

Backend reservas lago

Servicio Spring Boot (Java 21) para gestionar reservas del Lago Escondido. Usa PostgreSQL, Flyway para migraciones, JWT para autenticacion, envio de mails con Thymeleaf y Swagger/OpenAPI para documentacion.

Arquitectura

- **Frameworks:** Spring Boot 3 (web, security, data-jpa, validation, mail, thymeleaf), jjwt para tokens, springdoc-openapi.
- **Capas:**
 - **config:** seguridad JWT (filtro + `SecurityConfig`), CORS, OpenAPI, configuracion de swagger.
 - **controller:** endpoints publicos (`/api`) y administrativos (`/api/admin/**`, `/api/admin/users/**`), autenticacion (`/api/auth/login`), manejo centralizado de errores.
 - **dto:** contratos de entrada/salida para reservas, usuarios, login, exportaciones.
 - **service:** logica de negocio (cupos y disponibilidad, reservas, mapeos, exportacion XLSX, usuarios, email, JWT).
 - **repo:** `JpaRepository` para reservas, reglas de disponibilidad, configuracion del sistema y usuarios.
 - **model:** entidades y enums (reservation, availability rule, system config, user, circuit, visitor type, status, etc).
- **Base de datos:** PostgreSQL; Flyway migra al iniciar (`src/main/resources/db/migration`). Se crea un usuario admin por defecto `admin@lago-escondido.com / admin123`.
- **Correo:** plantillas HTML en `src/main/resources/templates/` para confirmar y cancelar reservas; se puede deshabilitar con `APP_MAIL_ENABLED=false`.
- **Documentacion:** swagger-ui en `/docs` (solo en perfil `dev`); definicion en `OpenApiConfig`.

Endpoints principales

- **Auth (público)**
 - `POST /api/auth/login` → devuelve JWT (usa email/contraseña).
 - Dev: `admin@lago-escondido.com / admin123`.
- **Público**
 - `GET /api/availability?date=YYYY-MM-DD` → capacidad y cupo restante del día.
 - `GET /api/availability?month=YYYY-MM` → lista de disponibilidad del mes.
 - `POST /api/reservations` → crea reserva (body `CreateReservationRequest`).
 - `GET /api/reservations/{id}` → resumen de reserva.
- **Administración (JWT Bearer requerido)**
 - `GET /api/admin/reservations` → lista (filtros opcionales `date`, `status`).
 - `POST /api/admin/reservations/{id}/confirm` → confirma y envía email.
 - `POST /api/admin/reservations/{id}/cancel` → cancela y envía email.
 - `GET /api/admin/reservations/export?date=YYYY-MM-DD[&status=...]`
`[&visitorType=...]` → XLSX.
 - `PUT /api/admin/availability/{date}` → upsert de capacidad del día `{ "capacity": 30 }`.
 - `POST /api/admin/eventos` → crea reserva tipo evento con cupo predefinido.
 - `GET /api/admin/config/educational-reservations` → estado on/off.

- `PUT /api/admin/config/educational-reservations` → toggle { "enabled": true|false }.
- Usuarios admin (`/api/admin/users`): `POST` crear, `GET` listar, `GET /{email}`, `PUT /{userId}`, `DELETE /{userId}`.
- **Docs**
 - Swagger UI: `/docs` (dev). OpenAPI JSON: `/v3/api-docs`.

Configuracion

- Profiles: `dev` por defecto (`SPRING_PROFILES_ACTIVE`), `prod` para despliegue.
- Perfil dev (`application-dev.yml`): base `jdbc:postgresql://localhost:5432/lago`, usuario `postgres/postgres`, MailHog opcional, CORS abierto a localhost.
- Perfil prod (`application-prod.yml`): usa `DATABASE_URL`, `DATABASE_USERNAME`, `DATABASE_PASSWORD`, `JWT_SECRET`, configuracion de mail y CORS via variables.
- Variables comunes utiles: `PORT`, `DEFAULT_CAPACITY`, `APP_MAIL_ENABLED`, `MAIL_HOST`, `MAIL_PORT`, `MAIL_USERNAME`, `MAIL_PASSWORD`, `ALLOWED_ORIGINS`, `JWT_EXPIRATION`.

Como correr en local (dev)

Opcion A: todo en Docker (sin instalar JDK)

1. `cd backend-reservas-lago`
2. `docker compose -f docker-compose.dev.yml up -d --build`
 - Backend queda en `http://localhost:8080`, base Postgres en `localhost:5432`, MailHog en `http://localhost:8025`.
 - El contenedor usa TZ UTC por defecto (JAVA_OPTS en compose).
 - Si reconfiguras la DB, ajusta `SPRING_DATASOURCE_URL`, `SPRING_DATASOURCE_USERNAME`, `SPRING_DATASOURCE_PASSWORD` en el compose.
3. Logs en vivo: `docker compose -f docker-compose.dev.yml logs -f app`
4. Login admin: email `admin@lago-escondido.com`, password `admin123` (token via `POST /api/auth/login`).

Opcion B: local con Maven

1. Entrar al proyecto: `cd backend-reservas-lago`
2. Levantar Postgres (y MailHog opcional) con Docker: `docker compose up -d`
3. Asegurar el perfil:
 - Windows PowerShell: `$Env:SPRING_PROFILES_ACTIVE="dev"`
 - Linux/macOS: `export SPRING_PROFILES_ACTIVE=dev`
4. Arrancar la app:
 - Windows: `mvnw.cmd spring-boot:run`
 - Linux/macOS: `./mvnw spring-boot:run`
5. Revisar: health en `http://localhost:8080/actuator/health`, docs swagger en `http://localhost:8080/docs`.
6. Login admin: email `admin@lago-escondido.com`, password `admin123` (token via `POST /api/auth/login`).

Tests

- Ejecutar suite:
 - Windows: `mvnw.cmd test`
 - Linux/macOS: `./mvnw test`

Ejecucion con Docker en produccion

1. Copiar variables: `cp env.prod.example .env.prod` y completar secretos (`DATABASE_PASSWORD`, `JWT_SECRET`, correo, etc).
2. Construir y levantar:
`docker compose -f docker-compose.prod.yml --env-file .env.prod up -d --build`
3. Servicios expuestos: app en `:8080`, Postgres en `:5432`. Nginx puede publicar en `80/443` si `nginx.conf` y certificados estan presentes.

Notas

- Flyway corre migraciones al iniciar; no necesita pasos manuales extra.
- Para desactivar correo en desarrollo: `APP_MAIL_ENABLED=false` o usar MailHog (`localhost:8025`).
- Exportaciones administrativas generan XLSX y respetan filtros por fecha, estado y tipo de visitante.