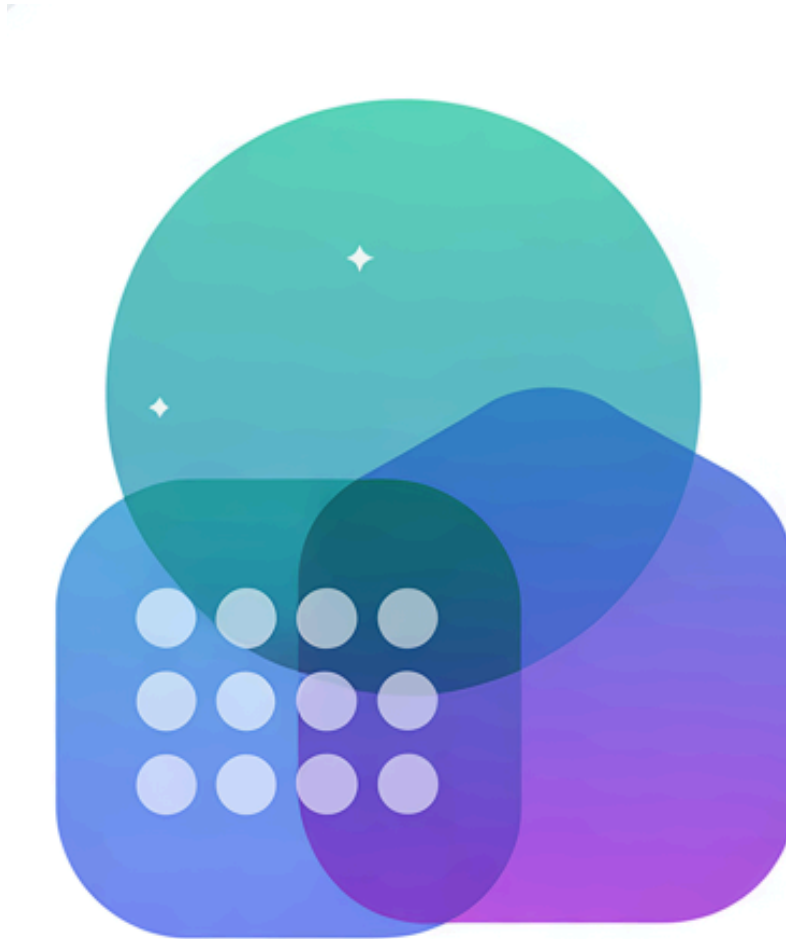


PLANORA



CFGs Desarrollo de Aplicaciones Multiplataforma

Agustín Luz Martín

ÍNDICE

1. INTRODUCCIÓN	2
2. ESTUDIO DE VIABILIDAD	3
2.1. Descripción del sistema actual	3
2.3. Identificación de requisitos del sistema	6
2.4. Descripción de la solución	16
2.5. Planificación del proyecto	16
2.6. Estudio del coste del proyecto	16
3. ANÁLISIS DEL SISTEMA DE INFORMACIÓN	17
3.1. Entorno tecnológico	17
3.2. Modelado de datos	18
3.3 Identificación de los usuarios participantes y finales	23
3.4 Identificación de subsistemas de análisis	30
3.5 Establecimiento de requisitos	33
3.6 Diagramas de Análisis	36
3.7.1 Especificación de principios generales de interfaz	40
3.7.2 Especificación de formatos individuales de la interfaz de pantalla	41
3.7.3 Matriz de acceso por pantalla	108
3.7.4 Especificación de formatos de impresión	109
3.7.5 Especificación de la navegabilidad entre pantallas	109
4. Construcción del sistema	109
4.1 Arquitectura técnica	110
4.2 Tecnologías utilizadas	110
4.3 Estructura del proyecto	111
Conclusión	113
Bibliografía	113

1. INTRODUCCIÓN

El presente documento expone el desarrollo del proyecto integrado (PI) denominado Planora, una aplicación móvil híbrida diseñada para dar respuesta a una necesidad común y cotidiana: la organización estructurada, colaborativa y eficiente de eventos y planes dentro de grupos cerrados de amigos.

En la actualidad, la organización de cenas, viajes, cumpleaños y otras actividades sociales entre amigos suele gestionarse mediante aplicaciones de mensajería instantánea o redes sociales como WhatsApp e Instagram. Sin embargo, estas herramientas no ofrecen funcionalidades específicas para coordinar decisiones clave como la asistencia confirmada, realización de votaciones internas, control de gastos compartidos o gestión centralizada de fotos y notas relacionadas con dichos eventos. Esta falta de estructura provoca frecuentes confusiones, pérdida de información importante y desorganización general, impactando negativamente en la experiencia de los usuarios.

Planora pretende cubrir esta brecha mediante una solución integral adaptada a las necesidades específicas de grupos de amigos. La aplicación permite a los usuarios crear eventos detallados, realizar votaciones sobre aspectos concretos, compartir fotografías en galerías colaborativas, generar notas conjuntas y llevar un control claro y transparente de los gastos grupales. Todo esto centralizado en una interfaz sencilla, intuitiva y accesible desde dispositivos móviles.

Este proyecto integrado conjuga tanto los conocimientos técnicos adquiridos durante el ciclo formativo (Ionic, Vue.js, Spring Boot, MySQL, gestión de estado, entre otros) como habilidades prácticas en áreas clave tales como el diseño de interfaces, análisis funcional, planificación y desarrollo ágil, así como la presentación final de un producto realista, profesional y funcional, preparado para una posible evolución futura..

2. ESTUDIO DE VIABILIDAD

2.1. Descripción del sistema actual

2.1 Sistema actual

Actualmente, la mayoría de las personas utiliza aplicaciones generales como **WhatsApp** o **Instagram** para organizar planes con amigos. Aunque son herramientas de comunicación ampliamente adoptadas y útiles para el intercambio casual de mensajes, no están orientadas a una planificación estructurada de eventos grupales. Esto conlleva varios inconvenientes significativos:

- **Falta de organización de la información:** mensajes importantes sobre asistencia confirmada, montos a pagar, actividades específicas o decisiones tomadas se pierden fácilmente entre decenas o cientos de mensajes sin estructura ni jerarquía clara.
- **Confusión en la toma de decisiones:** las votaciones se realizan manualmente en chats, sin quedar registradas de forma clara o persistente, lo que provoca duplicidades frecuentes y errores en la gestión de las decisiones tomadas.
- **Descontrol sobre los gastos compartidos:** habitualmente no se lleva un control sistemático ni transparente sobre cuánto dinero ha pagado cada persona o cómo se divide correctamente el gasto grupal, causando malentendidos o disputas ocasionales.
- **Ausencia de espacio para guardar recuerdos:** las fotografías importantes del evento suelen mezclarse con contenido irrelevante (como memes o mensajes temporales) o se pierden con facilidad debido a la falta de centralización.
- **Carencia de listas y recordatorios colaborativos:** las tareas y recordatorios suelen depender del esfuerzo individual, lo que limita la eficiencia y coordinación en grupos grandes o muy activos.

Además, aunque existen otras aplicaciones especializadas que cubren parcialmente algunas necesidades organizativas, presentan limitaciones importantes al no ofrecer una solución integral orientada específicamente a grupos cerrados de amigos:

Aplicación	Funcionalidad destacada	Limitación principal
Splitwise	Gestión de gastos entre amigos	No tiene eventos, votaciones ni galería
Doodle	Votaciones de fechas	Solo votaciones, no eventos ni gastos
Google Keep	Notas colaborativas	No tiene enfoque social, no hay grupos
Trello/Notion	Planificación avanzada	Poco amigable para usuarios no técnicos

Como se puede observar, no existe en la actualidad una aplicación específica "todo-en-uno" para grupos cerrados de amigos que combine en un mismo entorno amigable la gestión estructurada de eventos, control de gastos compartidos, votaciones internas, galerías colaborativas de fotos y notas conjuntas. Esta carencia constituye la justificación principal para el desarrollo de **Planora**, cuya propuesta busca cubrir integralmente estas necesidades y ofrecer una experiencia organizada y fluida en la planificación social.

Según un estudio reciente, aproximadamente el 40% de los usuarios experimentan dificultades recurrentes en la gestión clara de eventos utilizando apps no especializadas [1].

2.2 Sistema propuesto (Planora)

El sistema propuesto, denominado Planora, tiene como finalidad cubrir las carencias detectadas en el sistema actual, ofreciendo una solución integral adaptada específicamente a las necesidades concretas de grupos cerrados de amigos. Planora busca simplificar y estructurar la organización de eventos sociales mediante una única plataforma móvil que permita gestionar eficazmente todos los aspectos relevantes de manera colaborativa.

Entre las principales funcionalidades que incluirá Planora destacan:

- Gestión centralizada de eventos: creación, edición y visualización estructurada de eventos con detalles específicos como fecha, hora, ubicación y asistentes confirmados, evitando pérdida de información clave.
- Votaciones internas claras y registradas: sistema integrado de votaciones con opciones múltiples y resultados claramente visibles, lo que elimina las confusiones y evita duplicidades innecesarias.
- Gestión transparente de gastos compartidos: módulo específico para registrar quién paga cada gasto, cómo se distribuye entre los participantes y alertas automáticas sobre balances pendientes, garantizando transparencia y reduciendo conflictos por cuestiones monetarias.
- Galería colaborativa de fotos: espacio dedicado exclusivamente a guardar y compartir imágenes relacionadas con cada evento, facilitando un acceso cómodo, rápido y ordenado a los recuerdos.
- Notas colaborativas y recordatorios grupales: creación y gestión conjunta de listas y notas en tiempo real, asegurando la accesibilidad constante y coordinación efectiva.

Estas características permiten diferenciar claramente a Planora frente a las aplicaciones generales actualmente utilizadas (como WhatsApp o Instagram) y frente a aplicaciones especializadas que solo cubren parcialmente las necesidades detectadas (Splitwise, Doodle, Google Keep, Trello o Notion).

Además, Planora se concibe bajo objetivos concretos que aseguran su eficiencia y relevancia práctica, planteados bajo la metodología SMART:

- Objetivo general:
 - Desarrollar, en un periodo de cuatro meses, una aplicación móvil híbrida que reduzca en al menos un 50% el tiempo empleado por los usuarios para coordinar eventos grupales, en comparación con herramientas no especializadas.

- **Objetivos específicos:**
 - Implementar un módulo de gestión de gastos para registrar automáticamente el 100% de las deudas y gastos generados, acompañado por un sistema eficiente de notificaciones automáticas
 - Proporcionar un sistema interno de votaciones que logre al menos un 90% de participación efectiva durante las pruebas realizadas con grupos objetivo.
 - Diseñar y desarrollar una interfaz móvil responsive, evaluada mediante la herramienta Lighthouse, que obtenga una puntuación mínima de 80 sobre 100.

Finalmente, el alcance del proyecto abarca el desarrollo completo del backend utilizando Spring Boot, con API REST robusta y base de datos MySQL, junto con un frontend híbrido desarrollado en Ionic y Vue.js. En esta fase inicial, el proyecto no contempla la integración con pasarelas de pago externas ni el despliegue real en producción, enfocándose principalmente en la validación funcional integral y demostración técnica del producto desarrollado.

2.3. Identificación de requisitos del sistema

En este apartado se definen los requisitos identificados para el correcto desarrollo de Planora, organizados en tres bloques:

- **Requisitos de información:** describen las estructuras de datos y entidades principales que almacenarán la información de la aplicación (usuarios, grupos, eventos, gastos, votos, imágenes, etc.).
- **Requisitos funcionales:** detallan las acciones y casos de uso que el sistema debe permitir, como crear y gestionar eventos, realizar votaciones, controlar gastos compartidos, gestionar galerías de fotos y notas colaborativas.
- **Otros requisitos:** establecen las condiciones técnicas mínimas necesarias para el funcionamiento óptimo de la app, incluyendo compatibilidad híbrida (Android e iOS), rendimiento (tiempos de carga < 2 s), seguridad (autenticación JWT) y accesibilidad móvil.

2.3.1. Requisitos de información

A continuación se describen las entidades principales y sus estructuras de datos, tal como aparecen reflejadas en el esquema de base de datos MySQL:

- **Entidad Grupo (**grupos**)**
 - **Atributos:**
 - **id** (BIGINT, PK, AUTO_INCREMENT)
 - **codigo_invitacion** (VARCHAR(255))
 - **nombre** (VARCHAR(255))
 - **imagen_perfil** (LONGTEXT)
- **Entidad Usuario (**usuarios**)**
 - **Atributos:**
 - **id** (BIGINT, PK, AUTO_INCREMENT)
 - **email** (VARCHAR(255), UNIQUE, NOT NULL)
 - **nombre** (VARCHAR(255))
 - **foto_perfil** (LONGTEXT)
 - **password** (VARCHAR(255))
- **Entidad UsuarioGrupo (**usuarios_grupos**)**
 - **Atributos:**
 - **id** (BIGINT, PK, AUTO_INCREMENT)
 - **usuario_id** (BIGINT, FK → **usuarios.id**)
 - **grupo_id** (BIGINT, FK → **grupos.id**)
 - **rol** (VARCHAR(255))
- **Entidad Invitación (**invitaciones**)**
 - **Atributos:**

- `id` (BIGINT, PK, AUTO_INCREMENT)
- `grupo_id` (BIGINT, FK → `grupos.id`)
- `usuario_id` (BIGINT, FK → `usuarios.id`)
- `estado` (VARCHAR(20), NOT NULL)
- `fecha` (DATETIME, NOT NULL, DEFAULT CURRENT_TIMESTAMP)
- **Entidad Evento (`eventos`)**
 - **Atributos:**
 - `id` (BIGINT, PK, AUTO_INCREMENT)
 - `descripcion` (LONGTEXT)
 - `fecha` (DATETIME)
 - `titulo` (VARCHAR(255))
 - `ubicacion` (VARCHAR(255))
 - `grupo_id` (BIGINT, FK → `grupos.id`)
 - `creador_id` (BIGINT, FK → `usuarios.id`)
- **Entidad EventoAsistente (`evento_asistentes`)**
 - **Atributos:**
 - `id` (BIGINT, PK, AUTO_INCREMENT)
 - `evento_id` (BIGINT, FK → `eventos.id`)
 - `usuario_id` (BIGINT, FK → `usuarios.id`)
 - `asistio` (BOOLEAN, NOT NULL)

- **Entidad AsistenciaEvento (*asistencias_eventos*)**
 - **Atributos:**
 - *id* (BIGINT, PK, AUTO_INCREMENT)
 - *usuario_id* (BIGINT, FK → *usuarios.id*)
 - *evento_id* (BIGINT, FK → *eventos.id*)
 - *asistio* (BOOLEAN, NOT NULL)
- **Entidad Gasto (*gastos*)**
 - **Atributos:**
 - *id* (BIGINT, PK, AUTO_INCREMENT)
 - *monto* (DECIMAL(38,2))
 - *titulo* (VARCHAR(255))
 - *partes_iguales* (BOOLEAN, NOT NULL)
 - *fecha_creacion* (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP)
 - *pagado_por* (BIGINT, FK → *usuarios.id*)
 - *grupo_id* (BIGINT, FK → *grupos.id*)
 - *evento_id* (BIGINT, FK → *eventos.id*)
- **Tabla *gastos_usuarios* (asociativa)**
 - *gasto_id* (BIGINT, FK → *gastos.id*)
 - *usuario_id* (BIGINT, FK → *usuarios.id*)
- **Tabla *cantidades_personalizadas***

- `gasto_id` (BIGINT, FK → `gastos.id`)
- `usuario_id` (BIGINT, FK → `usuarios.id`)
- `monto` (DECIMAL(38,2))
- **Entidad DeudaGasto (`deudas_gastos`)**
 - **Atributos:**
 - `id` (BIGINT, PK, AUTO_INCREMENT)
 - `gasto_id` (BIGINT, FK → `gastos.id`)
 - `deudor_id` (BIGINT, FK → `usuarios.id`)
 - `acreedor_id` (BIGINT, FK → `usuarios.id`)
 - `monto` (DECIMAL(38,2))
 - `saldado` (BOOLEAN, NOT NULL)
 - `fecha_saldado` (DATETIME)
 - `metodo_pago` (VARCHAR(255))
 - `notas` (TEXT)
- **Entidad Imagen (`imagenes`)**
 - **Atributos:**
 - `id` (BIGINT, PK, AUTO_INCREMENT)
 - `nombre` (VARCHAR(255))
 - `tipo_contenido` (VARCHAR(100))
 - `tamaño` (BIGINT)

