NEILA, TU PUEBLO EN UN SOLO CLICK
NEILA, YOUR TOWN IN A SINGLE CLICK

TRABAJO FIN DE GRADO
CURSO 2024-2025

AUTOR
MARC SEGALÉS PARÉS

DIRECTOR
RAMÓN GONZÁLEZ DEL CAMPO RODRIGUEZ BARBERO

NOTA OBTENIDA POR EL TRIBUNAL: 8

GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

NEILA, TU PUEBLO EN UN SOLO CLICK
NEILA, YOUR TOWN IN A SINGLE CLICK

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA

AUTOR
MARC SEGALÉS PARÉS

DIRECTOR
RAMON GONZALEZ DEL CAMPO RODRIGUEZ BARBERO

CONVOCATORIA: SEPTIEMBRE 2025

GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

1 DE SEPTIEMBRE DE 2025

DEDICATORIAS

A mis padres, por su apoyo incondicional en cada paso de mi vida y por enseñarme a no rendirme nunca.

A mi hermano, por ser siempre un ejemplo y un pilar en el que apoyarme.

A mis amigos, por estar a mi lado en los momentos de esfuerzo y también en los de desconexión, que tanto han hecho falta en este camino.

Y a mi tutor de TFG, por su paciencia, orientación y por guiarme en la realización de este proyecto.

AGRADECIMIENTOS

A Gonzalo, por el tiempo empleado en hacer estas plantillas, a Adrián, Enrique y Nacho, por sus comentarios para mejorar lo que hicimos.

Quiero agradecer en primer lugar a mi familia por su apoyo constante a lo largo de toda esta etapa, especialmente en los momentos más difíciles. A mis amigos, por estar siempre presentes y ayudarme a desconectar cuando más lo necesitaba. Y a mi pareja, por su comprensión, paciencia y motivación inquebrantable durante todo el desarrollo de este proyecto.

También quiero destacar la labor de mi tutor, Ramón, por su acompañamiento y disponibilidad continua durante el trabajo. Y a Narciso, a quien no le ha hecho falta el Anillo Único para coordinarnos a todos.

RESUMEN

Neila, tu pueblo en un solo click

Este Trabajo de Fin de Grado consiste en el diseño y desarrollo de una aplicación móvil orientada a la mejora de la calidad de vida en los pequeños municipios de la llamada "España vaciada". La aplicación permite a los usuarios acceder a información actualizada sobre eventos locales, rutas naturales, puntos de interés y servicios municipales. Además, incluye funcionalidades de reserva de eventos, reporte de incidencias al ayuntamiento y una pasarela de pago integrada.

La solución desarrollada tiene como objetivo fomentar la participación ciudadana, facilitar la digitalización de los servicios municipales y promover el turismo rural. Existen distintos modos de acceso a la aplicación: como invitado, como usuario registrado, o como administrador del ayuntamiento, cada uno con sus permisos y funcionalidades específicas.

La arquitectura del sistema está basada en un cliente móvil desarrollado en Android con Java, y una API REST en Spring Boot conectada a una base de datos MySQL. Se implementa autenticación mediante JWT, persistencia de sesión, y una gestión optimizada del usuario y del pueblo seleccionado para ofrecer una experiencia fluida y personalizada.

Palabras clave

Aplicación móvil, España vaciada, Android, Spring Boot, JWT, eventos, ayuntamiento, API REST, MySQL.

ABSTRACT

Neila, your town in a single click

This Final Degree Project focuses on the design and development of a mobile application aimed at improving the quality of life in small municipalities located in the so-called "emptied Spain" (*España vaciada*). The application allows users to access updated information about local events, hiking routes, points of interest, and municipal services. It also includes features such as event booking, incident reporting to the town hall, and an integrated payment gateway.

The solution is intended to foster citizen participation, support the digitalization of public services, and promote rural tourism. The application offers different access modes: guest, registered user, and town hall administrator—each with specific permissions and functionalities.

The system architecture is based on a mobile client developed in Android using Java, and a REST API built with Spring Boot connected to a MySQL database. It implements authentication via JWT, session persistence, and optimized management of both user and selected town to ensure a smooth and personalized user experience.

Keywords

Mobile application, emptied Spain, Android, Spring Boot, JWT, events, town hall, REST API, MySQL.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

La despoblación rural y el envejecimiento progresivo de la población en pequeñas localidades españolas son fenómenos crecientes que han dado lugar al término "España vaciada". Muchos de estos municipios carecen de herramientas digitales modernas que faciliten la participación ciudadana o la gestión eficiente de servicios locales, lo que contribuye aún más a la desconexión entre residentes especialmente los más jóvenes y su entorno.

En este contexto surge el proyecto **Neila en un solo clic**, una aplicación móvil desarrollada para Android cuyo objetivo principal es digitalizar e integrar distintas funcionalidades municipales y sociales en un único espacio digital. La aplicación permite a los usuarios gestionar de forma centralizada la información y actividades de uno o varios pueblos asociados a su cuenta, rompiendo así con la tradicional fragmentación y limitación a un único municipio. Esta funcionalidad resulta especialmente útil para usuarios con vínculos en distintas localidades (familiares, segundas residencias, raíces personales, etc.).

La aplicación incluye los siguientes módulos clave:

- Consulta y reserva de **eventos locales** mediante una pasarela de pago.
- Reporte de **incidencias municipales**, con seguimiento por parte del ayuntamiento.
- Mecanismo de autenticación mediante **JWT** y gestión segura de sesión.
- **Persistencia** de los pueblos asociados a la cuenta del usuario.
- Sistema de roles (usuario invitado, registrado y administrador de ayuntamiento).

Durante el desarrollo, el proyecto evolucionó desde una idea inicial enfocada exclusivamente a un pueblo concreto (Neila, en la provincia de Burgos) hacia una solución más **escalable y conectada**, con enfoque de red social. Cada usuario puede estar al tanto de lo que ocurre en múltiples pueblos, y participar activamente en cada

uno de ellos, lo cual refuerza el vínculo con sus raíces y mejora la comunicación entre vecinos y administración local.

Además, el sistema está construido siguiendo una arquitectura cliente-servidor: el cliente móvil se ha desarrollado en **Java sobre Android Studio**, y el servidor backend se ha implementado con **Spring Boot**, gestionando la persistencia de datos en una base de datos **MySQL**. La API REST permite una comunicación fluida entre ambos componentes.

En los siguientes capítulos se describen con detalle la motivación del proyecto, sus objetivos, el análisis del estado del arte, las tecnologías utilizadas, la arquitectura de la aplicación, los casos de uso y el desarrollo técnico. Por último, se presentan las conclusiones alcanzadas, así como propuestas de mejora y posibles líneas de trabajo futuro.

1.1 Motivación

La motivación de este proyecto nace de una realidad preocupante pero común: la **España vaciada** continúa perdiendo habitantes año tras año, mientras las nuevas generaciones se alejan no solo físicamente, sino también emocionalmente de sus pueblos. Esta desconexión se ve acentuada por la falta de herramientas digitales que permitan mantener el vínculo con sus raíces y, sobre todo, participar activamente en la vida de los municipios.

Además, muchas de las gestiones que afectan al día a día de un pueblo como la inscripción en eventos, el reporte de incidencias o la comunicación con el ayuntamiento se siguen realizando mediante métodos analógicos: listas manuscritas, avisos en tablones o comunicaciones informales por teléfono o WhatsApp. Estos procesos resultan ineficientes, poco accesibles y, en ocasiones, excluyentes para los jóvenes o para quienes residen fuera del municipio, pero mantienen un vínculo afectivo con él.

Con este proyecto se busca **transformar esa forma de relacionarse con el entorno rural**, creando una aplicación que actúe como **puente digital entre los habitantes, los ayuntamientos y los visitantes**.

La aplicación tiene como propósito:

- **Revitalizar la comunidad local**, facilitando la participación en eventos y la interacción directa con las entidades municipales.
- **Modernizar y digitalizar trámites**, reduciendo la carga administrativa y los errores derivados de los métodos tradicionales.
- **Fomentar el sentimiento de pertenencia y cercanía**, ofreciendo a cada usuario una herramienta viva, práctica y accesible para sentirse parte activa de su pueblo o pueblos.
- **Promover el atractivo del mundo rural**, haciendo más accesibles sus propuestas culturales, deportivas y sociales mediante un canal digital moderno.

En definitiva, este proyecto quiere ofrecer una solución tecnológica realista y escalable, que contribuya a fortalecer el tejido social de los pequeños municipios, ayudando a que las tradiciones se mantengan vivas sin renunciar al progreso.

1.2 Objetivos

Los objetivos de este proyecto nacen como una continuación natural de la motivación planteada: acercar la tecnología al entorno rural, facilitar la participación ciudadana y ofrecer a los vecinos una herramienta sencilla, útil y accesible para mantenerse conectados con sus pueblos, incluso cuando están lejos.

La finalidad principal es desarrollar una **aplicación móvil para Android** que permita a los usuarios gestionar en un solo lugar varios pueblos a los que estén vinculados, accediendo a información relevante, participando en eventos locales y comunicándose con el ayuntamiento mediante una interfaz moderna y cómoda.

Para ello, se definen los siguientes **objetivos específicos**:

1. **Estudiar el contexto rural y definir las funcionalidades clave del proyecto**, analizando las necesidades reales de los municipios pequeños y de sus habitantes en cuanto a comunicación, gestión de eventos y trámites locales. A partir de este análisis, estructurar los módulos principales de la aplicación: eventos, incidencias, sistema multi-ayuntamiento, autenticación de usuarios, entre otros.
2. **Diseñar una solución pensada para el entorno rural**, que cubra necesidades reales como la reserva de eventos y el reporte de incidencias, simplificando procesos que tradicionalmente se hacían de forma manual o informal.
3. **Ofrecer una experiencia de usuario fluida y accesible**, que se adapte a personas con distintos niveles de familiaridad tecnológica, desde jóvenes hasta adultos mayores.
4. **Permitir la gestión de múltiples pueblos por parte de un mismo usuario**, centralizando la información de cada uno y facilitando la interacción directa con los respectivos ayuntamientos.
5. **Crear una base sólida, escalable y reutilizable**, que permita a otros municipios integrarse en el futuro sin necesidad de rehacer el sistema.

En conjunto, se busca crear una aplicación que no solo funcione correctamente desde el punto de vista técnico, sino que tenga un impacto tangible y positivo en la vida de las personas y en el funcionamiento diario de los pueblos, contribuyendo a su modernización sin perder su esencia.

1.3 Plan de trabajo

El desarrollo de este proyecto se ha organizado en distintas fases, siguiendo una estructura lógica y progresiva que ha permitido abordar de forma ordenada todas las tareas necesarias, desde el análisis inicial hasta la implementación final y la documentación. A continuación, se describen las principales etapas del plan de trabajo:

1. Estudio de necesidades y análisis del contexto:

Se realizó una reflexión inicial sobre la situación actual de los pequeños municipios, identificando las carencias en materia de digitalización, así como las oportunidades que ofrece una aplicación móvil para mejorar la participación ciudadana y la eficiencia en la gestión local.

2. Diseño de la aplicación:

A partir del análisis anterior, se procedió al diseño general del sistema. Se comenzó por la estructuración de la base de datos, seguida de la definición detallada de los casos de uso y de todas las funcionalidades clave que debía incluir la aplicación. También se diseñaron las vistas y pantallas principales.

3. Desarrollo e implementación:

En esta fase se abordó la construcción de la aplicación, dividiendo el desarrollo en bloques funcionales:

- Gestión de usuarios, autenticación y roles.
- Asociación y selección de ayuntamientos vinculados al perfil del usuario.
- Módulo de eventos, incluyendo reservas y pasarela de pago.
- Módulo de incidencias y su seguimiento.
- Vista del perfil de usuario y ajustes finales de la interfaz principal.

4. Fase de pruebas:

Una vez desarrolladas las funcionalidades principales, se llevó a cabo una fase de pruebas funcionales. Se validó el correcto funcionamiento de cada módulo, corrigiendo errores y mejorando la experiencia de usuario para garantizar una aplicación estable y usable.

5. Documentación del proyecto:

Para finalizar, se elaboró la memoria técnica del proyecto, recopilando todo el trabajo realizado: motivación, objetivos, diseño, desarrollo, arquitectura, pruebas y conclusiones.

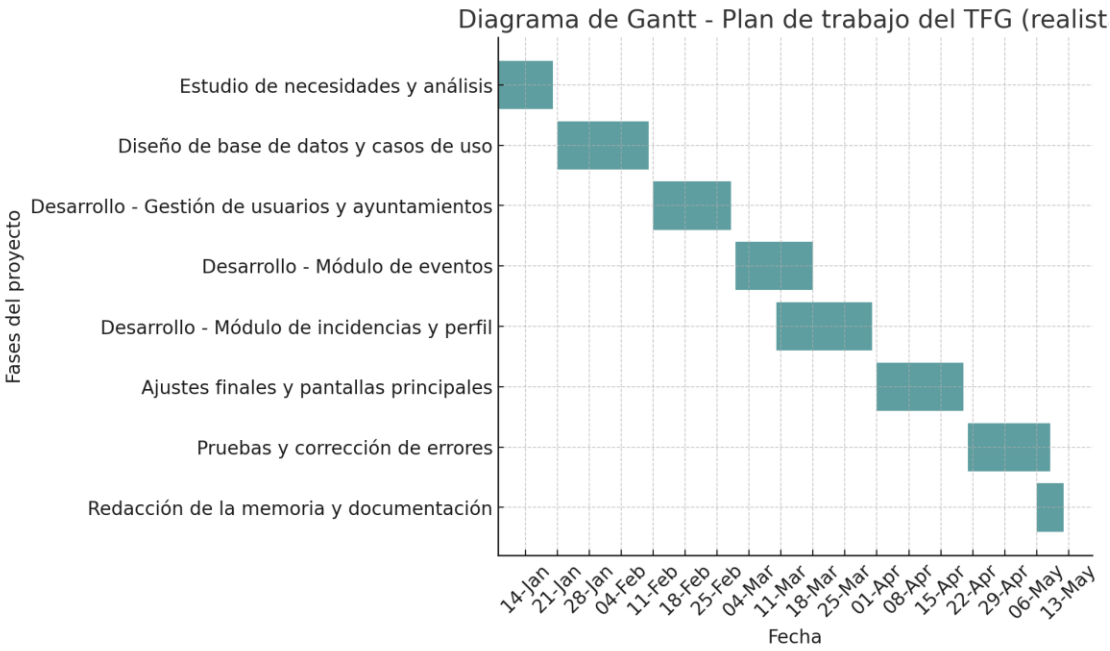

Figura 1-1. Diagrama de GANNT

Capítulo 2 - Estado de la cuestión

En este capítulo vamos a analizar el contexto actual en el que se enmarca este proyecto, incluyendo una revisión de herramientas tecnológicas similares, tanto del ámbito comercial como institucional, con el fin de identificar puntos de referencia, limitaciones y oportunidades de mejora. Este análisis comparativo permite entender cómo encaja la aplicación desarrollada dentro del ecosistema actual de soluciones digitales, especialmente en lo relativo a pueblos pequeños y zonas rurales.

2.1. Aplicaciones móviles con funcionalidades afines

2.1.1. Google Maps

Google Maps es una de las aplicaciones más utilizadas en todo el mundo para geolocalización, consulta de rutas y búsqueda de puntos de interés. Su mayor fortaleza radica en su precisión y en la enorme base de datos de ubicaciones que posee. Sin embargo, su enfoque generalista limita su utilidad para entornos rurales pequeños. En pueblos con escasa actividad digital, Google Maps muestra información incompleta, inexistente o poco relevante.

Figura 2-1. GoogleMaps

Ventajas:

- Alta precisión en la geolocalización.
- Interfaz conocida y fácil de usar.
- Integración con otros servicios de Google.

Limitaciones en comparación con mi aplicación:

- No permite interacción social ni participación ciudadana.
- No gestiona eventos ni permite realizar reservas.
- No ofrece personalización local ni se adapta a cada municipio.
- No está pensada para la comunicación con administraciones locales.

Mi aplicación, en cambio, se centra en facilitar **la vida diaria en el pueblo**, permitiendo consultar y reservar eventos, reportar incidencias directamente al ayuntamiento, y navegar por información relevante específicamente diseñada para ese municipio.

2.1.2. ViveZone

ViveZone es una aplicación orientada a mostrar información de ocio, eventos, comercios y noticias de distintas ciudades o regiones turísticas. Aunque su enfoque es más local que el de Google Maps, sigue estando lejos de adaptarse a la realidad de los pequeños municipios. Su modelo está pensado para zonas urbanas o costeras con afluencia de visitantes, no para pueblos donde el objetivo es sostener y revitalizar la comunidad.

Figura 2-2. ViveZoneApp

Ventajas:

- Muestra eventos, noticias y negocios de zonas urbanas.
- Permite consultar servicios cercanos.
- Dispone de notificaciones y alertas locales.

Limitaciones en comparación con mi aplicación:

- No permite a los usuarios gestionar varios pueblos a la vez.
- No está pensada para pueblos con baja digitalización.
- Su implementación depende de acuerdos comerciales con ciudades.
- No contempla funcionalidades de participación ciudadana como reporte de incidencias.

Mi **app**, sin embargo, ofrece una **herramienta directa para los vecinos**, independientemente de si el pueblo es turístico o no. Se adapta a las limitaciones de conectividad y personal administrativo de los ayuntamientos pequeños, algo que ViveZone no aborda.

2.1.3. Locus Map

Locus Map es una aplicación muy extendida entre aficionados al senderismo y cicloturismo. Su potencia reside en la navegación por rutas y la posibilidad de usar mapas offline. Aunque comparte el interés por los entornos naturales y rurales, su enfoque es totalmente distinto: se limita a la navegación y gestión de rutas, sin incluir funcionalidades sociales ni de gestión municipal.

Figura 2-3. LocusMapApp

Ventajas:

- Potente sistema de rutas con navegación y mapas sin conexión.
- Personalización de capas y compatibilidad con GPX/KML.
- Útil para explorar senderos y espacios naturales.

Limitaciones en comparación con mi aplicación:

- No contempla eventos ni actividades culturales o sociales.
- No permite la interacción entre usuario y ayuntamiento.
- Es una **app** orientada al turismo de naturaleza, no a la vida cotidiana del municipio.

Mi aplicación, en cambio, puede ser usada tanto por residentes como por visitantes, integrando **vida social, cultura, deporte y gestión pública** en un único espacio digital.

2.1.4. Aplicaciones y webs de diputaciones y comunidades autónomas

Durante el proceso de investigación, se analizaron numerosas páginas web y aplicaciones desarrolladas por diputaciones, como la de **Castilla y León, Burgos, Palencia, Soria o León**, así como de otras comunidades como **Aragón, Galicia o Castilla-La Mancha**. Aunque en muchos casos disponen de portales con información turística, noticias, y agenda cultural, su uso es principalmente unidireccional: **el usuario consume información, pero no interactúa**.

Estas plataformas suelen estar pensadas más como escaparate institucional que como herramientas funcionales para el día a día del ciudadano. Además, suelen carecer de adaptación móvil real, o bien su desarrollo se ha quedado desactualizado con el paso del tiempo.

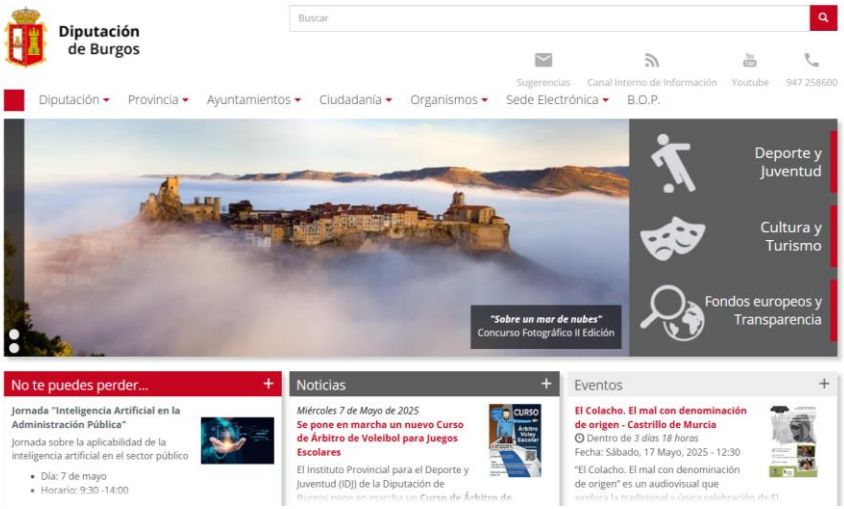

Figura 2-4. Página web Diputación de Burgos

Ventajas:

- Información oficial, validada y fiable.
- Cobertura amplia del territorio.
- En algunos casos, integración con redes sociales.

Limitaciones en comparación con mi aplicación:

- Interfaz poco intuitiva o no adaptada a móviles.
- Ausencia de funcionalidades dinámicas como reservas, notificaciones o incidencias.
- No permiten que un mismo usuario gestione o consulte varios pueblos.
- No contemplan perfiles ni personalización por usuario.

