# Gestão Fácil – API

Plataforma de gestão de ativos, serviços técnicos e SLA, escrita em Node.js/Express com Sequelize e autenticação JWT. Inclui painel web simples em `api/public` para testes rápidos, documentação Swagger e rotinas administrativas para manutenção de dados.

---

## Sumário



---

## Visão Geral

O Gestão Fácil centraliza o fluxo operacional de clientes, ativos e serviços técnicos. O backend expõe APIs REST com versionamento (`/v1/...`), autenticação JWT e documentação em `/docs`. O frontend estático distribuído em `api/public` consome as mesmas APIs e serve como painel de demonstração.

## Principais Recursos

- CRUD completo para Clientes, Ativos, Locais, Usuários, Serviços e Tipos de Serviço.
- Regras de negócio aplicadas nos controladores (validação de status, soft delete, restauração automática).
- Autenticação JWT com controle de acesso por cargo.
- Rota administrativa para corrigir serviços com ativos inconsistentes.
- Keep-alive configurável para impedir hibernação em provedores gratuitos.
- Logs verbosos opcionais (CRUD + SQL) para auditoria e troubleshooting.
- Scripts utilitários (`scripts/*.js`) para auditorias e testes.

## Arquitetura do Repositório

```
GestaoFacil/
├── api/
│   ├── app.js            # Servidor Express principal
│   └── public/           # Painel web e assets estáticos
├── src/
│   ├── config/           # Sequelize + leitura de variáveis de ambiente
│   ├── controllers/      # Regras de negócio
│   ├── middlewares/      # Autenticação JWT
│   ├── models/           # Definições Sequelize
│   ├── routes/           # Rotas REST organizadas por entidade
│   └── utils/            # Logger verboso/auditoria
├── scripts/              # Scripts de manutenção/testes
├── migrations/           # Migrations (sequelize-cli)
└── README.md
```

## Stack Tecnológico

- Node.js 18+
- Express 5 + Helmet + Rate Limit
- Sequelize + PostgreSQL (com suporte a `DATABASE_URL`)
- JWT (jsonwebtoken/bcrypt)
- Swagger (`swagger-jsdoc` + `swagger-ui-express`)
- Render Keep-Alive (axios)
- Nodemon, Sequelize CLI, Jest (placeholder)

## Pré-requisitos

- Node.js 18 ou superior
- PostgreSQL local ou serviço compatível
- `npx sequelize-cli` instalado globalmente (opcional)
- Variáveis de ambiente configuradas (vide `.env.example`)

## Configuração Rápida

1. **Instalar dependências**

   ```
   npm install
   ```

2. **Criar `.env`**

   ```
   DB_DIALECT=postgres
   DB_HOST=localhost
   DB_PORT=5432
   DB_USER=postgres
   DB_PASSWORD=postgres
   DB_NAME=gestaofacil
   ```

```
JWT_SECRET=troque_me
APP_MODE=local
PORT=3000
KEEP_ALIVE_ENABLED=false
VERBOSE_LOGS=false
```

> Para usar um `DATABASE_URL`, defina a variável e deixe os demais campos vazios.

3. **Rodar migrações e seeds**

```
npx sequelize-cli db:migrate
npm run seed
```

4. **Iniciar servidor**

```
npm run dev   # com nodemon
# ou
npm start
```

5. **Validar health-check**

```
GET http://localhost:3000/health
```

## Scripts Disponíveis

| Comando | Descrição |
|---|---|
| `npm run dev` | Inicia `api/app.js` com nodemon |
| `npm start` | Inicia o servidor em modo produção |
| `npm run seed` | Executa todos os seeds |
| `npm run seed:undo` | Reverte seeds |
| `npm run inspect:relations` | Inspeção de relacionamentos órfãos |
| `npm run inspect:orphans` | Lista entidades órfãs |
| `npm run test:create-servico` | Script de teste do trigger `create_servico` |

Scripts adicionais podem ser executados diretamente em `scripts/`.

## Migrações e Seeds

- Configure seu banco no `.env`.
- Utilize `npx sequelize-cli db:migrate` para aplicar migrations.

- `npm run seed` cria dados básicos (clientes, usuários, ativos etc.).
- Caso precise zerar, use `npx sequelize-cli db:migrate:undo:all` seguido de `db:migrate`.

## Autenticação e Autorização

- `POST /auth/register` – cria usuário (idealmente restrito ao time interno).
- `POST /auth/login` – retorna `{ token, user }`.
- Inclua o token no header `Authorization: Bearer <jwt>` para acessar `/v1/...`.
- Middleware aceita lista de cargos: `authMiddleware(['admin'])` para rotas restritas.