- `datos` (LONGTEXT)
- `fecha_creacion` (DATETIME, NOT NULL, DEFAULT CURRENT_TIMESTAMP)
- `evento_id` (BIGINT, FK → `eventos.id`)
- `usuario_id` (BIGINT, FK → `usuarios.id`)
- `grupo_id` (BIGINT, FK → `grupos.id`)
- Entidad Nota (**`notas`**)
 - Atributos:
 - `id` (BIGINT, PK, AUTO_INCREMENT)
 - `titulo` (VARCHAR(255))
 - `fecha_creacion` (DATETIME, NOT NULL, DEFAULT CURRENT_TIMESTAMP)
 - `contenido` (LONGTEXT)
 - `grupo_id` (BIGINT, FK → `grupos.id`)
 - `creada_por` (BIGINT, FK → `usuarios.id`)
 - `evento_id` (BIGINT, FK → `eventos.id`)
- Entidad Votación (**`votaciones`**)
 - Atributos:
 - `id` (BIGINT, PK, AUTO_INCREMENT)
 - `titulo` (VARCHAR(255), NOT NULL)
 - `descripcion` (TEXT)
 - `grupo_id` (BIGINT, FK → `grupos.id`)

- `creador_id` (BIGINT, FK → `usuarios.id`)
- `fecha_creacion` (DATETIME, DEFAULT CURRENT_TIMESTAMP)
- `fecha_cierre` (DATETIME)
- `estado` (VARCHAR(20), NOT NULL, DEFAULT 'ACTIVA')
- **Tabla `votacion_opciones`**
 - `votacion_id` (BIGINT, FK → `votaciones.id`)
 - `opcion` (VARCHAR(255), NOT NULL)
- **Entidad Voto (`votos`)**
 - **Atributos:**
 - `id` (BIGINT, PK, AUTO_INCREMENT)
 - `votacion_id` (BIGINT, FK → `votaciones.id`)
 - `usuario_id` (BIGINT, FK → `usuarios.id`)
 - `opcion` (VARCHAR(255), NOT NULL)
 - `fecha_voto` (DATETIME, NOT NULL, DEFAULT CURRENT_TIMESTAMP)

2.3.2. Requisitos funcionales

Los requisitos funcionales definen las acciones que pueden realizar los usuarios dentro de la aplicación, ya sean a través de la interfaz o internamente por el sistema.

RF_01 - Registro de usuario

Descripción: La aplicación permitirá que un usuario cree una cuenta introduciendo su nombre, correo electrónico y contraseña.

RF_02 - Inicio de sesión

Descripción: Un usuario registrado podrá iniciar sesión introduciendo su email y contraseña.

RF_03 - Crear grupo de amigos

Descripción: Un usuario podrá crear un grupo nuevo y convertirse automáticamente en su administrador.

RF_04 - Unirse a grupo mediante código

Descripción: Los usuarios podrán unirse a un grupo existente introduciendo un código de invitación.

RF_05 - Crear evento

Descripción: Los miembros del grupo podrán crear eventos indicando título, descripción, lugar y fecha.

RF_06 - Confirmar asistencia a evento

Descripción: Cada usuario podrá confirmar si asistirá o no a un evento creado dentro del grupo.

RF_07 - Crear votación

Descripción: Un usuario podrá crear una votación introduciendo una pregunta y varias opciones de respuesta.

RF_08 - Votar

Descripción: Los miembros del grupo podrán seleccionar una opción en una votación activa.

RF_09 - Crear gasto

Descripción: Cualquier miembro podrá registrar un gasto indicando quién pagó y entre quiénes se divide.

RF_10 - Ver deudas

Descripción: La app calculará automáticamente qué miembro debe dinero a otro, según los gastos registrados.

RF_11 - Subir foto a la galería

Descripción: Los usuarios podrán subir imágenes asociadas a eventos y verlas en una galería común.

RF_12 - Crear nota o lista

Descripción: Los usuarios podrán crear notas con formato de texto libre o lista de tareas para compartir en grupo.

RF_13 - Editar datos del perfil

Descripción: Los usuarios podrán editar su nombre, email o contraseña desde su perfil.

RF_14 - Cerrar sesión

Descripción: Cualquier usuario podrá cerrar su sesión desde su perfil.

RF_15 - Eliminar cuenta

Descripción: Un usuario podrá eliminar su cuenta, eliminando todos sus datos y su acceso a la app.

2.3.3. Otros requisitos

Estos requisitos definen las condiciones mínimas necesarias para el correcto funcionamiento de la aplicación y la infraestructura que la soporta.

OR_01 - Sistema operativo compatible

Descripción: La aplicación estará disponible para dispositivos Android con versión mínima 7.0 (API 24).

OR_02 - Conectividad

Descripción: La app requiere conexión a Internet para acceder a datos, sincronizar eventos, votaciones y fotos.

OR_03 - Backend / BBDD

Descripción: El sistema podrá funcionar con una de estas:
Spring Boot + MySQL + almacenamiento en servidor propio

OR_04 - Navegador web (opcional)

Descripción: Se prevé en el futuro una versión web de Planora accesible desde navegadores modernos (Chrome, Edge, Firefox).

OR_05 - Herramientas de desarrollo

Descripción: El desarrollo se realizará con:

- **Frontend:** Ionic + Vue
- **Editor:** Visual Studio Code
- **Control de versiones:** GitHub (privado o público)
- **Base de datos:** MySQL

2.4. Descripción de la solución

La solución Planora se fundamenta en una arquitectura de tipo cliente-servidor, donde el backend se implementa con Spring Boot y expone una API REST que gestiona toda la lógica de negocio y la persistencia en una base de datos MySQL. El frontend es una aplicación híbrida desarrollada con Ionic y Vue.js, que consume los servicios REST del backend y ofrece una experiencia nativa en dispositivos Android e iOS. La comunicación entre cliente y servidor se realiza mediante JSON sobre HTTPS, y la gestión de estado en el cliente se apoya en Vuex para mantener la coherencia de datos en tiempo real. Este enfoque modular permite escalar cada capa de forma independiente y facilita el mantenimiento y la futura incorporación de nuevas funcionalidades.

2.5. Planificación del proyecto

2.5.1. Equipo de trabajo

Proyecto individual realizado por Agustín Luz Martín.

2.5.2. Planificación temporal

Fecha	Tarea
Abril	Diseño base, login, estructura de datos
Principios de mayo	Eventos, votaciones, gastos
Finales de mayo	Galería, notas, mejoras
Primera semana junio	Pruebas con grupo real, captura de errores
Mitad de junio	Entrega, presentación y memoria final

2.6. Estudio del coste del proyecto

- Herramientas: Gratuitas (Firebase, Spring Boot, VS Code, GitHub)
- Publicación: Gratuita (si es para uso interno)
- No se contemplan gastos de hosting o dominios

3. ANÁLISIS DEL SISTEMA DE INFORMACIÓN

3.1. Entorno tecnológico

Frontend

- **Ionic + Vue.js**
 - **Versión:** Ionic 6.x, Vue 3.x
 - **Justificación:** permite construir aplicaciones híbridas con aspecto nativo en Android e iOS a partir de un único código base web, acelerando el desarrollo y reduciendo costes de mantenimiento.
- **Vuex**
 - **Función:** gestión centralizada del estado de la aplicación, garantiza consistencia de datos entre vistas y componentes.

Backend

- **Spring Boot**
 - **Versión:** Spring Boot 3.x
 - **Justificación:** marco robusto para construir micro-servicios y APIs REST, con amplia comunidad y potente ecosistema (seguridad, monitorización, testing).
- **Spring Security + JWT**
 - **Función:** autenticar y autorizar peticiones a la API mediante tokens JSON Web Token, garantizando seguridad sin estado.

Base de datos

- **MySQL**
 - **Justificación:** sistema relacional maduro y ampliamente soportado que garantiza integridad referencial y alto rendimiento para operaciones transaccionales.

Comunicación cliente-servidor

- Protocolo **HTTPS** (TLS 1.2+)
- Formato de datos **JSON** para intercambiar información entre frontend y backend.

Herramientas de desarrollo

- **Editor y middleware:** Visual Studio Code con el plugin Ionic + Vue
- **Control de versiones:** Git en GitHub (ramas feature/bugfix siguiendo GitFlow).

3.2. Modelado de datos

3.2.1. Modelo entidad-relación

- **Usuario (usuarios)**
 - PK: `id`
 - Atributos: `email`, `nombre`, `foto_perfil`, `password`
 - Relaciones:
 - Pertenece a muchos **Grupos** (M-a-M vía `usuarios_grupos`)
 - Crea Eventos, Votaciones y Notas (1-a-N)
 - Participa en Gastos (M-a-M vía `gastos_usuarios`)
 - Genera y recibe DeudasGasto (1-a-N en `deudas_gastos`)
 - Asiste a Eventos (M-a-M en `evento_asistentes` y 1-a-N en `asistencias_eventos`)

- **Grupo (grupos)**

- PK: `id`
- Atributos: `codigo_invitacion`, `nombre`, `imagen_perfil`
- Relaciones:
 - Agrupa Usuarios (M-a-M en `usuarios_grupos`)
 - Contiene Eventos, Gastos, Notas, Votaciones e Imágenes (1-a-N)

- **UsuarioGrupo (usuarios_grupos)**

- PK: `id`
- Atributo: `rol`
- Relaciones: FK a `usuarios.id` y a `grupos.id` (tabla intermedia para Usuarios–Grupos)

- **Invitacion (invitaciones)**

- PK: `id`
- Atributos: `estado`, `fecha`
- Relaciones: FK a `usuarios.id` y a `grupos.id`

- **Evento (eventos)**

- PK: `id`
- Atributos: `descripcion`, `fecha`, `titulo`, `ubicacion`
- Relaciones:
 - FK a `grupos.id` y a `usuarios.id` (creador)
 - Asistentes (M-a-M en `evento_asistentes`)

- Gastos, Imágenes y AsistenciasEvento (1-a-N)

- **EventoAsistente (evento_asistentes)**

- PK: `id`
- Atributo: `asistio`
- Relaciones: FK a `eventos.id` y a `usuarios.id`

- **AsistenciaEvento (asistencias_eventos)**

- PK: `id`
- Atributo: `asistio`
- Relaciones: FK a `eventos.id` y a `usuarios.id`

- **Gasto (gastos)**

- PK: `id`
- Atributos: `monto`, `titulo`, `partes_iguales`, `fecha_creacion`
- Relaciones:
 - FK a `usuarios.id` (`pagado_por`), a `grupos.id` y a `eventos.id`
 - Participantes (M-a-M en `gastos_usuarios`)
 - Cantidades personalizadas (`cantidades_personalizadas`)
 - DeudasGasto (1-a-N en `deudas_gastos`)

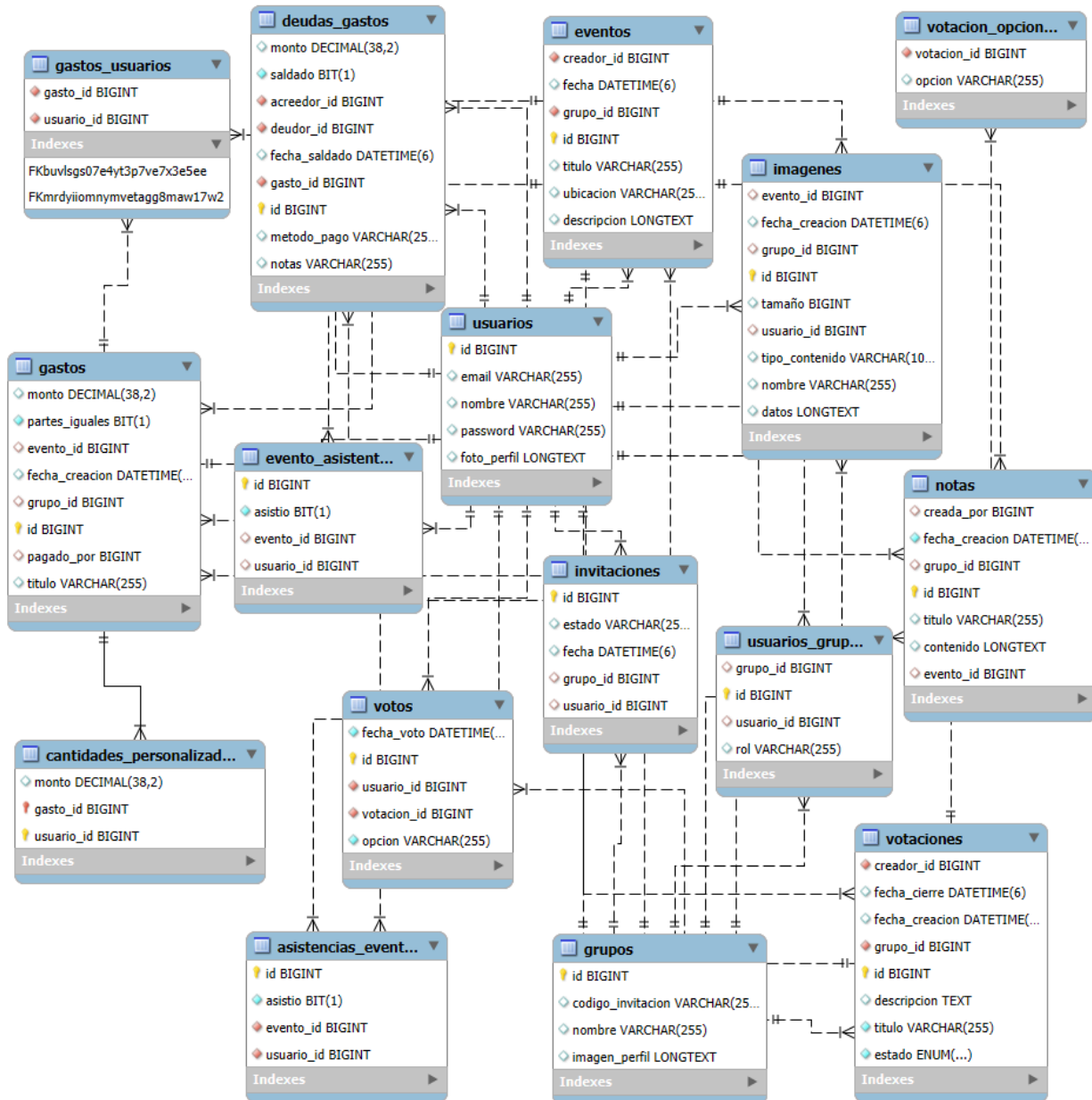
- **DeudaGasto (deudas_gastos)**

- PK: `id`
- Atributos: `monto`, `saldado`, `fecha_saldado`, `metodo_pago`, `notas`

- Relaciones: FK a `gastos.id`, a `usuarios.id` (deudor y acreedor)
- **Imagen (imagenes)**
 - PK: `id`
 - Atributos: `nombre`, `tipo_contenido`, `tamaño`, `datos`, `fecha_creacion`
 - Relaciones: FK a `eventos.id`, `usuarios.id` y `grupos.id`
- **Nota (notas)**
 - PK: `id`
 - Atributos: `titulo`, `fecha_creacion`, `contenido`
 - Relaciones: FK a `grupos.id`, a `usuarios.id` (creada_por) y a `eventos.id`
- **Votacion (votaciones)**
 - PK: `id`
 - Atributos: `titulo`, `descripcion`, `fecha_creacion`, `fecha_cierre`, `estado`
 - Relaciones: FK a `grupos.id`, a `usuarios.id` (creador); tiene Opciones y Votos
- **VotacionOpciones (votacion_opciones)**
 - PK compuesta: `votacion_id` + `opcion`
 - Relaciones: FK a `votaciones.id`
- **Voto (votos)**
 - PK: `id`
 - Atributos: `opcion`, `fecha_voto`

- Relaciones: FK a `votaciones.id` y a `usuarios.id`

3.2.2. Esquema de la base de datos



3.3 Identificación de los usuarios participantes y finales

En Planora interactúan cuatro tipos de agentes, cada uno con un nivel de acceso y unas responsabilidades específicas:

1. Usuario anónimo

- **Accede sin autenticar.**
- **Puede ver la pantalla de bienvenida, contenidos de ejemplo (capturas, descripciones) y los formularios de registro e inicio de sesión.**
- **No tiene acceso a grupos, eventos ni ninguna funcionalidad colaborativa.**

2. Usuario registrado (miembro de grupo)

- **Accede con su correo y contraseña, recibe un token JWT.**
- **Puede crear o unirse a grupos, gestionar eventos, votaciones, notas y gastos, confirmar asistencia, subir y consultar fotos, editar su perfil y cerrar sesión.**
- **Es el destinatario principal de todas las capacidades colaborativas de la aplicación.**

3. Administrador de grupo (subrol)

- **Es un usuario registrado que ha creado o ha recibido el rol “admin” en un grupo.**
- **Dispone de todos los permisos de un miembro, más la posibilidad de renombrar el grupo, generar o compartir códigos de invitación, invitar o expulsar participantes y eliminar el grupo.**

4. Sistema externo: API REST

- Backend implementado en Spring Boot que expone endpoints RESTful y aplica la lógica de negocio y las políticas de seguridad.
- Valida credenciales, protege rutas con JWT, administra la persistencia en MySQL y procesa la carga de imágenes y datos relacionales.

