaptador NFC y el sistema de texto a voz. Si el dispositivo no soporta NFC, se cierra la actividad.

En las funciones onResume() y onPause() se configuran el "Foreground Dispatch", que permite a la aplicación priorizar eventos NFC mientras está en primer plano.

En onDestroy(), se libera los recursos utilizados por el sistema de texto a voz para evitar fugas de memoria.

Ahora explicamos el manejo de eventos NFC:

En el método onNewIntent() se activa cuando se detecta una etiqueta NFC donde verifica el tipo de evento NFC, es decir, se comprueba si la acción es NfcAdapter.ACTION_TAG_DISCOVERED. A continuación se obtiene el ID del tag NFC, donde convierte los bytes del ID en un formato hexadecimal legible. Por último, se procesa el tag NFC, es decir, si el ID está permitido, llama a processNfcTag(hexId) y cambia la vista para mostrar información relacionada con el aula detectada.

En la función OnInit(), configuramos el idioma del sistema según la configuración regional del dispositivo. Si el idioma no es compatible, se registra un error.

Para la carga de datos, utilizamos archivos JSON que están organizados en dos estructuras:

- classroomTimetables: contiene horarios específicos para cada aula.
- personalTimetable: contiene horarios personales del usuario.

El método processNfcTag() analiza el tag NFC y determina la información que debe mostrarse. Primero, busca el aula asociada al ID del tag y si existe, obtiene el nombre del aula y el horario. Después, obtiene las clases del día actual donde busca la clase actual y la próxima basándose en la hora actual. A continuación, genera los mensajes del aula (indica qué clase hay ahora mismo y cual es la próxima) y verificamos si el usuario tiene clase ahora. Por último, muestra los mensajes en la

pantalla del móvil y los lee en voz alta por si alguna persona es ciega. Así incluimos también ayuda para personas con diversas funcionalidades.

El archivo que tiene la información necesaria están en la carpeta `/res/raw` y se llama `timetable.json`. Para poner mi ID de las tarjetas he puesto un log donde al acercarla me dice su ID. Así es muy fácil detectarla y cambiarla fácilmente. Además, en el este archivo está la información de los horarios correspondientes.

La aplicación incluye una funcionalidad que permite consultar el calendario académico y el calendario de exámenes directamente desde el submenú de docencia. Los documentos se muestran mediante un `WebView` que carga archivos PDF desde enlaces externos utilizando `Google Docs Viewer`. El visor está configurado para habilitar controles de zoom, lo que permite a los usuarios ampliar o reducir la vista según sus necesidades. Además, una barra de progreso informa sobre el estado de la carga del documento, garantizando una experiencia clara y sin interrupciones.

La implementación se lleva a cabo en la clase `CalendarioAcademicoActivity`, que utiliza un diseño definido en `calendario_academico.xml`. Este diseño incluye un encabezado con el título del calendario y un visor que ocupa el resto de la pantalla. La funcionalidad requiere permisos de internet, ya que los documentos se cargan desde una fuente externa, por lo que es necesario que el dispositivo esté conectado a una red. A continuación, se muestran capturas que ilustran tanto la visualización del calendario como la capacidad de zoom del visor:

■ Consulta de Calendarios

La aplicación incluye una funcionalidad que permite consultar el calendario académico y el calendario de exámenes directamente desde el submenú de docencia. Los documentos se muestran mediante un `WebView` que carga archivos PDF desde enlaces externos utilizando `Google Docs Viewer`. El visor está configurado para habilitar controles de zoom, lo que permite a los usuarios ampliar o reducir la vista según sus necesidades. Además, una barra de progreso informa sobre el estado de la carga del documento, garantizando una experiencia clara y sin interrupciones.

La implementación se lleva a cabo en la clase `CalendarioAcademicoActivity`, que utiliza un diseño definido en `calendario_academico.xml`. Este diseño incluye un encabezado con el título del calendario y un visor que ocupa el resto de la pantalla. De manera análoga se lleva a cabo para el calendario de exámenes. La funcionalidad requiere permisos de internet, ya que los documentos se cargan desde una fuente externa, por lo que es necesario que el dispositivo esté conectado a una red. A continuación, se muestran capturas que ilustran tanto la visualización del calendario como la capacidad de zoom del visor:



Figura 5: Visualización general del calendario académico.

2.1.2. Localización

2.1.3. Trámites

El submenú de trámites ofrece a los usuarios un acceso rápido a diferentes tipos de trámites a través de botones interactivos. Cada botón redirige a una nueva actividad relacionada con una carrera específica.



Figura 6: Zoom activado en el calendario académico.

Características:

- **Botones de Navegación:** Cuatro ImageButton están configurados para dirigir al usuario a distintas actividades correspondientes a áreas de trámites:
 - **InformaticaActivity:** Acceso a los trámites relacionados con el Grado en Informática.
 - **TelecomunicacionesActivity:** Acceso a trámites relacionados con el Grado en Telecomunicaciones.
 - **DGIIMActivity:** Acceso a trámites relacionados con el Doble Grado en Informática y Matemáticas.
 - **DGIIADEActivity:** Acceso a trámites relacionados con el Doble Grado en Informática y Administración y Dirección de Empresas.
- **Navegación:** Al hacer clic en cualquier botón, se inicia la actividad correspondiente mediante un Intent. Este comportamiento se implementa a través de listeners `setOnClickListener` asociados a cada botón.

Este diseño facilita una navegación rápida y sencilla entre diferentes tipos de trámites, mejorando la experiencia de usuario al ofrecer un acceso directo a la información relevante.

2.1.4. Mis pagos con sensor cámara para escanear QR

Se accede desde el menú principal pulsando sobre el submenú de Pagos UGR. Aquí tenemos cuatro opciones:

- **Escanear QR:** en este apartado se activa la cámara para poder escanear el QR correspondiente.
- **Canjear puntos:** una lista de cupones del comedor, deportes con distinto precio de descuento, camiseta de la UGR y sudadera de la UGR.
- **Historial de pagos:** lista de los pagos realizados
- **Mis bonos:** lista de los bonos disponibles para canjear

Tenemos un saldo como si fuese un monedero con 500 euros iniciales. Además, tenemos otro registro para los puntos ya que cada vez que pagas algo te dan puntos. Diseño del submenú mis pagos:

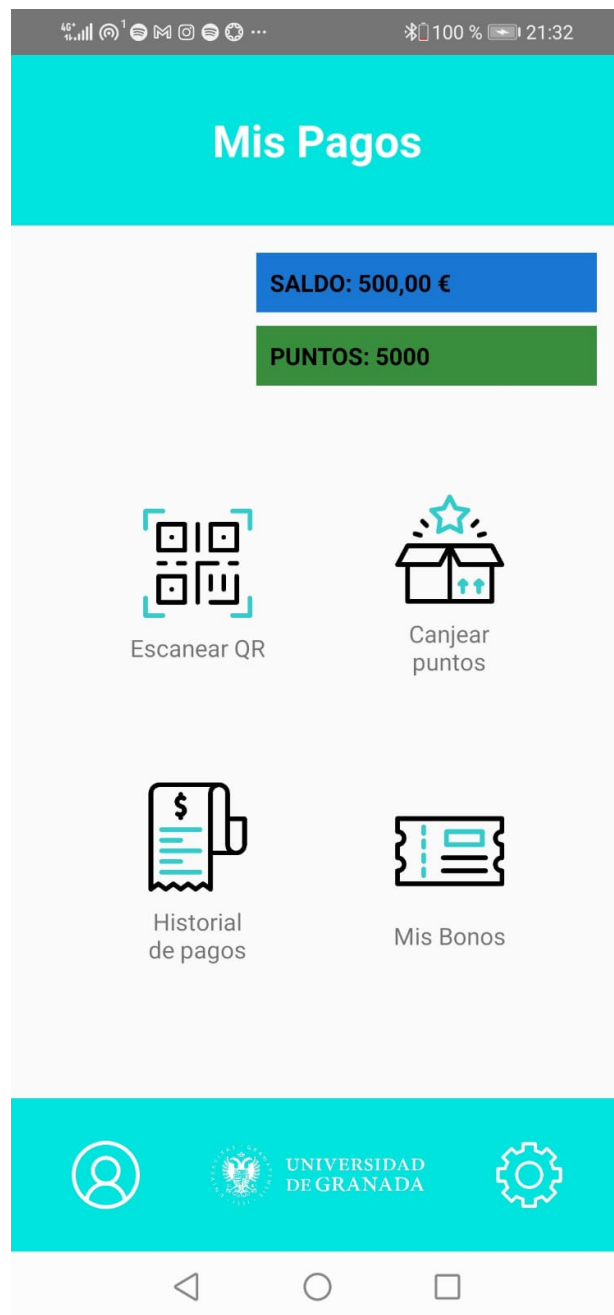


Figura 7: Submenú mis pagos

Tenemos los siguientes QR:



Figura 8: QR para el pago del comedor



Figura 9: QR para el pago de la matrícula



Figura 10: QR para el pago de otra matrícula



Figura 11: QR para el pago de una pista de pádel

En cuanto al tercer QR, al ser la matrícula de otro estudiante, no sería válido y no te deja pagar. Además, si se vuelve a escanear el cuarto QR, también da error ya que si pagas una pista a una hora y día predeterminado, no deja que vuelvas a comprarlo en ese mismo horario como es lógico. Por tanto, el único que puedes comprar muchas veces es el QR del comedor.

