PROYECTOS TRANSVERSALES

Módulos implicados en los proyectos

- **Programación de Servicios y Procesos**
- **Acceso a Datos**
- **Programación Multimedia y Dispositivos Móviles**

Requisitos transversales (para todos)

- **Seguridad:** hash de contraseñas (BCrypt/Argon2), JWT con expiración, roles/permisos.
- **CORS:** configuración segura para la app Android.
- **Paginación y filtros** en listados.
- **Logs y auditoría** de acciones clave.
- **Tests:** unitarios, integración y de contrato (OpenAPI).
- **CI/CD:** build, tests y despliegue con contenedores.
- **Observabilidad:** métricas y trazas básicas.
- **Documentación:** endpoints con ejemplos y estados de error.

Stack sugerido

Opción Backend Java

- **Framework:** Spring Boot 3
- **Auth:** Spring Security + JWT
- **ORM/Migraciones:** JPA/Hibernate + Flyway (o Liquibase)
- **DB:** PostgreSQL (prod), H2 (tests)
- **Cache/Colas:** Redis + Spring Data Redis
- **Tiempo real:** Spring WebSocket / STOMP
- **Tareas en background:** Spring Scheduler + (opcional) Spring Batch
- **Tests:** JUnit 5 + Mockito + Testcontainers
- **Build/CI:** Maven o Gradle + GitHub Actions
- **Documentación:** OpenAPI/Swagger (springdoc-openapi)

Opción Backend Python

- **Framework:** FastAPI (ASGI)
- **Auth:** OAuth2 + JWT con `fastapi.security`
- **ORM/Migraciones:** SQLAlchemy 2.0 + Alembic
- **DB:** PostgreSQL (prod), SQLite (dev/tests)
- **Cache/Colas:** Redis
- **Tiempo real:** WebSockets integrados en FastAPI
- **Tareas en background:** Celery (o RQ) + Celery Beat
- **Tests:** pytest + httpx (cliente async)
- **Contenedores:** Docker + docker-compose
- **Documentación:** OpenAPI automático + Redoc

Android (Android Studio)

- **Lenguaje:** Kotlin (recomendado) o Java
- **Networking:** Retrofit + OkHttp + Moshi/Gson
- **Local DB:** Room
- **Arquitectura:** MVVM + Jetpack (ViewModel, LiveData/Flow, Navigation)
- **Notificaciones:** Firebase Cloud Messaging (FCM)
- **Multimedia:** ML Kit (códigos de barras/QR), CameraX, ExoPlayer
- **Mapas:** Google Maps SDK
- **Background:** WorkManager

1. Gestor de Citas Médicas Multiplataforma

Descripción: Sistema para que pacientes y médicos gestionen citas, historiales y notificaciones.

Objetivo: Agendar, modificar y validar citas médicas con roles (paciente, médico, admin), QR y notificaciones.

Roles

- **Admin**
- **Médico**
- **Paciente**

Requisitos funcionales básicos

1. Registro e inicio de sesión con JWT.

2. Gestión de perfil (foto, datos básicos).
3. Búsqueda de médicos por especialidad.
4. Creación, modificación y cancelación de citas.
5. Prevención de solapamientos por médico/horario.
6. Generación y descarga de **QR** por cita.
7. Validación de cita mediante escaneo de **QR** (Android).
8. Notificaciones (email/push) al crear/modificar/cancelar.
9. Historial de citas por usuario.
10. Adjuntar documentos (informes, recetas) a la cita.
11. Panel de médico: su agenda y disponibilidad.
12. Panel de admin: gestión de usuarios y estadísticas.
13. Logs de actividad (quién hizo qué y cuándo).
14. Búsqueda y filtros por fecha/estado/doctor.
15. Exportación simple (CSV) de agenda.