A continuación se describen los módulos de la API REST y, para cada uno de sus endpoints, se indica si está disponible para usuarios anónimos o requiere un usuario autenticado (con token JWT) y, en su caso, el rol mínimo necesario:

3.3.1 Autenticación y registro

- **POST /api/auth/registro** – disponible para usuarios anónimos
- **POST /api/auth/login** – disponible para usuarios anónimos
- **GET /api/auth/invitacion/{codigo}** – disponible para usuarios anónimos

3.3.2 Gestión de usuarios

(autenticado; sólo el propietario de la cuenta)

- **GET /api/usuarios/{id}**
- **PUT /api/usuarios/{id}**
- **POST /api/usuarios/{id}/logout**

3.3.3 Gestión de grupos e invitaciones (autenticado; acceso según rol de grupo)

- **GET** `/api/grupos/buscar-usuario?email=...` – miembro o administrador
- **GET** `/api/grupos/{id}` – sólo miembros
- **GET** `/api/grupos/usuario/{id}` – propio usuario
- **POST** `/api/grupos` – cualquier usuario autenticado
- **PUT** `/api/grupos/{id}` – administrador de grupo
- **DELETE** `/api/grupos/{groupId}/usuarios/{usuarioId}` – administrador de grupo
- **GET** `/api/grupos/{id}/usuarios` – miembros
- **GET** `/api/grupos/{groupId}/participantes-con-roles` – miembros
- **POST** `/api/grupos/{id}/usuarios` – administrador o invitado
- **POST** `/api/grupos/{groupId}/invitar?emailUsuario=...` – administrador
- **PUT**
`/api/grupos/{groupId}/usuarios/{usuarioId}/rol?nuevoRol=...` – administrador
- **PUT**
`/api/grupos/{groupId}/transferir-admin?nuevoAdminId=...` – administrador

- **DELETE** /api/grupos/{grupoId}/salir?usuarioId=... – miembro o administrador
- **GET** /api/grupos/{grupoId}/usuarios/{usuarioId}?solicitanteId=... – miembro
- **POST** /api/grupos/{id}/generar-codigo?usuarioId=... – administrador
- **GET** /api/grupos/{id}/estadisticas?usuarioId=... – miembro
- **GET** /api/grupos/{id}/estadisticas/usuarios?usuarioId=... – miembro
- **GET** /api/invitaciones/usuario/{usuarioId} – miembro
- **POST** /api/invitaciones/{id}/aceptar – invitado
- **POST** /api/invitaciones/{id}/rechazar – invitado

3.3.4 Gestión de eventos y asistencias

- **GET** /api/eventos/{grupoId}/eventos – anónimo
- **GET** /api/eventos/{eventoId} – anónimo
- **POST** /api/eventos/{grupoId}/crear – miembro de grupo
- **PUT** /api/eventos/{eventoId} – creador o administrador de evento
- **DELETE** /api/eventos/{eventoId} – creador o administrador de evento

- **POST /api/eventos/{eventoId}/asistencia** – miembro autenticado
- **GET /api/eventos/{eventoId}/asistentes** – anónimo
- **GET /api/eventos/{eventoId}/asistencia/estadisticas** – anónimo

3.3.5 Gestión de gastos y deudas

- **GET /api/gasto/{grupoId}/gastos** – anónimo
- **GET /api/gasto/{gastoId}** – anónimo
- **POST /api/gasto/{grupoId}/crear** – miembro de grupo
- **PUT /api/gasto/{gastoId}** – miembro autenticado
- **DELETE /api/gasto/{gastoId}** – miembro autenticado
- **GET /api/gasto/grupos/{gastoId}/participantes** – anónimo
- **POST /api/gasto/grupos/{gastoId}/participantes/{participanteId}/saldado** – miembro autenticado
- **GET /api/gasto/grupos/{grupoId}/eventos/{eventoId}** – anónimo
- **GET /api/gasto/grupos/{grupoId}/deudas** – anónimo
- **GET /api/gasto/{gastoId}/deudas** – anónimo

3.3.6 Gestión de notas

- **GET /api/nota/{grupoId}/notas** – anónimo
- **GET /api/nota/{notaId}** – anónimo
- **POST /api/nota/{grupoId}/crear** – miembro de grupo
- **PUT /api/nota/{notaId}** – propietario de la nota
- **DELETE /api/nota/{notaId}** – propietario de la nota
- **GET /api/nota/usuario** – usuario autenticado

3.3.7 Gestión de imágenes

- **GET /api/imagenes/grupo/{grupoId}** – anónimo
- **GET /api/imagenes/evento/{eventoId}** – anónimo
- **GET /api/imagenes/usuario/{usuarioId}** – anónimo
- **GET /api/imagenes/{id}** – anónimo
- **GET /api/imagenes/{id}/datos** – anónimo
- **POST /api/imagenes/subir** – usuario autenticado
- **DELETE /api/imagenes/{id}** – propietario de la imagen

3.3.8 Gestión de votaciones y votos

- **GET /api/votaciones/{id}** – anónimo
- **GET /api/grupos/{grupoId}/votaciones** – anónimo

- **POST /api/grupos/{grupoId}/votaciones** – miembro de grupo
- **PUT /api/votaciones/{id}** – creador de votación
- **PUT /api/votaciones/{id}/cerrar** – creador de votación
- **DELETE /api/votaciones/{id}** – creador de votación
- **POST /api/votaciones/{id}/votar** – miembro de grupo
- **GET /api/votaciones/{id}/resultados** – anónimo
- **GET /api/votaciones/{id}/mi-voto** – usuario autenticado (propio voto)

3.4 Identificación de subsistemas de análisis

El sistema general se descompone en cinco subsistemas lógicos, cada uno agrupando un conjunto de funcionalidades coherentes. A continuación se describen, para cada subsistema, su responsabilidad principal, los componentes clave y sus dependencias con el resto de módulos.

3.4.1 Subsistema de gestión de sesiones

- **Responsabilidad:** Controlar el ciclo de vida de la sesión de usuario (inicio, mantenimiento y cierre).
- **Componentes clave:**
 - `AuthController` (endpoints `/api/auth/login`, `/api/auth/logout`)
 - `JwtUtil` y filtros de seguridad JWT
- **Dependencias:**
 - Subsistema de gestión de usuarios (obtención de credenciales y datos de perfil)

3.4.2 Subsistema de gestión de usuarios

- **Responsabilidad:** Registrar, modificar y eliminar cuentas de usuario; gestionar unión a grupos y asignación de roles.
- **Componentes clave:**
 - `UserController` (endpoints `/api/usuarios/...`)
 - `UserService` y `UsuarioRepository`
 - Conversores DTO para transferencia de datos
- **Dependencias:**
 - Subsistema de gestión de sesiones (autenticación)
 - Todos los subsistemas funcionales que requieren autorizar acciones de usuario autenticado

3.4.3 Subsistema de gestión de grupo y contenido colaborativo

- **Responsabilidad:** Crear y administrar grupos; gestionar contenidos compartidos (eventos, votaciones, notas, galerías).
- **Componentes clave:**
 - `GrupoController` e `InvitacionController`
 - `EventoController`, `VotacionController`, `NotaController`, `ImagenController`
 - Servicios y repositorios correspondientes (`GrupoService`, `EventoService`, etc.)
- **Dependencias:**
 - Gestión de usuarios (para asignar creadores, miembros y roles)
 - Gestión de sesiones (para validar accesos)

3.4.4 Subsistema de gestión de gastos

- **Responsabilidad:** Registrar gastos compartidos, calcular y actualizar deudas entre usuarios.
- **Componentes clave:**
 - `GastoController`
 - `GastoService`, `DeudaGastoService` y sus repositorios
 - Lógica de cálculo y persistencia de deudas (`cantidades_personalizadas`, `deudas_gastos`)
- **Dependencias:**
 - Gestión de usuarios (identificación de pagadores y participantes)
 - Gestión de grupo y eventos (vinculación de gastos a un contexto)

3.4.5 Subsistema backend / API REST

- **Responsabilidad:** Centralizar la lógica de negocio, exponer servicios RESTful y garantizar la persistencia en MySQL.
- **Componentes clave:**
 - Spring Boot con controladores, servicios y repositorios JPA
 - Configuración de seguridad (Spring Security + JWT), manejo global de excepciones
 - Conversores DTO, filtros de CORS, logging y métricas
- **Dependencias:**
 - Todos los subsistemas funcionales interactúan con esta capa para realizar sus operaciones a través de peticiones HTTP

3.5 Establecimiento de requisitos

En este apartado se detallan los requisitos funcionales asociados a cada uno de los cinco subsistemas identificados, especificando las operaciones principales que debe soportar Planora.

3.5.1 Subsistema de gestión de sesiones

- **Iniciar sesión**
El sistema permitirá a un usuario registrado autenticarse mediante correo electrónico y contraseña, validando sus credenciales y generando un token JWT que mantenga su sesión activa.
- **Cerrar sesión**
El usuario podrá invalidar su token y cerrar la sesión actual, eliminando cualquier estado de autenticación en la aplicación.
- **Mantener sesión activa**
Mientras no se produzca un cierre de sesión explícito o se cierre la aplicación, el token JWT permanecerá válido y la sesión se mantendrá abierta para mejorar la experiencia de uso.

3.5.2 Subsistema de gestión de usuarios

- **Registro de usuario**
Permitir la creación de nuevas cuentas solicitando nombre, correo y contraseña; verificar la unicidad del correo y almacenar los datos cifrados.
- **Edición de perfil**
Ofrecer la posibilidad de modificar nombre, dirección de correo y contraseña desde la sección de perfil de usuario.
- **Eliminación de cuenta**
Posibilitar la baja definitiva de la cuenta, borrando los datos personales, desasociando al usuario de sus grupos y eliminando su contenido asociado.
- **Crear grupo**
Un usuario autenticado podrá generar un grupo nuevo y pasar a ser su administrador por defecto.
- **Unirse a grupo**
Facilitar la adhesión a un grupo existente mediante la introducción de un código de

invitación.

- **Ver miembros del grupo**
Permitir la consulta de la lista de usuarios que forman parte de un grupo determinado.

3.5.3 Subsistema de gestión de grupo y contenido colaborativo

- **Crear evento**
Permitir la definición de un evento asociado a un grupo, con título, descripción, ubicación y fecha, y vincularlo al creador.
- **Confirmar asistencia**
Facilitar que cada miembro marque su asistencia (“asistiré” / “no asistiré”) a un evento.
- **Crear votación**
Ofrecer la creación de encuestas internas introduciendo una pregunta y varias opciones de respuesta.
- **Participar en votación**
Garantizar que un usuario pueda emitir un único voto por encuesta y consultar los resultados.
- **Crear nota colaborativa**
Permitir la generación de notas de texto libre o listas de tareas compartidas, visibles y editables por todo el grupo.
- **Subir imagen a galería**
Habilitar la selección y envío de fotografías a la galería del grupo, opcionalmente ligadas a un evento concreto.
- **Consultar galería**
Facilitar la visualización ordenada de todas las imágenes subidas por los miembros del grupo.

3.5.4 Subsistema de gestión de gastos

- **Añadir gasto**
Registrar un gasto especificando título, importe total, usuario que pagó y lista de participantes entre los que se reparte.
- **Consultar deudas**
Calcular y mostrar el saldo deudor / acreedor de cada miembro, en función de los

gastos acumulados.

- **Marcar deuda como saldada**

Permitir al usuario indicar que ha abonado una deuda pendiente, actualizando el balance y eliminando la entrada del resumen de deudas.

3.5.5 Subsistema backend / API REST (opcional)

- **Procesar peticiones desde el frontend**

Exponer y validar todos los endpoints REST necesarios para registro, login, gestión de grupos, eventos, gastos, notas, imágenes y votaciones.

- **Acceso y gestión de base de datos**

Realizar operaciones CRUD en MySQL garantizando la integridad referencial y el correcto mapeo objeto-relacional mediante Spring Data JPA.

- **Aplicación de reglas de negocio**

Centralizar la lógica de autorización, cálculo de deudas, control de acceso por roles y cualquier otra regla operativa.

- **Comunicación segura**

Asegurar todas las interacciones por HTTPS y proteger las rutas sensibles mediante autenticación con JWT.

3.6 Diagramas de Análisis

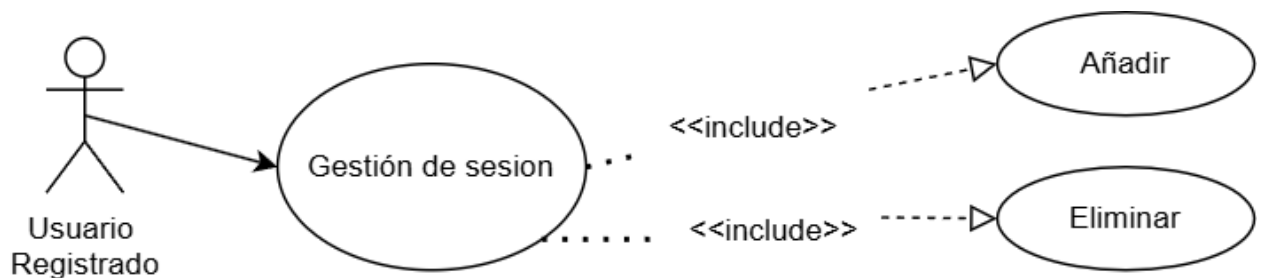
Durante la fase de análisis de Planora se han modelado los flujos de interacción entre los distintos actores y subsistemas mediante diagramas de casos de uso. A continuación se detallan los diagramas más relevantes y la forma de crearlos

3.6.1 Diagramas de Casos de Uso

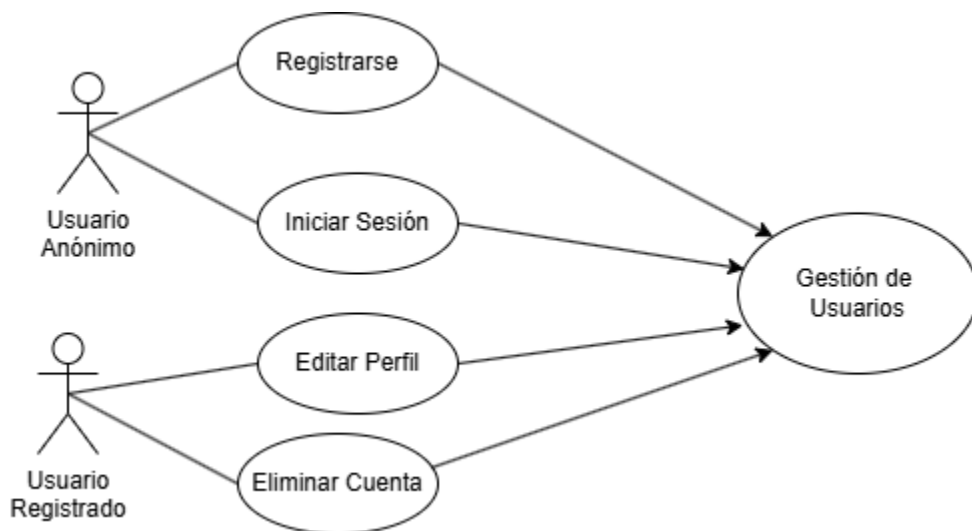
Este diagrama muestra a los tres actores principales —Usuario Anónimo, Usuario Registrado y Sistema Externo— y sus relaciones con los casos de uso globales de la aplicación: autenticación, gestión de grupos, eventos, gastos, notas, imágenes y votaciones.

