



Título del Proyecto: G.A.B.I

Ciclo Formativo: IFC302 - Desarrollo de Aplicaciones Multiplataforma

Curso: 2023-2024

Datos del Alumno: Elton Saravia Ibarra





| | |
|--|----|
| <u>1.1. Contexto del proyecto</u> | 5 |
| <u>1.2. Ámbito y Entorno:</u> | 5 |
| <u>1.3. Análisis de la Realidad:</u> | 5 |
| <u>1.4. Solución y Justificación de la Solución Propuesta:</u> | 6 |
| <u>1.4. Destinatarios:</u> | 7 |
| <u>1.5. Objetivo del Proyecto:</u> | 7 |
| <u>1.6. Marco Legal:</u> | 7 |
| <u>1.6.1. Reglamento General de Protección de Datos (RGPD) de la UE:</u> | 7 |
| <u>1.6.2 Ley Orgánica de Protección de Datos Personales y Garantía de los Derechos Digitales (LOPD-GDD) en España:</u> | 8 |
| <u>1.6.3. Ley de Servicios de la Sociedad de la Información y del Comercio Electrónico (LSSI):</u> | 8 |
| <u>1.6.4. Ley de Autonomía del Paciente y Derechos y Obligaciones en Materia de Información y Documentación Clínica:</u> | 8 |
| <u>1.6.5. Normativas sobre ciberseguridad y encriptación de datos:</u> | 8 |
| <u>1.6.6. Conformidad con estándares técnicos de seguridad informática:</u> | 9 |
| <u>1.7. Licencia de Uso:</u> | 9 |
| <u>1.8. Abstract</u> | 9 |
| <u>2.1. Requisitos funcionales y no funcionales.</u> | 10 |
| <u>2.1.1. Requisitos Funcionales</u> | 11 |
| <u>2.1.1.1 GESTIÓN DE USUARIOS</u> | 11 |
| RF1: Registro Usuarios | 11 |
| RF2: Registro Residentes | 11 |
| RF3: Inicio Sesión | 11 |
| RF4: Cierre de sesión | 11 |
| RF5: Baja de usuario | 12 |
| RF5: Baja de Residente | 12 |
| <u>2.1.1.2 Aplicacion</u> | 12 |
| RF6: Pantalla Usuario | 12 |
| RF7: Pantalla administrador | 13 |
| RF8: Pantalla usuario enfermero | 13 |
| <u>2.1.2. Requisitos No Funcionales</u> | 13 |
| RNF1: Documentación | 13 |
| RNF2: Seguridad | 14 |
| Tabla de Control de Acceso Basado en Roles (RBAC) | 14 |
| RNF3: Mantenimiento | 15 |
| RNF4: Interfaz y usabilidad | 15 |
| RNF5: Rendimiento | 16 |
| <u>2.2. Tareas</u> | 16 |
| RF1: Registro de Empleados | 16 |
| RF2: Registro de Residentes | 16 |
| RF3: Inicio de Sesión | 16 |
| RF4: Cierre de Sesión | 16 |
| RF5: Baja de Usuario y Residente | 17 |



| | |
|---|----|
| <u>RF6: Cuenta de Usuario</u> | 17 |
| <u>RF7: Pantalla Administrador</u> | 17 |
| <u>RF8: Pantalla Usuario Enfermero</u> | 17 |
| <u>RNF1: Documentación</u> | 18 |
| <u>RNF2: Seguridad</u> | 18 |
| <u>RNF3: Mantenimiento</u> | 18 |
| <u>RNF5: Rendimiento</u> | 18 |
| <u>2.3. Metodología a seguir para la realización del proyecto.</u> | 18 |
| <u>Elementos de un tablero de kanban</u> | 20 |
| <u>2.4. Planificación temporal de tareas. (Diagramas de Gantt)</u> | 22 |
| <u>2.5. Presupuesto (gastos, ingresos, beneficio) (formato formal)</u> | 24 |
| <u>2.5.1. Resumen del Presupuesto</u> | 24 |
| <u>2.5.2. Gastos</u> | 25 |
| <u>2.4.3. Ingresos</u> | 26 |
| <u>2.5.4. Beneficios</u> | 27 |
| <u>2.5.5. Comentarios</u> | 28 |
| <u>2.5.6 Propuesta de presupuesto</u> | 28 |
| <u>2.6. Contrato (y Pliego de condiciones) o Licencia de distribución (según proceda)</u> | 30 |
| <u>2.6.1. Definiciones</u> | 30 |
| <u>2.6.2. Concesión de la Licencia</u> | 30 |
| <u>2.6.2.1. Uso del Software</u> | 30 |
| <u>2.6.2.2. Restricciones</u> | 30 |
| <u>2.6.3. Propiedad Intelectual</u> | 31 |
| <u>2.6.4. Duración de la Licencia</u> | 31 |
| <u>2.6.5. Garantías y Renuncias</u> | 31 |
| <u>2.6.6. Limitación de Responsabilidad</u> | 32 |
| <u>2.7. Análisis de riesgos (DAFO)</u> | 32 |
| <u>2.7.1. Fortalezas</u> | 32 |
| <u>2.7.2. Debilidades</u> | 32 |
| <u>2.7.3. Oportunidades</u> | 33 |
| <u>2.7.4. Amenazas</u> | 33 |
| <u>3. ANÁLISIS Y DISEÑO</u> | 34 |
| <u>3.1.1 Arquitectura de la aplicación</u> | 34 |
| <u>3.1.2 Componentes clave de la arquitectura</u> | 36 |
| <u>3.2. Tecnologías/Herramientas usadas y descripción de las mismas</u> | 36 |
| <u>3.2.1. Frontend - Android Studio:</u> | 36 |
| <u>3.2.2. Backend - MySQL:</u> | 36 |
| <u>3.2.3. Backend - PHP:</u> | 37 |
| <u>3.2.4. Gestión de Sesiones - RBAC API:</u> | 37 |
| <u>3.2.5. Firebase - Realtime Database:</u> | 37 |
| <u>3.3. Arquitectura de componentes de la aplicación</u> | 38 |
| <u>3.3.1 Backend</u> | 41 |
| <u>3.3.3 Conexión al Servidor de Hostinger para Configuración de CORS e Instalación de Composer</u> | 44 |



| | |
|--|-----|
| <u>3.4. Modelado de datos</u> | 46 |
| <u>3.4.1. Tablas y descripción base de datos MYSQL</u> | 47 |
| <u>3.4.2 Documentación de Software del Modelado de Datos para Firebase</u> | 57 |
| <u>3.5. Análisis y diseño de la interfaz de usuario</u> | 58 |
| <u>3.5.1. User Persona</u> | 59 |
| <u>3.5.2. Frontend</u> | 63 |
| <u>4. IMPLEMENTACIÓN E IMPLANTACIÓN DEL SISTEMA</u> | 70 |
| <u>4.1. Implementación</u> | 70 |
| <u>4.1.1. Configuración Inicial del Proyecto Android</u> | 70 |
| <u>4.1.1.1. Archivo build.gradle.kts</u> | 70 |
| <u>4.1.1.2. Archivo AndroidManifest.xml</u> | 72 |
| <u>4.1.2. Interfaz de Usuario (UI)</u> | 73 |
| <u>4.1.2.1. Layouts Principales</u> | 74 |
| <u>4.1.2.2.. Menús</u> | 76 |
| <u>4.1.2.3. Definición de Items</u> | 77 |
| <u>4.1.2.4. Drawable Resources</u> | 78 |
| <u>4.1.3 Implementación de Activities y Fragments</u> | 78 |
| <u>4.1.3.1 Diagramas y descripción de clases</u> | 82 |
| <u>4.1.3.1.1 Flujo de Rol Administrador</u> | 83 |
| <u>4.1.3.1.2. Flujo Rol Auxiliar</u> | 101 |
| <u>4.1.3.1.3. Flujo ChatActivity</u> | 120 |
| <u>4.1.3. Conexión y Comunicación con el Servidor</u> | 124 |
| <u>4.1.3.1 Ejemplo de conectividad MYSQL</u> | 126 |
| <u>Clase SubirArchivoFragment:</u> | 131 |
| <u>Conclusión</u> | 133 |
| <u>5. CIERRE</u> | 134 |
| <u>5.1. Resultados obtenidos y conclusiones</u> | 134 |
| <u>5.2. Diario de bitácora</u> | 136 |
| <u>5.3. Temporalización y desviación sobre la planificación inicial</u> | 146 |
| <u>5.3.1. Temporalización del Proyecto</u> | 147 |
| <u>5.3.2. Desviaciones sobre la Planificación Inicial</u> | 147 |
| <u>5.3.3. Análisis del Gráfico de Incidencias</u> | 149 |
| <u>6. Bibliografía</u> | 150 |



1. Descripción del proyecto

1.1. Contexto del proyecto

El proyecto "G.A.B.I" es una iniciativa para desarrollar una aplicación móvil destinada a mejorar la gestión de residencias de ancianos. La idea surge ante la necesidad de proporcionar una herramienta eficiente para el personal de estas instituciones, facilitando el seguimiento de las tareas diarias y el acceso a información vital sobre los pacientes.

El nombre GABI es un acrónimo de Gabriela, que es un nombre de niña de origen hebreo cuyo significado es “fuerza de Dios”. Es la variante femenina del nombre de Gabriel, que era uno de los arcángeles más importantes: es el mensajero del cielo que anuncia a la Virgen María el nacimiento de Cristo..

1.2. Ámbito y Entorno:

Este proyecto se ubica en el ámbito de la gestión de salud y asistencia a personas mayores, específicamente en residencias de ancianos. El entorno de uso será principalmente dentro de estas instituciones, abarcando diversas áreas como gestión de residentes, comunicación con familiares, y administración de personal.

1.3. Análisis de la Realidad:

Actualmente, muchas residencias de ancianos enfrentan desafíos en la gestión eficiente de sus recursos y comunicación efectiva. Falta una solución integrada que facilite el manejo de la información de los residentes, la asignación de tareas al personal y la comunicación con los familiares de los residentes. Después de realizar una investigación en diversas residencias, llegue a la conclusión de que la administración y documentación de las tareas a dia de hoy en muchas de estas instituciones, muchas manejan toda la información a papel y otras ni siquiera tienen en cuenta el valor de la comunicación entre empleados para distribuir las tareas de forma apropiada.

Los encargados suelen crear hojas de bitácoras que los empleados deben llenar cada vez que realizan una acción, ya sea el fichaje, la limpieza de una habitación o alguna acción con un residente. Me comentan que lo habitual es que los empleados



rellenen al final de su jornada lo que han hecho a lo largo del día, cuando en realidad se les da instrucciones de que deben llenar las hojas en el acto.

Además me indican que la comunicación en residencias con superficies amplias es complicada ya que en muchas residencias utilizan Radios o directamente no utilizan nada y si necesitan ayuda necesitan recorrer el recinto hasta encontrar a la persona, por ejemplo para levantar a algún residente que debe ser cambiado de posición cada hora y se debe hacer entre 2 personas.

A las enfermeras se les dificulta la labor de consultar historiales médicos ya que todo lo almacenan en papel en la mayoría de ocasiones de residencias pequeñas y para suministrar medicamentos suelen imprimir una hoja cuando hay variaciones en algún residente que en ocasiones los auxiliares de geriatría no revisan y se pueden llegar a cometer errores.

1.4. Solución y Justificación de la Solución Propuesta:

La solución propuesta es la aplicación "G.A.B.I", diseñada para abordar las necesidades de gestión y comunicación en las residencias de ancianos. La aplicación permitirá la gestión eficiente de los residentes, la asignación de tareas al personal, el acceso a información rápida y un historial de las diferentes acciones que se llevan a cabo al final del día. Esto justifica su desarrollo, ya que mejorará significativamente la operatividad y la calidad del servicio en estas instituciones.

El feedback recibido tras comunicar a los diferentes administradores sobre mi intención de crear esta app ha sido positiva, hemos llegado a la conclusión de que tener la información de manera ágil y además llevar a cabo un control informatizada en una residencia acotará el margen de error del personal menos cualificado y ayudará a la administración de la residencia para optimizar los tiempos y el personal.

Los residentes notarán una mejoría en su ambiente debido a que por causa de errores se crean a veces conflictos entre empleados que pueden llegar a enturbiar el ambiente de calma y tranquilidad que las residencias quieren transmitir.

Los empleados agradecerán tener unas vías para reclamar reconocimiento por su trabajo y les permitirá centrarse únicamente en sus tareas sin tener que estar pensando en que se les culpe de un error de alguien más.



Para el administrador es conveniente obtener datos del trabajo de sus empleados ya que esto le puede impulsar a tomar medidas de refuerzo positivo para sus empleados u otras acciones.

1.4. Destinatarios:

Los principales destinatarios de "G.A.B.I" son las residencias de ancianos, incluyendo administradores, personal de salud y asistentes. Además, indirectamente beneficia a los residentes y sus familiares, mejorando la calidad del cuidado y la comunicación.

1.5. Objetivo del Proyecto:

El objetivo es desarrollar una aplicación móvil que centralice y facilite la gestión de las tareas en residencias de ancianos, mejorando la eficiencia operativa y la calidad de la comunicación entre el personal y los residentes.

Los empleados tendrán una herramienta para dejar reflejadas sus tareas realizadas y además tendrán acceso fácil a la información de cada paciente. El administrador podrá tener un histórico de las diferentes acciones realizadas para un residente. Es importante almacenar toda la información relevante para que los enfermeros puedan seguir las recetas de los doctores y controlar los calendarios de las dosis o vacunas que deben realizar por ello deben tener un acceso sencillo a la información de los residentes.

1.6. Marco Legal:

El proyecto se desarrollará cumpliendo con las normativas legales pertinentes, incluyendo la protección de datos personales y la seguridad de la información. Se tomarán medidas para garantizar la encriptación de datos sensibles y el uso de autenticación segura, cumpliendo con los estándares de seguridad y privacidad requeridos en el manejo de información de salud y datos personales.

1.6.1. Reglamento General de Protección de Datos (RGPD) de la UE:

Aunque el proyecto se desarrolle en España, es crucial adherirse al RGPD, que regula el tratamiento de datos personales en la Unión Europea. Esto incluye obtener



el consentimiento explícito de los usuarios para recopilar y procesar sus datos, permitir a los usuarios acceder a sus datos para verificación y corrección, y garantizar el derecho al olvido.

1.6.2 Ley Orgánica de Protección de Datos Personales y Garantía de los Derechos Digitales (LOPD-GDD) en España:

Esta ley complementa y adapta el RGPD a la legislación española. Implica medidas específicas como la designación de un Delegado de Protección de Datos en organizaciones que manejan información sensible y la realización de Evaluaciones de Impacto sobre la Protección de Datos cuando se traten datos a gran escala.

1.6.3. Ley de Servicios de la Sociedad de la Información y del Comercio Electrónico (LSSI):

Esta ley afecta a la aplicación en cuanto a la gestión de comunicaciones electrónicas, especialmente en el envío de notificaciones y comunicaciones a familiares y empleados a través de la aplicación.

1.6.4. Ley de Autonomía del Paciente y Derechos y Obligaciones en Materia de Información y Documentación Clínica:

Regula los derechos de los pacientes a ser informados sobre su salud y a la confidencialidad de su información médica.

1.6.5. Normativas sobre ciberseguridad y encriptación de datos:

Dado que la app maneja datos sensibles, será necesario implementar fuertes medidas de seguridad, como el uso de encriptación para datos almacenados y transmitidos, así como la implementación de autenticación robusta y control de accesos.



1.6.6. Conformidad con estándares técnicos de seguridad informática:

Adherencia a estándares como ISO/IEC 27001 para la gestión de la seguridad de la información y otras normas relevantes que aseguren la protección adecuada de los datos y la infraestructura tecnológica.

1.7. Licencia de Uso:

La aplicación "G.A.B.I" se proporciona bajo una licencia comercial propietaria. El pago de la licencia otorga al usuario el derecho no exclusivo y no transferible de usar la aplicación.

- Acceso al Código Fuente:

Tras la compra, el usuario tendrá acceso al código fuente de la aplicación "G.A.B.I". Este acceso permite al usuario realizar modificaciones al código fuente para su propio uso.

La redistribución de las versiones modificadas de la aplicación, ya sea de forma gratuita o mediante pago, queda expresamente prohibida, a menos que se obtenga un permiso por escrito del propietario del software.

- Protección de la Propiedad Intelectual:

A pesar de permitir modificaciones, todos los derechos de propiedad intelectual relacionados con la aplicación "G.A.B.I", incluidos los diseños, código fuente, nombre de la aplicación y logotipos, siguen siendo propiedad exclusiva del creador original.

Cualquier uso de la marca, nombre comercial o derechos de autor asociados con "G.A.B.I" debe contar con el consentimiento explícito del propietario.

1.8. Abstract

The "G.A.B.I" project is an initiative to develop a mobile application aimed at improving the management of nursing homes. This application addresses the need for an efficient tool for staff in these institutions, facilitating the tracking of daily tasks and providing access to vital information about residents. The name GABI is an acronym for Gabriela, which is a girl's name of Hebrew origin meaning "God's strength," and it represents a commitment to strength and support in elder care.



This project operates within the realm of health management and elder care, specifically targeting nursing homes. The application will be primarily used within these institutions, covering areas such as resident management, communication with family members, and staff administration. Currently, many nursing homes struggle with efficient resource management and effective communication. There is a lack of integrated solutions that facilitate the handling of resident information, task allocation to staff, and communication with residents' families. Research indicates that many nursing homes still rely on paper-based systems for documentation and lack proper communication tools, which leads to inefficiencies and potential errors.

The proposed solution is the "G.A.B.I" application, designed to meet the management and communication needs of nursing homes. The application will enable efficient resident management, task assignment to staff, quick access to information, and a log of various actions performed throughout the day. This will significantly improve operational efficiency and service quality in these institutions. Feedback from administrators has been positive, highlighting the benefits of having real-time information and computerized control, which will reduce errors and optimize time and personnel management.

The primary beneficiaries of "G.A.B.I" are nursing homes, including administrators, health staff, and assistants. Indirectly, it also benefits residents and their families by enhancing care quality and communication. The objective is to develop a mobile application that centralizes and facilitates task management in nursing homes, improving operational efficiency and communication quality between staff and residents. Employees will have a tool to document their tasks and access patient information easily, while administrators will maintain a historical record of actions performed on residents. This will help nurses follow doctors' prescriptions and control medication schedules, ensuring better patient care and safety.

2. ACUERDO DEL PROYECTO

2.1. Requisitos funcionales y no funcionales.

Se ha creado una lista de requisitos según la información de los datos y acciones que se realizan a lo largo del día en varias residencias de ancianos y se han predefinido una serie de requisitos funcionales que permitan a los usuarios realizar dichas tareas que se consideran de alta prioridad. A continuación se muestran los requisitos con una breve descripción



2.1.1. Requisitos Funcionales

2.1.1.1 GESTIÓN DE USUARIOS

RF1: Registro Usuarios

1. La aplicación debe tener una interfaz donde un administrador pueda insertar los datos de un nuevo usuario a través de un formulario.
2. El sistema se encargará de verificar que los datos son correctos.
3. El sistema debe mostrar un mensaje de error si algún dato falta o es incorrecto.
4. Al llenar los campos aparece un mensaje de pregunta de confirmación y posteriormente el sistema se encargará de almacenar los datos.
5. El sistema enviará un mensaje de bienvenida al nuevo usuario en el caso de que se haya almacenado de manera correcta.
6. El programa mostrará un mensaje de guardado correctamente y volverá a la pantalla principal.

RF2: Registro Residentes

1. La aplicación debe tener una interfaz donde un administrador pueda insertar los datos de un nuevo residente a través de un formulario.
2. El sistema se encargará de verificar que los datos son correctos.
3. El sistema debe mostrar un mensaje de error si algún dato falta o es incorrecto.
4. Al llenar los campos aparece un mensaje de pregunta de confirmación y posteriormente el sistema se encargará de almacenar los datos.
5. El programa mostrará un mensaje de guardado correctamente y volverá a la pantalla principal.

RF3: Inicio Sesión

1. Para identificarse el usuario deberá usar su correo electrónico o su nombre de usuario y además escribir la contraseña.
2. El sistema se encargará de validar al usuario y permitir el acceso o denegar a la aplicación.
3. El sistema mostrará un mensaje de error si el login ha sido fallido.
4. En el caso de que la validación sea correcta se accedera a la pantalla principal de la aplicación

RF4: Cierre de sesión

1. Cualquier usuario de la aplicación debe poder finalizar sesión en la aplicación mediante un botón que indique "Cierre de sesión".



2. Si el usuario presiona el botón de cierre de sesión se le mostrará un mensaje de confirmación de cierre.
3. Si se confirma el cierre de sesión el usuario será redirigido al apartado de Inicio de sesión de la aplicación.

RF5: Baja de usuario

1. El usuario ni el residente pueden tramitar la baja, solo la puede tramitar un administrador.
2. El sistema preguntará por un nombre de usuario o correo electrónico para tramitar la baja del sistema.
3. El sistema al presionar el botón eliminar sobre un usuario, hará una pregunta confirmación, permitiendo aceptar o cancelar.
4. Si el administrador confirma la baja del usuario, el sistema eliminará los datos de acceso del usuario pero no el histórico de sus tareas.

RF5: Baja de Residente

1. El usuario ni el residente pueden tramitar la baja, solo la puede tramitar un administrador.
2. El sistema preguntará por un DNI para poder tramitar la baja del residente..
3. El sistema al presionar el botón eliminar sobre un residente, hará una pregunta confirmación, permitiendo aceptar o cancelar.
4. Si el administrador confirma la baja del residente, el sistema dará de baja el acceso a los datos desde la aplicación para los usuarios.

2.1.1.2 Aplicacion

RF6: Pantalla Usuario

1. Tendrá una pantalla para consultar datos sobre su perfil de usuario, pero no modificar ningún dato.
2. Tendrá una pantalla para guardar los datos sobre una tarea llevada a cabo con un residente.
3. Si presiona al botón de guardar tarea debe salir un mensaje de confirmación.
4. El usuario tendrá un botón para buscar y acceder a la información de los controles recientes que se han llevado a cabo para un residente,
5. El usuario tendrá un botón para poder visualizar la información del perfil de un residente.
6. En la pantalla deberá haber un desplegable con las diferentes tareas que puede llevar a cabo con un residente y un cuadro de texto para anotar información adicional si así lo desea.
7. El usuario tendrá la lista de residentes para acceder rápidamente al apartado de controles de ese residente mediante las imágenes de estos.



RF7: Pantalla administrador

1. El administrador debe tener una pantalla con su perfil y todos los datos sobre su perfil y podrá modificar datos.
2. El administrador tendrá botones en la pantalla:
 - a. Agregar usuarios.
 - b. Eliminar usuarios
 - c. Modificar usuarios.
 - d. Agregar residente.
 - e. Eliminar residente
 - f. Modificar Residente
 - g. Acceder a información de usuario donde pueda revisar un histórico de las tareas.
 - h. Acceder a la información de los residentes donde pueda ver un histórico de los controles que se le han realizado.
 - i. Añadir datos de historial clínico a residentes.

RF8: Pantalla usuario enfermero

1. El usuario enfermero podrá acceder al historial clínico del residente además de poder acceder a todos los apartados que puede acceder un usuario empleado.
2. El usuario enfermero debe poder añadir o modificar datos sobre los controles que se le deben realizar a un residente.
3. El usuario enfermero contará con una pantalla de stock de las pastillas almacenadas y podrá decidir la dosis para cada residente.

2.1.2. Requisitos No Funcionales

RNF1: Documentación

1. Manual de usuario de la aplicación que incluya los diferentes usos de los usuarios:
 - a. Usuario empleado
 - b. Usuario enfermera
 - c. Usuario administrador
2. La codificación debe ser clara y seguir los estándares de codificación de Android.



RFN2: Seguridad

1. Para poder utilizar la aplicación siempre se debe autenticar al usuario.
2. Solo hará falta autenticarse una vez por dispositivo.
3. Si el usuario no cierra sesión se mantendrá abierta para seguir siendo utilizada en cualquier momento.
4. Los datos almacenados deben ser cifrados
5. Se debe aplicar un control de acceso basado en roles que distinga los siguientes permisos:

Tabla de Control de Acceso Basado en Roles (RBAC)

| Acción/Permiso | Administrador | Enfermero | Empleado |
|------------------------------------|---------------|-----------|----------|
| Registrar Usuario | ✓ | | |
| Registrar Residente | ✓ | | |
| Iniciar Sesión | ✓ | ✓ | ✓ |
| Cerrar Sesión | ✓ | ✓ | ✓ |
| Dar de Baja a Usuario | ✓ | | |
| Dar de Baja a Residente | ✓ | | |
| Modificar Datos de Usuario | ✓ | | |
| Modificar Datos de Residente | ✓ | ✓ | |
| Acceder a Información de Usuario | ✓ | | |
| Acceder a Información de Residente | ✓ | ✓ | ✓ |
| Gestionar Historial Clínico | ✓ | ✓ | |
| Registrar Tareas | ✓ | ✓ | ✓ |
| Modificar Tareas | ✓ | ✓ | |
| Consultar Historial de Tareas | ✓ | ✓ | ✓ |



| | | | |
|--|---|---|---|
| Gestionar Stock de Medicamentos | ✓ | ✓ | |
| Administrar Perfiles de Usuario | ✓ | | |
| Modificar Configuraciones de Seguridad | ✓ | | |
| Visualizar Alertas y Notificaciones | ✓ | ✓ | ✓ |

✓: Indica que el rol tiene permisos para realizar la acción.

(vacío): Indica que el rol no tiene permisos para realizar la acción.

Explicación de Roles:

- **Administrador:** Tiene control de los datos de la aplicación, incluyendo la capacidad de agregar o eliminar usuarios y residentes, modificar cualquier tipo documentación.
- **Enfermero:** Puede acceder y modificar información médica, gestionar el stock de medicamentos y realizar actualizaciones al historial clínico de los residentes. No puede modificar datos de usuarios ni realizar gestiones administrativas más allá de la parte clínica.
- **Empleado:** Principalmente interactúa con la aplicación para registrar tareas y acceder a la información de residentes necesaria para sus actividades diarias. No tiene permisos para modificar datos significativos ni acceso a configuraciones administrativas.

RFN3: Mantenimiento

1. Debe funcionar para la versión de android que abarque al menos el 85% de los dispositivos con este sistema operativo.
2. Será necesario disponer de internet en el dispositivo para poder interactuar con la aplicación.

RFN4: Interfaz y usabilidad

1. Interfaz sencilla e intuitiva. De tal manera que su uso no suponga un esfuerzo para el usuario que le pueda afectar en su desempeño.
2. La introducción de datos debe ser clara y estructurada para evitar errores.
3. La información de un residente no debe dar lugar a error en cuenta a confusión con la información de otro residente.
4. Mensajes informativos referentes a porque ha fallado alguna solicitud en la aplicación.
5. Agilidad entre cambiar de pantallas sin necesidad de pasar por menús innecesarios.



RFN5: Rendimiento

1. Se espera que las solicitudes se efectúen en un tiempo no superior a 2 segundos, respectos a inserción de datos en las tareas de los usuarios.
2. Se deben poder subir imágenes y documentos asociados a cada residente.
3. En el caso de la extracción de datos o subida de documentos se espera que los tiempos sean superiores ya que dependen del tamaño del archivo.

2.2. Tareas

A continuación, se detallan las tareas principales asociadas a cada requisito funcional, junto con un identificador único para cada tarea, que será útil para su incorporación en un diagrama de Gantt:

RF1: Registro de Empleados

- Tarea 1.1 (RF1): Diseñar la interfaz de usuario para el registro de empleados.
- Tarea 1.2 (RF1): Desarrollar el backend para la gestión de datos de empleados.
- Tarea 1.3 (RF1): Implementar validaciones de datos en el formulario de registro.
- Tarea 1.4 (RF1): Realizar pruebas unitarias y de integración para el registro de empleados.

RF2: Registro de Residentes

- Tarea 2.1 (RF2): Diseñar la interfaz de usuario para el registro de residentes.
- Tarea 2.2 (RF2): Desarrollar el backend para la gestión de datos de residentes.
- Tarea 2.3 (RF2): Implementar validaciones de datos en el formulario de registro.
- Tarea 2.4 (RF2): Realizar pruebas unitarias y de integración para el registro de residentes.

RF3: Inicio de Sesión

- Tarea 3.1 (RF3): Crear la interfaz de inicio de sesión.
- Tarea 3.2 (RF3): Implementar la autenticación y seguridad de la sesión.
- Tarea 3.3 (RF3): Pruebas de seguridad y funcionalidad del inicio de sesión.

RF4: Cierre de Sesión

- Tarea 4.1 (RF4): Desarrollar la funcionalidad de cierre de sesión.
- Tarea 4.2 (RF4): Integrar y probar el cierre de sesión con el sistema de autenticación.



RF5: Baja de Usuario y Residente

- Tarea 5.1 (RF5): Implementar la interfaz y lógica de negocio para la baja de usuarios.
- Tarea 5.2 (RF5): Implementar la interfaz y lógica de negocio para la baja de residentes.
- Tarea 5.3 (RF5): Realizar pruebas para asegurar el correcto funcionamiento de las bajas.

RF6: Cuenta de Usuario

- Tarea 6.1 (RF6): Diseñar y desarrollar la interfaz de gestión de cuenta de usuario.
- Tarea 6.2 (RF6): Integrar la funcionalidad de visualización y edición de datos de usuario.
- Tarea 6.3 (RF6): Testear la gestión de cuenta de usuario para garantizar la seguridad y correcta funcionalidad.

RF7: Pantalla Administrador

- Tarea 7.1 (RF7): Diseñar la interfaz de usuario para la pantalla del administrador.
- Tarea 7.2 (RF7): Desarrollar las funcionalidades de gestión de usuarios (agregar, eliminar, modificar).
- Tarea 7.3 (RF7): Desarrollar las funcionalidades de gestión de residentes (agregar, eliminar, modificar).
- Tarea 7.4 (RF7): Implementar acceso a historiales de tareas y controles de residentes.
- Tarea 7.5 (RF7): Integrar la funcionalidad para añadir y modificar datos del historial clínico de los residentes.
- Tarea 7.6 (RF7): Realizar pruebas de funcionalidad y seguridad de la pantalla del administrador.

RF8: Pantalla Usuario Enfermero

- Tarea 8.1 (RF8): Diseñar la interfaz de usuario para la pantalla del usuario enfermero.
- Tarea 8.2 (RF8): Implementar el acceso y modificación del historial clínico de los residentes.
- Tarea 8.3 (RF8): Desarrollar la gestión de stock de medicamentos y decisión de dosis.
- Tarea 8.4 (RF8): Probar la funcionalidad y seguridad de la pantalla del usuario enfermero.



RNF1: Documentación

- Tarea 9.1 (RNF1): Crear manuales de usuario para los roles de empleado, enfermero y administrador.
- Tarea 9.2 (RNF1): Revisar y asegurar que la documentación sigue los estándares de codificación y es comprensible.
- Tarea 9.3 (RNF1): Revisar y asegurar que la documentación del proyecto para la entrega final esté actualizada.

RNF2: Seguridad

- Tarea 10.1 (RNF2): Implementar mecanismos de cifrado de datos.
- Tarea 10.2 (RNF2): Implementar mecanismos de autenticación.
- Tarea 10.3 (RNF2): Configurar y probar el control de acceso basado en roles.

RNF3: Mantenimiento

- Tarea 11.1 (RNF3): Asegurar compatibilidad con la versión de Android que abarque al menos el 85% de los dispositivos.
- Tarea 11.2 (RNF3): Implementar y probar requerimientos de conectividad a Internet.
- Tarea 11.3 (RNF3): Implementar sistemas de versión de controles

RNF4: Interfaz y Usabilidad

- Tarea 12.1 (RNF4): Diseñar una interfaz intuitiva y fácil de usar para todos los roles de usuario.
- Tarea 12.2 (RNF4): Realizar pruebas de usabilidad para optimizar la experiencia del usuario.

RNF5: Rendimiento

- Tarea 13.1 (RNF5): Probar las operaciones de inserción de datos para que no superen los 2 segundos.
- Tarea 13.2 (RNF5): Implementar y probar la subida y extracción de imágenes y documentos.
- Tarea 13.3 (RNF5): Verificar el correcto funcionamiento de creación de nuevos usuarios.

2.3. Metodología a seguir para la realización del proyecto.

La metodología en cascada se seguirá en este proyecto a través de una serie de fases secuenciales que incluyen: análisis de requisitos, diseño del sistema y software, implementación y codificación, pruebas, despliegue, y mantenimiento. Cada fase tiene objetivos específicos y debe completarse antes de pasar a la

siguiente, con revisiones y validaciones al final de cada etapa para asegurar que todo está correcto antes de proceder.

Al ser un proyecto individual es el mejor modelo para este proceso de desarrollo de software.



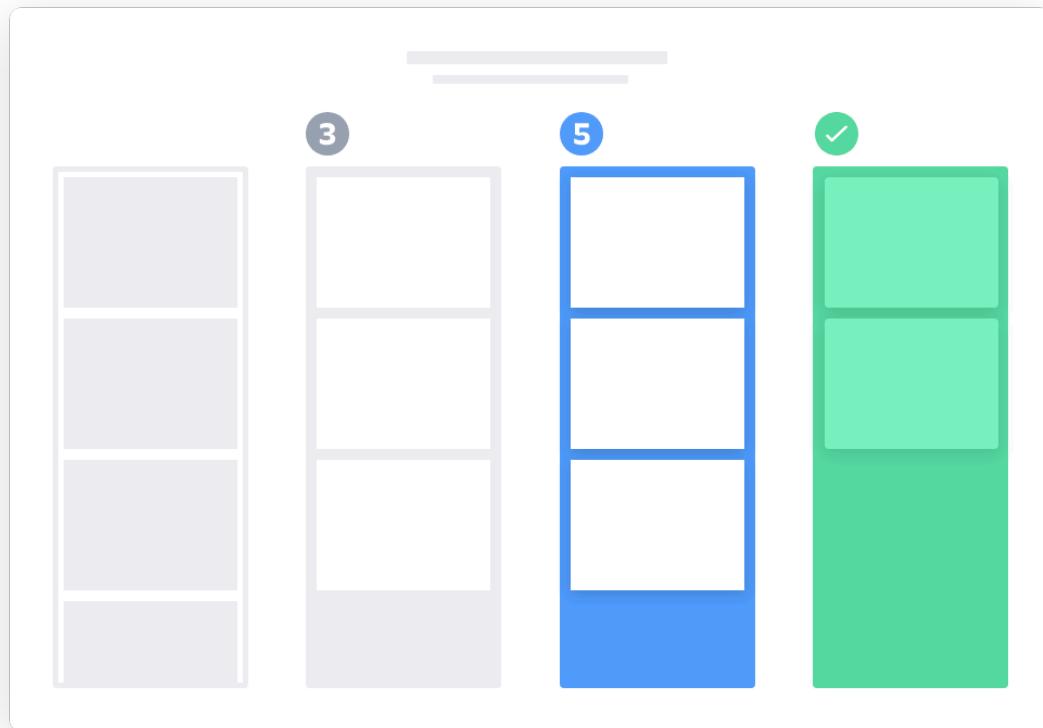
Fases de la Metodología en Cascada

- Análisis de Requisitos:** Se recopilarán y documentará de manera incremental todos los requisitos del software, basándose en el análisis de las necesidades de los usuarios finales y las condiciones del entorno de laboral en las residencias de ancianos.
- Diseño del Sistema y Software:** Se elaborarán los diseños técnicos del sistema, incluyendo la arquitectura de la base de datos y los prototipos de las interfaces de usuario.
- Implementación y Codificación:** Esta fase se centrará en la construcción real del software, programando las funcionalidades descritas en los documentos de diseño.
- Verificación:** Se realizarán pruebas para asegurar que el software funciona correctamente y cumple con los requisitos especificados inicialmente.
- Mantenimiento:** Despues del despliegue, se continuará supervisando y modificando el software para corregir errores o realizar mejoras necesarias.

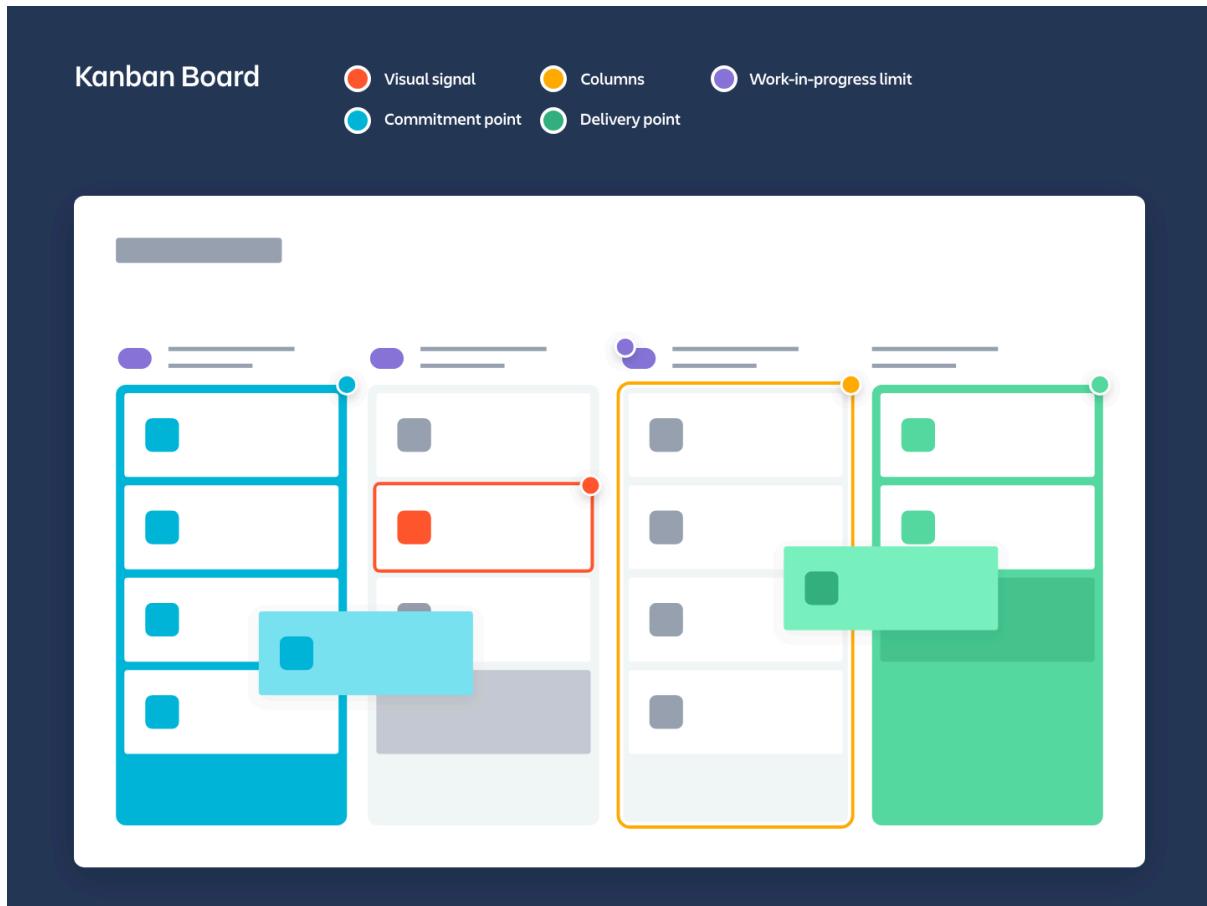
Kanban ha recorrido un largo camino desde sus orígenes en el ámbito de la producción lean gracias a un pequeño pero mañoso grupo de entusiastas del kanban. El trabajo de David Anderson para definir el método kanban ayudó a llevar el kanban al dominio del software y los servicios; y Personal Kanban, de Jim Benson



y Tonianne DeMaria, ha contribuido a extender las aplicaciones de kanban a ámbitos que ni te imaginarías.



Yo utilizo tableros de kanban todos los días y no podría imaginarme la vida sin ellos. Las ideas y prácticas recomendadas que aquí se recogen constituyen un popurrí de mi experiencia personal, mis investigaciones y las conversaciones que he mantenido con [Zach Nies](#), [Keith Nottinson](#) y [Jim Benson](#). Lo que hace que recorra constantemente al kanban son sus valores y su (sorprendente) ausencia de reglas. Los valores de kanban son el respeto a las personas y la mejora continua.

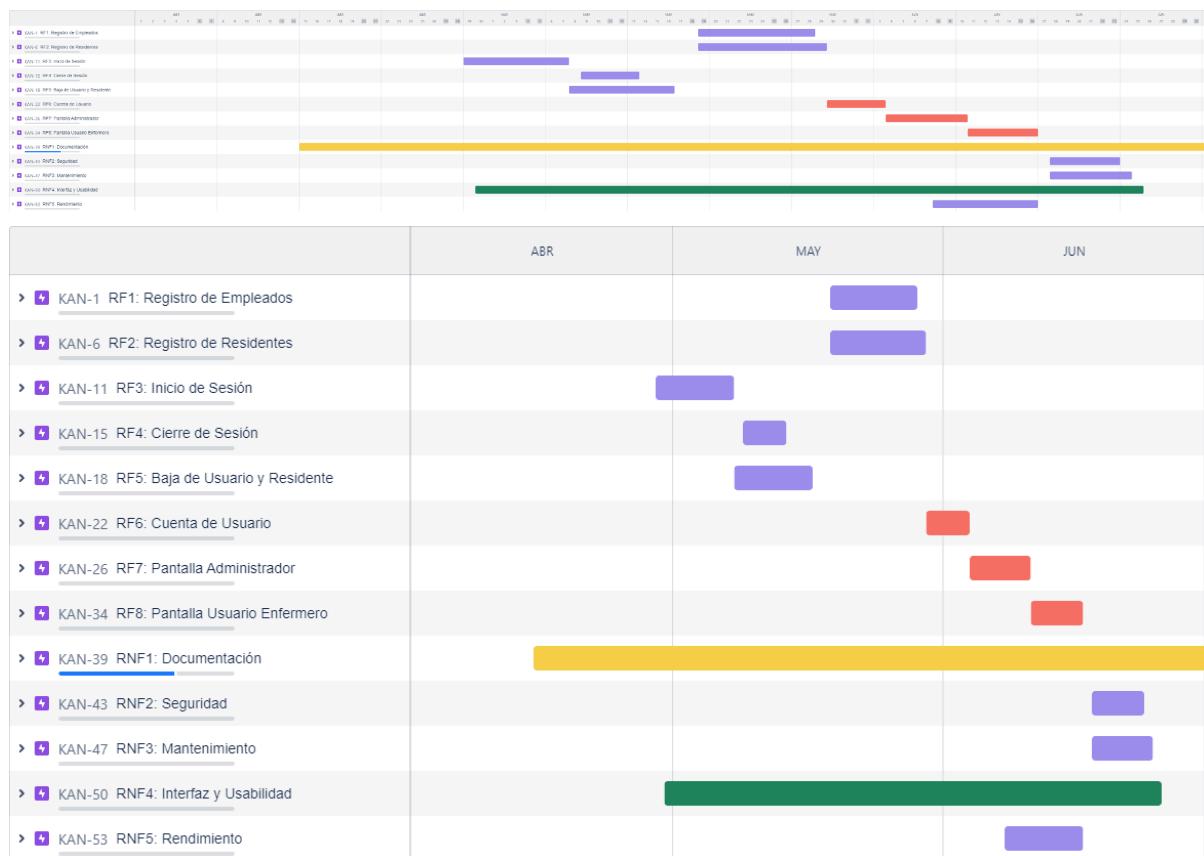


2.4. Planificación temporal de tareas. (Diagramas de Gantt)

El proyecto seguirá un cronograma claro en JIRA, con tareas y sub-tareas que corresponden a las fases del modelo en cascada. Cada tarea estará vinculada a un tiempo estimado de desarrollo y será asignada un nivel de prioridad y dependencia, facilitando la gestión del tiempo y recursos.

La planificación temporal se han registrado a través de un registro de las tareas en un jira que se ha creado al que se puede acceder con el siguiente link:

<https://e-itaca.atlassian.net/jira/software/projects/KAN/boards/1/timeline?selectedIssue=KAN-50&shared=&atlOrigin=eyJpIjoiMWI3YjRmN2FmNjk2NGRhMDg2ODE0MzhkNDk5MGY3Y2UiLCJwIjoiJ9>



Un tablero de kanban es una herramienta ágil de gestión de proyectos diseñada para ayudar a visualizar el trabajo, limitar el trabajo en curso y maximizar la eficiencia (o el flujo). Puede ayudar tanto a los equipos ágiles como a los de DevOps a definir el orden de su trabajo diario. Los tableros de kanban utilizan tarjetas, columnas y la mejora continua para ayudar a los equipos tecnológicos y de servicios a comprometerse con la cantidad de trabajo adecuada y, por supuesto, a llevarla a cabo.



