Documentación Detallada del Proyecto - Guía para Robert

Este documento explica en profundidad el funcionamiento técnico del backend, centrándose en la seguridad y el modelado de datos.

---

1. Seguridad: El Corazón del Sistema (`backend/core/security.py`)

Hemos implementado un sistema robusto basado en **Bcrypt** para proteger las credenciales de los usuarios.

¿Qué es Bcrypt y por qué lo usamos?

Bcrypt es un algoritmo de **hashing** (no encriptación, ya que el proceso es unidireccional). A diferencia de SHA-256 o MD5, Bcrypt está diseñado para ser "lento".

¿Cómo funciona de forma simplificada?

1. **Salt (Sal):** Por cada contraseña, Bcrypt genera una cadena aleatoria única llamada *Salt*. Si dos usuarios usan la misma contraseña "1234", sus hashes serán totalmente distintos gracias a la sal. Esto evita los ataques de "Tablas Arcoíris".
2. **Costo (Rounds):** El algoritmo aplica la función miles de veces. Esto hace que un atacante que intente adivinar contraseñas por fuerza bruta tarde muchísimo tiempo en cada intento, haciendo el ataque inviable.
3. **Haseo Unidireccional:** Una vez que convertimos "contraseña123" en `$2b$12$ExAmPlE...`, **nadie** (ni nosotros como desarrolladores) puede volver atrás para saber cuál era la contraseña original.

Gestión de Sesiones: JWT (JSON Web Tokens)

Usamos el algoritmo **HS256** para firmar tokens.

- Cuando haces login, el servidor te da un "pase" (el token).
- Ese pase contiene tu email y tu rol dentro del JSON.
- El servidor firma ese JSON con una variable secreta (`SECRET_KEY`).
- **Seguridad:** Si un atacante intenta cambiar su rol de "client" a "admin" en el token, la firma dejará de ser válida y el servidor rechazará la petición automáticamente.

---

2. Modelado de Datos con Pydantic (`backend/models`)

Pydantic es la librería que usamos para definir la estructura de nuestros datos. No solo define carpetas, sino que **valida en tiempo real** lo que llega a la API.

Usuarios (`user.py`)

- `UserBase`: Define lo básico (`email`, `full_name`, `role`). Usamos el tipo `EmailStr` para que Pydantic verifique automáticamente que el email tenga un formato válido (ej. que tenga @ y un dominio).
- `UserCreate`: Añade el campo `password`. Este es el modelo que el usuario envía al registrarse.

- `UserInDB`: Añade `hashed_password`. Es el modelo "interno" que habla con MongoDB.
- `UserResponse`: Lo que el mundo ve. **No incluye la contraseña**. Devuelve un `id` (string) para que el frontend pueda identificar al usuario fácilmente.

Actividades (`activity.py`)

Maneja las clases del gimnasio (Yoga, Crossfit, etc.).

- **Validación Crítica**: Hemos programado un `@field_validator` que lanza un error si la fecha de fin de la clase es anterior o igual a la de inicio.
- **Capacidad**: Usamos `Field(gt=0)` para asegurar que el aforo sea siempre un número positivo.
- `ActivityInDB`: Usa un `alias="_id"`. Esto es porque MongoDB guarda los IDs con un guion bajo delante, pero en Python preferimos trabajar sin él. Pydantic hace el mapeo automáticamente.

Reservas (`reservation.py`)

Es el nexo entre el Cliente y la Actividad.

- **Estados (Enum)**: Usamos una enumeración para que los estados sean estrictos: `active`, `cancelled`, `late_cancelled`, `attended`, `absent`. Esto evita errores tipográficos como escribir "cancelado" con una sola 'l'.
- `ReservationInDB`: Guarda quién reservó (`user_id`), qué actividad (`activity_id`) y cuándo ocurrió.

Resumen Estructural

Modelo	Función Principal	Seguridad / Validación
Base	Plantilla común	Tipado estricto (str, int, etc.)
Create	Entrada de datos	Recibe passwords en texto plano
InDB	Almacenamiento	Contiene hashes y metadatos de DB
Response	Salida de datos	**Filtra datos sensibles** (Passwords)

3. Lógica de Inicio Automática (`mongodb.py`)

Para facilitar el desarrollo, hemos modificado el arranque de la base de datos:

1. **Conexión:** Se conecta usando la URL del `.env`.
2. **Índices:** Crea reglas en MongoDB para que las búsquedas sean rápidas y los emails únicos.
3. **Auto-Admin:** Si la base de datos está vacía, crea el usuario `admin@admin.com` con contraseña `admin` automáticamente. Esto asegura que el equipo de frontend siempre pueda entrar a probar sin configurar nada manualmente.