Cuando nos metemos en el apartado de Escanear QR, se nos activa la cámara para poder escanear. Una vez escaneado:

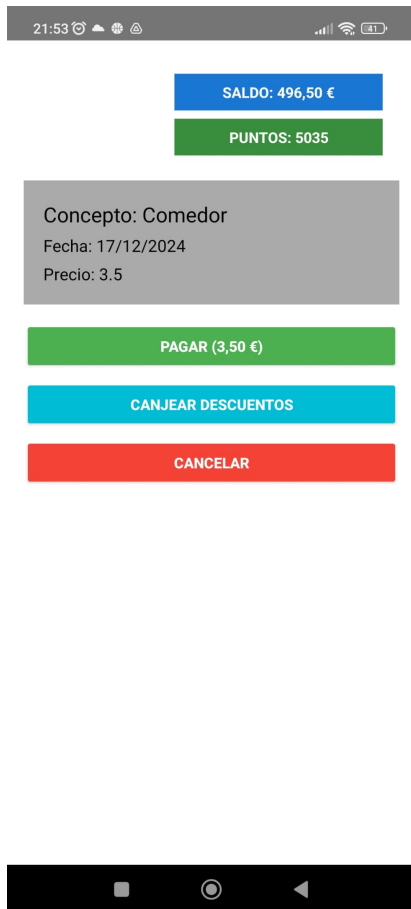


Figura 12: Pantalla al escanear un QR

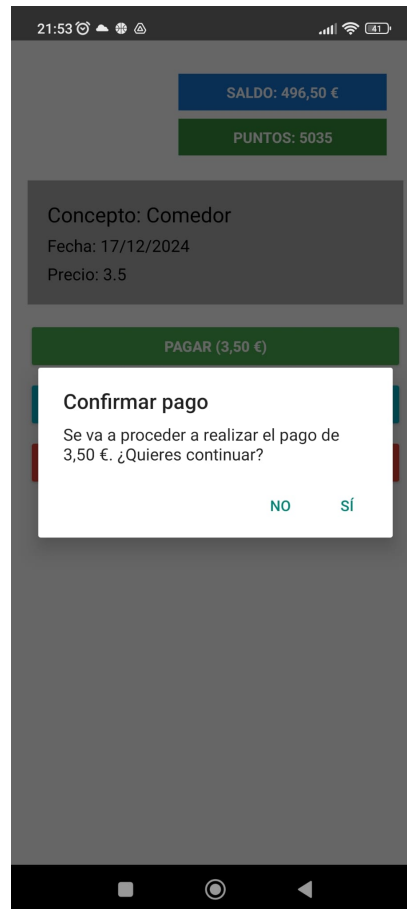


Figura 13: Pantalla al darle a pagar

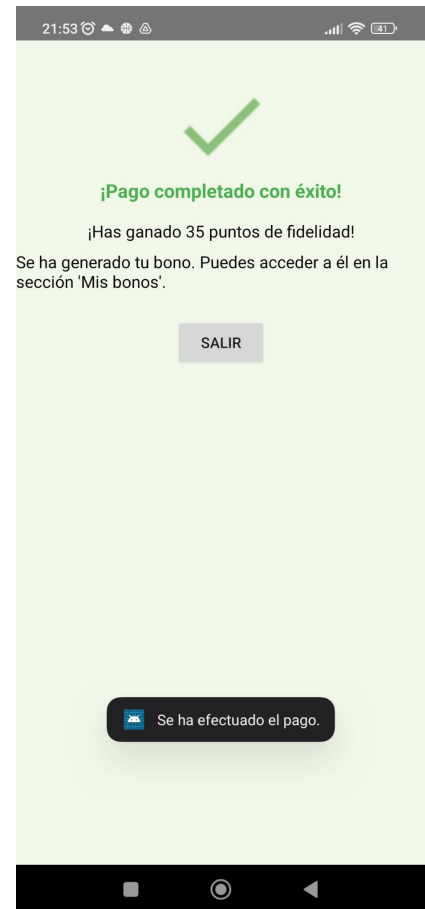


Figura 14: Pantalla una vez pagado

En el apartado de canjear puntos tenemos los siguientes descuentos:

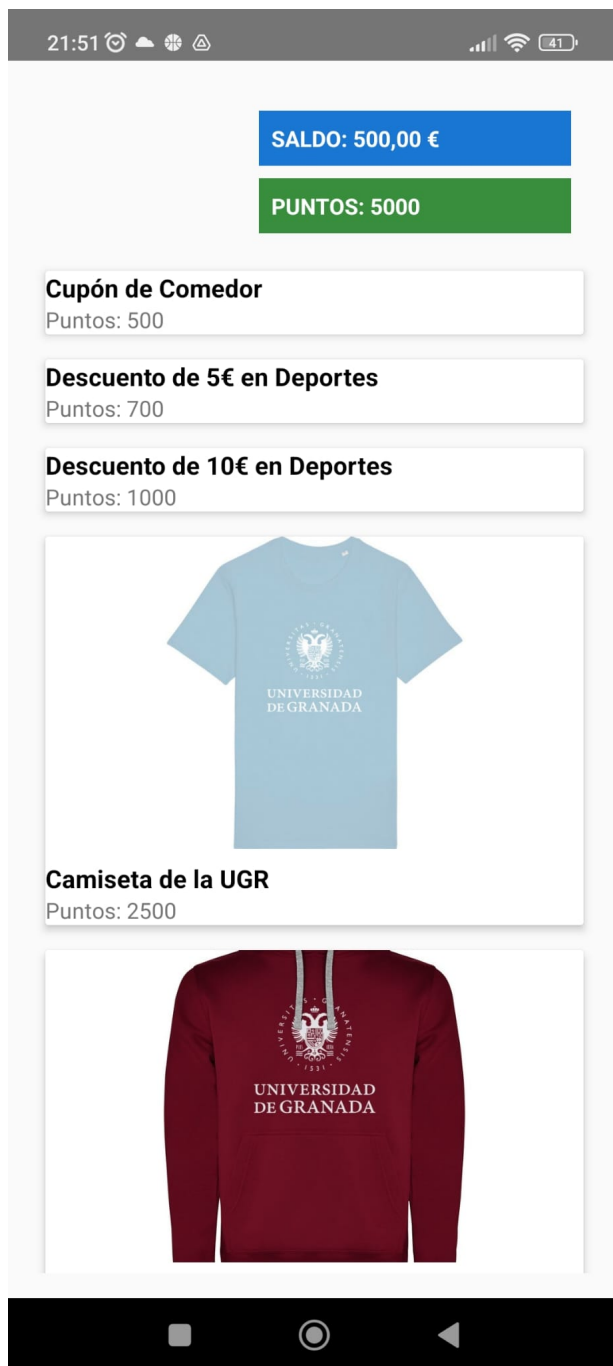


Figura 15: Canjear puntos

En los otros dos apartados tenemos:

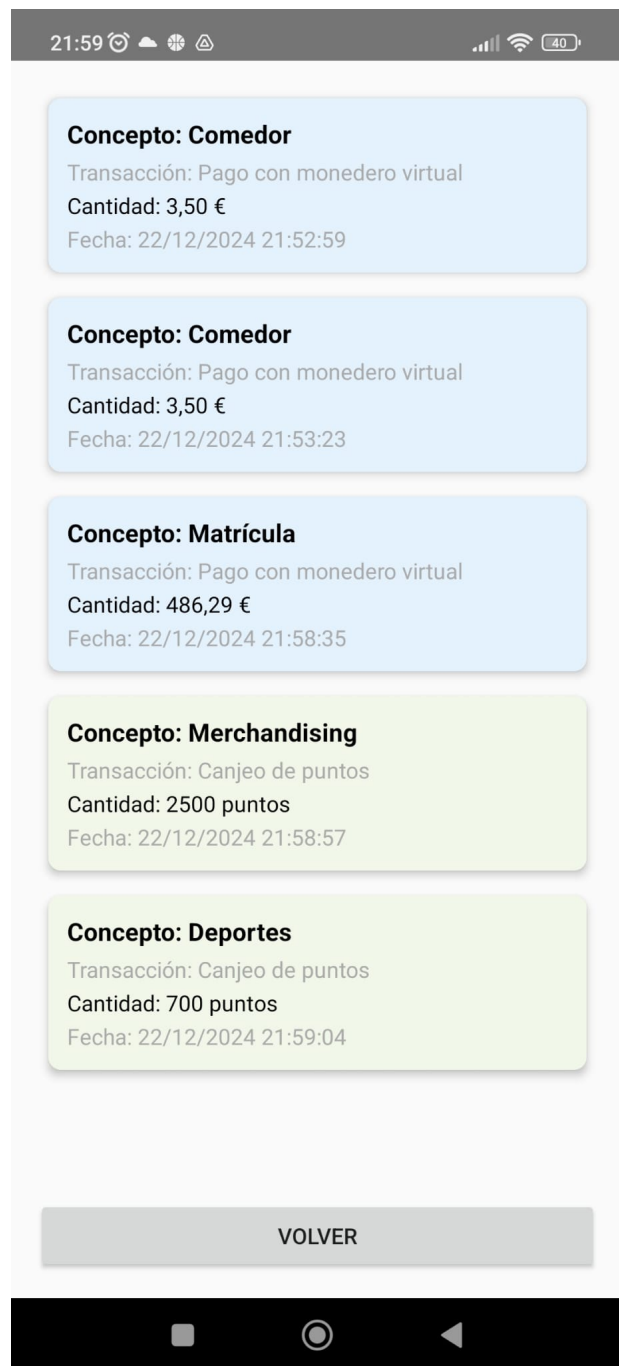


Figura 16: Pagos UGR

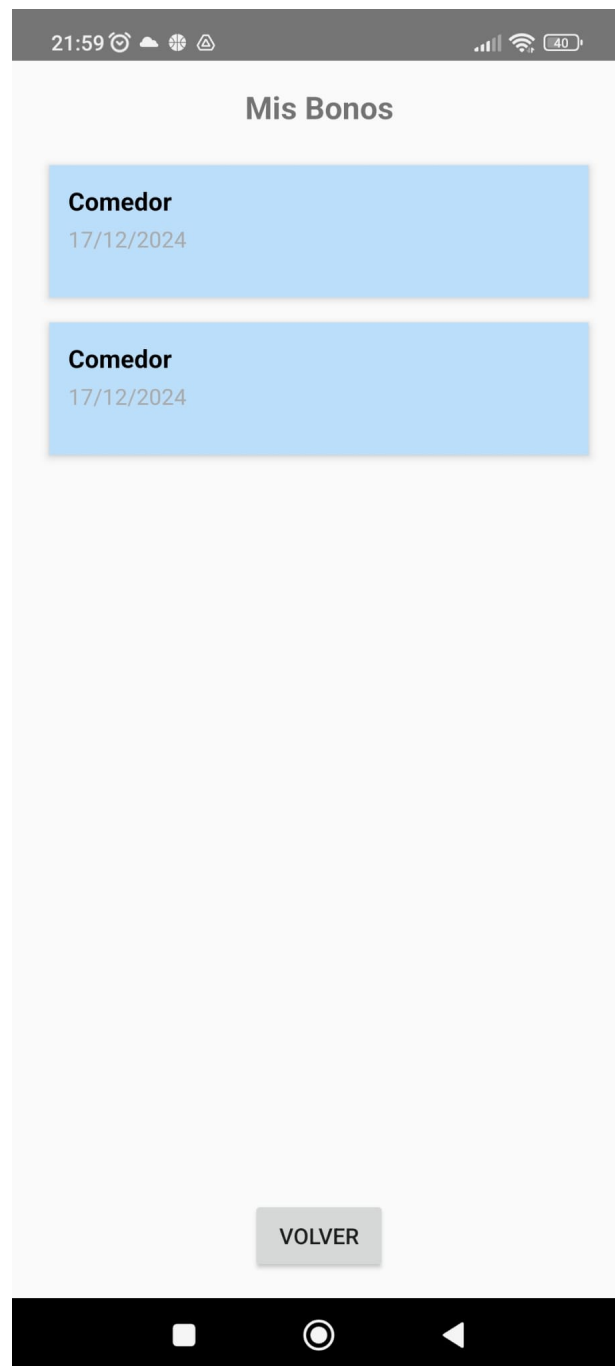


Figura 17: Mis bonos

Para poder configurar este sensor y crear la tienda, hemos creado muchas clases y hay una estructura complicada:

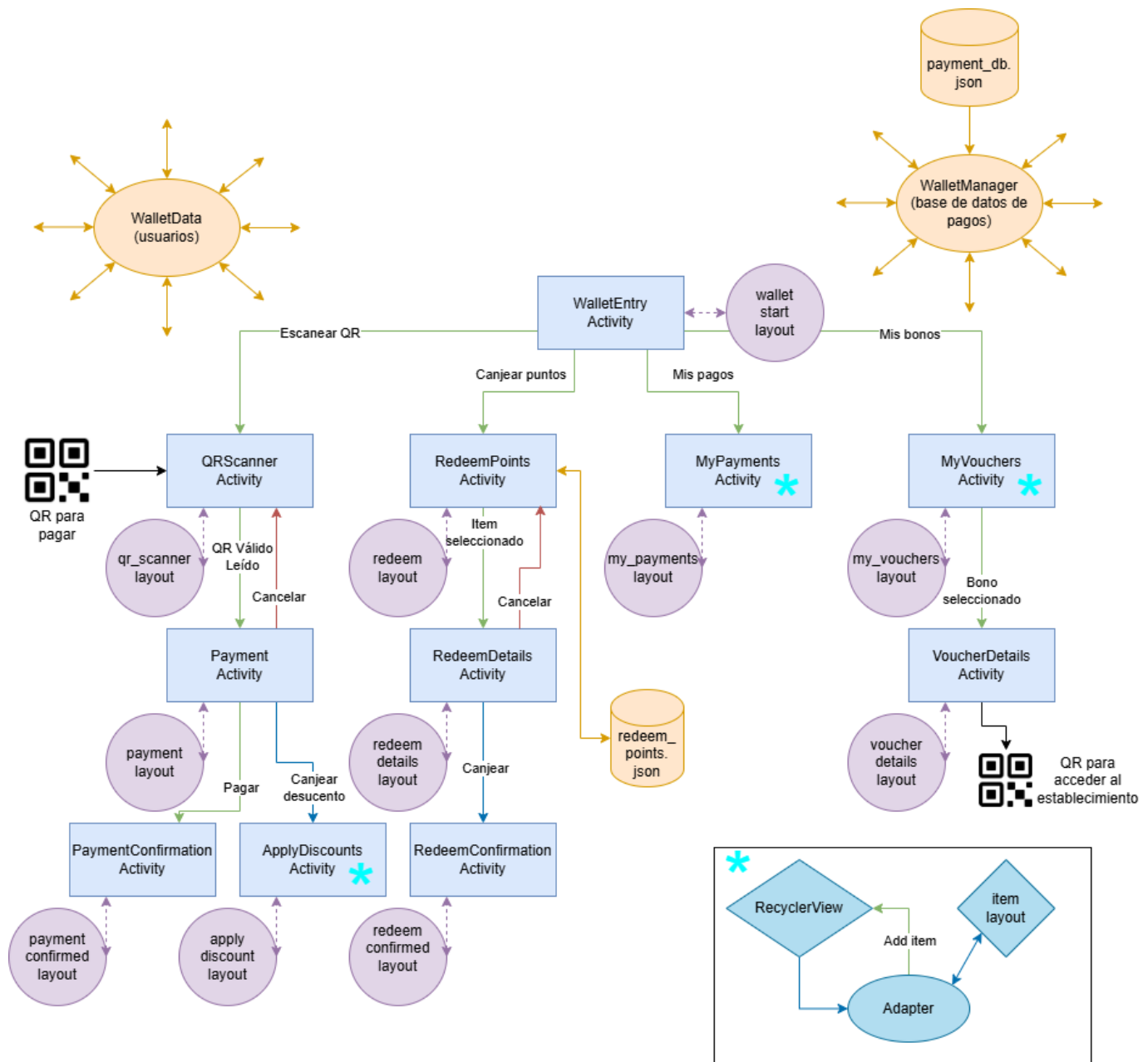


Figura 18: Clases para realizar el sensor del QR y la tienda

Vamos a analizar solo el fichero principal donde está la funcionalidad que es el script QRScannerActivity:

Primero, importamos todas las librería y dependencias que hacen falta. Algunas de ellas son: constantes de permisos para acceder a la cámara, para navegar entre actividades, para registrar mensajes de depuración, para mostrar mensajes al usuario, para mostrar la vista previa de la cámara, para escanear código...

Declaramos dos variables:

- `PreviewView`: renderiza la cámara en tiempo real
- `ExecutorService`: permite gestionar un grupo de hilos para realizar operaciones que no deben ejecutarse en el hilo principal de la aplicación como el procesamiento de imágenes.

A continuación, explicamos los diferentes métodos:

El `onCreate()` es el punto de entrada de la actividad. Configura la vista con el diseño `qr_scanner_layout`, inicializa el visor de la cámara (`PreviewView`) y el ejecutor de hilos (`cameraExecutor`). Además, verifica si los permisos de cámara están otorgados donde si están otorgados inicializa la cámara con `startCamera`, y si no lo están, solicita el permiso con `requestPermissionLauncher`.

El método de `requestPermissionLauncher()` maneja la solicitud de permisos de cámara. Si el usuario concede el permiso, inicia la cámara con `startCamera`, si lo deniega, muestra un mensaje informando que el permiso fue rechazado.

El método `allPermissionsGranted()`, comprueba si el permiso de cámara ha sido otorgado. Devuelve `true` si el permiso está otorgado y `false` en otro caso.

El método `startCamera()`, configura y enciende la cámara utilizando la API de `CameraX`. Además, crea un visor (`Preview`) que muestra la imagen de la cámara en el `PreviewView` y configura un caso de uso de análisis de imágenes (`ImageAnalysis`) para procesar los cuadros capturados por la cámara. Por último, vincula los casos de uso (visor y análisis) al ciclo de vida de la actividad.

En el método `handleQRCode()`, se procesa el contenido del código QR escaneado, se valida que el QR contenga un token. Si es válido y si el pago asociado al token es realizable se llama a `goToPaymentDetails` para continuar con el flujo de pago. Si no es válido, muestra un mensaje de error y detiene el procesamiento.

Con el método `goToPaymentDetails()`, navegamos a una nueva actividad (`PaymentActivity`) para mostrar los detalles del pago y envía los datos del pago como un extra en el `Intent`.

Para finalizar, con `onResume()` se ejecuta cuando la actividad vuelve a estar en primer plano y se restablece `isProcessingQR` a `false` para permitir nuevos escaneos de QR. Con `onDestroy()` se ejecuta cuando la actividad se destruye. Además, se cierra el ejecutor de hilos (`cameraExecutor`) para liberar recursos.