Para mayor claridad, también se han representado los casos de uso organizados por subsistemas:

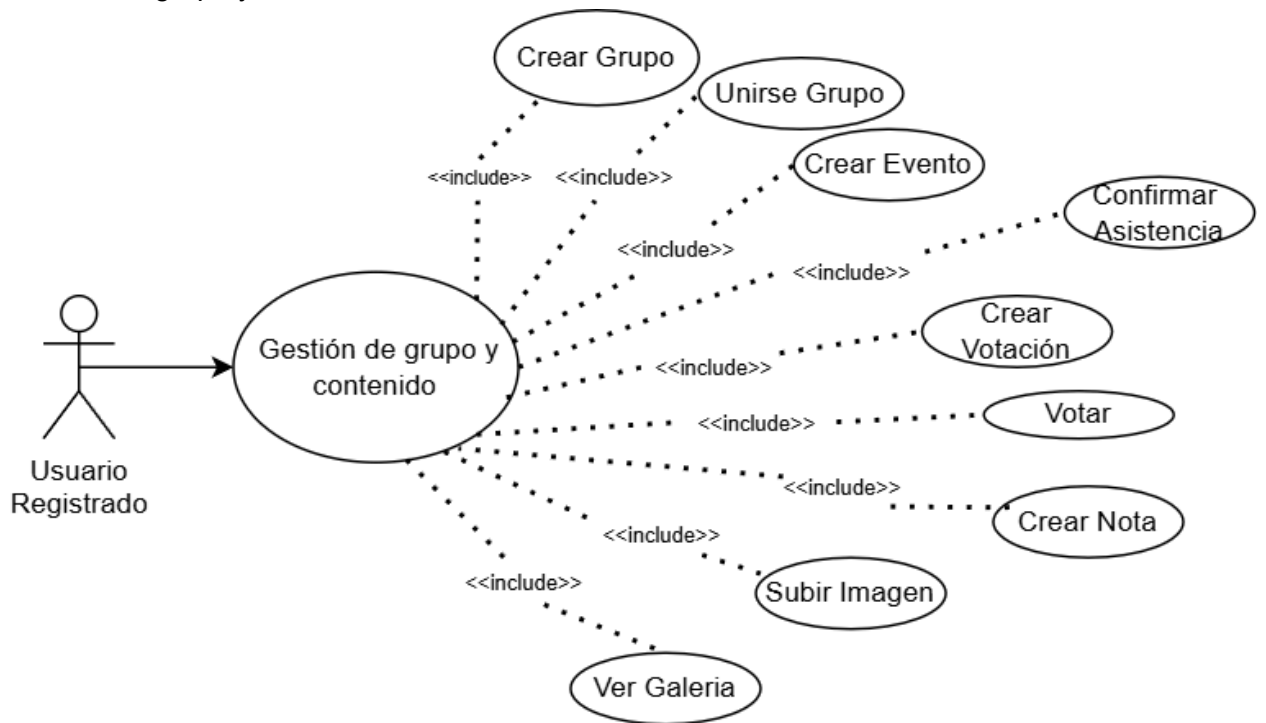
- Gestión de sesiones



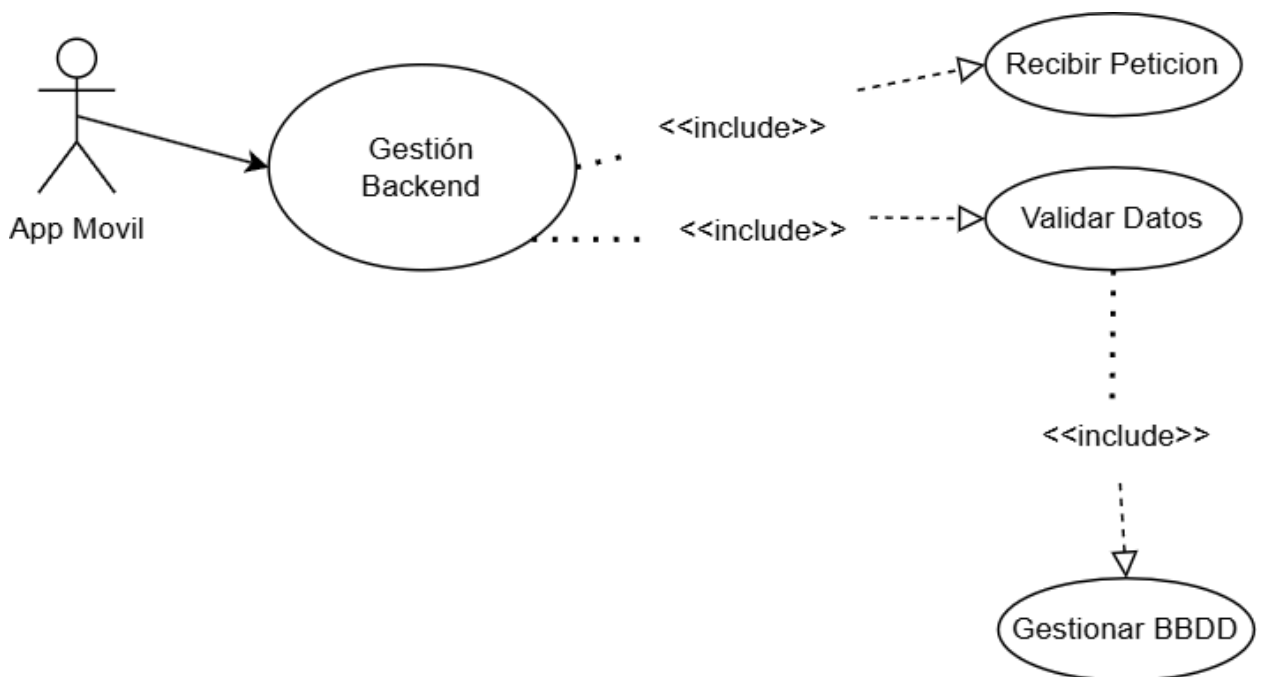
- Gestión de usuarios



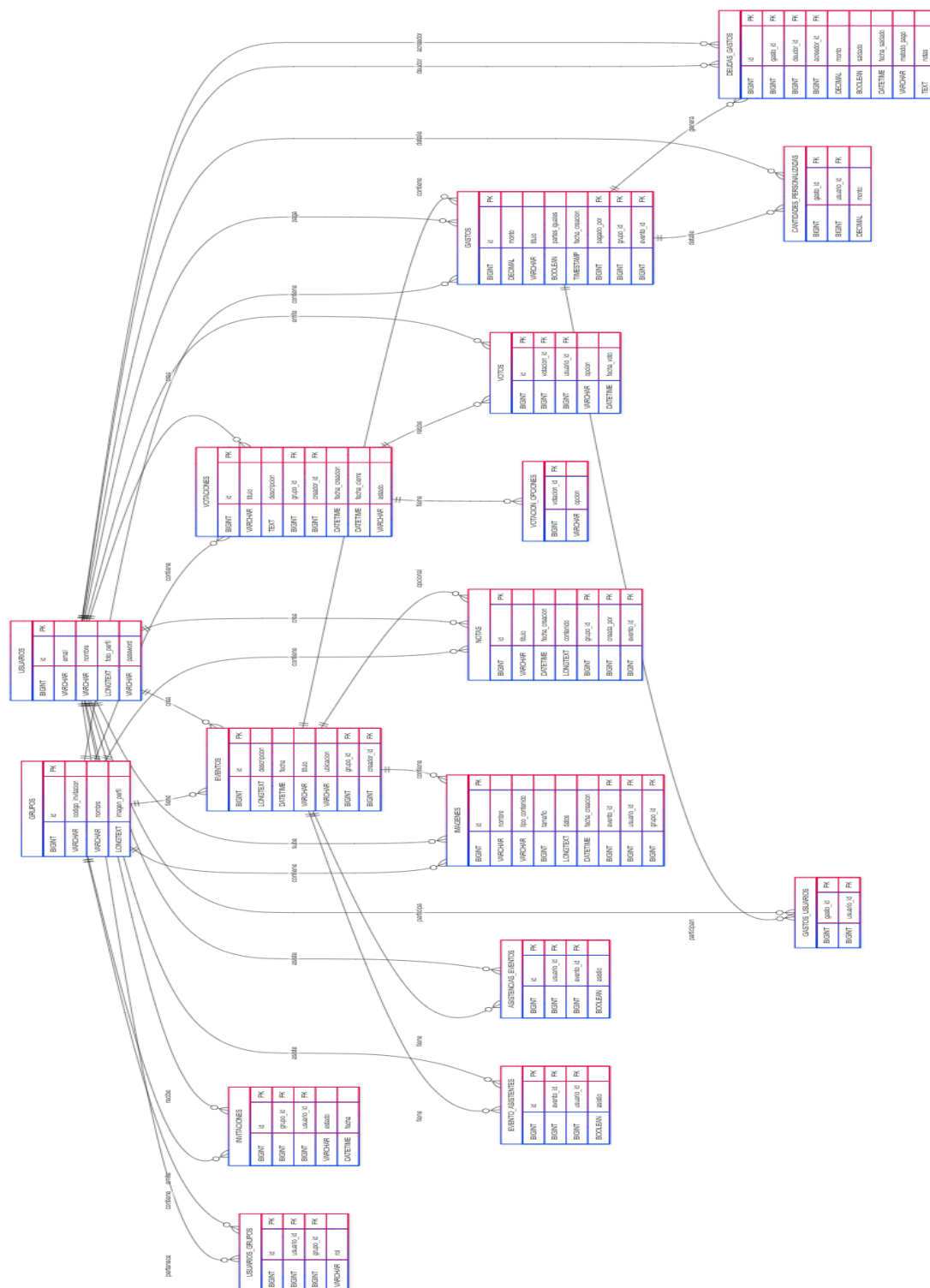
- Gestión de grupo y contenido colaborativo



- Backend/API



(MIRAR IMAGEN IMPORTADA DE RELACION DE TABLAS)



3.7.1 Especificación de principios generales de interfaz

La interfaz de usuario de **Planora** ha sido diseñada siguiendo principios de **claridad, coherencia y simplicidad**, enfocados a garantizar una experiencia de usuario intuitiva, cómoda y accesible para todos los perfiles.

Los principales principios de diseño aplicados son:

- **Diseño responsive y adaptable:**
La aplicación está pensada para dispositivos móviles, en orientación vertical, y adaptada a múltiples tamaños de pantalla. Se prioriza la legibilidad y la accesibilidad táctil.
- **Jerarquía visual clara:**
Se utiliza una estructura jerárquica basada en encabezados, secciones bien separadas y botones destacados, de forma que el usuario pueda identificar fácilmente las acciones principales en cada pantalla.
- **Consistencia visual:**
Se mantiene un sistema coherente de colores, tipografías, iconografía y distribución de elementos. Se aplican patrones repetibles en todas las vistas, como encabezados fijos, navegación inferior y botones flotantes para acciones principales.
- **Minimalismo funcional:**
Se evita la sobrecarga visual mostrando solo los elementos necesarios en cada contexto. Las acciones secundarias se agrupan en menús desplegables o contextuales.
- **Accesibilidad:**
Los textos, botones e iconos han sido diseñados con tamaños adecuados y contrastes correctos para garantizar una experiencia accesible. Las funciones clave están acompañadas de texto explicativo e iconos reconocibles.
- **Feedback inmediato:**
Cualquier acción del usuario (como guardar un dato, enviar un voto o añadir un gasto) va acompañada de una respuesta visual clara: mensajes de éxito, error o estados de carga.
- **Flujo de navegación fluido:**
La app permite moverse fácilmente entre pantallas mediante navegación inferior, enlaces contextuales y botones de retroceso, evitando bloqueos o confusión en la experiencia.

- **Uso de componentes estándar:**
Se emplean elementos nativos como cards, inputs, checkbox, modal, tabs y lists, siguiendo las guías de diseño de **Ionic Framework** y **Material Design**.

3.7.2 Especificación de formatos individuales de la interfaz de pantalla

Las siguientes pantallas componen la interfaz principal de la aplicación **Planora**. Cada una ha sido diseñada pensando en la simplicidad, claridad y eficiencia de uso, siguiendo principios visuales coherentes y adaptados a dispositivos móviles.

A continuación se detallan sus características principales:

Pantalla “Iniciar Sesión”

Ruta: /login

Acceso: Anónimo

1. Propósito

Autenticar al usuario y obtener un token JWT para acceder al resto de la aplicación.

2. Maquetación y componentes

```
<IonPage>

  └─ PageHeader(title="Iniciar Sesión", showMenu=false)

  └─ IonContent.fullscreen

    └─ div.login-background

      └─ div.login-card

        └─ img.logo

        └─ h1.title + p.subtitle

        └─ IonInput(label="Email", v-model="email", icon=mail-outline)

        └─ IonInput(label="Contraseña", v-model="password", icon=lock-closed-outline,
slot-end toggle eye-icon)

        └─ IonButton(Iniciar sesión) @click="login"

        └─ IonButton(Regístrate) @click="goToRegister"

        └─ IonButton(Restablécela) @click="goToReset"
```

```
└─ IonText.error-text (v-if="error")
```

3. Árbol de componentes / Bindings

- **Reactive refs:** email, password, showPassword, error
- **Métodos:**
 - login(): llama a auth.login(), maneja success o error.
 - toggleShowPassword(): alterna showPassword.
 - goToRegister(): router.push('/registro')
 - goToReset(): router.push('/restablecer')


4. Validaciones y estados

- **Formato email** validado en backend (error genérico si inválido).
- **Mostrar / ocultar contraseña** mediante icono de ojo.
- **Estados:**
 - **Normal:** inputs vacíos
 - **Cargando:** implícitamente con spinner de Ionic
 - **Error:** texto en rojo (error-text)

5. Flujo de navegación

- **Login exitoso** → router.push('/grupo')
- **"Regístrate"** → /registro
- **"Restablécela"** → /restablecer


Iniciar Sesión




Bienvenido a Planora


Organiza, vota y comparte con tu grupo

Email



Contraseña





INICIAR SESIÓN

¿NO TIENES CUENTA? REGÍSTRATE

¿OLVIDASTE TU CONTRASEÑA?

RESTABLÉCELA

Pantalla “Registro”

Ruta: /registro

Acceso: Anónimo

1. Propósito

Dar de alta a nuevos usuarios, validando email y complejidad de contraseña.

2. Maquetación y componentes

```
<IonPage>

  └─ PageHeader(title="REGISTRO", showMenu=false)

  └─ IonContent.fullscreen
    └─ div.register-background
      └─ div.register-card
        └─ img.logo
        └─ h1.title + p.subtitle
        └─ IonInput(label="Nombre", v-model="nombre", icon=person-outline)
        └─ IonInput(label="Email", v-model="email", icon=mail-outline)
        └─ IonInput(label="Contraseña", v-model="password", icon=lock-closed-outline,
toggle eye-icon)
        └─ IonButton(Registrarse) @click="registrar"
        └─ IonButton(Inicia sesión) @click="goToLogin"
      └─ IonText.error-text (v-if="error")
```

3. Árbol de componentes / Bindings

- **Reactive refs:** nombre, email, password, error
- **Métodos:**

- registrar():
 1. Comprueba regex de contraseña (mín. 8 caracteres, mayúscula y número).
 2. Llama a auth.register().
 3. En éxito guarda en localStorage y router.push('/grupo').
- goToLogin(): /login
- toggleShowPassword()


4. Validaciones y estados

- **Contraseña** validada en cliente con regex.
- **Toast** para avisos (toastController) de éxito/fracaso.
- **Error** en error-text para excepciones.

5. Flujo de navegación

- **Registro exitoso** → /grupo
- **“Inicia sesión”** → /login

REGISTRO




Crea tu cuenta

Únete a tu grupo y empieza a organizar

Nombre

Email

Contraseña



Registrarse

[¿Ya tienes cuenta? Inicia sesión](#)

Pantalla “Restablecer Contraseña”

Ruta: /restablecer

Acceso: Anónimo

1. Propósito

Permitir al usuario actualizar su contraseña cuando la ha olvidado.