Mi aplicación cubre ese hueco: **actúa como un canal bidireccional**, permite a los usuarios interactuar con el contenido, y a los ayuntamientos recibir y gestionar información de forma eficiente y directa, algo que las plataformas institucionales no están resolviendo actualmente.

2.2. Conclusiones del análisis

Del análisis anterior se concluye que, aunque existen múltiples soluciones tecnológicas relacionadas, **ninguna integra todas las funcionalidades que ofrece esta aplicación**, especialmente en lo que respecta a:

1. Gestión personalizada de **varios municipios en una sola cuenta**.
2. Interacción bidireccional con el ayuntamiento (no solo consulta).
3. Reserva de eventos y pasarela de pago integrada.
4. Enfoque realista y útil para **zonas rurales con baja digitalización**.
5. Escalabilidad para replicarse fácilmente en cualquier pueblo.

Este análisis demuestra que **mi aplicación no solo es técnicamente viable**, sino también **socialmente necesaria**, pues cubre un vacío evidente en el proceso de digitalización local, ofreciendo una herramienta pensada para los pueblos y no simplemente adaptada desde un modelo urbano.

Capítulo 3 - Tecnologías aplicadas

Este capítulo recoge en detalle todas las tecnologías, servicios y librerías externas empleadas en el desarrollo de la aplicación "Neila, tu pueblo a un solo Click", cuyo objetivo principal es conectar a los habitantes y visitantes de los pueblos de la España vaciada mediante una plataforma digital útil, intuitiva y moderna. La elección de cada tecnología ha estado condicionada por criterios de escalabilidad, compatibilidad con Android, facilidad de integración y optimización de la experiencia de usuario.

3.1 Android

La aplicación está desarrollada específicamente para dispositivos Android, dado que se trata del sistema operativo móvil más utilizado a nivel mundial y cuenta con un amplio ecosistema de herramientas y bibliotecas que facilitan el desarrollo.

Android ofrece un entorno de trabajo robusto, flexible y bien documentado, lo que ha permitido implementar una arquitectura sólida y funcional. Las principales tecnologías que se han empleado en el entorno Android son las siguientes:

3.1.1 Java

Java es un lenguaje de programación orientado a objetos, robusto, multiplataforma y con un fuerte tipado que permite desarrollar aplicaciones seguras y mantenibles. Gracias a la máquina virtual de **Java** (JVM), el mismo código puede ejecutarse en distintos dispositivos, lo que garantiza portabilidad.

En el contexto de este proyecto, Java se ha empleado para implementar la lógica del lado cliente en Android: controladores de vistas, gestión de fragmentos y actividades, comunicación con la **API REST** mediante **Retrofit**, así como la manipulación de datos recibidos en formato **JSON**.

3.1.2 Android Studio

El entorno de desarrollo elegido ha sido **Android Studio**, basado en IntelliJ IDEA y oficial para el desarrollo de aplicaciones Android. Su elección responde a las herramientas integradas que ofrece:

1. Compatibilidad con **Gradle**, sistema de compilación que facilita la inclusión de librerías externas.
2. **SDK Manager**, que permite instalar versiones específicas del SDK de Android necesarias para compilar la aplicación.
3. **Android Debug Bridge (ADB)**, empleado para probar la aplicación en emuladores y dispositivos físicos.
4. Compatibilidad con librerías de terceros como **Retrofit** (comunicación REST), **Glide** (carga de imágenes), **CameraX** (gestión de cámara), **Stripe** (pagos) o **Maps** (geolocalización).

Dentro de Android Studio, el archivo **build.gradle** ha permitido gestionar todas las dependencias del proyecto, garantizando que cada librería se integre correctamente en el ciclo de vida de la aplicación.

3.1.3 Servicios Google Play

El uso de **Google Play Services** ha sido fundamental para dotar a la aplicación de funcionalidades basadas en geolocalización. Concretamente, se han empleado:

1. **Google Maps SDK** for Android, que permite mostrar mapas interactivos dentro de la aplicación. Gracias a esta integración, el usuario puede visualizar la localización de eventos, ayuntamientos y otros puntos de interés.
2. **Play Services Location**, utilizado para obtener la ubicación precisa del usuario en tiempo real. Esta funcionalidad es especialmente relevante en el reporte de incidencias, que quedan vinculadas a coordenadas exactas.
3. **Google Places API**, que proporciona información sobre lugares cercanos y enriquece la experiencia de usuario.

Estas herramientas aportan precisión, fiabilidad y rapidez, al estar soportadas en la infraestructura global de Google.

3.1.4 CameraX

Para la captura de imágenes se ha optado por **CameraX** librería oficial de Android que simplifica el uso de la Cámara en comparación con APIs más antiguas. Su elección responde a su facilidad de integración la estabilidad que proporciona y la compatibilidad con una gran variedad de dispositivos Android.

En aplicación, **CameraX** se utiliza en el módulo de incidencias donde los usuarios pueden tomar fotografías de problemas detectados en el municipio para adjuntarlas en sus reportes del ayuntamiento al igual que también se usa en el módulo del perfil de usuario, teniendo la posibilidad de poder editar o cambiar la foto del propio usuario.

3.1.5 Permisos en Android

La aplicación hace uso de recursos sensibles por lo que ha sido necesario solicitar permisos específicos en tiempo de ejecución:

- **Permiso de ubicación**, requerido para mostrar la posición del usuario en los mapas y poder geo referenciar incidencias.
- **Permiso de Cámara,** imprescindible para capturar imágenes desde **CameraX.**
- **Permiso de almacenamiento,** necesario en algunos casos para gestionar imágenes antes de enviarlas al servidor.
- **Permiso de internet**, fundamental para poder realizar las peticiones **REST** al **backend** desarrollo en **Spring Boot**, así como para conectar los servicios externos empleados en la aplicación (**Stripe** para pagos, **Google Maps y Places** para geolocalización y **OpenWeatherMap** para obtener para obtener información meteorológica).

El sistema Android solicita explícitamente algunos de estos permisos al usuario en tiempo de ejecución, garantizando transparencia y seguridad en el uso de datos

sensibles. Otros, como el permiso de Internet, se gestionan directamente desde el **AndroidManifest.xml** de la aplicación, dado que son esenciales para el acceso a la red.

3.2 Backend con Spring Boot y API propia

La aplicación no se limita al cliente móvil: se ha desarrollado una **API** propia mediante **Spring Boot**, que actúa como puente entre el **frontend Android** y la base de datos (**BD**) **MySQL**.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    android:maxSdkVersion="28" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Figura 3-1. Permisos de la App

Spring Boot se ha elegido por su rapidez de configuración, su integración con **JPA/Hibernate** y la facilidad de exponer servicios **REST** de manera estructurada. La **API** implementa endpoints para todas las funcionalidades clave: gestión de usuarios, ayuntamientos, eventos, reservas, incidencias y favoritos.

Además, se ha incorporado un sistema de autenticación con **JWT (JSON Web Tokens)** que asegura que las peticiones sean realizadas únicamente por usuarios autorizados. El **backend** también se encarga de validar datos, aplicar lógica de negocio y coordinar las operaciones con la base de datos.

En definitiva, esta **API** propia garantiza la modularidad del sistema, permitiendo que otros clientes futuros (como una aplicación web o un panel de administración) puedan conectarse al mismo **backend**.

3.3 Base de datos MySQL

La base de datos empleada es **MySQL**, gestionada mediante **XAMPP** para el entorno de desarrollo. **MySQL** ha sido seleccionada por ser un sistema de gestión de

BBDD relacional, estable, ampliamente documentado y compatible con **JPA/Hibernate** en **Spring Boot.**

La base de datos contiene las principales entidades del sistema las cuales son:

- **Usuarios**, con información personal, credenciales y roles.
- **Ayuntamientos**, que representan a cada pueblo integrado en la aplicación.
- **Eventos**, asociados a cada ayuntamiento, con datos como título, fecha, precio y aforo.
- **Reservas**, **participantes, pagos y equipos**, que vinculan las reservas de los usuarios con los eventos.
- **Incidencias**, que almacenan los reportes de problemas enviados por los vecinos.
- **Favoritos**, tabla auxiliar que permite marcar eventos y puntos de interés.

El modelo entidad-relación está diseñado para garantizar integridad referencial mediante claves foráneas, optimizando al mismo tiempo las consultas frecuentes a través de índeces.

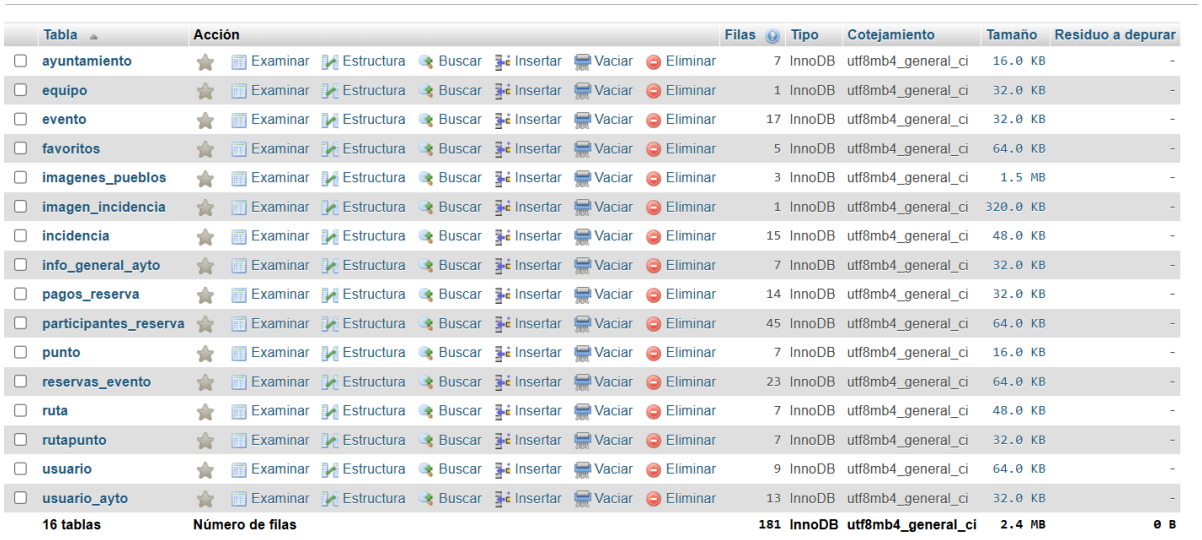

	Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
☐	ayuntamiento	Examinar Estructura Buscar Insertar Vaciar Eliminar	7	InnoDB	utf8mb4_general_ci	16.0 KB	-
☐	equipo	Examinar Estructura Buscar Insertar Vaciar Eliminar	1	InnoDB	utf8mb4_general_ci	32.0 KB	-
☐	evento	Examinar Estructura Buscar Insertar Vaciar Eliminar	17	InnoDB	utf8mb4_general_ci	32.0 KB	-
☐	favoritos	Examinar Estructura Buscar Insertar Vaciar Eliminar	5	InnoDB	utf8mb4_general_ci	64.0 KB	-
☐	imagenes_pueblos	Examinar Estructura Buscar Insertar Vaciar Eliminar	3	InnoDB	utf8mb4_general_ci	1.5 MB	-
☐	imagen_incidencia	Examinar Estructura Buscar Insertar Vaciar Eliminar	1	InnoDB	utf8mb4_general_ci	320.0 KB	-
☐	incidencia	Examinar Estructura Buscar Insertar Vaciar Eliminar	15	InnoDB	utf8mb4_general_ci	48.0 KB	-
☐	info_general_ayto	Examinar Estructura Buscar Insertar Vaciar Eliminar	7	InnoDB	utf8mb4_general_ci	32.0 KB	-
☐	pagos_reserva	Examinar Estructura Buscar Insertar Vaciar Eliminar	14	InnoDB	utf8mb4_general_ci	32.0 KB	-
☐	participantes_reserva	Examinar Estructura Buscar Insertar Vaciar Eliminar	45	InnoDB	utf8mb4_general_ci	64.0 KB	-
☐	punto	Examinar Estructura Buscar Insertar Vaciar Eliminar	7	InnoDB	utf8mb4_general_ci	16.0 KB	-
☐	reservas_evento	Examinar Estructura Buscar Insertar Vaciar Eliminar	23	InnoDB	utf8mb4_general_ci	64.0 KB	-
☐	ruta	Examinar Estructura Buscar Insertar Vaciar Eliminar	7	InnoDB	utf8mb4_general_ci	48.0 KB	-
☐	rutapunto	Examinar Estructura Buscar Insertar Vaciar Eliminar	7	InnoDB	utf8mb4_general_ci	32.0 KB	-
☐	usuario	Examinar Estructura Buscar Insertar Vaciar Eliminar	9	InnoDB	utf8mb4_general_ci	64.0 KB	-
☐	usuario_ayto	Examinar Estructura Buscar Insertar Vaciar Eliminar	13	InnoDB	utf8mb4_general_ci	32.0 KB	-
	16 tablas	Número de filas	181	InnoDB	utf8mb4_general_ci	2.4 MB	0 B

Figura 3-2. Tablas de la BBDD

Gracias a **Hibernate**, las entidades **Java** se mapean directamente a tablas en **MySQL**, lo que simplifica el acceso a datos desde el **backed** y reduce la probabilidad de errores.

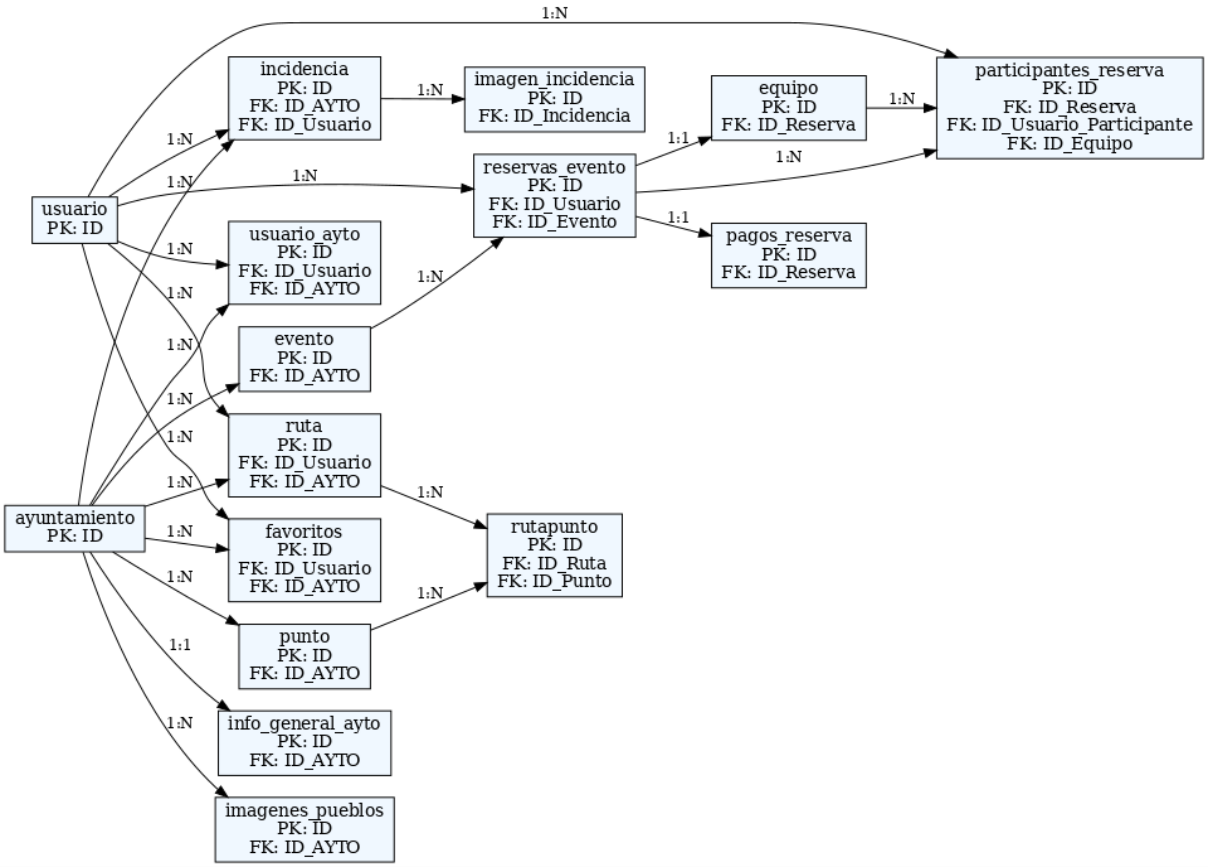

Figura 3-3. Diagrama entidad - relación BD

3.4 Servicios Externos

3.4.1 Stripe

Para la gestión de pagos dentro de la aplicación se ha integrado la plataforma **Stripe**, una de las soluciones más utilizadas y seguras a nivel mundial para el procesamiento de pagos online. El objetivo de esta integración ha sido permitir que los usuarios puedan realizar pagos de forma inmediata mediante tarjeta de crédito o débito al inscribirse en eventos, competiciones u otras actividades de pago disponibles en la aplicación.

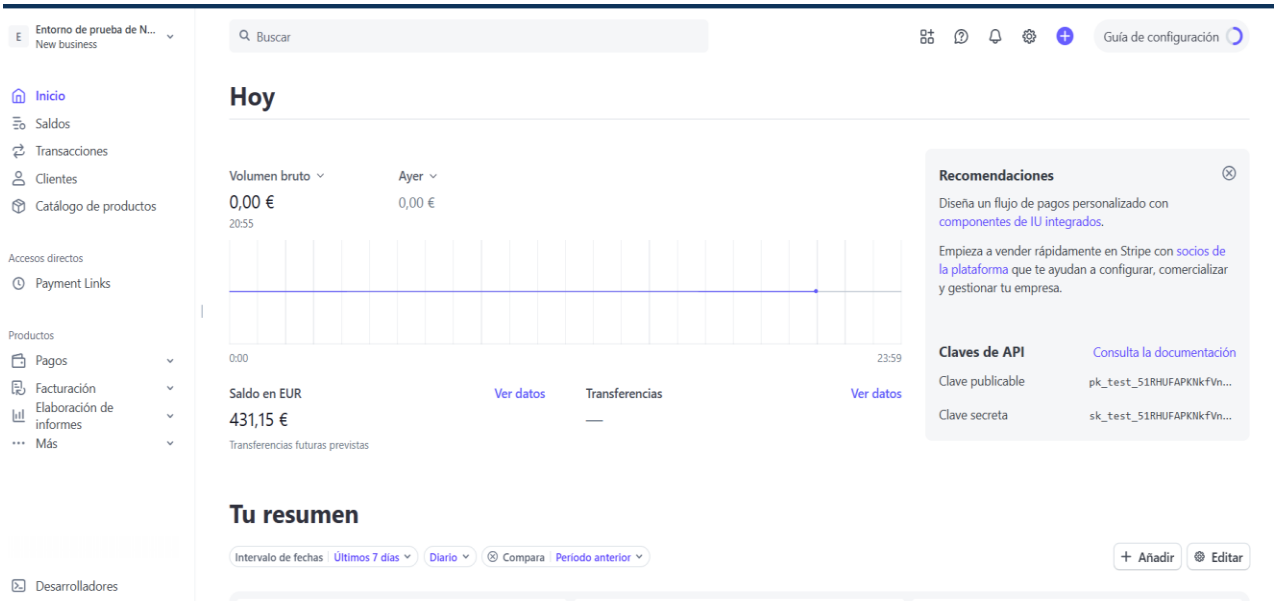

Figura 3-4. Panel de control Stripe

En el **front-end Android** se ha utilizado la librería oficial **com.stripe:stripe-android**, que proporciona componentes listos para generar formularios de pago seguros y gestionar la comunicación con los servidores de **Stripe** sin necesidad de manipular directamente datos sensibles de tarjetas. De esta manera, se cumple con los estándares **PCI DSS**, garantizando la seguridad de las transacciones.

Sin embargo, la integración no se limita únicamente al cliente. En el **backend desarrollado en Spring Boot** se han creado **endpoints propios** que actúan como intermediarios entre la aplicación y la plataforma **Stripe**. Estos endpoints permiten:

1. **Generar las pasarelas de pago** para cada reserva de evento, creando sesiones de pago personalizadas.

2. **Validar el resultado de la transacción**, confirmando si el pago ha sido aprobado o rechazado.

3. **Almacenar en la base de datos** los registros de cada operación, incluyendo el identificador de la transacción, el usuario asociado, el evento reservado, la cantidad abonada y el estado del pago.

De esta forma, el sistema no solo facilita la inscripción en actividades de pago en tiempo real, sino que también mantiene un **control exhaustivo de los pagos realizados** dentro de la propia base de datos del proyecto, permitiendo una correcta trazabilidad de todas las operaciones y facilitando futuras auditorías o gestiones administrativas por parte de los ayuntamientos.

3.4.2 OpenWeatherMap

Otro servicio externo usado es la **API** de **OpenWeatherMap**, la cual permite obtener datos meteorológicos en tiempo real. A diferencia de otros módulos, la integración con **OpenWeatherMap** se realiza directamente desde el **frontend**, evitando el paso por el **backend**.