Resumiendo un poco, en este código se siguen los siguientes pasos:

1. La actividad solicita permisos de cámara al iniciarse.
2. Si los permisos son concedidos, se inicializa la cámara y se activa el análisis de imágenes en tiempo real.
3. Cuando se detecta un código QR, se valida su contenido y se decide si es válido para un pago.
4. Si el QR es válido, se navega a otra actividad para mostrar detalles del pago.
5. Se asegura que solo se procese un QR a la vez (`isProcessingQR`).
6. Los recursos de la cámara y los hilos se liberan al destruir la actividad.

2.1.5. Comedor

El submenú Comedor está diseñado para ofrecer a los usuarios acceso al menú del comedor a través de un archivo PDF cargado desde una URL externa. La actividad está implementada en la clase `ComedorActivity`, que extiende de `BaseActivity`, y se encarga de gestionar la interfaz de usuario y la carga dinámica del PDF dentro de un `WebView`. A continuación, se detalla su funcionamiento:

1. Estructura y Funcionalidad Principal

La actividad está estructurada para mostrar un documento PDF, que en este caso corresponde al menú del comedor. El archivo PDF se carga mediante un `WebView`, un componente de Android que permite la visualización de contenido web o archivos dentro de la aplicación. El `WebView` se configura para soportar zoom y JavaScript, lo que proporciona una mejor experiencia al usuario al interactuar con el documento.

El archivo PDF se carga desde la URL `https://scu.ugr.es/pages/menu/comedor?theme=pdf`. Esta URL devuelve el archivo en formato PDF, que es visualizado directamente en la aplicación sin necesidad de abrir una aplicación externa.

Un `ProgressBar` es utilizado para indicar el progreso de la carga del PDF. Mientras el archivo se carga (es decir, cuando el progreso está por debajo del 100 %), la `ProgressBar` se muestra. Una vez que el archivo ha terminado de cargarse, la `ProgressBar` desaparece.

2. Componentes Principales

- **WebView:** Es el elemento principal encargado de cargar el PDF. Se le asigna un `WebViewClient` para gestionar los enlaces dentro del documento y un `WebChromeClient` para controlar el progreso de carga.

- **ProgressBar:** Se utiliza para mostrar un indicador visual de que el PDF se está cargando. La visibilidad de la ProgressBar está vinculada al progreso de carga del WebView.
- **Configuración del WebView:** Se habilita JavaScript (`webView.settings.javaScriptEnabled = true`) y se permite el zoom sobre el documento con las opciones `setSupportZoom(true)` y `builtInZoomControls=true`.

3. Manejo de Errores Se ha implementado un bloque try-catch para capturar y registrar cualquier excepción que pueda ocurrir durante el proceso de carga del PDF. Si se produce un error (por ejemplo, si la URL no se puede cargar o hay problemas de conexión), el error se captura y se registra en el log de la aplicación, lo que facilita la depuración.

4. Interacción del Usuario Los usuarios pueden interactuar con el contenido del PDF utilizando los controles de zoom habilitados por el WebView, lo que permite un ajuste cómodo del tamaño del texto o la imagen del menú. Esto mejora la accesibilidad, permitiendo que el contenido sea fácilmente legible en diferentes tamaños de pantalla.

2.1.6. Profesorado

La funcionalidad de Profesorado en la aplicación tiene como objetivo proporcionar una interfaz donde los usuarios (estudiantes) puedan consultar información relevante sobre los profesores: su nombre, departamento y horarios de tutorías (tanto del primer semestre como del segundo).

La implementación de la actividad ProfesoradoActivity permite a los usuarios visualizar fácilmente la información a través de una interfaz interactiva y eficiente. La utilización del RecyclerView y un adaptador personalizado asegura un rendimiento fluido incluso con una gran cantidad de datos.

A continuación, se detallan los componentes clave del código.

- Estructura general

La actividad ProfesoradoActivity está compuesta por:

- **RecyclerView:** Para mostrar la lista de profesores.
- **ProfesorAdapter:** Un adaptador personalizado para vincular los datos de los profesores con los elementos visuales.
- **ProfesorViewHolder:** Un ViewHolder que contiene las vistas para el nombre, departamento y horarios de tutorías.

- Clase Profesor

La clase Profesor tiene los siguientes atributos:

- nombre: Nombre del profesor.
- departamento: Departamento al que pertenece el profesor.
- tutorias_primer_semestre: Horarios de tutorías del primer semestre.
- tutorias_segundo_semestre: Horarios de tutorías del segundo semestre.

- Implementación del Adaptador: ProfesorAdapter

El adaptador ProfesorAdapter se encarga de:

- Mostrar el nombre, departamento y horarios de tutorías en cada ítem de la lista.
- Inflar la vista de cada ítem utilizando el layout item_profesor.xml.
- Asignar los datos del objeto Profesor a las vistas correspondientes en cada ViewHolder.

```
83 class ProfesorAdapter(private val profesores: List<Profesor>) :  
84     RecyclerView.Adapter<ProfesorAdapter.ProfesorViewHolder>() {  
85  
86     class ProfesorViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
87         val nombre: TextView = view.findViewById(R.id.nombre)  
88         val departamento: TextView = view.findViewById(R.id.departamento)  
89         val telefono: TextView = view.findViewById(R.id.telefono)  
90         val correo: TextView = view.findViewById(R.id.correo)  
91     }  
92  
93     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ProfesorViewHolder {  
94         val view = LayoutInflater.from(parent.context)  
95             .inflate(R.layout.item_profesor, parent, attachToRoot: false)  
96         return ProfesorViewHolder(view)  
97     }  
98  
99     override fun onBindViewHolder(holder: ProfesorViewHolder, position: Int) {  
100         val profesor = profesores[position]  
101         holder.nombre.text = profesor.nombre  
102         holder.departamento.text = profesor.departamento  
103         holder.telefono.text = profesor.tutorias_primer_semestre  
104         holder.correo.text = profesor.tutorias_segundo_semestre  
105     }  
106  
107     override fun getItemCount(): Int {  
108         return profesores.size  
109     }  
110 }
```

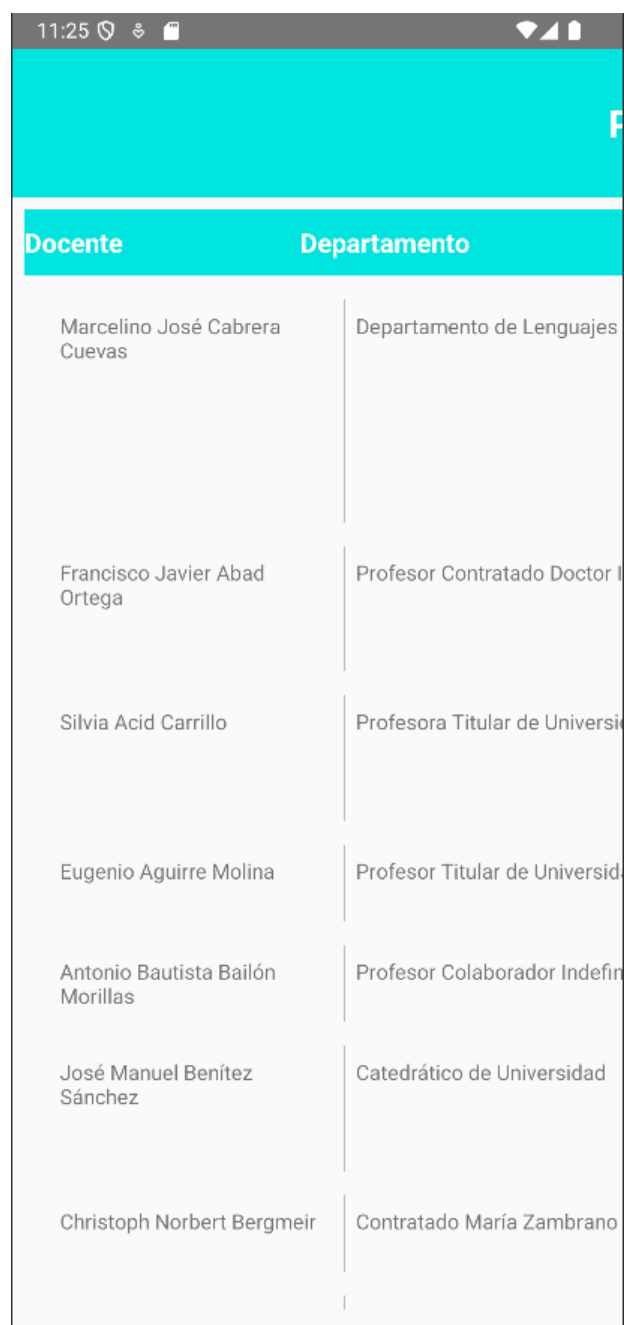
Figura 19: Implementación de la clase ProfesorAdapter

- Configuración del RecyclerView

En la actividad ProfesoradoActivity, se configura el RecyclerView con un LinearLayoutManager y el

adaptador `ProfessorAdapter`, el cual recibe una lista de objetos `Professor` como datos.

El submenú de profesorado permite consultar información sobre los docentes, incluyendo su nombre, departamento y horarios de tutorías para el primer y segundo semestre. El contenido se organiza en una tabla que requiere hacer scroll horizontal para visualizar toda la información. Esta disposición asegura que la interfaz sea clara y manejable, especialmente cuando se trata de una gran cantidad de datos.