Tablero KAN

The Kanban board displays the following tasks:

- Por Hacer:**
 - + Crear incidencia
- BLOQUEO 4:**
 - Tarea 8.1 (RF8): Diseñar la interfaz de usuario para la pantalla del usuario enfermero.
RF8: PANTALLA USUARIO ENFERMERO
KAN-35
 - Tarea 8.2 (RF8): Implementar el acceso y modificación del historial clínico de los residentes.
RF8: PANTALLA USUARIO ENFERMERO
KAN-36
 - Tarea 8.3 (RF8): Desarrollar la gestión de stock de medicamentos y decisión de dosis.
RF8: PANTALLA USUARIO ENFERMERO
KAN-37
 - Tarea 8.4 (RF8): Probar la funcionalidad y seguridad de la pantalla del usuario enfermero.
RF8: PANTALLA USUARIO ENFERMERO
KAN-38
- EN CURSO 2:**
 - Tarea 9.2 (RNF1): Revisar y asegurar que la documentación sigue los estándares de codificación y es comprensible.
RNF1: DOCUMENTACIÓN
KAN-41
 - Tarea 9.3 (RNF1): Revisar y asegurar que la documentación del proyecto para la entrega final esté actualizada.
RNF1: DOCUMENTACIÓN
KAN-42
- LISTO 28:**
 - Tarea 11.1 (RNF3): Asegurar compatibilidad con la versión de Android que abarque al menos el 85% de los dispositivos.
RNF3: MANTENIMIENTO
KAN-48
 - Tarea 7.2 (RF7): Desarrollar las funcionalidades de gestión de usuarios (agregar, eliminar, modificar).
RF7: PANTALLA ADMINISTRADOR
KAN-29
 - Tarea 7.5 (RF7): Integrar la funcionalidad para añadir y modificar datos del historial clínico de los residentes.
RF7: PANTALLA ADMINISTRADOR
KAN-32
 - Tarea 14.3: Poblar la base de datos con datos de prueba. Esto incluye ingresar datos ficticios que permitan realizar pruebas efectivas de la aplicación.
CREACIÓN Y CONFIGURACIÓN DE LA BASE DE DATOS
KAN-60
 - Tarea 1.3 (RF1): Implementar validaciones de datos en el formulario de registro.
RF1: REGISTRO DE EMPLEADOS
KAN-4
 - Tarea 10.1 (RNF2): Implementar mecanismos de cifrado de datos.
RNF2: SEGURIDAD
KAN-44

2.5. Presupuesto (gastos, ingresos, beneficio) (formato formal)

2.5.1. Resumen del Presupuesto

El presupuesto del proyecto "G.A.B.I" cubre todos los costos asociados al desarrollo, lanzamiento y mantenimiento de la aplicación, así como los ingresos esperados por las licencias de uso y otros posibles canales de monetización. Este presupuesto también incluye un cálculo del beneficio neto esperado tras la deducción de los gastos totales.

Justificación General: Los montos especificados son estimaciones basadas en los costos promedio de mercado para pequeños proyectos de desarrollo de software.



Este presupuesto está basado en que la mayoría del desarrollo es manejado por una sola persona y al no ser un cliente real es meramente informativo para la calificación

2.5.2. Gastos

Los gastos del proyecto se categorizan en desarrollo, herramientas y materiales, marketing y promoción, y otros gastos operativos.

| Categoría | Descripción | Costo Estimado |
|---------------------------|--|----------------|
| Desarrollo | Salarios, subcontrataciones, formación | €15,000 |
| Herramientas y Materiales | Licencias de software, hardware | €3,000 |
| Marketing y Promoción | Publicidad, material promocional | €2,000 |
| Gastos Operativos | Servicios de internet, mantenimiento | €1,000 |
| Total de Gastos | | €21,000 |

1. Desarrollo (€15,000)

- Salarios y Subcontrataciones: Aunque el proyecto se desarrollará individualmente, este costo podría incluir la posible subcontratación de expertos específicos como un diseñador UX/UI o un consultor de seguridad, que son comunes en el desarrollo de aplicaciones para asegurar la calidad y la conformidad con los estándares de la industria.
- Formación: Costos relacionados con la actualización de habilidades o la adquisición de nuevas competencias necesarias para el proyecto, como cursos de especialización en nuevas tecnologías o herramientas de desarrollo.

2. Herramientas y Materiales (€3,000)

- Licencias de Software: Se incluyen los costos de licencias de software esenciales para el desarrollo, como sistemas de gestión de bases de datos, herramientas de diseño y desarrollo, y plataformas de prueba.
- Hardware: Aunque no se mencionó específicamente, este presupuesto puede incluir la adquisición o actualización de hardware necesario para el desarrollo y prueba de la aplicación, como computadoras o dispositivos móviles.

3. Marketing y Promoción (€2,000)



- Publicidad: Incluye la inversión en campañas de marketing digital y tradicional para promover la aplicación entre las residencias de ancianos y otros posibles clientes.
- Material Promocional: Producción de materiales físicos y digitales para apoyar las acciones de marketing, como folletos, carteles, y material para conferencias o presentaciones.

4. Gastos Operativos (€1,000)

- Servicios de Internet: Costos asociados con el acceso a internet necesario para el desarrollo y pruebas de la aplicación.
- Mantenimiento: Incluye gastos recurrentes para el mantenimiento de equipos y software, así como cualquier servicio externo necesario para la operación del proyecto.

2.4.3. Ingresos

Los ingresos se proyectan basados en la venta de licencias de la aplicación y otros servicios asociados a la misma.

| Fuente de Ingresos | Descripción | Ingreso Estimado |
|-----------------------|-----------------------------------|------------------|
| Licencias de Uso | Ventas de licencias a residencias | €30,000 |
| Servicios Adicionales | Soporte y actualizaciones | €5,000 |
| Total de Ingresos | | €35,000 |

1. Licencias de Uso (€30,000)

- Estimación de la Base de Clientes: Se considera el número de residencias de ancianos que podrían estar interesadas en adquirir la aplicación. Por ejemplo, si se estima que 10 residencias podrían comprar la licencia en el primer año a un precio de 1000 cada una, esto resultaría en €30,000.
- Investigación de Mercado: Se basa en un análisis preliminar del mercado que según las encuestas y entrevistas con administradores de residencias para evaluar su interés y disposición a pagar por una solución como "G.A.B.I".
- Comparación con Productos Similares: Observar los precios y modelos de suscripción de aplicaciones similares en el mercado para establecer un precio competitivo.

2. Servicios Adicionales (€5,000)



- Soporte y Actualizaciones: Este ingreso se basa en ofrecer servicios adicionales post-venta, como soporte técnico, actualizaciones y nuevas funcionalidades que pueden ser necesarias o requeridas por las residencias.
- Modelo de Servicio: Se puede considerar un modelo de suscripción anual para estos servicios, por ejemplo, un 10% del costo de la licencia original, lo que incentiva a las residencias a mantener el software actualizado y recibir soporte. Aunque este apartado no es prioritario debido a que también se podría establecer un servicio a parte de puesta a punto cada 3 meses
- Estimación Conservadora: Dado que no todas las residencias podrían optar por estos servicios adicionales inmediatamente, se estima una adopción por parte de un porcentaje de la base de clientes inicial. Segundo los administradores no estarían muy interesados en adquirir este tipo de servicios.

2.5.4. Beneficios

| Descripción | Cálculo | Monto |
|----------------|-----------------------------------|-----------------------------|
| Beneficio Neto | Ingresos Totales - Gastos Totales | €35,000 - €21,000 = €14,000 |

La estimación de €35,000 en ingresos totales refleja una adopción conservadora del producto en el primer año. Estas cifras son ajustables basadas en la respuesta inicial del mercado y la efectividad de las estrategias de marketing y ventas.

El establecimiento de precios se basa en un equilibrio entre ser muy competitivos en el mercado y ofrecer un valor claro a las residencias de ancianos, destacando beneficios como la mejora en la eficiencia operativa y la calidad del cuidado.

Posteriormente se puede hacer un reajuste en los precios dependiendo de las innovaciones que se pueden llegar a aplicar a este software, así como también se pueden encontrar casos en que se requiera hacer modificaciones al software para que la compatibilidad con la realidad del trabajo realizadas en algunas residencias se adapte de forma personalizada.



2.5.5. Comentarios

- **Desarrollo:** Los costos incluyen el desarrollo de software, pruebas y ajustes. Este presupuesto asume que el proyecto será completado dentro del tiempo estimado sin requerir personal adicional.
- **Herramientas y Materiales:** Se contempla la compra de software de desarrollo y hardware necesario para pruebas y desarrollo.
- **Marketing y Promoción:** Inversiones iniciales en marketing digital y material promocional para alcanzar a la base de clientes objetivo.
- **Gastos Operativos:** Incluye gastos mensuales recurrentes necesarios para el funcionamiento del proyecto.
- **Ingresos:** Basados en un análisis de mercado preliminar y las proyecciones de ventas según el número de residencias de ancianos en el área de impacto y la competitividad del precio de la licencia.

Este presupuesto está diseñado para proporcionar una estimación del coste total del proyecto y su potencial de ingresos. Sin embargo, es importante monitorear y ajustar este presupuesto a medida que el proyecto avanza y se obtiene más información del mercado y del desarrollo del software.

2.5.6 Propuesta de presupuesto

| Tarea | Horas | | Costo Total Ajustado (€) | Total con IVA Ajustado (€) | |
|-----------------------------|-----------|----------------|-----------------------------|-------------------------------|--------------|
| | Ajustadas | Costo/Hora (€) | | IVA (21%) | Ajustado (€) |
| RF1: Registro de Empleados | 12.8 | 20 | 256 | 0 | 256 |
| RF2: Registro de Residentes | 9.6 | 20 | 192 | 0 | 192 |
| RF3: Inicio de Sesión | 6.4 | 20 | 128 | 0 | 128 |



| | | | | | |
|----------------------------------|------|----|------|-----|------|
| RF4: Cierre de Sesión | 3.2 | 20 | 64 | 0 | 64 |
| RF5: Baja de Usuario y Residente | 6.4 | 20 | 128 | 0 | 128 |
| RF6: Cuenta de Usuario | 12.8 | 20 | 256 | 0 | 256 |
| RF7: Pantalla Administrador | 19.2 | 20 | 384 | 0 | 384 |
| RF8: Pantalla Usuario Enfermero | 16 | 20 | 320 | 0 | 320 |
| RNF1: Documentación | 20 | 20 | 400 | 0 | 400 |
| RNF2: Seguridad | 16 | 20 | 320 | 0 | 320 |
| RNF3: Mantenimiento | 8 | 20 | 160 | 0 | 160 |
| RNF4: Interfaz y Usabilidad | 11.2 | 20 | 224 | 0 | 224 |
| RNF5: Rendimiento | 6.4 | 20 | 128 | 0 | 128 |
| Google Ads | - | - | 1000 | 210 | 1210 |
| Propaganda YouTube | - | - | 500 | 105 | 605 |
| SUBTOTAL | 148 | - | 6960 | 315 | 7275 |



2.6. Contrato (y Pliego de condiciones) o Licencia de distribución (según proceda)

2.6.1. Definiciones

- "Licenciatario": Elton Saravia Ibarra, el desarrollador y propietario de la aplicación "G.A.B.I".
- "Licenciado": Cualquier entidad, como una residencia de ancianos, que adquiera derechos para utilizar la aplicación bajo los términos de esta licencia.
- "Software": La aplicación móvil "G.A.B.I", que incluye todas sus funcionalidades, documentación asociada, y actualizaciones o modificaciones futuras

2.6.2. Concesión de la Licencia

2.6.2.1. Uso del Software

- Licencia otorgada: Se concede al Licenciado una licencia no exclusiva, no transferible y no sublicenciable para utilizar el Software únicamente en sus instalaciones y dispositivos permitidos, limitada específicamente al número de licencias adquiridas.
- Duración: La licencia es válida por un periodo indefinido, sujeta a los términos de cumplimiento y terminación establecidos en este contrato.
- Restricciones de uso: El Licenciado podrá utilizar el Software exclusivamente para fines internos y operacionales dentro de su residencia de ancianos y no podrá utilizar el Software para ofrecer servicios a terceros ni para operar un servicio de procesamiento de datos o similar para terceros.

2.6.2.2. Restricciones

- Modificaciones y adaptaciones: El Licenciado no podrá modificar, adaptar, alterar, traducir, o crear obras derivadas basadas en el Software. Cualquier intento de modificación será considerado una violación del contrato y resultará en la terminación inmediata de la licencia.
- Prohibición de ingeniería inversa: Está estrictamente prohibido realizar ingeniería inversa, descompilar, o desensamblar el Software, excepto y solo en la medida en que tal actividad esté expresamente permitida por la ley aplicable a pesar de esta limitación.
- Restricciones de sublicencia y alquiler: El Licenciado no podrá sublicenciar, alquilar, prestar, arrendar ni transferir el Software o cualquier parte del mismo a ningún tercero sin el consentimiento previo por escrito del Licenciatario.



- Copias de seguridad: El Licenciatario podrá hacer una cantidad razonable de copias del Software únicamente para fines de copia de seguridad o de archivo.

2.6.3. Propiedad Intelectual

- **Titularidad:** El Licenciante, Elton Saravia Ibarra, retiene en todo momento la titularidad y propiedad exclusiva del Software "G.A.B.I", incluyendo pero no limitado a, todos los derechos de autor, patentes, secretos comerciales, marcas registradas y otros derechos de propiedad intelectual inherentes o relacionados con el Software.
- **Registro y Protección:** El Software y todas sus partes constituyentes están protegidos por las leyes de derechos de autor nacionales e internacionales, así como por tratados internacionales. El Software, incluyendo su código, documentación, apariencia, estructura y organización, es una obra protegida por derechos de autor propiedad de Elton Saravia Ibarra.
- **Restricciones de Marca:** El Licenciatario no está autorizado a utilizar el nombre comercial, marcas registradas, logotipos o marcas de servicio de "G.A.B.I" sin el permiso previo y por escrito del Licenciante. Cualquier uso permitido deberá adherirse a las directrices de marca establecidas por el Licenciante.
- **Copias No Autorizadas:** Está estrictamente prohibido realizar copias no autorizadas del Software o de cualquier parte de este. Todas las copias realizadas conforme a este acuerdo deben contener las mismas notificaciones de propiedad intelectual y otros avisos de propiedad que aparecen en el Software original.
- **Seguridad y Confidencialidad:** El Licenciatario se compromete a tomar todas las medidas razonables para proteger el Software de accesos o usos no autorizados, y a mantener la confidencialidad de todas las tecnologías o información relacionada con el Software que el Licenciante haya marcado como confidencial.

2.6.4. Duración de la Licencia

- **Término:** Esta licencia está vigente desde la fecha de activación del Software por el Licenciatario y permanecerá en efecto hasta que sea terminada por cualquiera de las partes. El Licenciante puede terminar la licencia de forma inmediata si el Licenciatario no cumple con cualquier término o condición de este acuerdo.

2.6.5. Garantías y Renuncias

- **Garantías:** El Licenciante asegura que tiene los derechos necesarios para otorgar esta licencia.



- **Renuncia de Garantías:** El Software se proporciona "tal cual", sin garantías de funcionamiento ininterrumpido o libre de errores. El Licenciatario no garantiza que el Software satisfaga completamente las necesidades del Licenciatario.

2.6.6. Limitación de Responsabilidad

- **Limitaciones:** El Licenciatario no será responsable ante el Licenciatario por cualquier daño especial, incidental, indirecto o consecuente (incluidos, entre otros, daños por pérdida de beneficios, interrupción del negocio, pérdida de información empresarial, o cualquier otra pérdida pecuniaria) surgidos del uso o la incapacidad de usar el Software, incluso si el Licenciatario ha sido advertido de la posibilidad de tales daños.

2.7. Análisis de riesgos (DAFO)

2.7.1. Fortalezas

Especialización del Producto: "G.A.B.I" está diseñada específicamente para residencias de ancianos, lo que ofrece una solución personalizada a los problemas de gestión específicos de este sector.

Automatización y Eficiencia: La aplicación mejora significativamente la eficiencia en la gestión de las tareas diarias y el cuidado de los residentes.

Facilidad de Uso: Interfaz intuitiva que facilita su adopción por parte del personal de las residencias de ancianos, incluso para aquellos con limitada experiencia tecnológica.

: Diseñada para cumplir con RGPD y otras normativas importantes, proporcionando seguridad y confianza a los usuarios.

2.7.2. Debilidades

Recursos Limitados: Siendo un proyecto individual, los recursos para desarrollo, marketing y soporte son limitados.

Dependencia de Proveedores de Terceros: Para algunas funciones claves como el hosting o servicios de nube, el proyecto depende de terceros, lo que puede implicar riesgos en la disponibilidad o cambios en los costos.

Falta de Reconocimiento de Marca: "G.A.B.I" será nuevo en el mercado y carece de reconocimiento de marca, lo cual puede ser un obstáculo inicial para la adquisición de clientes.

2.7.3. Oportunidades

Creciente Demanda de Tecnología en Salud: Existe una tendencia creciente hacia la digitalización de la gestión en el sector salud, incluyendo las residencias de ancianos.

Subvenciones y financiación: Posibilidad de obtener subvenciones o incentivos gubernamentales para proyectos que mejoran la calidad del cuidado de los ancianos.

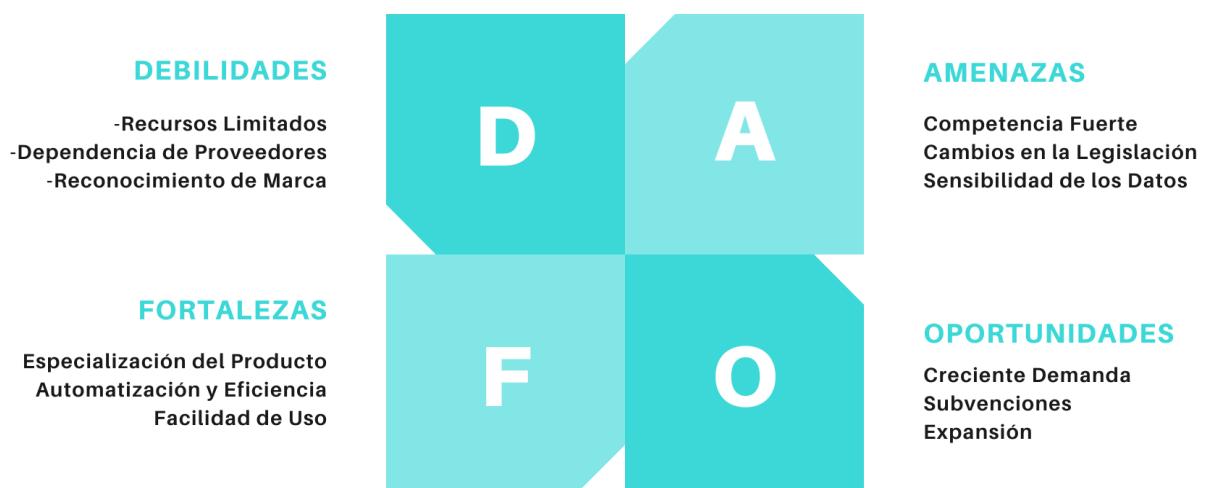
Expansión a Nuevos Mercados: Potencial para adaptar y expandir la aplicación a otros sectores de la salud o a nuevos mercados geográficos.

2.7.4. Amenazas

Competencia Fuerte: Existen grandes competidores con más recursos y presencia establecida en el mercado.

Cambios en la Legislación: Cambios regulatorios podrían aumentar los costos operativos o requerir ajustes significativos en el software.

Sensibilidad de los Datos: Riesgos asociados al manejo de datos personales y de salud, incluyendo potenciales brechas de seguridad.





3. ANÁLISIS Y DISEÑO

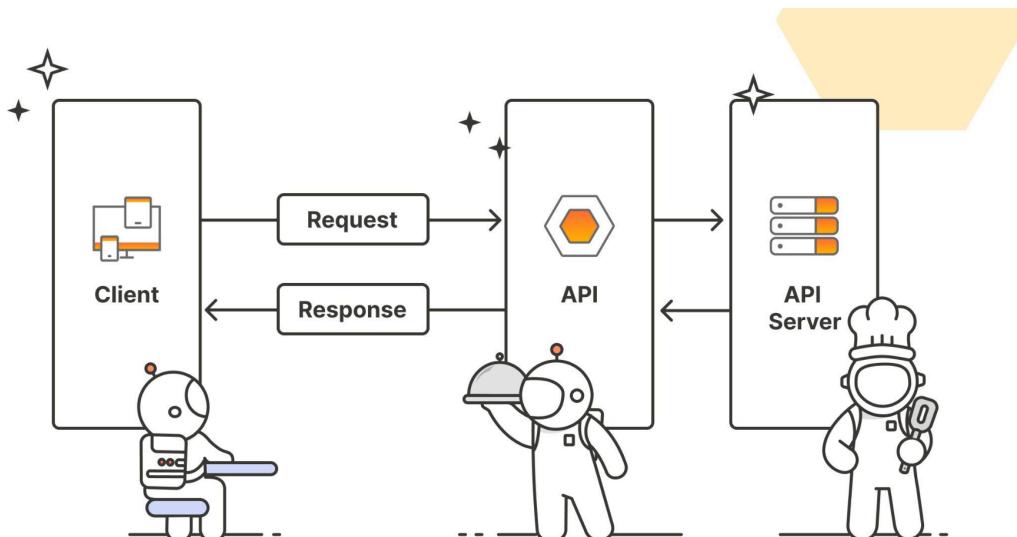
3.1.1 Arquitectura de la aplicación

La arquitectura cliente-servidor es la elegida debido a las múltiples ventajas que tiene este modelo de desarrollo. No existe la necesidad de almacenar datos en los móviles de los usuarios.

La arquitectura se basará en llamadas a APIs (Interfaz de Programación de Aplicaciones) para acceder a datos, funcionalidades o recursos externos.

Las API desempeñan un papel crucial en el desarrollo de software, ya que proporcionan numerosos beneficios:

- Reutilización: las API permiten reutilizar el código y la funcionalidad existentes en lugar de crear nuevos componentes desde cero. Esto puede reducir significativamente el tiempo y el esfuerzo de desarrollo.
- Modularidad: las API permiten un enfoque modular de la arquitectura de software al separar la funcionalidad en componentes discretos con límites bien definidos. Esto mejora la estabilidad y el mantenimiento del sistema.
- Interoperabilidad: al estandarizar la forma en que las aplicaciones comparten datos, las API permiten una integración perfecta entre diferentes sistemas, plataformas y lenguajes de programación.
- Escalabilidad: las API pueden ayudar con la escalabilidad de una aplicación al permitir el equilibrio de carga, el almacenamiento en caché y otras técnicas para optimizar el rendimiento y el consumo de recursos en diferentes condiciones de carga.





Las APIs REST (Interfaz de Programación de Aplicaciones Representacional) tienen numerosos beneficios que las hacen muy populares en el desarrollo de software. Aquí están algunos de los más destacados:

- Arquitectura Orientada a Recursos: Las APIs REST están basadas en el concepto de recursos, que son objetos o datos a los que se puede acceder a través de un identificador único (URL). Esto proporciona una estructura clara y coherente para la organización de los datos y operaciones.
- Simplicidad y Facilidad de Uso: Las APIs REST utilizan los métodos HTTP estándar (GET, POST, PUT, DELETE) para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en los recursos. Esto hace que sea fácil entender y utilizar la API, ya que sigue las convenciones establecidas por HTTP.
- Independencia de Plataforma: Las APIs REST son independientes de la plataforma y el lenguaje de programación, lo que significa que pueden ser consumidas por cualquier cliente que pueda hacer solicitudes HTTP. Esto facilita la integración con diferentes tecnologías y sistemas.
- Escalabilidad: Las APIs REST son altamente escalables, ya que pueden manejar grandes cantidades de tráfico y solicitudes simultáneas. Al seguir los principios de la arquitectura REST, como la separación de estado y la caché, se pueden mejorar aún más las capacidades de escalabilidad.
- Flexibilidad: Las APIs REST permiten el acceso a diferentes tipos de datos, incluyendo texto, JSON, XML, imágenes, etc. Esto proporciona flexibilidad para adaptarse a las necesidades específicas de diferentes clientes y aplicaciones.
- Desacoplamiento Cliente-Servidor: Las APIs REST promueven el desacoplamiento entre el cliente y el servidor, lo que significa que los cambios en una parte del sistema no afectan necesariamente a la otra. Esto facilita la evolución y mantenimiento independiente de las partes del sistema.
- Visibilidad y Portabilidad: Las APIs REST son auto-descriptivas, lo que significa que la información sobre los recursos y operaciones está integrada en la propia API. Esto hace que sea fácil de entender y utilizar, y también facilita la portabilidad de la API entre diferentes entornos y aplicaciones.
- Facilita el Desarrollo de Aplicaciones Web y Móviles: Las APIs REST son ideales para el desarrollo de aplicaciones web y móviles, ya que proporcionan una forma sencilla y eficiente de acceder a datos y funcionalidades a través de internet.



3.1.2 Componentes clave de la arquitectura

- **Interfaz de Usuario (UI):** Debe ser intuitiva y adaptada para facilitar el uso por parte del personal de las residencias, con una navegación sencilla que permita a los usuarios realizar tareas sin complicaciones y acceder a la información necesaria con eficiencia.
- **Servicios Backend:** Deberán incluir la lógica de negocio, la gestión de la base de datos y la integración con otros sistemas. Esto también abarca la autenticación y la autorización de usuarios, la gestión de roles y el cifrado de datos para cumplir con las normativas de privacidad y protección de datos.
- **Base de Datos:** Diseñar una base de datos robusta y segura, preferiblemente utilizando un servicio de base de datos en la nube que ofrece escalabilidad y disponibilidad. Se deben establecer esquemas claros y eficientes que permitan consultas rápidas y almacenamiento seguro de datos sensibles.
- **Almacenamiento en la Nube:** Para el almacenamiento de imágenes, historiales clínicos y otros documentos importantes. El uso de servicios en la nube facilitará el acceso y la gestión de estos archivos de manera segura y conforme a la normativa.

3.2. Tecnologías/Herramientas usadas y descripción de las mismas

3.2.1. Frontend - Android Studio:

- **Descripción:** Se utilizará Android Studio para el desarrollo de la interfaz de usuario, aprovechando sus amplias librerías y el soporte para el diseño de interfaces adaptativas y compatibles con diferentes dispositivos y versiones de Android.
- **Justificación:** Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android, proporcionando herramientas de depuración, simulación y pruebas necesarias para un desarrollo eficiente.

3.2.2. Backend - MySQL:

- **Descripción:** MySQL servirá como el sistema de gestión de bases de datos para almacenar y gestionar datos de los usuarios, residentes, y registros clínicos.
- **Justificación:** MySQL es un sistema de base de datos relacional robusto y eficiente, ideal para manejar las transacciones complejas que son habituales



en una aplicación de gestión de residencias. Su escalabilidad y seguridad lo hacen una elección sólida para este proyecto.

3.2.3. Backend - PHP:

- **Descripción:** PHP será utilizado para la creación de la lógica del servidor, gestionando las solicitudes de los clientes, procesando datos y generando respuestas en formato JSON. Este lenguaje se empleará para manejar la autenticación, la gestión de usuarios, y la interacción con la base de datos MySQL.

Justificación: PHP es un lenguaje de scripting del lado del servidor ampliamente utilizado y especialmente adecuado para el desarrollo web. Su facilidad de integración con MySQL y su capacidad para gestionar solicitudes HTTP lo hacen ideal para este proyecto. Además, PHP proporciona un ecosistema robusto con múltiples frameworks y bibliotecas que facilitan el desarrollo rápido y seguro de APIs RESTFUL.

3.2.4. Gestión de Sesiones - RBAC API:

- **Descripción:** Para la autenticación de usuarios se implementará un sistema basado en tokens utilizando la API de login, la cual generará tokens de tiempo limitado para el inicio de sesión seguro.
- **Justificación:** El crear la API de tokens permite flexibilidad para la comunicación con otras APIs que requieran ese token a demás de devolver el rol correcto para mostrar solo las pantallas a las que puede acceder ese usuario.

3.2.5. Firebase - Realtime Database:

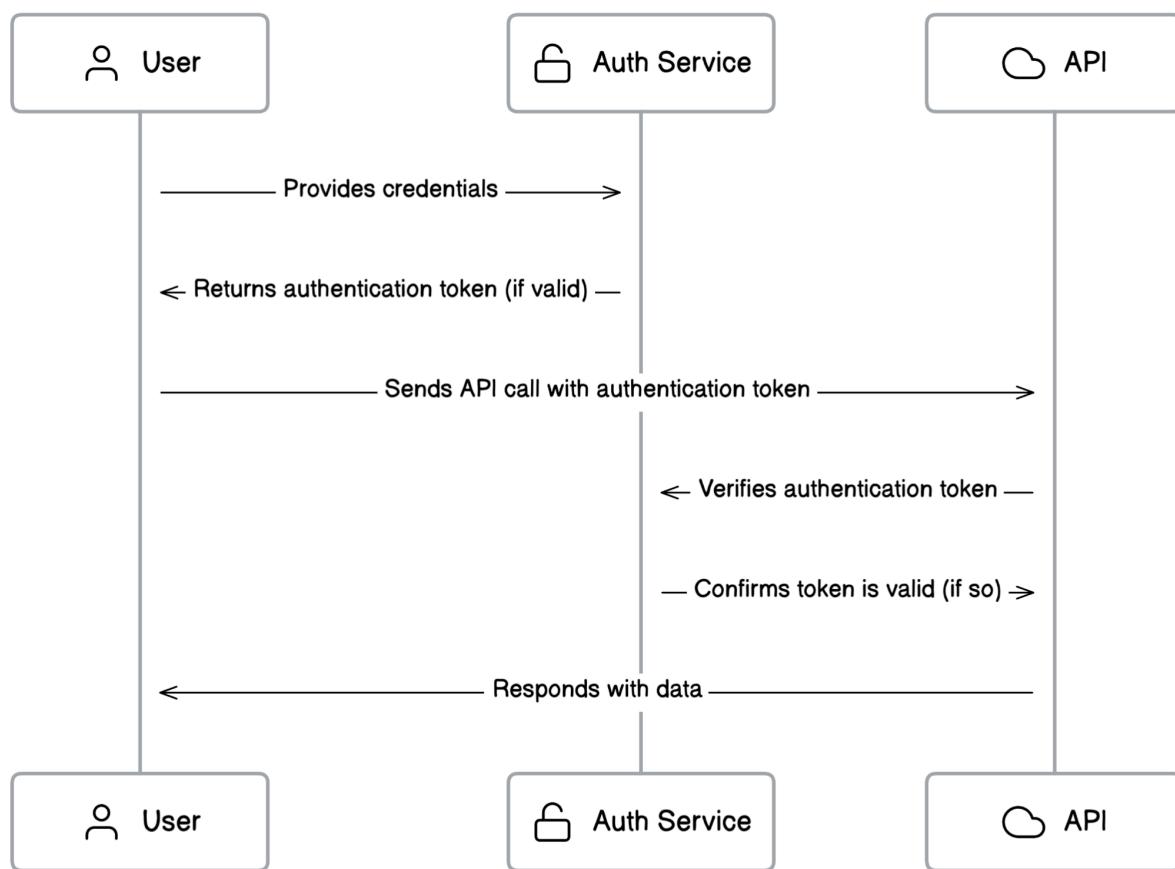
- **Descripción:** Firebase Realtime Database se utilizará para implementar la funcionalidad de chat en tiempo real entre los empleados. Esta base de datos NoSQL en tiempo real permite la sincronización instantánea de datos entre los clientes y la base de datos en la nube, garantizando que todos los usuarios vean las actualizaciones en tiempo real.
- **Justificación:** Firebase Realtime Database ofrece una solución eficiente y escalable para manejar datos en tiempo real, lo cual es crucial para la implementación de funcionalidades de chat. Su integración con Android Studio y su capacidad para gestionar grandes volúmenes de datos sin una configuración de servidor compleja lo hacen ideal para este propósito. Además, Firebase proporciona una serie de herramientas y servicios adicionales que facilitan el desarrollo, la implementación y la gestión de aplicaciones móviles.



3.3. Arquitectura de componentes de la aplicación

La arquitectura de mi aplicación consiste en el sistema de cliente servidor, con esta arquitectura, las tareas se distribuyen entre los servidores (que proveen los servicios) y los clientes (que demandan dichos servicios). Dicho de otro modo: el cliente le pide un recurso al servidor, que brinda una respuesta. Para esto crearé una sistema que funcione con API para que respondan a las necesidades del los usuarios sin tener que preocuparse por sobrecargar los dispositivos en local o almacenar información útil en el momento del uso pero no necesariamente util a lo largo del tiempo.

Una de las ventajas menos aparentes de la organización en servidores y clientes es que la capacidad de procesamiento y memoria de estos últimos no debe ser tan grande como la de los primeros, lo cual beneficia al consumidor final permitiéndole usar un equipo relativamente antiguo para disfrutar de servicios generalmente muy avanzados.



Para acceder a cada servicio se necesita un token, por lo cual esto maximiza la seguridad, este token sera variable y expirara p, para eso se utilizara JWT instalado lo necesario en el servidor, al tramitar toda la información a través de la base de datos al dispositivo estas solicitudes deben ser securizadas y por ello se usa el el x-www-form-urlencoded.



x-www-form-urlencoded es especialmente ventajoso cuando se integra con APIs PHP y posteriormente con bases de datos MySQL en una arquitectura cliente-servidor. En PHP, los datos enviados mediante x-www-form-urlencoded se acceden fácilmente usando superglobales como `$_POST`. Esto significa que los datos del formulario se pueden procesar de inmediato sin necesidad de decodificación adicional. PHP está diseñado para manejar datos x-www-form-urlencoded de manera nativa, lo que reduce la complejidad en la configuración del servidor. No se requieren librerías adicionales para procesar este tipo de datos.

Cuando se reciban datos a través de las APIs, los datos obtenidos mediante `$_POST`, pueden ser directamente utilizados en consultas SQL después de ser validados y sanitizados, lo cual es esencial para prevenir inyecciones SQL.

```
$dni = $_POST['dni'] ?? null;
$nombre = $_POST['nombre'] ?? null;
$apellidos = $_POST['apellidos'] ?? null;
$fecha_nacimiento = $_POST['fecha_nacimiento'] ?? null;
$ar = $_POST['ar'] ?? null;
$nss = $_POST['nss'] ?? null;
$numero_cuenta_bancaria = $_POST['numero_cuenta_bancaria'] ?? null;
$observaciones = $_POST['observaciones'] ?? null;
$activo = $_POST['activo'] ?? "true";
$empadronamiento = $_POST['empadronamiento'] ?? null;
$edad = $_POST['edad'] ?? 1;
$mes_cumple = $_POST['mes_cumple'] ?? 1;
$foto = $_POST['foto'] ?? null;
$estado = $_POST['estado'] ?? 1;
$telefono = $_POST['telefono'] ?? null;
$email = $_POST['email'] ?? null;
$fecha_ingreso = date('Y-m-d H:i:s'); // Obtener la fecha y hora actual

if (isset($dni, $nombre, $apellidos, $ar, $nss, $fecha_nacimiento)) {
    $hashedDNI = encrypt($dni, $encryptionKey);
    $hashedAR = encrypt($ar, $encryptionKey);
    $hashedNSS = encrypt($nss, $encryptionKey);
    $hashedCuenta = encrypt($numero_cuenta_bancaria ?? "No hay", $encryptionKey);
    $hashedEmpadronamiento = encrypt($empadronamiento ?? "No hay", $encryptionKey);
    $fotoData = $foto ? $foto : null;

    $telefono = $telefono ?? "No hay";
    $email = $email ?? "No hay";
    $observaciones = $observaciones ?? "No hay";

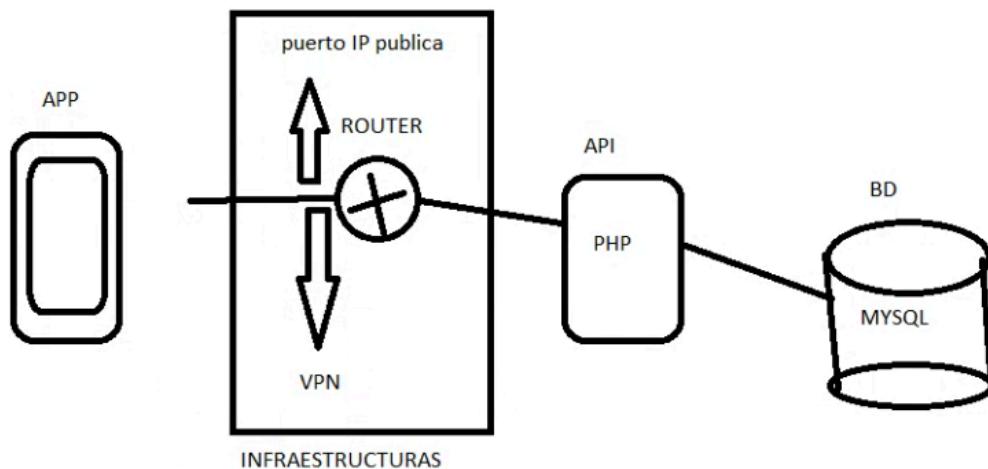
    $sql = "INSERT INTO residentes (dni, nombre, apellidos, fecha_nacimiento, ar, nss,
        numero_cuenta_bancaria, observaciones, activo, empadronamiento, edad, mes_cumple,
        foto, estado, telefono, email, fecha_ingreso) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
        ?, ?, ?, ?, ?, ?, ?)";
    $stmt = $conn->prepare($sql);

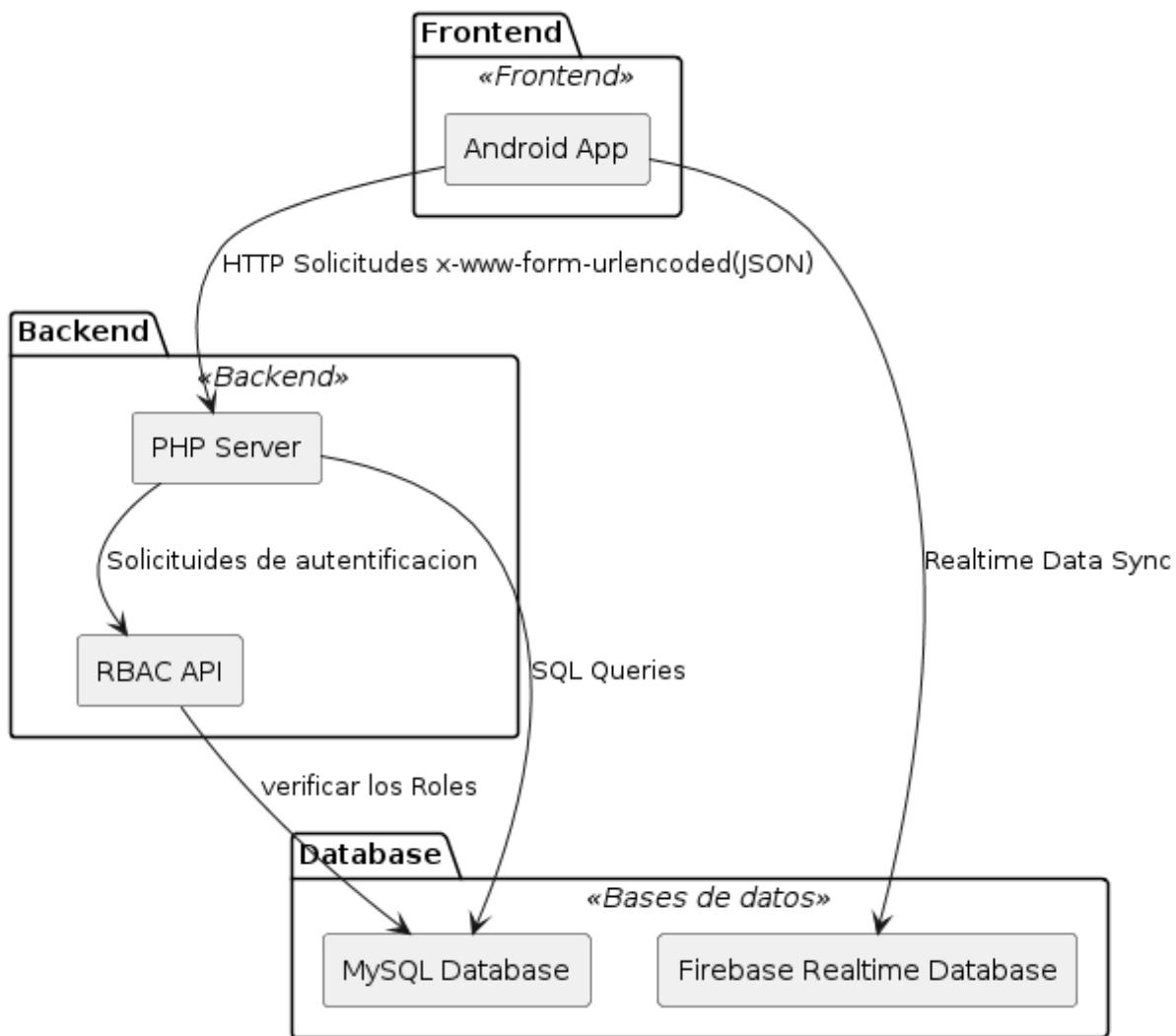
    if ($stmt === false) {
        echo json_encode(array("status" => "error", "message" => "Error en la preparación
            de la consulta: " . $conn->error));
        exit();
    }
}
```



La estructura de pares clave-valor de x-www-form-urlencoded se mapea fácilmente a columnas de una base de datos, simplificando el proceso de inserción y actualización de registros. Además lo principal es la seguridad y aunque x-www-form-urlencoded es fácil de usar, es crucial usar HTTPS para proteger los datos transmitidos entre el cliente y el servidor, evitando la interceptación y manipulación de datos por tercero.

En cuanto a la seguridad, uno de los aspectos más importantes es el uso de HTTPS (Hypertext Transfer Protocol Secure). HTTPS encripta los datos transmitidos entre el cliente y el servidor, lo que impide que terceros intercepten o manipulen la información durante su transmisión. Esta capa de seguridad es fundamental para proteger la confidencialidad e integridad de los datos, especialmente en aplicaciones que manejan información sensible, como datos personales o financieros. Implementar HTTPS ayuda a garantizar que los datos enviados mediante x-www-form-urlencoded no sean vulnerables a ataques como el "man-in-the-middle", donde un atacante podría interceptar y alterar la comunicación entre el cliente y el servidor.





3.3.1 Backend

Para la gestión de una residencia, una base de datos relacional (como MySQL con InnoDB) es preferible a una base de datos NoSQL por varias razones fundamentales.

En primer lugar, la estructura de los datos en una residencia incluye múltiples entidades relacionadas entre sí, tales como residentes, trabajadores, medicamentos, tratamientos y citas. Las bases de datos relacionales están específicamente diseñadas para manejar este tipo de datos con sus tablas y relaciones bien definidas. El uso de claves foráneas permite garantizar que, por ejemplo, cada medicamento registrado esté vinculado a un residente válido, ayudando a mantener la integridad de los datos y prevenir inconsistencias.

Además, las bases de datos relacionales soportan transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), lo que asegura que todas las operaciones de la base de datos se realicen de manera segura y coherente. Por ejemplo, cuando



se administra un medicamento, es necesario actualizar varias tablas (registro del medicamento, inventario, historial del residente, etc.) y asegurar que todas estas operaciones se completen correctamente. Si ocurre un fallo durante el proceso, el sistema puede revertir los cambios para mantener la integridad de los datos, lo cual es crucial en aplicaciones donde la precisión de los datos es vital.

Otra ventaja significativa es la capacidad de realizar consultas complejas. SQL permite ejecutar consultas muy específicas y detalladas que permiten obtener exactamente la información necesaria. Por ejemplo, se puede obtener fácilmente una lista de todos los residentes que tienen citas médicas en la próxima semana, o buscar todos los medicamentos administrados a un residente específico en los últimos tres meses. SQL está optimizado para este tipo de consultas, permitiendo obtener resultados rápidos y precisos.

La base de datos está diseñada para gestionar una residencia, cubriendo diversos aspectos cruciales como la administración de residentes, trabajadores, medicamentos y control de actividades. La estructura de esta base de datos es para asegurar que la mayoría de las operaciones de la residencia se registren y se gestionen de manera sencilla.

En el centro de la base de datos se encuentran las tablas relacionadas con los residentes, como Residentes, Familiares y Habitaciones. La tabla Residentes almacena información personal y detalles importantes sobre cada residente, mientras que Familiares mantiene un registro de los familiares asociados a cada residente, lo que facilita la comunicación y el seguimiento de contactos de emergencia. La tabla Habitaciones asigna a cada residente una habitación específica, permitiendo un control preciso de la ocupación de la residencia.

La gestión de trabajadores es otra parte esencial de la base de datos. La tabla Trabajadores contiene información sobre los empleados, incluyendo sus datos personales y credenciales para el uso de la aplicación. Se crea una tabla para el manejo de turnos, permitiendo una asignación clara de tareas y horarios de trabajo. Además, el registro de entradas y salidas de trabajadores se mantiene en la tabla Registro_Entradas_Salidas, lo cual puede llegar a ser una información valiosa aunque de momento no se refleja esta información en la aplicación en el futuro se podrán crear informes.

El cuidado de los residentes también se ve reflejado en las tablas Tratamiento, Medicamentos y AdministracionMedicamentos. La tabla Tratamiento detalla los planes de tratamiento individuales de cada residente, incluyendo los medicamentos prescritos, que se especifican en la tabla Medicamentos. La administración de estos



medicamentos se controla mediante la tabla AdministracionMedicamentos, asegurando que cada residente recibe su medicación según lo prescrito.