## Logs Verbosos e SQL

- Controle pela variável `VERBOSE_LOGS` ou flag CLI `--verbose-logs`.
- Quando ativo, `src/utils/logger.js` imprime entradas `[VERBOSE ...]`.
- Hooks globais (`src/utils/auditLogger.js`) registram `create/update/delete/restore`.
- O Sequelize usa o mesmo logger (`src/config/database.js`), então consultas SQL aparecem como `[VERBOSE ...] [SQL] SELECT ...`.

## Testando o Trigger `create_servico`

1. Gere um token admin via `POST /auth/login`.
2. Chame `POST /v1/servicos` com corpo:

```
{
  "descricao": "Visita de manutenção",
  "ativoId": 1,
  "clienteId": 1,
  "usuarioId": 1,
  "tipoServicoId": 1,
  "status": "pendente",
  "dataAgendada": "2025-11-10T10:00:00Z",
  "detalhes": { "prioridade": "alta" }
}
```

3. Observe no console os eventos:
   - `[TRIGGER] create_servico:request`
   - `[SQL] SELECT create_servico(...)`
   - `[TRIGGER] create_servico:response`
4. Para automatizar, edite `scripts/test-create-servico.js` e execute `npm run test:create-servico`.

## Rotas e Documentação

- Swagger disponível em `http://localhost:<PORT>/docs`.
- Rotas principais:
  - `/auth/*` – registro, login, rota protegida de teste (`/auth/dados-secretos`).
  - `/v1/clientes`, `/v1/ativos`, `/v1/servicos`, `/v1/locais`, `/v1/usuarios`, `/v1/tipos-servicos`.

    - `GET /health`, `GET /teste` – monitoramento/keep-alive.
- Painel estático: `http://localhost:<PORT>/public`.

## Exemplo de CRUD (Clientes)

Use um token válido (cargo `admin` ou `gestor`). Exemplos com `curl`:

1. **Criar cliente**

```
curl -X POST http://localhost:3000/v1/clientes \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "nome": "Condomínio Aurora",
    "cnpj": "12.345.678/0001-90",
    "contatos": "(11) 99999-0000"
  }'
```

2. **Listar clientes**

```
curl http://localhost:3000/v1/clientes \
  -H "Authorization: Bearer $TOKEN"
```

3. **Atualizar cliente**

```
curl -X PUT http://localhost:3000/v1/clientes/1 \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{ "contatos": "(11) 98888-1234" }'
```

4. **Soft delete**

```
curl -X DELETE http://localhost:3000/v1/clientes/1 \
  -H "Authorization: Bearer $TOKEN"
```

O mesmo padrão se aplica para `/v1/ativos`, `/v1/servicos`, `/v1/locais`, `/v1/usuarios` e `/v1/tipos-servicos`.

## Exemplo de CRUD (Usuários)

As rotas de usuários exigem token admin. Exemplos:

1. **Criar usuário**

```
curl -X POST http://localhost:3000/v1/usuarios \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "nome": "Ana Souza",
    "cargo": "gestor",
    "email": "ana.souza@example.com",
    "telefone": "(11) 91234-5678",
    "password": "senhaSegura123"
  }'
```

2. **Listar usuários (sem senhas)**

```
curl http://localhost:3000/v1/usuarios \
  -H "Authorization: Bearer $TOKEN"
```

3. **Buscar por ID**

```
curl http://localhost:3000/v1/usuarios/2 \
  -H "Authorization: Bearer $TOKEN"
```

4. **Atualizar**

```
curl -X PUT http://localhost:3000/v1/usuarios/2 \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{ "cargo": "admin", "telefone": "(11) 97777-0000" }'
```

5. **Soft delete**

```
curl -X DELETE http://localhost:3000/v1/usuarios/2 \
  -H "Authorization: Bearer $TOKEN"
```

Os controladores removem o campo `password` das respostas e aplicam soft delete, permitindo restauração futura se necessário.

## Rotas Administrativas

- `POST /v1/servicos/admin/fix-client-services`

```
{
  "clienteId": 12,
```

```
  "numeroSerie": "C52-HIK-2025",
  "nome": "Câmera Pátio Central",
  "dryRun": true
}
```

- `dryRun=true` (query ou body) apenas simula a execução.
- Apenas usuários com cargo `admin` conseguem acessar.

## Boas Práticas de Segurança

- **Nunca** versione `.env`. Use `.env.example` como referência.
- Rotacione `JWT_SECRET` e credenciais do banco em ambientes reais.
- Não ative `NODE_TLS_REJECT_UNAUTHORIZED=0` em produção.
- Restrinja o uso de `/auth/register`; prefira criar usuários manualmente.
- Revise permissões de CORS em `ALLOWED_ORIGINS`.
- Ative o modo manutenção via `MAINTENANCE_MODE=true` quando necessário (o middleware libera apenas `/health`, `/config.js`, `/public` e `/docs`).

## Licença

Distribuído sob a licença **MIT**. Sinta-se livre para usar e adaptar mantendo os créditos originais.