Docente	Departamento
Marcelino José Cabrera Cuevas	Departamento de Lenguajes
Francisco Javier Abad Ortega	Profesor Contratado Doctor I
Silvia Acid Carrillo	Profesora Titular de Universi
Eugenio Aguirre Molina	Profesor Titular de Universid
Antonio Bautista Bailón Morillas	Profesor Colaborador Indefin
José Manuel Benítez Sánchez	Catedrático de Universidad
Christoph Norbert Bergmeir	Contratado María Zambrano

Figura 20: Vista inicial del submenú de profesorado.



Tutorías primer semestre	Tutorías segundo semestre
<p>Miércoles 11:00 - 14:00 D2F (Traducción e Interpretación)</p> <p>Jueves 9:00 - 10:30 D25</p> <p>Jueves 12:30 - 14:00 D25</p>	<p>Lunes 10:30 - 13:30 D25</p> <p>Martes 10:30 - 13:30 D25</p>
<p>Miércoles 9:30 - 13:30 D68-4P</p> <p>Miércoles 18:30 - 20:30 D68-4P</p>	<p>Jueves 9:30 - 12:30 D20</p>
<p>Lunes 10:00 - 12:00 D32</p> <p>Jueves 15:00 - 17:00 D32</p>	<p>Miércoles 11:00 - 13:00 D22</p>
<p>Martes 9:30 - 11:30 D45</p>	<p>Viernes 16:00 - 18:00 D20</p>
<p>Miércoles 8:30 - 10:30 D30</p>	<p>Jueves 14:00 - 16:00 D40</p>
<p>Martes 9:00 - 11:00 D44</p> <p>Viernes 13:00 - 15:00 D45</p>	<p>Jueves 12:00 - 14:00 D21</p>
<p>Jueves 10:30 - 12:30 D31</p>	<p>Lunes 11:00 - 13:00 D25</p>

Figura 21: Vista al hacer scroll hacia la derecha.

2.1.7. OyeETSIIT

La implementación de un chatbot en la aplicación es un ejemplo de uso de tecnologías modernas como Jetpack Compose y el reconocimiento de voz para mejorar la experiencia del usuario. La arquitectura sigue un patrón de separación de responsabilidades, con un ViewModel que gestiona la lógica del modelo y una actividad que maneja la interfaz. La integración con el modelo generativo permite proporcionar respuestas dinámicas y precisas a las preguntas de los usuarios, mientras que la gestión de voz agrega una capa adicional de accesibilidad y facilidad de uso.

El código se organiza principalmente en dos componentes: la actividad principal ChatBot, que se encarga de manejar la interfaz de usuario, y el ChatViewModel, que gestiona la lógica del modelo de inteligencia artificial y el envío de mensajes. A continuación, se detallan los componentes clave del código.

- ChatBot (Actividad Principal):

La clase ChatBot extiende BaseActivity e implementa el ciclo de vida de la actividad en Android. A través de esta actividad, los usuarios pueden interactuar con el chatbot mediante una interfaz gráfica que incluye un área de chat y un botón para activar el reconocimiento de voz.

- Reconocimiento de voz: La aplicación utiliza la API SpeechRecognizer de Android para activar el reconocimiento de voz. Cuando el usuario hace clic en el ícono del micrófono, se inicia la escucha de voz. Si el reconocimiento es exitoso, el texto transcrito se pasa al ChatViewModel para ser procesado por el modelo generativo. En caso de error, se muestran mensajes informativos.
- Gestión de permisos: Se verifica si la aplicación tiene permisos para acceder al micrófono. Si no se han concedido, se solicita al usuario el permiso correspondiente.
- Interfaz de usuario: Utiliza Jetpack Compose para crear una interfaz moderna y flexible. La pantalla muestra un área de mensajes, un campo de texto para introducir mensajes manualmente y un botón de micrófono para activar el reconocimiento de voz.

- ChatViewModel (Vista del Modelo):

El ChatViewModel es el encargado de gestionar el estado del chatbot, como la lista de mensajes y la interacción con el modelo generativo de inteligencia artificial.

- Manejo de mensajes: El messageList es una lista mutable que almacena los mensajes que se muestran en el chat. Esta lista contiene objetos MessageModel, que incluyen el mensaje y el rol (usuario o modelo). Cuando el usuario envía un mensaje, este se agrega a la lista, y luego el modelo generativo genera una respuesta.
- Generación de respuestas: Se utiliza la API GenerativeModel para interactuar con un modelo

de lenguaje (en este caso, el modelo "gemini-pro") que proporciona respuestas a las preguntas del usuario. Se utiliza la función `startChat()` para iniciar una conversación con el modelo, y el método `sendMessage()` para enviar la pregunta y recibir la respuesta generada. Si ocurre un error en la generación de la respuesta, se agrega un mensaje de error a la lista.

- Manejo de excepciones: En caso de error (por ejemplo, falta de conexión a Internet o fallo al comunicarse con el modelo), el `ChatViewModel` maneja la excepción y agrega un mensaje de error informativo a la lista de mensajes.

- Interfaz de Usuario con Jetpack Compose

La interfaz de usuario se construye utilizando Jetpack Compose, lo que permite una mayor flexibilidad y un diseño más moderno. Se detallan los siguientes componentes clave:

- Mensaje de texto: El componente `MessageRow` se utiliza para mostrar cada mensaje en el chat. Se distingue entre los mensajes enviados por el usuario y las respuestas del modelo mediante el cambio de alineación y colores de los cuadros de texto.
- Campo de entrada de mensaje: El `MessageInput` es un campo de texto donde el usuario puede escribir su mensaje. Además, se proporciona un botón para enviar el mensaje manualmente y otro para activar el micrófono.
- Lista de mensajes: La función `MessageList` utiliza un `LazyColumn` para mostrar los mensajes de forma eficiente. La lista se muestra en orden invertido para que los mensajes más recientes aparezcan al final.
- Encabezado de la aplicación: La función `AppHeader` muestra un título en la parte superior de la interfaz con el nombre de la aplicación ("OyeETSIIT").

- Funcionalidades de Voz

El sistema de reconocimiento de voz está basado en la API `SpeechRecognizer` de Android. Cuando el usuario hace clic en el ícono del micrófono, la aplicación comienza a escuchar. Si se detecta un mensaje de voz, se convierte a texto y se envía al modelo generativo para obtener una respuesta. El texto transcrito se muestra como un mensaje en el chat.

El estado de la escucha se maneja a través de la variable `isListening`, que controla si el micrófono está activo o no. Si el micrófono está activado, se muestra un mensaje de `.escuchando` se espera la entrada del usuario.

- Comunicación con el Modelo Generativo El modelo generativo utilizado en esta aplicación es el "gemini-pro", que se comunica a través de una API. La clase `GenerativeModel` se encarga de realizar la interacción con el modelo, enviando el texto de la conversación y recibiendo la respuesta

generada.

2.1.8. Gestión del login

La gestión del login en la aplicación es una funcionalidad clave para garantizar que solo los usuarios autorizados puedan acceder a las funcionalidades principales. La actividad de login permite la autenticación mediante un formulario sencillo con campos para ingresar un nombre de usuario y una contraseña. A continuación, se detalla el proceso técnico de esta gestión.

1. Estructura y Funcionalidad Principal

La actividad de login está implementada en la clase LoginActivity, la cual maneja la interfaz de usuario y la validación de credenciales.

- Interfaz de Usuario: El diseño de la pantalla de login se basa en un RelativeLayout, que tiene un fondo con el logo de la aplicación y un diseño centrado utilizando un CardView que contiene los siguientes elementos:
 - Campo de Usuario (EditText): Permite al usuario ingresar su nombre de usuario. Este campo está configurado para aceptar texto normal.
 - Campo de Contraseña (EditText): Permite ingresar la contraseña de manera oculta (caracteres enmascarados). Está configurado para evitar la visualización de texto y proteger la privacidad.
 - Botón de Login (Button): Este botón, al ser presionado, activa la validación de las credenciales. Si las credenciales son correctas, se permite el acceso; de lo contrario, se muestra un mensaje de error.
 - Opciones Adicionales:
 - CheckBox: Una opción para Recordar.^{a1} usuario, aunque no está implementada en esta versión de la aplicación.
 - Texto de "He olvidado mi contraseña": Enlace para recuperar la contraseña, que no tiene funcionalidad activa en esta implementación.

El diseño es sencillo, con un enfoque en la accesibilidad y la claridad, y un fondo translúcido para mantener el enfoque en los campos de entrada.

2. Validación de Credenciales

La validación de las credenciales del usuario se realiza a través del método validateCredentials(), que recupera los valores ingresados en los campos de nombre de usuario y contraseña:

- **Usuario y Contraseña Predefinidos:** En este caso, la validación está diseñada para aceptar un nombre de usuario `userz` una contraseña `"1234"`. Estos valores son comparados con los datos introducidos en los campos correspondientes.
- Si las credenciales coinciden con los valores predefinidos, la aplicación marca al usuario como autenticado mediante la variable estática `DownToolBar.login = true`. Esta variable es utilizada en otras partes de la aplicación para verificar el estado del login.

3. Flujo de Navegación y Acciones

- **Acceso Exitoso:** Si las credenciales son correctas, el usuario es redirigido a la actividad principal de la aplicación (`MainActivity`) mediante un `Intent`. Además, la actividad de login se cierra utilizando `finish()`, lo que asegura que el usuario no pueda volver a la pantalla de login mediante la acción de retroceso. Al volver a `MainActivity` el icono de login habrá cambiado a la imagen del perfil del usuario.
- **Acceso Fallido:** En caso de que las credenciales sean incorrectas, se muestra un mensaje de error utilizando un `Toast`, que informa al usuario que el nombre de usuario o la contraseña son incorrectos. El usuario puede intentar nuevamente.

