

Backend Overview — Inventario (resumen para presentación)

1. Resumen

El backend es una API REST construida con NestJS y TypeORM para gestionar un sistema de inventario multi-grupos. Soporta:


- Autenticación por credenciales (username/password) con JWT.
- Autenticación OAuth (Google/GitHub) con sincronización (linking) de cuentas por email mediante el endpoint `/auth/sync-oauth`.
- Gestión de grupos, roles por grupo (GroupRole), productos y categorías.

2. Modelo de datos (entidades clave)

- **User**
 - id (uuid), username (unique), email (unique), password (nullable), emailVerified (bool), lastLogin (Date), hidden (bool), isOwner (bool)
 - Relación: **OneToMany** → **GroupRole**

auth			^
POST	/auth/signin	Iniciar sesión	▼
POST	/auth/signup	Registrar un nuevo usuario	▼
POST	/auth/profile	Obtener perfil de usuario	▼

- **Group**
 - id (uuid), nombre, logo (nullable), hidden (bool)
 - Relaciones: **OneToMany** → **GroupRole**, **OneToMany** → **Product**

groups			^
POST	/groups	Create a new group	▼
GET	/groups	Get all groups	▼
GET	/groups/{id}	Find a group by ID	▼
PUT	/groups/{id}	Update a group	▼
DELETE	/groups/{id}	Hide a group by ID	▼
GET	/groups/{id}/products	Get all products of a group ordered by stock	▼ 
GET	/groups/{groupId}/users	Find all users in a group	▼

- **GroupRole**
 - id (uuid), role (enum: **LEADER** | **ADMIN** | **USER**), hidden (bool)
 - Relaciones: **ManyToOne** → **User**, **ManyToOne** → **Group**
 - Propósito: permisos a nivel de grupo

groups/roles

POST /groups/roles Create a new role de grupo

PUT /groups/roles/{id} Update a role de grupo

- **Product**

- id (uuid), name, description, price (decimal), stock (int), image (nullable), hidden (bool)
- Relaciones: **ManyToOne** → **Group**, **ManyToMany** ↔ **Category** (con `@JoinTable()`)

products

POST /products Create a new product

GET /products Search products

GET /products/{id} Find a product by ID

PUT /products/{id} Update a product

DELETE /products/{id} Hide a product

- **Category**

- id, name, relación **ManyToMany** con **Product**

categories

POST /categories Create a new category

GET /categories Get all categories

GET /categories/{id} Get a category by ID

PUT /categories/{id} Update a category

DELETE /categories/{id} Hide a category (soft delete)

Notas:

- **password** es **nullable** para soportar cuentas OAuth que no usan password.
- **isOwner** identifica un super-usuario global (creación manual via CLI helper).

3. Endpoints y responsabilidades

- **POST /auth/signin** — login por credenciales. Devuelve token JWT del backend.
- **POST /auth/signup** — registro de usuario (hashea password).
- **POST /auth/sync-oauth** — sincroniza/crea usuario a partir de datos OAuth (email linking).
- **POST /auth/profile** — requiere JWT, devuelve perfil.

Además existen módulos para CRUD de **groups**, **products**, **categories** y **group-roles**. Con eso se puede seguir avanzando

4. Flujos de autenticación importantes

- Credentials

- Frontend (NextAuth credentials provider) envía credenciales al backend `/auth/signin`.
 - Backend valida (bcrypt), actualiza `lastLogin` y devuelve `access_token` JWT.
 - OAuth (Google/GitHub)
 - NextAuth gestiona el handshake OAuth.
 - Tras éxito, **frontend debe** llamar a `POST /auth/sync-oauth` con `{ email, name, picture, email_verified }`.
 - Backend busca usuario por email: si no existe crea (password=null); si existe actualiza `emailVerified/lastLogin`.
 - Resultado: usuario vinculado en la base de datos y disponible para control de permisos.
-

5. Estado actual y requisitos

- Código: implementadas autenticación por credenciales y `sync-oauth` (backend).
- Requisitos para correr localmente:
 - Node.js (>= v18+ recomendado)
 - MySQL accesible (TypeORM usa MySQL por defecto)
 - Variables de entorno en la raíz de `inventario-backend` (ejemplo abajo)
 - Dependencia nativa `bcrypt` requiere build tools; en desarrollo puede usarse `bcryptjs` si es necesario.