Estas tablas están pensadas para el rol de auxiliar, para que pueda llevar un control en un futuro cuando se desarrolle ese apartado y se pueda combinar con un módulo de inventario.

Otro aspecto importante es el control de actividades y citas. La tabla Controles registra las actividades diarias de los residentes que tengan algún enlace con un trabajador (aunque de momento solo se desarrolla la tabla tareas qué tareas genéricas que abarcan a varios residentes).

La tabla CitasExternas mantiene un seguimiento de las citas médicas o externas que los residentes deben atender. Esto permite una organización eficiente y un seguimiento de las actividades de cada residente que sean casos excepcionales o a tener en cuenta, por ejemplo una operación, rehabilitación fuera de la residencia o simplemente marcar como que el residente no dormirá en la residencia durante algunos días.

La seguridad de la base de datos es fundamental, especialmente cuando se integran solicitudes mediante APIs PHP en una aplicación cliente-servidor desarrollada en Android con Java. Es crucial implementar medidas de validación y sanitización de datos para evitar vulnerabilidades como inyecciones SQL. Además, el uso de HTTPS para cifrar la comunicación entre el cliente y el servidor garantiza que los datos personales y médicos se mantengan seguros durante su transmisión.

Elegir MySQL alojado en Hostinger, corriendo con InnoDB, es una elección tomada por varias razones que benefician a la aplicación cliente-servidor.

Primero, MySQL es uno de los sistemas de gestión de bases de datos más populares y ampliamente utilizados en el mundo. Su popularidad se debe a su fiabilidad, rendimiento y flexibilidad. Al optar por MySQL, te aseguras de estar utilizando una base de datos que tiene un sólido historial de estabilidad y eficiencia.

Hostinger, como proveedor de alojamiento, ofrece una infraestructura optimizada para bases de datos MySQL. Esto incluye servidores rápidos y fiables, soporte técnico disponible las 24 horas y herramientas de gestión fáciles de usar. Con Hostinger, aprovechamos un entorno seguro y eficiente para alojar la base de datos, lo que garantiza un rendimiento óptimo y tiempos de inactividad mínimos.

InnoDB, el motor de almacenamiento elegido, ofrece varias ventajas críticas para aplicaciones que manejan datos sensibles y requieren alta disponibilidad. InnoDB soporta transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), lo que asegura que todas tus operaciones de base de datos sean seguras y



coherentes. Esto es especialmente importante en aplicaciones donde la integridad de los datos es crucial, como en la gestión de una residencia.

Además, InnoDB proporciona soporte para claves foráneas y restricciones de integridad referencial, lo que te permite mantener relaciones consistentes entre las tablas de tu base de datos. Esto garantiza que los datos sean precisos y que las operaciones no violen las reglas de integridad, reduciendo el riesgo de errores y datos inconsistentes.

Otra ventaja significativa de InnoDB es su capacidad de bloqueo a nivel de fila. Esto permite que múltiples transacciones ocurran simultáneamente sin bloquearse entre sí, mejorando la concurrencia y el rendimiento de tu aplicación. En un entorno con múltiples usuarios, como una residencia con varios trabajadores accediendo a la base de datos, esta característica es vital para mantener la eficiencia y la rapidez en las operaciones.

3.3.3 Conexión al Servidor de Hostinger para Configuración de CORS e Instalación de Composer

Para nuestro proyecto de gestión de una residencia, fue necesario configurar el servidor en Hostinger para habilitar CORS y gestionar tokens JWT mediante Composer. A continuación, se describe el proceso detallado de conexión y configuración.

Generación y Configuración de Claves SSH

Paso 1: Generar una Clave SSH

Primero, generé una clave SSH utilizando un tutorial detallado sobre cómo configurar claves SSH. Este proceso incluye la generación de un par de claves pública y privada que sirven como credenciales de acceso seguro.

Paso 2: Añadir la Clave SSH a Hostinger

Después de generar la clave SSH, la añadí a mi servidor VPS desde el hPanel de Hostinger. Navegué a la sección de configuración del VPS, seleccioné la pestaña de claves SSH y añadí la nueva clave.

Conexión al Servidor VPS mediante SSH

Paso 3: Acceso al Servidor

Utilicé la clave SSH para conectarme al servidor VPS. En el hPanel, encontré los detalles de acceso SSH, incluyendo la IP, el puerto y el nombre de usuario.

Paso 4: Conexión mediante Terminal

Abrí una terminal e ingresé el comando de conexión proporcionado

Esto me permitió acceder de manera segura al servidor VPS para realizar las configuraciones necesarias.



Instalación de Composer e Implementación de JWT

Paso 5: Instalación de Composer

Una vez conectado al servidor, instalé Composer, una herramienta esencial para gestionar dependencias en PHP. Ejecuté el siguiente comando para instalar la librería firebase/php-jwt que nos permite crear tokens JWT:

```
composer require firebase/php-jwt
```

Paso 6: Configuración de CORS en PHP

Para habilitar CORS en nuestro servidor, edité el archivo .htaccess en el directorio raíz de nuestro proyecto, añadiendo las siguientes líneas:

```
Header set Access-Control-Allow-Origin "*"
```

```
Header set Access-Control-Allow-Methods "GET, POST, PUT, DELETE, OPTIONS"
```

```
Header set Access-Control-Allow-Headers "Content-Type, Authorization"
```

Esta configuración permite a nuestra API aceptar solicitudes desde cualquier origen, facilitando la comunicación entre el cliente y el servidor.

The screenshot shows a web-based SSH interface. On the left, there's a sidebar with 'Acceso SSH' and a navigation bar with 'Sitios web - residencialoranzan.com - Avanzado - Acceso SSH'. The main area has two tabs: 'Detalles de SSH' and 'Estado de SSH'. In 'Estado de SSH', it says 'INACTIVO' and provides a brief description of what SSH is. Below these are sections for 'Iniciar sesión en SSH' (with Putty and PenguinNet options) and 'Comandos de SSH básicos' (with a command input field). On the right, there's a terminal window titled 'u783570590@fr-int-web990...' showing server details like Model: ProLiant DL325 Gen10 Plus v2, IPMI IP address: 10.4.0.30, and IPMI MAC address: 5c:ed:8c:43:aa:a4. It also shows a failed login attempt log and a welcome message: 'Welcome back! The time now is 20:49 UTC'.



```
-rw-r--r-- 1 u783570590 o200474363 3238 Jun 16 2023 wmlpc.php
[u783570590@fr-int-web990 public_html]$ composer require firebase/php-jwt
Warning from https://repo.packagist.org: Support for Composer 1 is deprecated and some packages will not be available. You should upgrade to Composer 2. See https://blog.packagist.com/deprecating-composer-1-support/
Using version ^6.10 for firebase/php-jwt
./composer.json has been created
Loading composer repositories with package information
Warning from https://repo.packagist.org: Support for Composer 1 is deprecated and some packages will not be available. You should upgrade to Composer 2. See https://blog.packagist.com/deprecating-composer-1-support/
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Installing firebase/php-jwt (v6.10.0): Downloading (100%)
    firebase/php-jwt suggests installing paragonie/sodium_compat (Support EdDSA (Ed25519) signatures when libsodium is not present)
    firebase/php-jwt suggests installing ext-sodium (Support EdDSA (Ed25519) signatures)
Writing lock file
Generating autoload files
[u783570590@fr-int-web990 public_html]$ |
```

```
drwxr-xr-x 2 u783570590 o200474363 4096 May 14 15:48 api
-rw-r--r-- 1 u783570590 o200474363 63 May 15 20:58 composer.json
-rw-r--r-- 1 u783570590 o200474363 2798 May 15 20:58 composer.lock
-rw-r--r-- 1 u783570590 o200474363 16406 Jun 16 2023 default.php
-rw-r--r-- 1 u783570590 o200474363 405 Jun 16 2023 index.php
```

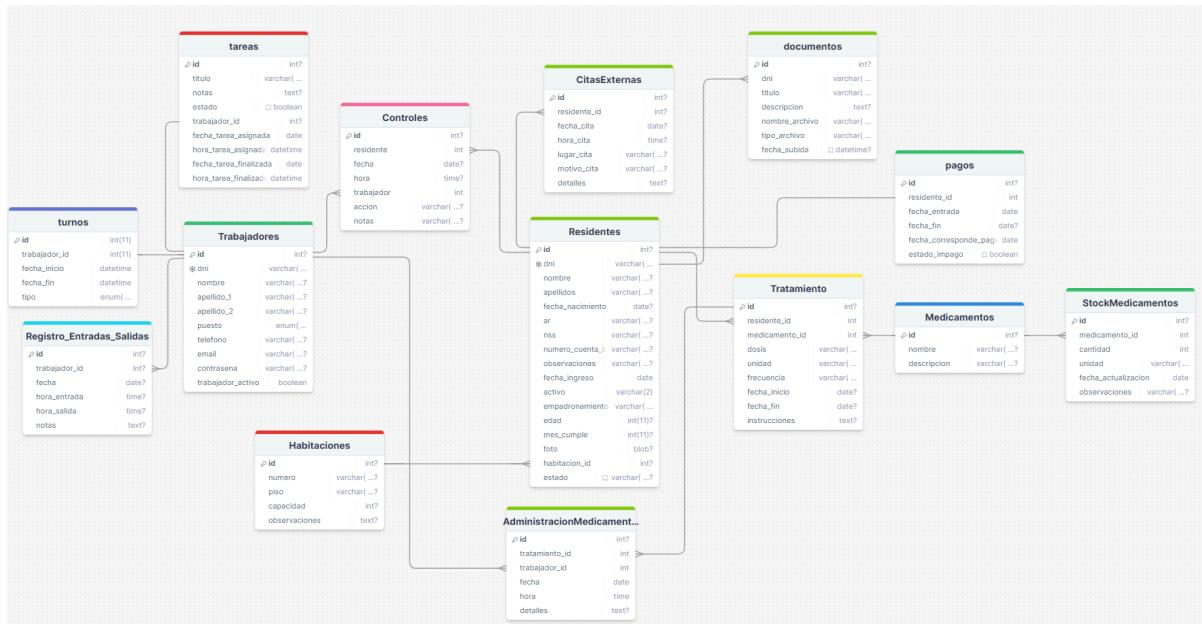
3.4. Modelado de datos

El modelado de datos es una técnica utilizada para definir y analizar los requisitos de datos necesarios para soportar los procesos de negocio dentro del alcance de sistemas de información en organizaciones. Este proceso comienza con la recolección de requisitos y se extiende hasta el diseño lógico y físico de la base de datos.

La base de datos **u783570590_GABI** está diseñada para gestionar la información de una residencia, incluyendo residentes, trabajadores, medicamentos, pagos, citas externas, entre otros.



3.4.1. Tablas y descripción base de datos MySQL



<https://drawsql.app/teams/gabi-8/diagrams/gabi>

Tabla administracion_medicamentos

Esta tabla almacena información sobre la administración de medicamentos a los residentes. Esta tabla debería tener acceso y generar información el perfil de enfermero para mantener un historial de en qué momento se le ha dado la medicación indicada por un médico y quien es el responsable de dicha acción

- id: Identificador único de la administración.
- medicamento_id: Referencia al medicamento administrado.
- residente_id: Referencia al residente al que se le administra el medicamento.
- trabajador_id: Referencia al trabajador que administra el medicamento.
- fecha_administracion: Fecha de administración del medicamento.
- hora_administracion: Hora de administración del medicamento.
- dosis_administrada: Dosis administrada del medicamento.
- observaciones: Observaciones sobre la administración del medicamento.



The diagram illustrates a relationship between two tables. A blue line with arrows points from the 'medicamento_id' column in the 'citasExternas' table to the 'id' column in the 'administracion_medicamentos' table, indicating a foreign key relationship.

| u783570590_GABI administracion_medicamentos | |
|---|--|
| • id : int(11) | |
| # medicamento_id : int(11) | |
| # residente_id : int(11) | |
| # trabajador_id : int(11) | |
| □ fecha_administracion : date | |
| □ hora_administracion : time | |
| □ dosis_administrada : varchar(50) | |
| □ observaciones : varchar(255) | |

Tabla citas_externas

Esta tabla almacena información sobre las citas médicas externas de los residentes. Sirve para que el perfil de enfermería vaya almacenando la información de las citas de los residentes con médicos de cabeceras u otras acciones ya sean dentro o fuera del centro pero que sean de formato particular para un residente.

- id: Identificador único de la cita.
- residente_id: Referencia al residente que tiene la cita.
- fecha_cita: Fecha de la cita.
- hora_cita: Hora de la cita.
- lugar_cita: Lugar donde se realizará la cita.
- motivo_cita: Motivo de la cita.
- detalles: Detalles adicionales sobre la cita.

The diagram illustrates a relationship between two tables. A blue line with arrows points from the 'residente_id' column in the 'controles' table to the 'id' column in the 'citasExternas' table, indicating a foreign key relationship.

| u783570590_GABI citas_externas | |
|--------------------------------|--|
| • id : int(11) | |
| # residente_id : int(11) | |
| □ fecha_cita : date | |
| □ hora_cita : time | |
| □ lugar_cita : varchar(100) | |
| □ motivo_cita : varchar(255) | |
| □ detalles : text | |

Tabla controles

Esta tabla registra los controles realizados a los residentes. La información recabada sería el seguimiento que hace el rol enfermero para tener la información de las inyecciones, goteros o curas que realiza.

- id: Identificador único del control.



- residente: Referencia al residente controlado.
- Fecha: Fecha del control.
- hora: Hora del control.
- trabajador: Referencia al trabajador que realiza el control.
- acción: Acción realizada durante el control.
- Notas: Notas adicionales sobre el control.

| u783570590_GABI controles | |
|---------------------------|----------------------|
| # | id : int(11) |
| # | residente : int(11) |
| # | fecha : date |
| # | hora : time |
| # | trabajador : int(11) |
| # | accion : varchar(50) |
| # | notas : varchar(255) |

Tabla documentos

Esta tabla almacena documentos relacionados con los residentes. Para cada residente se requiere almacenar fotocopia de empadronamiento, foto de la tarjeta de la seguridad social y otros documentos de relevancia como el dni. Este almacenamiento se identificará por el DNI del residente y permitirá la subida y descarga de distintos formatos. El encargado de mantener la información y modificarla sería el administrador y el enfermero,

- id: Identificador único del documento.
- dni: DNI del residente al que pertenece el documento.
- título: Título del documento.
- descripcion: Descripción del documento.
- nombre_archivo: Nombre del archivo del documento.
- tipo_archivo: Tipo de archivo del documento.
- contenido: Contenido del documento en formato binario.
- fecha_subida: Fecha y hora en que se subió el documento.



| u783570590_GABI documentos | |
|----------------------------|----------------|
| id | : int(11) |
| dni | : varchar(255) |
| titulo | : varchar(255) |
| descripcion | : text |
| nombre_archivo | : varchar(255) |
| tipo_archivo | : varchar(50) |
| contenido | : longblob |
| fecha_subida | : datetime |

Tabla habitaciones

Esta tabla almacena información sobre las habitaciones de la residencia. Permite que el Administrador asigne habitaciones y poder administrar cambios manteniendo la logística.

- id: Identificador único de la habitación.
- número: Número de la habitación.
- piso: Piso donde se encuentra la habitación.
- observaciones: Observaciones adicionales sobre la habitación.
- ocupada: Indica si la habitación está ocupada.

| u783570590_GABI habitaciones | |
|------------------------------|---------------|
| id | : int(11) |
| numero | : int(10) |
| piso | : varchar(10) |
| observaciones | : text |
| ocupada | : tinyint(1) |

Tabla medicamentos

Esta tabla almacena información sobre los medicamentos asignados a los residentes. Esta tabla es para el rol de enfermero que permitirá ingresar las medicinas o medicamentos de los cuales dispone y después poder asignarlos a un paciente, desde pastillas hasta inyecciones por ejemplo.

- id: Identificador único del medicamento.
- dni_residente: Referencia al residente que utiliza el medicamento.
- nombre_medicamento: Nombre del medicamento.
- dosis: Dosis del medicamento.
- fecha_inicio: Fecha de inicio del uso del medicamento.
- fecha_fin: Fecha de finalización del uso del medicamento.
- observaciones: Observaciones sobre el medicamento.



- documentos: Documentos asociados al medicamento

| v u783570590_GABI medicamentos | |
|------------------------------------|--|
| # id : int(11) | |
| # dni_residente : int(11) | |
| ③ nombre_medicamento : varchar(50) | |
| ③ dosis : varchar(50) | |
| ④ fecha_inicio : date | |
| ④ fecha_fin : date | |
| ③ observaciones : varchar(255) | |
| ◆ documentos : blob | |

Tabla pagos

Esta tabla registra los pagos de los residentes a alto nivel. Esta tabla servirá para mostrar información de forma rápida del estado de los pagos de los residentes y que dará información para tomar acciones, no está pensada para tener toda la información sobre la contabilidad con los residentes pero en el futuro se podría implementar un módulo de contabilidad y sería desde allí donde se tomaron estos datos.

- id: Identificador único del pago.
- residente_id: Referencia al residente que realiza el pago.
- fecha_entrada: Fecha de entrada del residente.
- fecha_fin: Fecha de salida del residente (si aplica).
- fecha_corresponde_pago: Fecha correspondiente al pago.
- estado_impago: Indica si el pago está pendiente.

| v u783570590_GABI pagos | |
|---------------------------------|--|
| # id : int(11) | |
| # residente_id : int(11) | |
| ④ fecha_entrada : date | |
| ④ fecha_fin : date | |
| ④ fecha_corresponde_pago : date | |
| # estado_impago : tinyint(1) | |

Tabla registro_entradas_salidas

Esta tabla registra las entradas y salidas de los trabajadores. Esta tabla debería permitir en el futuro obtener informes sobre el desempeño o días de vacaciones que pertenecen a una persona trabajador.

- id: Identificador único del registro.
- trabajador_id: Referencia al trabajador.
- fecha: Fecha del registro.
- hora_entrada: Hora de entrada.



- hora_salida: Hora de salida.
- notas: Notas adicionales sobre el registro.



Tabla residentes

Esta tabla almacena información sobre los residentes. Es el nucleo de muchas tablas relacionales y ademas la que contiene más informacion relevante y sensible para el manejo de la logistica y las tareas que rodean a un residente.

- id: Identificador único del residente.
- dni: DNI del residente.
- nombre: Nombre del residente.
- telefono: Teléfono del residente.
- email: Email del residente.
- apellidos: Apellidos del residente.
- fecha_nacimiento: Fecha de nacimiento del residente.
- ar: AR del residente.
- nss: NSS del residente.
- numero_cuenta_bancaria: Número de cuenta bancaria del residente.
- observaciones: Observaciones adicionales sobre el residente.
- medicamentos: Referencia a los medicamentos del residente.
- fecha_ingreso: Fecha de ingreso del residente.
- activo: Indica si el residente está activo.
- empadronamiento: Empadronamiento del residente.
- edad: Edad del residente.
- mes_cumple: Mes de cumpleaños del residente.
- foto: Foto del residente en formato binario.
- habitacion_id: Referencia a la habitación del residente.
- estado: Estado del residente.
- tlfn_familiar_1: Teléfono del familiar 1.
- tlfn_familiar_2: Teléfono del familiar 2.



| u783570590_GABI_residentes | |
|----------------------------|---------------------------------------|
| • | id : int(11) |
| • | dni : varchar(255) |
| • | nombre : varchar(30) |
| • | telefono : varchar(15) |
| • | email : varchar(255) |
| • | apellidos : varchar(50) |
| • | fecha_nacimiento : date |
| • | ar : varchar(255) |
| • | nss : varchar(255) |
| • | numero_cuenta_bancaria : varchar(255) |
| • | observaciones : varchar(255) |
| # | medicamentos : int(11) |
| • | fecha_ingreso : date |
| • | activo : varchar(2) |
| • | empadronamiento : varchar(255) |
| # | edad : int(11) |
| # | mes_cumple : int(11) |
| ◆ | foto : longblob |
| # | habitacion_id : int(11) |
| # | estado : tinyint(1) |
| • | tlfn_familiar_1 : varchar(15) |
| • | tlfn_familiar_2 : varchar(15) |

Tabla stock_medicamentos

Esta tabla almacena el stock de medicamentos. Para el rol de enfermería para que pueda saber en qué momento debe solicitar más medicamentos según los datos que tenga sobre los controles que se le deban hacer a los pacientes

- id: Identificador único del stock.
- medicamento_id: Referencia al medicamento.
- cantidad: Cantidad de medicamento en stock.
- unidad: Unidad de medida del medicamento.
- fecha_actualizacion: Fecha de actualización del stock.
- observaciones: Observaciones adicionales sobre el stock.



| u783570590_GABI stock_medicamentos | |
|------------------------------------|--------------------------------|
| • | id : int(11) |
| • | # medicamento_id : int(11) |
| • | # cantidad : int(11) |
| • | □ unidad : varchar(50) |
| • | □ fecha_actualizacion : date |
| • | □ observaciones : varchar(255) |

Tabla tareas

Esta tabla almacena las tareas asignadas a los trabajadores. Esta tabla almacenara tareas generalistas no relacionadas con residentes si no más bien con las diferentes tareas que realiza un auxiliar de geritria, asegurando la realizacion de dichas tareas y teniendo en cuenta la informacion para en el futuro poder ser interpretada y optimizar la asignacion de tareas.

- id: Identificador único de la tarea.
- titulo: Título de la tarea.
- notas: Notas adicionales sobre la tarea.
- estado: Estado de la tarea.
- trabajador_id: Referencia al trabajador asignado.
- fecha_tarea_asignada: Fecha de asignación de la tarea.
- hora_tarea_asignada: Hora de asignación de la tarea.
- fecha_tarea_finalizada: Fecha de finalización de la tarea.
- hora_tarea_finalizada: Hora de finalización de la tarea.

| u783570590_GABI tareas | |
|------------------------|------------------------------------|
| • | id : int(11) |
| • | □ titulo : varchar(255) |
| • | □ notas : text |
| • | # estado : tinyint(1) |
| • | # trabajador_id : int(11) |
| • | □ fecha_tarea_asignada : date |
| • | □ hora_tarea_asignada : datetime |
| • | □ fecha_tarea_finalizada : date |
| • | □ hora_tarea_finalizada : datetime |

Tabla trabajadores

Esta tabla almacena información sobre los trabajadores de la residencia. Contendrá los datos relevantes para el sistema entorno a la administración con lo residentes pero no tendrá relación con contabilidad por ejemplo para las nominas, se podría adaptar en un futuro.

- id: Identificador único del trabajador.



- dni: DNI del trabajador.
- nombre: Nombre del trabajador.
- apellido_1: Primer apellido del trabajador.
- apellido_2: Segundo apellido del trabajador.
- puesto: Puesto del trabajador (enfermero, administrador, auxiliar).
- telefono: Teléfono del trabajador.
- email: Email del trabajador.
- contrasena: Contraseña del trabajador.
- trabajador_activo: Indica si el trabajador está activo.

The screenshot shows the MySQL Workbench interface with a table named 'trabajadores'. The table has the following columns:

| | id : int(11) | dni : varchar(255) | nombre : varchar(50) | apellido_1 : varchar(50) | apellido_2 : varchar(50) | puesto : enum('enfermero','administrador','auxiliar') | telefono : varchar(20) | email : varchar(100) | contrasena : varchar(255) | # trabajador_activo : tinyint(1) |
|--|--------------|--------------------|----------------------|--------------------------|--------------------------|---|------------------------|----------------------|---------------------------|----------------------------------|
|--|--------------|--------------------|----------------------|--------------------------|--------------------------|---|------------------------|----------------------|---------------------------|----------------------------------|

Tabla turnos

Esta tabla almacena los turnos de trabajo asignados a los trabajadores. Pensada para el rol de administrador para que pueda asignar los turnos a los trabajadores y en un futuro poder asignar los turnos en base a la estadística si por ejemplo como en muchas residencias los empleados rotan.

- id: Identificador único del turno.
- trabajador_id: Referencia al trabajador asignado.
- fecha_inicio: Fecha y hora de inicio del turno.
- fecha_fin: Fecha y hora de fin del turno.
- tipo: Tipo de turno (diurno, nocturno, vespertino).



| u783570590_GABI turnos | |
|------------------------|---|
| # | id : int(11) |
| # | trabajador_id : int(11) |
| # | fecha_inicio : datetime |
| # | fecha_fin : datetime |
| # | tipo : enum('diurno','nocturno','vespertino') |

Relaciones entre las Tablas

administracion_medicamentos:

1. Relación con medicamentos mediante medicamento_id.
2. Relación con residentes mediante residente_id.
3. Relación con trabajadores mediante trabajador_id.

citas_externas:

4. Relación con residentes mediante residente_id.

controles:

5. Relación con residentes mediante residente.
6. Relación con trabajadores mediante trabajador.

medicamentos:

7. Relación con residentes mediante dni_residente.

pagos:

8. Relación con residentes mediante residente_id.

registro_entradas_salidas:

9. Relación con trabajadores mediante trabajador_id.

residentes:

10. Relación con habitaciones mediante habitacion_id.

stock_medicamentos:

11. Relación con medicamentos mediante medicamento_id.

tareas:



12. Relación con trabajadores mediante trabajador_id.

turnos:

13. Relación con trabajadores mediante trabajador_id.

3.4.2 Documentación de Software del Modelado de Datos para Firebase

Firebase Realtime Database es una base de datos NoSQL alojada en la nube que permite almacenar y sincronizar datos entre usuarios en tiempo real. Este documento describe la estructura de la base de datos Firebase utilizada para el chat grupal entre los distintos trabajadores, independientemente de su rol, de la aplicación de gestión de residencias.

Estructura de la Base de Datos Firebase

La base de datos Firebase está estructurada para almacenar mensajes de chat entre los empleados. Cada mensaje contiene información relevante como el nombre del remitente, el contenido del mensaje, la hora de envío y el tipo de mensaje. Se empezó a desarrollar el envío de imágenes y también tener una foto de perfil, también se empezó a probar a crear canales individuales para cada chat pero de momento se ha dejado para el uso de un chat global y que se almacenen solamente los mensajes, siendo todos los usuarios remitentes

Nodos y Subnodos

La estructura de la base de datos se organiza en nodos y subnodos como se muestra en las imágenes proporcionadas. Aquí se presenta una descripción detallada de los nodos principales:

Nodo Principal: chat

- Este nodo contiene todos los mensajes de chat.
- Cada mensaje se identifica por un ID único generado por Firebase.

Subnodos dentro de chat

Cada subnodo representa un mensaje individual con la siguiente estructura:

- **hora:** Timestamp del mensaje en milisegundos.
- **mensaje:** Contenido del mensaje.
- **nombre:** Nombre del remitente.



- **typeMensaje:** Tipo de mensaje (por ejemplo, texto, imagen).

```
https://gabi-b4507-default-rtdb.firebaseio.app/  
  ↓  — chat  
    ↓  — -NyzB-1m3fkvZ0o_ysGu  
      — hora: 1716899545309  
      — mensaje: "DASD"  
      — nombre: "G"  
      — typeMensaje: "text"
```

3.5. Análisis y diseño de la interfaz de usuario

La creación de User Personas es clave en el proceso de diseño y mejora de la experiencia de las personas usuarias. Este método proporciona un marco de referencia para entender a los usuarios finales, y guía el desarrollo de productos y servicios de manera más empática y centrada en las personas.

La base de una User Persona efectiva es una investigación meticulosa y la recolección de datos reales. Antes de dibujar cualquier conclusión o esbozar perfiles, es esencial sumergirse en el mundo de nuestros usuarios, entendiendo sus necesidades, frustraciones y deseos a través de una lente objetiva y basada en evidencia.

En base a la investigación hecha en varias residencias de ancianos y hablando de las necesidades administrativas con sus directores y teniendo en cuenta lo poco eficaz que es el sistema actual en la mayoría de residencias que no formen parte de una gran franquicia el tema de archivar la información y obtener información de manera rápida y sencilla he tomado como user persona a un administrador, en base a la información que me han dado sobre cómo gestionan el tiempo de los empleados y la organización. No obstante cabe destacar que para mi diseño tambien he tenido en cuenta a los otros trabajadores como auxiliares y enfermeros, sus necesidades y el cómo poder facilitarles el uso de la aplicación para en general tener una mejor



3.5.1. User Persona

User Persona: Marta López

Información Demográfica

Nombre: Marta López

Edad: 34 años

Ocupación: Administradora de Residencia

Ubicación: Madrid, España

Educación: Licenciatura en Administración y Dirección de Empresas

Estado Civil: Soltera

Perfil Personal

Marta es una mujer joven y profesional que trabaja como administradora en una residencia de ancianos. Es muy organizada y meticulosa en su trabajo, con una fuerte inclinación por la eficiencia y la tecnología. Marta pasa gran parte de su tiempo gestionando turnos de trabajadores, programando citas para los residentes y asegurándose de que todo en la residencia funcione sin problemas.

Objetivos

Optimizar la gestión del personal: Quiere asegurarse de que todos los trabajadores estén bien coordinados y que los turnos se asignen de manera justa y eficiente.

Asegurar la satisfacción de los residentes: Desea que todos los residentes reciban la atención adecuada y a tiempo, sin retrasos ni olvidos.

Mejorar la comunicación interna: Quiere una plataforma de comunicación rápida y efectiva para mantenerse en contacto con el personal y resolver problemas rápidamente.

Pain Points

- Gestión complicada de turnos: Le resulta difícil coordinar los turnos de manera eficiente, especialmente cuando hay cambios de última hora.
- Falta de visibilidad de tareas: Necesita una visión clara y actualizada de las tareas asignadas a cada trabajador y el estado de las mismas.
- Problemas de comunicación: A veces, la comunicación entre el personal no es tan fluida como debería ser, lo que puede llevar a malentendidos y errores.

Necesidades

- Herramientas de gestión de turnos fáciles de usar: Una interfaz que le permita asignar y ajustar turnos rápidamente.
- Visibilidad clara de las citas y eventos: Un calendario que muestre todas las citas y eventos del día.
- Comunicación eficiente: Un sistema de mensajería interno que permita una comunicación rápida y efectiva con todo el personal.

Menú de Navegación



El menú de navegación de la aplicación es claro y funcional, con iconos intuitivos y etiquetas que facilitan el acceso rápido a las secciones principales. El botón central para el chat está estratégicamente colocado para un acceso rápido y directo a la comunicación, lo que mejora la eficiencia en la gestión de la residencia.

Pantalla de Inicio de Sesión

La pantalla de inicio de sesión presenta un diseño moderno y limpio, con campos claros para el usuario y la contraseña, y un botón prominente de "Entrar". Esta interfaz asegura una experiencia de usuario sencilla y eficiente desde el primer punto de contacto con la aplicación.

Pantalla de Gestión de Turnos

La pantalla para asignar turnos a los auxiliares ofrece un calendario intuitivo y botones claros para buscar trabajadores y asignar nuevos turnos. La lista de turnos del día actual se muestra claramente, permitiendo una rápida visualización de los horarios y roles asignados. Esta funcionalidad es crucial para Marta, ya que le permite gestionar los turnos de manera eficiente y asegurarse de que todos los trabajadores tengan descansos adecuados entre sus turnos.

Pantalla de Gestión de Residentes

La pantalla de gestión de residentes es intuitiva y bien organizada, con opciones para agregar, listar, modificar y asignar habitaciones a los residentes. Los botones son grandes y claros, lo que facilita la usabilidad y permite a Marta realizar estas tareas de manera rápida y eficiente.

Pantalla de Buscador de Archivos

El buscador de archivos permite a Marta buscar rápidamente todos los documentos relacionados con un DNI específico. La lista de resultados muestra el nombre del archivo, una breve descripción, la fecha y hora de subida, y botones para eliminar o descargar el archivo. Este diseño eficiente facilita el acceso y gestión de documentos importantes de manera rápida y organizada.

Colores:

La relación de contraste es crucial para asegurar la legibilidad del contenido en cualquier aplicación, incluyendo nuestra aplicación de gestión de residencia. La relación de contraste mide la diferencia entre la máxima y mínima luminosidad de una pantalla, lo que directamente afecta la facilidad con la que los usuarios pueden leer el contenido.

Un número más alto en la relación de contraste indica una mejor legibilidad. Los números más bajos, y los marcados en rojo en nuestra herramienta, indican una legibilidad pobre. Según lo define el Consorcio Mundial de la Web (W3C), la



luminancia relativa es la luminosidad relativa de cualquier punto en un espacio de color, normalizada de 0 para el negro más oscuro a 1 para el blanco más claro.

Para asegurar una buena legibilidad, especialmente cuando hay texto pequeño, una buena relación de contraste es cualquier número por encima de 4.5 o 4.5:1. Según el W3C, el texto más grande puede tener una relación de contraste de hasta 3:1, pero idealmente debería ser más alta.

En la interfaz de nuestra aplicación, utilizamos herramientas para ajustar los colores de fondo y de texto, creando una experiencia más accesible. La relación de contraste óptima es fundamental para que todos los usuarios, independientemente de sus condiciones visuales, puedan utilizar la aplicación de manera eficiente y sin esfuerzo. Al asegurar que nuestros colores cumplen con estos estándares, garantizamos que nuestra aplicación no solo sea funcional sino también accesible para todos. Esto no solo mejora la usabilidad, sino que también promueve una inclusión verdadera, permitiendo que más personas se beneficien de nuestra tecnología.

The screenshot shows the Contrast Ratio tool interface. It displays two color combinations side-by-side:

- Left Combination:** Background: #10425E (Relative Luminance: 0.0481) and Text color: white (#10425E). The contrast ratio is 10.69, highlighted with a green circle. A "Swap colors" button is present.
- Right Combination:** Background: white (#10425E) and Text color: #10425E (Relative Luminance: 0.0481). The contrast ratio is 10.69, highlighted with a green circle. A "Swap colors" button is present.

How to use: As you type, the contrast ratio indicated will update. Hover over the circle to get more detailed information. When semi-transparent colors are involved as backgrounds, the contrast ratio will have an error margin, to account for the different colors they may be over.

This sample text attempts to visually demonstrate how readable this color combination is, for normal, *italic*, **bold**, or **bold italic** text of various sizes and font styles.

It's critical for content readability and web performance that your website's ratio be in the green zone.

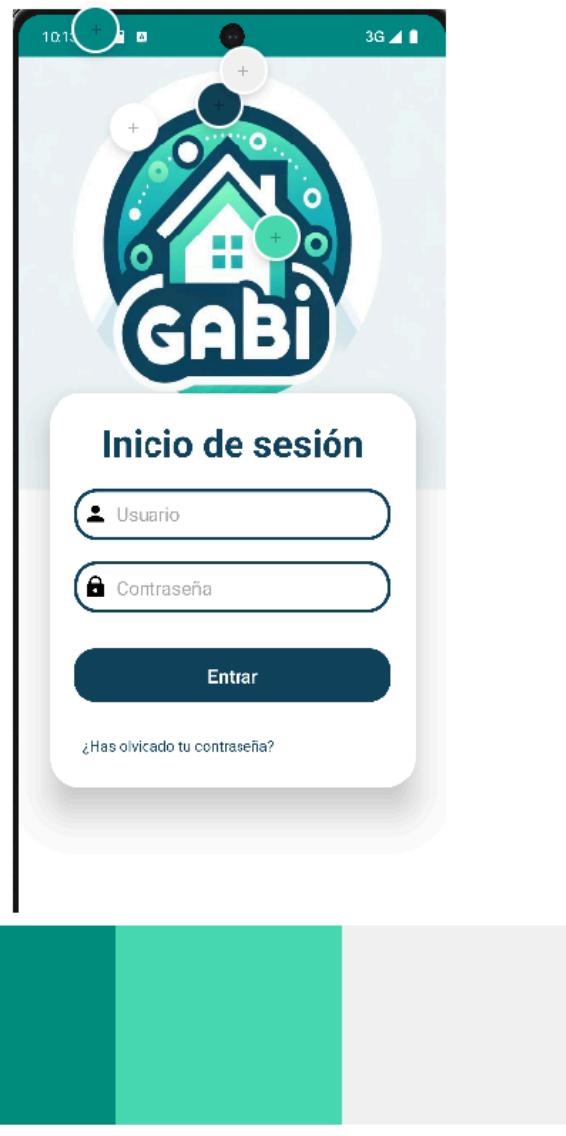
Contrast Ratio was created by Lea Verou.

Hint: Press the up and down keyboard arrows while over a number inside a functional color notation. Watch it increment/decrement. Try with the Shift or Alt key too!

WCAG 2.1 on contrast ratio

Activar Windows
Ve a Configuración para activar Windows.

<https://www.siegemedia.com/contrast-ratio#%2310425E-on-white>



#114559

#038C7F

#48D9B0

#F2F2F2

#FFFFFF

Colores claros

Ideales para fondos y sustratos

#FFFFFF

#EFFCF8

#DEF8F1

#CEF5EB

#BEF2E4

#ADEEDD

#9DEBD6

#8CE7CF

#7CE4C8

#6CE1C2



#5BDDDB

#4BDAB4

Colores oscuros

Se pueden usar para texto y encabezados

#000000

#071410

#0E2821

#143B31

#1B4F41

#226352

#297762

#308B73

#379F83

#3DB293

#44C6A4

#4BDAB4

3.5.2. Frontend

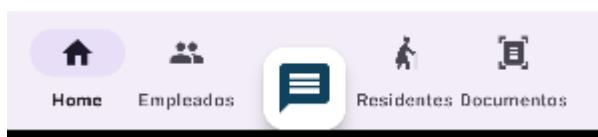
Login:

La pantalla de inicio de sesión de la app "GABI" presenta un diseño moderno y limpio. Incluye un logotipo atractivo en la parte superior, dos campos de entrada (usuario y contraseña) con íconos claros, un botón de "Entrar", y un enlace para recuperar la contraseña. La interfaz es intuitiva, asegurando una experiencia de usuario sencilla y eficiente.



menu:

Menú de navegación claro y funcional, diseñado para facilitar el acceso rápido a las secciones principales de la aplicación. Los iconos y etiquetas son intuitivos, permitiendo a los usuarios entender fácilmente su propósito. El botón central destacado, que se utiliza para el chat, está estratégicamente colocado para un acceso rápido y directo a la comunicación. La selección actual, "Home", está resaltada con un color de fondo suave, ayudando a los usuarios a identificar su ubicación dentro de la app.



chat:

Esta pantalla de chat ofrece una experiencia de usuario limpia y sencilla. El mensaje se muestra en un cuadro de diálogo con bordes redondeados, indicando el remitente, el contenido del mensaje y la hora de envío. La interfaz incluye un campo de entrada para escribir mensajes y un botón de "Enviar" accesible y destacado.



Este diseño minimalista y funcional asegura que los usuarios puedan comunicarse de manera rápida y eficiente, sin distracciones.



Home (administrador) :

Esta pantalla para administradores de la residencia ofrece una visión clara y organizada de la información relevante del día. En la parte superior, se muestra una lista de trabajadores activos con sus roles, facilitando la gestión del personal. A continuación, el calendario de eventos del día presenta las revisiones médicas programadas de forma cronológica, permitiendo un seguimiento fácil de las citas. Al final, la hoja de turnos de hoy detalla los turnos de cada trabajador, asegurando que todos estén al tanto de sus responsabilidades.



Trabajadores Activos

- H H H administrador
- OSCAR JIP OIS auxiliar

Calendario de Eventos del Día

- 09:00:00 Revisión médica DD Dd Dd
- 10:00:00 Revisión médica ZZ ZZ ZZ
- 11:00:00 Revisión médica RR RR RR
- 09:30:00 Revisión médica Juan Gomez
- 10:30:00 Revisión médica Luis MARTINES PERALTA

Hoja de Turnos de Hoy

| L L L | administrador |
|-----------------|---------------|
| Y Y Y | administrador |
| H H H | administrador |
| M M m | administrador |
| G G G | administrador |
| OSCAR JIP OIS | auxiliar |
| Mario ORTI ASSD | auxiliar |
| Alba sañé nel | auxiliar |

Home **Empleados** **Residentes Documentos**

Documentos:

Este buscador de archivos permite a los administradores buscar rápidamente todos los documentos relacionados con un DNI específico. Al ingresar el DNI en el campo de búsqueda y hacer clic en "Buscar", se muestra una lista de archivos asociados con ese DNI. Cada archivo en la lista incluye el nombre del archivo, una breve descripción, la fecha y hora de subida, y botones para eliminar o descargar el archivo. Este diseño es eficiente y funcional, facilitando el acceso y gestión de documentos importantes de manera rápida y organizada.

Buscar

| | |
|---|--|
| RRDASDA RRDASDASDASD 2024 05 26 12:23:45 | |
| RRDASDADASDA RRDASDASDASDASDASD 2024 05 26 12:51:51 | |
| RRDASDADASDA RRDASDASDASDASDASD 2024 05 26 14:35:49 | |
| DASDA ASDS 2024-05-26 14:46:31 | |

Subir Archivo

DNI: 12345678A

Título: Título del archivo

Descripción: Descripción del archivo

Seleccionar Archivo

Archivo no seleccionado

Subir Archivo

Home **Empleados** **Residentes Documentos**



Residentes (administrador):

Esta pantalla de gestión de residentes es intuitiva y bien organizada, diseñada para facilitar las tareas administrativas. Ofrece cuatro opciones principales: agregar un nuevo residente, listar todos los residentes, modificar la información de un residente existente y asignar habitaciones. Los botones están claramente etiquetados y acompañados de íconos representativos, lo que mejora la usabilidad y permite a los administradores realizar acciones específicas de manera rápida y eficiente.

The image displays four screenshots of the Residentes (administrator) application:

- Screenshot 1: Home Screen**
Shows a dark blue header with the title "Residentes". Below it is a grid of four buttons: "Agregar Residente" (Add Resident), "Listar Residentes" (List Residents), "Modificar Residente" (Modify Resident), and "Asignar Habitación" (Assign Room). At the bottom is a navigation bar with icons for Home, Empleados, Residetes (highlighted), and Documentos.
- Screenshot 2: Registro de Residente (Resident Registration)**
A form for registering a new resident. Fields include DNI (12345678A), Nombre (Juan), Primer Apellido (Pérez), Segundo Apellido (Gómez), Fecha de Nacimiento (01/01/1980), Teléfono (600123456), Email (juan.perez@example.com), and AR (ZV1514612156). A note at the bottom states: "This app can only access the photos you select".
- Screenshot 3: Asignar Habitación (Assign Room)**
A list of room assignments:
 - Número: 101, Piso: 1, Observaciones: Sin observaciones
 - Número: 102, Piso: 1, Observaciones: Sin observaciones
 - Número: 103, Piso: 1, Observaciones: Sin observaciones
 - Número: 104, Piso: 1, Observaciones: Sin observacionesA large blue button at the bottom right says "Confirmar Asignación" (Confirm Assignment).
- Screenshot 4: Resident Profile View**
Shows a detailed profile for a resident named Luis Martínez Péralta. The profile includes a photo, and fields for Name (Luis), Surname (MARTINES PERALTA), DNI (11545154A), Phone (juan.perez@example.com), Email (600123456), Birthdate (1980-01-01), AR (151215610305), NSS (50211651631651), Bank Account (ES7620770024003102575766), Observations (COSAS dasdasAPUNTASR), and Medicamentos (0). The "Residetes" tab is highlighted in the navigation bar.



Empleados (Administrador) :

menú central para la gestión de empleados. Ofrece opciones para crear, listar, actualizar y eliminar trabajadores, así como asignarles tareas y turnos. Los botones son grandes y claros, lo que mejora la usabilidad y permite a los administradores realizar estas tareas con facilidad.

Estas pantallas de la aplicación están diseñadas para facilitar la gestión de turnos y tareas de los trabajadores de la residencia.

Presenta un calendario intuitivo para seleccionar la fecha, y botones claros para buscar trabajadores y asignar nuevos turnos. La lista de turnos del día actual se muestra debajo, permitiendo una rápida visualización de los horarios y roles asignados.

Al seleccionar "Asignar Turno", los administradores pueden elegir un trabajador de la lista y asignarles un turno diurno, vespertino o nocturno. Esta interfaz es sencilla y directa, facilitando la asignación rápida y eficiente de turnos.

En conjunto, estas pantallas proporcionan una solución integral y fácil de usar para la gestión de personal, optimizando la asignación de turnos y tareas, y asegurando que todas las operaciones se realicen de manera eficiente y organizada.

The image displays three screenshots of the GABI application's employee management interface:

- Screenshot 1: Main Dashboard (Left)**
 - Header: "Crear Trabajador" and "Listar Trabajadores".
 - Buttons: "Actualizar Trabajador", "Eliminar Trabajador", "Asignar Tareas", and "Asignar Turnos".
 - Bottom navigation: "Home", "Empleados", "Residentes", "Documentos".
- Screenshot 2: Assign Shifts to Auxiliaries (Middle)**
 - Title: "Asignar Turnos a los Auxiliares".
 - Calendar: May 2024 (26 to 31).
 - Buttons: "Buscar Trabajadores" and "Asignar Nuevo Turno".
 - Shift assignments:
 - L L L (administrador diurno)
 - Y Y Y (administrador nocturno)
 - H H H (administrador vespertino)
 - Bottom navigation: "Home", "Empleados", "Residentes", "Documentos".
- Screenshot 3: Assign Shift Details (Right)**
 - Title: "Asignar Turno".
 - Text: "Elige un trabajador para asignarle un turno:".
 - Table of workers:

| Nombre | Apellido | Role |
|--------|-----------|---------------|
| L L L | | administrador |
| Y Y Y | | administrador |
| H H H | | administrador |
| M M M | | administrador |
| G G G | | administrador |
| OSCAR | JIP OIS | auxiliar |
| Mario | ORTI ASSD | auxiliar |
| Alba | safie pol | auxiliar |
 - Shift selection radio buttons:
 - Diurno
 - Vespertino
 - Nocturno
 - Bottom navigation: "Home", "Empleados", "Residentes", "Documentos".