4. Elementos de Diseño

El archivo `activity_login.xml` define el diseño visual de la pantalla de login. Utiliza un `RelativeLayout` como contenedor principal, sobre el cual se posicionan los siguientes elementos:

- Un fondo con el logo de la aplicación (`android:background="@drawable/logo_etsiit"`).
- Un `CardView` que contiene los campos de entrada y el botón de login. Este `CardView` tiene bordes redondeados (`cardCornerRadius="28dp"`) para un aspecto más moderno y visualmente atractivo.
- `TextViews` para las instrucciones y etiquetas.
- `EditText` para ingresar el nombre de usuario y la contraseña, con texto de ayuda (`hint`) para guiar al usuario.
- Un `Button` para el login, que activa el proceso de validación.

5. Flujo de Interacción con el Usuario

1. El usuario abre la pantalla de login.
2. Ingresa su nombre de usuario y contraseña en los campos correspondientes.

3. Presiona el botón de login.

- Si las credenciales son correctas, se inicia la actividad principal y se cierra la pantalla de login.
- Si las credenciales son incorrectas, se muestra un mensaje de error y el usuario puede volver a intentar el acceso.

6. Seguridad y Consideraciones Futuras

Actualmente, el sistema de validación de credenciales está basado en valores predefinidos. Sin embargo, en una versión futura, este proceso debería conectarse a un sistema de autenticación más seguro, como una base de datos o un servicio de autenticación en línea (por ejemplo, Firebase Authentication), para manejar credenciales de usuarios reales y garantizar una mayor seguridad en el acceso.

En resumen, la gestión del login proporciona una forma simple y eficaz de controlar el acceso a la aplicación, con una interfaz de usuario amigable y una validación de credenciales básica. Este flujo es adecuado para una versión inicial de la aplicación y puede ser ampliado en futuras actualizaciones para mejorar la seguridad y la funcionalidad.

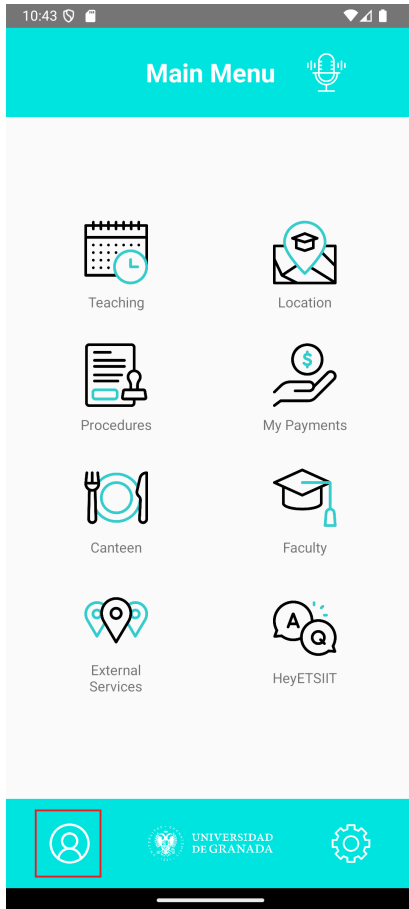


Figura 22: Main Activity antes de un login exitoso



Figura 23: Pantalla de login

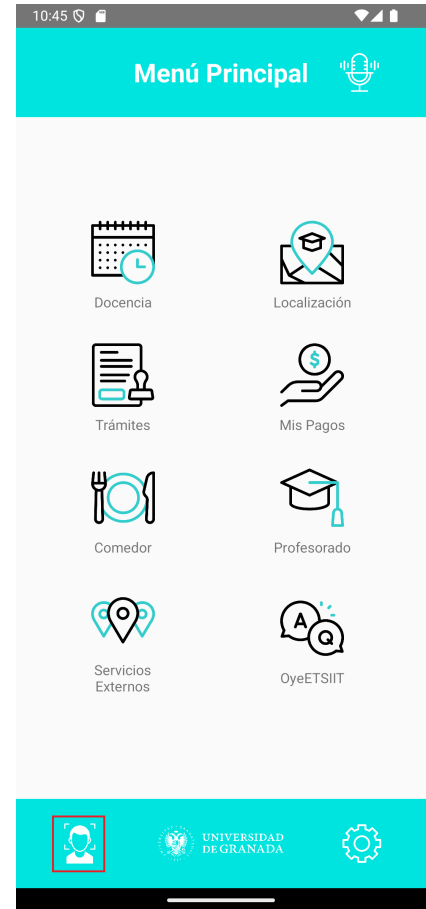


Figura 24: Main Activity después de un login exitoso

2.1.9. Soporte multilingüe

La aplicación soporta los idiomas español e inglés, ofreciendo a los usuarios la posibilidad de seleccionar su idioma preferido desde un botón de ajustes ubicado en la barra inferior. Este botón, situado a la derecha, está presente en la mayoría de las pantallas de la aplicación, lo que facilita el acceso rápido a esta configuración. Al pulsarlo, se muestra una ventana emergente con las opciones de idioma disponibles, y una vez seleccionado, los textos de la interfaz se actualizan automáticamente.

Para gestionar las traducciones, la aplicación utiliza un sistema basado en los archivos `strings.xml`, localizados en la carpeta `res/values/`. Estos archivos contienen las cadenas de texto para cada idioma soportado, con un archivo dedicado a cada uno (por ejemplo, `strings.xml` para español y `strings.xml (en)` para inglés). Este enfoque permite centralizar y organizar todas las cadenas traducibles, asegurando consistencia en aquellas partes de la interfaz que han sido traducidas y facilitando futuras modificaciones o ampliaciones de idiomas.

Algunas funcionalidades, como el chatbot y los comandos de voz, solo están disponibles en español. Esto se debe a dependencias técnicas del sistema de reconocimiento de voz de Android y del diseño interno del chatbot. Este comportamiento está controlado para evitar errores inesperados, pero puede limitar la experiencia para usuarios que prefieran interactuar en inglés. Además, no todos los elementos visuales de la aplicación han sido traducidos aún, lo que puede generar inconsistencias en ciertas pantallas.

El cambio de idioma se implementa mediante el método `setLocale()`, que actualiza la configuración del idioma en el dispositivo y reinicia la actividad actual para aplicar los cambios. La clase `DownToolBar`, responsable de gestionar la barra inferior, incluye el botón de ajustes que llama al método `showLanguageDialog()` para mostrar las opciones de idioma y aplicar la selección del usuario. Si bien el sistema funciona correctamente, en ocasiones puede requerir interacciones adicionales para reflejar los cambios, lo que representa un área de mejora.

En general, esta funcionalidad representa un paso importante hacia la accesibilidad y la personalización. Aunque las limitaciones actuales restringen algunas experiencias, el diseño actual ofrece una base sólida para futuras ampliaciones del soporte multilingüe.

3. Sensor NFC

Para poder implementar el sensor de NFC hemos utilizado los siguientes archivos:

- **Diseño de las vistas:** en la carpeta `/res/layout` hemos implementado `nfc_loaded.classroom.layout.xml` y `nfc_loading.layout.xml`
- **Código con la implementación del NFC:** script `NFCActivity` dentro de la carpeta: `/kotlin+java/com.e`

Cuando nos metemos en el submenú Docencia, en el apartado de Consulta de horarios, hemos implementado el sensor NFC para que nos indique en cada clase que horario hay. Nos sale la primera pantalla (`nfc_loading.layout.xml`):



Figura 25: Pantalla inicial para escanear NFC

En general, tenemos pensado que en cada clase haya un chip NFC, donde al detectarlo, nos dice que clases hay actualmente y que clases hay en la siguientes horas. Nosotros lo hemos implementando de una manera reducida: con nuestras tarjetas como si fuesen los chips de las diferentes clases ya que es lo que teníamos a disposición. Cuando pasamos una tarjeta que tenemos almacenada por su ID (como si fuese una clase), nos dirá el horario correspondiente. Tenemos implementadas tres tarjetas como si fuesen tres clases diferentes. Además, si pasas una tarjeta no registrada, te da aviso de que no está contemplada ese ID del NFC. Una vez detectada una tarjeta válida, dependiendo del horario y del día de la semana en que estemos saldrá las horas de la clase correspondiente (`nfc_loaded_classroom_layout.xml`):

METER SEGUNDA PANTALLA

En cuanto al script con la funcionalidad del NFC tenemos el `NFCActivity`:

Primero, importamos las librerías necesarias para trabajar con NFC, Text-to-Speech (TTS), y otras

funciones de Android. Por ejemplo: `NfcAdapter` y `Tag` para trabajar con dispositivos NFC, `TextToSpeech` para convertir texto a voz y `JSONObject` para procesar datos en formato JSON.

Las propiedades principales son:

- `nfcAdapter`: es el adaptador NFC que se utiliza para interactuar con etiquetas NFC.
- `textToSpeech`: controla el sistema de texto a voz.

En la función `onCreate()` cargamos el horario de clases desde un archivo JSON que lo tenemos guardado en la carpeta `res/raw`. A continuación, se inicializa el adaptador NFC y el sistema de texto a voz. Si el dispositivo no soporta NFC, se cierra la actividad.

En las funciones `onResume()` y `onPause()` se configuran el "Foreground Dispatch", que permite a la aplicación priorizar eventos NFC mientras está en primer plano.