7. Estado operativo actual / notas técnicas

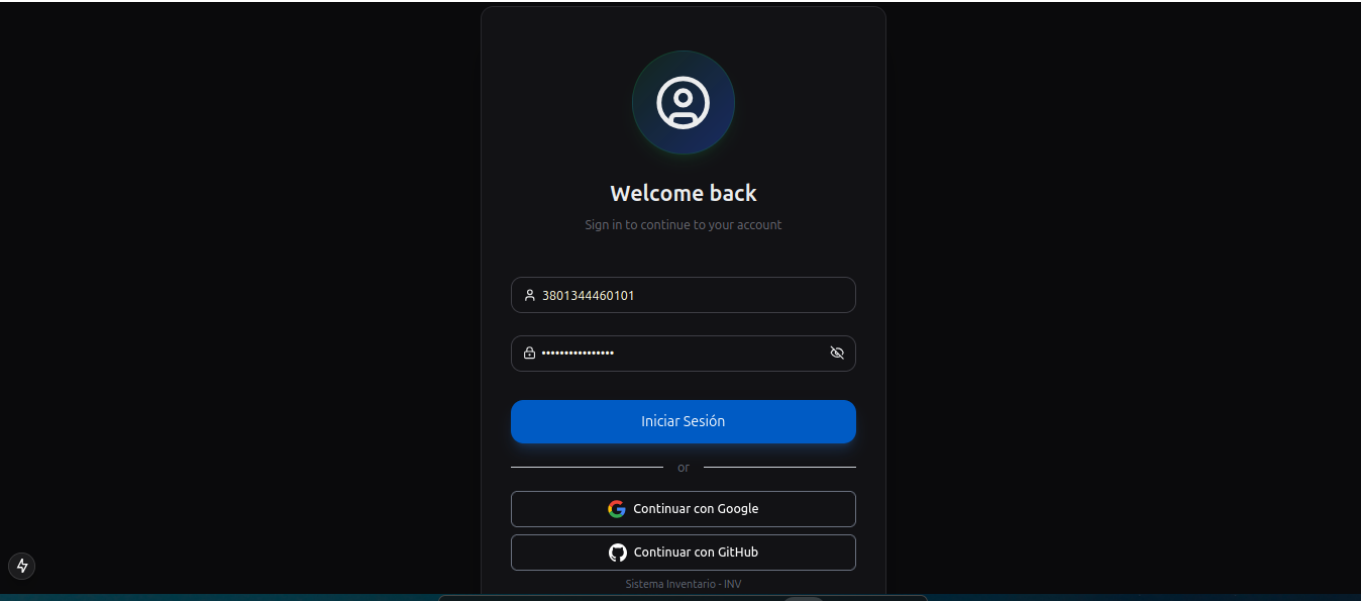
- `synchronize: true` en TypeORM está activo para desarrollo; NO usar en producción.
 - El sistema vincula OAuth por `email` (asume email único). Si el mismo email existe con password, `sync-oauth` actualiza `emailVerified` y `lastLogin`.
 - Existe método CLI `promptForOwnerCreation()` en `UsersService` para crear el `owner` (isOwner=true) al iniciar.
-

Parte 2 — Frontend (autenticación)

La app frontend usa Next.js y NextAuth para gestionar sesiones y providers; aquí se documenta únicamente la parte de autenticación y cómo integrarla con el backend.

- Stack y responsabilidades
 - Next.js + NextAuth (Providers: Credentials, Google, GitHub).
 - NextAuth maneja el handshake OAuth; el backend es responsable de emitir y validar el JWT de la API y de almacenar/vincular usuarios (`/auth/signin`, `/auth/sync-oauth`).

Login page:



Owner:

