Server como puente de datos

Esta guía detalla la estructura, configuraciones y archivos necesarios para poner en marcha el servidor de la aplicación, integrando **Spring Boot 4**, **Supabase (PostgreSQL/Storage)**, **Redis** y **RSocket**.

1. Estructura de Proyecto (Archivos a Colocar)

El colega del server debe asegurar que los siguientes archivos estén en estas rutas exactas dentro de `server/src/main/java/com/intermodular/server/`:

📂 Configuración y Seguridad
- `config/AppConfig.java`: Configuración de BCrypt y CORS.
- `security/SecurityConfig.java`: Filtros de seguridad, rutas públicas/privadas.
- `security/JwtUtil.java`: Lógica de generación y validación de tokens.
- `security/JwtAuthenticationManager.java`: Manager reactivo para JWT.

📂 Modelos (Entidades R2DBC)
- `model/Player.java`
- `model/Sighting.java`
- `model/Post.java`
- `model/BirdCard.java`
- `model/Bandada.java`

📂 Repositorios (Interfaces R2DBC)
- `repository/PlayerRepository.java`
- `repository/SightingRepository.java`
- `repository/PostRepository.java`
- `repository/BirdCardRepository.java`
- `repository/BandadaRepository.java`

📂 Controladores (Endpoints REST)
- `controller/AuthController.java`: `/api/auth/register`, `/api/auth/login`.
- `controller/PlayerController.java`: `/api/players/me`.
- `controller/SightingController.java`: `/api/sightings`.
- `controller/PostController.java`: `/api/posts`.
- `controller/BandadaController.java`: `/api/bandadas`.

📂 Tiempo Real (RSocket)

- `rsocket/BattleController.java`: Manejo de ataques y estados de batalla.

---

2. Configuración del Entorno (`application.yml`)

Ubicación: `server/src/main/resources/application.yml`

Asegúrate de configurar estas variables (usa los secretos de Supabase):

```yaml
spring:
  r2dbc:
    url: r2dbc:postgresql://[SUPABASE_HOST]:5432/postgres
    username: postgres
    password: [PASSWORD]
  data:
    redis:
      host: [REDIS_HOST]
      port: 6379
    rsocket:
      server:
        port: 7000

jwt:
  secret: [TU_SECRETO_SEGURO]
  expiration: 86400 # 24 horas
```

---

3. Integración con Supabase Storage

Para las fotos y audios, el backend actúa como un proxy. Debes asegurar que las políticas de RLS en Supabase permitan al `service_role` escribir en los buckets:
- `user-audios`: Audios de pájaros grabados por usuarios.
- `user-photos`: Fotos de expedición.

---

4. Notas para el Desarrollador del Server

1. **Spring Boot 4**: El proyecto ha sido actualizado a la versión 4.0.0. Asegúrate de usar un JDK compatible (mínimo Java 17, recomendado Java 21/25).
2. **Reactive Stack**: Todo el código usa **Project Reactor** (Flux/Mono). No uses bloqueos (`Thread.sleep`, etc.).
3. **Lombok**: Es obligatorio tener el plugin de Lombok en el IDE.
4. **CORS**: El servidor permite peticiones desde cualquier origen (`*`) para la APK.

Manual de Persistencia Híbrida y Movilidad

Este documento describe cómo la aplicación gestiona los datos entre el servidor (Spring Boot/Supabase) y el almacenamiento local (Room/SQLite) para garantizar una experiencia fluida incluso sin conexión.

1. Estrategia de Sincronización

La aplicación sigue un patrón de **"Caché con Sostenibilidad Remota"**:

1. **Arranque**: Al iniciar sesión o refrescar, el dispositivo consulta la API (`/collection`, `/inventory`).
2. **Persistencia**: Los datos recibidos se guardan inmediatamente en Room mediante `AvisCore.saveBirds()` y `saveInventory()`.
3. **Fallback**: Si el servidor no responde (ej: sin túnel Tailscale), el Store de Zustand carga los datos directamente de Room.

2. Gestión de Medios (Audios y Fotos)

Debido al peso de los archivos multimedia, se ha implementado este flujo:

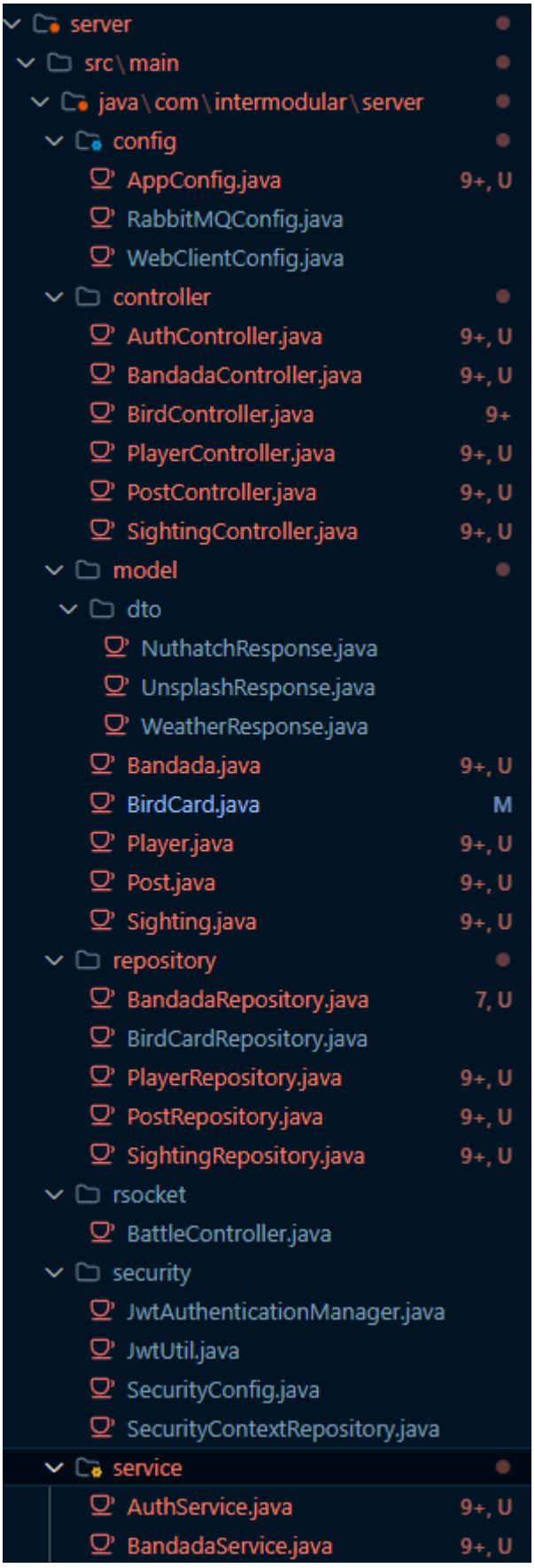
- **Local First**: Cuando un usuario registra un ave en una expedición, el plugin nativo guarda el audio/foto en el almacenamiento interno del teléfono y crea una entrada en la tabla `sightings` de Room con `isSynced = false`.
- **Sincronización Diferida**: Existe un método en el Store (`syncPendingSightings`) que el sistema llama periódicamente para subir los archivos pendientes al backend cuando detecta conectividad alta.

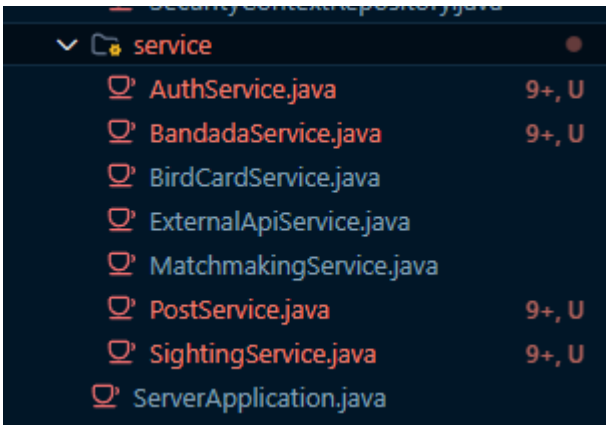
3. Seguridad de Datos

- Los tokens JWT se almacenan en el **EncryptedSharedPreferences** de Android, inaccesibles para otras apps.
- La base de datos Room no es accesible por depuración normal si el dispositivo no está rooteado, protegiendo las estadísticas locales del juego.

4. Troubleshooting Local

Si los datos no coinciden con el servidor:
1. Verificar que el servicio Tailscale esté activo.
2. El servidor debe estar accesible en la IP configurada en `api.ts`.
3. Si hay errores en el backend, la app seguirá funcionando con la última copia local guardada en SQLite.


server
src\main
java\com\intermodular\server
config
AppConfig.java 9+, U
RabbitMQConfig.java
WebClientConfig.java
controller
AuthController.java 9+, U
BandadaController.java 9+, U
BirdController.java 9+
PlayerController.java 9+, U
PostController.java 9+, U
SightingController.java 9+, U
model
dto
NuthatchResponse.java
UnsplashResponse.java
WeatherResponse.java
Bandada.java 9+, U
BirdCard.java M
Player.java 9+, U
Post.java 9+, U
Sighting.java 9+, U
repository
BandadaRepository.java 7, U
BirdCardRepository.java
PlayerRepository.java 9+, U
PostRepository.java 9+, U
SightingRepository.java 9+, U
rsocket
BattleController.java
security
JwtAuthenticationManager.java
JwtUtil.java
SecurityConfig.java
SecurityContextRepository.java
service
AuthService.java 9+, U
BandadaService.java 9+, U


service
AuthService.java 9+, U
BandadaService.java 9+, U
BirdCardService.java
ExternalApiService.java
MatchmakingService.java
PostService.java 9+, U
SightingService.java 9+, U
ServerApplication.java