Asignar Tareas

Y YY
Revisar Armarios

Notas

9:45 AM

11 12 1 2
10 9 3
8 7 4
6 5

Asignar Tarea

Tareas Asignadas

Home Empleados Residentes Documentos

Registro de Trabajador

DNI
12345678A

Nombre
Juan

Primer Apellido
Pérez

Segundo Apellido
Gómez

Puesto
administrador

Teléfono
600123456

Email
juan.perez@example.com

Contraseña

Buscar trabajador

L
L
L
administrador
5
L

Y
Y
Y
administrador
9
Y

H
H
H
administrador
23
H

M
M
m
administrador
23544564
M

G
G
G
administrador
23
G

Home Empleados Residentes Documentos

DNI

Buscar

Nombre

Primer Apellido

Segundo Apellido

administrador

Teléfono

Email

Contraseña

Eliminar

Cancelar



4. IMPLEMENTACIÓN E IMPLANTACIÓN DEL SISTEMA

4.1. Implementación

La aplicación desarrollada sigue una arquitectura cliente-servidor. La parte del cliente está construida en Android utilizando Java, mientras que el servidor utiliza PHP y MySQL para manejar la lógica de negocio y la persistencia de datos.

4.1.1. Configuración Inicial del Proyecto Android

4.1.1.1. Archivo build.gradle.kts

El archivo build.gradle.kts define la configuración del proyecto y las dependencias necesarias. Aquí se configuran los aspectos básicos del proyecto, como el ID de la aplicación, las versiones de SDK, y las dependencias externas necesarias para el desarrollo como dependencias para el uso de Volley para manejar las solicitudes de red, Glide para el manejo de imágenes, y Firebase para funcionalidades como autenticación y base de datos en tiempo real.



```
1  plugins { this: PluginDependenciesSpecScope
2      id("com.android.application")
3      id("com.google.gms.google-services") // Añade esta linea
4  }
5
6  android { this: BaseAppModuleExtension
7      namespace = "com.example.gabi"
8      compileSdk = 34
9
10     defaultConfig { this: ApplicationDefaultConfig
11         applicationId = "com.example.gabi"
12         minSdk = 26
13         targetSdk = 34
14         versionCode = 1
15         versionName = "1.0"
16
17         testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
18     }
19
20     buildTypes { this: NamedDomainObjectContainer<ApplicationBuildType>
21         release { this: ApplicationBuildType
22             isMinifyEnabled = false
23             proguardFiles(
24                 getDefaultProguardFile( name: "proguard-android-optimize.txt"),
25                 "proguard-rules.pro"
26             )
27         }
28     }
29     buildFeatures { this: ApplicationBuildFeatures
30         dataBinding = true
31         viewBinding = true
32     }
33
34     compileOptions { this: CompileOptions
35         sourceCompatibility = JavaVersion.VERSION_1_8
36         targetCompatibility = JavaVersion.VERSION_1_8
37     }
38 }
39
40 dependencies { this: DependencyHandlerScope
41     // Para conectar con las diferentes APIs PHP
42     implementation("com.android.volley:volley:1.2.1")
43
44     implementation ("com.github.bumptech.glide:glide:4.12.0")
45     annotationProcessor ("com.github.bumptech.glide:compiler:4.12.0")
46
47     // Implementaciones necesarias
48     implementation("androidx.core:core-ktx:1.6.0")
49     implementation("androidx.appcompat:appcompat:1.6.1")
50     implementation("com.google.android.material:material:1.11.0")
51     implementation("androidx.constraintlayout:constraintlayout:2.1.4")
52
53     // Dependencias para RecyclerView y CardView
54     implementation("androidx.recyclerview:recyclerview:1.2.1")
55     implementation("androidx.cardview:cardview:1.0.0")
```



4.1.1.2. Archivo AndroidManifest.xml

La aplicación Android requiere varios permisos para funcionar correctamente. Estos permisos se declaran en el archivo AndroidManifest.xml y permiten a la aplicación acceder a internet, leer y escribir en el almacenamiento externo, acceder al estado de la red y enviar SMS.

El archivo AndroidManifest.xml define los permisos necesarios y las actividades de la aplicación.

Este archivo declara los permisos para el uso de Internet, lectura y escritura en el almacenamiento externo, acceso al estado de la red, y el envío de SMS. También define las actividades principales de la aplicación.



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <!-- Permisos para usar internet -->
    <uses-feature
        android:name="android.hardware.telephony"
        android:required="false" />
    <uses-permission android:name="android.permission.INTERNET" />
    <!-- Permisos para usar archivos -->
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <!-- Permisos para usar estado de la red -->
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <!-- Permisos para envíos de SMS -->
    <uses-permission android:name="android.permission.SEND_SMS"/>

<application
    android:name=".MyApplication"
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@drawable/icon_gabi"
    android:label="GABI"
    android:roundIcon="@drawable/icon_gabi"
    android:supportsRtl="true"
    android:requestLegacyExternalStorage="true"
    android:theme="@style/Theme.GABI"
    tools:targetApi="31">

        <activity
            android:name=".AdministradorActivity"
            android:exported="false" /> <!-- Evita que pueda accederse si no es desde el login -->

        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".ChatActivity"></activity>
        <activity android:name=".auxiliares.AuxiliarActivity"></activity>
    </application>
</manifest>
```

4.1.2. Interfaz de Usuario (UI)

La interfaz de usuario de la aplicación se ha desarrollado utilizando archivos XML para definir los layouts y componentes visuales, así como clases en Java para manejar la lógica y la interacción con estos componentes. A continuación, se detallan algunos ejemplos y descripciones de cómo se implementan estos elementos.



4.1.2.1. Layouts Principales

Los archivos XML se utilizan para definir la estructura y el diseño de las actividades y fragmentos de la aplicación. Estos archivos se encuentran en el directorio res/layout.

- **activity_main.xml**: Define el layout principal de la aplicación.
- **fragment_home_auxiliar.xml**: Define el layout del fragmento de inicio para los auxiliares.
- **activity_administrador.xml**: Define el layout de la actividad principal para el administrador.

El siguiente es un ejemplo de un archivo de layout en XML para la actividad principal.



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
    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/turquesa"
    tools:context=".AdministradorActivity">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/frame_layout"/>

    <com.google.android.material.bottomappbar.BottomAppBar
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/bottomAppBar"
        android:layout_gravity="bottom"
        android:background="@color/white"
        app:fabCradleMargin="10dp"
        app:fabCradleRoundedCornerRadius="50dp">

        <com.google.android.material.bottomnavigation.BottomNavigationView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/bottomNavigationView"
            android:layout_marginEnd="20dp"
            app:labelVisibilityMode="labeled"
            android:background="@android:color/transparent"
            app:menu="@menu/bottom_menu"/>

    </com.google.android.material.bottomappbar.BottomAppBar>

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fabAbrirChat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:backgroundTint="@color/white"
        android:contentDescription="Enviar mensaje"
        android:src="@drawable/baseline_message_24"
        app:layout_anchor="@+id/frame_layout"
        app:layout_anchorGravity="bottom|center"
        app:maxImageSize="40dp"
        app:tint="@color/turquesa" />

</androidx.coordinatorlayout.widget.CoordinatorLayout>
```



El siguiente es un ejemplo de una actividad simple que utiliza el layout anterior.

```
1 package com.example.gabi;
2
3 > import ...
11
12 </> public class AdministradorActivity extends AppCompatActivity {
13
14     private ActivityAdministradorBinding binding;
15
16     @Override
17     protected void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         binding = ActivityAdministradorBinding.inflate(getLayoutInflater());
20         setContentView(binding.getRoot());
21
22         String nombreUsuario = getIntent().getStringExtra("nombre");
23
24         replaceFragment(new HomeAdministrador());
25         binding.bottomNavigationView.setBackground(null);
26
27         binding.bottomNavigationView.setOnItemSelectedListener(item -> {
28             int id = item.getItemId();
29             if (id == R.id.homeAdministrador) {
30                 replaceFragment(new HomeAdministrador());
31             } else if (id == R.id.empleadosAdministrador) {
32                 replaceFragment(new TrabajadoresAdministrador());
33             } else if (id == R.id.residentesAdministrador) {
34                 replaceFragment(new ResidentesAdministrador());
35             } else if (id == R.id.documentosAdministrador) {
36                 replaceFragment(new DocumentosAdministrador());
37             } else if (id == R.id.chatAdministrador) {
38                 Intent intent = new Intent(getApplicationContext(), ChatActivity.class);
39                 intent.putExtra("nombre", nombreUsuario); // Pasa el nombre del usuario al ChatActivity
40                 startActivity(intent);
41             }
42             return true;
43         });
44
45         FloatingActionButton fabAbrirChat = findViewById(R.id.fabAbrirChat);
46         fabAbrirChat.setOnClickListener(v -> {
47             Intent intent = new Intent(getApplicationContext(), ChatActivity.class);
48             intent.putExtra("nombre", nombreUsuario); // Pasa el nombre del usuario al ChatActivity
49             startActivity(intent);
50         });
51     }
52
53     private void replaceFragment(Fragment fragment) {
54         FragmentManager fragmentManager = getSupportFragmentManager();
55         FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
56         fragmentTransaction.replace(R.id.frame_layout, fragment);
57         fragmentTransaction.commit();
58     }
59 }
```

4.1.2.2.. Menús

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



```
<menu
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/homeAdministrador"
        android:title="@string/homeadministrador"
        android:icon="@drawable/baseline_home_24"/>
    <item
        android:id="@+id/empleadosAdministrador"
        android:title="Empleados"
        android:icon="@drawable/baseline_people_24"/>
    <item
        android:id="@+id/chatAdministrador"
        android:title="Chat"
        android:enabled="false"/>
    <item
        android:id="@+id/residentesAdministrador"
        android:title="Residentes"
        android:icon="@drawable/baseline_elderly_woman_24"/>
    <item
        android:id="@+id/documentosAdministrador"
        android:title="Documentos"
        android:icon="@drawable/baseline_document_scanner_24"/>
</menu>
```

Los menús de la aplicación se definen en archivos XML dentro del directorio res/menu. Estos archivos especifican los elementos del menú y sus íconos están en la carpeta Drawable como todos los demás.

4.1.2.3. Definición de Items

Además de los layouts principales, se han creado varios archivos XML para definir elementos individuales que se utilizarán dentro de listas y otros componentes reutilizables. Ejemplo de item_residente.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="8dp"
    android:background="@drawable/rounded_corners">

    <TextView
        android:id="@+id/textViewTitulo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Titulo"
```



```
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:layout_marginBottom="4dp"/>

    <TextView
        android:id="@+id/textViewNotas"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Notas"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:layout_marginBottom="4dp"/>

    <TextView
        android:id="@+id/textViewFechaHora"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Fecha y Hora"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:layout_marginBottom="4dp"/>
</LinearLayout>
```

4.1.2.4. Drawable Resources

Los recursos de imágenes y gráficos se almacenan en el directorio res/drawable. Estos recursos incluyen íconos y fondos utilizados en la aplicación.

Ejemplo de íconos en drawable:

- **baseline_home_24.xml**
- **baseline_calendar_month_24.xml**
- **icon_tareas.xml**

Estos íconos se utilizan para mejorar la interfaz visual y proporcionar una experiencia de usuario más intuitiva.

```
<vector android:height="24dp" android:tint="#000000"
        android:viewportHeight="24" android:viewportWidth="24"
        android:width="24dp" xmlns:android="http://schemas.android.com/apk/res/android">
    <path          android:fillColor="@android:color/white"
        android:pathData="M10,20v-6h4v6h5v-8h3L12,3 2,12h3v8z"/>
</vector>
```

4.1.3 Implementación de Activities y Fragments

Las Activities y Fragments se implementan en clases Java, donde se define la lógica para interactuar con los componentes de la interfaz de usuario.

El MainActivity es el punto de entrada de la aplicación, encargada de gestionar el inicio de sesión y almacenar el token de autenticación, el cual será utilizado por otras actividades y fragmentos.



Se gestiona el inicio de sesión y se guarda el token en SharedPreferences, que luego se utilizará en otras actividades y fragmentos para realizar solicitudes autenticadas. Además es donde se determina el rol que tiene el usuario y el punto de partida para pasar a las otras actividades que contendrán los fragmentos que corresponden a las acciones que pueden llevar a cabo

```
package com.example.gabi;
public class MainActivity extends AppCompatActivity {

    EditText usuario;
    EditText contrasena;
    Button botonEntrar;
    TextView recuperarContrasena;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        usuario = findViewById(R.id.usuario);
        contrasena = findViewById(R.id.contrasena);
        botonEntrar = findViewById(R.id.botonEntrar);
        recuperarContrasena = findViewById(R.id.recuperarContrasena);

        botonEntrar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                login(usuario.getText().toString(),
                    contrasena.getText().toString());
            }
        });

        recuperarContrasena.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mostrarDialogoRecuperarContrasena();
            }
        });
    }

    private void login(final String username, final String password) {
        String url = "https://residencialontananza.com/api/login.php"; // URL del script PHP

        StringRequest stringRequest = new StringRequest(Request.Method.POST, url,
            response -> {
                try {
                    if (response.startsWith("(")) {
                        JSONObject jsonResponse = new JSONObject(response);
                        if (jsonResponse.getString("status").equals("success"))
{
                            String token = jsonResponse.getString("token");
                            String nombre = jsonResponse.getString("nombre");
                            int trabajadorId = jsonResponse.getInt("user_id");
// Obtener el ID del trabajador
                            String role = jsonResponse.getString("role");
                        }
                    }
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
        );
    }
}
```



```
// Guarda el token, el nombre y el ID del trabajador
en SharedPreferences
SharedPreferences sharedpreferences =
getSharedPreferences("MyAppPrefs", MODE_PRIVATE);
SharedPreferences.Editor editor =
sharedpreferences.edit();
editor.putString("token", token);
editor.putString("nombre", nombre);
editor.putInt("trabajador_id", trabajadorId);
editor.apply();

// Navegar a la actividad adecuada
Intent intent;
if (role.equals("administrador")) {
    intent = new Intent(MainActivity.this,
AdministradorActivity.class);
} else if (role.equals("auxiliar")) {
    intent = new Intent(MainActivity.this,
AuxiliarActivity.class);
} else {
    Toast.makeText(MainActivity.this, "Rol no
reconocido: " + role, Toast.LENGTH_SHORT).show();
    return;
}

intent.putExtra("nombre", nombre); // Pasa el nombre
del usuario
startActivity(intent);
finish();
} else {
    Toast.makeText(MainActivity.this,
jsonResponse.getString("message"), Toast.LENGTH_SHORT).show();
}
} else {
    Toast.makeText(MainActivity.this, "Respuesta del
servidor no válida: " + response, Toast.LENGTH_SHORT).show();
}
} catch (JSONException e) {
    Toast.makeText(MainActivity.this, "Error en la respuesta del
servidor: " + e.getMessage(), Toast.LENGTH_SHORT).show();
}
},
error -> {
    Toast.makeText(MainActivity.this, "Error de red: " +
error.getMessage(), Toast.LENGTH_SHORT).show();
})
{
@Override
protected Map<String, String> getParams() {
    Map<String, String> params = new HashMap<>();
    params.put("username", username);
    params.put("password", password);
    return params;
}
};

RequestQueue queue = Volley.newRequestQueue(this);
queue.add(stringRequest);
}
```



```
private void mostrarDialogoRecuperarContrasena() {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    LayoutInflater inflater = getLayoutInflater();
    View dialogView = inflater.inflate(R.layout.dialog_recuperar_contrasena,
null);
    builder.setView(dialogView);

    EditText dniEditText = dialogView.findViewById(R.id.dniEditText);
    Button btnEnviar = dialogView.findViewById(R.id.btnEnviar);
    Button btnCancellar = dialogView.findViewById(R.id.btnCancellar);

    AlertDialog dialog = builder.create();

    btnEnviar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String dni = dniEditText.getText().toString().trim();
            if (!dni.isEmpty()) {
                enviarSolicitudRecuperacion(dni, dialog);
            } else {
                Toast.makeText(MainActivity.this, "Por favor, ingrese su DNI",
Toast.LENGTH_SHORT).show();
            }
        }
    });

    btnCancellar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            dialog.dismiss();
        }
    });
}

dialog.show();
}

private void enviarSolicitudRecuperacion(String dni, AlertDialog dialog) {
    String url =
"https://residencialontananza.com/api/restaurarPasswordViaMail.php";

    StringRequest stringRequest = new StringRequest(Request.Method.POST, url,
        response -> {
            try {
                JSONObject jsonResponse = new JSONObject(response);
                Toast.makeText(MainActivity.this,
jsonResponse.getString("message"), Toast.LENGTH_SHORT).show();
                dialog.dismiss();
            } catch (JSONException e) {
                Toast.makeText(MainActivity.this, "Error en la respuesta del
servidor: " + e.getMessage(), Toast.LENGTH_SHORT).show();
            }
        },
        error -> {
            Toast.makeText(MainActivity.this, "Error de red: " +
error.getMessage(), Toast.LENGTH_SHORT).show();
        })
    );
}

@Override
protected Map<String, String> getParams() {
    Map<String, String> params = new HashMap<>();
    params.put("dni", dni);
    return params;
}
```



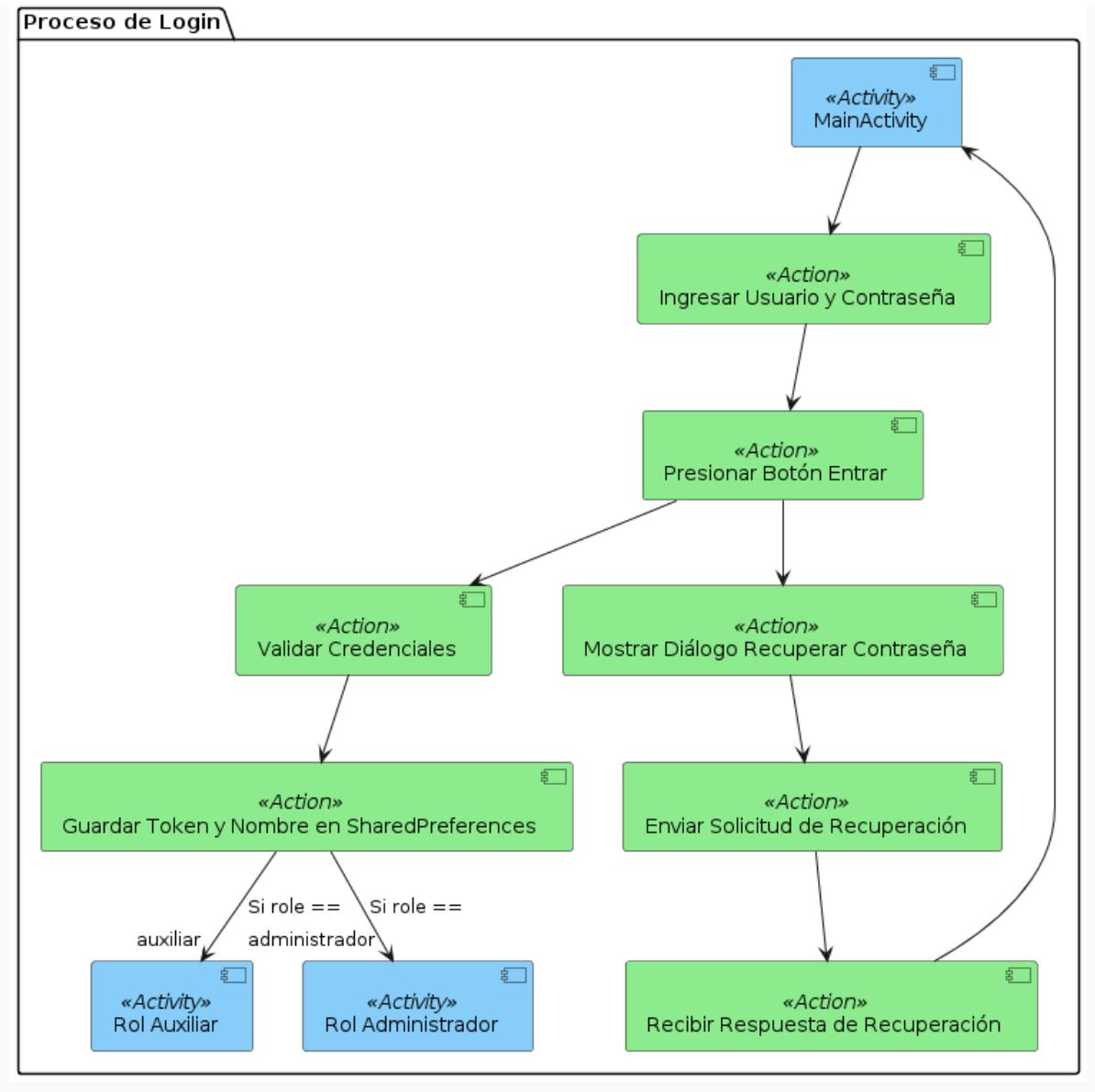
```
        params.put("dni", dni);
        return params;
    }

    RequestQueue queue = Volley.newRequestQueue(this);
    queue.add(stringRequest);
}