Esto asegura una mayor inmediatez en la respuesta y reduce la carga en el servidor. Los datos meteorológicos se muestran en la pantalla principal del pueblo seleccionado, de forma que el usuario puede conocer el clima actual antes de decidir asistir a un evento, o consultarlo por cualquier otro motivo.

3.4.3 Google Maps y Places

Además de los servicios ya mencionados de **Google Play**, la integración con **Google Maps y Google Places** ha sido clave para dotar la aplicación de un componente geográfico realista.

Los mapas interactivos permiten visualizar de manera clara tanto los pueblos, asi como también las ubicaciones de las incidencias registradas, y el acceso al mapa y a

la propia ubicación a la hora de la creación de las incidencias por parte de los usuarios. Asimismo, la **API** de **Places** enriquece la experiencia al proporcionar información de contexto sobre la localización.

En conjunto, estas integraciones ofrecen al usuario una aplicación visual, intuitiva y conectada al propio pueblo o territorio.

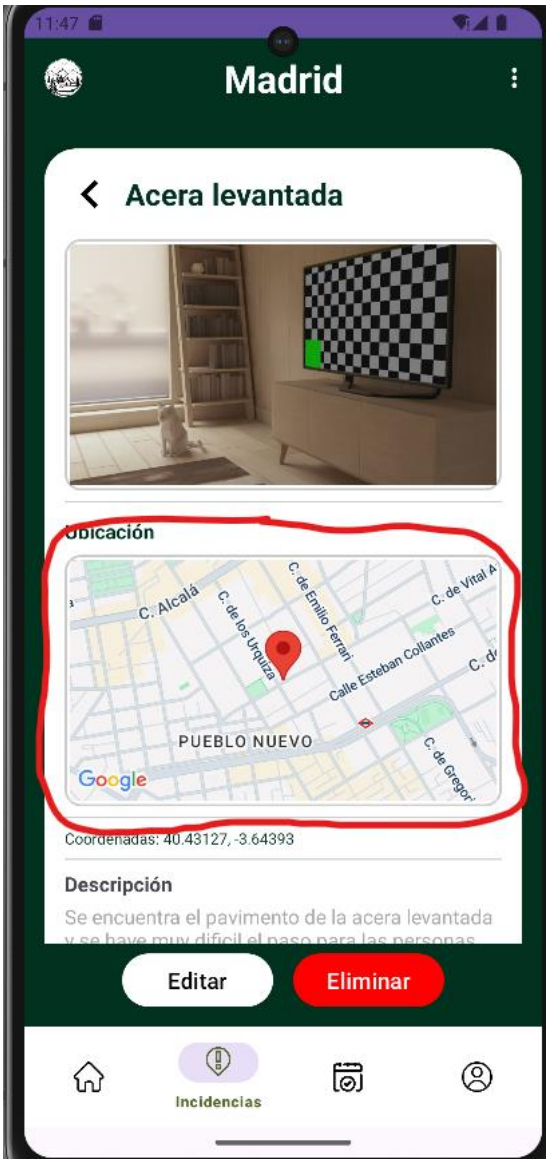

Figura 3-5. Ubicación incidencia

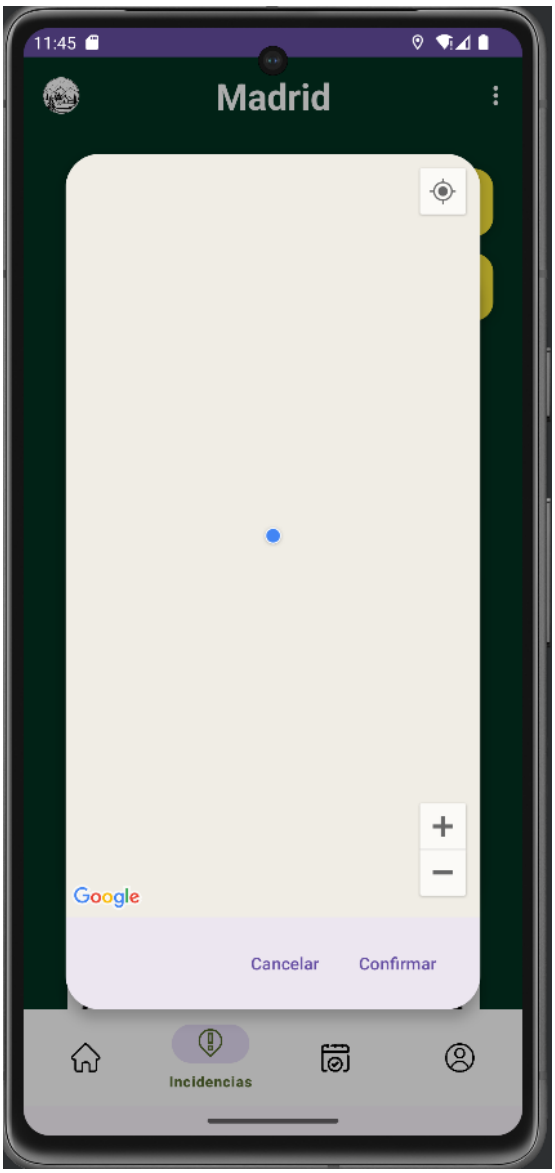

Figura 3-6. Selección ubicación

Capítulo 4 - Especificación de casos de uso

En este capítulo se detallará los casos de uso y los distintos actores dentro de la aplicación, así como la descripción de los diagramas caso de uso.

4.1 Actores

En la aplicación se distinguen tres actores definidos por su nivel de acceso y por las acciones que pueden realizar en el sistema:

- **Usuario invitado:** Puedo utilizar la **app** sin registro previo. Su alcance incluye: seleccionar o cambiar de pueblo/ayuntamiento, consultar la información general y el clima, buscar/filtrar y consultar eventos, ver el detalle de un evento, reservar plaza como invitado, buscar/consultar incidencias, ver el detalle de una incidencia y crear una incidencia. No dispone de perfil coma favoritos ni gestión de ayuntamientos coma si no tiene trazabilidad personal de reservas más allá del justificante de operación.

- **Usuario registrado:** Accede con credenciales a la **app**, lo cual hace que se disponga de una experiencia completa y personalizada. Además de todo lo que pueda hacer un usuario invitado, o puede: iniciar/cerrar sesión y registrarse, **ver/editar su perfil, añadir/consultar sus favoritos**, reservar eventos y **consultar/cancelar sus reservas**, crear y eliminar sus propias incidencias, y gestionar su relación con los municipios, es decir, seleccionar, unirse y gestionar ayuntamientos asociados a su cuenta. Este actor concentra la mayoría de los casos de uso avanzados.

- **Usuario admin_Ayto:** Es el actor institucional que recibe y gestiona el resultado operativo de las acciones de los usuarios. Puede crear/editar/cancelar eventos administrar asistentes y a foro, recibir/validar/actualizar/cerrar incidencias reportadas por los ciudadanos y publicar avisos oficiales. Aunque parte del panel municipal esté en desarrollo se incluye por ser núcleo del objetivo del proyecto

- **Super Usuario**: Es el actor con el rol más elevado dentro de la aplicación y su función principal es supervisar la creación y validación de nuevos ayuntamientos en la plataforma. Su cometido consiste en gestionar las solicitudes realizadas por aquellos usuarios que desean darse de alta como **Admin_Ayto** y registrar oficialmente un nuevo municipio en el sistema. El Super Usuario es, por tanto, quien autoriza o rechaza dichas peticiones, asegurando que solo se registren administradores legítimos y que los pueblos dados de alta sean verificados correctamente. De esta manera, se garantiza el control, la fiabilidad y la seguridad en el crecimiento de la red de municipios disponibles en la aplicación.

4.2 Casos de uso

Teniendo en cuenta los actores mencionados en el punto anterior, se muestran una descripción de las distintas acciones y funciones principales que pueden llevar a cabo cada uno de dichos actores, descritas en el diagrama de casos de uso, que se puede ver en las siguientes figuras. También se detallan los casos de uso por cada módulo.

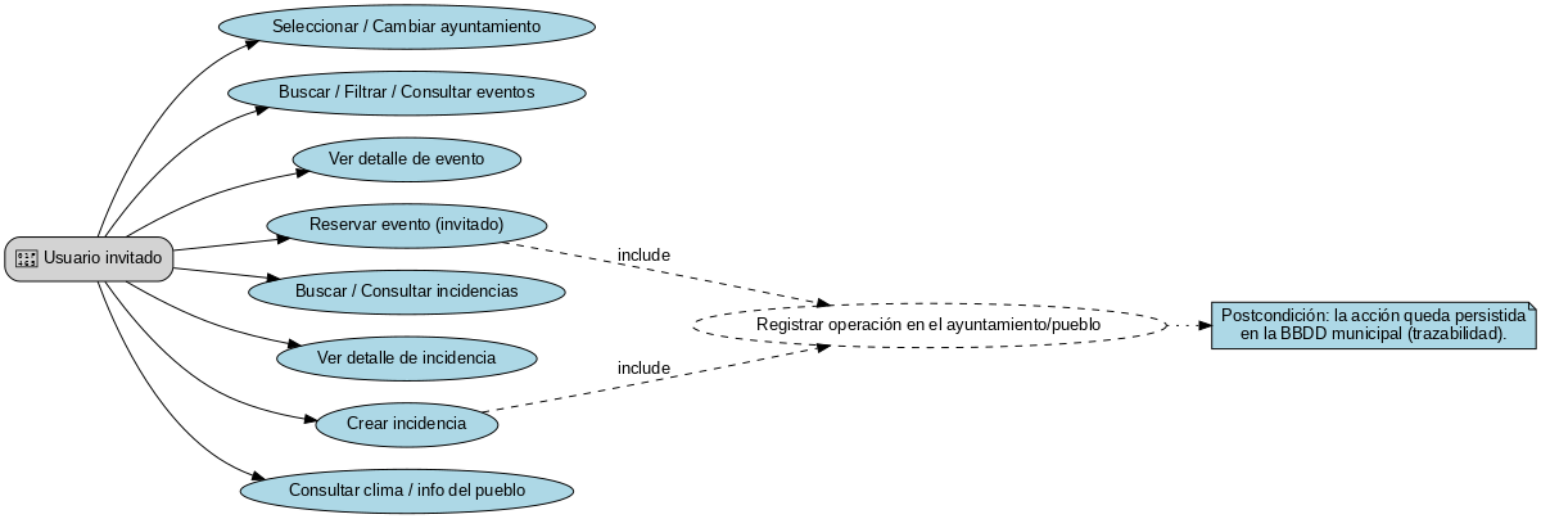

Figura 4-1. Casos de uso Usuario invitado

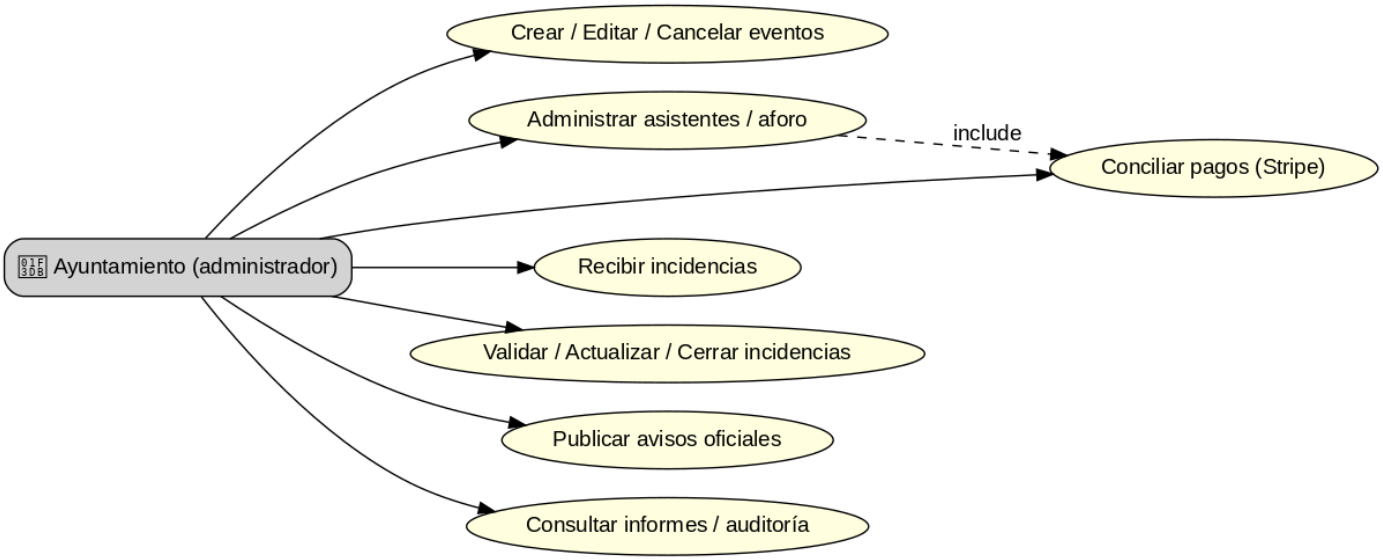

Figura 4-2. Casos de uso Ayuntamiento

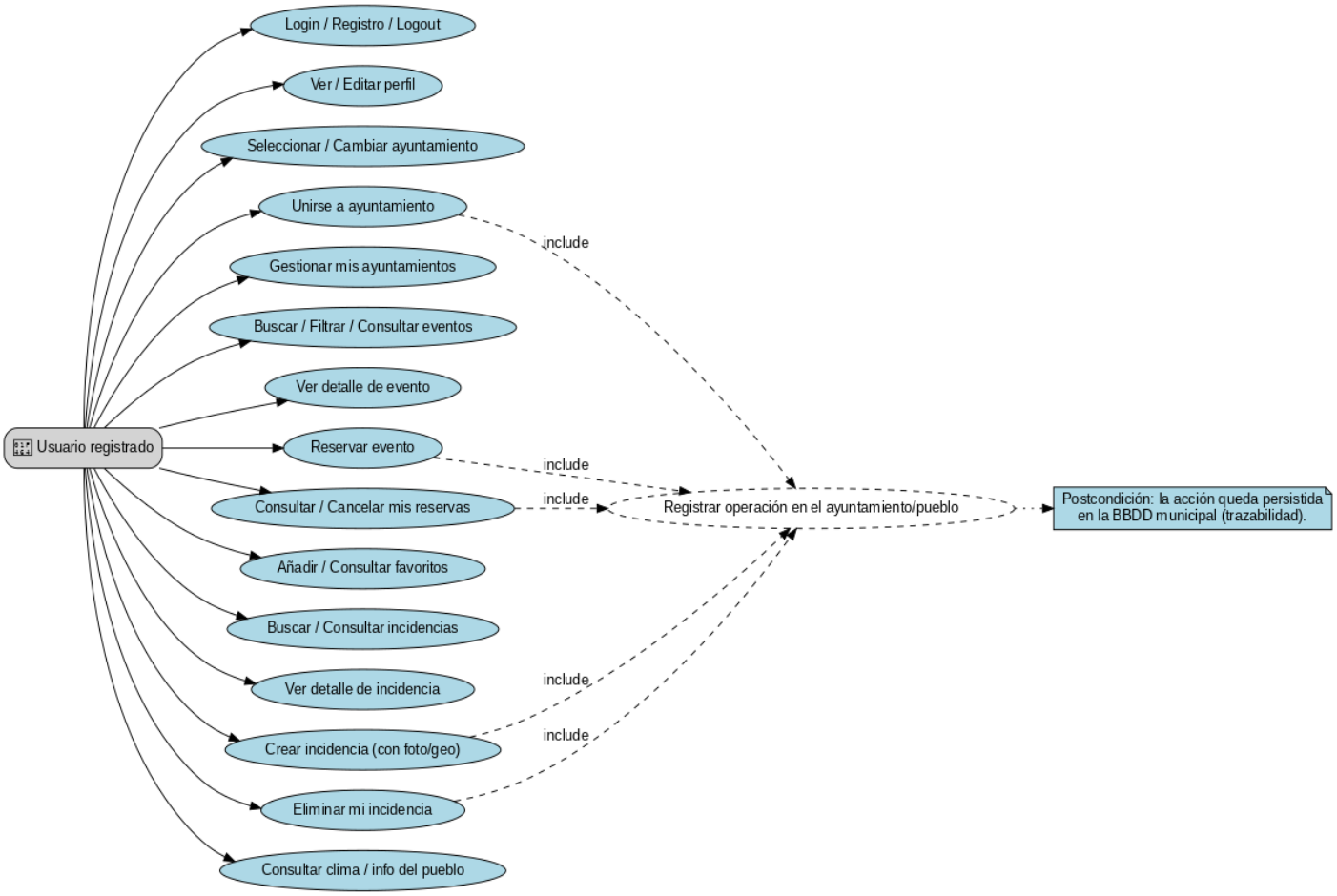

Figura 4-3. Casos de uso Usuario registrado

En el anexo 1 se muestran los casos de uso o escenarios más relevantes de la app de cada uno de los actores que interactúan.

Capítulo 5 - Arquitectura y modelo de datos

En este capítulo se detalla la arquitectura de la **app** (cliente **Android** y **backend** con **API propia**) y el modelo de datos empleado para soportar todas las funcionalidades descritas con anterioridad.

5.1 Arquitectura de la app

La arquitectura de la aplicación se estructura en dos grandes bloques; El cliente móvil y la **API** del **backend**.

Por un lado, el cliente de Android dispone de cuatro actividades principales, entre las que destaca el **DrawerMainActivity** que actúa como el esqueleto central de la interfaz. Gracias al uso de un **navigation drawer** (el panel lateral desplegable) hoy el usuario puede desplazarse fácilmente entre las distintas secciones de la **app**. Dentro de esta actividad principal se carga la mayoría de las pantallas como **Fragments** lo que aporta flexibilidad y modularidad coma y permitiendo reutilizar componentes de interfaz y mantener el estado del usuario y del municipio activo sin necesidad de crear actividades continuamente.

El uso de fragments como patrón de diseño evita tener que crear un **Activity** para cada pantalla lo que redunda en una aplicación más ligera y mantenida. Además, se complementa con el ciclo de vida de los fragmentos, que permite gestionar de manera eficiente la navegación hacia atrás, la recuperación de Estado hola actualización dinámica de la información.

Por otro lado, el **backend** se ha construido con **Spring Boot**, si de un enfoque de **API REST.** En esta etapa se centralizan todas las operaciones y lógica de negocio. El **backend** expone los servicios a través de **enpoints** seguros, que son consumidos por la aplicación móvil mediante **Retrofit**. Se ha implementado un sistema de interceptores para añadir automáticamente la cabecera de autenticación a la mayoría de las peticiones, lo que significa la gestión de credenciales en el cliente.

Un aspecto clave de la arquitectura es la integración con el servicio de externos. Para los pagos se ha utilizado **Stripe**, cuya lógica se reparte en el cliente y el **backend**, para la información geográfica y geolocalización se ha integrado **Google Maps** y para la consulta del clima en tiempo real se ha integrado la **API OpenWeatherMap**.

En conjunto, la arquitectura adaptada permite ofrecer una experiencia fluida y coherente. El usuario ya se ha registrado o invitado selecciona un ayuntamiento desde el que operar y a partir de ese momento, toda la interacción queda asociada a ese municipio. El **backend** garantiza que cada acción se registra en la base de datos del ayuntamiento correspondiente, mientras que la **app** móvil se encarga de ofrecer una interfaz intuitiva y consistente gracias al uso del **drawerMainActivity**. y los **Fragments**.

5.2 Modelo de datos

En cuanto a la estructura de la base de datos y el modelo de los datos de la aplicación, se ha implementado en **MySQL**, Un sistema de gestión de bases de datos relacional ampliamente extendido por su fiabilidad, eficiencia y facilidad de integración con aplicaciones web y móviles. El modelo de datos diseñado para esta aplicación se estructura en varias tablas que representan las principales entidades del sistema.: usuarios como ayuntamientos. Eventos, reservas, pagos, incidencias y favoritos. Entre otras. La relación entre éstas permite gestionar de forma consistente las operaciones habituales de la aplicación. A continuación, se describen las principales tablas junto con sus atributos y relaciones existentes entre ellas.