En `onDestroy()`, se libera los recursos utilizados por el sistema de texto a voz para evitar fugas de memoria.

Ahora explicamos el manejo de eventos NFC:

En el método `onNewIntent()` se activa cuando se detecta una etiqueta NFC donde verifica el tipo de evento NFC, es decir, se comprueba si la acción es `NfcAdapter.ACTION_TAG_DISCOVERED`. A continuación se obtiene el ID del tag NFC, donde convierte los bytes del ID en un formato hexadecimal legible. Por último, se procesa el tag NFC, es decir, si el ID está permitido, llama a `processNfcTag(hexId)` y cambia la vista para mostrar información relacionada con el aula detectada.

En la función `OnInit()`, configuramos el idioma del sistema según la configuración regional del dispositivo. Si el idioma no es compatible, se registra un error.

Para la carga de datos, utilizamos archivos JSON que están organizados en dos estructuras:

- `classroomTimetables`: contiene horarios específicos para cada aula.
- `personalTimetable`: contiene horarios personales del usuario.

El método `processNfcTag()` analiza el tag NFC y determina la información que debe mostrarse. Primero, busca el aula asociada al ID del tag y si existe, obtiene el nombre del aula y el horario. Después, obtiene las clases del día actual donde busca la clase actual y la próxima basándose en la hora actual. A continuación, genera los mensajes del aula (indica qué clase hay ahora mismo y cual es la próxima) y verificamos si el usuario tiene clase ahora. Por último, muestra los mensajes en la pantalla del móvil y los lee en voz alta por si alguna persona es ciega. Así incluimos también ayuda para personas con diversas funcionalidades.

El archivo que tiene la información necesaria están en la carpeta /res/raw y se llama timetable.json. Para poner mi ID de las tarjetas he puesto un log donde al acercarla me dice su ID. Así es muy fácil detectarla y cambiarla fácilmente. Además, en el este archivo está la información de los horarios correspondientes.

4. Sensor multitáctil

La aplicación implementa una funcionalidad basada en el sensor multitáctil para facilitar la navegación. Específicamente, el usuario puede regresar a la pantalla principal (**MainActivity**) realizando un gesto intuitivo: deslizar dos dedos simultáneamente de izquierda a derecha en la pantalla. Esta característica combina naturalidad y eficiencia, mejorando la experiencia de uso y simplificando la interacción con la aplicación.

4.1. Diseño e Implementación Técnica

La funcionalidad se ha desarrollado en la clase base **BaseActivity**, que sirve como punto de partida común para las demás actividades de la aplicación. El sensor multitáctil se integra mediante un **OnTouchListener** asociado al diseño principal de la actividad, permitiendo la captura y el manejo de eventos táctiles en tiempo real.

El sistema sigue un flujo bien definido para detectar y procesar el gesto:

1. **Captura inicial:** Cuando el usuario toca la pantalla con al menos dos dedos, el listener registra las posiciones iniciales de ambos dedos en el eje X. Estas posiciones se almacenan en un arreglo para su posterior análisis.
2. **Seguimiento del movimiento:** Mientras los dedos permanecen en la pantalla, el sistema monitorea sus desplazamientos horizontales. El cálculo de este movimiento compara las posiciones actuales de los dedos con las posiciones iniciales registradas.
3. **Detección del gesto:** Si el desplazamiento de ambos dedos supera un umbral predefinido (800 píxeles en el eje X), el sistema interpreta que el gesto ha sido completado con éxito.
4. **Prevención de múltiples activaciones:** Para evitar que el gesto se registre más de una vez durante la misma interacción, se emplea una bandera lógica (**isActivityStarted**). Esta variable asegura que la acción solo se ejecute una vez, incluso si el gesto persiste.
5. **Acción desencadenada:** Una vez detectado el gesto, el sistema lanza un **Intent** para redirigir al usuario a la pantalla principal (**MainActivity**). Este **Intent** utiliza una configuración especial para limpiar cualquier actividad previa en la pila de navegación, asegurando que el usuario comience de nuevo desde el menú principal.

6. **Reinicio del estado:** Tras completar la acción, las posiciones iniciales de los dedos y la bandera lógica se reinician para preparar el sistema para futuros gestos.

La implementación incluye diversas medidas para garantizar la robustez y prevenir errores, como reiniciar el estado cuando los dedos se levantan de la pantalla o manejar adecuadamente situaciones en las que el gesto no se completa.

4.2. Ventajas de la Funcionalidad

La inclusión de este gesto multitáctil aporta numerosos beneficios:

- **Interacción Natural:** El gesto de deslizar dos dedos simultáneamente es intuitivo y fácil de recordar, proporcionando una experiencia de usuario fluida.
- **Ahorro de Tiempo:** Los usuarios pueden regresar rápidamente al menú principal sin necesidad de buscar botones específicos o navegar por varias pantallas.
- **Accesibilidad:** Este método es especialmente útil para personas con dificultades para interactuar con botones pequeños o precisos, mejorando la accesibilidad general de la aplicación.
- **Prevención de Errores:** La implementación asegura que el gesto sea lo suficientemente distintivo como para evitar activaciones accidentales.

4.3. Pruebas y Validación

Se llevaron a cabo pruebas exhaustivas para verificar la funcionalidad del sensor multitáctil en diferentes escenarios:

- Pruebas en dispositivos con distintas resoluciones de pantalla para garantizar la consistencia del umbral de desplazamiento.
- Simulación de gestos incompletos o incorrectos para confirmar que no activan la funcionalidad.
- Validación del manejo de errores, como el levantamiento prematuro de uno o ambos dedos.

Los resultados demostraron que la funcionalidad cumple con los objetivos de diseño, ofreciendo una experiencia confiable y libre de errores.

4.4. Conclusión

El uso del sensor multitáctil para la navegación dentro de la aplicación representa un avance significativo en términos de usabilidad y accesibilidad. Al proporcionar una interacción intuitiva y eficiente,

esta funcionalidad mejora la experiencia general del usuario y refuerza el compromiso de la aplicación con los principios de diseño inclusivo y moderno.

5. Uso del Sensor GPS

La aplicación utiliza el sensor GPS para ofrecer funcionalidades de localización dentro de la ETSIIT. Permite a los usuarios acceder desde la opción "Localización" del menú principal para ver puntos de interés en el mapa, como la biblioteca, cafetería o aulas, y trazar rutas hacia ellos si están dentro del campus. La lógica de la navegación y los puntos de interés se gestiona directamente en la aplicación, sin depender de servicios externos como Google Maps para el trazado de rutas.

5.1. Funcionalidad General

En el mapa, los puntos de interés aparecen marcados con iconos. Si el usuario pulsa sobre uno de estos marcadores, se muestra el nombre del lugar y, si está dentro del campus, también la opción de trazar una ruta hacia él. Además, la aplicación proporciona información adicional, como la planta donde se encuentra el destino, para que el usuario pueda decidir si necesita usar escaleras o ascensor. Si el usuario está fuera de los límites del campus, la aplicación muestra un mensaje indicándolo.

5.2. Implementación Técnica

Cálculo del área de la facultad: La ETSIIT se define como un área triangular delimitada por tres coordenadas GPS que representan sus vértices. Para determinar si el usuario está dentro de este triángulo, se utiliza un algoritmo basado en el método de intersección de bordes. Este algoritmo cuenta el número de veces que un rayo horizontal, trazado desde el punto del usuario, cruza los bordes del triángulo. Si el número de intersecciones es impar, el punto está dentro del área.

Puntos de interés y conexiones: Los puntos de interés, como la biblioteca o los despachos, están representados como nodos con coordenadas GPS, un nombre y la planta en la que se encuentran. Estos nodos están interconectados para formar una red de navegación que refleja la disposición real de pasillos y accesos en el edificio.

Gestión del mapa: La interfaz del mapa utiliza la API de Google Maps para mostrar los nodos como marcadores y permitir la interacción con ellos. Cuando el usuario selecciona un marcador, el sistema verifica si está dentro de los límites del campus antes de habilitar el trazado de rutas.

Cálculo y trazado de rutas: Si el usuario está dentro del área definida, la aplicación calcula la ruta más corta entre su ubicación actual y el punto de destino utilizando un algoritmo de búsqueda en grafos. La ruta se representa en el mapa mediante una línea roja que conecta los nodos de paso.

5.3. Ventajas del Enfoque Implementado

- **Independencia de servicios externos:** La lógica del trazado de rutas y los puntos de interés está integrada en la aplicación, evitando la dependencia de servicios como Google Maps.
- **Información detallada:** Además del trazado de rutas, la aplicación proporciona información adicional como la planta donde se encuentra el destino, mejorando la orientación del usuario.
- **Precisión:** El cálculo del área triangular garantiza una detección precisa de si el usuario está dentro del campus.
- **Personalización:** Los puntos de interés están adaptados específicamente a la ETSIIT, ofreciendo una experiencia relevante y útil.