Módulos / Endpoints (ejemplos)

- `/auth`: registro/login/refresh
- `/users`: perfiles y roles
- `/doctors`: especialidades, disponibilidad
- `/patients`: datos del paciente
- `/appointments`: CRUD, reprogramar, cancelar
- `/appointments/{id}/qr`: generar/descargar
- `/appointments/{id}/verify`: validar QR
- `/files`: subir/descargar adjuntos
- `/notifications`: registro de token FCM, pruebas

Entidades principales

- `User(id, email, password_hash, role, created_at)`
- `Doctor(id, user_id, specialty, availability[])`
- `Patient(id, user_id, insurance_number)`
- `Appointment(id, doctor_id, patient_id, datetime, status, qr_code_url)`
- `File(id, owner_id, url, type, created_at)`

Servicios y procesos

- **Scheduler**: recordatorios de cita (24h/2h antes).
- **Job**: limpieza de QR expirados.
- **Cola**: envío de notificaciones (email/push).

Android – Pantallas

Login, registro, búsqueda de médicos, agenda, detalle de cita, escaneo de **QR**, perfil, adjuntos.

Multimedia

- **QR**: generación y validación.
- **Adjuntos**: PDFs/imágenes (con visor).
- **Notificaciones push** (FCM).

Backend Java (Spring Boot)

- Spring Security (JWT), JPA/Hibernate, Flyway, WebSocket para avisos en tiempo real.

Backend Python (FastAPI)

- FastAPI + SQLAlchemy + Alembic, Celery para notificaciones, WebSockets para avisos.

Entregables por sprints

- **Sprint 1:** auth, CRUD de usuarios, CRUD de citas, validación de solapamientos.
- **Sprint 2:** QR, adjuntos, notificaciones, tokens FCM, perfiles y disponibilidad.
- **Sprint 3:** estadísticas, exportación, logs, hardening de seguridad y pruebas.

---

2. Red Social Académica para Estudiantes

Descripción: Plataforma donde los estudiantes pueden compartir apuntes, eventos, y comunicarse.

Objetivo: Feed de publicaciones, comentarios, multimedia y chat en tiempo real.

Roles

- **User**, **Moderator**, **Admin**

Requisitos funcionales básicos

1. Registro/login JWT.
2. Creación de publicaciones (texto/imagen/video).
3. Comentarios y “me gusta”.

4. Perfil de usuario (bio, avatar, intereses).
5. **Chat** uno-a-uno y por grupos (**WebSocket**).
6. Moderación: marcar contenido y resolver reportes.
7. Búsqueda por usuarios/etiquetas.
8. Paginación del feed (infinite scroll).
9. Subida de imágenes con thumbnails.
10. Reproductor de vídeo (en móvil).
11. Notificaciones push (nuevos mensajes/likes).
12. Configuración de privacidad (cuenta pública/privada).
13. Estadísticas básicas (posts/día, engagement).
14. Lista de seguidores/seguidos.
15. Gestión de grupos/temas (opcional).

Módulos / Endpoints

- `/auth`, `/profiles`, `/posts`, `/comments`, `/likes`, `/media`
- `/chat` (REST para historial) y `/ws/chat/{roomId}` (WebSocket)
- `/moderation`: reports, acciones

Entidades

- `User`, `Profile`, `Post`, `Comment`, `Like`, `Message(room_id, sender_id, content, created_at)`, `Report`

Servicios y procesos

- **Cola**: generación de thumbnails, transcodificación ligera (opcional).
- **WebSocket**: chat y notificaciones.
- **Job**: purga de media no referenciada.

Android – Pantallas

Login/registro, feed, perfil, creación de post, chat, media viewer, moderación.

Multimedia

Imágenes/vídeos; chat en tiempo real; notificaciones.

Backend Java

- Spring Boot, WebSocket/STOMP, almacenamiento S3 (MinIO), moderación con colas.

Backend Python

- FastAPI + WebSockets, MinIO/S3, Celery para media.

Sprints

- **S1:** auth, perfiles, posts, comentarios, paginación.
- **S2:** likes, media upload/thumbnail, chat WebSocket, notificaciones.
- **S3:** moderación, estadísticas, endurecimiento y testing.

---

3. App de Turismo Interactivo

Descripción: Guía turística con rutas, puntos de interés, y recomendaciones personalizadas.

Objetivo: POIs, rutas, reseñas, recomendaciones y modo offline.