Tabla "usuario"

- ID → Tipo: INT (PK, autoincremental)
- Nombre → Tipo: VARCHAR(255)
- Apellido → Tipo: VARCHAR(255)
- Email → Tipo: VARCHAR(255), **único**
- ContraseñaHasheada → Tipo: VARCHAR(255)
- Salt → Tipo: VARBINARY(255)

- Tfno → Tipo: VARCHAR(255)
- DNI → Tipo: VARCHAR(255), **único**
- Administrador → Tipo: TINYINT(1)
- FotoPerfil → Tipo: BLOB

Tabla "ayuntamiento"

- CLAVE_UNICA_AYTO → Tipo: INT (PK, autoincremental)
- Localidad → Tipo: VARCHAR(255)
- Provincia → Tipo: VARCHAR(255)
- Latitud → Tipo: DOUBLE
- Longitud → Tipo: DOUBLE
- Telefono → Tipo: VARCHAR(255)
- Email → Tipo: VARCHAR(255)
- Historia → Tipo: TEXT
- Imagen_Ppal → Tipo: VARCHAR(255)
- Escudo → Tipo: VARCHAR(255)
- Contacto → Tipo: TEXT
- Corporacion_Municipal → Tipo: TEXT

Tabla "usuario_ayto" *(relación N:M usuario–ayuntamiento)*

- ID_USUARIO → Tipo: INT (FK a usuario)
- ID_AYTO → Tipo: INT (FK a ayuntamiento)
- *(Clave primaria compuesta por ID_USUARIO + ID_AYTO)*

Tabla "evento"

- ID → Tipo: INT (PK, autoincremental)
- ID_filtrable → Tipo: VARCHAR(255)

- Nombre → Tipo: VARCHAR(255)
- Descripcion → Tipo: VARCHAR(255)
- Fecha_Inicio → Tipo: DATE
- Fecha_Fin → Tipo: DATE
- Hora → Tipo: TIME
- Direccion → Tipo: VARCHAR(255)
- Latitud_evento → Tipo: DOUBLE
- Longitud_evento → Tipo: DOUBLE
- ID_AYTO → Tipo: INT (FK a ayuntamiento)
- Foto_Principal → Tipo: VARBINARY(255)
- Precio → Tipo: VARCHAR(255)
- Capacidad → Tipo: INT
- Apuntados → Tipo: INT
- TipoEvento → Tipo: ENUM('normal','competicion_individual','competicion_grupal')
- MaxEquipos → Tipo: INT
- NumEquipos → Tipo: INT
- MaxParticipantesEquipo → Tipo: INT

Tabla "reservas_evento"

- ID → Tipo: INT (PK, autoincremental)
- ID_Usuario → Tipo: INT (FK a usuario, NULL si invitado)
- ID_Evento → Tipo: INT (FK a evento)
- ID_AYTO → Tipo: INT (FK a ayuntamiento)
- Estado → Tipo: ENUM('confirmada','pendiente_pago','cancelada')
- CantidadEntradas → Tipo: INT

- Fecha_Reserva → Tipo: DATETIME
- Mail_guest → Tipo: VARCHAR(255)

Tabla “pagos_reserva”

- ID → Tipo: INT (PK, autoincremental)
- ID_Reserva → Tipo: INT (FK a reservas_evento)
- Monto → Tipo: DECIMAL(10,2)
- MetodoPago → Tipo: ENUM('efectivo','tarjeta','bizum')
- Pagado → Tipo: TINYINT(1)
- Fecha_Pago → Tipo: DATETIME

Tabla “equipo”

- ID → Tipo: INT (PK, autoincremental)
- Nombre → Tipo: VARCHAR(255)
- ID_Reserva → Tipo: INT (FK a reservas_evento)
- NumParticipantes → Tipo: INT

Tabla “participantes_reserva”

- ID → Tipo: INT (PK, autoincremental)
- ID_Reserva → Tipo: INT (FK a reservas_evento)
- Nombre → Tipo: VARCHAR(255)
- Apellido → Tipo: VARCHAR(255)
- Email → Tipo: VARCHAR(255)
- EsLogado → Tipo: TINYINT(1)
- ID_Usuario_Participante → Tipo: INT (FK a usuario, NULL si invitado)
- ID_Equipo → Tipo: INT (FK a equipo, NULL si no pertenece a ninguno)

Tabla “incidencia”

- ID → Tipo: INT (PK, autoincremental)
- ID_AYTO → Tipo: INT (FK a ayuntamiento)
- ID_Usuario → Tipo: INT (FK a usuario, NULL si invitado)
- Nombre → Tipo: VARCHAR(255)
- Descripcion → Tipo: VARCHAR(255)
- Latitud → Tipo: DOUBLE
- Longitud → Tipo: DOUBLE
- Direccion → Tipo: VARCHAR(255)
- Estado → Tipo: ENUM('Cerrada','EnProceso','Pendiente') DEFAULT 'Pendiente'
- Fecha_Creacion → Tipo: DATETIME
- Ultima_Actualizacion → Tipo: DATETIME

Tabla “imagen_incidencia”

- ID → Tipo: INT (PK, autoincremental)
- ID_INCIDENCIA → Tipo: INT (FK a incidencia)
- Base64 → Tipo: LONGTEXT

Tabla “favoritos”

- ID → Tipo: INT (PK, autoincremental)
- ID_Usuario → Tipo: INT (FK a usuario)
- Tipo → Tipo: ENUM('evento','ruta','punto_interes')
- ID_Referencia → Tipo: INT
- ID_AYTO → Tipo: INT (FK a ayuntamiento)
- Fecha_Favorito → Tipo: DATETIME

Tabla “info_general_ayto”

- ID → Tipo: INT (PK, autoincremental)
- ID_AYTO → Tipo: INT (FK a ayuntamiento)
- Telefono → Tipo: VARCHAR(20)
- Email → Tipo: VARCHAR(100)
- Direccion → Tipo: VARCHAR(255)
- WebAyuntamiento → Tipo: VARCHAR(255)
- HorarioAtencion → Tipo: VARCHAR(255)
- RedesSociales → Tipo: TEXT
- CorporacionMunicipal → Tipo: TEXT
- OtrosDatosInteres → Tipo: TEXT

Tabla “imagenes_pueblos”

- ID → Tipo: INT (PK, autoincremental)
- ID_Ayuntamiento → Tipo: INT (FK a ayuntamiento)
- ImagenBase64 → Tipo: LONGTEXT

Capítulo 6 - Funcionamiento de la Aplicación

En este capítulo se describen detalladamente el contexto de uso y las distintas funcionalidades que conforman la aplicación, explicando el propósito, alcance y las vistas que permiten al usuario interactuar con el sistema.

6.1 Contexto

El objetivo principal de la aplicación es proporcionar una herramienta integral que acerque la gestión municipal y la vida cotidiana de los pueblos a los ciudadanos, facilitando la digitalización de los procesos que tradicionalmente se realizan de manera manual o presencial. La aplicación está concebida con un espacio digital en el que los habitantes, visitantes e incluso propios ayuntamientos pueden puede interactuar en torno a 3 ejes principales, los eventos locales, incidencias ciudadanas y la gestión de perfiles tanto de usuario como del municipio.

La navegación dentro de la aplicación combina dos patrones habituales en **Android**. Por un lado, el menú lateral **(Drawer),** que ofrece acceso rápido a las secciones principales del ayuntamiento opciones de gestión. Por otro lado, el **menú inferior de pestañas**, que permite alternar de manera inmediata entre las funcionalidades más usadas por los usuarios, como son los eventos, las incidencias, el perfil personal y la página inicial. Esta **doble vía de navegación** facilita una experiencia **intuitiva** y **accesible** tanto para usuarios novatos como para usuarios habituales.

Figura 6-1. Menú inferior de navegación

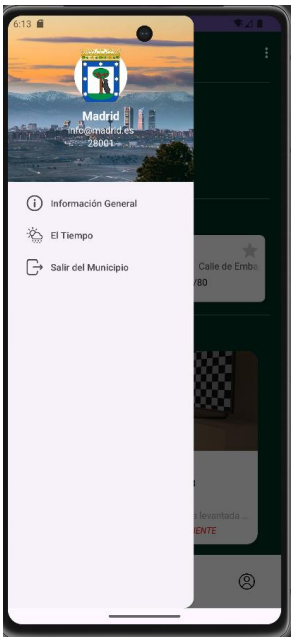

Figura 6-2. Menú lateral de navegación

6.2 Gestión de usuarios y perfil

El acceso a la aplicación se organiza en torno a la gestión de usuarios, que constituye el punto de partida para poder interactuar con las distintas funciones. Para ello, se ha desarrollado un sistema propio de autenticación basado en **API REST** y **JWT,** con almacenamiento en **MySQL**, que permite tanto el registro de nuevos usuarios como el inicio y cierre de sesión de los ya existentes.

Además de la autenticación, la aplicación ofrece un flujo de selección o Unión a ayuntamientos, ya que cada acción realizada por el usuario debe quedar vinculada a un municipio concreto.

La aplicación contempla dos tipos de principales de usuarios, cada 1,1 alcance diferente sobre las funcionalidades.

- **Usuario registrado**: dispone de una cuenta personal creada mediante registro. Este tipo de usuario tiene acceso completo a la aplicación, pudiendo unirse a varios ayuntamientos seleccionar a un municipio activo. Reservar eventos con pago integrado, gestionar sus reservas, marcar favoritos, editar su perfil, además de todas las acciones que quedan asociadas a su cuenta en la **bd**, lo que permite mantener la trazabilidad y la persistencia.

- **Usuario invitado**: Puede acceder sin necesidad de registro, seleccionado únicamente un ayuntamiento desde el cual operar. Sus funcionalidades son más limitadas, puede consultar la información del pueblo, acceder al detalle de eventos, reservar plazas en eventos y reportar incidencias, pero no dispone de un perfil persistente ni de acceso a favoritos o gestión de reservas.

- **Usuario Admin_Ayto**: Representa a los alcaldes o responsables municipales encargados de gestionar un ayuntamiento dentro de la **app**. Tiene la capacidad de crear y editar eventos, administrar asistentes y aforo, así como modificar el estado de incidencias, reportadas por los vecinos o visitantes del pueblo, hasta su cierre definitivo. Además, puede publicar información oficial del municipio y mantener actualizado el perfil del ayuntamiento (funcionalidades en proceso, a futuro), actuando como punto central de comunicación digital entre el consistorio y la ciudadanía.

- **Super Usuario**: Es el rol de mayor nivel dentro de la **app** y su función principal es supervisar la creación y validación de nuevos municipios. Se encarga de revisar y aprobar las solicitudes de alta realizadas por los usuarios que desean crear su pueblo y convertirse en **Admin_Ayto,** garantizando que solo se registren administradores legítimos y que los pueblos creados en la plataforma estén verificados. De este modo, asegura la fiabilidad y el correcto crecimiento de la red de municipios disponibles en la **app**.

Esta diferenciación de roles permite que cualquier persona puede usar la aplicación de manera inmediata como invitado. Al tiempo que se ofrece una experiencia más completa y personalizada a los usuarios registrados.

6.2.1 Registro

El primer paso, en caso de que el usuario no disponga de una cuenta, es completar el proceso de registro. Para ello, la aplicación ofrece un formulario accesible desde la pantalla inicial en el que se solicitan varios datos personales necesarios para crear una cuenta válida en el sistema. A diferencia de otros sistemas más simples que únicamente requieren correo electrónico y contraseña, en esta aplicación se han incluido más campos para garantizar una mayor identificación de los usuarios.

Hoy el formulario de registro solicita los siguientes datos:

- **Nombre y apellidos,** que permite identificar al usuario en el perfil y en las reservas.
- **Email,** Que actúa como el identificador único de acceso. El sistema valida que el correo tenga un formato correcto y que no exista ya en la **bd.**
- **DNI,** Requerido para asegurar la unidad de cada usuario. Al igual que el correo, este campo debe ser único y validado antes de proceder.
- **Número de teléfono**, que puede usarse como dato de contacto adicional.

- **Contraseña y confirmación de contraseña,** que deben coincidir y cumplir unos requisitos mínimos de seguridad (Longitud suficiente, combinación de caracteres, etc).

Una vez introducido los datos y validados en el cliente, la aplicación enviará la

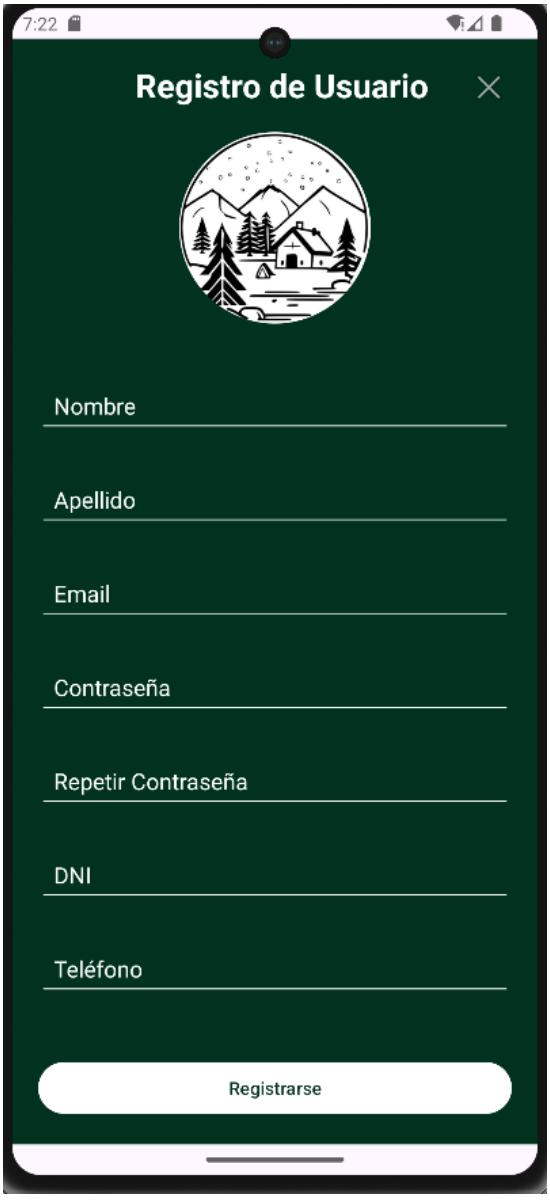

Figura 6-3. Plantilla de registro

solicitud de registro al **backend** que gestiona la operación mediante la **API REST**. El servidor se encarga de comprobar la unicidad del correo y el DNI. Cifrar la contraseña mediante un algoritmo de hash. Con sal. Y crear un nuevo registro en la tabla de usuario de la base de datos **MySQL**.

En caso de éxito, el sistema devuelve una confirmación y el usuario queda registrado en la aplicación, pudiendo acceder inmediatamente con sus credenciales. En caso de error, se informa al usuario mediante mensajes de validación en la propia interfaz.

Tras registrarse, el usuario podrá iniciar sesión y en su primer acceso, seleccionar o unirse a un ayuntamiento para comenzar a utilizar funcionalidades disponibles.

```
public UsuarioDTO registrarUsuario(UsuarioRegistroDTO userRegistro) {
    if (!userRegistro.getContraseña().equals(userRegistro.getContraseña2())) {
        return null; // Devuelve null si las contraseñas no coinciden
    }

    if (usuarioRepository.existsByEmail(userRegistro.getEmail())) {
        // El nombre de usuario ya está en uso, retorna null
         return null;
    }else{

        // Crear usuario
        Usuario usuario = new Usuario();
        usuario.setNombre(userRegistro.getNombre());
        usuario.setApellido(userRegistro.getApellido());
        usuario.setEmail(userRegistro.getEmail());
        usuario.setDni(userRegistro.getDni());
        usuario.setTfno(userRegistro.getTelefono());

        // Generar hash y salt para la contraseña
        byte[] salt = UserUtils.generateSalt();
        String hashedPassword = hashPassword(userRegistro.getContraseña(), salt);
        usuario.setContraseñaHasheada(hashedPassword);
        usuario.setSalt(salt);

        // Guardar el usuario en la base de datos
        usuarioRepository.save(usuario);
        //jwtUtil.generateToken(usuario.getEmail());
        return converter.toDTO(usuario);
    }
}
```

Figura 6-4. Método Registro API

6.2.2 Login

El inicio de sesión se realiza introduciendo el **email** y la **contraseña** del usuario. El **backend** validar las credenciales y si son correctas genera el **token JWT** que se almacena en el dispositivo para autenticar las siguientes peticiones. Tras acceder, el

sistema solicita a seleccionar un ayuntamiento activo o, en caso de no tener ninguno, unirse a 1 antes de continuar.

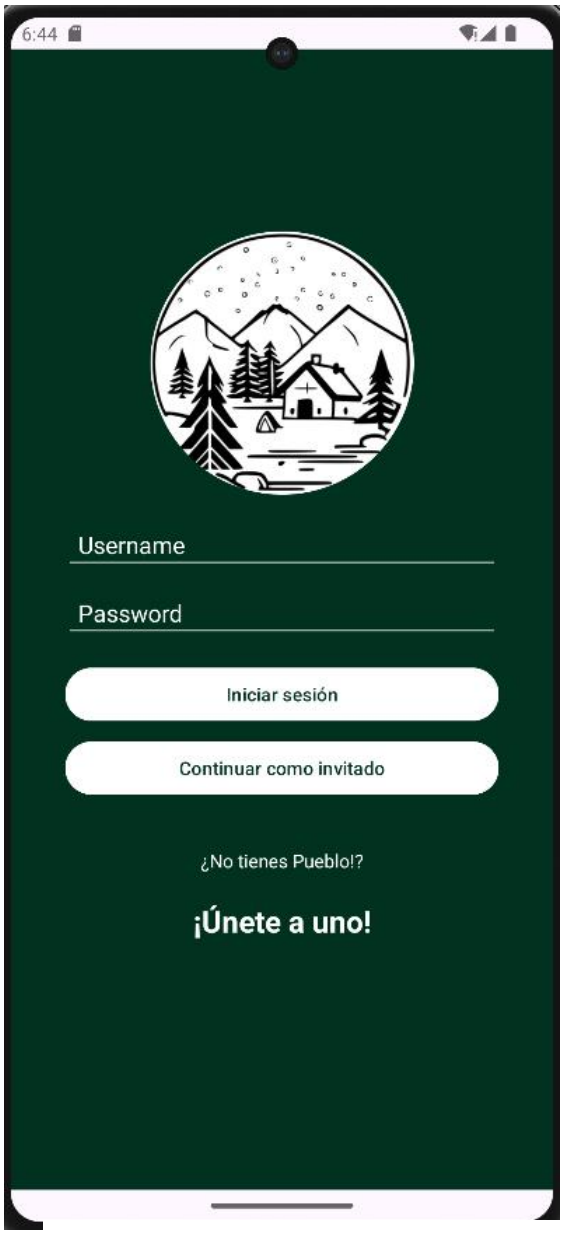

Figura 6-5. Pantalla de Login

```java
public UsuarioLogado login(String email, String password) {
    try {
        Optional<Usuario> usuarioOpt = usuarioRepository.findByEmail(email);
        if (usuarioOpt.isEmpty()) {
            throw new RuntimeException("Usuario no encontrado.");
        }

        Usuario usuario = usuarioOpt.get();
        String hashedPassword = hashPassword(password, usuario.getSalt());

        if (!hashedPassword.equals(usuario.getContraseñaHasheada())) {
            throw new RuntimeException("Contraseña incorrecta.");
        }

        // Generar token y devolver usuario logado
        String token = jwtUtil.generateToken(usuario.getEmail());
        userSession = new UsuarioLogado();
        userSession.setToken(token);
        userSession.setUsuario(converter.toDTO(usuario));

        return userSession;

    } catch (Exception e) {
        throw new RuntimeException("Error en el inicio de sesión: " + e.getMessage());
    }
}
```

Figura 6-6. Método Login API

6.2.3 Perfil

La pantalla de **perfil de usuario** constituye un espacio central de la aplicación, ya que concentra tanto la información personal del usuario como el acceso a todos los datos relacionados con su actividad dentro del sistema. Desde aquí, el usuario puede consultar sus datos básicos. Y acceder a cuatro secciones clave, mis pueblos, mis eventos, mis incidencias y favoritos.

En el caso de un **usuario invitado**, el perfil se representa una versión reducida en la que únicamente aparece un botón para **iniciar sesión**.

La interfaz se ha diseñado utilizando un **ViewPager2** combinando con un **TabLayout**, lo que permite implementar el sistema de pestañas horizontales en la parte superior del perfil. Cada pestaña corresponde a un fragmento diferente.

- **"Mis pueblos"**, donde se listan los ayuntamientos a los que pertenece el usuario.
- **"Mis eventos"**, que muestra las reservas realizadas por el usuario.
- **"Mis incidencias"**, muestra las incidencias creadas desde la cuenta del usuario.
- **"Favoritos"**, donde se almacenan los eventos marcados como destacados.

Este patrón de navegación ofrece una visualización limpia y organizada, permitiendo al usuario moverse de manera fluida entre distintos apartados sin abandonar la pantalla de perfil. Internamente, cada pestaña está gestionada por un adaptador personalizado que enlaza los **Fragments** con **ViewPager2** y el **TabLayout**.

Además de la navegación por pestañas, es la pantalla de perfil incorporado funcionalidades adicionales:

- **Editar perfil**, mediante un botón que redirige a un fragmento específico donde el usuario puede actualizar su información personal. Este flujo se implementa mediante una transición de fragmentos que sustituye temporalmente la vista al perfil por la del editor, conservando la posibilidad de volver atrás gracias al **back stack**.
- **Cerrar sesión (Logout),** accesible también desde esta pantalla, al pulsar el botón de cerrar sesión, el sistema elimina el **token JWT** y limpia la sesión guardada del dispositivo. A continuación, redirige al usuario la pantalla de inicio de sesión.

