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.

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.
- **Estrutura 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 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`, `codigo`, `nome`, `tipo`, `status`
- **Relacionamentos**: Pertence a um **Cliente**, está alocado em um **Local**, associado a vários **Serviços**

Tipo de Serviço

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

Serviço

- **Atributos**: `id`, `titulo`, `descricao`, `status`, `data_inicio`, `data_fim`
- **Relacionamentos**: Associado a um **Cliente**, vinculado a um **Ativo**, possui um **Tipo de Serviço**, 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`, `codigo`, `nome`, `tipo`, `status`	`belongsTo(Cliente)` → `cliente`
		`belongsTo(Local)` → `local`
		`hasMany(Servico)` → `servicos`

Servico	`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`
Usuario	`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 das Entidades

Cliente

```
{
  "nome": "Empresa Exemplo",
  "cnpj": "12345678000199",
  "contatos": "contato@empresa.com"
}
```

Usuário

```
{
  "nome": "João Silva",
  "cargo": "tecnico",
  "email": "joao@empresa.com",
  "telefone": "85988888888",
  "password": "senha123",
  "confirmPassword": "senha123"
}
```

Ativo

```
{
  "codigo": "ATV001",
  "nome": "Impressora HP",
  "tipo": "Impressora",
  "status": "ativo",
  "clienteId": 1,
  "localId": 2
}
```

Local

```
{
  "nome": "Sala de TI"
}
```

Tipo de Serviço

```
{
  "nome": "Manutenção Preventiva",
  "descricao": "Serviço de manutenção periódica",
  "tempo_medio": 2,
  "sla_horas": 24
}
```

Serviço

```
{
  "titulo": "Troca de toner",
  "descricao": "Troca de toner da impressora HP",
  "status": "Aberto",
  "data_inicio": "2025-09-06T10:00:00Z",
  "data_fim": null,
  "clienteId": 1,
  "ativoId": 1,
  "tipoServicoId": 1,
  "solicitanteId": 2,
  "responsavelId": 3
}
```

Fluxo do Ciclo de Vida de um Serviço

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

📚 Recomendações e Dicas de Documentação

- Teste todos os endpoints com dados válidos e inválidos.
- Use tokens válidos para rotas protegidas.
- Valide respostas e status HTTP em cada cenário.
- Para documentação automática da API, utilize Swagger com os pacotes `swagger-ui-express` e `swagger-jsdoc`.
- Adicione campos como `"homepage"`, `"repository"` e `"bugs"` ao seu `package.json` para facilitar o acesso à documentação e suporte.

📖 Exemplo de Integração Swagger

Instale:

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

No seu `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, sua documentação ficará disponível em `/docs