Roles

- **User**, **Editor**, **Admin**

Requisitos funcionales básicos

1. Registro/login.
2. Listado y detalle de POIs (mapa).
3. Rutas (secuencias de POIs).
4. Reseñas y puntuaciones.
5. Descarga de contenido para uso **offline**.
6. Recomendaciones básicas (popularidad y afinidad por tags).
7. Favoritos e historial de visitas.
8. Filtros por distancia, tipo, puntuación.
9. Multimedia: fotos, audios, vídeos de los POIs.
10. Modo oscuro, UI accesible.
11. Sincronización de cambios (delta por `updated_at`).
12. Notificaciones (nuevos POIs en tu zona).
13. Importación/exportación de rutas (JSON).
14. Reporte de POIs incorrectos.
15. Panel editor: alta/edición de POIs.

Módulos / Endpoints

- `/auth`, `/pois`, `/routes`, `/reviews`, `/media`, `/recommendations`, `/sync/delta`

Entidades

- `POI(id, name, lat, lng, tags, media_url)`
- `Route(id, name, poi_ids[])`
- `Review(user_id, poi_id, rating, comment)`
- `UserPreference(tags[])`

Servicios y procesos

- **Job**: recomputación de recomendaciones (batch).
- **Scheduler**: notificaciones geolocalizadas (opt-in).
- **Cola**: generación de audio-guias (opcional).

Android – Pantallas

Mapa, listado, detalle POI, rutas, reseñas, favoritos, descargas offline.

Multimedia

Galerías, audio guías, vídeo; mapas interactivos.

Backend Java

- Spring Boot + JPA, servicios REST, recomendaciones por batch (Spring Batch).

Backend Python

- FastAPI + Celery Beat para recomendaciones; endpoints delta.

Sprints

- **S1:** POIs, rutas, reseñas, autenticación.
- **S2:** recomendaciones, offline sync, favoritos.
- **S3:** notificaciones geolocalizadas, exportaciones, pruebas.

---

4. Sistema de Gestión de Inventario con Sensores Simulados

Descripción: Control de stock con simulación de sensores IoT que actualizan el inventario.

Objetivo: Gestión de inventario con actualización por "eventos" de sensores simulados y escaneo móvil.

Roles

- **User**, **Manager**, **Admin**

Requisitos funcionales básicos

1. Auth y roles.
2. CRUD de productos y stocks.
3. Escaneo de **códigos de barras** (Android) para entradas/salidas.
4. Simulación de sensores: emisión de eventos (alta/baja stock).
5. Procesamiento de eventos y actualización de stock.
6. Alertas por stock bajo (email/push).
7. Historial de movimientos.
8. Auditoría de cambios (usuario/fecha).
9. Búsqueda por SKU/nombre/categoría.
10. Importación de catálogo (CSV).
11. Reportes: top consumidos, rotación.
12. Control de lotes o ubicaciones (opcional).
13. Interfaz para "recepción" y "despacho".
14. Cache para lecturas rápidas.
15. API para integraciones externas (opcional).

Módulos / Endpoints

- `/auth`, `/products`, `/stocks`, `/events`, `/scan`, `/reports`

Entidades

- `Product(id, sku, name, barcode, category)`
- `Stock(product_id, quantity, location)`
- `Event(id, type, product_id, delta, source, created_at)`
- `Movement(product_id, quantity, user_id, created_at)`

Servicios y procesos

- **Hilos/Jobs**: generador de eventos (sensores simulados).
- **Consumidor**: procesa eventos desde Redis y actualiza stock.
- **Job**: alertas y reportes programados.

Android – Pantallas

Login, listado y detalle de productos, escaneo, entrada/salida, reportes.

Multimedia

Escaneo de **códigos de barras** (ML Kit) y confirmaciones sonoras/visuales.

Backend Java

- Microservicios (opcional) con Spring Boot; Redis Streams; Scheduler para alertas.

Backend Python

- FastAPI, Celery para consumidor de eventos, Redis pub/sub.

Sprints

- **S1:** productos/stocks CRUD, escaneo móvil, eventos básicos.
- **S2:** consumidor de eventos, alertas, reportes.
- **S3:** importación CSV, auditoría, optimizaciones.

---

5. Plataforma de Gestión de Eventos y Entradas

Descripción: Organización de eventos con compra de entradas, control de acceso y estadísticas.

Objetivo: Publicación de eventos, venta de entradas, control de acceso con **QR**.