Desde el punto de vista técnico, la **foto de perfil** se carga a partir de un campo formato **Base64** almacenado en la **bd**. Si el usuario no tiene imagen, se utiliza un recurso predeterminado.

Para la gestión de sesión y de datos de usuario se utiliza la clase auxiliar **SessionManager**, que permite acceder en cualquier momento a la información de la cuenta en uso.

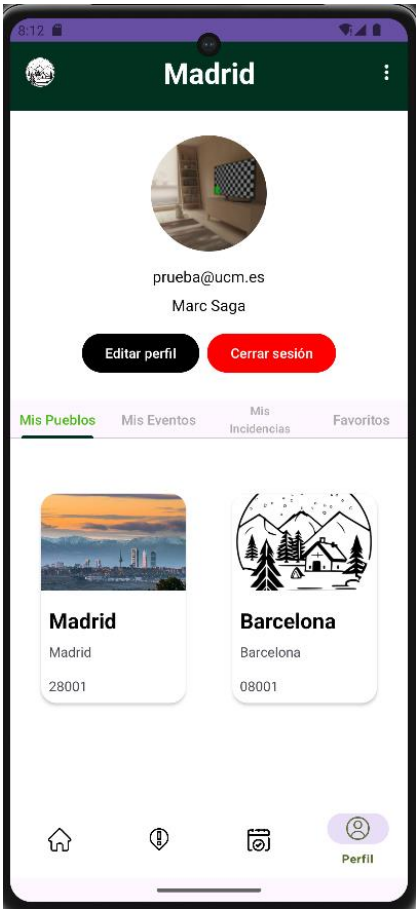

Figura 6-7. Pantalla perfil de Usuario

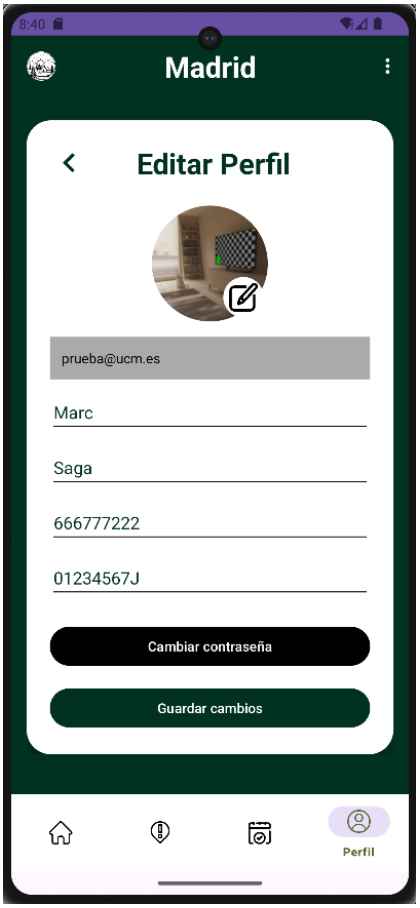

Figura 6-8. Pantalla editar perfil

En cuanto a la pestaña mis eventos, el usuario registrado puede consultar todas las reservas realizadas en los distintos pueblos a los que pertenece. Al seleccionar una de ellas en el listado, selecciona la pantalla. No se accede a la pantalla de detalle de reserva, donde se muestra información completa.

- Nombre del evento, fecha y lugar.
- Número de entradas reservadas.
- Estado de la reserva (confirmada, pendiente de pago, cancelada etc.).
- Información del pago en caso de que haya sido realizado a través de la pasarela de **Stripe** (monto, método de pago, fecha).

Desde esta misma vista, el usuario puede **cancelar la reserva**, lo que a actualiza el estado tanto en la base de datos como en el aforo respectivo del elemento correspondiente. Esta funcionalidad aporta transparencia y control al usuario sobre su participación en los eventos.

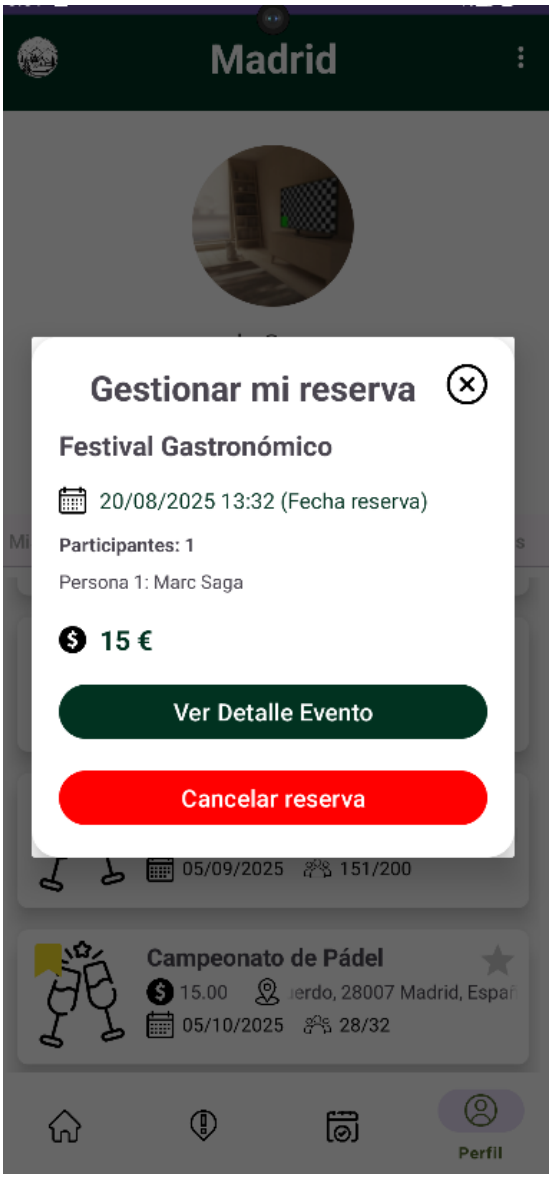

Figura 6-9. Pantalla gestión de reserva

6.3 Funcionalidad: Eventos

La sección de **"Eventos"** permite a los usuarios descubrir y participar en la agenda y actividades del ayuntamiento/pueblo activo en sesión. La vista principal presenta un listado de cronológico de actividades con un **buscador** en la parte superior, un botón de selección de fecha, y un acceso a un panel de filtros. La lista se alimenta mediante una llamada a la **API** del **backend** (mediante una llamada **Retrofit**) que devuelve los eventos del municipio seleccionado tras recibir la respuesta. Los datos se ordenan por fecha y hora para mostrar primero los próximos eventos. Este enfoque evita llamadas a la red innecesarias durante la interacción se carga una vez y toda la refinación. Se realiza la memoria garantizando la fluidez.

Figura 6-10. Pantalla de eventos

El **buscador** funciona en tiempo real, es decir, a medida que el usuario escribe, se actualiza el listado con coincidencias por título o palabras clave. El **selector de fecha** abre un date **DatePickerDialog** localizado en español. Al confirmar, la lista se restringe a la fecha elegida (o el criterio temporal definido), combinándose con el texto del buscador. El **panel de filtros** se presenta como un diálogo inferior **"bottom sheet"** que permite establecer un **precio máximo**, el **tipo del evento** (normal o competición) y el **estado temporal** (futuros o terminados). Al aplicar, la lógica central con combina **texto** + **fecha** + **tipo** + **precio** + **estado** y devuelve el conjunto final. Existe, además, un botón de **restablecer** que limpia todos los criterios y devuelve el listado inicial.

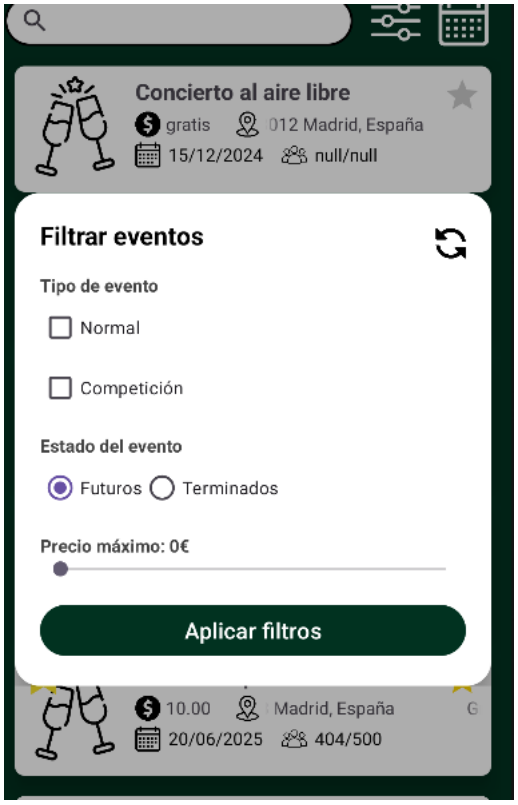

Figura 6-11. Filtros de eventos

Figura 6-12. Filtrar por fecha

Si el usuario está autenticado puede marcar o desmarcar el botón de **favorito** del propio evento, y a posteriori el cliente actualiza el icono y sincroniza el estado con el servidor. En caso de que el usuario no haya iniciado sesión y pulse el botón de **favorito** se informa de que para marcar un evento como favorito es necesaria la autenticación del propio usuario.

6.3.1 Gestión de reservas

Al seleccionar una tarjeta se accede al **detalle del evento**, un fragmento que muestra la información completa: título, descripción, fecha, hora, ubicación, precio o indicación de gratuidad, aforo total y plazas disponibles. Desde esta pantalla, se inicia el flujo de **reserva**.

El proceso de **reserva** de eventos constituye el núcleo funcional de la aplicación, ya que permite a los usuarios confirmar su asistencia a las actividades publicadas por cada ayuntamiento. La lógica en implementada contempla distintos escenarios en función del **tipo de evento** (gratuito o de pago) y el **rol del usuario** (registrado o invitado).

6.3.1.1 Eventos gratuitos

En el caso de eventos gratuitos, la **reserva** se realiza de manera directa y sin necesidad de pasar por ningún flujo de pago. Tanto los **usuarios registrados** como los **invitados** pueden seleccionar el número de entradas deseado en el diálogo de reserva y, al confirmar, la aplicación, envía la petición al **backend** para crear la reserva y guardar los datos en la **bd**.

- Si el usuario es registrado, la reserva queda vinculada a su perfil y puede consultarse desde en la sección de **"Mis eventos"** dentro del perfil.
- Si el usuario es invitado, la reserva queda registrada asociada al correo electrónico proporcionado y el sistema envía un **correo de confirmación** con el justificante de la reserva.

Hoy ambos casos. La reserva se confirma inmediatamente, el aforo del evento se actualiza y el usuario recibe la notificación correspondiente.

Figura 6-13. Reserva evento gratuito (invitado)

Figura 6-14. Reserva confirmada (usuario registrado)

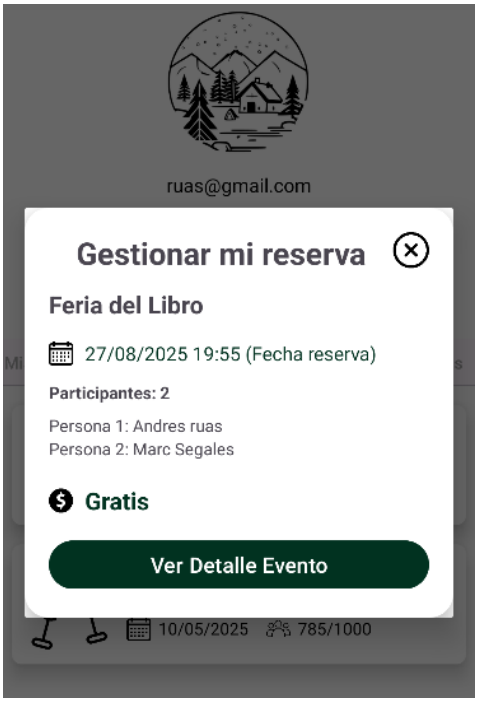

Figura 6-15. Reserva evento gratis (mis reservas)

6.3.1.2 Eventos de pago

Para los eventos que requieren un pago, la aplicación ofrece dos modalidades: **pago en efectivo** o **pago online** mediante pasarela de pago **Stripe**.

- **Pago en efectivo:** El usuario registrado o invitado selecciona la cantidad de entradas y confirma la reserva. En este caso, El sistema crea la reserva con el estado **"pendiente de pago"**, de modo que en la plaza queda bloqueada en el aforo. Pero aún no se ha validado el pago. Posteriormente, el usuario de realizar el abono en persona, por ejemplo, en el ayuntamiento o en el punto habilitado. Una vez verificado el pago, también te puede actualizar el estado de la reserva a **"confirmada"**. Este enfoque permite mantener una trazabilidad de reservas incluso cuando no se utiliza la pasarela digital.

En este caso concreto de pago, los usuarios registrados tienen la ventaja de poder cancelar la reserva 48h antes del evento. Esto lo podrían hacer accediendo a la

pestaña de "Mis eventos", en el perfil del usuario pulsando en el propio evento y entrando al detalle de la gestión de la reserva.

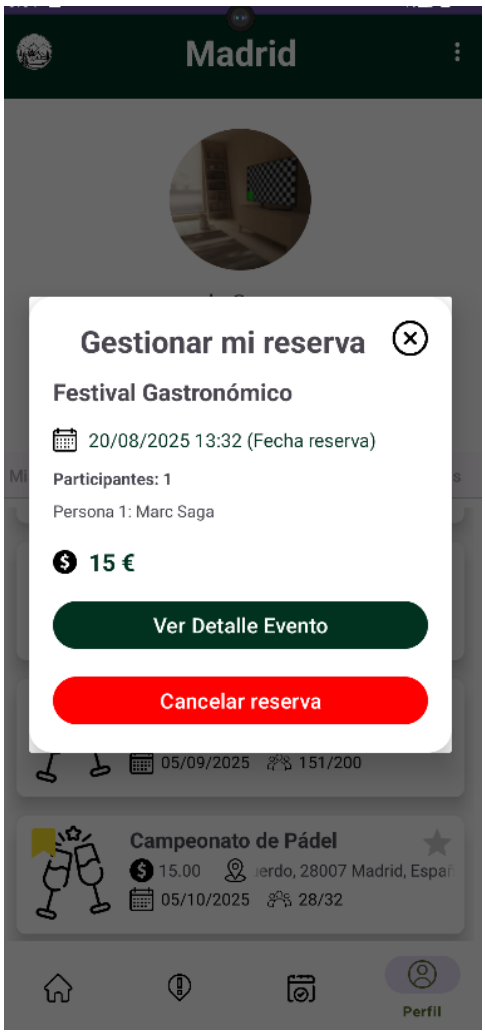

Figura 6-17. Gestionar reserva

Figura 6-16. Reserva pendiente de pago

- **Pago online (Stripe):** En esta modalidad, tras seleccionar el número de entradas, el usuario inicia Hoy, 1678. el flujo de pago integrado por **Stripe.**

1. En el **backend** se crea una **reserva en estado "Pendiente de pago"** y se solicita **Stripe**, la generación de una integración de pago.
2. En el cliente **Android** se abre la pasarela segura de **Stripe**, donde el usuario introduce los datos de su tarjeta. **Stripe** valida los datos de forma cifrada y, si el cobro se autoriza, notifica al **backend**.

3. El **backend** actualiza la reserva a estado **"Confirmada"**, guarda el registro de pago en la tabla **"pagos_reserva"** de la **bd** y genera el justificante de pago.

La entrega de justificante varía según el rol del usuario:

- Para un **usuario registrado**, la confirmación y el justificante quedan almacenados en la sección mis eventos dentro del perfil del usuario.
- Para **un usuario invitado**, la confirmación se envía por correo electrónico con todos los datos de la reserva.

En caso de que el pago falle (tarjeta denegada, **autenticación 3D, Secure** no superada, etc.) la reserva se elimina al momento y el sistema informa al usuario del error. Si el pago queda incompleto, la reserva se mantiene en estado **"Pendiente de pago"** hasta que se complete o caduque, evitando. Inconsistencias.

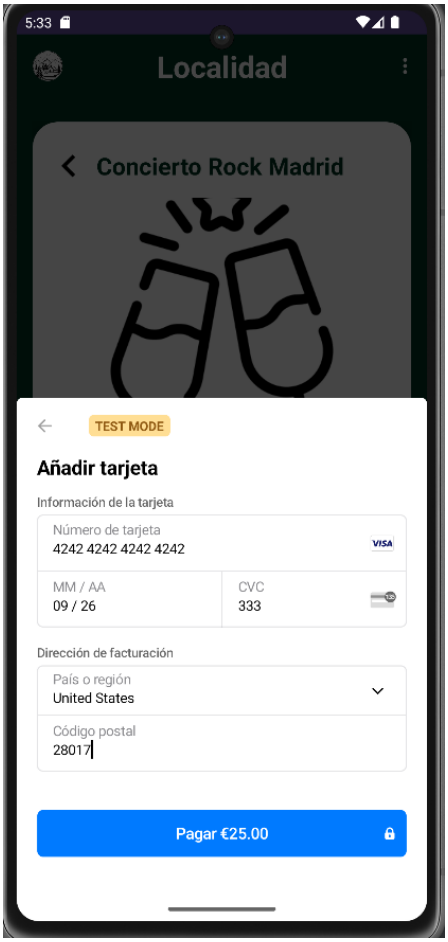

Figura 6-18. Pasarela de pago

Figura 6-19. Reserva confirmada

6.4 Funcionalidad: Incidencias

La funcionalidad de **incidencias** permite a todos los usuarios comunicar los problemas detectados en su municipio de forma rápida y estructurada, quedando todos los reportes vinculados al **ayuntamiento/pueblo activo en sesión**. Este módulo se ha concebido como un canal de comunicación digital entre los vecinos y el consistorio, reemplazando a procesos tradicionales basados en llamadas o visitas presenciales.

La pantalla de ciencia se compone de un **listado vertical** que muestra todas las incidencias del ayuntamiento seleccionado. Cada tarjeta del listado incluye el título de la incidencia, su estado actual (Pendiente, en proceso o cerrada), la fecha de creación y la imagen asociada. El listado está implementado con un **RecyclerView**, lo que permite manejar de forma eficiente incluso un gran número de registros.

En la parte superior derecha de la pantalla se encuentran dos botones flotantes **(FAB):**

- **Añadir incidencia:** Abre el flujo de creación de una nueva incidencia
- **Filtros:** Abre un panel inferior **(bottom sheet)** con opciones para restringir la vista según distintos criterios.

El panel de filtros ofrece dos grandes categorías:

- **Estado:** Permite seleccionar si se desean ver incidencias pendientes, en proceso o cerradas (de manera individual o combinada).
- **Autoría:** Se puede filtrar por incidencias creadas por usuarios registrados, por invitados o por todos los autores.

El resultado de aplicar los filtros se refleja inmediatamente en el listado, sin necesidad de volver a cargar datos del servidor, ya que todos los cálculos se realizan sobre la lista completa almacenada en memoria **(all Incidents)**. Un botón adicional de **"Restablecer"** devuelve la vista a la configuración inicial.

Figura 6-21. Pantalla de incidencias

Figura 6-22. Filtro de incidencias

```
private void fetchIncidencias() {  1 usage
    int idAyto = SessionManager.getInstance()
            .getPuebloEnSession().getClaveUnicaAyto();
    ApiClient.getIncidenciaService()  IncidenciaService
            .listarIncidencias(idAyto)  Call<List<...>>
            .enqueue(new Callback<List<IncidenciaResponseDTO>>() {
                @Override public void onResponse(Call<List<IncidenciaResponseDTO>> call,
                                                 Response<List<IncidenciaResponseDTO>> resp) {
                    if (resp.isSuccessful() && resp.body()!=null) {
                        allIncidents.clear();
                        allIncidents.addAll(resp.body());
                        // primero mostrarlas todas
                        lista.clear();
                        lista.addAll(allIncidents);
                        adapter.notifyDataSetChanged();
                    } else {
                        Toast.makeText(getContext(),
                                text: "Error: " + resp.code(),
                                Toast.LENGTH_SHORT).show();
                    }
                }
                @Override public void onFailure(Call<List<IncidenciaResponseDTO>> call, Throwable t) {
                    Toast.makeText(getContext(),
                            text: "Fallo de red",
                            Toast.LENGTH_SHORT).show();
                }
            });
}
```

Figura 6-23. Método de carga de incidencias

6.4.1 Creación de incidencias

El proceso de creación de **incidencias** se ha implementado a través de un **diálogo modal (AddIncidentDialogFragment**) que concentra en una misma vista todos los pasos necesarios para reportar un problema en el municipio. De esta forma, el usuario no abandona la pantalla de incidencias y la interacción resulta más rápida e intuitiva.

El diálogo solicita 3 bloques de información principales:

- **Título y descripción:** Se añade un título a la incidencia al igual que un resumen corto de lo que ocurre en la misma. Estos son datos obligatorios y capturados mediante campos de texto.

- **Ubicación:** Es un dato obligatorio que aportar en la incidencia. Se selecciona mediante un botón que abre un **MapPickerDialogFragment**. Al elegir la localización en el mapa, se devuelven al diálogo a las coordenadas exactas, latitud y longitud.