2. Maquetación y componentes

```
<IonPage>

  |— PageHeader(title="Restablecer Contraseña", showMenu=false)

  |— IonContent.fullscreen

    |— div.reset-background

      |— div.reset-card

        |— img.logo

        |— h1.title + p.subtitle

        |— IonInput(label="Email", v-model="email", icon=mail-outline)

        |— IonInput(label="Nueva contraseña", v-model="password",
icon=lock-closed-outline, toggle eye-icon)

        |— IonButton(Restablecer contraseña) @click="reset"

        |— IonButton(Volver al inicio de sesión) @click="goToLogin"

        |— IonText.error-text (v-if="error")

        |— IonText.success-text (v-if="mensaje")
```

3. Árbol de componentes / Bindings

- **Reactive refs:** email, password, error, mensaje
- **Métodos:**
 - reset():

1. Valida regex de contraseña.
2. Llama a `auth.resetPassword()`.
3. En éxito muestra mensaje y navega a `/login`.
 - `goToLogin()`


4. Validaciones y estados

- **Toast** para feedback de validación y resultado.
- **Error** y **Success** messages en pantalla.

5. Flujo de navegación

- **Reset exitoso** → `/login`
- **“Volver al inicio de sesión”** → `/login`

Restablecer Contraseña



Recuperar acceso

Ingresa tu email y una nueva contraseña

Email

Nueva contraseña

☐

Restablecer contraseña

[Volver al inicio de sesión](#)

Pantalla “Invitaciones”

Ruta: /invitaciones

Acceso: Autenticado

1. Propósito

Mostrar al usuario todas las invitaciones a unirse a grupos y permitir aceptar o rechazar cada una.

2. Maquetación y componentes

```
<IonPage>
```

```
  |— PageHeader(title="Invitaciones", showMenu=true)
```

```
  |— IonContent.ion-padding
```

```
    |— IonList (v-if="invitaciones.length")
```

```
      |   |— IonItem* (v-for="inv in invitaciones")
```

```
        |       |— IonLabel.ion-text-wrap
```

```
          |       |   |— <h2>{{ inv.grupoNombre }}</h2>
```

```
          |       |   |— <p>{{ inv.fecha }} - {{ inv.estado }}</p>
```

```
          |       |— IonButton(Aceptar) v-if estado=PENDIENTE @click="aceptar(inv.id)"
```

```
          |       |— IonButton(Rechazar) v-if estado=PENDIENTE @click="rechazar(inv.id)"
```

```
    |— div.ion-text-center (v-else)
```

```
      |— <p>No tienes invitaciones pendientes.</p>
```

3. Árbol de datos / Bindings

- **Reactive refs:**

- invitaciones: Invitacion[]

- **Servicios:**

- invitacionService.listar(usuario.id)

- `invitacionService.aceptar(id)`
- `invitacionService.rechazar(id)`

4. Estados y validaciones

- **Lista vacía** → mensaje “No tienes invitaciones pendientes.”
- **Carga inicial** en `onMounted(cargar)` → poblado de invitaciones
- **Acción Aceptar/Rechazar:** refresca lista tras éxito.

5. Flujo de navegación

- Ninguna ruta secundaria. Botones ejecutan acciones y permanecen en la misma vista.



Pantalla “Mis Grupos”

Ruta: /grupo**Acceso:** Autenticado

1. Propósito

Mostrar al usuario los grupos de los que forma parte, permitir seleccionar uno, crear nuevos o unirse a otro.

2. Maquetación y componentes

```

<IonPage>
  └─ IonHeader
    |   └─ IonToolbar (color=primary)
    |       └─ IonMenuButton
    |       └─ IonTitle("Mis Grupos")
    |       └─ IonButtons.slot="end"
    |           └─ IonButton(icon=person) @click="/perfil"
    |           └─ IonButton(icon=mailOutline) @click="/invitaciones"
    |           └─ IonButton(icon=logOutOutline) @click="logout"
    └─ IonContent.fullscreen.content-bg
        └─ ActiveGroupBanner(grupo=grupoActivo)
        └─ div.container-content
            |   └─ GroupList (v-if grupos.length>0)
            |       |   └─ props: grupos, activo-id, is-admin
            |       |   └─ @select="entrarEnGrupo"
            |       └─ NoGroups (v-else)
            |           └─ v-model:code, loading, error
            |           └─ @create="abrirModalCrear", @join="unirseGrupo"
            └─ IonFab (v-if grupos.length>0)
                └─ IonFabButton(icon=add) @click="abrirModal"
                └─ CreateGroupModal(v-model:name, preview... @create="crearGrupo")

```

```
|— JoinGroupModal(v-model:code... @join="unirseGrupo")  
|— IonLoading(is-open=cargando)
```

3. Árbol de datos / Bindings

- **Reactive refs:** usuario, grupos, cargando, error, codigo, modalAbierto, modalCrearGrupo, nombreGrupo, imagenGrupo, previsualizacionImagen
- **Servicios:**
 - groupService.getUserGroups(usuario.id)
 - groupService.createGroup({ nombre, imagenPerfil })
 - invitacionService.obtenerGrupoPorCodigo(codigo) + groupService.registerUserInGroup(...)

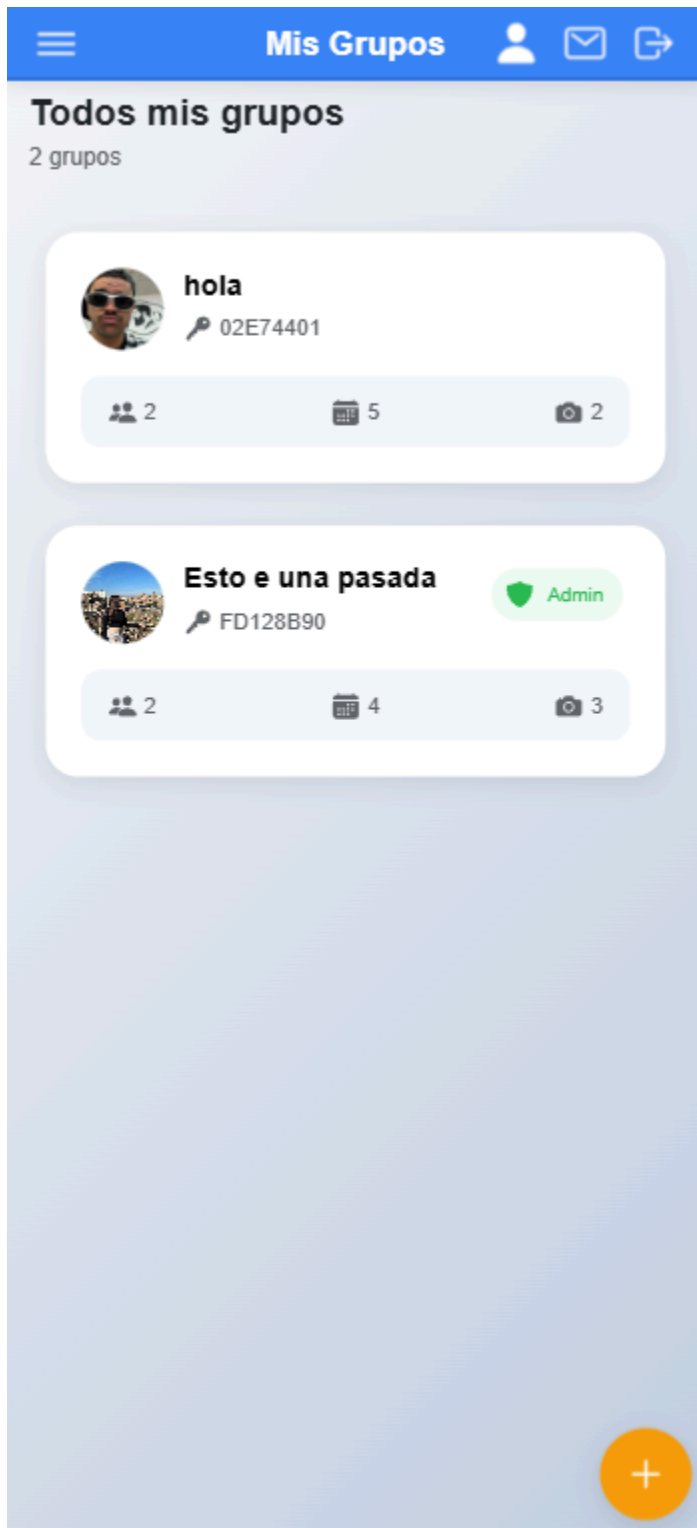
4. Estados y validaciones

- **Carga inicial** en onMounted(cargarGrupos)
- **Sin grupos** → componente NoGroups muestra formulario de código o crear grupo
- **Crear / Unirse** → valida nombre o código, muestra error inline
- **Logout** → limpia localStorage y redirige a /login

5. Flujo de navegación

- **Entrar en grupo** → /dashboard/{grupoid}
- **Crear grupo** → abre modal CreateGroupModal
- **Unirse a grupo** → abre modal JoinGroupModal

- **Perfil / Invitaciones / Logout** → rutas respectivas



Pantalla “Dashboard”

Ruta: /dashboard/:grupold

Acceso: Autenticado (miembro de grupo)

1. Propósito

Ofrecer un panel global con estadísticas, calendario de eventos y accesos rápidos a acciones (crear evento, crear gasto, ver participantes...).

2. Maquetación y componentes

```
<IonPage>

  └─ PageHeader(title=store.grupo.nombre, showMenu=true)

  |   └─ template#end

  |       └─ IonButton(icon=settingsOutline) @click="goToConfig"

  |       └─ IonButton(icon=logOutOutline) @click="goToLogout"

  └─ IonContent

      └─ IonCard.dashboard-section (Estadísticas)

          |   └─ IonCardHeader → title

          |   └─ IonCardContent

          |       └─ StatsGrid(membersCount, upcomingCount)

          |       └─ IonSkeletonText (v-if loading)

      └─ IonCard.dashboard-section (Calendario y Eventos)

          |   └─ IonCardHeader → title

          |   └─ IonCardContent.calendar-grid

          |       └─ CompactCalendar(@prev-month, @next-month, @select-day)

          |       └─ div.events-panel

          |           └─ EventsList(events, limit=4)

          |           └─ IonButton(Crear evento este día) @click="createEventOnDay"

      └─ IonCard.dashboard-section (Participantes y Acciones)
```

```

|   |— IonCardHeader → title
|   |— IonCardContent
|       |— ParticipantGrid(participants)
|       |— QuickActions()
|— IonFab (fixed) → IonFabButton(icon=add) @click="showFabSheet = true"
|— IonActionSheet(is-open=showFabSheet, buttons=fabButtons)
    |— Crear Evento → createEvent()
    |— Crear Gasto → createGasto()
    |— Cancel
|— EventModal(v-if=selectedEvent, @edit-event, @delete-event)
|— UserStatsModal(abierto=showStatsModal)

```

3. Árbol de datos / Bindings


- **Stores / Composables:**
 - useDashboardStore() → store.grupo, store.participantes, store.eventos, store.loading
 - useCalendar() → calendarDays, eventDates, etc.
- **Reactive refs:** selectedEvent, showStatsModal, showFabSheet
- **Computed:** currentMonthYear, upcomingEventos, daySelected, eventsToShow
- **Buttons:** fabButtons con handlers createEvent, createGasto

4. Estados y validaciones


- **Loading** → IonSkeletonText en lugar de contenido real
- **Eventos pasados vs futuros** decididos en upcomingEventos
- **Modales** se abren/cierran cambiando refs


5. Flujo de navegación

- **Ver detalle evento** → `router.push('asistir-evento', { eventold, grupold })`
- **Editar/Eliminar evento** desde EventModal
- **Configuración de grupo** → `/configuracion/{grupold}`
- **Logout** → limpia localStorage y `/login`

 Esto e una pasada

Estadísticas del Grupo

 2
Miembros

 3
Próximos

Calendario y Eventos


Lun	Mar	Mié	Jue	Vie	Sáb	Dom
< junio de 2025 >						
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6

14 jun

Cena De Grupo
BBY

> >

Crear evento este día



Pantalla “Configuración de Grupo”

Ruta: /configuracion/:grupold

Acceso: Autenticado (miembro/admin)

1. Propósito

Permitir ver y modificar los datos del grupo activo, gestionarlo (roles, invitaciones, estadísticas) y ejecutar acciones de administración (generar nuevo código, transferir admin, salir del grupo).

2. Maquetación y componentes

```
<IonPage>

  └─ PageHeader(title="Configuración del Grupo", showMenu=true)

  └─ IonContent

    └─ div.loading-container (v-if="loading")

      └─ IonSpinner

      └─ <p>Cargando información del grupo...</p>

    └─ div.settings-container (v-else)

      └─ BasicGroupInfo

        └─ props: grupo, isAdmin

        └─ @update-group, @generate-code

      └─ GroupStats

        └─ props: stats

        └─ @refresh, @show-users

      └─ ParticipantManagement

        └─ props: grupo-id, isAdmin, participants, stats

        └─ @refresh-participants

      └─ GroupActions

        └─ props: isAdmin, grupo-id, participants

        └─ @transfer-admin, @leave-group
```

```
└─ UserStatsModal(abierto=showUserStats, stats=statsUsuarios)
```

3. Árbol de datos / Bindings

- **Reactive refs / reactive:**
 - grupold, grupo, groupStats, participants
 - loading, showUserStats, statsUsuarios
- **Computed:**
 - isAdmin (basado en participants y usuario actual)
- **Servicios / llamadas:**
 - groupService.getGroupById(grupold)
 - groupService.getGroupStats(grupold)
 - groupService.getParticipantsWithRoles(grupold)
 - groupService.updateGroup(grupold, data)
 - groupService.generateInviteCode(grupold)
 - groupService.transferAdmin(grupold, newAdminId)
 - groupService.leaveGroup(grupold)
 - dashboardService.getUsuarioStats(grupold, userId)

4. Estados y validaciones

- **loading:** muestra spinner hasta completar las tres peticiones iniciales
- **Errores de red o datos faltantes:** toast de error y redirección a /grupo
- **Acciones sensibles (transferir admin, salir):** muestran alertController para confirmar
- **Actualización de grupo:** refresca vista y muestra toast de éxito o fallo

5. Flujo de navegación

- Tras “Salir del grupo” → redirige a /grupo
- Si no puede identificar grupold válido → toast y /grupo
- Modales (UserStatsModal) se abren/cieran con reactive refs

**Configuración del Gru...**

Información del Grupo



Cambiar

Nombre del Grupo

Esto e una pasada

Código de Invitación 

FD128B90

 Editar Información

 Generar Nuevo Código

Estadísticas



2

Participantes



1

Administradores



4

Eventos



5

Gastos

Agustín Luz Martín

63

Pantalla “Editar Perfil”

Ruta: /perfil

Acceso: Autenticado (propietario)

1. Propósito

Permitir al usuario ver y editar su información personal (nombre, email, foto) y cambiar su contraseña de forma segura.