}
```

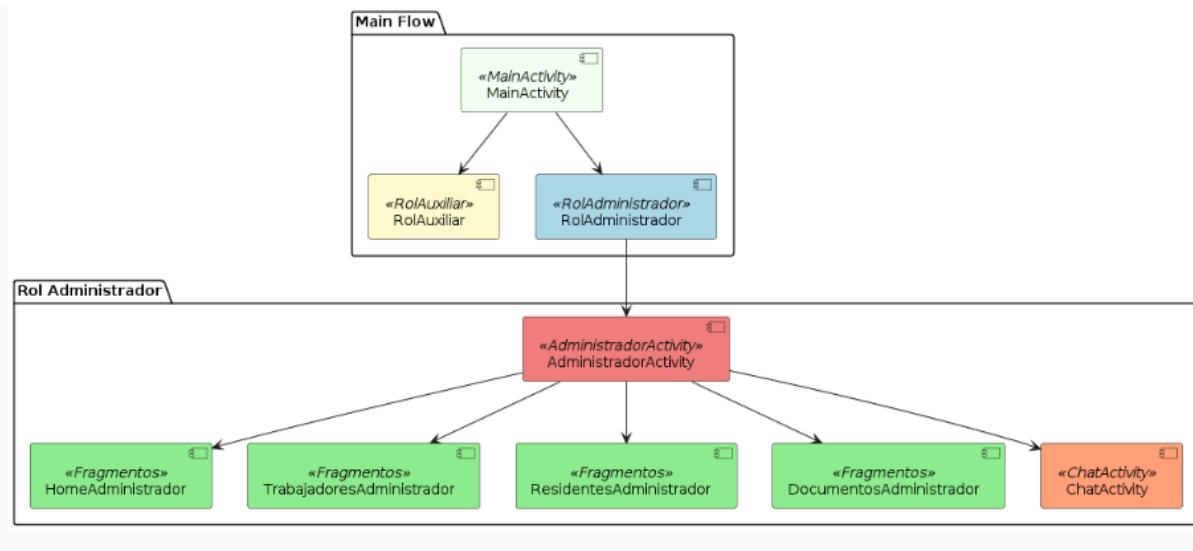
4.1.3.1 Diagramas y descripción de clases

Inicio de sesión





4.1.3.1.1 Flujo de Rol Administrador

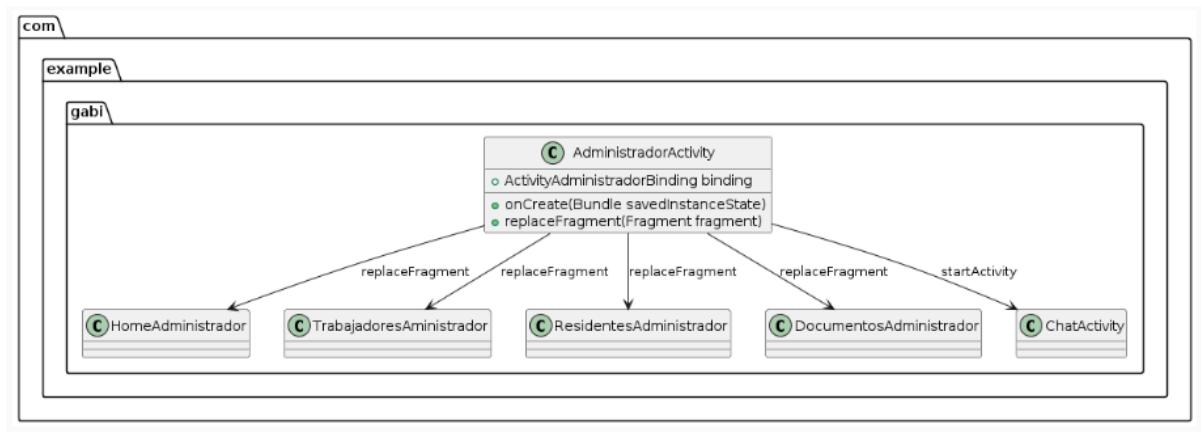


Esta sección representa el flujo principal de la aplicación.

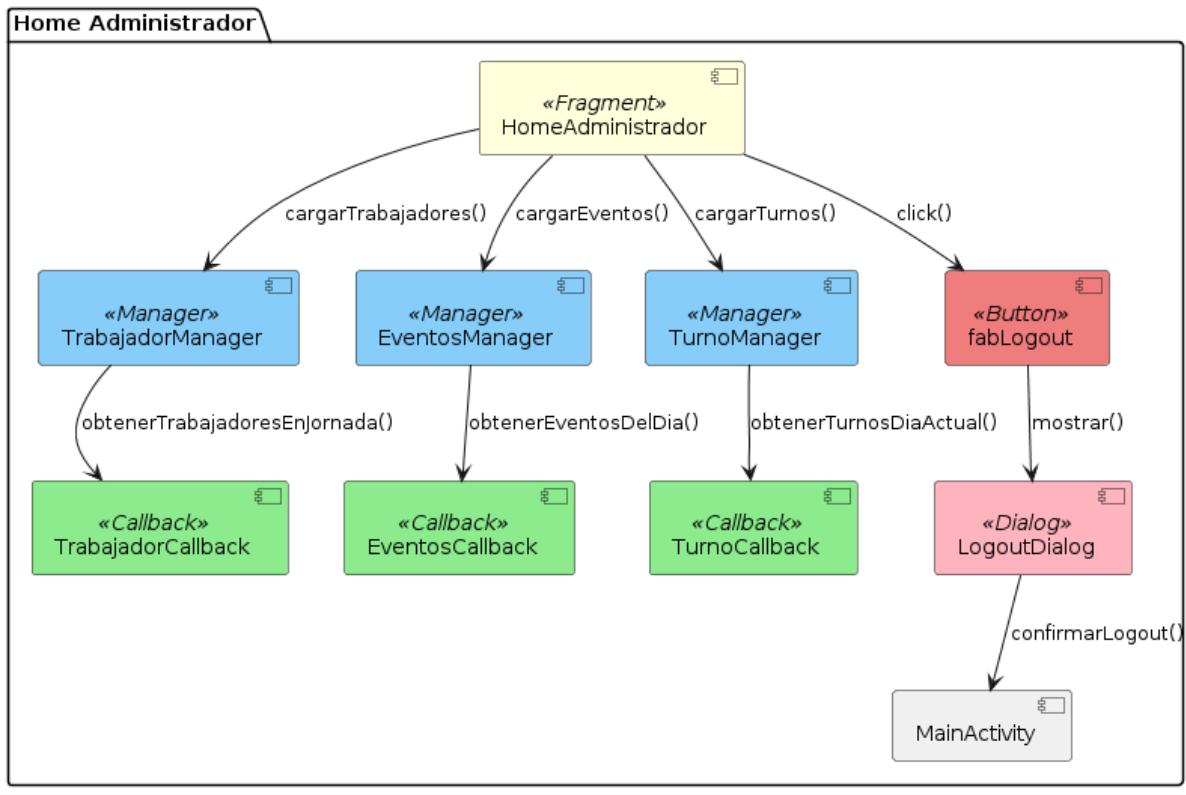
- **MainActivity:** Es el punto de entrada de la aplicación. Desde aquí, se puede navegar a dos roles diferentes:
 - **RolAuxiliar**
 - **RolAdministrador**

Esta sección muestra las diferentes opciones y fragmentos disponibles cuando un usuario con rol de administrador inicia sesión.

- **AdministradorActivity:** Desde aquí, se puede navegar a diferentes fragmentos dentro del rol de administrador:
 - **HomeAdministrador:** Fragmento de inicio para el administrador.
 - **TrabajadoresAdministrador:** Fragmento para gestionar a los trabajadores.
 - **ResidentesAdministrador:** Fragmento para gestionar los residentes.
 - **DocumentosAdministrador:** Fragmento para gestionar los documentos.
 - **ChatActivity:** Actividad de chat para la comunicación dentro de la aplicación.



HomeAdministrador Fragment



El diagrama representa el funcionamiento del fragmento `HomeAdministrador` dentro de la aplicación Android. Este fragmento es la vista principal para el administrador y contiene varias funcionalidades claves para la gestión de trabajadores, eventos y turnos.

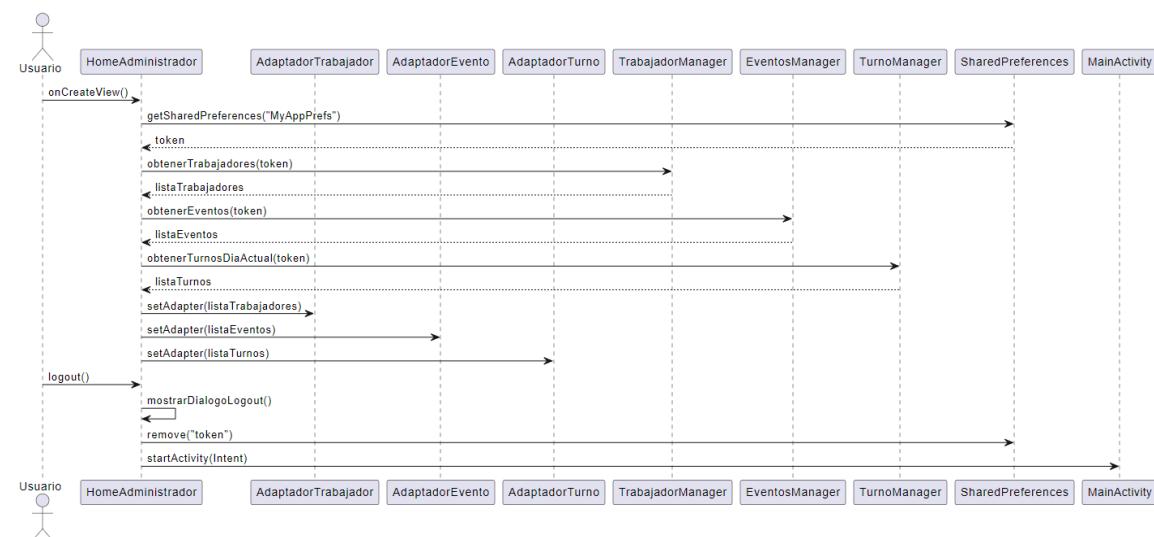
Dentro del fragmento `HomeAdministrador`, se inicializan varios componentes de la interfaz de usuario, incluyendo Recyclers para mostrar listas de trabajadores, eventos y turnos. También se incluye un FloatingActionButton llamado `fabLogout`, que permite al usuario cerrar sesión.

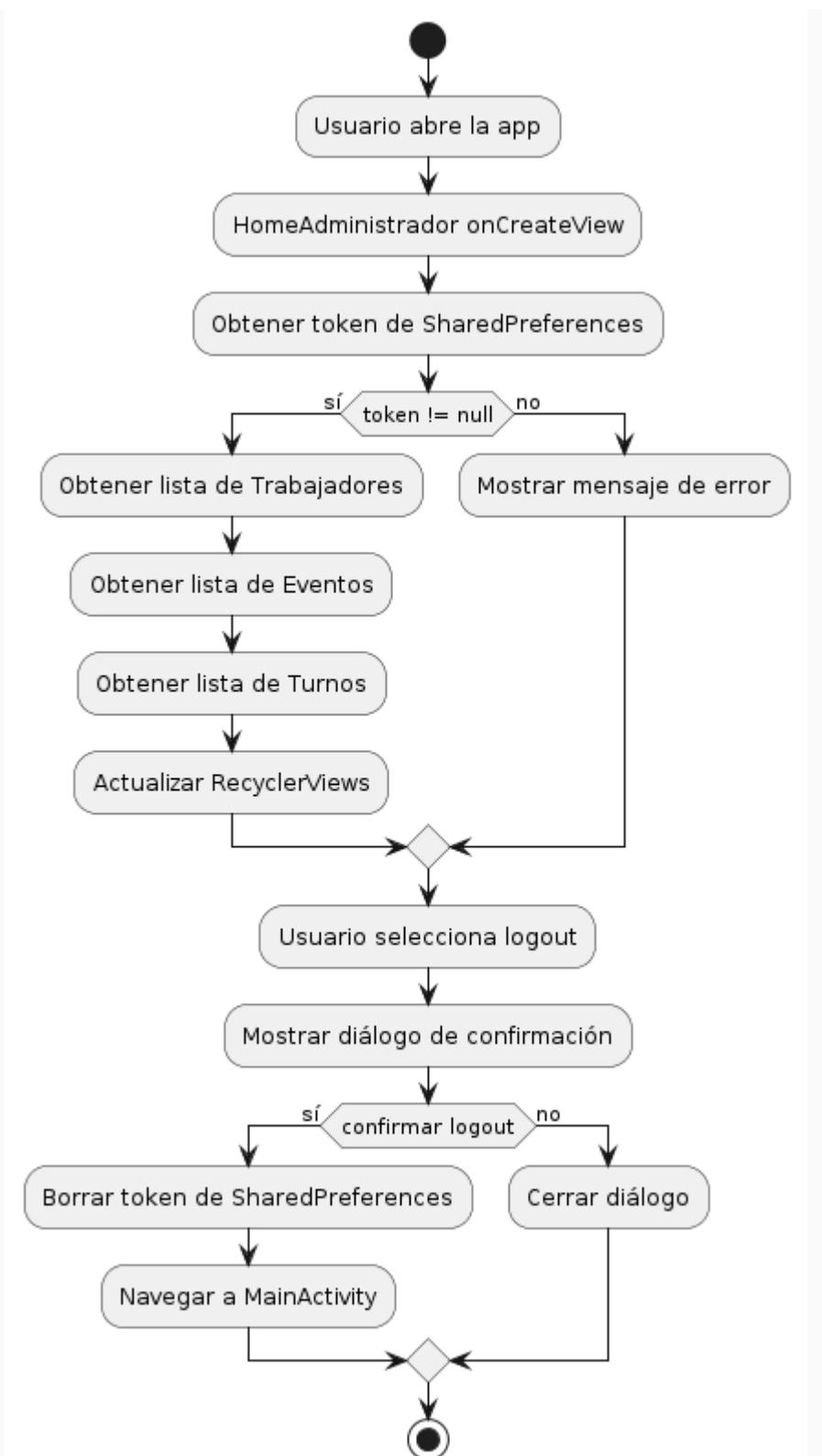


El fragmento HomeAdministrador interactúa con tres managers diferentes: TrabajadorManager, EventosManager y TurnoManager. Cada manager tiene su propio callback (TrabajadorCallback, EventosCallback, y TurnoCallback) para manejar las respuestas al realizar operaciones de red, como obtener la lista de trabajadores, eventos del día, y turnos actuales.

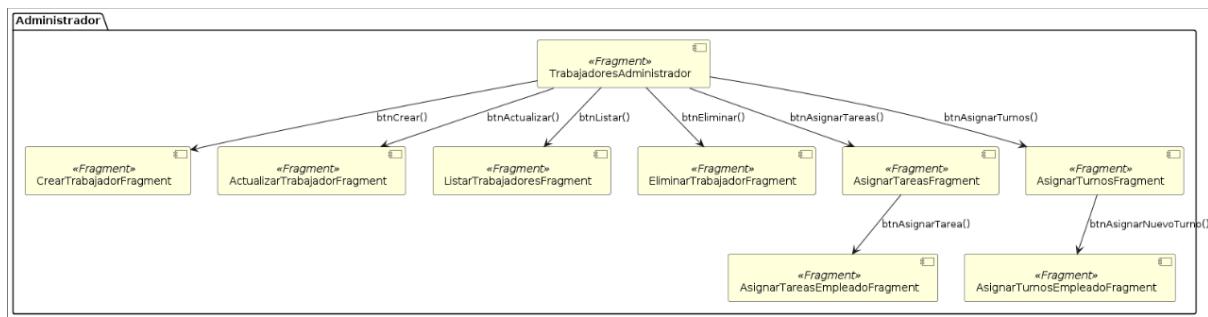
Cuando se carga el fragmento, se llaman los métodos cargarTrabajadores(), cargarEventos() y cargarTurnos(), que inician las solicitudes a los respectivos managers para obtener los datos necesarios. Cada manager, a su vez, utiliza su callback correspondiente para manejar la respuesta y actualizar la interfaz de usuario con los datos recibidos.

Adicionalmente, el FloatingActionButton (fabLogout) está configurado para mostrar un diálogo de confirmación (LogoutDialog) cuando se hace clic en él. Si el usuario confirma la acción de logout, el diálogo elimina el token de autenticación guardado en SharedPreferences y redirige al usuario de vuelta a MainActivity, efectivamente cerrando la sesión.





TrabajadoresFragment



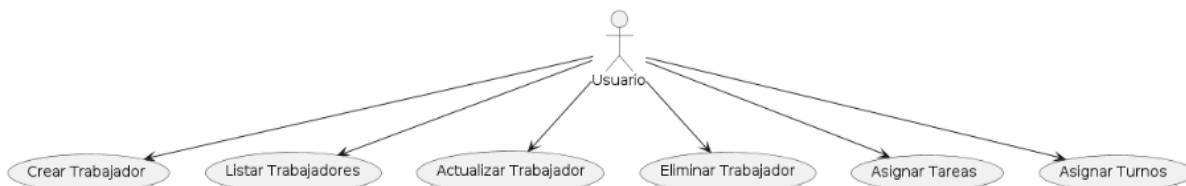
En este diagrama de flujo, representamos el flujo de trabajo del administrador y las diferentes interacciones con los fragmentos responsables de la gestión de los trabajadores.

El TrabajadoresAdministrador es el fragmento principal desde donde el administrador puede navegar a diferentes funcionalidades relacionadas con la gestión de trabajadores. Este fragmento actúa como un hub central desde el cual el administrador puede elegir la acción que desea realizar.

Desde el TrabajadoresAdministrador, el administrador puede acceder al CrearTrabajadorFragment. Este fragmento permite al administrador registrar nuevos trabajadores en el sistema, proporcionando un formulario para ingresar los detalles del trabajador como DNI, nombre, apellidos, teléfono, correo electrónico, contraseña y puesto.

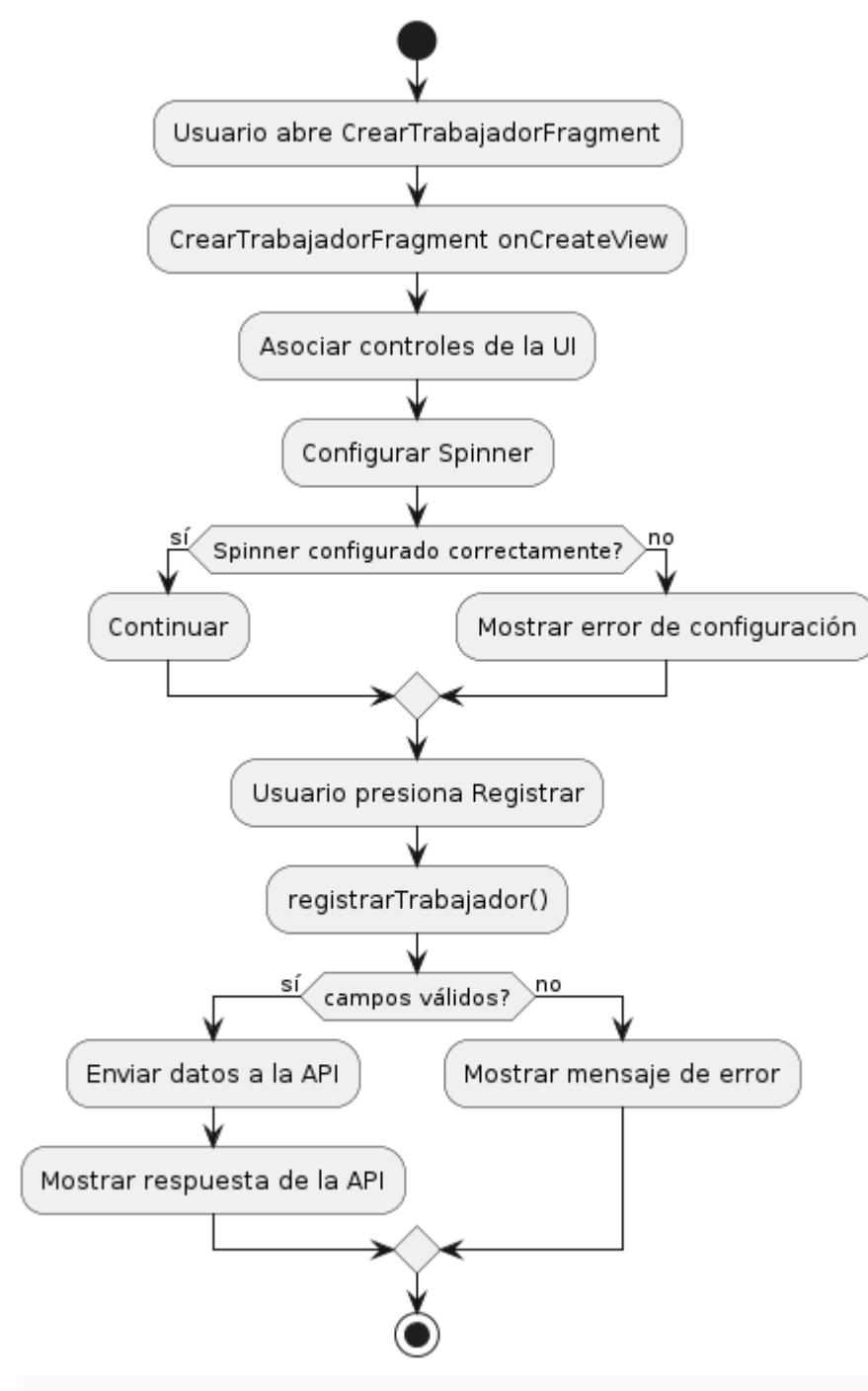
El administrador también puede navegar al ListarTrabajadoresFragment desde el TrabajadoresAdministrador. Este fragmento muestra una lista de todos los trabajadores registrados en el sistema. Además, incluye una funcionalidad de búsqueda que permite filtrar los trabajadores por nombre, apellidos, puesto o correo electrónico.

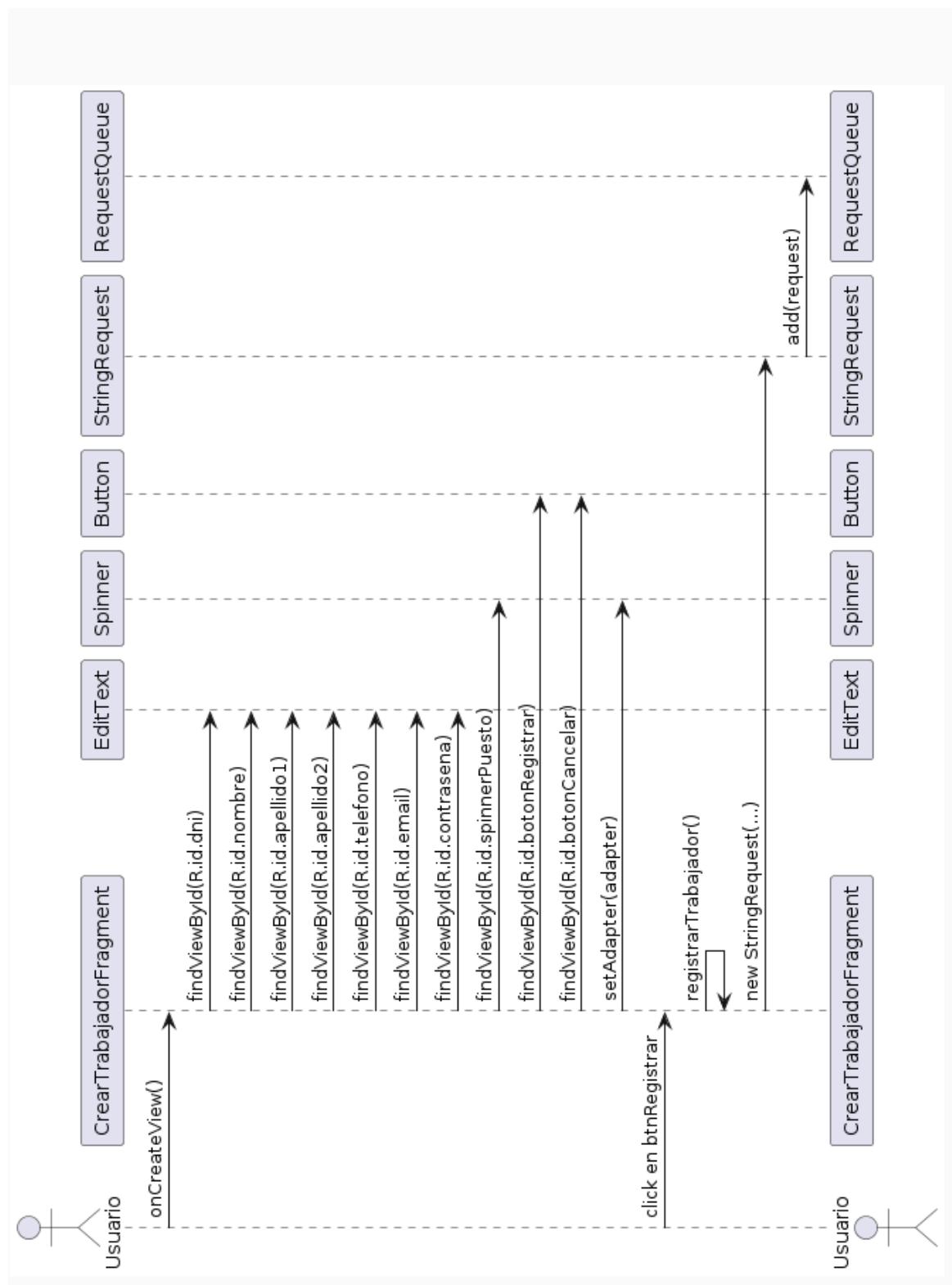
Para actualizar la información de un trabajador existente, el administrador puede acceder al ActualizarTrabajadorFragment desde el TrabajadoresAdministrador. Este fragmento permite al administrador buscar un trabajador específico por DNI y actualizar sus datos según sea necesario.





CrearTrabajadorFragment

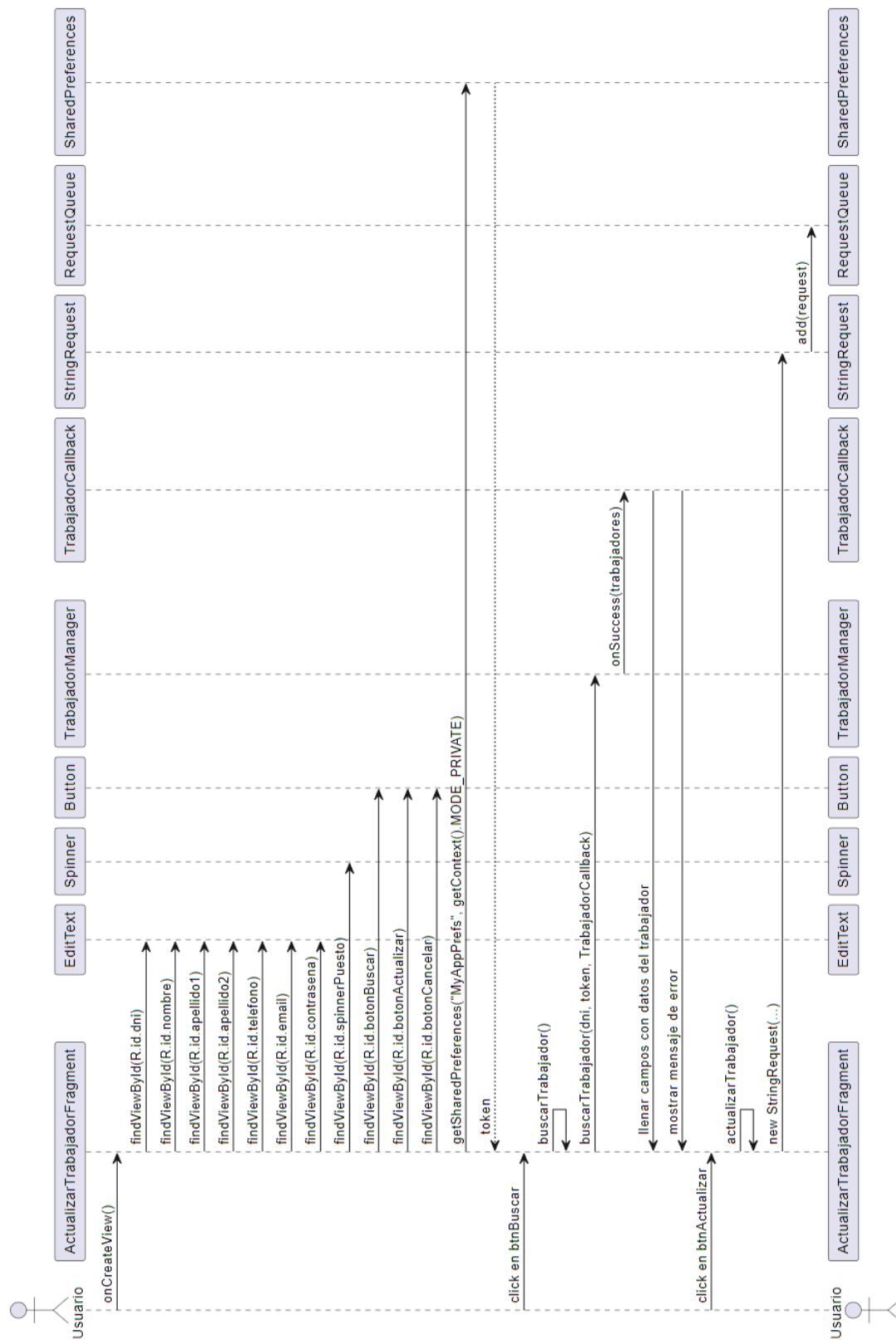






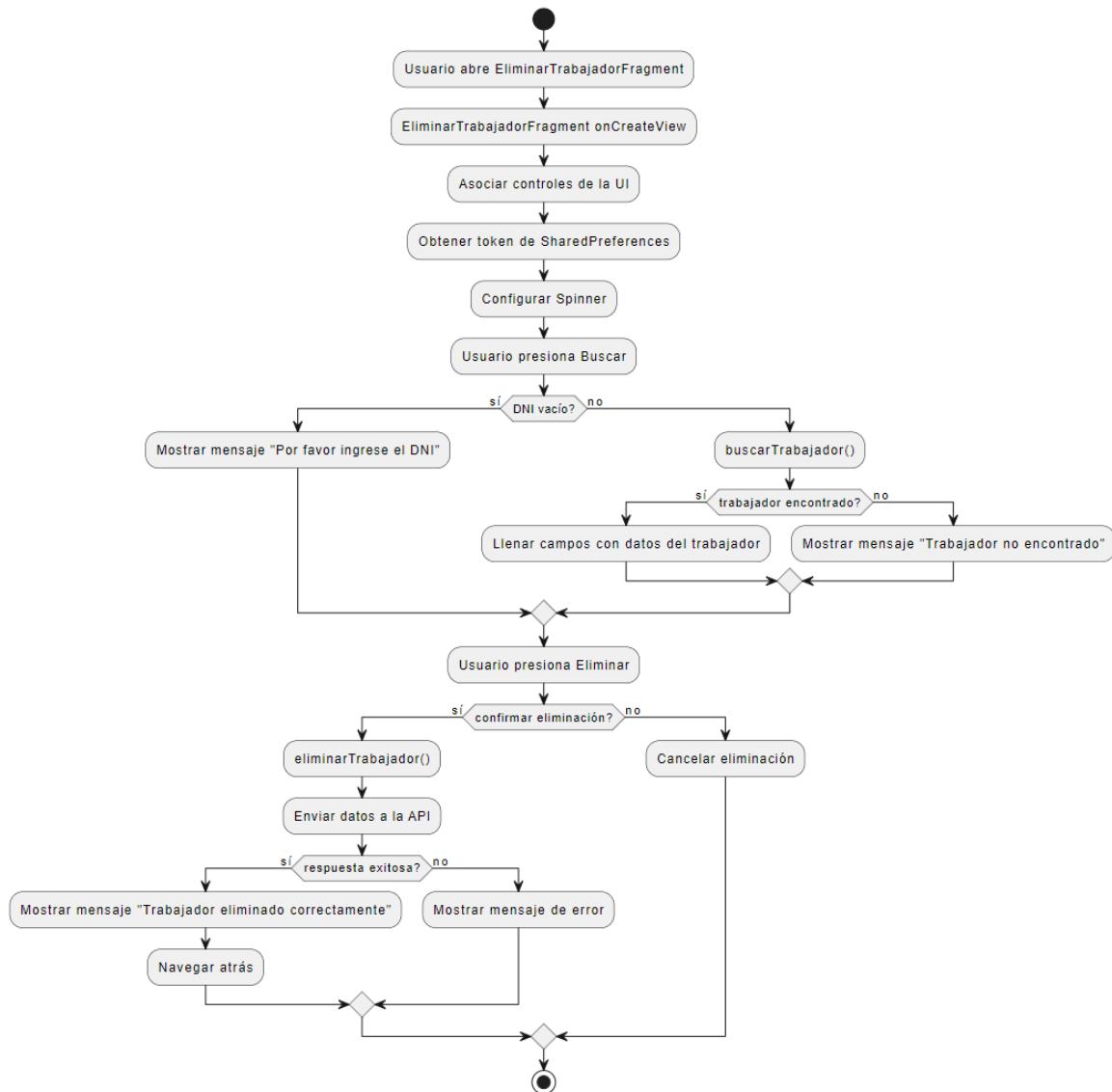
ActualizarTrabajadorFragment

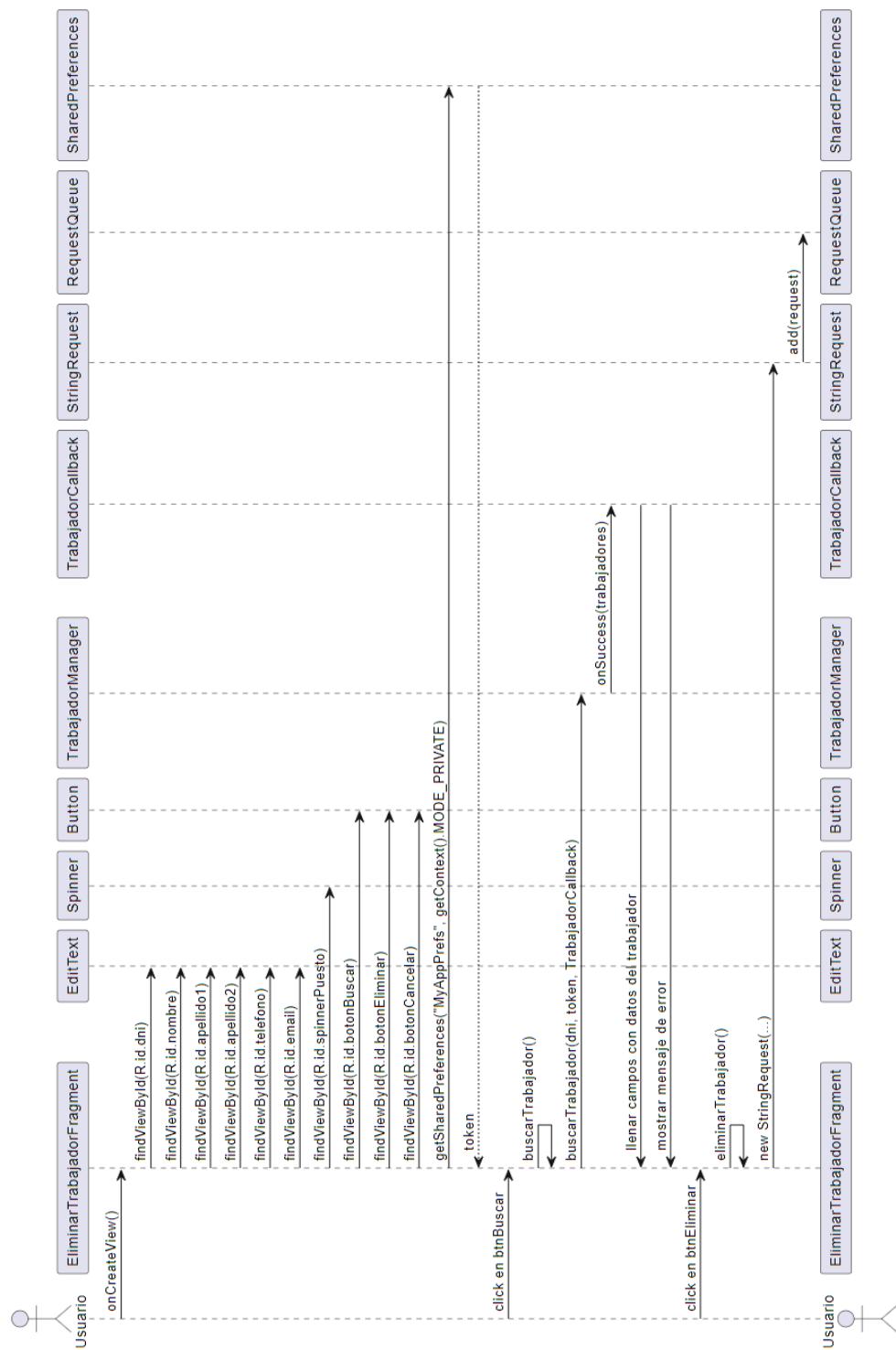




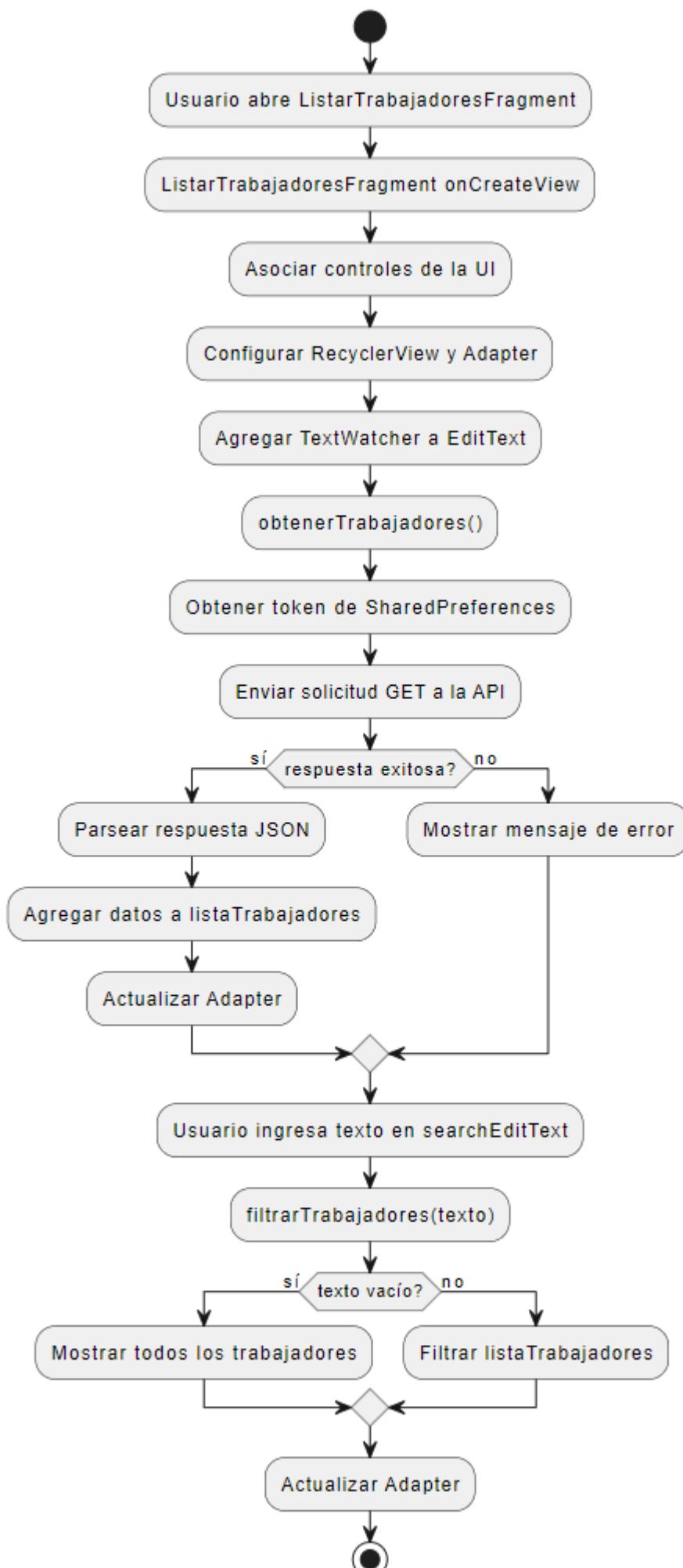


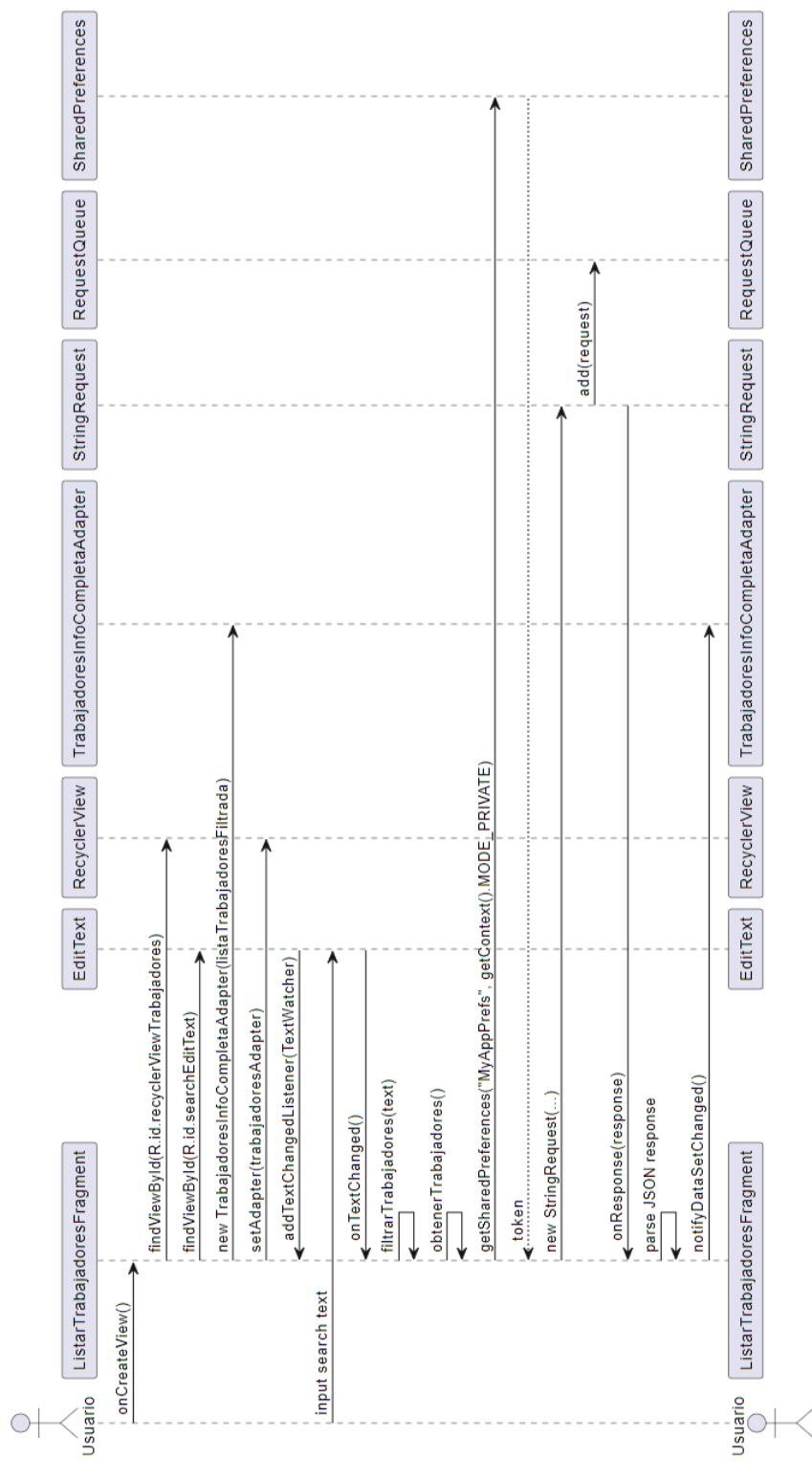
EliminarTrabajadorFragment



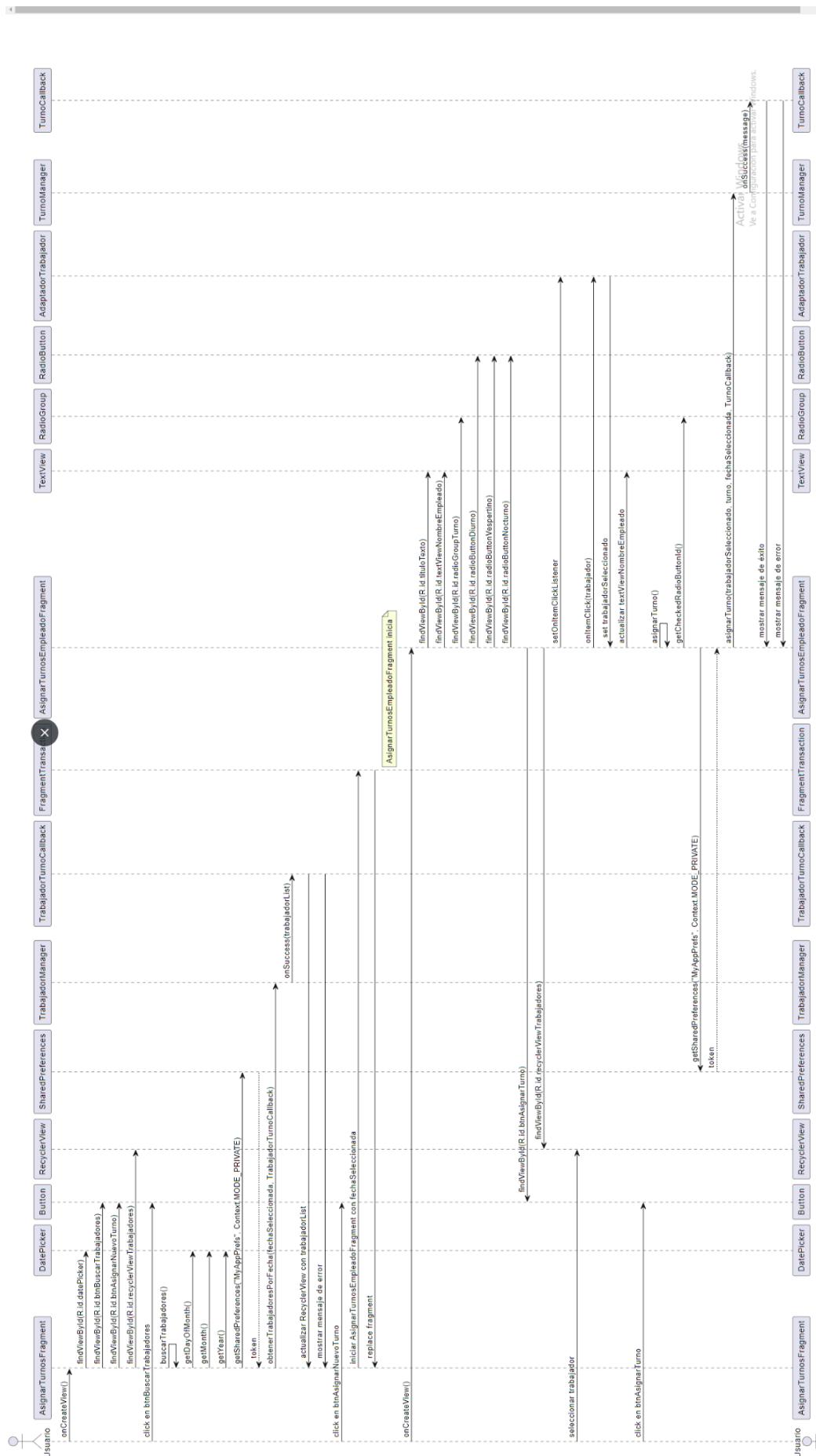


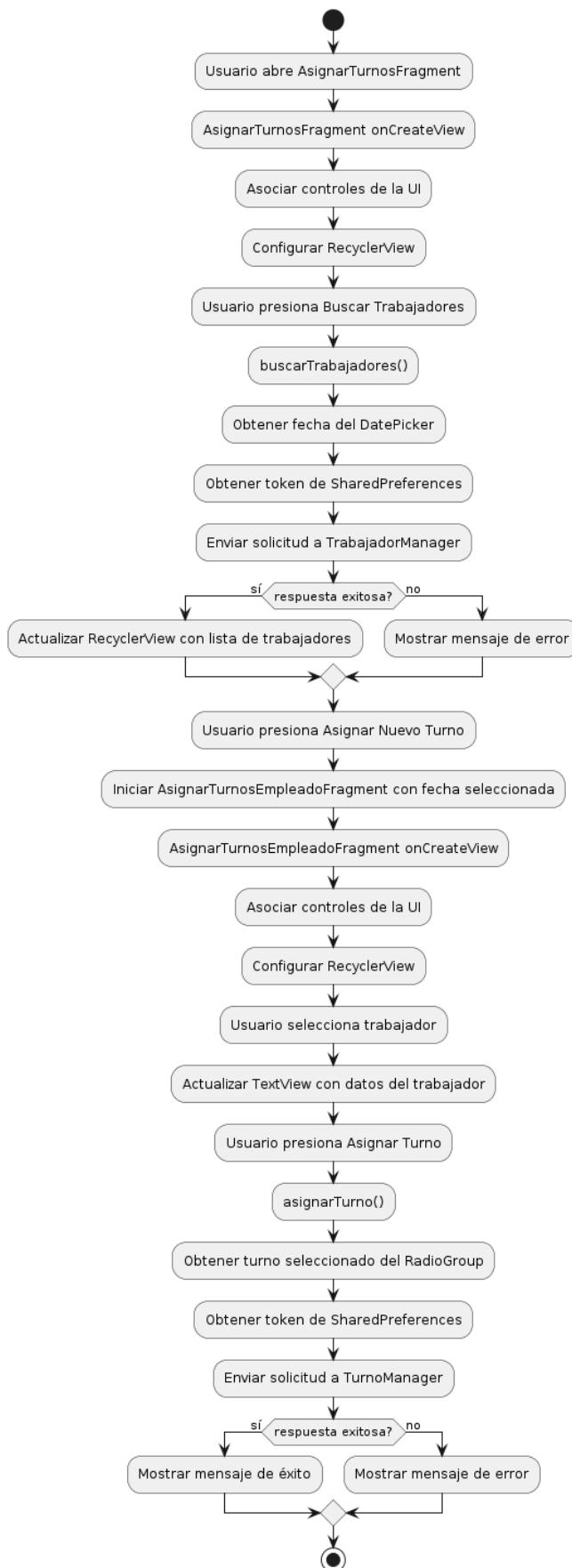
ListarTrabajadoresFragment





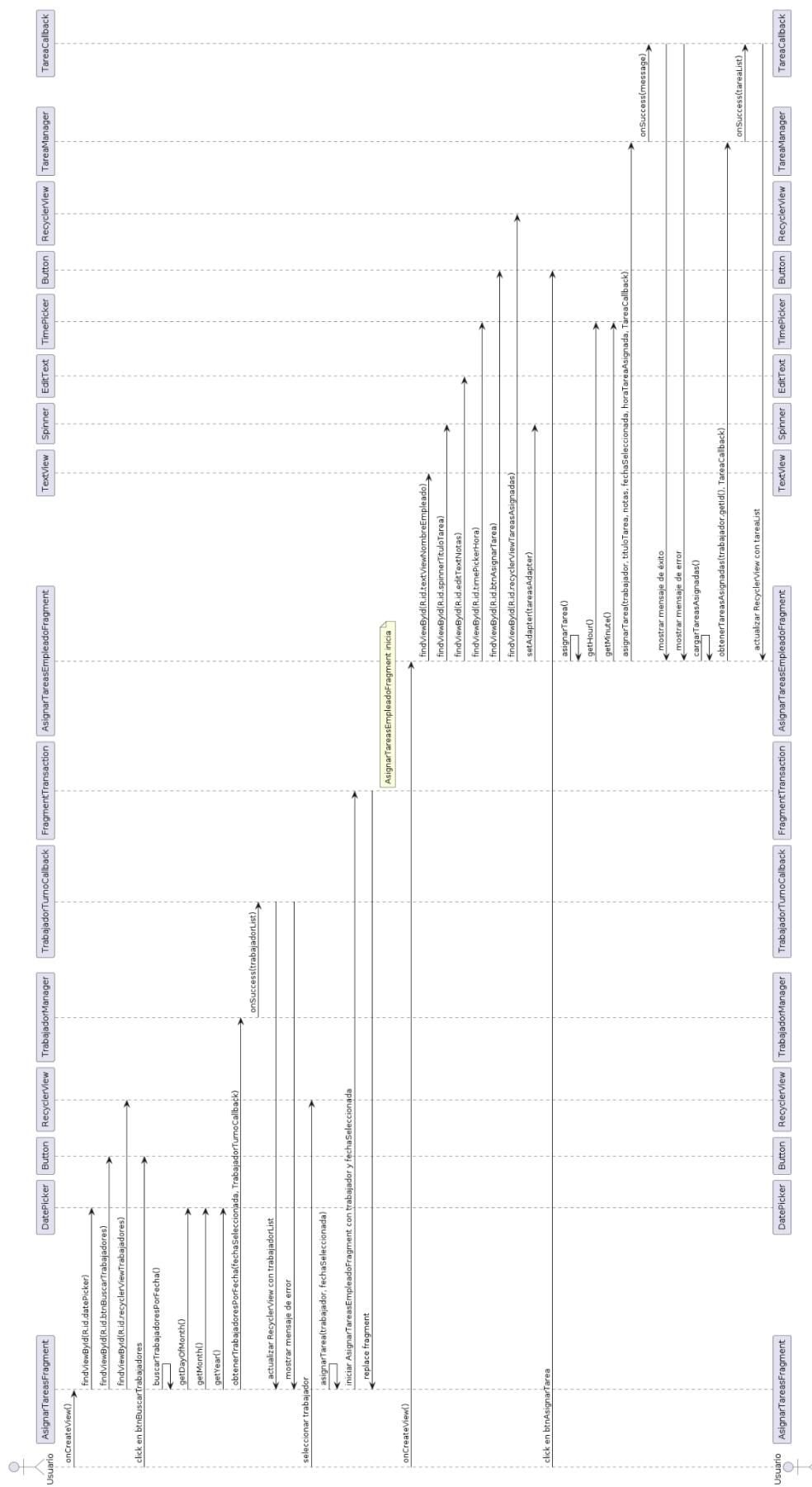
AsignarTurnosFragment y AsignarTurnosEmpleadoFragment



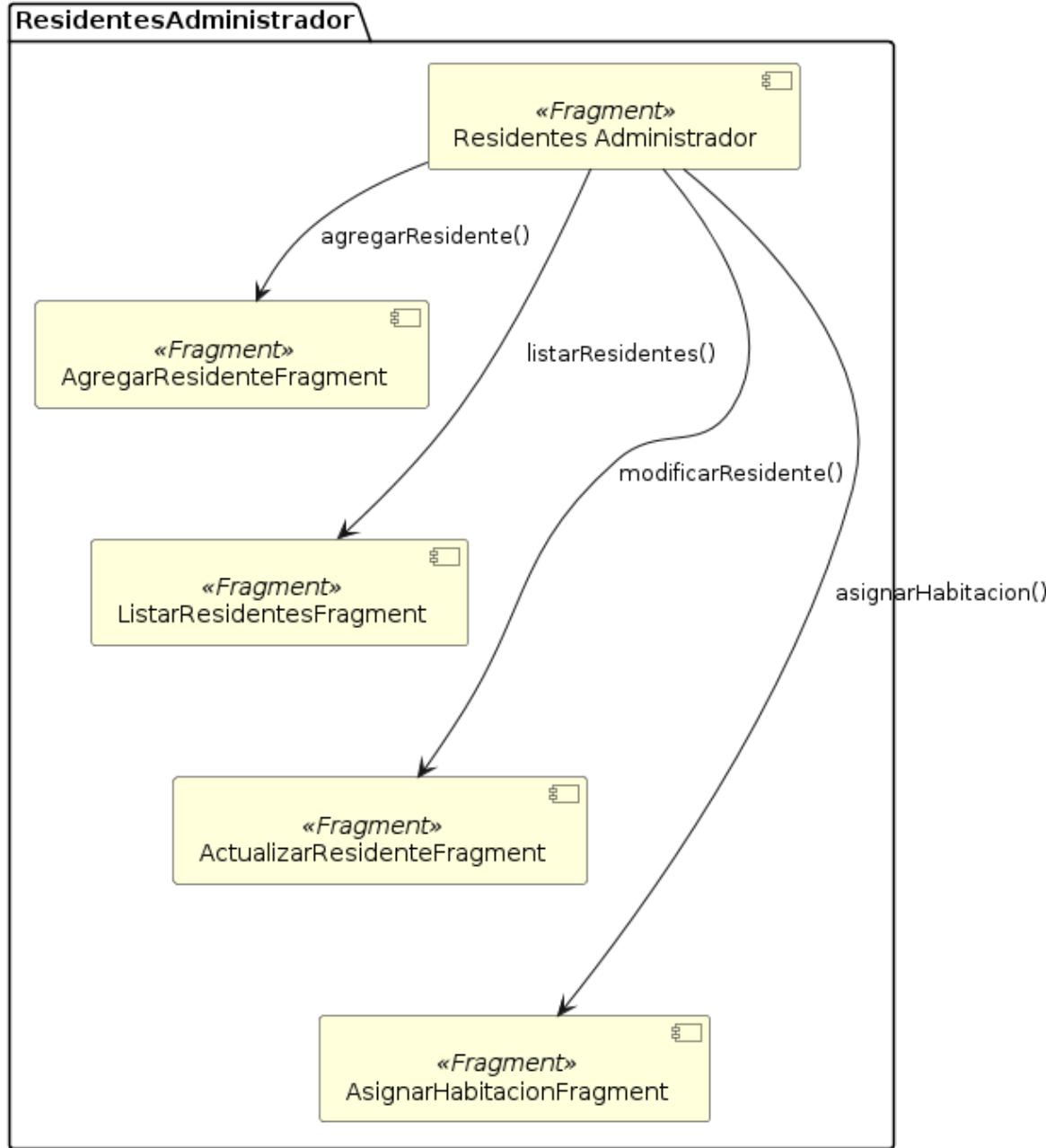




AsignarTareasFragment y AsignarTareasEmpleadoFragment



ResidentesAdministrador



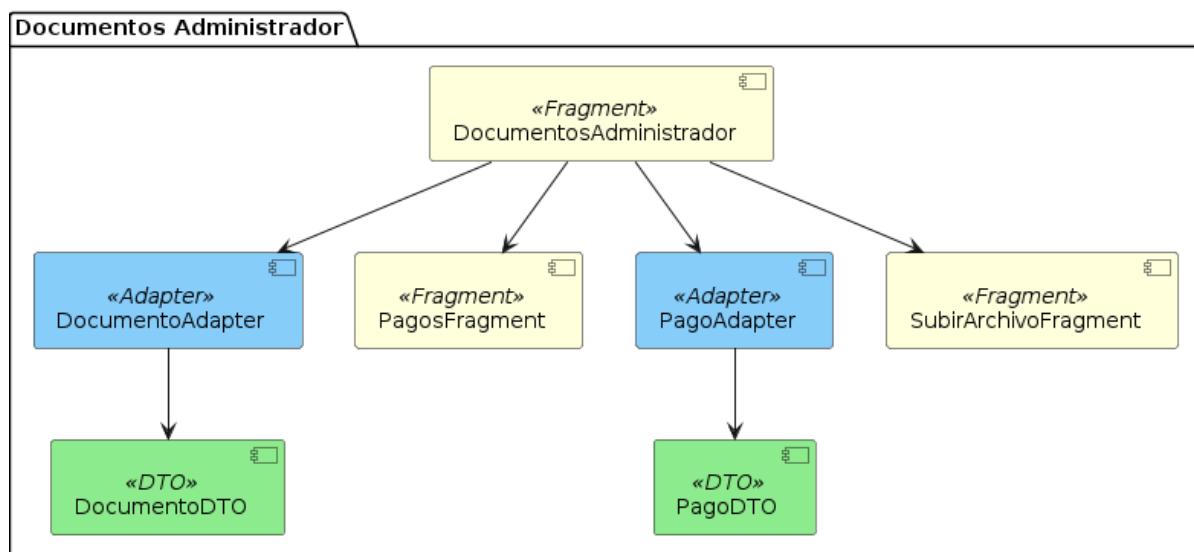
Si el administrador necesita eliminar a un trabajador del sistema, puede ir al EliminarTrabajadorFragment desde el TrabajadoresAdministrador. Este fragmento proporciona una interfaz para buscar y eliminar a un trabajador seleccionado.

Además de la gestión básica de trabajadores, el administrador puede asignar tareas y turnos. Desde el TrabajadoresAdministrador, el administrador puede ir al AsignarTareasFragment para asignar tareas a los trabajadores. En este fragmento,

el administrador puede seleccionar una fecha y ver la lista de trabajadores disponibles para esa fecha. Desde aquí, puede navegar a AsignarTareasEmpleadoFragment para asignar tareas específicas a un trabajador seleccionado.

De manera similar, el administrador puede ir al AsignarTurnosFragment desde el TrabajadoresAdministrador para asignar turnos a los trabajadores. En este fragmento, el administrador puede seleccionar una fecha y ver la lista de trabajadores disponibles para esa fecha. Desde aquí, puede navegar a AsignarTurnosEmpleadoFragment para asignar un turno específico a un trabajador seleccionado.

DocumentosAdministrador



El componente principal es el fragmento DocumentosAdministrador, que actúa como el punto de entrada para las funcionalidades relacionadas con documentos. Este fragmento tiene conexiones con varios otros fragmentos y adaptadores que realizan tareas específicas.

El DocumentoAdapter es un adaptador que se utiliza para mostrar una lista de documentos. Este adaptador permite realizar acciones sobre los documentos, como eliminarlos o descargarlos, y está vinculado con el DTO (Objeto de Transferencia de Datos) DocumentoDTO, que define la estructura de los datos de los documentos.

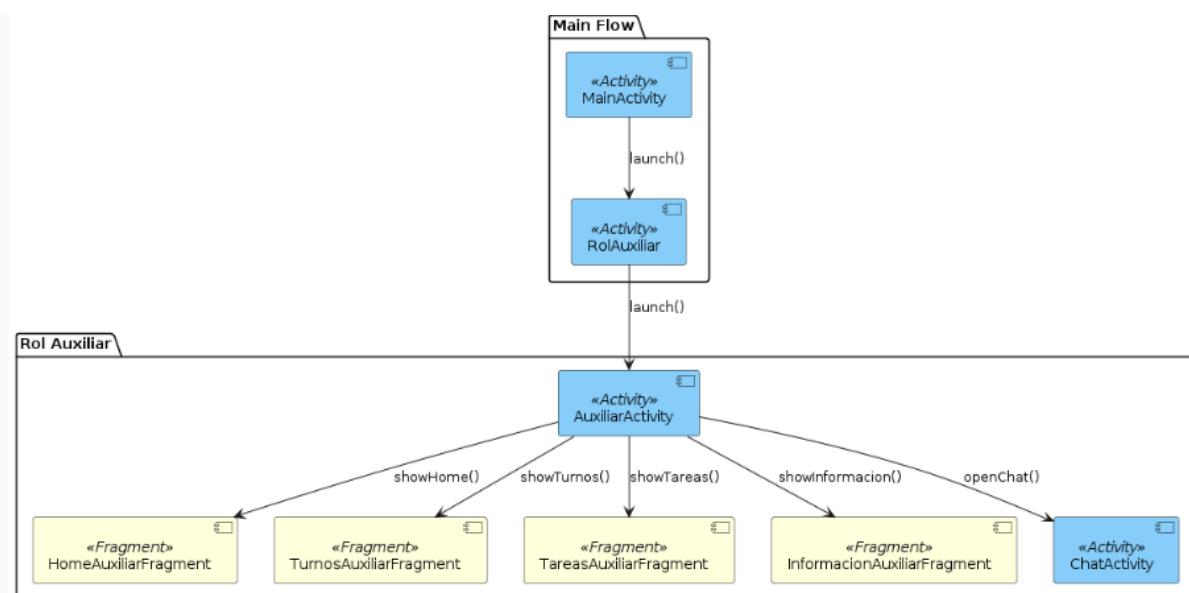
El PagosFragment es otro fragmento importante que se utiliza para mostrar el estado de los pagos de los residentes. Este fragmento muestra una lista de pagos y utiliza el PagoAdapter para gestionar la visualización de esta lista. Al igual que el DocumentoAdapter, el PagoAdapter está vinculado con un DTO, en este caso,



PagoDTO, que define la estructura de los datos de los pagos.

Finalmente, el SubirArchivoFragment es el fragmento que permite a los usuarios subir nuevos archivos a la aplicación. Los usuarios pueden seleccionar un archivo desde su dispositivo, ingresar detalles adicionales como el DNI, título y descripción del archivo, y luego subirlo al servidor.

4.1.3.1.2. Flujo Rol Auxiliar



En la estructura de navegación del rol auxiliar en la aplicación, se tiene una actividad principal llamada `AuxiliarActivity`, que actúa como el contenedor principal para varios fragmentos que gestionan diferentes funcionalidades específicas para los auxiliares. Esta actividad proporciona la estructura de navegación utilizando un `BottomNavigationView` y un `FloatingActionButton` para la navegación y la interacción.

La actividad `AuxiliarActivity` maneja cuatro fragmentos principales:

1. **HomeAuxiliarFragment:** Este es el fragmento principal que se muestra inicialmente cuando el auxiliar inicia sesión. Aquí se puede presentar un resumen de la información relevante para el auxiliar, como notificaciones o actualizaciones recientes.
2. **TurnosAuxiliarFragment:** Este fragmento permite a los auxiliares ver y gestionar sus turnos de trabajo. Aquí se puede visualizar el calendario de turnos, detalles de los turnos asignados y cualquier cambio relacionado con los turnos.



3. **TareasAuxiliarFragment:** En este fragmento, los auxiliares pueden ver y gestionar las tareas asignadas. Pueden ver una lista de tareas, marcar tareas como completadas y recibir nuevas asignaciones de tareas.
4. **InformacionAuxiliarFragment:** Este fragmento proporciona acceso a la información importante para los auxiliares. Puede incluir manuales, procedimientos, políticas y cualquier otro documento relevante que los auxiliares necesiten consultar.

Además de estos fragmentos, la actividad principal AuxiliarActivity también proporciona acceso a una funcionalidad de chat a través de un FloatingActionButton. Al hacer clic en este botón, se abre la actividad ChatActivity, donde los auxiliares pueden comunicarse con otros miembros del personal o recibir soporte.

El flujo de navegación entre estos fragmentos se maneja mediante un BottomNavigationView, que permite a los usuarios cambiar entre los fragmentos principales con facilidad. Cada elemento del BottomNavigationView está asociado con un fragmento específico, y al seleccionarlo, la actividad reemplaza el fragmento actualmente visible con el seleccionado.

Diagrama de Flujo

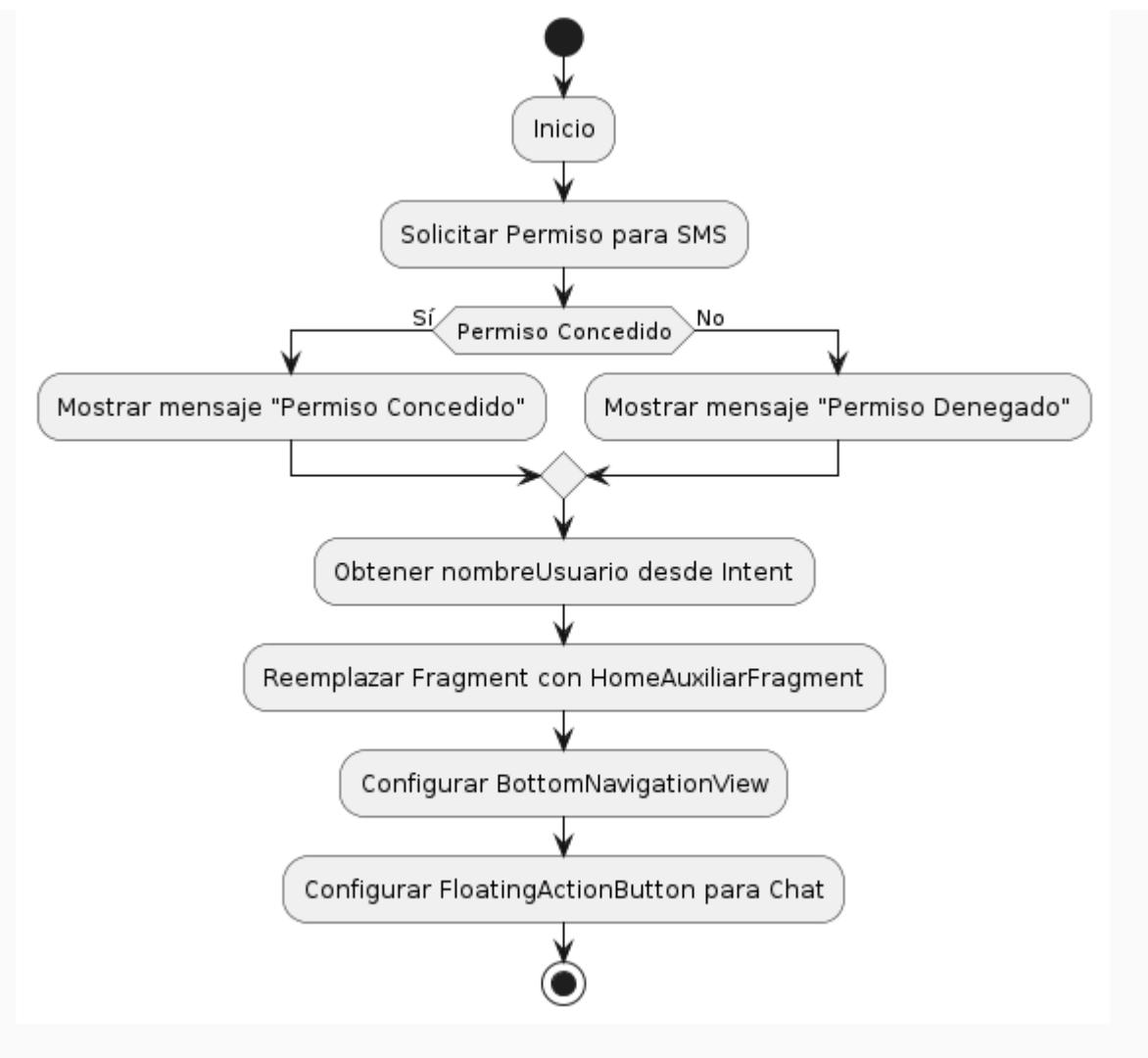


Diagrama de Casos de Uso

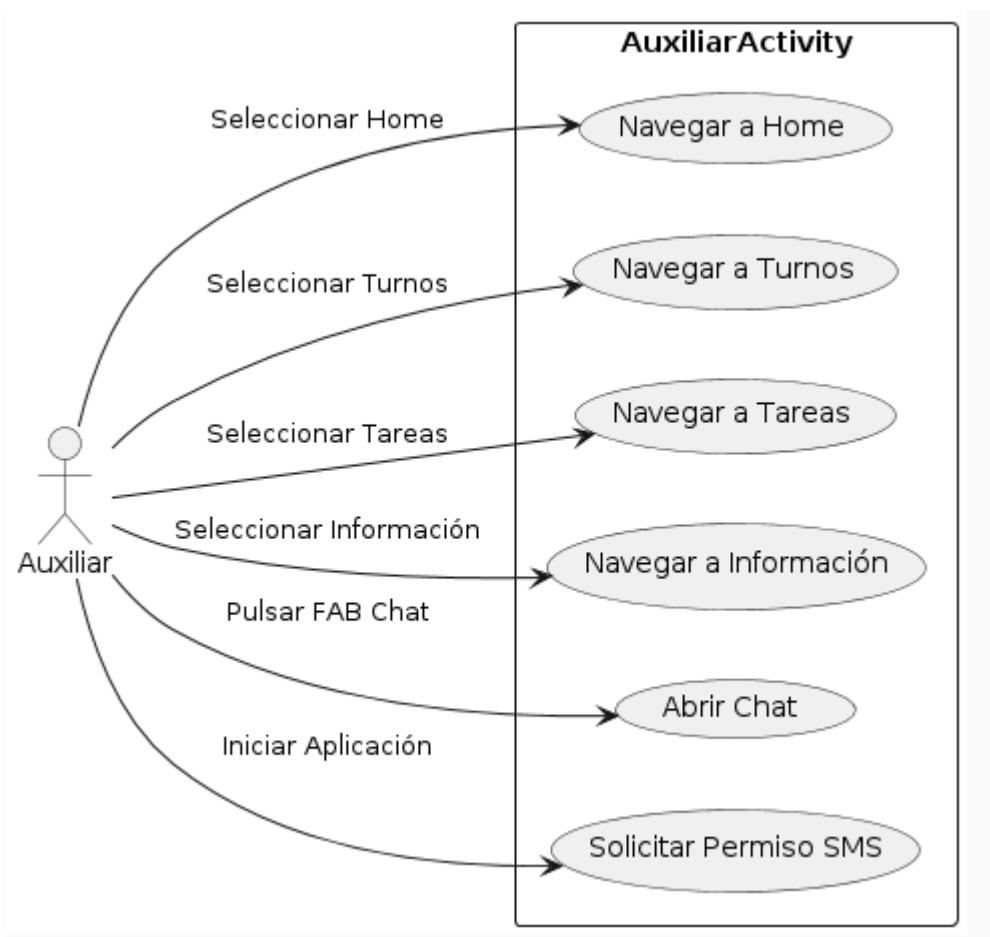


Diagrama de Secuencia





HomeAuxiliarFragment

Este apartado del software está implementado en el fragmento HomeAuxiliarFragment. Utiliza la biblioteca Volley para gestionar las solicitudes HTTP y SharedPreferences para manejar las credenciales del usuario. Los elementos de la interfaz de usuario (UI) incluyen botones para registrar la entrada y salida, así como un botón para cerrar sesión. Cada interacción del usuario con estos botones está respaldada por un proceso que garantiza la confirmación de acciones importantes y maneja las respuestas del servidor de manera adecuada, proporcionando feedback inmediato al usuario.