A continuación, se muestran capturas de pantalla de la funcionalidad. Como el usuario no está dentro de los límites del campus, no es posible mostrar el trazado de una ruta:

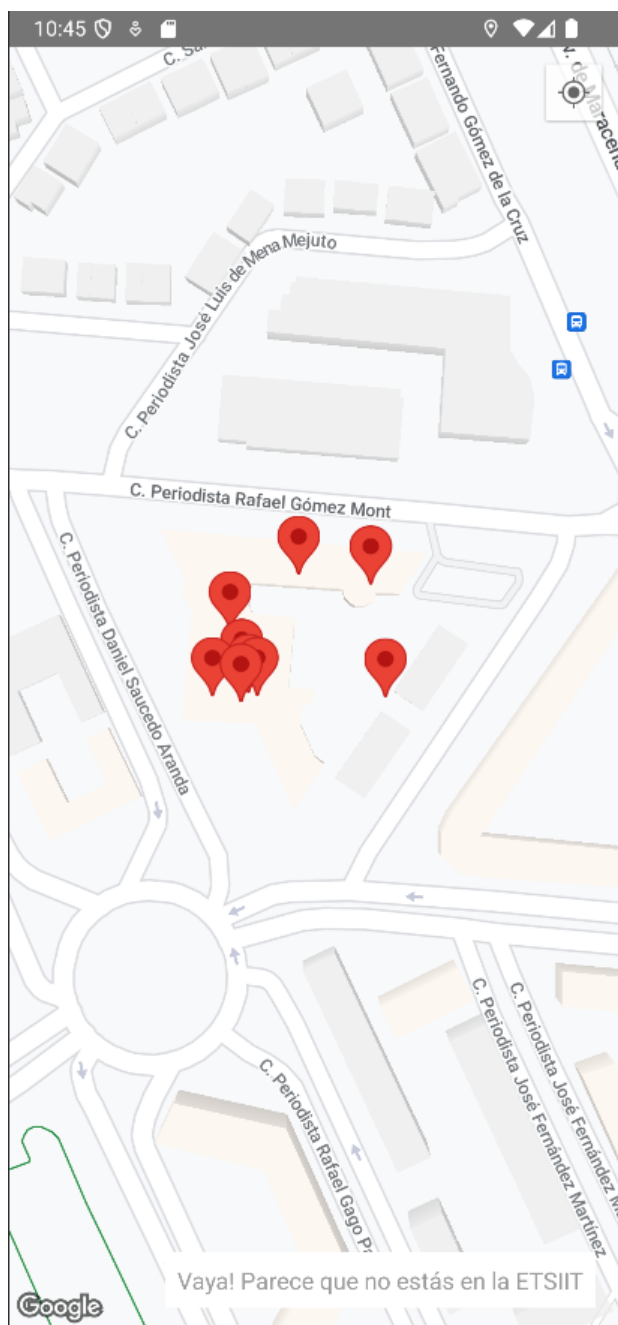


Figura 26: Mapa con puntos de interés marcados y mensaje indicando que el usuario está fuera de la escuela.

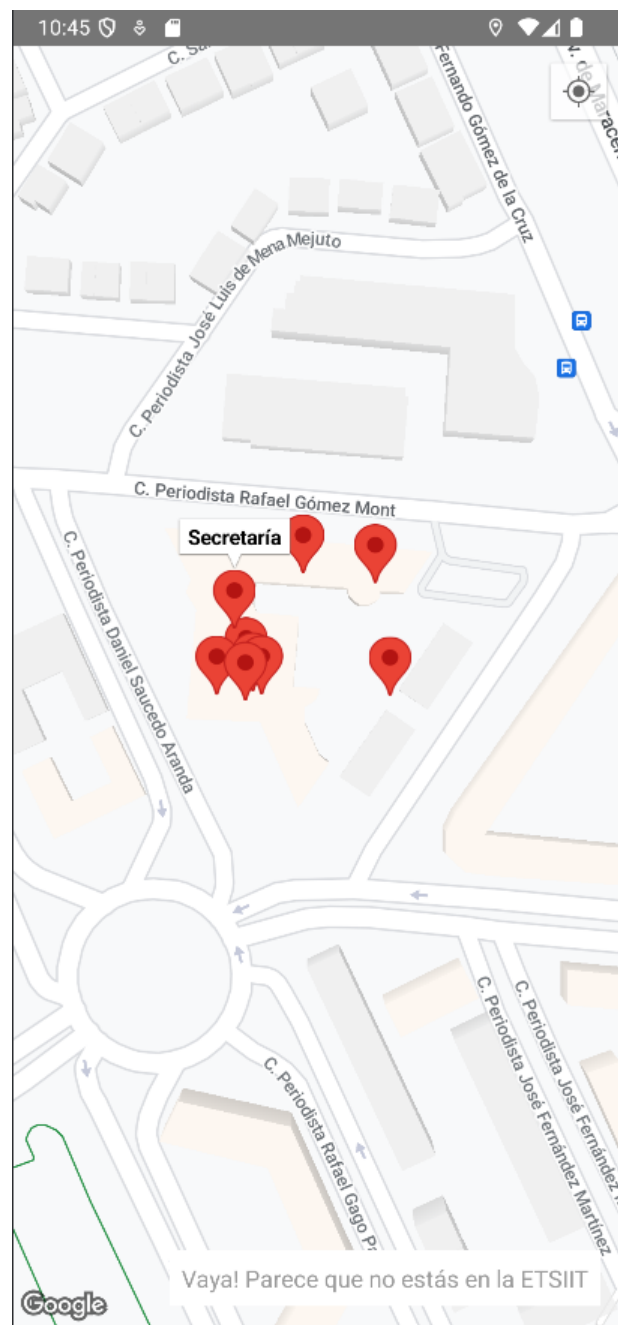


Figura 27: Etiqueta que se muestra al pulsar el marcador.

6. Navegación por Voz en el Menú Principal

La aplicación incorpora una funcionalidad que permite navegar por el menú principal utilizando comandos de voz. Esta característica, accesible desde un botón con un icono de micrófono ubicado en la barra superior junto al título *Menú Principal*, mejora la accesibilidad al sistema y facilita el acceso a los distintos submenús para usuarios con diferentes capacidades.



Figura 28: Botón de micrófono en el menú principal.

6.1. Funcionamiento General

El sistema permite al usuario activar el reconocimiento de voz pulsando un botón claramente identificado con un icono de micrófono. Una vez activado, la aplicación escucha comandos hablados y los asocia con las distintas opciones del menú principal. Comandos como *docencia*, *trámites* o *comedor* permiten acceder rápidamente a los submenús correspondientes.

El sistema maneja correctamente los permisos de grabación de audio, solicitándolos únicamente cuando son necesarios.

Para evitar posibles errores de reconocimiento, se han añadido varias formas alternativas para ciertos comandos. Por ejemplo, el acceso al chatbot, cuyo comando principal es *OyeETSIIT*, también responde a variaciones como *OyeETSI* o *OyeEPSY*. Este enfoque permite gestionar diferencias en pronunciación o incluso limitaciones del sistema de reconocimiento.

Sin embargo, una de las limitaciones actuales de esta funcionalidad es que los comandos de voz solo están disponibles en español. Incluso si la aplicación está configurada en inglés, los usuarios deberán emitir los comandos en español para que el sistema los reconozca. Este aspecto puede ser una barrera para usuarios no hispanohablantes, pero se espera abordar esta limitación en futuras mejoras.

6.2. Implementación Técnica

La funcionalidad está implementada utilizando la API de reconocimiento de voz de Android, configurada con un modelo de lenguaje libre (*free-form language model*). Este modelo permite que el sistema procese comandos hablados de forma flexible, sin estar restringido a un conjunto fijo de palabras clave. Los comandos reconocidos se comparan con un mapeo interno de actividades predefinidas, y si hay coincidencia, se inicia el submenú correspondiente mediante un **Intent**. En caso de no encontrar coincidencias, el sistema informa al usuario mediante un mensaje emergente.

El botón para activar el reconocimiento está ubicado en una posición destacada de la barra superior, junto al título del menú principal. Este diseño busca ser intuitivo y facilitar su identificación para todos los usuarios.

6.3. Accesibilidad y Usabilidad

Esta funcionalidad mejora significativamente la accesibilidad, ya que elimina la necesidad de interacción táctil para navegar por el menú. Esto beneficia especialmente a usuarios con movilidad reducida o dificultades para manipular una pantalla. Además, el sistema maneja variaciones de pronunciación para aumentar su efectividad, especialmente en palabras que podrían ser más complejas de reconocer.

Sin embargo, la dependencia del idioma español limita la accesibilidad para usuarios no hispanohablantes. Esta restricción ha sido identificada como una oportunidad de mejora, ya que permitir comandos en otros idiomas, como el inglés, aumentaría considerablemente el alcance y la utilidad de la funcionalidad.

6.4. Conclusión

La navegación por voz en el menú principal es una funcionalidad útil y accesible que facilita la interacción con la aplicación, especialmente para personas con limitaciones físicas o preferencias por métodos no táctiles. Sin embargo, también es importante reconocer sus limitaciones, como la dependencia del idioma español, que restringe su uso en configuraciones de idioma distintas. Aunque el sistema no es perfecto, representa un paso importante hacia una experiencia de usuario más inclusiva, con potencial de mejora en futuras versiones.

7. Reparto de trabajo

Toda la parte relacionada con la aplicación móvil y el chat de voz integrado en Andorid ha sido realizado por: Ana Graciani, Jaime Martínez y María Cribillés. Aquí queda detallado más concretamente:

- Diseño general de la aplicación: Ana
- Soporte multilingüe: Ana
- Sensor NFC: María
- Consulta de calendarios: Ana
- Sensor QR y tienda online: María
- Sensor multitáctil: Ana

- Sensor GPS y trazado de rutas: Ana
- Navegación por Voz en el Menú Principal: Ana
- Gestión de login: Jaime
- OyeETSIIT: Jaime
- Profesorado: Jaime
- Trámites: Jaime