2. Maquetación y componentes

```
<IonPage>

└─ PageHeader(title="Editar Perfil", showMenu=true)

└─ IonContent.fullscreen.profile-content
    └─ div.profile-container
        └─ div.avatar-section
            └─ div.avatar-wrapper @click="ampliarImagen"
                └─ IonAvatar.main-avatar → 
                └─ div.upload-controls → <label.upload-btn><input type="file"
@change="onFileChange">Cambiar foto</label>

└─ IonCard.info-card (Información Personal)
    └─ IonCardHeader → title + icon
    └─ IonCardContent (v-if usuarioCargado)
        └─ <form.profile-form>
            └─ IonItem.input-field → IonInput(v-model="formData.nombre")
            └─ IonItem.input-field → IonInput(v-model="formData.email")

└─ IonCard.security-card (Cambiar Contraseña)
    └─ IonCardHeader → title + subtitle
    └─ IonCardContent (v-if usuarioCargado)
        └─ IonItem → currentPassword + toggle show/hide
        └─ IonItem → new password + toggle
```

```

|           └─ IonItem → confirm password + toggle
|
|─ div.action-buttons
|
|   └─ IonButton.save-button @click="actualizarPerfil" :disabled="loading"
|
|   └─ IonButton.cancel-button @click="cancelar" :disabled="loading"
|
└─ IonModal(showImageModal) → muestra imagen ampliada

```

3. Árbol de datos / Bindings


- **reactive formData:** { nombre, email, fotoPerfil, currentPassword, password }
- **refs:** confirmPassword, loading, showImageModal, showCurrentPassword, showNewPassword, showConfirmPassword, usuarioCargado
- **Servicios:**
 - usuarioService.getByld(usuariold)
 - usuarioService.update(usuariold, datosActualizacion)

4. Estados y validaciones

- **Carga de datos** en onMounted(cargarDatosUsuario) → muestra skeleton hasta terminar
- **Validación previa a guardar:**
 - Nombre/email no vacíos
 - Email en formato correcto
 - Si cambia contraseña: comprueba campo actual, formato nuevo (passwordRegex), no igual a anterior, coincide con confirmación
- **Errores:** toast de error (posición top o bottom según contexto)
- **Éxito:** toast “Perfil actualizado correctamente” y router.back()
- **Cancelar:** alertController para confirmar pérdida de cambios

5. Flujo de navegación

- **Volver atrás** tras guardar o cancelar → `router.back()`
- **Si falla obtención de usuario** → toast y redirige a `/login`



Editar Perfil



 Cambiar foto

 **Información Personal**

Nombre completo

agustin

Correo electrónico

agustinluz@gmail.com

DEJA LOS CAMPOS VACÍOS SI NO DESEAS CAMBIAR TU CONTRASEÑA

 **Cambiar Contraseña**

Contraseña actual



Nueva contraseña



Pantalla “Eventos del Grupo”

Ruta: /dashboard/:grupold/eventos

Acceso: Autenticado (miembro)

1. Propósito

Mostrar la lista completa de eventos del grupo, filtrar por texto o fecha, refrescar la lista, y permitir crear/editar/borrar eventos.

2. Maquetación y componentes

```
<IonPage>

  └─ IonHeader

    └─ IonToolbar(color="primary")

      └─ IonBackButton(default-href=`/dashboard/${grupoId}`)

      └─ IonTitle "Eventos del Grupo"

      └─ IonButton(add) @click="irCrearEvento"

  └─ IonContent.ion-padding

    └─ IonRefresher @ion-refresh="manejarRefresh"

    └─ EventoFiltros(v-model:filtro-texto, v-model:filtro-fecha) @filtrar

    └─ IonSpinner(v-if="loading")

    └─ IonList(v-if="!loading && eventosFiltrados.length")

      └─ EventoCard * v-for="evento in eventosFiltrados"

    └─ div.empty-state(v-if="!loading && !eventosFiltrados.length")

    └─ EventoModal(is-open=modalAbierto...) @guardar @cerrar

    └─ IonToast(is-open=toast.mostrar...)

    └─ IonAlert(is-open=alertEliminar.mostrar...)
```

3. Datos y bindings

- **Composables:** useEventos(grupold, token) →
 - eventos, eventosFiltrados (computed),
 - loading, guardando,
 - filtroTexto, filtroFecha,
 - cargarEventos(), crearEvento(), actualizarEvento(), eliminarEvento()
- **Local refs:**
 - modalAbierto, eventoEditando
 - toast = { mostrar, mensaje, color }
 - alertEliminar = { mostrar, evento }

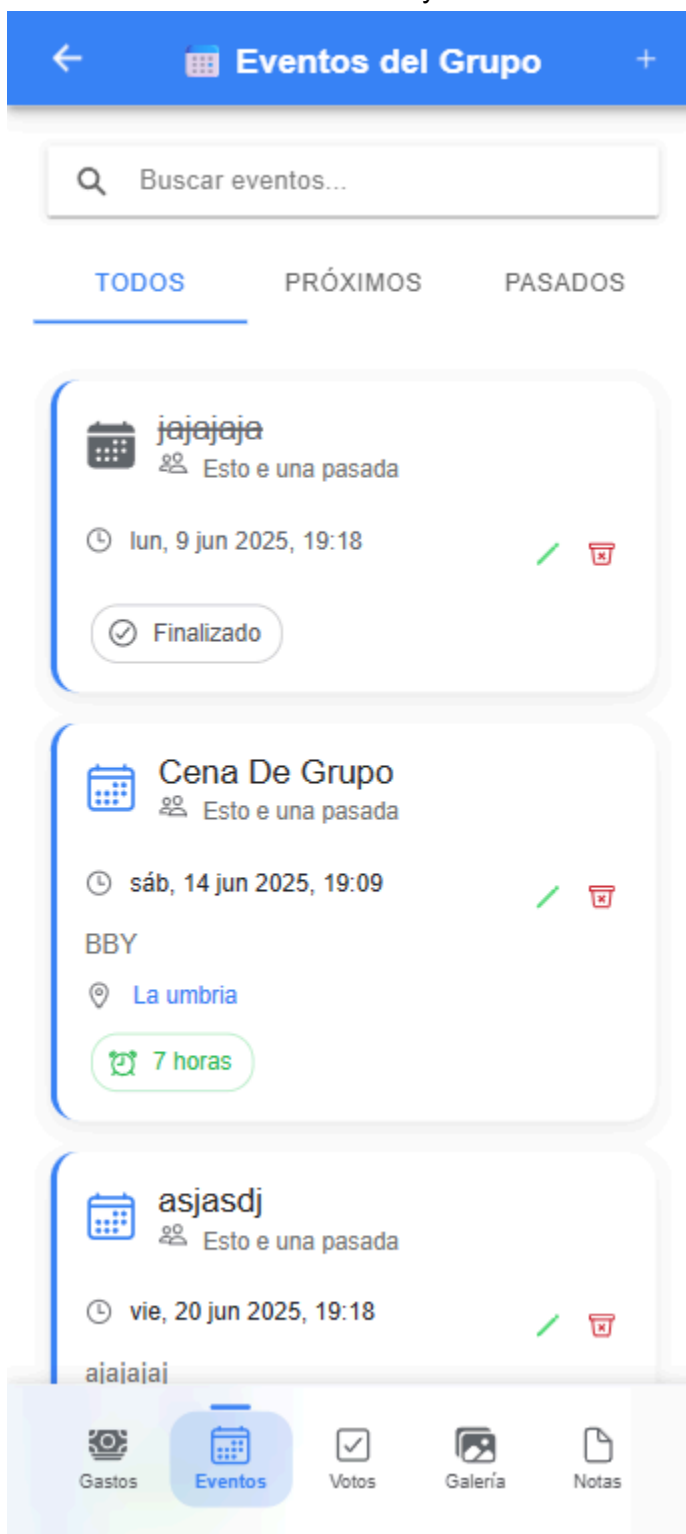
4. Estados y validaciones

- loading: spinner mientras carga
- refresher: completa con ev.target.complete()
- Empty state: distinto mensaje si no hay eventos vs. no resultados de filtro
- Confirmación de borrado con IonAlert
- Modal de crear/editar reutilizable
- Toast de éxito/error en cada operación

5. Flujo de navegación

- “+” → nueva ruta /dashboard/\${grupold}/crear/evento
- Back → vuelve al Dashboard

- Editar/Eliminar abre modal/alert y tras acción refresca lista



Pantalla “Galería de Imágenes”

Ruta: /dashboard/:grupold/imagenes

Acceso: Autenticado (miembro)

1. Propósito

Ver todas las imágenes subidas al grupo, filtrar por evento, previsualizar, descargar o eliminar (si es propio), y navegar a subir nueva imagen.

2. Maquetación y componentes

<IonPage>

```

└─ IonHeader(translucent)
  |   └─ IonToolbar
  |     |   └─ IonBackButton(default-href=`/dashboard/${grupoId}`)
  |     |   └─ IonTitle "Imágenes del Grupo"
  |     |   └─ IonButton(add) @click="irACrearImagen"
  └─ IonContent.fullscreen
    └─ IonCard.context-card → contexto (chips grupo/usuario)
      └─ IonItem → IonSelect(v-model="filtros.eventoId")
        └─ div.loading-container(v-if="estado.cargando")
          └─ div.estado-vacio(v-if="!cargando && imagenes.length===0")
            └─ div.gallery-container(v-if="!cargando && imagenes.length>0")
              └─ IonCard.gallery-header → título + count
                └─ IonGrid → IonRow → IonCol * v-for="imagen in imagenes"
                  └─ IonCard.imagen-card →
                    └─ div.imagen-container → 
                      └─ ion-overlay buttons (ver, descargar, borrar)
                    └─ IonModal(is-open=modalVistaAbierto) → vista completa + meta
                    └─ IonAlert + IonToast para confirmaciones y errores

```


3. Datos y bindings

- **refs/reactive:**
 - imagenes, eventosDisponibles, contexto = { grupold, usuariold, grupoNombre }
 - filtros.eventold, estado.cargando
 - imagenCompletaSeleccionada, modalVistaAbierto, cargandoImagenCompleta
 - cacheImágenesCompletas (Map)
- **Servicio/API:**
 - imageService.getByGrupo(grupold)
 - imageService.getFullImage(id)
 - imageService.deleteImage(id)
 - EventosService.obtenerEventosGrupo(grupold)
 - groupService.getGroupById(grupold)

4. Estados y validaciones

- Carga inicial: spinner
- Empty state con llamada a subir primera
- Filtrado en onEventoChange recarga imágenes
- Overlay en hover para acciones
- Solo propio puede eliminar (puedeEliminar)
- Descarga genera <a download> con Base64
- Toasts/alerts para confirmación y errores

5. Flujo de navegación

- “+” → /dashboard/\${grupold}/crear/imagen
- Back → Dashboard
- Modal completa → cierra con botón o al eliminar la imagen en vista

Pantalla “Gastos del Grupo”

Ruta: /dashboard/:grupold/gastos

Acceso: Autenticado (miembro)

1. Propósito

Listado de gastos del grupo, ver estadísticas, filtrar por pendientes/saldados/todos, crear nuevo gasto, ver detalle, editar, eliminar o marcar como saldado.

2. Maquetación y componentes

```
<IonPage>

└─ GastosHeader(grupo, cantidad, total, grupoId) @abrirOpciones

└─ IonContent.ion-padding

    └─ div.loading-container(v-if="cargando")

    └─ div(v-else)

        └─ GastosStats(total, cantidad, pendientes)

        └─ GastosSegment(total, pendientes, saldados) @filtrar

        └─ GastosLista(v-if="gastosFiltrados.length") @verDetalles

        └─ GastosEmpty(v-else) @crear

    └─ GastoDetalleModal(abierto, gastoSeleccionado) @editar @eliminar @marcarSaldado

    └─ IonActionSheet(is-open=mostrarOpciones) → resumen, crear, cancelar

    └─ GastosFab @crear
```

3. Datos y bindings

- **Composable:** useGastos(grupold) →
 - gastos, gastoSeleccionado, cargando, filtro, totalGastos, gastosPendientes, gastosSaldados, gastosFiltrados
 - cargarGastos(), seleccionarGasto(), marcarSaldado(), eliminarGasto()

- **Local refs:** mostrarOpciones, modalAbierto

4. Estados y validaciones

- cargando: spinner inicial
- Segment para filtrar lista
- Modal detalle maneja editar, eliminar o marcar saldado
- Action sheet con opción de ver resumen o crear
- Empty state con botón para añadir gasto

5. Flujo de navegación

- “Crear” → /dashboard/\${grupold}/crear/gasto
- Detalle modal abre/actualiza o marca saldado
- Al cerrar modal o acción, refresca lista

←

Grupo

5 gastos • 99,00 €

99,00 €

Total gastado

5

Gastos

0

Pendientes

TODOS (5)

PENDIENT...

SALDADO...

jsajasdj

agustin • Invalid Date

22,00 €

Saldado

jasdjasjd

jose • Invalid Date

23,00 €

Saldado

ajajaja

jose • Invalid Date

22,00 €

Saldado

jsd

jose • Invalid Date

22,00 €

Saldado

asdj

jose • Invalid Date

10,00 €

Saldado

Gastos

Eventos

Votos

Galería

Notas

Agustín Luz Martín

76

Pantalla “Notas del Grupo”

Ruta: /dashboard/:grupold/notas

Acceso: Autenticado (miembro)

1. Propósito

Ver todas las notas colaborativas del grupo, buscar por texto, filtrar por evento, crear, editar, ver en detalle o eliminar notas.

2. Maquetación y componentes

```
<IonPage>

  └─ IonHeader

    └─ IonToolbar

      └─ IonBackButton(default-href="/grupo")

      └─ IonTitle "Notas del Grupo"

      └─ IonButton(add) @click="irACrearNota"

  └─ IonContent

    └─ div.loading-container(v-if="cargando")

    └─ div(v-else)

      └─ IonSearchbar(v-model="busqueda")

      └─ IonItem → IonSelect(v-model="eventoId")

      └─ IonList(v-if="notasFiltradas.length")

        └─ NotaCard * v-for="nota in notasFiltradas"

        └─ NotaEmptyState(v-else) @crear

      └─ NotaDetalle(visible=mostrarModalVer) @cerrar

      └─ IonAlert(is-open=mostrarAlertaEliminar) @buttons
```

3. Datos y bindings

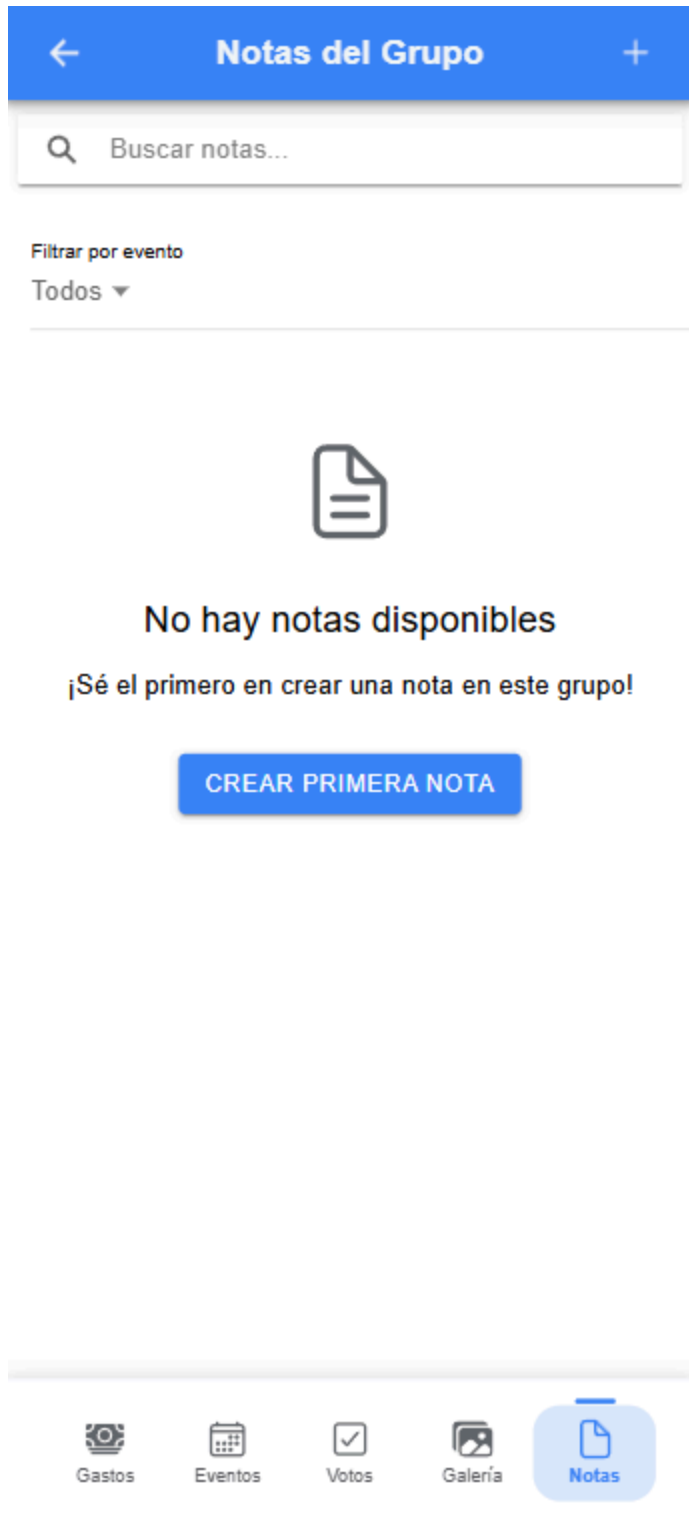
- **Composable:** useNotas(grupold, eventold) →
 - notasFiltradas, busqueda, cargando, cargarNotas(), eliminarNota(id, token)
- **Refs:** eventold, eventos (lista de eventos), notaSeleccionada, mostrarModalVer, mostrarAlertaEliminar, notaPendienteEliminar

4. Estados y validaciones

- Spinner al cargar
- Watch en eventold para recargar notas
- Searchbar filtra localmente
- Confirmar eliminación con IonAlert
- Empty state con acción “Crear nota”

5. Flujo de navegación

- “+” → /dashboard/\${grupold}/notas/crear
- Editar → /dashboard/\${grupold}/notas/editar/:notald
- Ver detalle → abre modal



Pantalla “Votaciones del Grupo”

Ruta: /dashboard/:grupold/votaciones

Acceso: Autenticado (miembro)

1. Propósito

Listar las votaciones del grupo, buscar, crear/editar/cerrar/eliminar (si es creador), votar, ver resultados.