- **Dirección:** Es opcional, de este modo, la incidencia queda asociada tanto a las coordenadas para la gestión técnica como a un texto comprensible para el personal del ayuntamiento.

- **Fotografías:** El usuario aporta evidencias gráficas de la incidencia. Para ello dispone de 2 accesos.

- **Botón de Cámara:** Abre la Cámara del dispositivo previa comprobación y solicitud de permisos de **CAMERA** en tiempo de ejecución. Las fotos se almacenan en un archivo temporal mediante un **FileProvider** y su URI se añade a la lista de imágenes adjuntas.

- **Botón de galería:** Permite seleccionar una o varias imágenes desde el almacenamiento de dispositivo utilizando **Storage Access Framework.**

Cada fotografía seleccionada se muestra inmediatamente en el contenedor y central del diálogo como miniatura gracias a la librería **Glide**, de forma que el usuario puede comprobar que imágenes se enviarán antes de confirmar.

Figura 6-25. Creación de incidencia

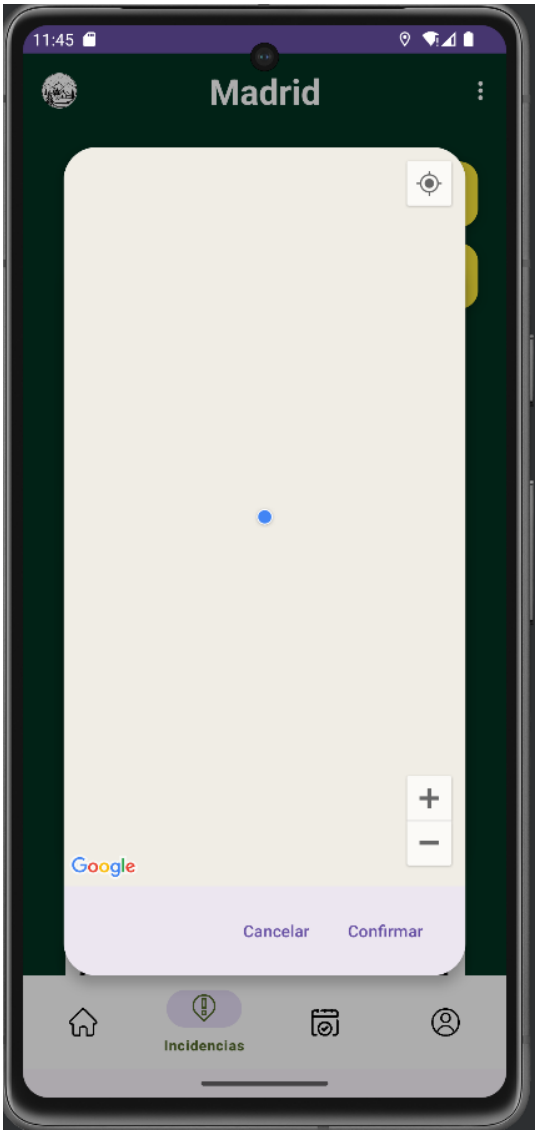

Figura 6-24. Selección de ubicación

Al pulsar en **"Guardar",** el diálogo entrega al **"Incidents_Fragment"** todos los datos recogidos: Título, descripción, dirección, coordenadas, lista de **URIs** de imágenes. El fragmento convierte las imágenes a **Base64** y construye el **IncidenciaRequestDTO** que se envía al **backend** a través del **Retrofit**.

Aquí se establece una diferencia importante según el rol del usuario:

- Si el usuario está **registrado**, el envío incluye la cabecera **token** de sesión lo que permite, identificar el autor y asociar la incidencia a su cuenta personal. En el listado y en el detalle aparecerá su nombre como autor y en el perfil dispondrá de la pestaña de

mis incidencias para consultarlas y gestionarlas. A esto debemos añadirle, que el usuario registrado tiene la ventaja de que mientras la incidencia creada esté en estado pendiente tiene la **posibilidad de editarla** a su gusto o incluso de **eliminarla**.

- Si el usuario es **invitado**, la petición se envía sin **token** y el **backend** marca al autor como **invitado**. La incidencia se crea igualmente, pero sin historial personal asociado y sin posibilidad de edición posterior.

En ambos casos, una vez confirmada la creación en el servidor, la aplicación añade una nueva incidencia al inicio del listado. Y desplaza la vista para que el usuario vea su reporte de inmediato. Reforzando la sensación. De que la acción se ha completado correctamente.

```java
public IncidenciaResponseDTO crearIncidencia(String authorizationHeader,
                                             IncidenciaRequestDTO dto) {
    //Extraer idUsuario si hay token
    Integer idUsuario = null;
    if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
        idUsuario = jwtUtil.getUsuarioFromToken(authorizationHeader.substring( beginIndex: 7))
                .map( Usuario u -> u.getId()) Optional<Integer>
                .orElse( other: null);
    }
    if(idUsuario == null){
        idUsuario = 0;
    }

    //Construir y salvar Incidencia
    Timestamp fecha = new Timestamp(System.currentTimeMillis());
    Incidencia inc = Incidencia.builder()
            .idAyto(dto.getIdAyto())
            .idUsuario(idUsuario)
            .Nombre(dto.getNombre())
            .Descripcion(dto.getDescripcion())
            .Latitud(dto.getLatitud())
            .Longitud(dto.getLongitud())
            .estado(Estado.Pendiente)
            .direccion(dto.getDireccion())
            .FechaCreacion(fecha)
            .build();
    Incidencia guardada = incidenciaRepository.save(inc);

    //Guardar cada foto en blob y acumular lista de base64
    List<String> fotos = new ArrayList<>();
     if (dto.getFotosBase64() != null) {
         for (String b64 : dto.getFotosBase64()) {
             ImagenIncidencia img = ImagenIncidencia.builder()
                     .incidencia(guardada)
                     .base64(b64)
                     .build();
             imagenRepo.save(img);
             fotos.add(b64);
         }
     }

     IncidenciaResponseDTO out = mapToDto(guardada);
     out.setFotosBase64(fotos);
     return out;
}
```

Figura 6-26. Método crear incidencia desde el backend

6.4.2 Detalle de incidencias

Al seleccionar una incidencia en el listado, la aplicación abre una pantalla especifica de **detalle** implementada con **"IncidentDetail_Fragment".** Esta vista tiene como objetivo mostrar toda la información completa de un reporte concreto y, al mismo tiempo ofrecer al doctor las acciones de gestión disponible.

La interfaz del **detalle** se construye a partir de los datos que recibe el fragmento: Título, descripción, estado actual, fecha de creación. Última actualización, ubicación. autoría y fotografías asociadas. La información se muestra en un formato legible y visual, con especial relevancia para el **estado** de la incidencia, de manera que el usuario pueda comprobar rápidamente si está **"Pendiente"**, **"En proceso"** o ya **"Cerrada"**. En el caso de las **fotografías**, se cargan en un carrusel o listado gráfico, permitiendo ampliarlas para ver los detalles concretos.

La sección de ubicación ha aprovechado las coordenadas geográficas guardadas en el momento de la creación. Usando la latitud y longitud, se realiza un mapa en miniatura mediante **Google Maps,** con un marcador en el punto exacto, lo que facilita al ayuntamiento y a otros usuarios localizar el lugar físico del problema.

En cuanto a las acciones disponibles, la pantalla el detalle de diferencia claramente entre usuarios registrados y usuarios invitados. Así como entre los distintos Estados de la incidencia.

- **Usuarios registrados:** Si la incidencia fue creada por el propio usuario y todavía se encuentra en estado **"Pendiente"**, se habilitan dos opciones de gestión: **Editar y eliminar.**

- La **edición** abre un formulario similar al de creación precargado con los datos actuales, permitiendo modificar cualquier dato, incluso hasta añadir o eliminar fotografías antes de volver confirmar. El **backend** valida el cambio y actualiza la **incidencia** en la **bd**, tras la cual se refresca el listado y la vista al detalle con la información corregida.

- La **eliminación** envía una petición de borrado al servidor. Una vez confirmada la **incidencia**, desaparece tanto del detalle como el listado principal y el usuario recibe un mensaje de confirmación.

Estas opciones desaparecen en cuanto a la **incidencia** pasa a estado **"En proceso"** o **"Cerrada"**, ya que a partir de ese momento es el ayuntamiento que gestiona su evolución y resolución.

- **Usuarios invitados:** Al no disponer de un perfil persistente, los invitados pueden consultar el detalle completo de cualquier incidencia del municipio, pero **no** pueden realizar acciones ni de **edición** ni de **eliminación**, ni siquiera En aquellas que ellos mismos hayan creado.

Figura 6-28. Pantalla detalle de incidencia

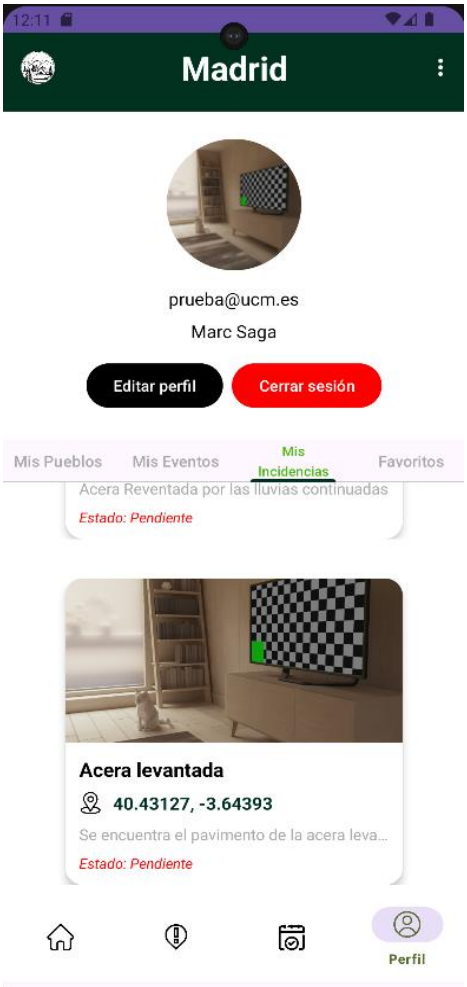

Figura 6-27. Mis incidencias

6.5 Pantalla inicial y perfil del municipio

La experiencia de usuario arranca en una pantalla de inicio que ofrece una vista condensa del ayuntamiento/pueblo activo en sesión: previsión del tiempo, próximos eventos e incidencias recientes. Esta pantalla se sustenta sobre un **Fragment** principal **(Home_Fragment_old)** y se integra en la actividad contenedora **drawer_main_activity,** que resuelve la navegación con **menú lateral (Drawer + NavigationView)** y **barra inferior (BottomNavigationView).**

En la parte superior del **toolbar** se muestra el nombre del municipio activo, sincronizado desde **"SessionManager"** al restaurar la sesión. La cabecera del menú lateral (**header** del **NavigationView**) actúa como tarjeta del ayuntamiento: incluye el **escudo** o **logotipo**, el **nombre del pueblo**, el **correo de contacto** y otra información básica (por ejemplo, el código postal), cargada dinámicamente con **"CargaDatosAyuntamiento()".** Este bloque refuerza al usuario que todas las operaciones que haga (consultar eventos, crear incidencias, reservar, etc.) están **contextualizadas en el municipio activo** en sesión; en cualquier momento puede salir de ese contexto con la opción **"Salir el pueblo"**, que limpia la sesión y redirige a la sección de ayuntamiento (**usuarios registrados**) o a la búsqueda de **auyuntamiento** (**invitados**).

6.5.1 Pantalla de inicio: tiempo, eventos e incidencias

La pantalla de inicio **"fragment_home"** compone tres carruseles horizontales independientes (tres **RecyclerView** con **Layout manager** horizontal) que ponen en primer plano a información más útil del día a día:

1. **Tiempo:** Se consulta el tiempo del municipio activo a través de **"WeatherApiClient.fetchWeather(city)",** donde **city** se obtiene de **Ayto** guardado en **"SessioManager"**. El adaptador **"TiemoAdapter"** formatea la información meteorológica y la muestra como tarjetas horizontales (máx./min., estado, el día y hora, etc.). Un clic sobre el bloque abre el **módulo del Tiempo "tiempoFragment",** para ver el pronóstico con mayor detalle. Esta decisión de arquitectura (pedido desde el **front** a la **API**

meteorológica OpenweatherMap) elimina latencia innecesaria y mejora la sensación de respuesta.

```
protected String doInBackground(Void... voids) {
        String ciudad = SessionManager
                .getInstance()
                .getPuebloEnSession()
                .getLocalidad();
        URL url = new URL(
                spec: "https://api.openweathermap.org/data/2.5/weather" +
                        "?q=" + ciudad +
                        "&units=metric" +
                        "&appid=" + API_KEY
        );
        connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        InputStream  is = connection.getInputStream();
        BufferedReader reader = new BufferedReader(
                new InputStreamReader(is, charsetName: "UTF-8")
        );
        StringBuilder sb = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
            sb.append(line);
        }
        reader.close();
        return sb.toString();
    } catch (Exception e) {
        return null;
    } finally {
        if (connection != null) {
            connection.disconnect();
        }
    }
```

Figura 6-29. Llamada a la API del tiempo

2. **Próximos eventos:** Se recuperarán todos los eventos del ayuntamiento mediante **"ApiClient.getEventService().getEventosPueblo(idAyto);"** tras la llamada, se ordenan por fecha con **EventoUtils.ordenarEventos()** y se filtran los **eventos futuros** (la utilidad **"obtenerProximosEventos()")** para quedarse con los más cercanos en el tiempo. El carrusel muestra hasta **cuatro** eventos a modo de escaparate. Cada tarjeta incluye información sintética (título, fecha, precio si aplica) y dos interacciones: apertura del detalle del evento, al pulsar en él, y **Favorito**, al pulsar en el botón de la **estrella**.

```
private void cargarProximosEventos() {  1 usage
    SessionManager session = SessionManager.getInstance();
    Integer idPueblo = session.getPuebloEnSession().getClaveUnicaAyto();

    if (idPueblo == null) {
        Toast.makeText(getContext(),  text: "No se ha seleccionado ningún pueblo", Toast.LENGTH_SHORT).show();
        return;
    }

    Call<List<Event>> call = ApiClient.getEventService().getEventosPueblo(idPueblo);
    call.enqueue(new Callback<List<Event>>() {
        @Override
        public void onResponse(@NonNull Call<List<Event>> call, @NonNull Response<List<Event>> response) {
            if (response.isSuccessful() && response.body() != null) {

                List<Event> eventosPueblo = response.body();
                EventoUtils.ordenarEventos(eventosPueblo);
                session.setEventosPueblo(eventosPueblo);

                // Filtrar los 4 eventos más cercanos
                List<Event> eventosFuturos = obtenerProximosEventos(eventosPueblo);
                List<Event> proximosEventos = eventosFuturos.stream()
                        .limit( l: 4)
                        .collect(Collectors.toList());

                actualizarRecyclerView(proximosEventos);

            } else {
                Toast.makeText(getContext(),  text: "Error al obtener eventos", Toast.LENGTH_SHORT).show();
                Log.e( tag: "Events_Fragment",  msg: "Error en la respuesta: " + response.code());
            }
        }
```

Figura 6-30. Método de carga eventos desde API

3. **Incidencias recientes:** Para dar visibilidad a la participación ciudadana, se cargan las incidencias del municipio con **"listarIncidencias(idAyto)"** y se ordenan por fecha de creación en sentido descendente, mostrando solo las tres más recientes. El adaptador **"IncidenciaAdapter"** presenta cada incidencia con su título, estado y miniatura (si existe). Igual que en eventos, al pulsar una tarjeta se abre el **detalle de la incidencia "IncidentDetail_Fragment",** donde el usuario puede seguir su evolución o, si es el autor y está en estado **"Pendiente"**, editarla o eliminarla**.**

```
private void cargarUltimasIncidencias(){  1 usage
    int idAyto = SessionManager.getInstance()
            .getPuebloEnSession().getClaveUnicaAyto();
    ApiClient.getIncidenciaService()  IncidenciaService
            .listarIncidencias(idAyto)  Call<List<…>>
            .enqueue(new Callback<List<IncidenciaResponseDTO>>() {
                @Override public void onResponse(Call<List<IncidenciaResponseDTO>> call,
                                                 Response<List<IncidenciaResponseDTO>> resp) {
                    if (resp.isSuccessful() && resp.body() != null) {
                        //Copiamos la lista que viene del servidor
                        List<IncidenciaResponseDTO> todas = new ArrayList<>(resp.body());
                        //Ordenamos por fecha descendente
                        Collections.sort(todas, (a, b) ->
                                b.getFechaCreacion().compareTo(a.getFechaCreacion())
                        );
                        // Tomamos solo las 3 primeras (o menos si no hay)
                        List<IncidenciaResponseDTO> top3;
                        if (todas.size() > 3) {
                            top3 = new ArrayList<>(todas.subList( i: 0,  i1: 3));
                        } else {
                            top3 = new ArrayList<>(todas);
                        }
                        //Actualizamos el adapter
                        adapter.setItems(top3);
                    } else {
                        Toast.makeText(getContext(),
                                text: "Error cargando incidencias: " + resp.code(),
                                Toast.LENGTH_SHORT).show();
                    }
                }
```

Figura 6-31. Método de carga de incidencias desde API

Esta organización en **tres carruseles horizontales** permite consumir la información "de un vistazo" sin saturar la pantalla ni exigir desplazamientos largos. A nivel técnico, cada bloque mantiene su propio adaptador y ciclo de vida, lo que facilita **actualizaciones parciales** (por ejemplo, refrescar solo el tiempo o solo los eventos) y reduce acoplamientos. Cuando el **backend** responde, los adaptadores actualizan su lista (**actualizarLista**, **setItems**) y notifican los cambios para redibujar con eficiencia.

Figura 6-32. Pantalla de inicio

6.5.2 Navegación: Drawer e Información del municipio

La actividad **drawer_main_activity** orquesta la navegación con un enfoque doble:

- **Menú lateral (Drawer)**: agrupa **secciones de contexto municipal** que no necesariamente encajan en la navegación del día a día. Entre ellas, "Información general" y "Tiempo" están accesibles desde el **NavigationView** y cargan sus fragmentos (**InfoGeneralFragment**, **tiempoFragment**) con **loadFragment**(...). También aquí reside la opción **"Salir del pueblo"**, que cierra la sesión contextual del ayuntamiento y redirige al flujo adecuado (selección de ayuntamiento para registrados, búsqueda para invitados). Este menú lateral integra además la **cabecera con la identidad del ayuntamiento** (escudo, nombre, contacto), reforzando el marco de trabajo.

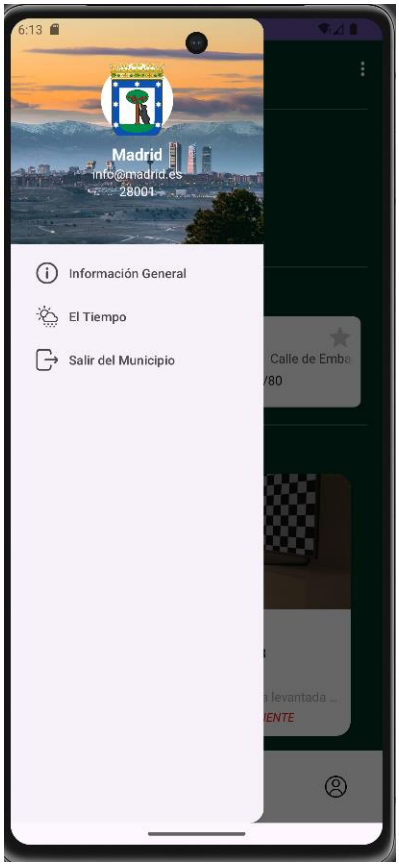

Figura 6-33. Menú de navegación lateral

- **Barra inferior (BottomNavigationView)**: concentra la **navegación operativa** cotidiana: **Inicio**, **Eventos**, **Incidencias** y **Perfil**. La selección de cada pestaña reemplaza el contenedor principal **Frame_Layout_NavView2** por el fragmento correspondiente (**homeFragment**, **eventsFragment**, **incidentsFragment**, **perfilFragment**). Esta separación consigue que el usuario rote de forma natural entre "ver qué hay hoy", "buscar o filtrar eventos", "reportar/consultar incidencias" y "gestionar su cuenta" sin perder el contexto del municipio activo.

En el arranque, la actividad recupera o reconstruye la sesión con **recuperarSesionDesdeAPI**(**userId**, **pebloId**): se solicita el usuario y el ayuntamiento, se invoca a servicios auxiliares para **poblar el estado local** (favoritos de eventos, reservas del usuario, incidencias del usuario), y se actualiza el ***toolbar*** con el nombre del pueblo. En caso de **invitado**, se salta la carga de datos personales y se habilita directamente la navegación sobre el municipio elegido.