Estos diagramas y la explicación detallada proporcionan una comprensión clara de cómo el HomeAuxiliarFragment facilita las operaciones diarias del auxiliar, asegurando un flujo de trabajo eficiente y fiable dentro de la aplicación.

Diagrama de clases

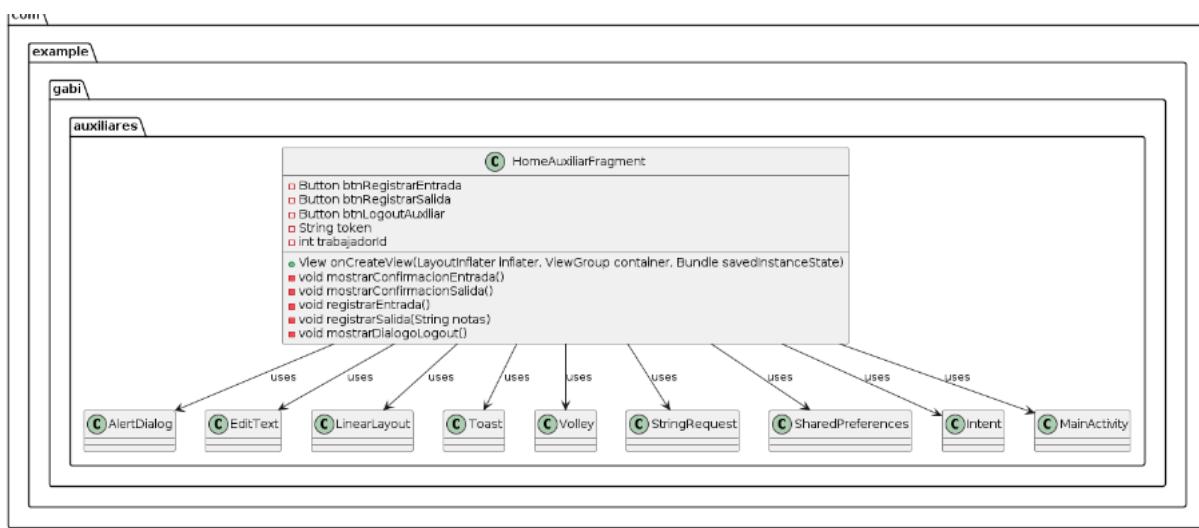


Diagrama de secuencia

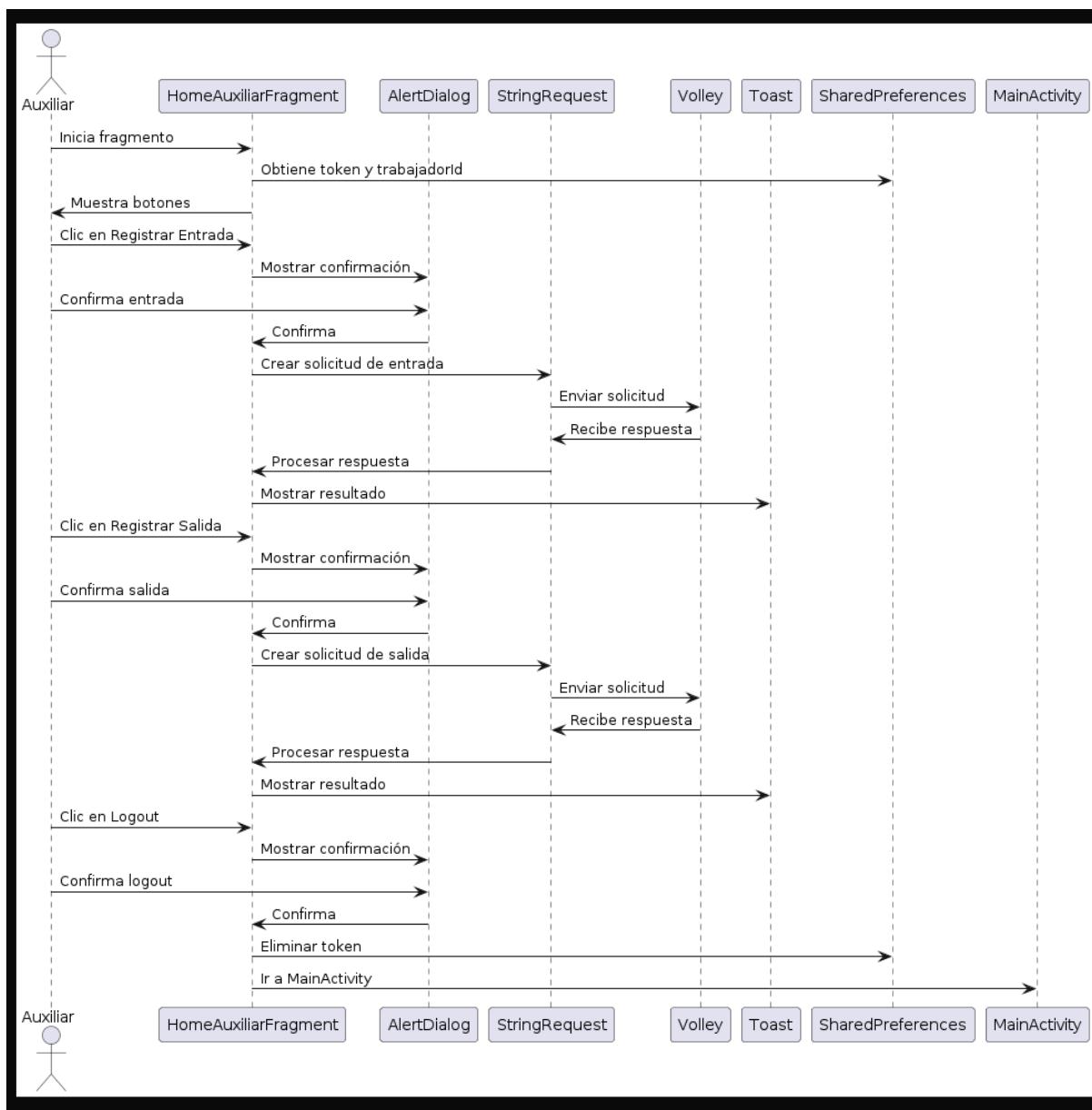
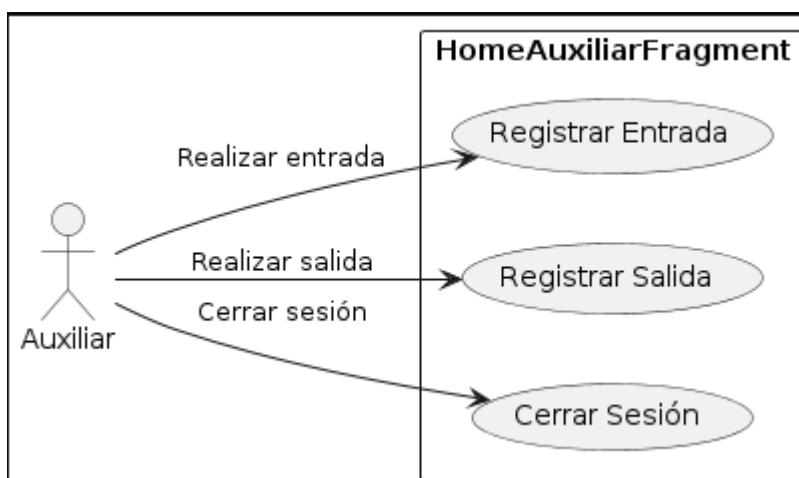


Diagrama de Casos de Uso





El HomeAuxiliarFragment está diseñado para facilitar las tareas del auxiliar de forma eficiente y organizada. Este fragmento ofrece tres funcionalidades principales: registrar la entrada, registrar la salida y cerrar sesión.

1. **Registrar Entrada:** Al iniciar el HomeAuxiliarFragment, el auxiliar puede optar por registrar su entrada. Al hacer clic en el botón "Registrar Entrada", se presenta un cuadro de diálogo de confirmación. Una vez confirmado, la aplicación envía una solicitud HTTP POST al servidor para registrar la entrada del trabajador, utilizando la biblioteca Volley para gestionar la solicitud. Si la respuesta del servidor es exitosa, se notifica al usuario mediante un mensaje emergente (Toast). De lo contrario, se muestra un mensaje de error.
2. **Registrar Salida:** Similar al proceso de registro de entrada, el auxiliar puede registrar su salida a través del botón "Registrar Salida". Este proceso incluye la posibilidad de añadir notas opcionales. Al confirmar la salida, se envía una solicitud HTTP POST al servidor con los detalles pertinentes. La respuesta del servidor determina si la operación fue exitosa o si hubo un error, notificando al usuario en consecuencia.
3. **Cerrar Sesión:** El HomeAuxiliarFragment también permite al auxiliar cerrar su sesión. Al hacer clic en el botón "Cerrar Sesión", se muestra un cuadro de diálogo de confirmación. Una vez confirmado, se eliminan las credenciales almacenadas (token) en las preferencias compartidas (SharedPreferences) y se redirige al usuario a la pantalla principal de la aplicación (MainActivity).

TurnosAuxiliarFragment

TurnosAuxiliarFragment es una clase que extiende Fragment y está diseñada para gestionar la visualización y actualización, por parte del administrador, de los turnos mensuales de los auxiliares en una vista de calendario. La clase permite a los usuarios navegar entre diferentes meses y ver los turnos asignados a cada día en el calendario.

- **Inicialización y configuración de la vista:** Configura los elementos de la interfaz de usuario como el GridLayout para el calendario, los botones para navegar entre meses y el TextView para mostrar el mes actual.
- **Navegación entre meses:** Permite a los usuarios navegar al mes anterior o siguiente, actualizando la vista del calendario en consecuencia.
- **Obtención de turnos desde el servidor:** Realiza una solicitud a un servidor para obtener los turnos del mes seleccionado y los almacena en un mapa.



- **Población del calendario:** Llena el GridLayout con los días del mes y marca los turnos correspondientes en diferentes colores según el tipo de turno (diurno, nocturno, vespertino).

Diagrama de Clases

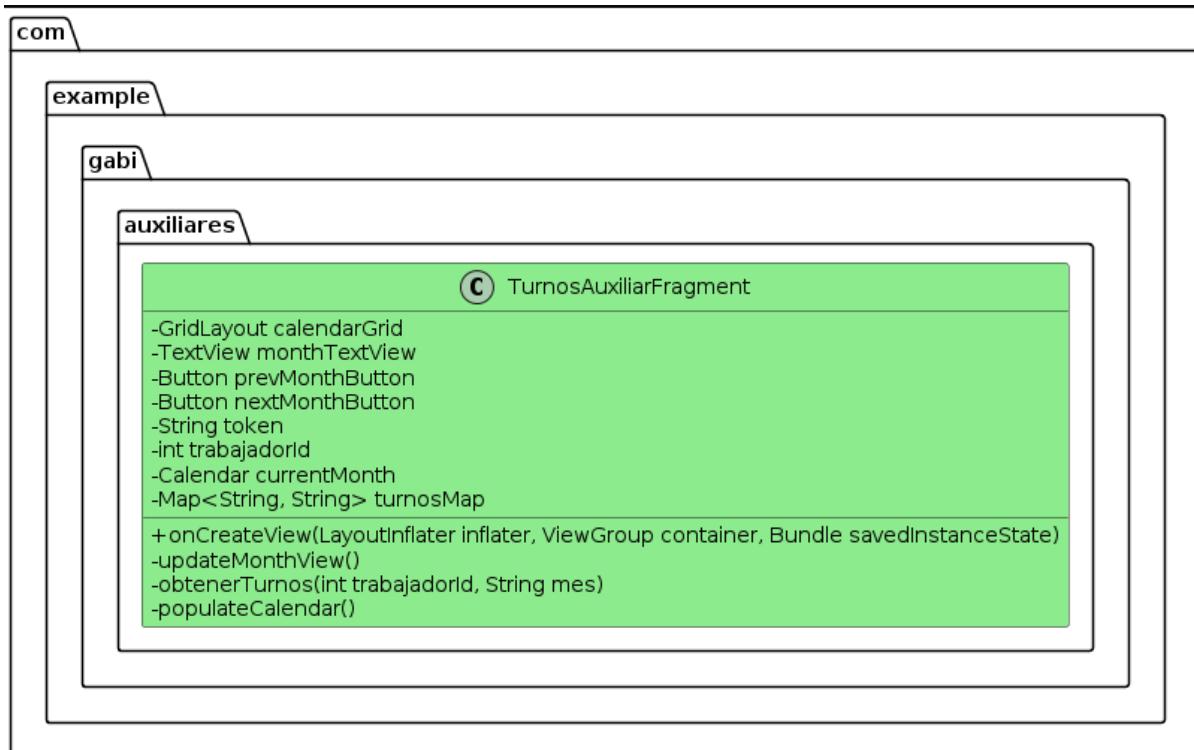


Diagrama de Secuencia

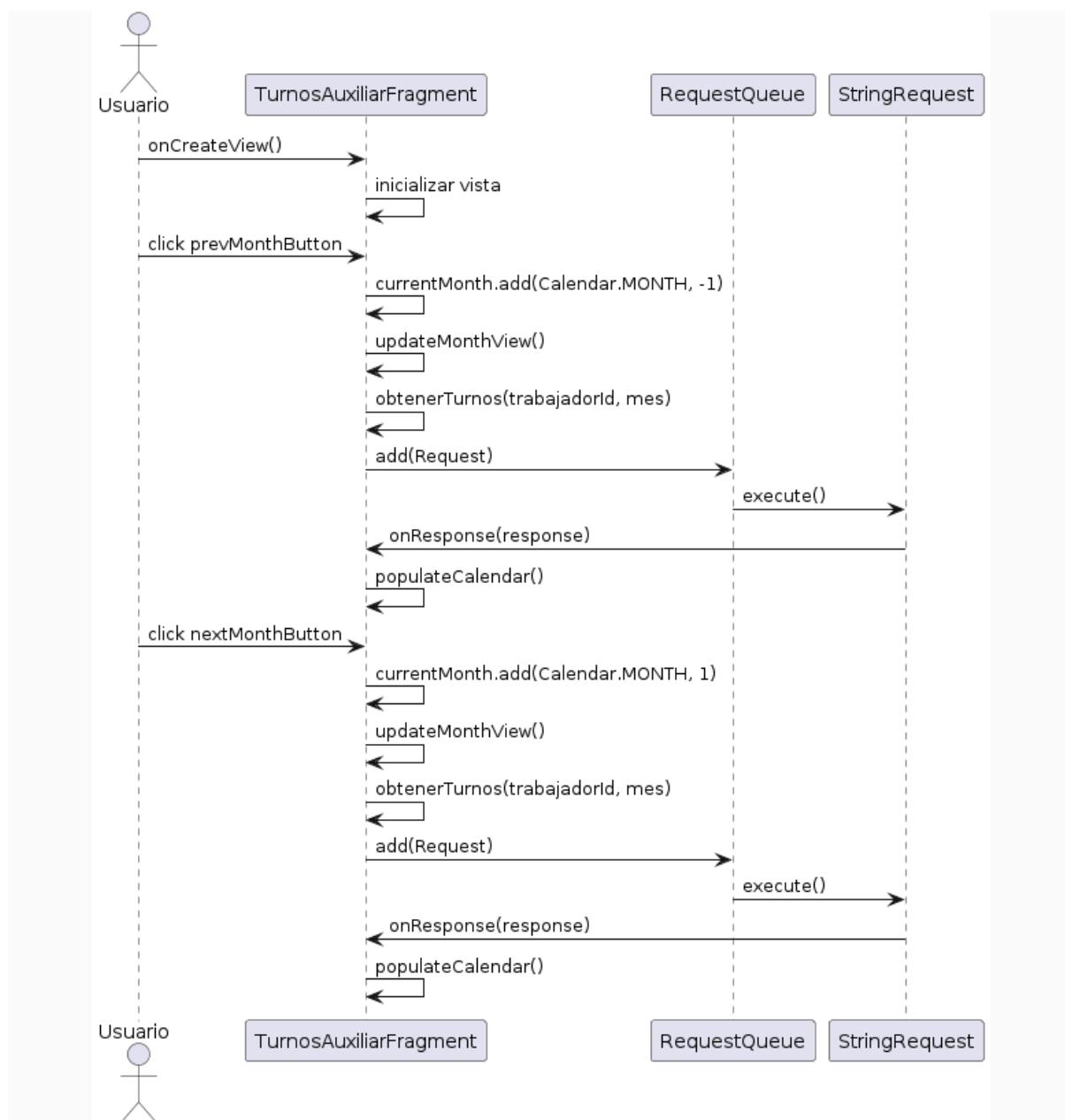


Diagrama de Flujo

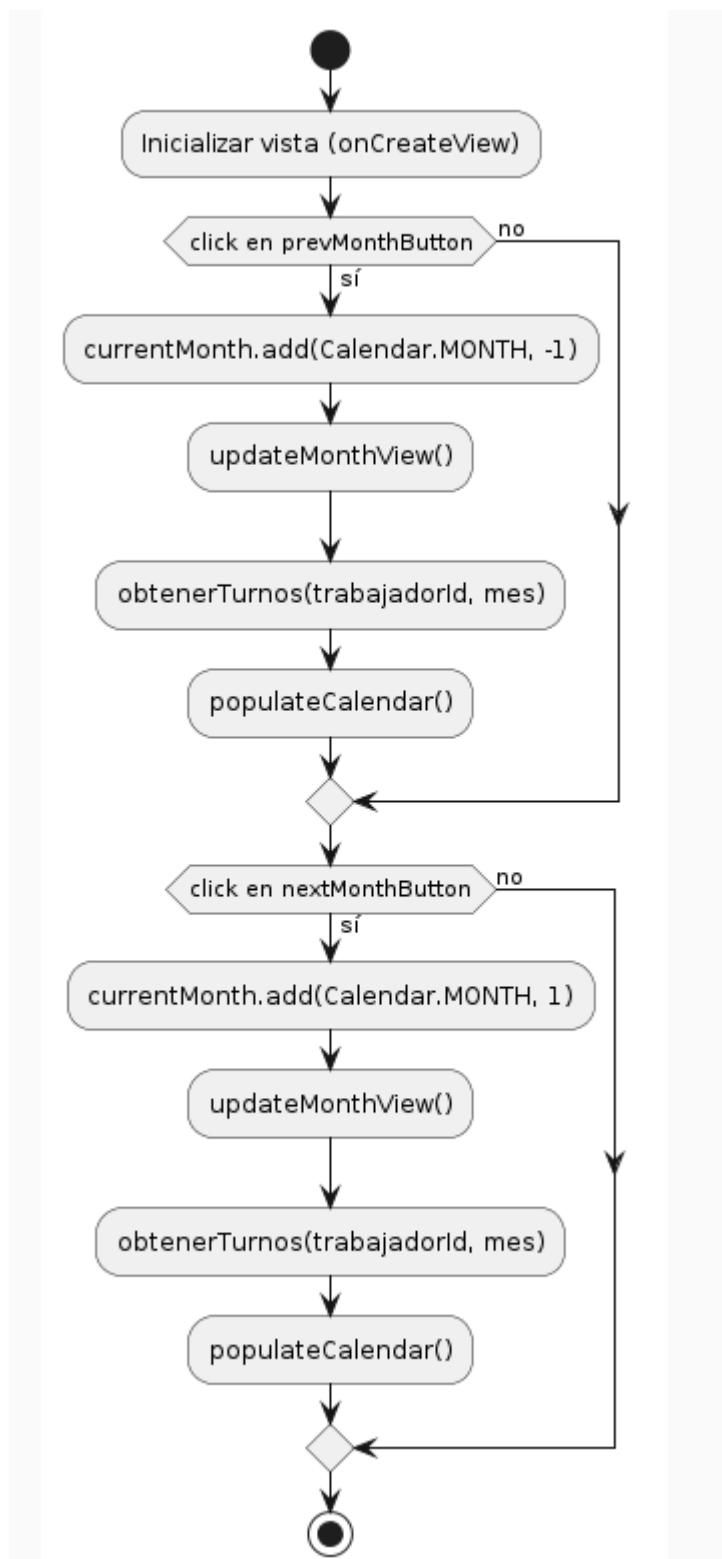
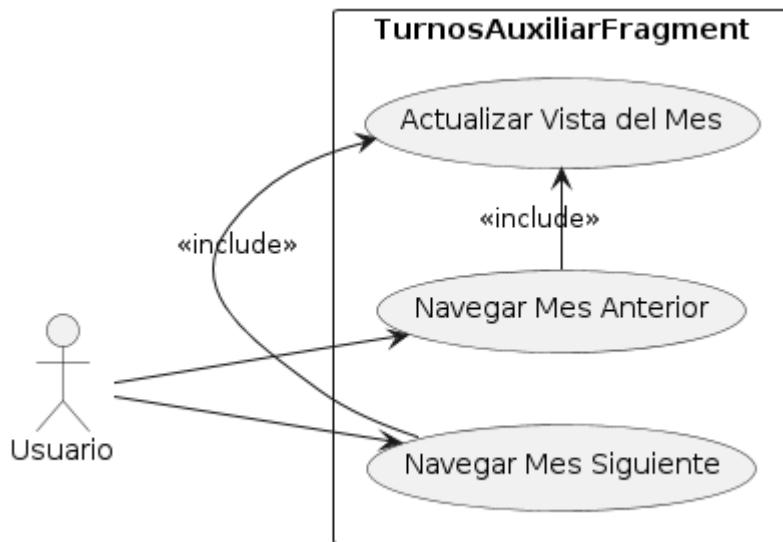


Diagrama de Casos de Uso



TareasAuxiliarFragment

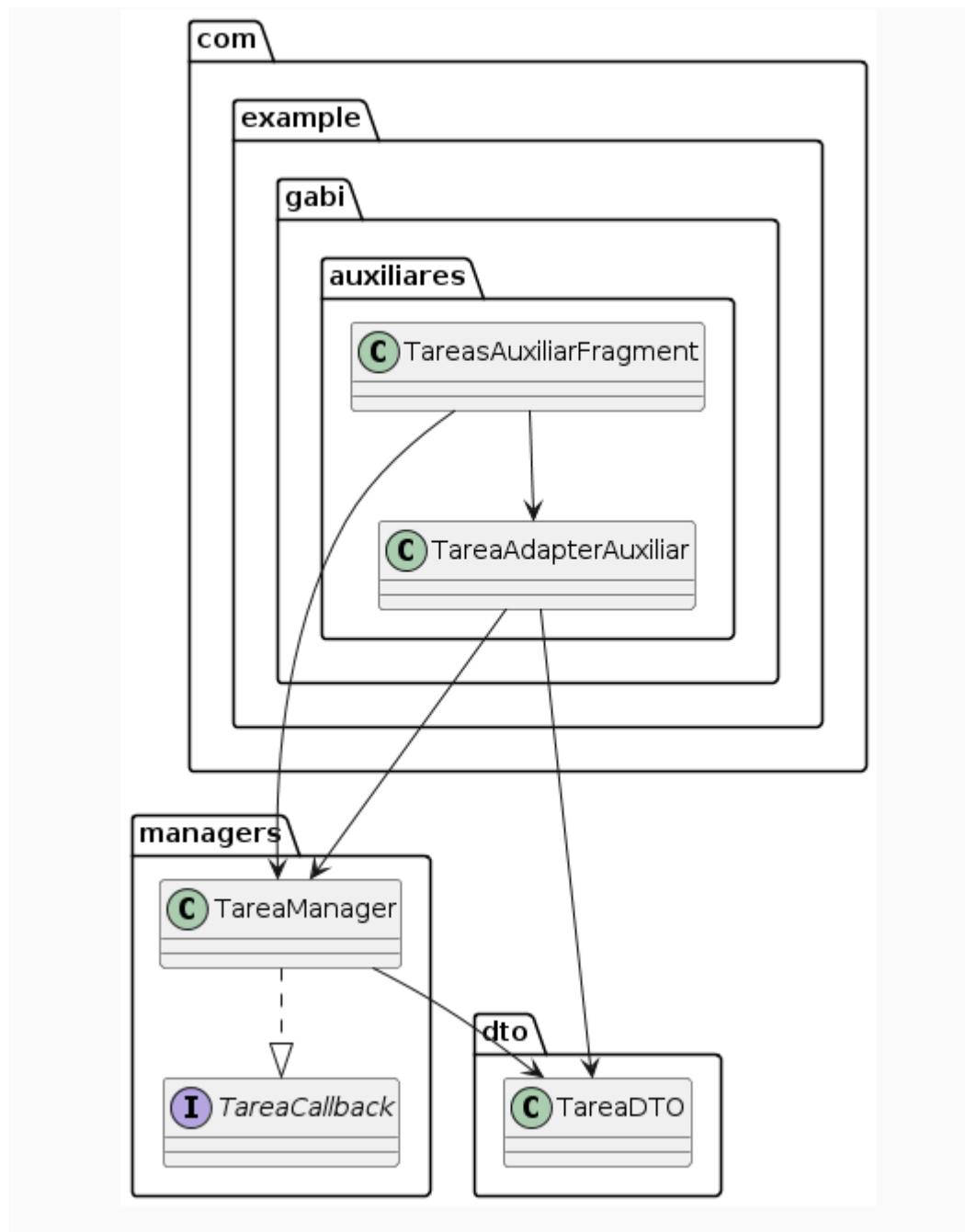
TareasAuxiliarFragment es una clase que extiende Fragment y representa la vista de las tareas asignadas a un auxiliar. Esta clase se encarga de mostrar una lista de tareas en un RecyclerView y permite al usuario marcar las tareas como completadas. Los datos de las tareas se obtienen desde un servidor mediante la clase TareaManager, la cual se comunica con el servidor utilizando la biblioteca Volley.

Métodos principales:

- **onCreateView()**: Configura la vista del fragmento, inicializa el RecyclerView y obtiene las tareas asignadas al auxiliar.
- **obtenerTareasParaAuxiliar()**: Utiliza TareaManager para obtener la lista de tareas desde el servidor.
- **onSuccess()**: Método del callback que se llama cuando las tareas se obtienen exitosamente y se configura el RecyclerView con la lista de tareas.
- **onError()**: Método del callback que se llama cuando ocurre un error al obtener las tareas.

El TareaAdapterAuxiliar es un adaptador que gestiona la visualización de cada tarea en el RecyclerView, permitiendo al usuario marcar las tareas como completadas mediante una casilla de verificación .

Diagrama de Clases de TareasAuxiliarFragment



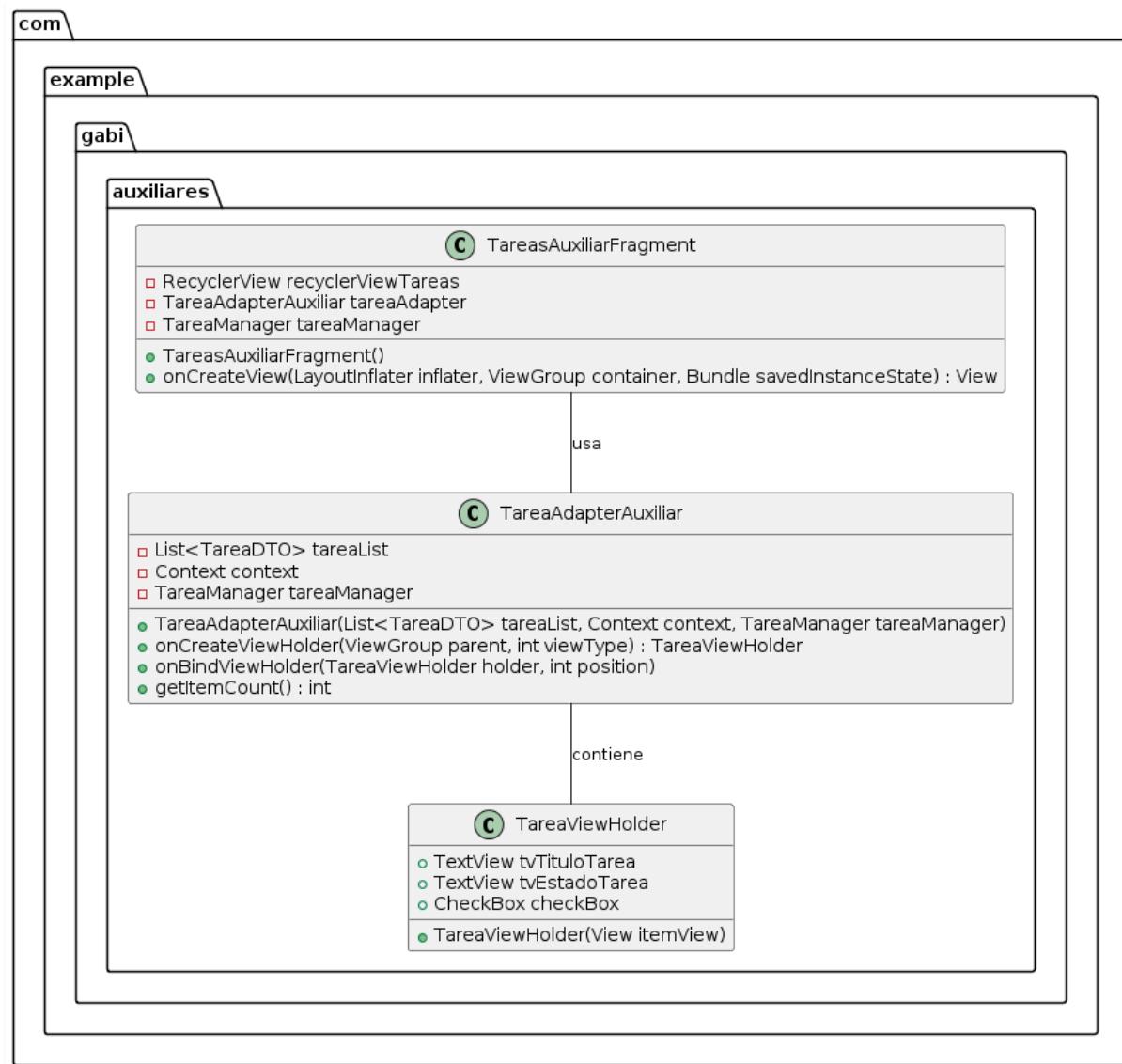
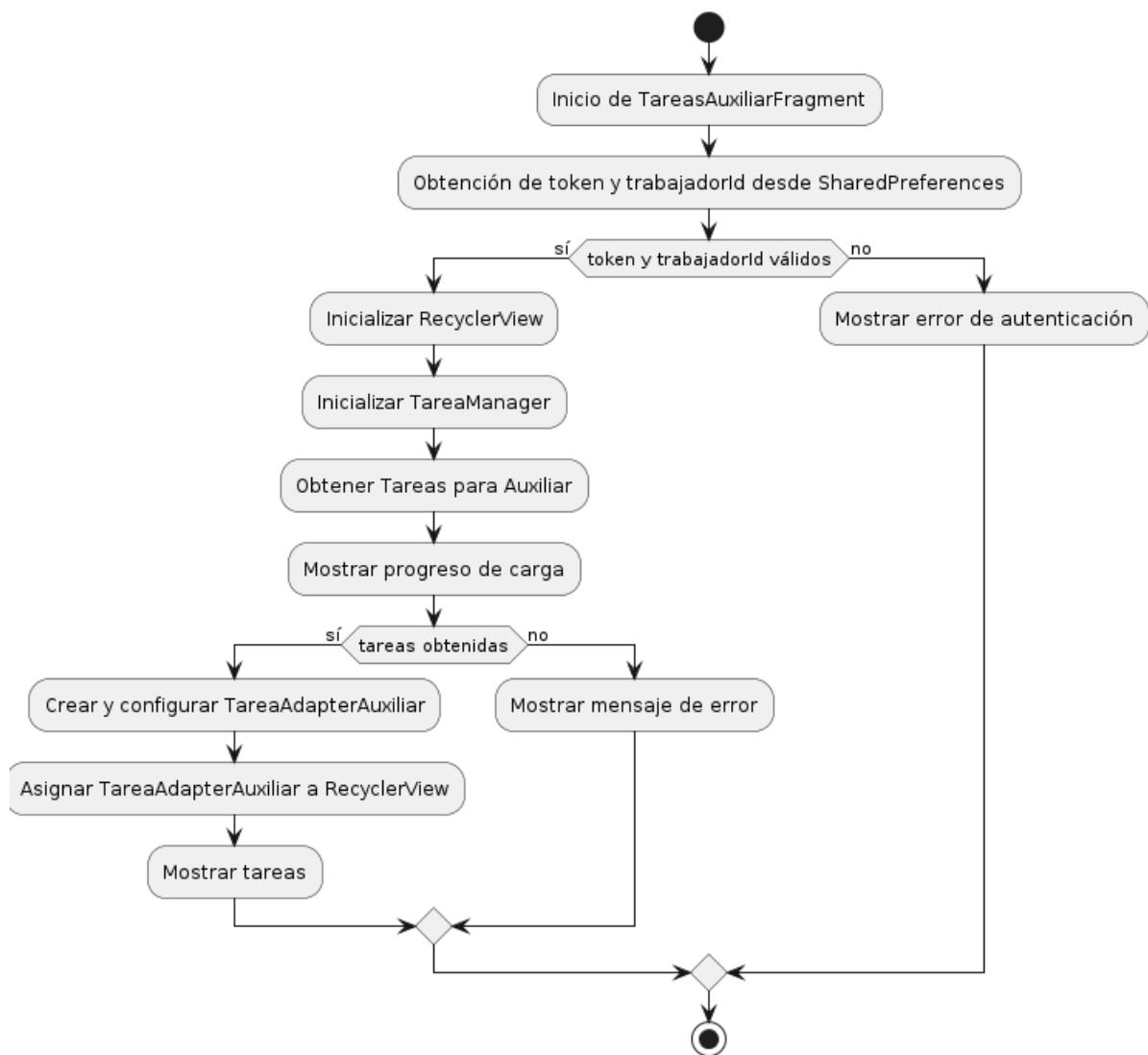


Diagrama de Flujo de TareasAuxiliarFragment

**Diagrama de Secuencia de TareasAuxiliarFragment**

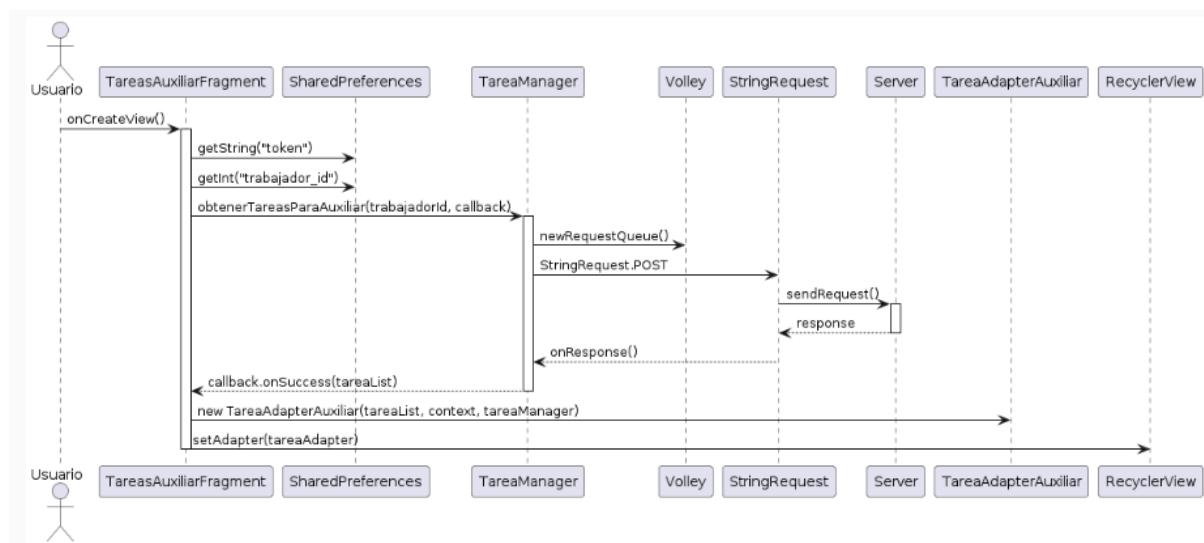
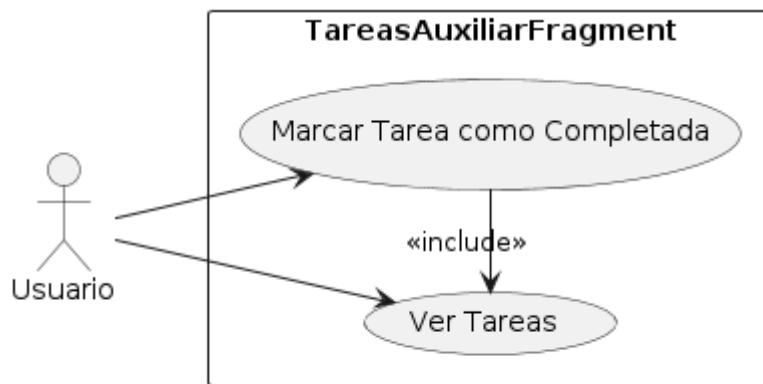


Diagrama de Casos de Uso de TareasAuxiliarFragment



InformacionAuxiliarFragment

InformacionAuxiliarFragment es una clase que extiende Fragment y representa la vista de la información detallada de los residentes en una residencia. Esta clase se encarga de mostrar una lista de residentes en un RecyclerView y permite al usuario filtrar la lista según el nombre, apellidos, email o DNI del residente. Además, permite enviar mensajes SMS de emergencia a los familiares de un residente manteniendo presionado el nombre del residente durante 10 segundos.

Métodos Principales:

- **onCreateView():** Este método configura la vista del fragmento, inicializa el RecyclerView, el EditText para la búsqueda y establece los TextWatcher para filtrar la lista de residentes. Además, llama al método obtenerResidentes() para cargar la lista de residentes desde el servidor.
- **obtenerResidentes():** Este método utiliza la biblioteca Volley para enviar una solicitud al servidor y obtener la lista de residentes. Los datos recibidos se



parsean y se almacenan en listaResidentes y listaResidentesFiltrada. Luego, se notifica al adaptador para actualizar la vista del RecyclerView.

- **filtrarResidentes():** Este método se llama cada vez que el texto del EditText cambia. Filtra la lista de residentes según el texto ingresado y actualiza listaResidentesFiltrada en consecuencia. Finalmente, notifica al adaptador para que actualice la vista.
- **mostrarDialogoConfirmacion():** Este método muestra un diálogo de confirmación para enviar un SMS de emergencia a los familiares del residente seleccionado. Se llama cuando el usuario mantiene presionado el nombre del residente durante 10 segundos.
- **enviarSMS():** Este método utiliza el SmsManager de Android para enviar un SMS al número de teléfono proporcionado.

Descripción de la Funcionalidad:

- **Visualización de Residentes:** La clase muestra una lista de residentes con detalles como nombre, apellidos, fecha de nacimiento, observaciones, teléfono, ID de habitación y una foto.
- **Filtrado de Residentes:** Los usuarios pueden buscar residentes por nombre, apellidos, email o DNI utilizando un campo de texto. La lista de residentes se filtra en tiempo real a medida que el usuario escribe.
- **Envío de Mensajes de Emergencia:** Si un usuario mantiene presionado el nombre de un residente durante 10 segundos, se muestra un diálogo de confirmación para enviar un SMS de emergencia a los familiares del residente. Si el usuario confirma, se envía el SMS a los números de teléfono de los familiares.

El ResidentesInfoCompletaAdapter es un adaptador que gestiona la visualización de cada residente en el RecyclerView, permitiendo al usuario interactuar con la lista de residentes y enviar mensajes de emergencia cuando sea necesario.

Diagrama de Clases de InformacionAuxiliarFragment

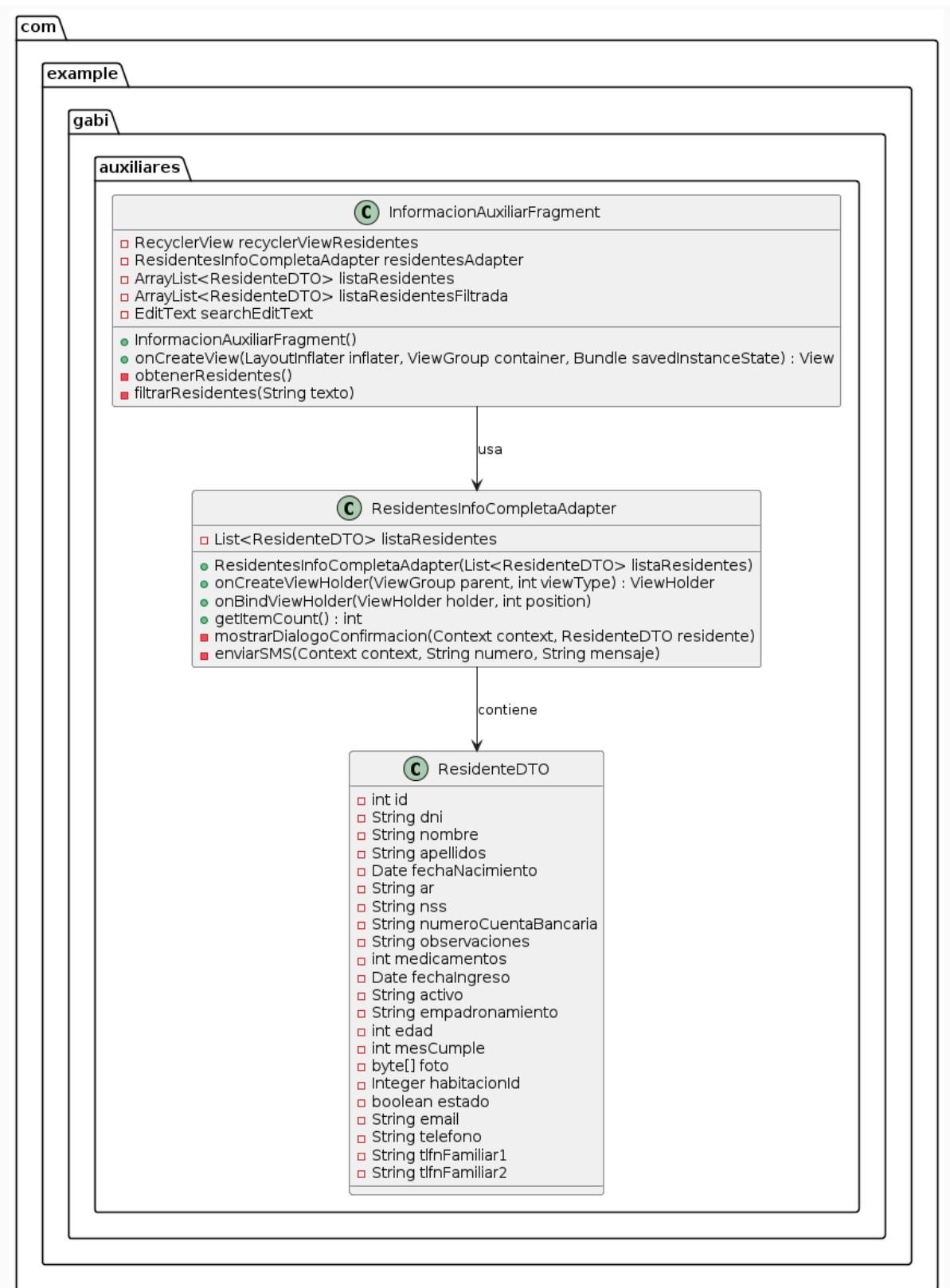


Diagrama de Flujo de InformacionAuxiliarFragment

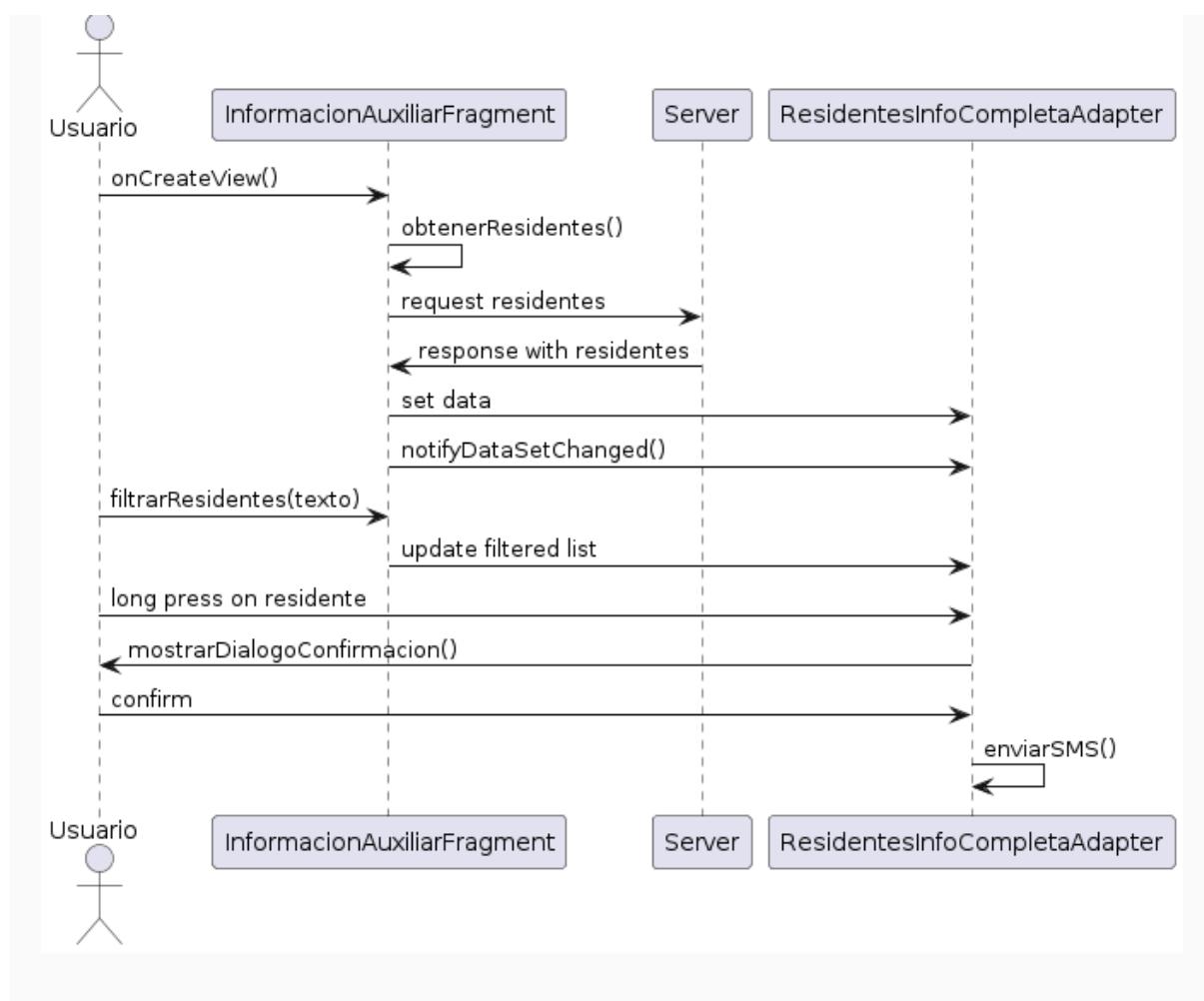


Diagrama de Secuencia de InformacionAuxiliarFragment

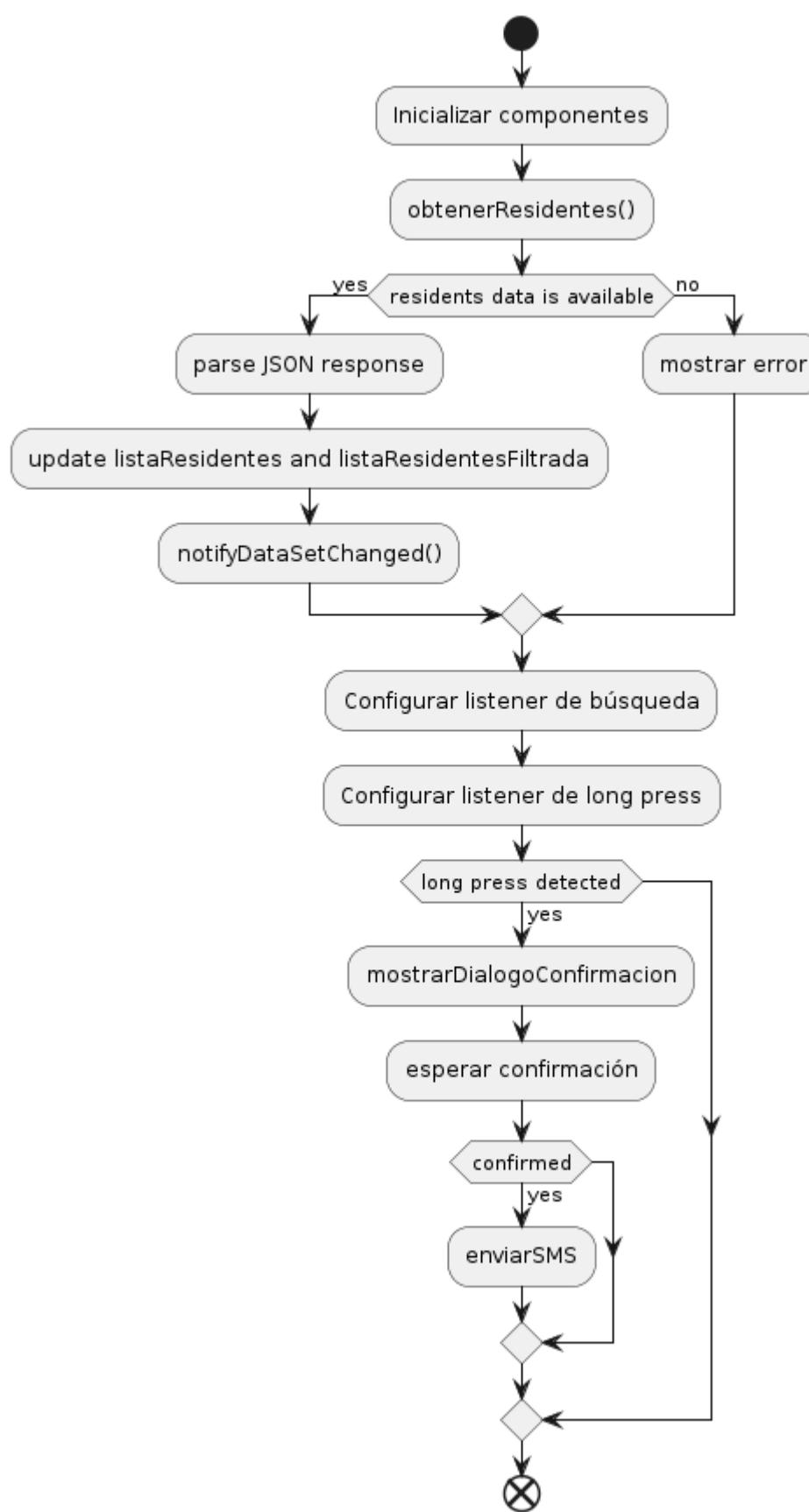
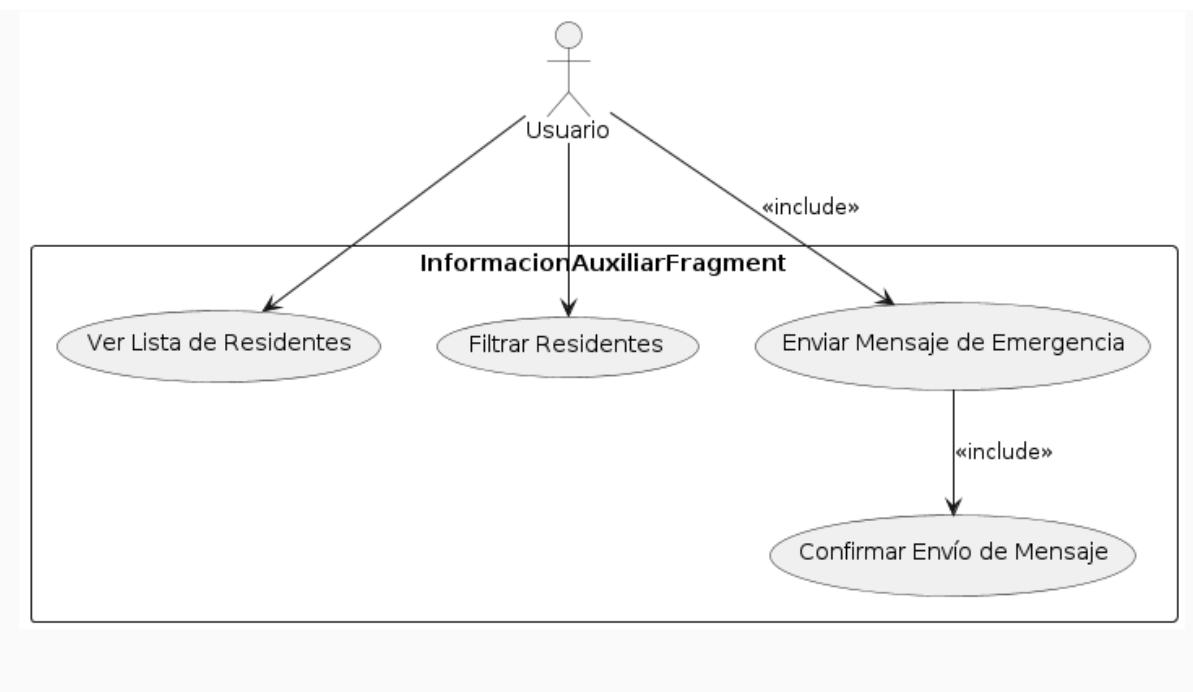


Diagrama de Casos de Uso de InformacionAuxiliarFragment



4.1.3.1.3. Flujo ChatActivity

ChatActivity es una actividad que permite a los usuarios enviar y recibir mensajes en tiempo real mediante Firebase. La actividad inicializa los componentes necesarios, configura la conexión con Firebase y gestiona la interfaz de usuario para el envío y la visualización de mensajes.

Métodos principales:

- **onCreate(Bundle savedInstanceState)**: Configura la vista de la actividad, inicializa los componentes de la interfaz de usuario y establece la conexión con Firebase.
- **buttonSend.setOnClickListener**: Envía un mensaje a Firebase cuando el usuario presiona el botón de enviar.
- **databaseReference.addChildEventListener**: Escucha los nuevos mensajes en Firebase y actualiza el RecyclerView a través del AdapterMensajes.

Interacciones con otras actividades:

- ChatActivity puede ser abierta desde AuxiliarActivity y AdministradorActivity cuando el usuario hace clic en el botón para abrir el chat.
- MainActivity maneja la navegación inicial después del inicio de sesión, redirigiendo al usuario a AdministradorActivity o AuxiliarActivity según su rol pasando la información del nombre de usuario que después es usada para



mostrarlo en los mensajes que este envia, desde ambos actibitys se accede al mismo chat.

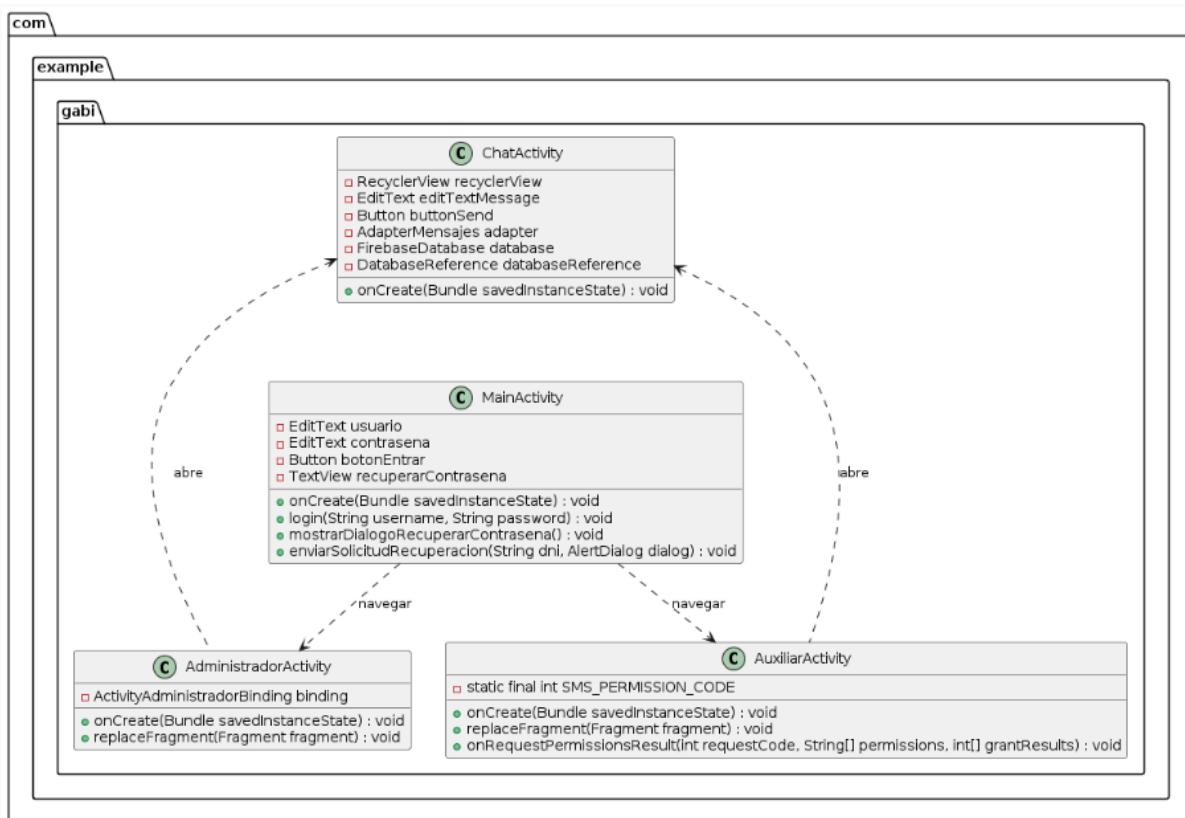
Interacciones con Firebase:

- FirebaseDatabase.getInstance(): Obtiene una instancia de Firebase.
- databaseReference.push().setValue(mensaje): Envía un mensaje a la base de datos Firebase.
- databaseReference.addChildEventListener(new ChildEventListener() {...}): Escucha los mensajes nuevos y actualiza la interfaz de usuario en tiempo real.

Componentes involucrados:

- RecyclerView: Muestra la lista de mensajes.
- AdapterMensajes: Adapta los mensajes para ser mostrados en el RecyclerView.
- HolderMensaje: Representa la vista de cada mensaje individual.
- MensajeEnviar y MensajeRecibir: Modelos de datos para los mensajes enviados y recibidos.

Diagrama de Clases de ChatActivity



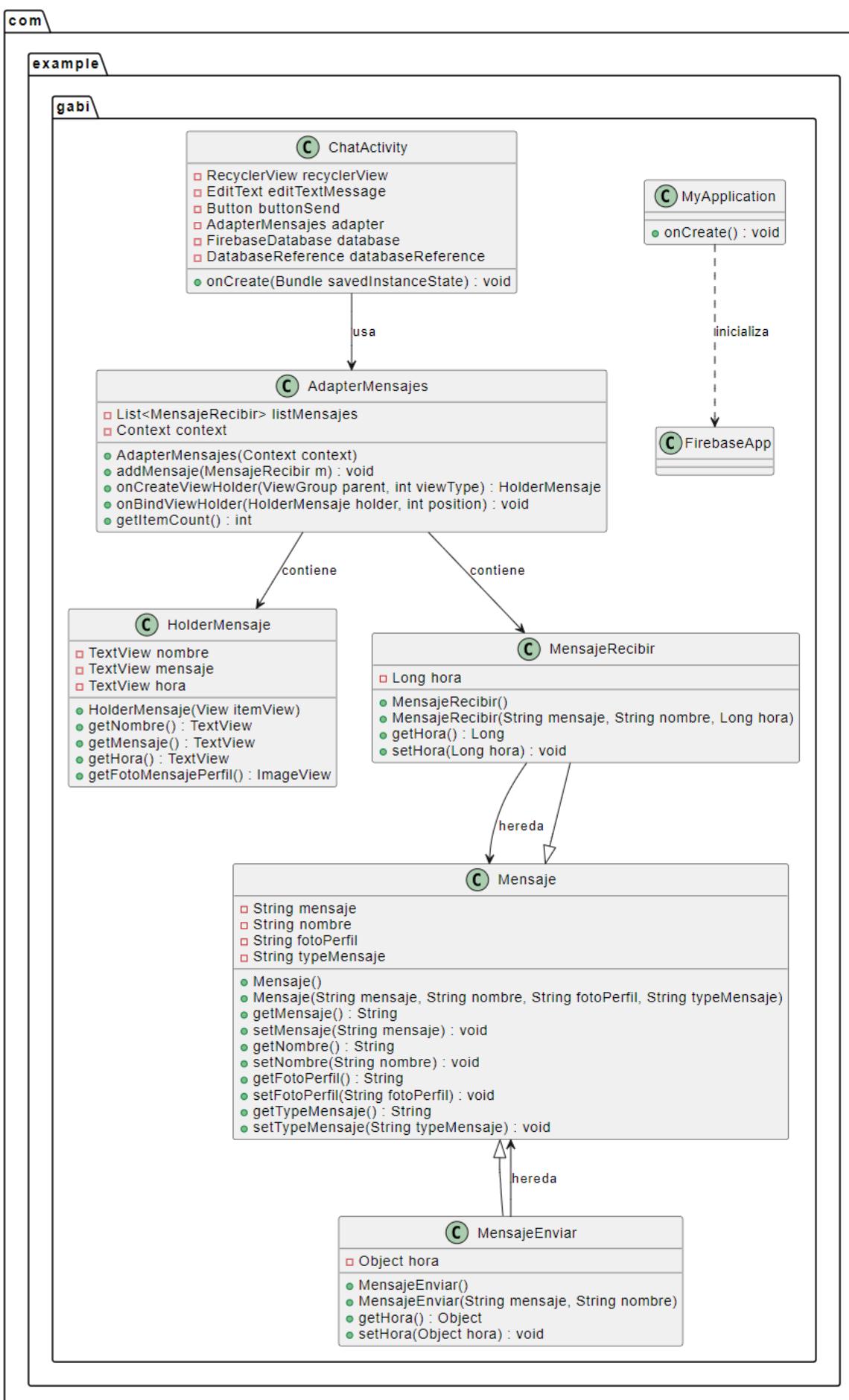




Diagrama de Flujo de ChatActivity

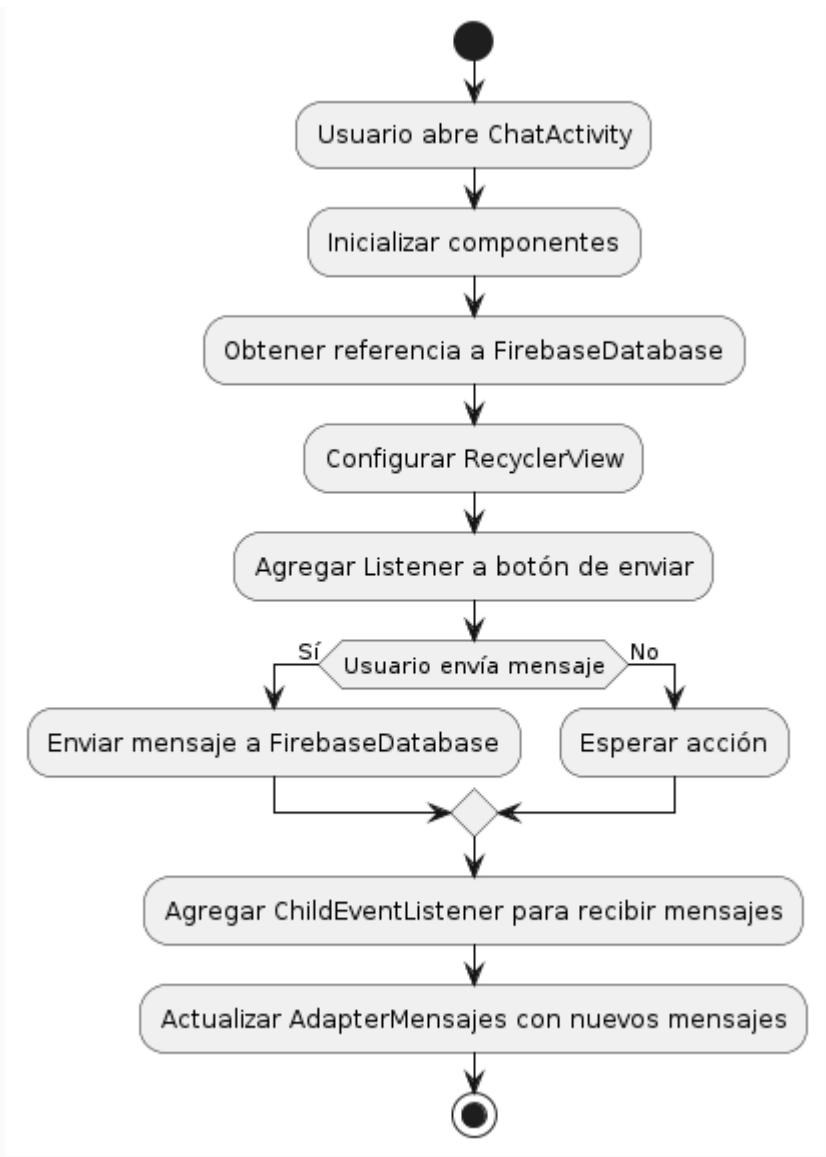
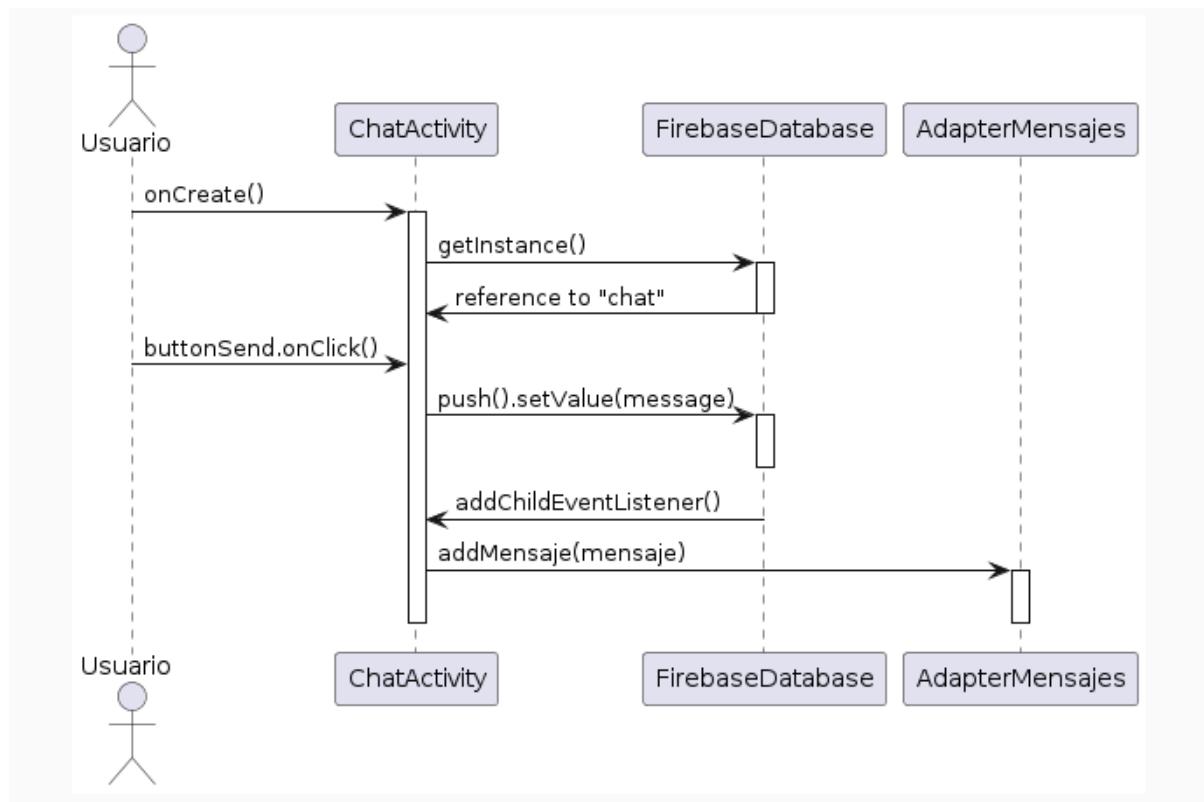


Diagrama de Secuencia de ChatActivity



4.1.3. Conexión y Comunicación con el Servidor

La aplicación se comunica con el servidor mediante APIs RESTful. Volley es una biblioteca de red en Android que facilita la comunicación con un servidor backend. A continuación, se muestra un ejemplo de cómo se realiza una solicitud GET para obtener la lista de residentes desde el servidor:



```
1 usage ± Elton Saravia
private void obtenerResidentes() {
    String url = "https://residencialontananza.com/aux/listarResidentesAuxiliares.php";

    SharedPreferences sharedPreferences = getActivity().getSharedPreferences("name: MyAppPrefs", getContext().MODE_PRIVATE);
    final String token = sharedPreferences.getString(key: "token", defaultValue: "");

    ± Elton Saravia
    StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
        response -> {
            try {
                JSONObject jsonResponse = new JSONObject(response);
                if (jsonResponse.getString(name: "status").equals("success")) {
                    JSONArray jsonArray = jsonResponse.getJSONArray(name: "data");
                    for (int i = 0; i < jsonArray.length(); i++) {
                        JSONObject jsonObject = jsonArray.getJSONObject(i);

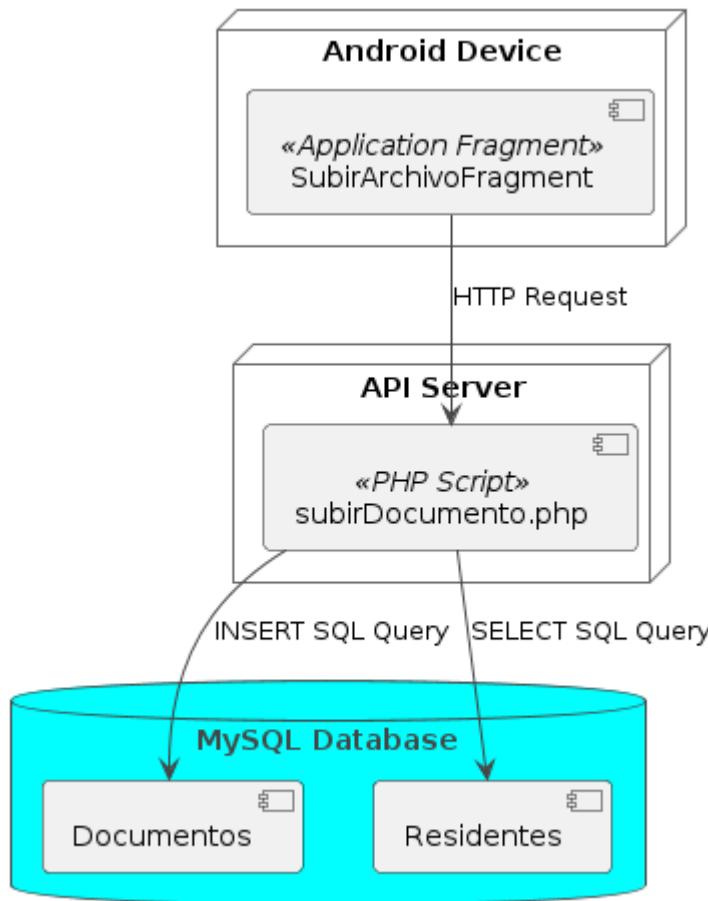
                        int id = jsonObject.getInt(name: "id");
                        String dni = jsonObject.getString(name: "dni");
                        String nombre = jsonObject.getString(name: "nombre");
                        String apellidos = jsonObject.getString(name: "apellidos");
                        String fechaNacimientoStr = jsonObject.getString(name: "fecha_nacimiento");
                        java.sql.Date fechaNacimiento = fechaNacimientoStr.equals("0000-00-00") ? null : java.sql.Date.valueOf(fechaNacimientoStr);
                        String ar = jsonObject.getString(name: "ar");
                        String nss = jsonObject.getString(name: "nss");
                        String numeroCuentaBancaria = jsonObject.getString(name: "numero_cuenta_bancaria");
                        String observaciones = jsonObject.getString(name: "observaciones");
                        int medicamentos = jsonObject.isNull(name: "medicamentos") ? 0 : jsonObject.getInt(name: "medicamentos");
                        String fechaIngresoStr = jsonObject.getString(name: "fecha_ingreso");
                        java.sql.Date fechaIngreso = fechaIngresoStr.equals("0000-00-00") ? null : java.sql.Date.valueOf(fechaIngresoStr);
                        String activo = jsonObject.getString(name: "activo");
                        String empadronamiento = jsonObject.getString(name: "empadronamiento");
                        int edad = jsonObject.isNull(name: "edad") ? 0 : jsonObject.getInt(name: "edad");
                        int mesCumple = jsonObject.isNull(name: "mes_cumple") ? 0 : jsonObject.getInt(name: "mes_cumple");
                        byte[] foto = null;
                        if (!jsonObject.isNull(name: "foto")) {
                            String fotoBase64 = jsonObject.getString(name: "foto");
                            foto = Base64.decode(fotoBase64, Base64.DEFAULT);
                        }
                        int habitacionId = jsonObject.isNull(name: "habitacion_id") ? 0 : jsonObject.getInt(name: "habitacion_id");
                        boolean estado = jsonObject.getInt(name: "estado") == 1;
                        String telefono = jsonObject.getString(name: "telefono");
                        String email = jsonObject.getString(name: "email");
                        String tlfnFamiliar1 = jsonObject.optString(name: "tlfn_familiar_1", fallback: null);
                        String tlfnFamiliar2 = jsonObject.optString(name: "tlfn_familiar_2", fallback: null);

                        ResidenteDTO residente = new ResidenteDTO(id, dni, nombre, apellidos, fechaNacimiento, ar, nss, numeroCuentaBancaria);
                        listaResidentes.add(residente);
                    }
                    listaResidentesFiltrada.addAll(listaResidentes);
                    residentesAdapter.notifyDataSetChanged();
                } else {
                    Toast.makeText(getContext(), text: "Error: " + jsonResponse.getString(name: "message"), Toast.LENGTH_SHORT).show();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    );
}
```

En el servidor, la API listarResidentesAuxiliares.php maneja la solicitud y devuelve los datos en formato JSON. La autenticación se realiza mediante un token JWT para garantizar la seguridad de las operaciones.



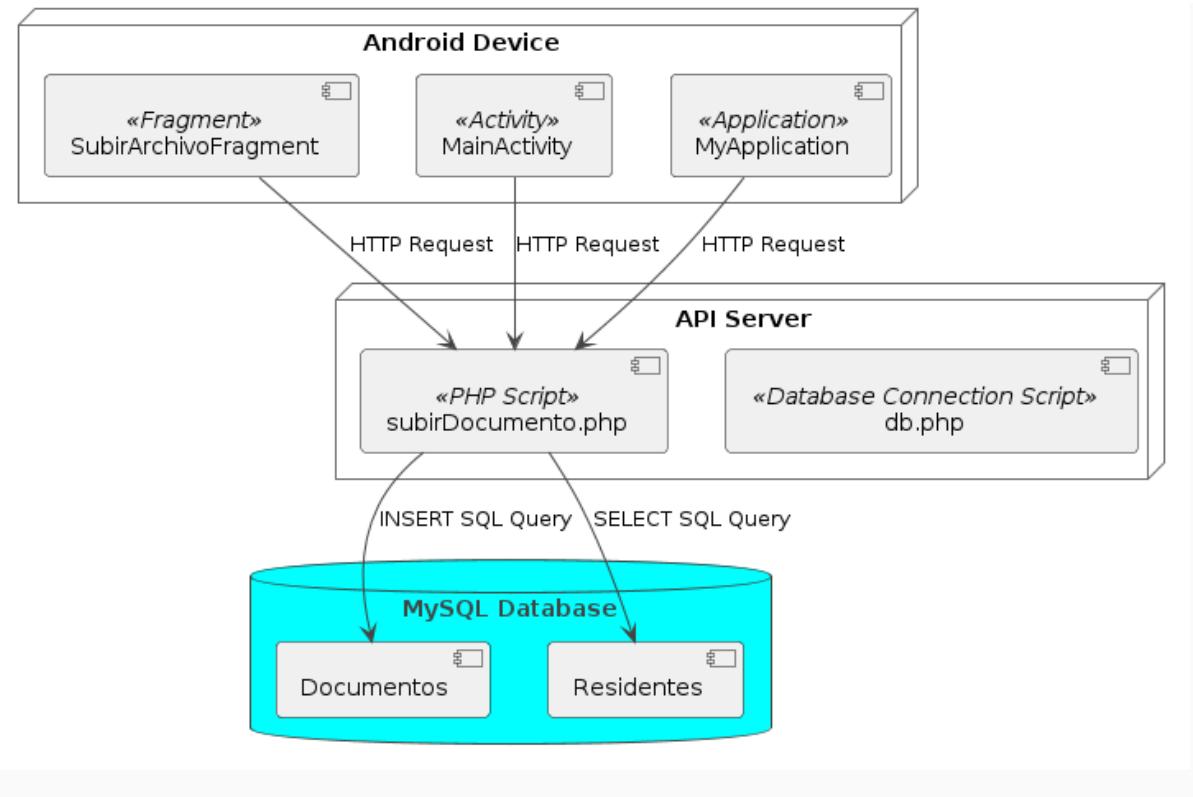
4.1.3.1 Ejemplo de conectividad MYSQL



En la arquitectura de la aplicación, el flujo de subir un documento se inicia desde el fragmento SubirArchivoFragment en la aplicación Android. El usuario selecciona un archivo que se convierte a Base64 y se envía junto con otros datos (DNI, título, descripción) a la API PHP en el servidor mediante una solicitud HTTP POST. La API PHP (subirDocumento.php) recibe esta solicitud y realiza varias tareas:

- Autenticación:** Verifica la autenticidad del token JWT incluido en los encabezados de la solicitud.
- Desencriptación:** Desencripta el DNI recibido para buscar al residente en la base de datos.
- Validación:** Comprueba si el DNI existe en la tabla Residentes.
- Almacenamiento:** Si se valida, el archivo y los metadatos (DNI, título, descripción, nombre del archivo, tipo de archivo, contenido, fecha de subida) se almacenan en la tabla Documentos en la base de datos MySQL.

Este diagrama y la descripción proporcionan una visión clara de cómo la aplicación maneja la subida de documentos, asegurando la autenticidad y la seguridad de los datos mediante el uso de JWT y encriptación, y almacenando los documentos de manera segura en la base de datos.



Explicación del Código

Código PHP

```
<?php

ini_set('display_errors', 1);

ini_set('display_startup_errors', 1);

error_reporting(E_ALL);

require __DIR__ . '/../vendor/autoload.php';

use \Firebase\JWT\JWT;

use \Firebase\JWT\Key;

include 'db.php';
```



```
header("Access-Control-Allow-Origin: *");

header("Content-Type: application/json; charset=UTF-8");

header("Access-Control-Allow-Headers: Authorization, Content-Type");

$logFile = __DIR__ . '/api.log';

$secretKey = "your_secret_key"; // Clave secreta para JWT

$encryptionKey = "your_32_character_encryption_key"; // Clave de cifrado de 32 caracteres

$authHeader = $_SERVER['HTTP_AUTHORIZATION'] ?? '';
```

Estas líneas configuran el entorno de errores, cargan bibliotecas necesarias y establecen encabezados HTTP, estas bibliotecas se instalaron en el servidor Hostinguer mediante unos pasos, alghunas activaciones eran mediante y interfaz y otras a través de conexión ssl

Validación del Token JWT:

```
if (!$authHeader) {

    http_response_code(401);

    file_put_contents($logFile, "Token no proporcionado.\n", FILE_APPEND);

    echo json_encode(array("message" => "Token no proporcionado."));

    exit;

}

$arr = explode(" ", $authHeader);

$jwt = $arr[1] ?? '';

if ($jwt) {

    try {

        $decoded = JWT::decode($jwt, new Key($secretKey,
'HS256'));

        $userId = $decoded->user_id;
```



```
    } catch (Exception $e) {

        http_response_code(401);

        file_put_contents($logFile, "Error al decodificar JWT:
" . $e->getMessage() . "\n", FILE_APPEND);

        echo json_encode(array(
            "message" => "Acceso denegado.",
            "error" => $e->getMessage()
        ));
    }

} else {

    http_response_code(401);

    file_put_contents($logFile, "Token no proporcionado.\n",
FILE_APPEND);

    echo json_encode(array("message" => "Acceso denegado."));
}
```

Este segmento valida el token JWT recibido en los encabezados de la solicitud. Si el token es válido, se decodifica para obtener el ID de usuario.

Funciones de Encriptación y Desencriptación:

```
function encrypt($string, $key) {
    $iv = openssl_random_pseudo_bytes(16);
    $encryptedData = openssl_encrypt($string, 'AES-256-CBC',
$key, 0, $iv);
    return base64_encode($encryptedData . '::::' . $iv);
}

function decrypt($string, $key) {
    list($encryptedData, $iv) = explode(':::', base64_decode($string), 2);
    return openssl_decrypt($encryptedData, 'AES-256-CBC', $key,
0, $iv);
}
```

Estas funciones manejan la encriptación y desencriptación de los datos del DNI.

**Procesamiento del Archivo:**

```
if      (isset($dni,      $titulo,      $descripcion,      $nombreArchivo,
$documento)) {
    // Obtener todos los DNIs de la base de datos
    $sql = "SELECT id, dni FROM residentes";
    $result = $conn->query($sql);

    if ($result === false) {
        file_put_contents($logFile, "Error en la consulta SQL:
" . $conn->error . "\n", FILE_APPEND);
        echo json_encode(array("status" => "error", "message"
=> "Error en la consulta: " . $conn->error));
        exit();
    }

    file_put_contents($logFile, "Consulta SQL ejecutada
correctamente\n", FILE_APPEND);

$hashedDNI = null;

while ($row = $result->fetch_assoc()) {
    $decryptedDNI = decrypt($row['dni'], $encryptionKey);

    if ($decryptedDNI === $dni) {
        file_put_contents($logFile, "Residente encontrado
con el DNI desencriptado: " . $decryptedDNI . "\n",
FILE_APPEND);
        $hashedDNI = $row['dni'];
        break;
    }
}

if (is_null($hashedDNI)) {
    echo json_encode(array("status" => "error", "message"
=> "DNI no encontrado en la base de datos."));
    exit();
}

// Procesar el archivo
$typeArchivo = "application/octet-stream"; // Ajustar según
el tipo de archivo real
$contenidoArchivo = base64_decode($documento);
$fechaSubida = date('Y-m-d H:i:s');
```



```
// Insertar el registro en la base de datos
    $sql = "INSERT INTO documentos (dni, titulo, descripcion,
nombre_archivo, tipo_archivo, contenido, fecha_subida) VALUES
(?, ?, ?, ?, ?, ?, ?)";
    $stmt = $conn->prepare($sql);

    if ($stmt === false) {
        echo json_encode(array("status" => "error", "message"
=> "Error en la preparación de la consulta: " .
$conn->error));
        exit();
    }

    $stmt->bind_param('sssssss', $hashedDNI, $titulo,
$descripcion, $nombreArchivo, $tipoArchivo, $contenidoArchivo,
$fechaSubida);

    if ($stmt->execute()) {
        echo json_encode(array("status" => "success", "message"
=> "Documento subido correctamente."));
    } else {
        echo json_encode(array("status" => "error", "message"
=> "Error al subir documento: " . $stmt->error));
    }

    $stmt->close();
    $conn->close();
} else {
    echo json_encode(array("status" => "error", "message" =>
"Faltan datos requeridos o archivo no proporcionado."));
}
```

Este código verifica la existencia del DNI en la base de datos, descifra el DNI, y guarda el documento en la base de datos si todo está correcto. Es importante entender este apartado porque prácticamente todas las peticiones siempre aparte de validar el dni

Android

Clase SubirArchivoFragmet:

```
private void uploadFile() {
    final String dni = etDNI.getText().toString().trim();
    final String titulo = etTitulo.getText().toString().trim();
    final String descripcion = etDescripcion.getText().toString().trim();
```



```
Log.d(TAG, "DNI: " + dni);
Log.d(TAG, "Titulo: " + titulo);
Log.d(TAG, "Descripción: " + descripcion);
Log.d(TAG, "fileUri: " + fileUri);
Log.d(TAG, "fileName: " + fileName);

    if (dni.isEmpty() || titulo.isEmpty() || descripcion.isEmpty() || fileUri == null || fileName == null) {
        Toast.makeText(getApplicationContext(), "Faltan datos obligatorios o archivo no seleccionado.", Toast.LENGTH_SHORT).show();
        return;
    }

    StringRequest request = new StringRequest(Request.Method.POST,
"https://residencialontanza.com/api/subirDocumento.php",
    response -> {
        try {
            JSONObject jsonResponse = new JSONObject(response);
            String status = jsonResponse.getString("status");
            String message = jsonResponse.getString("message");

            if (status.equals("success")) {
                Toast.makeText(getApplicationContext(), "Documento subido correctamente.", Toast.LENGTH_LONG).show();
            } else {
                Toast.makeText(getApplicationContext(), message,
Toast.LENGTH_LONG).show();
            }
        } catch (JSONException e) {
            e.printStackTrace();
            Toast.makeText(getApplicationContext(), "Error al procesar la respuesta del servidor.", Toast.LENGTH_LONG).show();
        }
    },
    error -> Toast.makeText(getApplicationContext(), "Error al subir documento: " +
error.getMessage(), Toast.LENGTH_LONG).show()
) {
    @Override
    protected Map<String, String> getParams() {
        Map<String, String> params = new HashMap<>();
        params.put("dni", dni);
        params.put("titulo", titulo);
        params.put("descripcion", descripcion);
        params.put("nombreArchivo", fileName);

        try {
            InputStream inputStream = getActivity().getContentResolver().openInputStream(fileUri);
            ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
            byte[] buffer = new byte[1024];
            int len;
            while ((len = inputStream.read(buffer)) != -1) {
                byteArrayOutputStream.write(buffer, 0, len);
            }
            inputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        String encodedFile =
Base64.encodeToString(byteArrayOutputStream.toByteArray(), Base64.DEFAULT);
    }
}
```



```
        params.put("documento", encodedFile);
                Log.d(TAG, "Archivo codificado en base64: " +
encodedFile.substring(0, 30) + "...");
    } catch (IOException e) {
        e.printStackTrace();
        Log.e(TAG, "Error al leer el archivo: " + e.getMessage());
    }

    return params;
}
```

En esta clase:

- **onCreateView:** Se inicializan los componentes de la interfaz y se configura el lanzador de actividad para seleccionar archivos.
- **openFileChooser:** Abre el selector de archivos del dispositivo.
- **getFileName:** Obtiene el nombre del archivo seleccionado.
- **uploadFile:** Envía los datos del archivo y la información adicional al servidor PHP.

Este método obtiene los parámetros que se enviarán en la solicitud POST. Primero, se agregan el DNI, título, descripción y nombre del archivo al mapa params. Luego, se lee el archivo seleccionado, se codifica en Base64 y se agrega al mapa de parámetros bajo la clave "documento".

```
@Override
public Map<String, String> getHeaders() throws AuthFailureError {
    SharedPreferences sharedpreferences =
getActivity().getSharedPreferences("MyAppPrefs", Context.MODE_PRIVATE);
    String token = sharedpreferences.getString("token", "");
    Map<String, String> headers = new HashMap<>();
    headers.put("Authorization", "Bearer " + token);
    return headers;
}
};
```

Este método agrega los encabezados de la solicitud, específicamente el token de autorización JWT, que se obtiene de las preferencias compartidas.

```
RequestQueue queue = Volley.newRequestQueue(getContext());
queue.add(request);
```

Finalmente, se crea una cola de solicitudes RequestQueue y se agrega la solicitud configurada a la cola para su ejecución.

Conclusión



1. El método uploadFile recopila los datos ingresados por el usuario y verifica que todos los campos necesarios estén presentes.
2. Configura una solicitud HTTP POST para enviar estos datos al servidor.
3. Incluye el archivo seleccionado, codificado en Base64, junto con otros datos en los parámetros de la solicitud.
4. Añade un encabezado de autorización con el token JWT.
5. Envía la solicitud al servidor y maneja la respuesta o cualquier error que ocurra durante el proceso.

Se explica el flujo completo y la interacción entre los componentes de la aplicación, desde la selección del archivo en Android hasta su almacenamiento en la base de datos mediante PHP. Cada sección del código ha sido descrita para proporcionar una comprensión de cómo funciona la aplicación en su totalidad.

Para entender mejor la singularidad de cada apartado se deberá revisar el código de la API al mismo tiempo que el código de la aplicación.

5. CIERRE

5.1. Resultados obtenidos y conclusiones

El proyecto comenzó a principios de enero creando la idea y pensando en los casos de usos. En abril de 2024 se comienza a dedicarle el tiempo una vez aprobada la propuesta y se crea el servidor, marcando el inicio de un ambicioso desarrollo que se ha centrado en construir un sistema robusto y funcional para la gestión de empleados, residentes, y diversas operaciones relacionadas. Antes del desarrollo se invirtieron muchas horas investigando el funcionamiento de las residencias hablando con los empleados y directores antes de empezar para entender sus necesidades e intentar desarrollar la herramienta abarcando cada una de ellas. Desde el primer commit inicial, se estableció una base sólida con la creación de la conexión a la base de datos, que fue fundamental para el desarrollo posterior de las funcionalidades del sistema.

A lo largo de mayo, el enfoque se desplazó hacia el diseño y la implementación de funcionalidades críticas. Se desarrollaron módulos esenciales como el registro de empleados y residentes, las funcionalidades de inicio y cierre de sesión, y la gestión de cuentas. Cada módulo se implementó, comenzando con la creación de la API y verificando su conectividad con Postman haciendo pruebas a los endpoints y



verificando las respuestas de las conexiones con la base de datos, después el diseño de la interfaz de usuario y siguiendo con la implementación de la lógica de backend y las validaciones necesarias. Este enfoque iterativo aseguró que cada componente fuera probado y validado antes de pasar al siguiente, garantizando así la estabilidad y la coherencia del sistema.

El proceso de desarrollo no estuvo exento de desafíos. Se encontraron y solucionan diversos problemas, desde errores en la asignación de fechas hasta dificultades en la modificación de datos de residentes, al tener datos encriptados o cifrados y no tener una conectividad en entorno local será difícil detectar los fallos cuando el resultado no era el esperado. Sin embargo, cada problema se abordó con una combinación de correcciones rápidas y mejoras a largo plazo, lo que permitió mantener el proyecto en curso sin comprometer su calidad.

Se detectó rápidamente que cada módulo era un desafío mayor al del tiempo estimado por lo cual se decidió crear el rol de administrador y el rol de auxiliar y preparar la incorporación del rol enfermero para el futuro.

Durante todo el proceso, se puso un fuerte énfasis en la mejora continua y en la optimización del sistema. Las interfaces de usuario fueron refinadas para ser más intuitivas y fáciles de usar tras conversaciones con expertas en este ámbito y aún quedan ideas por implementar pero que un futuro harán que la aplicación mejoré su usabilidad, se añadieron funcionalidades avanzadas como la recuperación de contraseñas y un chat en tiempo real. Estas mejoras no solo aumentaron la funcionalidad del sistema, sino que también mejoraron significativamente la experiencia del usuario final.

El uso de herramientas modernas como GitHub para la gestión del código fuente y Firebase para el manejo de datos en tiempo real facilitó enormemente el desarrollo y la integración de nuevas características. La garantía de tener GitHub como fuente de la verdad sirvió de gran manera para introducir cambios y experimentar con diferentes librerías ayudando a que cuando cambiaba alguna funcionalidad crítica por error pudiese hacer un rollback de la lógica de negocio en Java. Firebase era sencillo de implementar en comparación con lo anterior a pesar de tener unos problemas que detuvieron el avance por un error al detectar la región de la app.

A medida que el proyecto avanzaba, se implementaron estrictas medidas de seguridad, incluyendo el uso de JWT para autenticación y mecanismos de cifrado de datos, garantizando así la protección de la información sensible. Estas medidas, aseguraron que el sistema cumpliera con los estándares de seguridad y calidad esperados.

En conclusión, el proyecto ha alcanzado un estado avanzado de desarrollo, con la mayoría de sus funcionalidades clave completadas y operativas. A través de una gestión eficaz de tareas, un enfoque iterativo en el desarrollo y un compromiso



constante con la mejora continua, se ha creado un sistema robusto y eficiente. Este proyecto está bien posicionado para futuras expansiones y mejoras, habiendo establecido una base sólida sobre la cual construir. La dedicación y el esfuerzo invertidos en este proyecto reflejan un trabajo bien hecho y una visión clara para el futuro desarrollo del sistema.

5.2. Diario de bitácora

| JIRA | | | |
|-------------|---------|--|------------------------|
| Fecha | Horas | Descripción | Tarea Asignada |
| 12 de abril | 4 horas | Investigación sobre seguridad en bases de datos y mejores prácticas, enfocándose en la protección de datos sensibles. | KAN-43, KAN-44, KAN-45 |
| 13 de abril | 5 horas | Ánalisis de tecnologías disponibles para desarrollo de aplicaciones móviles, incluyendo comparativas entre ellas. | KAN-53, KAN-55 |
| 14 de abril | 6 horas | Investigación sobre cómo implementar autenticación segura en PHP y técnicas de cifrado. | KAN-43, KAN-45 |
| 15 de abril | 5 horas | Investigación sobre cómo conectar Android Studio con bases de datos MySQL, incluyendo tutoriales y ejemplos prácticos. | KAN-57, KAN-59 |
| 16 de abril | 4 horas | Ánalisis de frameworks y bibliotecas PHP para desarrollo rápido de APIs RESTFUL, evaluando sus pros y contras. | KAN-57, KAN-61 |
| 17 de abril | 5 horas | Investigación sobre mejores prácticas para diseño de interfaces en Android Studio, incluyendo guías de estilo y UX. | KAN-50, KAN-51 |
| 18 de abril | 6 horas | Estudio de Firebase y sus capacidades para manejar datos en tiempo real, incluyendo configuración inicial. | KAN-34, KAN-36 |
| 19 de abril | 5 horas | Ánalisis comparativo de diferentes servicios de backend para aplicaciones móviles, evaluando costo y rendimiento. | KAN-53, KAN-55 |
| 20 de abril | 4 horas | Investigación sobre uso de JWT para autenticación en APIs PHP, incluyendo ejemplos de implementación. | KAN-43, KAN-45 |
| 21 de abril | 5 horas | Estudio de técnicas de cifrado y encriptado para proteger datos sensibles, con enfoque en AES y RSA. | KAN-43, KAN-44 |
| 22 de abril | 6 horas | Investigación sobre patrones de diseño en el desarrollo de aplicaciones móviles, | KAN-50, KAN-51 |
| 23 de abril | 5 horas | Ánalisis de herramientas de depuración y pruebas en Android Studio, probando su efectividad y facilidad de uso. | KAN-50, KAN-52 |
| 24 de abril | 4 horas | Estudio de cómo implementar notificaciones push en Android con Firebase, incluyendo la configuración del servicio. | KAN-34, KAN-36 |
| 25 de abril | 5 horas | Investigación sobre el uso de bases de datos relacionales vs | KAN-57, KAN-59 |



| | | | |
|-------------|---------|--|-----------------------|
| abril | horas | no relacionales en aplicaciones móviles, evaluando casos de uso. | |
| 26 de abril | 6 horas | Análisis de diferentes estrategias de almacenamiento en caché en aplicaciones móviles, probando varias opciones. | KAN-53, KAN-55 |
| 27 de abril | 5 horas | Investigación sobre cómo optimizar consultas SQL en MySQL para mejorar el rendimiento de la base de datos. | KAN-53, KAN-69 |
| 28 de abril | 4 horas | Initial commit del proyecto, incluyendo investigación sobre el funcionamiento de Git y configuración inicial del repositorio. | KAN-1, KAN-11, KAN-57 |
| 29 de abril | 5 horas | Planificación del proyecto y configuración inicial de herramientas y entorno de desarrollo. | KAN-11, KAN-57 |
| 30 de abril | 6 horas | Investigación y configuración de herramientas (Android Studio, MySQL, PHP), además de configuración inicial de bases de datos. | KAN-1, KAN-11, KAN-57 |
| 1 de mayo | 5 horas | Creación de la base de datos inicial en MySQL, definiendo tablas y relaciones necesarias. | KAN-57, KAN-59 |
| 2 de mayo | 4 horas | Diseño de la arquitectura de la aplicación, incluyendo diagramas y planificación de la estructura del código. | KAN-50, KAN-61 |
| 3 de mayo | 5 horas | Creación del archivo de conexión a la base de datos, probando su funcionalidad. | KAN-57, KAN-59 |
| 4 de mayo | 3 horas | Creación de esquemas iniciales de la base de datos, incluyendo índices y restricciones. | KAN-57, KAN-58 |
| 5 de mayo | 6 horas | Implementación de las tablas principales en MySQL, asegurando la integridad referencial. | KAN-57, KAN-59 |
| 6 de mayo | 4 horas | Configuración del entorno de desarrollo en Android Studio, incluyendo emuladores y dispositivos de prueba. | KAN-50, KAN-51 |
| 7 de mayo | 5 horas | Desarrollo del primer prototipo de la interfaz de usuario, creando layouts y actividades básicas. | KAN-2, KAN-28 |
| 8 de mayo | 6 horas | Implementación de la lógica básica de la aplicación en Android, conectando con la base de datos. | KAN-13, KAN-65 |
| 9 de mayo | 6 horas | Añadido uso de internet en el manifest y recursos de string para el spinner de puestos de trabajo. | KAN-63, KAN-64 |
| 10 de mayo | 5 horas | Creación de layouts adicionales y ajustes en la interfaz de usuario para mejorar la usabilidad. | KAN-2, KAN-28 |
| 11 de mayo | 4 horas | Creación del layout para añadir usuarios y lógica para conectar con la API, probando la integración. | KAN-12, KAN-19 |
| 12 de mayo | 6 horas | Creación de DTOs y adaptadores para listas en el fragmento Home de administrador, mejorando la organización del código. | KAN-34, KAN-36 |
| 13 de mayo | 5 horas | Desarrollo de la lógica de login y verificación de usuarios y roles, implementando medidas de seguridad. | KAN-13, KAN-45 |
| 14 de mayo | 7 horas | Implementación del calendario de eventos en el fragmento Home del administrador, incluyendo visualización y navegación. | KAN-34, KAN-36 |



| | | | |
|------------|----------|--|----------------|
| 15 de mayo | 4 horas | Mejoras en la interfaz del calendario y solución de bugs, asegurando una experiencia de usuario fluida. | KAN-34, KAN-36 |
| 16 de mayo | 5 horas | Uso de tokens JWT en las peticiones de API, asegurando la autenticación y seguridad de las comunicaciones. | KAN-43, KAN-45 |
| 17 de mayo | 3 horas | Corrección de errores en el formulario de inserción de usuarios, mejorando la validación de datos. | KAN-19, KAN-63 |
| 18 de mayo | 5 horas | Añadido botón para cerrar sesión y eliminar el token, implementando la lógica de seguridad necesaria. | KAN-16, KAN-17 |
| 19 de mayo | 6 horas | Modificación de la lógica para insertar tareas asociadas a un trabajador (auxiliar), asegurando su correcta asignación. | KAN-31, KAN-8 |
| 20 de mayo | 5 horas | Creación de nuevos fragmentos y adaptadores para manejar los turnos de auxiliares, mejorando la gestión de horarios. | KAN-26, KAN-29 |
| 21 de mayo | 5 horas | Creación de la API para crear residentes, interfaz y lógica para llamar la API, asegurando su correcta integración. | KAN-7, KAN-9 |
| 22 de mayo | 4 horas | Ajustes en la interfaz para interactuar con los fragmentos de documentos, mejorando la experiencia del usuario. | KAN-32, KAN-58 |
| 23 de mayo | 4 horas | Solución del problema de desencriptado del DNI de los trabajadores, asegurando la integridad de los datos. | KAN-1, KAN-4 |
| 24 de mayo | 5 horas | Mejoras en la visualización de listas de residentes, optimizando el rendimiento y la usabilidad. | KAN-1, KAN-23 |
| 25 de mayo | 6 horas | Mostrar imagen de cada residente en la lista de residentes, mejorando la identificación visual. | KAN-1, KAN-24 |
| 26 de mayo | 11 horas | Modificación de la interfaz para interactuar en el mismo fragmento de documentos y mejoras en la lógica. | KAN-24, KAN-58 |
| 27 de mayo | 5 horas | Configuración inicial de Firebase y ajustes en la lógica del chat, probando la integración en tiempo real. | KAN-34, KAN-35 |
| 28 de mayo | 5 horas | Agregado activity y fragmentos para empezar con los casos de uso de un auxiliar, mejorando la funcionalidad general. | KAN-28, KAN-29 |
| 29 de mayo | 4 horas | Pruebas y corrección de errores en la funcionalidad del chat en tiempo real, asegurando su correcta operación. | KAN-34, KAN-35 |
| 30 de mayo | 6 horas | Implementación de funcionalidades adicionales en la interfaz de usuario, mejorando la usabilidad y la experiencia. | KAN-34, KAN-35 |
| 31 de mayo | 5 horas | Ajustes finales y preparación para la presentación del primer mes de progreso, asegurando que todos los componentes funcionen correctamente. | KAN-39, KAN-42 |
| 1 de junio | 5 horas | Implementación de la funcionalidad de mirar turnos del mes, creada la API y mejorada la interfaz. | KAN-26, KAN-31 |
| 2 de junio | 2 horas | Mejoras en la interfaz de varios apartados, añadiendo textos descriptivos e iconos, eliminando fondo provisional. | KAN-50, KAN-72 |
| 3 de junio | 4 horas | Creación de la funcionalidad para recuperar contraseña, se crea la API y funciona enviando el mail al correo del DNI proporcionado. | KAN-50, KAN-23 |



| | | | |
|------------|-------------|--|----------------|
| 4 de junio | de 2 horas | Incorporada funcionalidad para marcar tarea terminada, se crea la API y se prueba en Postman, falta cambiar interfaz cuando tarea está terminada. Implementación de funcionalidades adicionales en la interfaz de usuario, mejorando la usabilidad y la experiencia. | KAN-53, KAN-55 |
| 5 de junio | de 2 horas | Corrección de error de asignación de fecha errónea en el apartado de asignar tarea, asegurando la precisión de las asignaciones. | KAN-53, KAN-55 |
| 6 de junio | de 2 horas | Funcionalidad de marcar tarea como hecha, implementada y probada, asegurando el correcto registro de las tareas completadas. | KAN-53, KAN-55 |
| 7 de junio | de 6 horas | Cambios en la interfaz de búsqueda, mejorando la funcionalidad y la experiencia del usuario. | KAN-50, KAN-71 |
| 8 de junio | de 12 horas | Creación del apartado de pagos en la documentación, se crea la API y también la tabla y el trigger correspondientes. | KAN-50, KAN-72 |
| 9 de junio | de 9 horas | Mejorar la documentación y crear digramas. | KAN-39 |

Commits GitHub Android studio

| Fecha | Commit | Hora | Descripción |
|-------------|--------|------|-------------|
| 12 de abril | - | | |
| 13 de abril | - | | |
| 14 de abril | - | | |
| 15 de abril | - | | |
| 16 de abril | - | | |
| 17 de abril | - | | |
| 18 de abril | - | | |
| 19 de abril | - | | |
| 20 de abril | - | | |
| 21 de abril | - | | |



| | | | |
|-------------|---------|-----------------------|--|
| 22 de abril | - | | |
| 23 de abril | - | | |
| 24 de abril | - | | |
| 25 de abril | - | | |
| 26 de abril | - | | |
| 27 de abril | - | | |
| 28 de abril | d9ff914 | 19:03: 33 +0200 | probando segundo commit |
| | 5213c69 | 18:51: 35 +0200 | Initial commit |
| 29 de abril | - | | |
| 30 de abril | 61a28c7 | 19:02: 50 +0200 | Creado el archivo de conexión a base de datos |
| 1 de mayo | - | | |
| 2 de mayo | d02d096 | 23:20: 29 +0200 | Se añade uso de internet en manifest, se crea layout para añadir usuarios y se crea la lógica para conectar con la API. Se añaden recursos de string para el spinner de puestos de trabajo |
| 3 de mayo | 49ef58d | 23:54: 43 +0200 | Se crean nuevas clases Java para manejar el CRUD con las APIs PHP relacionadas a trabajadores, queda funcionando insertar |
| 4 de mayo | - | | |
| 5 de mayo | - | | |
| 6 de mayo | - | | |
| 7 de mayo | - | | |
| 8 de mayo | - | | |
| 9 de mayo | - | | |



| | | | |
|------------|---------|-----------------------|--|
| 10 de mayo | - | | |
| 11 de mayo | 2c80968 | 18:18: 31 +0200 | Nuevos fragmentos creados, error en el activity del administrador pdt de investigar |
| | 8e45d80 | 13:34: 22 +0200 | Creada las clases de la tabla trabajador y su respectivo DTO |
| 12 de mayo | c5f5654 | 13:45: 12 +0200 | Funcionando el login verificando usuario y rol desde la base de datos |
| | bf646ce | 22:46: 45 +0200 | Todos los DTO creados pero sin métodos personalizados, se crean adaptadores para las listas del fragmento Home de administrador, pdt: terminar configuración de métodos obtenerEventosDelDia() y obtenerTurnosDelDia() |
| 13 de mayo | - | | |
| 14 de mayo | 242888f | 19:12: 39 +0200 | Ya compila el código tras insertar las 3 listViews en el fragmento Home de administrador |
| | d98f71f | 21:24: 35 +0200 | Ya muestra información en la lista de empleados en el turno actual, quedan pendientes las otras 2, debo asegurar los tipos de datos Date de Java en comparación con los de MYSQL |
| | bc9ac71 | 22:20: 49 +0200 | Mejorada interfaz de las listas de Home administrador |
| | aa51788 | 23:48: 26 +0200 | Calendario de eventos en el fragmento Home de administrador funciona |
| | b7df60a | 23:53: 38 +0200 | Calendario aplicada espacio para descripción al presionar encima |
| 15 de mayo | e65065c | 22:25: 35 +0200 | Se muestra el nombre del residente en el calendario de eventos |
| 16 de mayo | f6b8677 | 01:01: 59 +0200 | Se establecido el uso de JWT, para ello he tenido que hacer modificaciones en las APIs PHP. Se ha agregado el uso solo a la petición de obtenerEventosDelDia, debo agregarlo a todas las peticiones |
| | 905c314 | 20:34: 07 +0200 | Se agrega uso de token a obtener turnos de empleados, ya muestra la información tras la validación del token |
| | b838e80 | 21:04: 29 +0200 | Mejora visual del apartado que muestra a los empleados que tienen turnos en el día actual |



| | | | |
|------------|---------|-------------------|---|
| 17 de mayo | 61ea846 | 13:11:35 +0200 | Corregidos errores en formulario de insertar usuarios |
| | fc71b7f | 13:11:28 +0200 | Corregidos errores en formulario de insertar usuarios |
| | f3726bb | 22:50:55 +0200 | Funciona listar trabajadores en el apartado empleados de administrador |
| | 91cf87d | 22:55:08 +0200 | Se agrega buscador para la lista de empleados |
| 18 de mayo | 1e4da25 | 15:35:18 +0200 | Se crea interfaz para actualizar datos de usuario haciendo 2 llamadas a las APIs de buscarTrabajadorParaActualizar.php y actualizarDatosTrabajador, se modifica base de datos para que el DNI almacenado admita 128 caracteres por el hash en PHP SHA256 |
| | 6e6cb5d | 16:20:40 +0200 | Funciona eliminar pero faltan ajustes de interfaz, además se modifica tabla trabajadores añadiendo el apartado trabajadores activos como booleano para no perder los datos tras la baja de un trabajador |
| | 5738958 | 17:03:49 +0200 | Funciona eliminar, falta corregir errores en el login para que si el empleado ha dejado de ser activo no pueda entrar |
| | 57e5bc4 | 18:37:30 +0200 | Se añade botón para cerrar sesión que también elimina el token |
| | 9e64d8e | 22:34:17 +0200 | Se crean nuevas clases y adaptadores para poder manejar los turnos de auxiliares, he creado el fragmento para asignar tareas según la fecha y el turno, aún no está acabada, falta activar que al presionar a un auxiliar que trabaje en el día se abra el AsignarTareasEmpleadoFragment.java y posteriormente se le asigne la tarea creada |
| | cd8eaa5 | 22:34:40 +0200 | Se crean nuevas clases y adaptadores para poder manejar los turnos de auxiliares, he creado el fragmento para asignar tareas según la fecha y el turno, aún no está acabada, falta activar que al presionar a un auxiliar que trabaje en el día se abra el AsignarTareasEmpleadoFragment.java y posteriormente se le asigne la tarea creada |
| 19 de mayo | 8b953fd | 12:34:00 +0200 | Funciona el apartado de que aparezcan las tareas en asignar tareas y se asignan de forma correcta |
| | f400510 | 20:34:47 +0200 | Se crean APIs para insertar datos de tarea asignada para un trabajador |
| | 8a9a36f | 19:52:09 +0200 | Se crean nuevos fragmentos para poder asignar turnos a los trabajadores, falta crear la API para crear turno y modificar la API a la que ataca cuando carga los empleados por fecha para que muestre todos los empleados |



| | | | |
|------------|---------|-----------------------|--|
| 20 de mayo | - | | |
| 21 de mayo | 2b7740c | 19:53: 57 +0200 | Se crea API para crear residentes, se crea interfaz y se añade lógica para llamar API que de momento me da un error |
| | 3de0992 | 17:35: 38 +0200 | Asignar turno y mostrar la lista de los turnos por fecha funciona en los fragmentos AsignarTurnos, se me ha dañado el fragmento Home por la modificación de AdaptadorTrabajador.java, debo revisarlo |
| | a1c8061 | 20:49: 37 +0200 | Funciona insertar residente, falta crear fragmento para añadirle familiares y una habitación, esos fragmentos deben ser accesibles desde el fragmento de residentes |
| 22 de mayo | a2261c7 | 21:57: 42 +0200 | Se añade modificación para que la app funcione en al menos el 95% de los teléfonos Android poniéndolo a minsdk 26 |
| | 4488de0 | 23:22: 09 +0200 | Se añade fragmento para añadir habitación y se modifican las APIs para poder asignarlas a un residente |
| 23 de mayo | e322d1c | 18:07: 41 +0200 | Ya funciona listar residentes pero sin la decodificación de SHA256 |
| | 2f06769 | 18:19: 28 +0200 | Mejora visual lista de residentes |
| | b0c3734 | 18:44: 21 +0200 | Mejora visual lista de residentes |
| | a511320 | 21:25: 44 +0200 | Solucionado el problema de desencriptado que tenía con respecto al DNI de trabajadores, ya funciona buscar |
| | a8742c1 | 21:41: 07 +0200 | Ajustes en login y en contraste de color |
| 24 de mayo | - | | |
| 25 de mayo | 67b584d | 19:19: 24 +0200 | Se arregla el bug de modificar residentes, queda pendiente resolver lo de las fotos |
| | 33db619 | 21:24: 11 +0200 | Se muestra imagen de cada residente en la lista de residentes |
| 26 de mayo | b17d61e | 18:56: 48 +0200 | Ya funciona buscar documentos, falta la lógica para descargar o borrar |
| | 5adc386 | 19:43: 08 +0200 | Se modifica interfaz para interactuar en el mismo fragmento de documentos y se mejora |



| | | | |
|------------|---------|-----------------------|---|
| | e347675 | 20:10: 00 +0200 | Funciona lógica para eliminar documentos que da pendiente descargar, mejorada interfaz |
| | 5ec6d4b | 21:08: 41 +0200 | Ya descarga el archivo y el contenido está bien, tengo un problema con la extensión y el nombre |
| | d469f8d | 21:23: 10 +0200 | Ya descarga los archivos de forma correcta probado con JPG y PDF, debo reestructurar la lógica de eliminación y pasar a crear el activity para el Trabajador |
| | f9a0b2a | 22:02: 34 +0200 | Reestructuro con funcionalidad de eliminar resuelta de nuevo |
| | fdd0a1e | 22:28: 32 +0200 | Se añade logo y se tocan colores en modo oscuro, no queda terminado, se ponen algunos comentarios |
| 27 de mayo | ce4137a | 17:46: 08 +0200 | Configuración inicial de Firebase |
| | 10bee1d | 19:19: 48 +0200 | Chat implementado como anónimo, me falta darles persistencia y actualización además de pasarle al chat activity el nombre del usuario |
| | cad5de7 | 22:26: 24 +0200 | Chat ya tiene el nombre, falta persistencia y Firebase no reconoce la app |
| 28 de mayo | be64a47 | 14:38: 49 +0200 | Funcionalidad del chat en tiempo real agregada, queda pendiente para más adelante incluir foto de perfil y poder subir fotos pero ya se guardan los datos en Firebase en un chat grupal |
| | 9751025 | 22:01: 17 +0200 | Se agrega activity y fragmentos para empezar con los casos de usos de un auxiliar |
| 29 de mayo | - | | |
| 30 de mayo | - | | |
| 31 de mayo | ed7480a | 17:22: 01 +0200 | Se crean APIs y lógica para fichar la entrada y salida del auxiliar |
| | 6da4fa3 | 23:46: 25 +0200 | Funcionalidad de mirar turnos del mes, creada la API, pendiente mejorar interfaz |
| 1 de junio | e9cee11 | 00:09: 19 +0200 | Iconos provisionales removidos del menú de auxiliar |
| | ca6cc5e | 21:30: 31 +0200 | Se mejoran los hints de los formularios y se hacen retoques en la interfaz |



| | | | |
|------------|-----------|-----------------------|--|
| | 2243956 | 21:54: 24 +0200 | Se mejoran los hints de los formularios y se mejora la interfaz para administrar empleados |
| | f89225a | 23:20: 06 +0200 | Se mejora la interfaz de varios apartados añadiendo textos descriptivos e iconos, eliminando fondo provisional |
| 2 de junio | e9cee11 | 00:09: 19 +0200 | Iconos provisionales removidos del menú de auxiliar |
| | ca6cc5e | 21:30: 31 +0200 | Se mejoran los hints de los formularios y se hacen retoques en la interfaz |
| | 2243956 | 21:54: 24 +0200 | Se mejoran los hints de los formularios y se mejora la interfaz para administrar empleados |
| | f89225a | 23:20: 06 +0200 | Se mejora la interfaz de varios apartados añadiendo textos descriptivos e iconos, eliminando fondo provisional |
| 3 de junio | 866f028 | 20:27: 17 +0200 | Se agrega logotipo en 2 fragmentos |
| | 1,66E+207 | 21:15: 12 +0200 | Nueva funcionalidad de recuperar contraseña agregada, se crea la API y funciona enviando el mail al correo del DNI proporcionado |
| 4 de junio | 90611b3 | 18:33: 54 +0200 | Incorporada funcionalidad para marcar tarea terminada, se crea API y se prueba en Postman, falta cambiar interfaz cuando la tarea está terminada |
| | a00134b | 18:34: 12 +0200 | Incorporada funcionalidad para marcar tarea terminada, se crea API y se prueba en Postman, falta cambiar interfaz cuando la tarea está terminada |
| 5 de junio | a9dc8f1 | 01:08: 10 +0200 | Corregido error de asignación de fecha errónea en el apartado de asignar tarea |
| | 1f934d9 | 01:02: 50 +0200 | Funciona mostrar tareas asignadas en el fragmento auxiliar |
| 6 de junio | 84542b8 | 21:28: 09 +0200 | Funciona marcar tarea como hecha :) |
| 7 de junio | 6a210b8 | 23:51: 12 +0200 | Cambios en interfaz de buscar |
| | e2f5586 | 23:29: 14 +0200 | Funciona mostrar residentes |



| | | | |
|------------|---------|-----------------------|---|
| 8 de junio | 7725935 | 00:30: 16 +0200 | Añadidos valores para tema nocturno y ajuste en el home auxiliar fragment |
| | 141134 | 12:09: 20 +0200 | Añadida funcionalidad de SMS de emergencia tras presionar un botón, no está testeada en ámbito real |
| | d9adec0 | 16:38: 51 +0200 | Corregido error al insertar residente |
| | 5022cd0 | 19:23: 48 +0200 | Funciona modificar residentes y ya permite subir imágenes más grandes |
| | ff62183 | 23:08: 21 +0200 | Se crea apartado de pagos en documentación, se crea la API y se crea también la tabla y el trigger |
| 9 de junio | - | | |

BITACORA PROYECTO

<https://e-itaca.atlassian.net/jira/software/projects/KAN/boards/1/timeline?selectedIssue=KAN-42&timeline=MONTHS>

5.3. Temporalización y desviación sobre la planificación inicial



El proyecto de desarrollo del sistema de gestión para empleados y residentes comenzó a finales de abril de 2024 y se extendió hasta principios de junio de 2024. A lo largo de este periodo, se ejecutaron múltiples tareas planificadas y no planificadas, reflejando una evolución constante del proyecto. A continuación, se detalla la temporalización y las desviaciones sobre la planificación inicial.



5.3.1. Temporalización del Proyecto

- **28 de abril de 2024:** Inicio del proyecto con los commits iniciales, estableciendo la base del sistema y la conexión a la base de datos.
- **30 de abril de 2024:** Se avanzó en el diseño del esquema de la base de datos y en la implementación de la conexión.
- **Primera mitad de mayo de 2024:** Desarrollo de funcionalidades críticas para el registro de empleados y residentes. Se implementan las interfaces de usuario y la lógica de backend para estas funcionalidades, siguiendo con pruebas unitarias y de integración.
- **Segunda mitad de mayo de 2024:** Enfoque en la creación y mejora de la interfaz de usuario, junto con la implementación de mecanismos de autenticación y seguridad. Se realizaron ajustes y correcciones de errores, asegurando la estabilidad del sistema.
- **Finales de mayo de 2024:** Integración de funcionalidades avanzadas como el chat en tiempo real y la gestión de turnos para los empleados. Se añadieron mejoras visuales y se optimizaron las interfaces de usuario.
- **Principios de junio de 2024:** Finalización de las funcionalidades principales, incluyendo la recuperación de contraseñas, marcación de tareas terminadas y mejoras en la interfaz de usuario. Se realizaron pruebas de carga y optimización de consultas a la base de datos.

5.3.2. Desviaciones sobre la Planificación Inicial

La planificación inicial del proyecto contemplaba una serie de hitos y fechas de vencimiento para las diversas tareas y epics. A medida que el proyecto avanzó, surgieron desviaciones tanto positivas como negativas sobre la planificación inicial.

Adelantamientos:

- Algunas tareas, como la implementación de mecanismos de autenticación (KAN-45) y la creación de manuales de usuario (KAN-40), se completaron antes de lo previsto. Este adelanto permitió dedicar más tiempo a la optimización y mejora de otras funcionalidades debido a que ya se había implementado el sistema de comunicación con todos sus apartados de seguridad.

Retrasos:



- Hubo tareas que experimentaron retrasos debido a problemas técnicos y errores no previstos. Por ejemplo, la implementación de las funcionalidades e interfaces para el rol de enfermería no se pudieron llegar a completar pero se dejó preparada la lógica en la base de datos para meter ese u otros roles en el futuro.
- El ajuste de la interfaz y la corrección de errores en la gestión de residentes (KAN-30 y KAN-32) también llevaron más tiempo de lo anticipado debido a problemas de integración y pruebas.
- Lo que más retraso incorporó a las fechas fue el apartado de seguridad puesto que en diferentes puntos del proyecto se iba a cambiar el sistema de cifrado o encriptación ya que no se entendía muchas veces porque algunos datos si que se encontraban y otros no, por ejemplo la longitud variaba con algunos sistemas y la base de datos tenía los bits necesarios para esperar ese rango de caracteres u otros sistemas que no se podrían desencriptar solo permiten la comparación, y después otros sistemas que no permiten la comparación, por lo cual esto llevaba a cambiar la lógica en todas las APIs para que pudieran responder al nuevo sistema establecido.
- Los intentos de creación de diferentes tipos de interfaz causaban bastante retraso llegado al punto de tener que cambiar la idea inicial de la interfaz y de cómo el usuario debe interactuar con ella, en la mayoría de ocasiones por documentación antigua de librerías de android studio y no haber desarrollado en código Kotlin que tiene más comunidad que java para el desarrollo de app en Android studio.

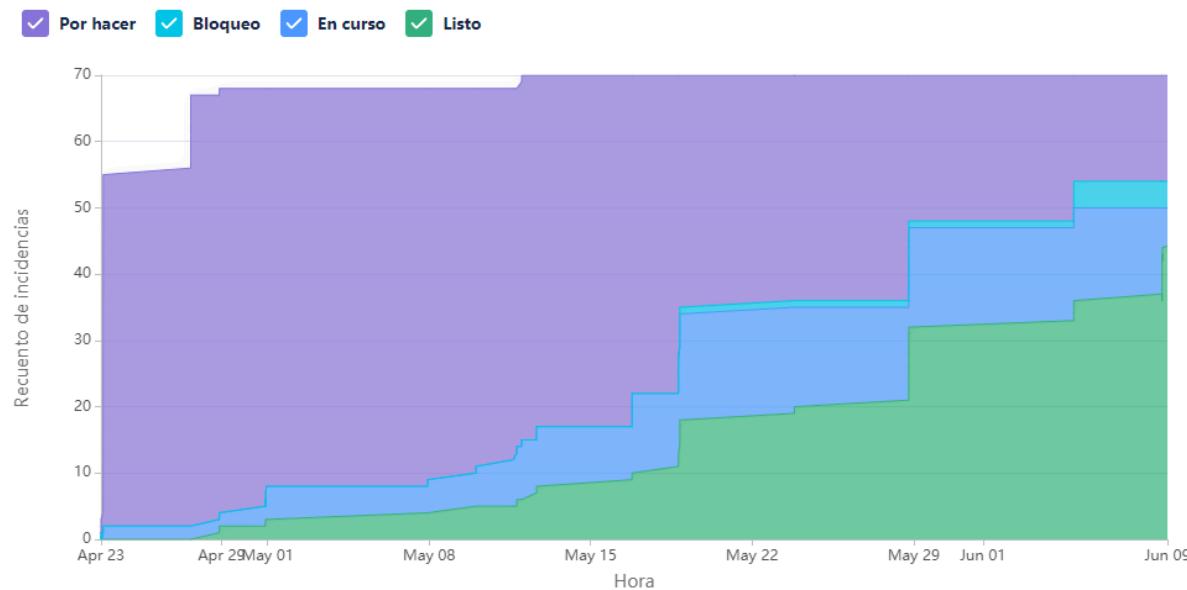
Incorporación de Tareas Adicionales:

- A lo largo del desarrollo, se identificaron nuevas necesidades y se añadieron tareas adicionales que no estaban contempladas inicialmente. La funcionalidad de chat en tiempo real y la gestión de turnos para empleados (KAN-19 y KAN-27) fueron incorporaciones clave que, aunque no planificadas inicialmente, enriquecieron significativamente el sistema. La creación de esta nueva base de datos que a priori no debería haber tardado más de 4 horas pasó a 14 horas por el error en la región de la app.

Reasignación de Prioridades:

- Algunas tareas fueron reprogramadas para acomodar cambios en las prioridades del proyecto. Por ejemplo, la implementación de mejoras visuales y la optimización de la interfaz de usuario (KAN-50 y KAN-51) fueron adelantadas para mejorar la experiencia del usuario antes de lo previsto.

5.3.3. Análisis del Gráfico de Incidencias



El gráfico muestra el recuento de incidencias a lo largo del tiempo, clasificadas por su estado (Por hacer, Bloqueo, En curso, Listo). Desde el 23 de abril hasta el 9 de junio, se observa una tendencia de incremento en el número de incidencias registradas y resueltas, reflejando el progreso del proyecto.

- **Fase Inicial (Finales de abril a principios de mayo):** En esta fase, la mayoría de las tareas se encuentran en estado "Por hacer". Se establecen las bases del proyecto, con un aumento gradual de tareas en curso y completadas a medida que se avanza en la implementación de funcionalidades básicas.
- **Desarrollo Medio (Mediados de mayo):** Hay un aumento significativo en las tareas en curso y completadas. Este periodo coincide con la implementación de funcionalidades críticas y ajustes en la interfaz de usuario. Las tareas en bloqueo son mínimas, lo que indica una buena gestión de los recursos y resolución de problemas.
- **Fase Final (Finales de mayo a principios de junio):** Se observa un aumento en las tareas completadas, indicando que el proyecto está en su fase de madurez y estabilización. Las incidencias en estado "En curso" disminuyen progresivamente, mientras que las tareas "Listo" incrementan, mostrando la culminación de las funcionalidades y la preparación para las pruebas finales.

El proyecto mostró flexibilidad y capacidad de adaptación frente a desafíos y cambios en las prioridades. Las desviaciones sobre la planificación inicial fueron



gestionadas eficazmente, lo que permitió mantener el avance del proyecto sin comprometer su calidad. La capacidad de adelantar algunas tareas y de integrar nuevas funcionalidades no planificadas inicialmente refleja un manejo eficiente del tiempo y los recursos disponibles. A pesar de algunos retrasos, el proyecto se completó en su mayoría dentro del marco temporal previsto, logrando cumplir con los objetivos establecidos.

6. Bibliografía

<https://www.aragon.es/tramitador/-/tramite/subvenciones-de-apoyo-al-software-libre>

Subvenciones de apoyo al software libre

Gobierno de Aragón

2024

<https://www.youtube.com/watch?v=orXJRPNvAc4&t=147s>

(“Android Knowledge”)

Login Page in Android Studio using Java | Explanation Video

Android Knowledge

2023

(“Android Knowledge”)

<https://www.youtube.com/watch?v=0x5kmLY16qE>

Bottom Navigation Bar in Android Studio using Java | Explanation

Android Knowledge

2023

<https://www.youtube.com/watch?v=9FPtSGBHLKY>

Curso de MySQL #40 II Encriptación de datos (aes_encrypt - aes_decrypt)

2022

La Tecnología Avanza

<https://www.youtube.com/watch?v=PKXWfjsl928>

How to [Encrypt and Decrypt] data in Mysql

2021

Angel Geraldo Tech

https://www.youtube.com/watch?v=RK8Xrk-4bwM&list=PLk7v1Z2rk4hjQaV062aE_CW68xgXdYFpV&index=2

2. Android PHP MySQL Tutorial | PHP Database Connection

2016

Simplified Coding



<https://www.youtube.com/watch?v=3vWfd7Y4-mM&list=PL09fOmiA-UwiTiMC0fhb66OC4BnKNu1ZB>

Insertar datos Operación CRUD en MySQL usando Php, Volley Android studio Parte 1
M Guerrero
2021

<https://www.oscarblancarteblog.com/2018/11/30/data-transfer-object-dto-patron-diseno/>

Data Transfer Object (DTO) – Patrón de diseño
Oscar Blancarte
2018

<https://stackoverflow.com/questions/9092712/switch-case-statement-error-case-expressions-must-be-constant-expression>

switch case statement error: case expressions must be constant expression
2023
Benito Bertoli

<https://definicion.de/cliente-servidor/>

Cliente servidor - Qué es, ventajas, definición y concepto
2022
PJulián Pérez Porto y Ana Gardey.

<https://appmaster.io/es/blog/apis-en-arquitectura-de-software#que-son-las-api-y-por-que-son-importantes>

Todo lo que necesita saber sobre las API en la arquitectura de software en 2024
2023
Kareem Nader

<https://www.youtube.com/watch?v=S6i-svU2jOE>

Android Studio.- Subir una imagen a tu base de Datos con MySQL [TUTORIAL]
La cueva del programador
2021

https://www.youtube.com/watch?v=9jx_nE0ok3I

Como encriptar y desencriptar datos en PHP
MemoCode
2022

https://www.youtube.com/watch?v=W_4W5dQg7Vk

MOSTRAR IMAGEN CON ANDROID STUDIO PHP MYSQL
MemoCode
2021

https://www.youtube.com/watch?v=DFnxY_PEnYY

Crear Un Chat
KAD



2017

<https://www.hostinger.es/tutoriales/enviar-emails-usando-php-mail>

Cómo usar PHP Mail y PHPMailer – Guía Completa

Gustavo B.

2023

<https://firebase.google.com/docs/database/android/start?hl=es-419#java>

Conecta tu app a Firebase

Firebase

2023

<https://www.atlassian.com/es/agile/kanban/boards#:~:text=Un%20tablero%20de%20kanban%20es,orden%20de%20su%20trabajo%20diario.>

¿Qué es un tablero de kanban?

MAX REHKOPF

2022

<https://www.ibm.com/docs/es/order-management?topic=users-jwt-authentication>

Llamada a las API REST con autenticación JWT

IBM Corporation

2024

<https://www.ibm.com/docs/es/order-management?topic=users-jwt-authentication>

Autenticación JWT

IBM Corporation

2024

<https://bluuweb.github.io/node/07-jwt/>

Login + JWT

bluuweb

2022

<https://blog.postman.com/what-is-jwt/>

What is JWT?

Gbadebo Bello

2023

<https://es.stackoverflow.com/questions/345918/postman-test-api-y-jwt>

Postman test API y JWT

Alrodrig

2019

