Sistema de Gestão de Ativos e Serviços

Este projeto oferece uma estrutura completa para o **gerenciamento de ativos**, **serviços técnicos** e **fluxo operacional**, com **controle de SLA** e autenticação segura via **JWT**.

Tecnologias Utilizadas

- **Node.js** — Plataforma principal do backend
- **Express** — Framework para rotas, middlewares e estrutura REST
- **Sequelize** — ORM para integração com banco de dados relacional
- **JWT (JSON Web Token)** — Autenticação segura nas rotas protegidas
- **Arquitetura em camadas** — Separação clara entre *models*, *controllers* e *routes*
- **RESTful API** — Padrão de rotas e operações CRUD
- **Testes via JSON** — Exemplos práticos para cada entidade e autenticação

Estrutura do Projeto

```
GestaoFacil/
├── src/
│   ├── app.js              # Ponto de entrada principal do backend
│   ├── config/             # Configurações (ex: database.js)
│   ├── models/             # Modelos Sequelize das entidades
│   ├── controllers/        # Lógica das rotas e regras de negócio
│   ├── routes/             # Rotas Express para cada entidade
│   ├── middlewares/        # Autenticação, validação, etc.
│
├── api/
│   ├── app.js              # Ponto de entrada alternativo para a API
│   ├── public/             # Interface web para testes da API
│   ├── assets/             # Imagens, ícones, etc.
│   ├── cert/               # Certificados SSL
│   ├── build/              # Arquivos de build
│
├── migrations/             # Scripts de migração do banco de dados
├── .env                    # Variáveis de ambiente
├── package.json            # Dependências e scripts do projeto
```

Entidades Principais

Cliente

- **Atributos**: `id`, `nome`, `cnpj`, `contatos`
- **Relacionamentos**: Possui vários **Ativos** e solicita vários **Serviços**

👤 Usuário

- **Atributos**: `id`, `nome`, `cargo`, `email`, `telefone`
- **Relacionamentos**: Pode ser **Solicitante** ou **Responsável** por serviços

🏢 Local

- **Atributos**: `id`, `nome`
- **Relacionamentos**: Contém vários **Ativos**

⚙️ Ativo

- **Atributos**: `id`, `nome`, `numeroSerie`, `status`, `detalhes`
- **Relacionamentos**: Pertence a um **Cliente**, está alocado em um **Local**, e é associado a vários **Serviços**

🧩 Tipo de Serviço

- **Atributos**: `id`, `nome`, `descricao`, `ativo`
- **Relacionamentos**: Classifica vários **Serviços**
- **Regras**: Permite verificar se um serviço foi concluído dentro do SLA

🛠️ Serviço

- **Atributos**: `id`, `descricao`, `status`, `dataAgendada`, `dataConclusao`, `detalhes`
- **Relacionamentos**: Associado a um **Cliente**, vinculado a um **Ativo**, possui um **Tipo de Serviço**, e possui um **Solicitante** e um **Responsável** (Usuários)

🔗 Entidades e Relacionamentos

Modelo	Campos principais	Relacionamentos
Cliente	`id`, `nome`, `cnpj`, `contatos`	`hasMany(Ativo)` → `ativos` `hasMany(Servico)` → `servicos`

Ativo	`id`, `nome`, `numeroSerie`, `status`, `detalhes`	`belongsTo(Cliente)` → `cliente`
		`belongsTo(Local)` → `local`
		`hasMany(Servico)` → `servicos`

Serviço	`id`, `titulo`, `descricao`, `status`, `data_inicio`, `data_fim`	`belongsTo(Cliente)` → `cliente`
		`belongsTo(Ativo)` → `ativo`
		`belongsTo(TipoServico)` → `tipoServico`
		`belongsTo(Usuario)` → `solicitante`, `responsavel`

Local	`id`, `nome`	`hasMany(Ativo)` → `ativos`
Usuário	`id`, `nome`, `cargo`, `email`, `telefone`	Relacionado a `Servico` como solicitante ou responsável

---

🧭 Diagrama Conceitual Resumido

```
Cliente 1---* Ativo *---1 Local
Cliente 1---* Servico *---1 Ativo
Usuario 1---* Servico (solicitante/responsavel)
Servico *---1 TipoServico
```

Testes de Autenticação

Registro de Usuário

Endpoint: `POST /auth/register`
Body:

```
{
  "nome": "Erico",
  "email": "erico@teste.com",
  "cargo": "admin",
  "telefone": "85999999999",
  "password": "123456",
  "confirmPassword": "123456"
}
```

Respostas:

- `201 Created`: Usuário registrado com sucesso
- `400 Bad Request`: Senhas não coincidem
- `409 Conflict`: E-mail já cadastrado

Login

Endpoint: `POST /auth/login`
Body:

```
{
  "email": "erico@teste.com",
  "password": "123456"
}
```

Respostas:

- `200 OK`: Retorna `{ "token": "<JWT>" }`
- `401 Unauthorized`: Usuário não encontrado ou senha incorreta

Rota Protegida

Endpoint: `POST /auth/dados-secretos`
Headers:
`Authorization: Bearer <seu_token_aqui>`
Resposta esperada:

```
{
  "message": "Acesso autorizado, erico@teste.com"
}
```

Respostas:

- `401 Unauthorized`: Token inválido ou ausente
- `403 Forbidden`: Cargo não autorizado (se restrição de roles estiver ativa)

Testes Recomendados

- Registro com senhas diferentes
- Registro com e-mail já existente
- Login com senha incorreta
- Acesso à rota protegida sem token
- Acesso com token expirado ou malformado

Estruturas JSON para Testes

(Mesmos blocos de exemplo que você já tinha, agora corrigidos e acentuados corretamente.)

Fluxo do Ciclo de Vida de um Serviço

```
Aberto → Em andamento → Concluído → Encerrado
```

Documentação Automática (Swagger)

Instale:

```
npm install swagger-ui-express swagger-jsdoc
```

Exemplo no `src/app.js`:

```
const swaggerUi = require('swagger-ui-express');
const swaggerJsdoc = require('swagger-jsdoc');
```

```
const swaggerOptions = {
  swaggerDefinition: {
    openapi: '3.0.0',
    info: {
      title: 'Gestão Fácil API',
      version: '1.0.0',
      description: 'Documentação da API de Gestão de Ativos e Serviços'
    }
  },
  apis: ['./src/routes/*.js']
};

const swaggerDocs = swaggerJsdoc(swaggerOptions);
app.use('/docs', swaggerUi.serve, swaggerUi.setup(swaggerDocs));
```

Assim, a documentação estará disponível em `/docs`.