2. Maquetación y componentes

```
<IonPage>

  └─ IonHeader

    └─ IonToolbar

      └─ IonBackButton(default-href=`/dashboard/${grupoId}`)

      └─ IonTitle "Votaciones del Grupo"

      └─ IonButton(add) @click="mostrarModalCrear = true"

  └─ IonContent

    └─ div.loading-container(v-if="cargando")

    └─ div(v-else)

      └─ IonSearchbar(v-model="busqueda") @ionInput="filtrarVotaciones"

      └─ IonList(v-if="votacionesFiltradas.length")

        └─ IonCard * v-for="votacion in votacionesFiltradas"

          └─ CardHeader: título, badge estado, acciones editar/cerrar/eliminar

          └─ CardContent: descripción, opciones (chips), votar/ver resultados

        └─ div.empty-state(v-else) con "Crear primera votación"

      └─ IonModal(is-open=mostrarModalCrear||mostrarModalEditar) → formulario crear/editar

      └─ IonModal(is-open=mostrarModalVotar) → formulario voto

      └─ IonModal(is-open=mostrarModalResultados) → resultados con progress bars
```

3. Datos y bindings

- **refs/reactive:**
 - votaciones, votacionesFiltradas, busqueda, cargando
 - mostrarModalCrear, mostrarModalEditar, mostrarModalVotar, mostrarModalResultados
 - formularioVotacion, votacionEditando, votacionParaVotar, opcionSeleccionada, resultadosVotacion
 - usuarioActual
- **Servicios:**
 - votacionService.listarPorGrupo(grupold, token)
 - votacionService.obtenerMiVoto(votacionId, token)
 - votacionService.guardar/actualizar/cerrar/eliminar/votar/resultados

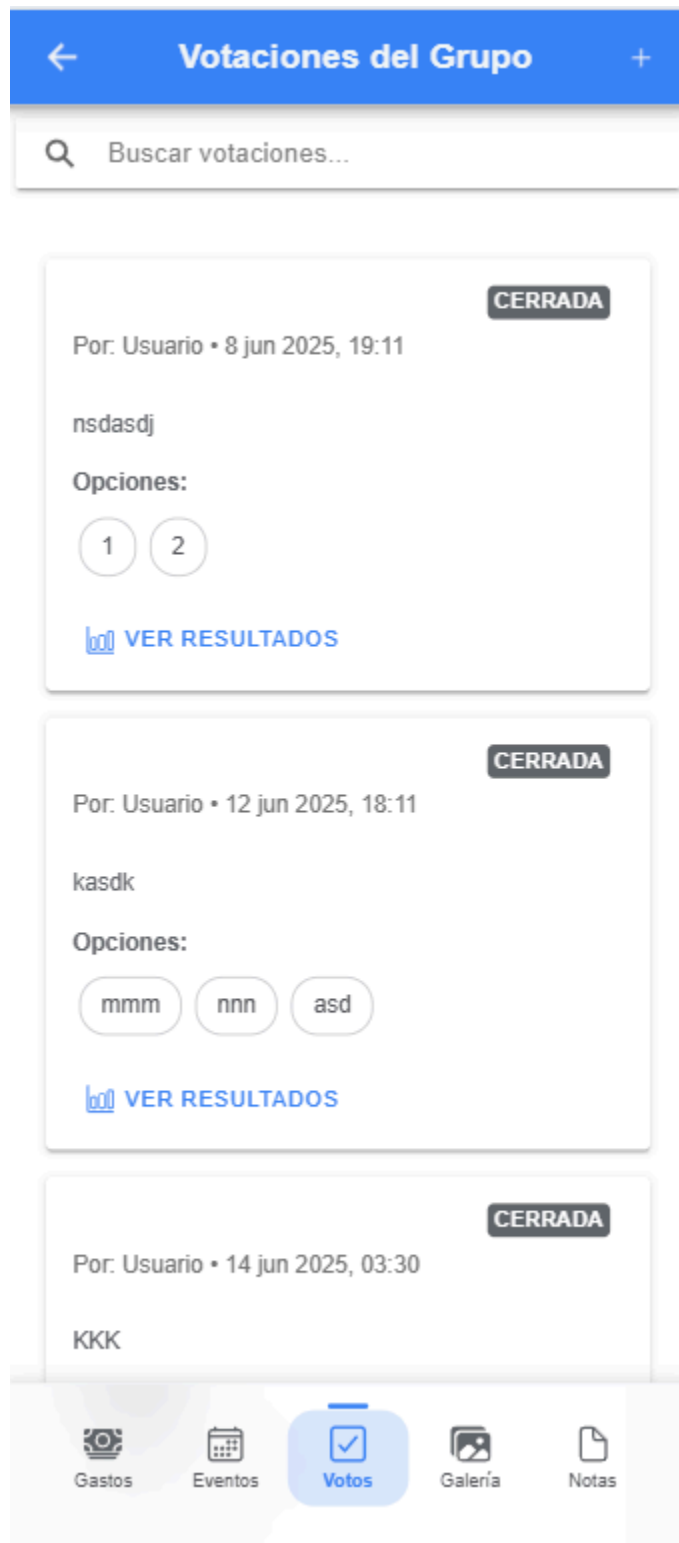
4. Estados y validaciones

- Spinner en carga y guardado/voto
- Sólo creador ve botones editar/cerrar/eliminar
- Validación mínimo 2 opciones y texto de pregunta no vacío
- Prevent multiple votes (yaVote)
- Toasts y alerts para confirmaciones y errores

5. Flujo de navegación

- “+” abre modal nuevo
- Editar/Eliminar/Cerrar en el listado abre alert/modal
- Votar abre modal voto; al confirmar refresca lista

- Ver resultados abre modal resultados



Pantalla “Asistencia a Evento”

Ruta: /dashboard/:grupold/eventos/:eventold/asistencia

Acceso: Autenticado (miembro)

1. Propósito

Mostrar detalle de un evento (título, fecha, descripción), confirmar o cambiar tu asistencia, y ver listados de asistentes y no-asistentes.

2. Maquetación y componentes

```
<IonPage>

└─ IonHeader

  └─ IonToolbar(color="primary")

    └─ IonBackButton(default-href=`/dashboard/${grupoId}/eventos`)

    └─ IonTitle "Asistencia"

└─ IonContent.ion-padding

  └─ Spinner(v-if="!evento")

  └─ .content(v-else)

    └─ h2 {{ evento.titulo }}

    └─ p {{ formatFecha(evento.fecha) }}

    └─ p(descripción opcional)

    └─ Botón 🧑🏻‍🤝‍🧑🏻 asistentesConfirmados.length

    └─ Botón "Confirmar asistencia" o "Cambiar asistencia"

    └─ IonSegment(v-model="segment") → "Asistentes" / "No asistentes"

    └─ IonList *v-if="segment==='asistentes' "

      └─ IonList *v-else

└─ IonToast para feedback
```

3. Datos / bindings


- **refs/reactive:** evento, asistentes, enviando, segment

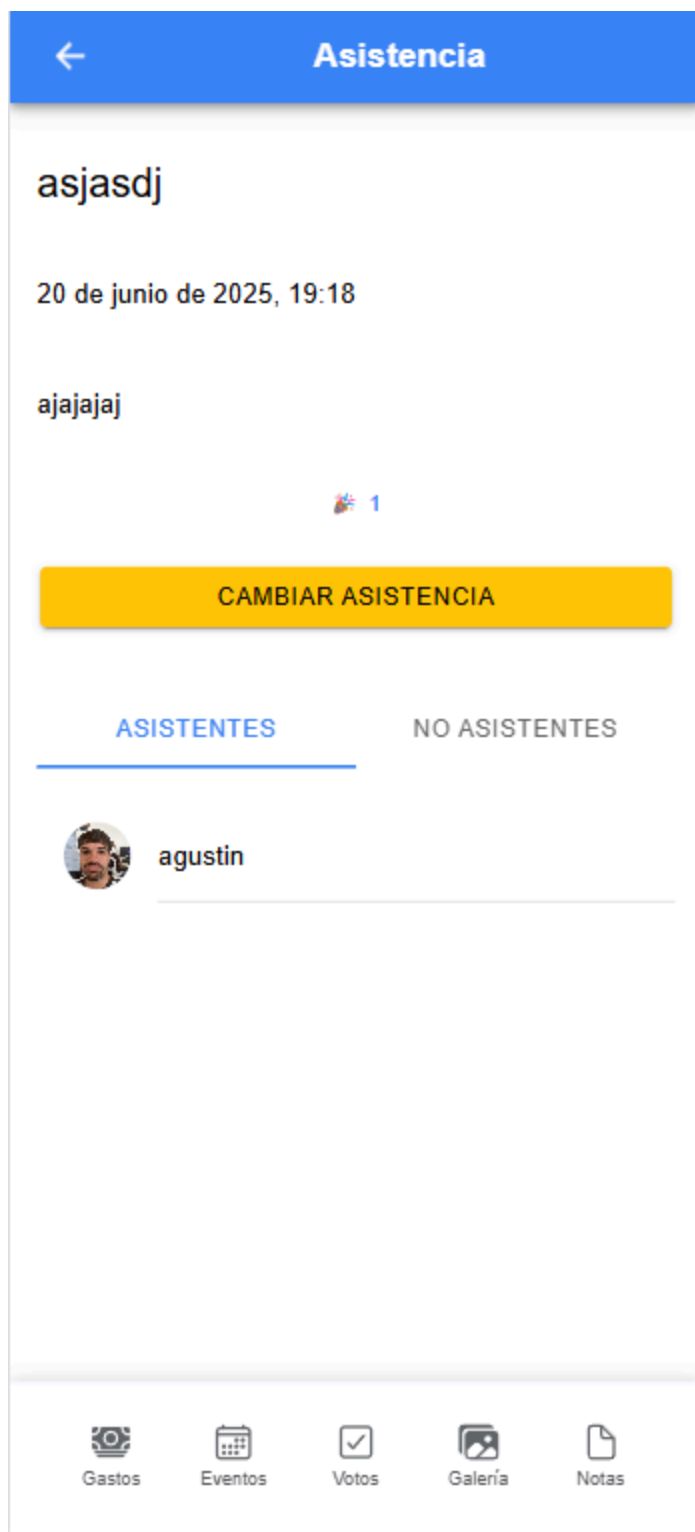
- **computed:** asistentesConfirmados, asistentesRechazados, asistenciaUsuario
- **servicio:**
 - EventosService.obtenerEventoPorId(eventId)
 - EventosService.obtenerAsistentes(eventId)
 - EventosService.marcarAsistencia(eventId, asistio, token)

4. Estados y validaciones

- Spinner hasta cargar
- Botón deshabilitado mientras enviando
- Segment controla qué lista mostrar
- Toast con mensaje de éxito o error

5. Flujo de navegación

- Back → listado de eventos
- Click en “” no navega, solo muestra cuenta
- Confirmar/asistir recarga datos de asistentes



Pantalla “Detalle de Deudas de Usuario”

Ruta: /dashboard/:grupold/resumen/:usuariold

Acceso: Autenticado (miembro)

1. Propósito

Mostrar el resumen de deudas de un usuario concreto: a quién debe y quién le debe.

2. Maquetación

```
<IonPage>

  └─ IonHeader

    |   └─ IonToolbar(color="primary")

    |   |   └─ IonBackButton @click="volver()"

    |   |   └─ IonTitle "Detalle de Deudas"

  └─ IonContent.ion-padding

    |   └─ h2 "Deudas de {{ usuarioNombre }}"

    |   └─ IonList *v-if="resumenDeudas.length"

    |       └─ IonItem *v-for="item in resumenDeudas"

    |           └─ <p>Le debe a {{ item.acreedorNombre }}</p>

    |           └─ <p :class="item.saldado ? 'text-success' : 'text-danger'">

    |               Monto {{ formatMonto(item.monto) }}

    └─ IonText(v-else) "No tiene deudas..."
```

3. Datos / bindings

- **refs:** deudas, luego compute resumenDeudas agrupando por acreedor
- **servicio:** varias llamadas a GET /gasto/\${gastold}/deudas para todos los gastos del grupo; filtro deudorId===usuariold

4. Estados

- Empty state si no hay deudas
- Formateo de montos con Intl.NumberFormat

5. Navegación

- Back → Resumen general de deudas



Pantalla “Editar Gasto”

Ruta: /dashboard/:grupold/gastos/:gastold/editar

Acceso: Autenticado (miembro)

1. Propósito

Modificar un gasto existente: título, monto, pagador, participantes, reparto (iguales o personalizado), evento opcional.

2. Maquetación

- Header con **Back** y **Guardar**
- IonContent:
 - **Información básica** (título, monto, pagadoPor)
 - **Participantes** (grid con checkbox y avatar)
 - **División** (radio “iguales”/“personalizadas” + inputs personalizados)
 - **Evento** (select opcional)
 - **Resumen** (totales calculados)

3. Datos / bindings

- **refs/reactive:** formulario, errores, miembrosGrupo, eventos, guardando, cargandoInicial
- **computed:** participantesSeleccionados, montoPorPersona, totalPersonalizado, formularioValido
- **servicio:**
 - GET /gasto/\${gastold}
 - PUT /gasto/\${gastold}
 - GET /grupos/\${grupold}/usuarios
 - GET /eventos/\${grupold}/eventos

4. Validaciones

- Campos obligatorios, total personalizado coincide con monto
- Mensajes de error bajo cada input

5. Flujo

- Back → lista de gastos
- Guardar deshabilitado si no es válido
- Toasts para éxito/fracaso

←

Editar Gasto

✓

Información del Gasto

Título del gasto *

jsajasdj

Monto *

22

€

Pagado por *

agustin

▼

Participantes

Selecciona quién participa en este gasto

A

agustin

✓

J

jose

✓

División del Gasto

Partes iguales

☒ Dividir el monto equitativamente entre todos

Cantidades personalizadas

Gastos

Eventos

Votos

Galería

Notas

Pantalla “Resumen de Deudas del Grupo”

Ruta: /dashboard/:grupold/resumen

Acceso: Autenticado (miembro)

1. Propósito

Listar todos los miembros con su balance neto: cuánto debe o le deben.

2. Maquetación

```
<IonPage>

  └─ IonHeader

    |   └─ IonBackButton

    |   └─ IonTitle "Resumen de Deudas"

  └─ IonContent.ion-padding

    └─ IonList *v-if="usuarios.length"

      └─ IonItem *v-for="u in usuarios" @click="verDetalle(u)"

        └─ <p v-if="debe">Debe {{ formatMonto }}</p>

        └─ <p v-if="leDeben">Le deben {{ formatMonto }}</p>

        └─ <p v-else>✅ Sin deudas
```

3. Datos / bindings

- **refs:** usuarios, resmenes
- **servicio:**
 - GET /grupos/\${grupold}/usuarios
 - GET /gasto/grupos/\${grupold}/resumen

4. Estados

- Empty state con “Cargando participantes...” hasta cargar

- Click en usuario navega a detalle

5. Flujo

- Tap en usuario → /resumen/:usuariold



Pantalla “¿Qué Crear?” (Selector)

Ruta: /dashboard/:grupold/crear

Acceso: Autenticado (miembro)

1. Propósito

Permitir al usuario elegir rápidamente qué tipo de elemento quiere crear: evento, gasto, votación, nota o imagen.