Roles

- **Organizer**, **Attendee**, **Admin**

Requisitos funcionales básicos

1. Auth y gestión de perfil.
2. CRUD de eventos (aforo, fecha, ubicación).
3. Carrito y "compra" de entradas (pago **simulado**).
4. Generación de **QR** por ticket.
5. Validación de entrada (check-in con **QR**).
6. Estadísticas por evento (vendidas, check-ins, ocupación).
7. Notificaciones (recordatorio de evento).
8. Políticas de reembolso (simuladas).
9. Exportación CSV de asistentes.
10. Roles y permisos (organizer ve sus eventos).
11. Búsqueda y filtros (fecha, ciudad, tipo).
12. Gestión de promociones/códigos de descuento.
13. Modo offline para validación (sync posterior).
14. Protección contra doble escaneo (idempotencia).
15. Confirmaciones vía email.

Módulos / Endpoints

- `/auth`, `/events`, `/tickets`, `/orders`, `/scan`, `/stats`, `/discounts`

Entidades

- `Event(id, name, venue, date, capacity)`
- `Order(id, user_id, event_id, total, paid, created_at)`
- `Ticket(id, event_id, user_id, qr_url, checked_in, checked_in_at)`
- `Discount(code, percentage, expires_at)`

Servicios y procesos

- **Job**: recordatorios antes del evento.
- **Cola**: generación de QR y envío de emails.
- **Validación**: endpoint idempotente para check-in.

Android – Pantallas

Explorar eventos, detalle, compra, wallet de entradas, escaneo/validación.

Multimedia

QR, imágenes promocionales, mapas de ubicación.

Backend Java

- Spring Boot, JPA, Spring Security; endpoint idempotente y control de concurrencia.

Backend Python

- FastAPI + SQLAlchemy; Celery para QR/emails; control transaccional en check-in.

Sprints

- **S1:** eventos, tickets, pedidos, compra simulada.
- **S2:** QR, validación, notificaciones, estadísticas.
- **S3:** descuentos, modo offline, exportaciones, pruebas.

---

6. Sistema de Gestión de Tareas y Productividad Personal

Descripción: App para gestionar tareas, recordatorios, calendario y seguimiento de hábitos.

Objetivo: Tareas, hábitos, calendario, métricas y recordatorios.

Roles

- **User**, **Admin**

Requisitos funcionales básicos

1. Auth y perfiles.
2. CRUD de tareas (título, fecha, prioridad, etiquetas).
3. Estados: pendiente, en curso, completada, vencida.
4. **Hábitos** con cadencia (diario/semanal).
5. **Recordatorios** programados (push/email).
6. Calendario y vista de agenda.
7. Métricas: rachas, tiempo dedicado, completadas vs. asignadas.
8. Búsqueda y filtrado (fecha, estado, etiqueta).
9. Sincronización con DB local (Room) y servidor.
10. Adjuntos simples (imagen/notas de voz).

11. Notificaciones locales y remotas.
12. Exportación/importación de tareas (JSON/CSV).
13. Temas y personalización.
14. Compartir tareas con otros usuarios (opcional).
15. Accesibilidad (gestos, tamaños de texto).

Módulos / Endpoints

- `/auth`, `/tasks`, `/habits`, `/stats`, `/reminders`, `/attachments`

Entidades

- `Task(id, user_id, title, description, due_date, priority, status, tags[])`
- `Habit(id, user_id, name, cadence)`
- `HabitLog(habit_id, date, completed)`
- `Reminder(id, user_id, target_id, scheduled_at, channel)`

Servicios y procesos

- **Scheduler**: recordatorios.
- **Job**: recomputación de métricas.
- **Cola**: envío de push/emails.

Android – Pantallas

Lista y detalle de tareas, calendario, hábitos, estadísticas, adjuntos, ajustes.

Multimedia

Notificaciones sonoras, adjuntos de audio/foto, gráficos en móvil (Vista/Compose).

Backend Java

- Spring Boot, Scheduler para recordatorios, endpoints de estadísticas agregadas.

Backend Python

- FastAPI + Celery Beat para recordatorios, agregaciones en tareas programadas.

Sprints

- **S1:** auth, tareas CRUD, hábitos, sincronización básica.

- **S2:** recordatorios, estadísticas, adjuntos.
- **S3:** compartición, exportación/importación, accesibilidad y pruebas.