Figura 6-34. Menú inferior de navegación

- **Módulo de Tiempo (detalle): e**l **detalle del tiempo** (**tiempoFragment**) complementa el carrusel de inicio mostrando la **previsión ampliada** para el municipio: valores actuales, evolución de las próximas horas y tendencia de varios días. Aunque los datos se obtienen con la misma fuente que el inicio, aquí se presentan con **mayor profundidad** (más campos y/o mayor rango temporal). El acceso al fragmento se hace desde el propio carrusel de inicio, manteniendo la coherencia de navegación (volver restituye la pantalla principal sin recargas innecesarias).

Figura 6-35. Módulo de tiempo

- **Información general del municipio:** El **módulo de Información general** (**InfoGeneralFragment**) centraliza los contenidos estáticos o semi estáticos del ayuntamiento: **descripción/historia**, **datos de contacto** (correo, dirección), y cualquier bloque informativo relevante que el consistorio desee publicar. El acceso se ofrece desde el menú lateral (**Drawer**), subrayando que se trata de contenido **identitario** del municipio. En la pantalla de inicio, la aplicación puede incluir "resúmenes" o llamadas a esta información (p. ej., una introducción recortada con enlace a la ficha completa),

reforzando el vínculo entre lo operativo (eventos, incidencias) y lo informativo (quiénes somos, cómo contactarnos).

Figura 6-36. Módulo información general

6.6 Registro de Admin_Ayto y de Ayuntamiento

6.6.1 Solicitud de creación de Admin_Ayto y validación

Una de las funcionalidades diferentes del proyecto es la posibilidad de que los representantes municipales puedan crear y gestionar su propio ayuntamiento dentro de la aplicación. Para ello se introduce la figura del usuario con rol **Admin_Ayto**. Que equivale al alcalde o representación autorizada al municipio. Este actor no puede registrarse de forma directa como el resto de los usuarios, sino que debe enviar una solicitud formal al sistema, la cual será válida manualmente por un **Super_Admin** antes de concederle acceso.

El flujo es el siguiente:

- El usuario interesado accede a la pantalla de **"solicitud de alta ayuntamiento"**, disponible desde la vista inicial de la aplicación.
- A continuación, se despliega un formulario en el que debe rellenar sus datos personales y los datos básicos de su municipio. Entre los campos obligatorios se incluyen:
    - **Nombre y apellidos** del solicitante.
    - **Email** (Que será también el identificador de acceso).
    - **DNI** como documento identificativo para validar la legitimidad de la solicitud.
    - **Teléfono de contacto**, para verificar al solicitante si fuera necesario.
    - **Nombre del pueblo/ayuntamiento** a dar de alta.
    - **Provincia a la que pertenece.**
    - **Código postal** del municipio.
    - **Motivo de la solicitud**, donde el usuario explica brevemente su rol o relación con el ayuntamiento y justifica el alta.

- Una vez completado el formulario, el solicitante pulsa el botón **"enviar solicitud"**, generándose un registro en la **bd** de la aplicación.
- Esta solicitud queda marcada con estado **"pendiente"** hasta que el usuario controle **Super_Admin** la revisa.

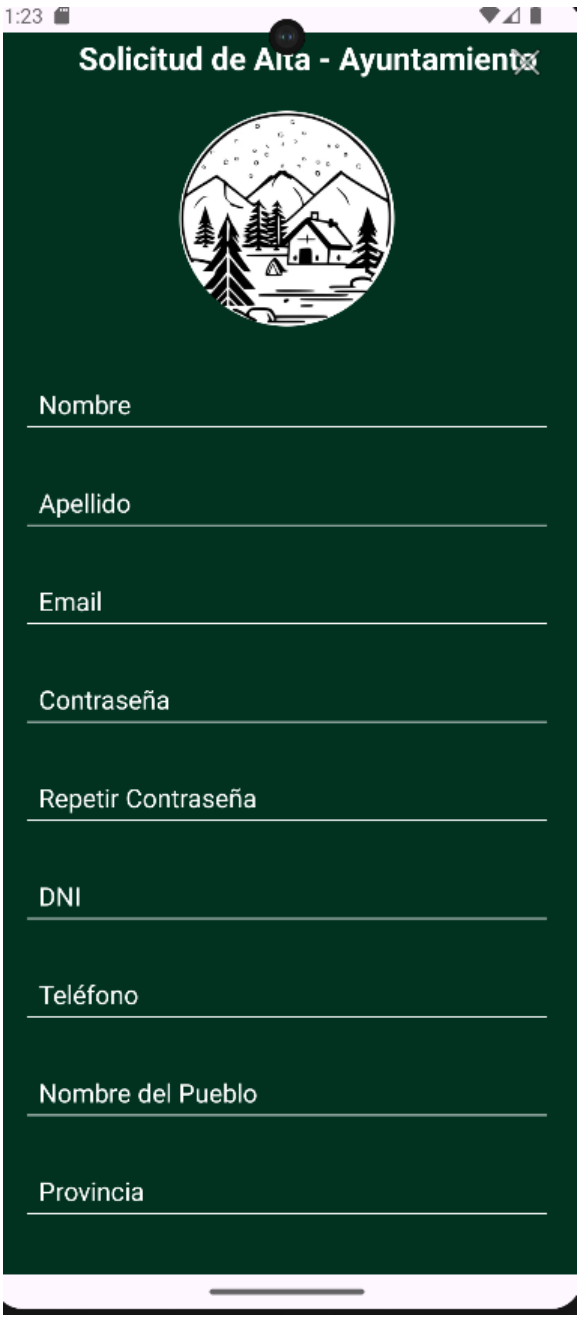

Figura 6-38. Pantalla Registro Admin_Ayto

Figura 6-37. Pantalla Registro Admin_Ayto2

El **Super_Admin** es el único con la capacidad de aprobar o rechazar solicitudes en caso de aprobación.

En caso de ser **aprobada**, el solicitante pasa a ser registrado automáticamente en la base de datos con el rol **Admin_Ayto**. Sin embargo, este paso no implica todavía que el municipio que se registraron la plataforma a partir de ese momento, el nuevo administrador debe iniciar sesión en la aplicación y completar el flujo de creación del ayuntamiento, en el que se solicita introducir la información oficial necesaria para configurar su pueblo dentro del sistema.

Este proceso incluye la carga de datos de contacto como teléfono, correo y dirección, información descriptiva o histórica, elementos gráficos como imágenes o escudo municipales y cualquier otra información complementaria que permita enriquecer la ficha al mundo.

Solo tras esta acción, el **ayuntamiento/pueblo** queda definitivamente creado en la aplicación y asociado a la cuenta del nuevo administrador, quien pasa a tener acceso completo a las funcionalidades avanzadas, creación y edición de eventos, gestión de reservas y aforos, actualización de incidencias reportadas por los vecinos o la publicación de avisos oficiales.

En caso contrario, si la solicitud es **rechazada**, esta se marca la **bd** con estado **"denegada"** y solicitante recibe la notificación informando de informándole de la resolución.

Este procedimiento refuerza la fiabilidad y trazabilidad del sistema, al mismo tiempo que delega en el propio administrador municipal la responsabilidad de crear y mantener la información oficial de su ayuntamiento. Así se quede librado el control centralizado el **Super_Admin** con la autonomía de cada **Admin_Ayto**, garantizando que la aplicación crezca de manera ordenada y segura en el tiempo.

Figura 6-39. Pantalla de solicitudes (Super_Admin)

6.6.2 Registro de Ayuntamiento/Pueblo

Una vez validada la solicitud de alta por parte del **Super_Admin**, el nuevo usuario con rol **Admin_Ayto** adquiere pleno acceso a la aplicación. Sin embargo, a diferencia de un usuario normal, al iniciar sesión por primera vez no accede directamente al menú principal, sino que es redirigido a una pantalla específica para **registrar su ayuntamiento**. Esta fase es clave, ya que permite dar de alta en el sistema al municipio que el usuario representará y gestionará dentro de la aplicación.

La pantalla de registro de ayuntamiento se presenta como un **formulario** detallado en el que deben introducirse todos los datos necesarios para la **identificación** y gestión de la **entidad**. Entre los campos obligatorios se encuentran el **nombre del ayuntamiento**, la **provincia**, el **código postal**, la **dirección de contacto**, el **correo electrónico oficial** y un **teléfono** de contacto. Estos datos garantizan que el consistorio quede correctamente referenciado en el sistema y pueda comunicarse de manera oficial con los ciudadanos a través de la **app**.

Además de la información básica, el formulario permite registrar datos complementarios como la **web** institucional, la **corporación municipal** (donde pueden figurar los representantes principales del ayuntamiento), así como otros apartados opcionales destinados a enriquecer la ficha del municipio: enlaces a **redes sociales** oficiales, el **horario de atención** al público o cualquier otro detalle relevante que el administrador considere necesario incluir.

Un aspecto diferenciador de esta pantalla es la posibilidad de **añadir imágenes representativas del pueblo**, lo que contribuye a personalizar la ficha del ayuntamiento y ofrecer a los usuarios una experiencia más cercana y visual. El botón de "Añadir imágenes" permite cargar fotografías directamente desde el dispositivo, que luego quedarán asociadas al perfil del municipio.

Figura 6-40. Pantalla de registro de Ayuntamiento

Figura 6-41. Pantalla de registro de Ayuntamiento2

Finalmente, una vez completado el formulario, el administrador debe pulsar el botón **"Registrar Ayuntamiento"**. Al hacerlo, toda la información introducida se envía al **backend** de la **app**, donde se crea el nuevo registro en la **bd** y queda vinculado al usuario administrador correspondiente. A partir de este momento, el ayuntamiento se considera dado de alta en la plataforma y el **Admin_Ayto** podrá acceder al resto de funcionalidades avanzadas, como la gestión de incidencias reportadas por los ciudadanos o la creación y publicación de eventos locales.

De este modo, la pantalla de registro no solo constituye un requisito inicial para habilitar la administración del municipio, sino que también supone el primer paso en la digitalización de la gestión local, sentando las bases para que la aplicación funcione como un verdadero punto de encuentro entre los ayuntamientos y los vecinos.

6.7 Gestión de estado incidencias

Una de las principales responsabilidades de los usuarios con rol **ADMIN_AYTO** es la gestión de las incidencias reportadas por los vecinos o invitados de la aplicación. Este módulo permite llevar un control estructurado de los problemas comunicados en el municipio, asegurando que cada incidencia avance en un flujo de estados hasta su resolución final.

El flujo de gestión contempla tres estados principales: **Pendiente**, **En proceso** y **Cerrada**.

- Una incidencia **Pendiente** representa un reporte recién creado que todavía no ha sido atendido por el consistorio o ayuntamiento.
- Al pasar a estado **En proceso**, se indica que el ayuntamiento ya está trabajando en la resolución de esta.
- Finalmente, al marcarla como **Cerrada**, se da por concluido el problema, quedando archivado en el sistema.

El cambio de estado puede realizarse desde dos puntos de la aplicación:

- **Desde el detalle de la incidencia:** Cuando el administrador accede a la pantalla de detalle de una incidencia concreta, dispone de un botón de **"Cambiar estado"**, como se observa en la primera captura. Al presionarlo, la aplicación actualiza la incidencia y la mueve automáticamente al estado siguiente en el flujo. De esta forma, si estaba en Pendiente, pasa a En proceso, y si estaba en En proceso, pasa a Cerrada. En este último caso, la incidencia queda bloqueada y no permite más cambios, ya que ha sido resuelta oficialmente.

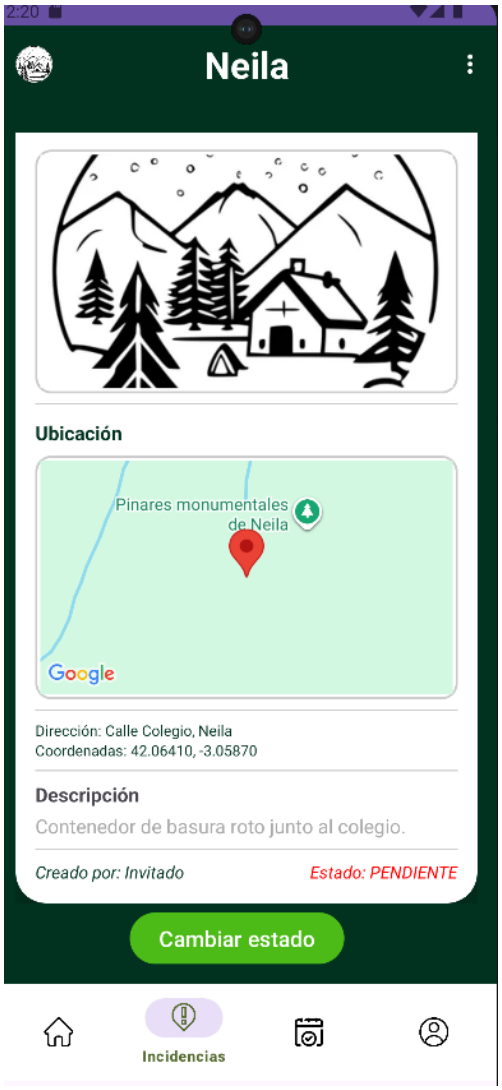

Figura 6-42. Cambio de estado desde detalle de la Incidencia.

- **Desde el perfil de administrador:** El perfil del usuario con rol **Admin_Ayto** está diseñado con varias pestañas (**tabs**) que clasifican las incidencias según su estado: Pendientes, En proceso y Cerradas. En cada **tab**, se listan todas las incidencias correspondientes. Dentro de cada tarjeta, se ofrece un botón contextual de **"Cambiar estado"**, con el mismo funcionamiento que en el detalle. Al actualizar el estado, la incidencia desaparece inmediatamente del listado en el que se encontraba y pasa al **tab** correspondiente a su nuevo estado. Esto permite al administrador gestionar incidencias de manera más ágil, sin necesidad de entrar en cada detalle individual.

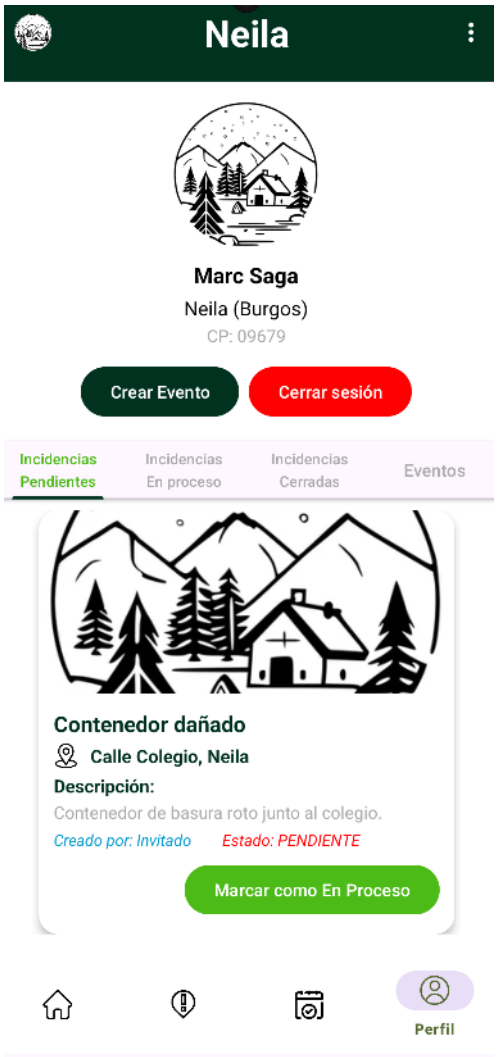

Figura 6-43. Cambio de estado desde el perfil de usuario.

Este sistema asegura una gestión **transparente** y **trazable**, donde los vecinos pueden ver cómo **evoluciona** el **estado** de los problemas que han reportado. Además, al limitar los cambios en las incidencias cerradas, se evita la alteración de datos históricos y se preserva la coherencia de la información.

6.8 Creación de eventos

La creación de eventos es una de las funcionalidades más relevantes para los usuarios con rol **ADMIN_AYTO**, ya que les permite dinamizar la vida del municipio anunciando actividades culturales, deportivas o sociales directamente desde la aplicación. Este proceso ha sido diseñado para ser intuitivo, guiado y seguro, de manera que incluso un usuario sin conocimientos técnicos pueda publicar un evento en cuestión de minutos.

El flujo de creación de eventos comienza en el **perfil del administrador**, donde se muestra el botón *Crear evento*. Al pulsarlo, la aplicación abre un primer fragmento en el que se solicita al administrador que seleccione el **tipo de evento** que desea registrar. En este paso inicial se elige entre tres opciones: evento normal, competición individual o competición grupal. Esta selección no es trivial, ya que determina qué campos serán obligatorios en el formulario posterior y cómo se mostrará el evento en su vista de detalle.

Una vez seleccionado el tipo de evento, el sistema redirige al administrador a un **formulario de creación "FormularioEventoFragment"**. Este formulario recoge en primer lugar los datos comunes a todos los eventos:

- Nombre y descripción.
- Ubicación (dirección manual y coordenadas geográficas).
- Fecha de inicio y fin.
- Hora del evento.
- Precio (permitiendo indicar si es gratuito).

- Capacidad máxima de asistentes.

Además, según el tipo de evento elegido, el formulario muestra secciones adicionales:

- En competiciones individuales se solicita el **número máximo de participantes**.
- En competiciones grupales se añaden los campos de **número de equipos** y **máximo de participantes por equipo**, que permiten organizar torneos o ligas locales.

El formulario está enriquecido con funcionalidades extra que aumentan su valor: el administrador puede **seleccionar la ubicación exacta en un mapa**, gracias al **MapPickerDialogFragment**, y esta acción almacena automáticamente latitud y longitud para mayor precisión. Asimismo, se ofrece la posibilidad de añadir **imágenes representativas del evento**, ya sea capturándolas directamente con la cámara o seleccionándolas desde la galería del dispositivo. La imagen elegida se muestra en una **previsualización inmediata** y posteriormente se convierte a formato **Base64**, de modo que puede enviarse fácilmente en la petición al servidor junto al resto de información.

Una vez completado y validado el formulario, el administrador pulsa el botón ***Siguiente/Resumen***. Aquí entra en juego el **fragmento de resumen "ResumenEventoFragment",** que muestra una tarjeta con todos los datos introducidos, maquetada de forma similar a como la verán los usuarios finales en el detalle del evento. En este resumen se incluyen el título, la descripción, la fecha y hora, la ubicación, la imagen y, en caso de eventos de competición, los datos específicos de equipos o participantes. Este paso es clave porque permite al administrador revisar la información con una **vista previa realista** antes de publicarla.

Si el administrador detecta algún error, puede retroceder al formulario para corregirlo sin perder los datos. Si todo está correcto, pulsa el botón *Confirmar y crear*. En ese momento la aplicación construye un objeto **EventoRequest** y lo envía mediante **Retrofit** al **backend de la aplicación**, junto con el token de autenticación del administrador. El servidor valida el rol del usuario, vincula el evento al ayuntamiento activo en la sesión y lo registra en la base de datos.

Cuando la creación es exitosa, la aplicación muestra un mensaje de confirmación y el nuevo evento queda disponible de forma inmediata. Se refleja automáticamente en dos secciones: en la **lista general de eventos del municipio**, accesible a todos los vecinos e invitados, y en el **perfil del administrador**, dentro del **tab** de *Eventos*. De esta manera, el consistorio puede comprobar que el alta se ha realizado correctamente y gestionar desde allí los eventos futuros.

Figura 6-44. Pantalla de selección del tipo de evento.

En caso de error durante la validación o de fallo de conexión, se notifica al administrador sin borrar los datos del formulario, lo que facilita corregir y reintentar sin necesidad de comenzar desde cero.

En resumen, el proceso de creación de eventos combina simplicidad de uso con robustez técnica: se estructura en varios pasos (selección de tipo, formulario dinámico, previsualización y confirmación) para minimizar errores y asegurar que la información publicada es completa y coherente. Esto convierte la aplicación en una herramienta ágil y accesible para los ayuntamientos, capaces de comunicar de forma instantánea cualquier iniciativa a sus vecinos, fomentando así la participación ciudadana y la transparencia en la gestión local.

Figura 6-46. Pantalla de Introducción de datos del evento

Figura 6-45. Pantalla de resumen del evento a crear.

En conjunto, esta arquitectura ofrece una **entrada unificada** al municipio: clima para planificar, eventos para participar, incidencias para colaborar y un perfil institucional claro alojado en el ***drawer***. La fragmentación en módulos con adaptadores independientes permite **actualizaciones eficientes**, y la combinación **Drawer + Bottom** bar aporta una navegación **predecible y rápida**, manteniendo siempre el **contexto del ayuntamiento activo** visible y bajo control del usuario.

Capítulo 7 - Conclusiones y trabajo futuro

7.1 Conclusiones

El desarrollo de este proyecto ha permitido crear una aplicación móvil funcional que contribuye a la digitalización de los pueblos de la llamada ***España vaciada***. A través de funcionalidades de gestión de eventos, reserva, reporte de incidencias, consulta, información municipal y previsión meteorológica, la aplicación ofrece a vecinos y visitantes un canal de comunicación más ágil y accesible con su ayuntamiento. Lo que genera una mayor atracción al mundo rural.