2. Maquetación

- Header con Back
- Content:
 - Título y subtítulo
 - Grid 2×3 de **IonCard** → “Evento”, “Gasto”, “Votación”, “Nota”, “Imagen”

3. Datos / bindings

- **refs:** grupold
- Navegación: router.push(/dashboard/\${grupold}/crear/\${tipo})

4. Estados

- Spinner solo si grupold no está resuelto (fallback de localStorage)

5. Flujo

- Tap en tarjeta → ruta específica de creación

Crear nuevo elemento

¿Qué deseas crear?

Selecciona el tipo de elemento que quieres añadir al grupo



Evento

Organiza actividades y reuniones



Gasto

Registra gastos compartidos



Votación

Toma decisiones grupales



Gastos



Eventos



Votos



Galería



Notas

Pantalla “Crear/Editar Nota”

Ruta: /dashboard/:grupold/notas/(crear|editar/:notald)

Acceso: Autenticado (miembro)

1. Propósito

Formulario para introducir título, contenido y evento opcional; soporta crear o editar.

2. Maquetación

- Header con Back y Toast+Loading
- <NotaForm> reutilizable con props titulo, contenido, eventold, eventos
- Botón “Guardar” desde el mismo form component

3. Datos / bindings

- **refs/reactive:** titulo, contenido, eventold, eventos, isLoading, showToast, toastMessage
- **computed:** esEdicion
- **servicio:**
 - GET NotasService.obtenerNotaPorId (solo en edición)
 - POST/PUT NotasService.crearNota o actualizarNota

4. Validaciones

- Título y contenido no vacíos
- Toasts para feedback

5. Flujo

- En edición: carga datos en onMounted
- Tras guardar: router.back() tras mostrar toast



Nueva Nota

Título de la nota

Ej. Reunión semanal

Contenido

0 / 100

Escribe el contenido...

0 / 2000

Evento (opcional)

Sin evento ▼

CREAR NOTA

Gastos

Eventos

Votos

Galería

Notas

Pantalla “Crear Evento”

Ruta: /dashboard/:grupold/crear/evento

Acceso: Autenticado (miembro)

1. Propósito

Formulario estructurado en tres secciones: información básica (título/desc), fecha/hora, ubicación.

2. Maquetación

- Header con Back y botón “check”
- Content:
 - <BasicInfoSection> (título, descripción)
 - <DateTimeSection> (fecha)
 - <LocationSection> (ubicación)
 - Toast + Loading

3. Datos / bindings

- **refs:** titulo, descripcion, fecha, ubicacion, coordenadas, isLoading, showToast
- **computed:** puedeGuardar (todos los campos no vacíos)
- **servicio:** EventosService.crearEvento(grupold, eventData, token)

4. Validaciones

- Campos obligatorios no vacíos
- Fecha futura por defecto (puede venir en query)

5. Flujo

- Guardar → POST → toast → router.back()

Nuevo evento ✓

Información del evento

Título del evento


Ej: Cena de cumpleaños

0 / 100

Descripción

Describe los detalles del evento...


0 / 500

 **Fecha y hora**

Fecha del evento


sábado, 14 de junio de 2025, 12:09

>


 **Ubicación**


Dirección


Ej: Calle Principal 123, Madrid


 **CREAR EVENTO**


CANCELAR

 Gastos

 Eventos

 Votos

 Galería

 Notas

Pantalla “Crear Gasto”

Ruta: /dashboard/:grupold/crear/gasto

Acceso: Autenticado (miembro)

1. Propósito

Formulario dividido en componentes: básico (título, monto, pagador), selección de participantes, división, evento opcional, resumen y acciones.

2. Maquetación

- Header con Back y spinner durante carga
- <FormularioBasico>, <SeleccionParticipantes>, <DivisionGasto>, <SelectorEvento>, <ResumenGasto>, <BotonesAccion>
- Toast

3. Datos / bindings


- **reactive:** formulario, errores, usuarios, eventos, cargando, enviando, showToast
- **computed:** participantesSeleccionados, pagadorSeleccionado, formularioValido
- **servicio:** GastoService.crearGasto(...)


4. Validaciones

- Igual que en editar: monto>0, al menos un participante, total personalizado coincide

5. Flujo

- Tras crear, toast y redirección a /dashboard/\${grupold}

 **Nuevo Gasto**
Esto e una pasada

 **Información del Gasto**


Título del gasto

Título del gasto

MONTO

0.00

€

 **Participantes**

agustin ▼


☒ Participantes (1)

A

agustin☒


J


jose☐


 **División del Gasto**


☒ Partes iguales


☐ Cantidades personalizadas

 Gastos

 Eventos

 Votos

 Galería

 Notas

Pantalla “Crear Imagen”

Ruta: /dashboard/:grupold/crear/imagen

Acceso: Autenticado (miembro)

1. Propósito

Subir una imagen al grupo/evento: selección de archivo, preview, asociación a evento opcional.

2. Maquetación

- Header con Back, título, botón de abrir modal de subida
- Content:
 - Filtros (tipo “grupo”/“usuario”/“evento”)
 - Grid de imágenes con selección múltiple y me-long-press
 - Modal de Subida (<input type="file">, preview, evento select)
 - Modal de Vista Previa completa (imagen + metadata + eliminar)

3. Datos / bindings

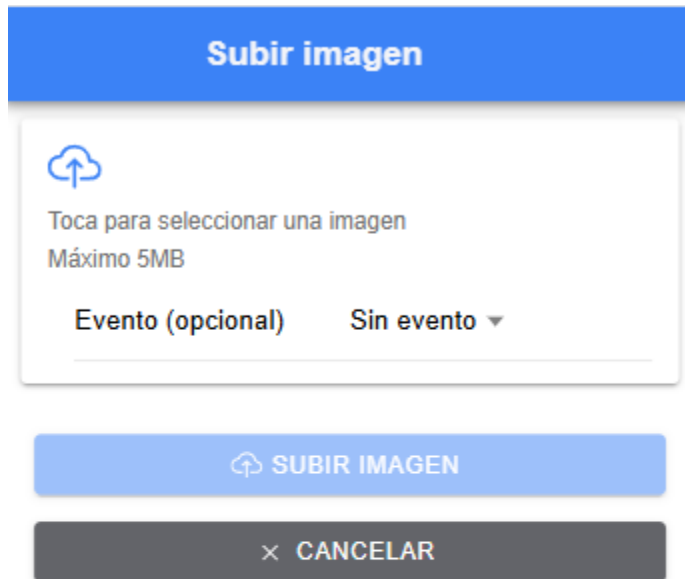
- **reactive:** imagenes, eventos, filtros, estado, modalSubida, modalVistaPrevia, modoSeleccion, imagenesSeleccionadas
- **servicio:**
 - `imageService.getByGrupo/Usuario/Evento`
 - `imageService.uploadImage(FormData)`
 - `imageService.getFullImage(id)`
 - `imageService.deleteImage(id)`

4. Estados

- Carga inicial, subiendo, selección múltiple, toast/alert

5. Flujo

- Filtrar recarga Gallery
- Long-press → modo selección → acciones múltiples
- Upload → recarga
- Delete → recarga y cerrar modales



Pantalla “Crear Votación”

Ruta: /dashboard/:grupold/crear/votacion

Acceso: Autenticado (miembro)

1. Propósito

Configurar nueva votación: título, descripción, al menos 2 opciones, fecha límite opcional.

2. Maquetación

- Header con Back y botón “check”
- Content:
 - Sección “Información básica” (título, descripción)
 - Sección “Opciones” (lista dinámica de inputs + botones añadir/eliminar)
 - Sección “Configuración” (fecha límite con <ion-datetime>)
 - Toast + Loading

3. Datos / bindings

- **refs:** titulo, descripcion, opciones, fechaLimite, mostrarFechaLimite, isLoading, showToast
- **computed:** puedeGuardar (título no vacío + ≥2 opciones no vacías)
- **servicio:** votacionService.guardar(grupold, payload, token)

4. Validaciones

- ≥2 opciones no vacías, fecha límite futura si existe

5. Flujo

- Guardar → POST → toast → router.back()

Nueva Votación

Pregunta/Título *

¿Cuál es tu pregunta?

Descripción

Descripción opcional de la votación...

Opciones de votación *

Opción 1

Opción 2

⊕ AGREGAR OPCIÓN

Fecha límite (opcional)

junio de 2025 ▾

<

>

D	L	M	X	J	V	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Time

12:18

Especificación

1. Propósito

Ofrecer navegación global hacia las secciones principales de la app: gestión de grupos, dashboard, eventos, gastos, votaciones, notas, galería, resumen, configuración, perfil e invitaciones.

2. Estructura de la UI

```
<IonMenu content-id="main-content">
  └─ IonHeader
    └─ IonToolbar(color="primary")
      └─ IonTitle "Menú"
    └─ IonContent
      └─ IonList
        └─ IonItem router-link="/grupo" → "Grupos"
        └─ IonItem *v-if="grupoId" → Dashboard
        └─ IonItem *v-if="grupoId" → Eventos
        └─ IonItem *v-if="grupoId" → Gastos
        └─ IonItem *v-if="grupoId" → Votaciones
        └─ IonItem *v-if="grupoId" → Notas
        └─ IonItem *v-if="grupoId" → Galería
        └─ IonItem *v-if="grupoId" → Resumen
        └─ IonItem *v-if="grupoId" → Configuración
        └─ IonItem router-link="/perfil" → "Perfil"
        └─ IonItem router-link="/invitaciones" → "Invitaciones"
```

3. Datos / bindings

- `const grupoid = localStorage.getItem('grupoActivoid') || ''`
 - Controla la visibilidad de rutas relacionadas con un grupo activo.

- Cada <ion-item> usa router-link (o :router-link) para navegación interna.

4. Estados y condiciones

- **Siempre visible:** “Grupos”, “Perfil”, “Invitaciones”.
- **Sólo si grupold existe:** enlaces a dashboard y sub-secciones del grupo activo.

5. Flujo de navegación

- Al abrir el menú, el usuario puede saltar a cualquier sección listada.
- El contenido principal (con id="main-content") se desliza cuando el menú está abierto, garantizando contexto visual.

6. Reutilización

- Incluido habitualmente en el layout principal (App.vue o plantilla base) para que aparezca en todas las vistas que dispongan del botón de menú.

3.7.3 Matriz de acceso por pantalla

Pantalla / Vista	Anónimo	Sin grupo	Con grupo	Admin
Login / Registro	✓	✓	✓	✓
Crear / Unirse a grupo	✗	✓	✓	✓
Dashboard (Home)	✗	✗	✓	✓
Eventos	✗	✗	✓	✓
Gastos	✗	✗	✓	✓
Notas	✗	✗	✓	✓
Galería	✗	✗	✓	✓
Perfil	✗	✓	✓	✓
Configuración de grupo	✗	✗	✓ ¹	✓

¹ El menú muestra la opción de configuración también para usuarios “con grupo”, pero los botones de acción (generar código, expulsar, etc.) solo aparecen si `isAdmin === true`.

3.7.4 Especificación de formatos de impresión

En esta primera versión de Planora **no** se ha previsto generación ni exportación de documentos imprimibles.

Toda la información se presenta en pantalla (texto, listas, imágenes, gráficos básicos) y el usuario podrá utilizar las funciones nativas del dispositivo (captura de pantalla, compartir) si desea conservar o imprimir los datos.

3.7.5 Especificación de la navegabilidad entre pantallas

1. Menú lateral (Drawer / Side-Menu)

- Presente en la mayoría de pantallas (a través de AppMenu.vue).
- Acceso directo a: Grupos, Dashboard, Eventos, Gastos, Notas, Galería, Resumen, Configuración, Perfil e Invitaciones.

2. Barra de navegación superior (Header)

- Botón de **volver** (<ion-back-button>) en subpantallas para retornar a la anterior.
- Título dinámico y botones de acción contextuales (por ejemplo: “+” para crear un nuevo evento, editar perfil, cerrar sesión).

3. Rutas principales

- Definidas en router/index.ts mediante guardias (beforeEach) que redirigen según estado de autenticación y pertenencia a grupo.
- Al cambiar de pestaña (Dashboard ↔ Eventos ↔ Gastos ↔ Notas ↔ Galería) se reemplaza la vista activa, manteniendo el historial de navegación.

4. Flujo “Crear” un elemento

- Desde un FAB o icono “+” en el header se abre Creacion/Crear.vue, que redirige a la pantalla específica (e.g. CrearEvento.vue, CrearGasto.vue, etc.).
- Al guardar o cancelar, el usuario vuelve automáticamente al listado o dashboard correspondiente.

4. Construcción del sistema

4.1 Arquitectura técnica

Cliente (Mobile App)

- **Framework:** Ionic Vue (Vue 3 + Composition API) sobre Capacitor (Android / iOS).
- **Patrón MVVM:**
 - **View:** .vue (pantallas en src/views y componentes en src/components).
 - **ViewModel / Composables:** lógica UI y estado local en src/Composable.
 - **Model / Servicios:** llamadas HTTP con Axios en src/service.
- **Estado global:** Pinia (src/store) con módulos por dominio (usuarios, grupos, eventos...).
- **Navegación:** Vue Router (src/router/index.ts), menú lateral (AppMenu.vue) y navegación inferior en PageHeader.vue.

Servidor (API REST)

- **Framework:** Spring Boot
- **Seguridad:** JWT (filtros y Authorization: Bearer <token>).
- **Persistencia:** MySQL con JPA/Hibernate.
- **Almacenamiento de archivos:** AWS S3 (o Firebase Storage).
- **Comunicación:** cliente ↔ servidor por HTTPS/JSON y multipart-form para imágenes.

4.2 Tecnologías utilizadas

Capa	Herramienta / Librería
Frontend	Ionic Vue, Vue 3, TypeScript, Composition API
Estado	Pinia

HTTP	Axios
Navegación	Vue Router
Backend	Spring Boot, Java
Persistencia	MySQL, JPA/Hibernate
Archivos	AWS S3 (SDK Java)
Autenticación	JSON Web Tokens (JWT)
Control de versiones	Git, GitHub
Testing	Jest / Vitest (frontend), JUnit (backend)
API client	Postman / Insomnia

4.3 Estructura del proyecto

src/

├─ assets/ # Imágenes, fonts...

```

└─ components/           # Comunes (AppMenu, PageHeader...)
└─ Composable/          # Lógica reusable (useCalendar, useGastos...)
└─ router/              # Definición de rutas (router/index.ts)
└─ service/             # Clientes API (AuthService, GrupoService...)
└─ store/               # Pinia stores (dashboardStore.ts)
└─ styles/              # SCSS global y variables
└─ theme/               # Configuración de tema Ionic
└─ types/               # Interfaces y tipos TS
└─ utils/               # Helpers (date.ts, string.ts...)
└─ views/
    └─ Components/       # UI específicas por dominio
        └─ Dashboard/    # StatsGrid, CompactCalendar...
        └─ Grupo/        # GroupList, ActiveGroupBanner...
        └─ CreacionGasto/
            └─ ...        # Notas, Votaciones, Galería...
    └─ Configuracion/    # Grupo.vue, Perfil.vue
    └─ Creacion/         # Crear.vue + pantallas crear X
    └─ Detalles/        # DetalleEvento, DetalleGasto...
    └─ Dashboard.vue
    └─ Grupo.vue
    └─ Invitaciones.vue
    └─ LoginPage.vue
    └─ RegisterPage.vue
    └─ RestPasswordPage.vue

```


Conclusión

La combinación Ionic Vue + MVVM + Pinia produce un código limpio y modular en el cliente. Un backend ligero en Node.js/Express con JWT y MongoDB/Firestore aporta seguridad y escalabilidad. El resultado es una plataforma colaborativa ágil, preparada para añadir notificaciones en tiempo real o microservicios en el futuro.

Bibliografía

- Ionic Framework Docs: <https://ionicframework.com/docs>
- Vue 3 Composition API: <https://v3.vuejs.org/guide/composition-api-introduction.html>
- Pinia State Management: <https://pinia.vuejs.org/>
- Axios HTTP Client: <https://axios-http.com/>
- Node.js + Express: <https://expressjs.com/>
- Mongoose ODM: <https://mongoosejs.com/>
- JSON Web Tokens: <https://jwt.io/>