Desde el punto de vista técnico, el proyecto ha supuesto un aprendizaje significativo en el uso de tecnologías como **Android** con **Java**, **Spring Boot**,, **MySQL**, autenticación mediante **JWT**, e integraciones externas como **Stripe** y varias **APIs** como la de **OpenWeatherMap**. La implementación de estas herramientas ha permitido construir una base sólida, sobre la que ampliar la solución en un futuro.

No obstante, no se han alcanzado todos los objetivos planteados inicialmente debido a la complejidad y extensión del trabajo. Entre las funciones descartadas destacan el módulo de **rutas y puntos de interés**, que habría permitido a los usuarios crear y valorar recorridos turísticos. Estas limitaciones no deben entenderse como fracasos, sino con un ejercicio de. Acotación que ha permitido centrar los esfuerzos en el núcleo funcional robusto y plenamente operativo.

Más allá de lo técnico, este proyecto pone de relieve la importancia de acercar soluciones digitales a los pueblos pequeños. La aplicación puede convertirse en una herramienta de cohesión social y participación ciudadana, reforzando el vínculo entre los vecinos y ayuntamientos, simplificando trámites que hoy siguen siendo analógicos y aportando visibilidad a los municipios rurales en un contexto de despoblación.

7.2 Trabajo futuro

De cara a ver si les posteriores, la aplicación ofrece múltiples posibilidades de ampliación:

- **Rutas y puntos de interés,** recuperando la idea original de que cualquier usuario pueda crearlos, añadir valoraciones y compartir experiencias turísticas.
- **Módulo de noticias y actualidad local,** para centralizar la comunicación municipal.
- **Notificaciones push en tiempo real** sobre incidencias, eventos o cambios relevantes que pueda recibir el usuario en su dispositivo móvil.
- **Mejoras en la pasarela de pago,** ampliando los métodos disponibles más allá de **Stripe.**
- **Optimización del modo multi municipio,** potenciando la **app** como una red social rural para que los usuarios puedan interactuar en varios pueblos de forma simultánea.

En conclusión, este proyecto no ha sido solo un ejercicio de desarrollo tecnológico, sino también una reflexión sobre el papel que las aplicaciones móviles pueden jugar en la **revitalización de los municipios rurales**. Aunque aún queda camino por recorrer, la base construida abre la puerta a un futuro en el que la **digitalización** contribuye de manera decisiva a mantener vivos y conectados a los pueblos de la ***España vaciada***.

Chapter 7- Conclusions and future work

7.1 Conclusions

The development of this project has resulted in a functional mobile application aimed at supporting the digital transformation of towns in what is commonly referred to as the *emptied Spain*. Through its main features event management, reservations, incident reporting, municipal information, and weather forecastingthe application provides both residents and visitors with a more accessible and agile communication channel with their local councils.

From a technical perspective, the project has represented a significant learning experience in the use of technologies such as Android with Java, Spring Boot, MySQL, JWT-based authentication, and external integrations including Stripe and OpenWeatherMap. The implementation of these tools has enabled the creation of a solid foundation on which the solution can be further expanded in future iterations.

However, not all of the initial objectives could be achieved due to the complexity and scope of the project. Among the functionalities left aside were the **routes and points of interest module**, which would have allowed users to create and rate tourist paths, and the incorporation of **municipal administrator profiles**, intended to directly manage events and incidents within the application. These limitations should not be regarded as failures, but rather as a necessary acotación of scope that made it possible to focus efforts on building a robust and fully operational core.

Beyond the technical aspects, this project highlights the importance of bringing digital solutions closer to small towns. The application can become a tool for social cohesion and citizen participation, strengthening the connection between residents and their councils, simplifying processes that are still analog today, and increasing the visibility of rural municipalities in the current context of depopulation.

7.2 Future Work

For future versions, the application offers several promising directions for improvement and expansion:

- **Routes and points of interest**, reviving the original idea of allowing any user to create, share, and review tourist routes.
- **Local news and updates module**, centralizing municipal communication in a single platform.
- **Push notifications** in real time for incidents, events, and relevant updates.
- **Enhancements to the payment gateway**, expanding beyond Stripe to include additional methods more adapted to rural settings.
- **Multi-municipality optimization**, strengthening the **app** as a small-scale rural social network where users can interact with multiple towns simultaneously.

In conclusion, this project has not only been an exercise in technological development but also a reflection on the role mobile applications can play in revitalizing rural municipalities. Although there is still work to be done, the foundation built here opens the door to a future in which digitalization may play a decisive role in keeping the towns of the emptied Spain alive, connected, and thriving.

BIBLIOGRAFÍA

[1] Spring, «Spring Boot Reference Documentation. Structuring Your Code,» [En línea]. Available: https://docs.spring.io/spring-boot/reference/using/structuring-your-code.html.

[2] I. Š. y. T. Ranisavljević, «Optimization of MySQL database.,» *Journal of Process Managements and New Technologies,* vol. 11, pp. 141- 151, 2023.

[3] C. Tudose, Java Persistence with Spring Data and Hibernate, Manning, 2023.

[4] Spring, «Spring Data JPA Reference Documentation. Core concepts,» [En línea]. Available: https://docs.spring.io/spring-data/jpa/reference/repositories/core-concepts.html.

[5] H. L. Morvan, Java Spring: la base técnica de las aplicaciones Jakarta EE, Ediciones ENI, 2023.

[6] A. Developers, «Android Studio Guide,» [En línea]. Available: http://developer.android.com/studio/intro?hl=es-419#manage_dependencies.

[7] J. Franganillo, Formatos digitales: propiedades técnicas y contextos de uso, UOC, 2022.

[8] J. T. G. y. J. L. Mauri, El gran libro de Android, Marcombo, 2022.

[9] Spring, «Spring Data JPA Reference Documentation. Repository interfaces,» [En línea]. Available: https://docs.spring.io/spring-data/jpa/reference/repositories/definition.html.

[10] S. Patni, Pro RESTful APOs with Micronaut: build Java-based microservices with REST, JSON, and XML., Apress, 2025.

[11] T. Richard, Java: los fundamentos del lenguaje, Ediciones ENI, 2022.

APÉNDICE A - CASOS DE USO

Gestión de sesiones

- **Registrar usuario**

Actores	Usuario registrado y Ayuntamiento
Precondición	Ninguna
Flujo Normal	1. El usuario inicia la aplicación. 2. El usuario pulsa en el botón "Únete a uno" 3. El sistema solicita nombre, apellido, email, contraseña el DNI y el número de teléfono. 4. El usuario rellena todos los datos necesarios. 5. El usuario pulsa en "Registrarse"
Postcondición	1. El usuario queda registrado en la bd 2. El usuario Vuelve a la pantalla de inicio para que inicie sesión y pueda acceder a la app logueado ya.

Tabla 1. Registrar usuario

- **Login**

Actores	Usuario Registrado
Precondición	Usuario registrado previamente.
Flujo Normal	1. El usuario inicia la app. 2. El sistema solicita email y contraseña. 3. El usuario rellena las credenciales. 4. El usuario pulsa en "Iniciar sesión" 5. El sistema valida las credenciales. 6. El sistema muestra la pantalla de selección de pueblo.

Postcondición	1. La sesión queda iniciada pero pendiente de pueblo activo. 2. El usuario solo acede a la app tras seleccionar/unirse a un pueblo.

Tabla 2. Login

- **Entrar como invitado**

Actores	Usuario Invitado
Precondición	Ninguna
Flujo Normal	1. Usuario inicia la app. 2. El usuario pulsa en "Continuar como invitado". 3. El sistema muestra la pantalla de "Buscar municipio"
Postcondición	El usuario accede a la app en modo invitado.

Tabla 3. Entrar como invitado

- **Seleccionar pueblo/ayuntamiento**

Actores	Usuario Registrado
Precondición	Usuario ya logueado previamente y tener uno o más ayuntamientos asociados.
Flujo Normal	1. El sistema muestra la lista de ayuntamientos en los que el usuario está registrado. 2. El usuario pulsa en el botón "Seleccionar".
Postcondición	1. El sistema guarda el pueblo en cuestión seleccionado por el usuario, activo en sesión. 2. El usuario accede a la app con la sesión iniciada en el municipio seleccionado.

Tabla 4. Seleccionar pueblo/ayuntamiento

- **Unirse pueblo/ayuntamiento**

Actores	Usuario Registrado, Usuario Invitado
Precondición	Haber iniciado sesión o haber accedido a la app como invitado.
Flujo Normal	1. Si el usuario ha entrado como invitado, el sistema abre directamente la pantalla de "Buscar Municipio". 2. De otra manera, si el usuario ha accedido a la app iniciando sesión, pulsa en "Unirme a uno". 3. El sistema muestra el buscador y el listado de los pueblos de la bd. 4. El usuario pulsa encima del nombre del pueblo al que quiera unirse. 5. El sistema marca como seleccionado el pueblo el cual ha sido pulsado por el usuario. 6. El usuario pulsa en "Unirme".
Postcondición	1. El usuario queda asociado/registrado en el pueblo seleccionado. 2. Accede directamente a la app con ese pueblo seleccionado. 3. El sistema registra la acción de que el usuario logueado se haya registrado en el pueblo en cuestión.

Tabla 5. Unirse pueblo/ayuntamiento

- **Salir del municipio**

Actores	Usuario registrado, Usuario invitado
Precondición	Usuario se encuentra dentro de la app, en sesión con ayuntamiento activo
Flujo Normal	1. El usuario pulsa en el boton de arriba a la izquierda para abrir el menú vertical y lateral. 2. El usuario pulsa en "Salir del municipio". 3. El sistema muestra un dialogo de texto para confirmar la salida del municipio. 4. El usuario pulsa en "Si" para salir definitivamente del municipio actual.

	5. El sistema redirige el flujo a la pantalla de "Seleccionar pueblo", en el caso de que el usuario se haya logueado, y sino redirigirá a la pantalla de" Unirse a municipio"
Postcondición	1. El usuario sale del municipio/pueblo. 2. No hay ayuntamiento/pueblo activo hasta que el usuario seleccione/se una a otro nuevo.

Tabla 6. Salir del municipio

- **Cerrar sesión**

Actores	Usuario registrado
Precondición	Usuario con sesión iniciada (token válido).
Flujo Normal	1. El usuario pulsa en "Mi perfil", en el menú inferior. 2. El sistema muestra la información del perfil. 3. El usuario pulsa en "Cerrar sesión". 4. El sistema invalidará el token de sesión, y borra las credenciales locales.
Postcondición	1. El sistema automáticamente redirige el flujo a la pantalla de inicio de la app. 2. La sesión queda cerrada; no hay municipio/pueblo activo. 3. No se realizan cambios en datos municipales.

Tabla 7. Cerrar sesión

Eventos

- **Buscar/consultar listado de eventos**

Actores	Usuario registrado, Usuario invitado
Precondición	Ayuntamiento/pueblo activo en sesión.
Flujo Normal	1. El usuario pulsa en "Eventos" 2. El usuario, opcionalmente, introduce criterios de búsqueda y/o filtros.

	3. El sistema hace la búsqueda instantánea.
Postcondición	El sistema muestra el listado de eventos del municipio activo atendiendo a los filtros que opcionalmente había puesto el usuario.

Tabla 8. Buscar/consultar listado de eventos

- **Ver detalle evento**

Actores	Usuario registrado, Usuario invitado
Precondición	Listado de eventos visible.
Flujo Normal	1. El usuario pulsa en el evento el cual quiera ver su detalle. 2. El sistema muestra la pantalla del detalle del evento.
Postcondición	El sistema muestra la información al completo del evento en cuestión.

Tabla 9. Ver detalle evento

- **Reservar evento de pago (INVITADO)**

Actores	Usuario invitado
Precondición	Ayuntamiento activo en sesión, evento disponible y con aforo suficiente.
Flujo Normal	1. El usuario pulsa en "Reservar" estando en la pantalla de detalle del evento. 2. El sistema muestra un dialogo de compra/reserva. 3. El usuario introduce los datos mínimos requeridos (nombre, email). 4. El usuario selecciona el número de entradas o participaciones que quiere adquirir y apunta los datos de las personas en cuestión. 5. El sistema iniciara una pasarela de pago (usando Stripe). 6. El usuario introduce los datos fiscales, y pulsa en "Confirmar y Pagar".
Postcondición	1. El sistema crea la nueva reserva. 2. La operación queda registrada en el ayuntamiento/pueblo en cuestión

Tabla 10. Reservar evento de pago (invitado)

- **Reservar evento gratuito (INVITADO)**

Actores	Usuario invitado
Precondición	Ayuntamiento activo en sesión, evento disponible y con aforo suficiente.
Flujo Normal	1. El usuario pulsa en "Reservar" estando en la pantalla de detalle del evento. 2. El sistema muestra un dialogo de compra/reserva. 3. El usuario selecciona el número de entradas o participaciones que quiere adquirir y apunta los datos de las personas en cuestión. 4. El usuario introduce los datos fiscales, y pulsa en "Confirmar reserva".
Postcondición	1. El sistema crea la nueva reserva. 2. La operación queda registrada en el ayuntamiento/pueblo en cuestión

Tabla 11. Reservar evento gratuito (invitado)

- **Reservar evento de pago (REGISTRADO)**

Actores	Usuario registrado
Precondición	Sesión iniciada, Ayuntamiento activo en sesión, evento disponible y con aforo suficiente.
Flujo Normal	1. El usuario pulsa en "Reservar" estando en la pantalla de detalle del evento. 2. El sistema muestra un dialogo de compra/reserva. 3. El usuario introduce los datos mínimos requeridos (nombre, email). 4. El usuario selecciona el número de entradas o participaciones que quiere adquirir y apunta los datos de las personas en cuestión. 5. El sistema iniciara una pasarela de pago (usando Stripe). 6. El usuario introduce los datos fiscales, y pulsa en "Confirmar y Pagar".
Postcondición	1. El sistema crea la nueva reserva.

	2. El sistema hace visible la reserva en el apartado "Mis reservas" de "Mi perfil". 3. La operación queda registrada en el ayuntamiento/pueblo en cuestión

Tabla 12. Reservar evento de pago (registrado)

- **Reservar evento gratuito (REGISTRADO)**

Actores	Usuario registrado
Precondición	Ayuntamiento activo en sesión, evento disponible y con aforo suficiente.
Flujo Normal	1. El usuario pulsa en "Reservar" estando en la pantalla de detalle del evento. 2. El sistema muestra un dialogo de compra/reserva. 3. El usuario selecciona el número de entradas o participaciones que quiere adquirir y apunta los datos de las personas en cuestión. 4. El usuario introduce los datos fiscales, y pulsa en "Confirmar reserva".
Postcondición	1. El sistema crea la nueva reserva. 2. El sistema hace visible la reserva en el apartado "Mis reservas" de "Mi perfil". 3. La operación queda registrada en el ayuntamiento/pueblo en cuestión

Tabla 13. Reservar evento gratuito (registrado)

- **Cancelar reserva**

Actores	Usuario registrado
Precondición	Sesión iniciada, resera activa y cancelable (que no sea un evento de poago)
Flujo Normal	1. El usuario pulsa en "Mi perfil". 2. El sistema muestra la pantalla del perfil de usuario. 3. El usuario pulsa en "Mis reservas". 4. El sistema muestra las reservas realizadas por el usuario. 5. El usuario pulsa sobre la reserva en cuestión 6. El sistema abre un dialogo de detalle de la reserva en el que aparece un botón de cancelar reserva. 7. El usuario pulsa en "Cancelar reserva"
Postcondición	1. Reserva cancelada. 2. El sistema registra la reserva cancelada, y la elimina de la bbdd y actualiza los registros de los eventos y reservas del pueblo/ayuntamiento.

Tabla 14. Cancelar reserva

Favoritos

- **Añadir a favoritos**

Actores	Usuario registrado
Precondición	Sesión iniciada, y evento todavía disponible en cuanto a fechas
Flujo Normal	1. El usuario pulsa en el dibujo de a estrella que se sitúa en la esquina superior derecha de un evento. 2. El sistema pinta automáticamente esa misma estrella de color amarillo.
Postcondición	1. Evento asociado a favoritos del usuario (No impacta al ayuntamiento/pueblo).

	2. El sistema hace visible el evento pulsado por el usuario como favorito, en la pestaña de "Mis favoritos", en la pantalla del "Mi perfil" (en el perfil de usuario).

Tabla 15. Añadir a favoritos

- **Eliminar de favoritos**

Actores	Usuario registrado
Precondición	Sesión iniciada, y evento todavía disponible en cuanto a fechas
Flujo Normal	1. El usuario pulsa en el dibujo de a estrella que se sitúa en la esquina superior derecha de un evento. 2. El sistema pinta automáticamente esa misma estrella de color blanco.
Postcondición	1. Evento es eliminado de favoritos del usuario (No impacta al ayuntamiento/pueblo). 2. El sistema elimina el evento pulsado por el usuario como favorito de la pestaña de "Mis favoritos", en la pantalla del "Mi perfil" (en el perfil de usuario).

Tabla 16. Eliminar de favoritos

- **Consulta de favoritos**

Actores	Usuario registrado
Precondición	Sesión iniciada
Flujo Normal	1. El usuario pulsa en el menú inferior en "Mi perfil". 2. El sistema abre la pantalla del perfil de usuario. 3. El usuario pulsa en la pestaña de "Mis favoritos"
Postcondición	El sistema lista todos los eventos que el usuario haya guardado como favorito.

Tabla 17. Consulta de favoritos

Incidencias

- **Buscar/consultar incidencias**

Actores	Usuario registrado, Usuario invitado.
Precondición	Sesión con ayuntamiento/pueblo activo
Flujo Normal	1. El usuario pulsa en el menú inferior en "Incidencias". 2. El usuario, opcionalmente, introduce criterios de búsqueda y/o filtros. 3. El sistema hace la búsqueda instantánea.
Postcondición	El sistema muestra el listado de incidencias del municipio activo atendiendo a los filtros que opcionalmente había puesto el usuario.

Tabla 18. Buscar/consultar incidencias

- **Ver detalle de incidencia**

Actores	Usuario registrado, Usuario invitado.
Precondición	Listado de incidencias visible.
Flujo Normal	El usuario pulsa en una de las incidencias de todo el listado de estas.
Postcondición	El sistema muestra la información al completo de la incidencia en cuestión.

Tabla 19. Ver detalle de incidencia

- **Crear incidencia**

Actores	Usuario registrado, Usuario invitado.
Precondición	Ayuntamiento/pueblo activo en sesión, (opcional) permisos cámara y ubicación.
Flujo Normal	1. El usuario pulsa en el botón de crear incidencia 2. El sistema abre el dialogo de crear incidencia.

	3. El usuario rellena todos los datos necesarios o imprescindibles (título y descripción), además de los datos opcionales (Adjuntar foto y geolocalización) 4. El usuario pulsa en "Confirmar".
Postcondición	1. El sistema crea la incidencia y la envía al ayuntamiento. 2. Se registra todo en el ayuntamiento/pueblo. 3. En el caso de que el usuario este registrado, el sistema mostrará la incidencia creada en la pestaña de "Mis incidencias" del perfil de usuario, ofreciendo la posibilidad de eliminarla, mientras el estado de esta sea "Pendiente".

Tabla 20. Crear incidencia

- **Eliminar incidencia**

Actores	Usuario registrado.
Precondición	Sesión iniciada, ser autor de la incidencia en cuestión, y que esta esté en estado eliminable
Flujo Normal	1. El usuario pulsa en "Editar incidencia". 2. El sistema abre el dialogo de editar incidencia. 3. El usuario edita todos los datos que quiera modificar. 4. El usuario pulsa en "Confirmar cambios".
Postcondición	1. El sistema elimina la incidencia 2. Se registra todo en el ayuntamiento/pueblo. 3. El sistema eliminará la incidencia creada en la pestaña de "Mis incidencias" del perfil de usuario.

Tabla 21. Eliminar incidencia

Información municipal y clima

- **Consultar información del ayuntamiento/pueblo**

Actores	Usuario registrado, Usuario invitado.
Precondición	Ayuntamiento/pueblo activo en sesión.
Flujo Normal	1. El usuario pulsa en el botón de información del pueblo, en la esquina superior izquierda. 2. El sistema despliega el menú vertical y aparecen las opciones del ayuntamiento/pueblo. 3. El usuario pulsa en "Información del ayuntamiento".
Postcondición	El sistema muestra la pantalla de información del municipio, con todos los detalle y datos correspondientes al ayuntamiento/pueblo.

Tabla 22. Consultar información del ayuntamiento/pueblo

- **Consultar el clima pueblo/municipio**

Actores	Usuario registrado, Usuario invitado.
Precondición	Ayuntamiento/pueblo activo en sesión.
Flujo Normal	1. El usuario pulsa en el botón de información del pueblo, en la esquina superior izquierda. 2. El sistema despliega el menú vertical y aparecen las opciones del ayuntamiento/pueblo. 3. El usuario pulsa en "El tiempo". Por otro lado, en la pantalla principal de la app, una vez en el pueblo, si el usuario pulsa en la pestaña del clima que sale al inicio, ocurre lo mismo.
Postcondición	El sistema muestra la pantalla de información del tiempo del pueblo/municipio.

Tabla 23. Consultar el clima pueblo/municipio