Sistema de Estoque SpringBoot

Desenvolvido por: Décio Carvalho Faria

📁 Sumário

📌 1. Introdução

Sistema de Estoque em **Spring Boot** com autenticação JWT, permissionamento (admin = false/true), uso de BCrypt na senha e gerenciamento de produtos, categorias, usuários e empresas.

🛠️ 2. Preparação do Ambiente

Requisitos:

- Java 17
- Maven
- MySQL 8+
- IDE (ex: IntelliJ, Eclipse, VS Code)

Passos:

1. Clone o repositório
2. Crie um banco de dados MySQL chamado “ `estoque` ”
3. Configure `application.properties` com as credenciais do banco
4. Rode: `mvn clean install ,depois, mvn spring-boot:run`
5. Inicie o projeto via IDE ou com: `mvn spring-boot:run`

📟 3. Funcionamento do Sistema

3.1 Autenticação e Registro

- Registro de usuários admin (sem empresa) via `/register`
- Login com geração de token via `/auth/login`
- Atualização de token via `/auth/refresh`

3.2 Usuários

- Admins podem criar, editar, listar e excluir usuários da própria empresa
- Usuários comuns não têm permissão para criar novos usuários

3.3 Empresas

- Apenas administradores podem criar sua empresa
- Cada admin pode ter somente 1 empresa
- Admin pode editar ou excluir sua própria empresa

3.4 Categorias

- CRUD de categorias vinculadas à empresa do usuário logado

3.5 Produtos

- CRUD de produtos vinculados a uma categoria (e portanto, a uma empresa)

🧱 4. Padrão de Projeto Monolítico

Este sistema adota o **padrão monolítico**, em que toda a aplicação (lógica de negócio, camada de controle, persistência e configuração) reside dentro de um único projeto e código-base.

Organização do Projeto

Toda a aplicação está empacotada em um único `jar`, com a seguinte estrutura de pacotes:

```
estoque.estoque
├── controller
├── service
├── model
├── repository
├── dto
├── filter
```

```
├── config
└── EstoqueApplication.java
```

🤭 5. Arquitetura MVC (Model-View-Controller)

Model (entidades JPA)

- `User.java`
- `Company.java`
- `Category.java`
- `Product.java`

Controller (camada de entrada REST)

- `AuthController.java`
- `RegisterController.java`
- `UserController.java`
- `CompanyController.java`
- `CategoryController.java`
- `ProductController.java`

Service (lógica de negócio)

- `AuthService.java`
- `UserService.java`
- `CompanyService.java`
- `CategoryService.java`
- `ProductService.java`

Repository (acesso ao banco de dados)

- `UserRepository.java`
- `CompanyRepository.java`
- `CategoryRepository.java`
- `ProductRepository.java`

DTOs (transporte de dados)

- `AuthRequestDTO.java`
- `AuthResponseDTO.java`
- `UserDTO.java`
- `CompanyDTO.java`
- `CategoryDTO.java`
- `ProductDTO.java`

Configurações

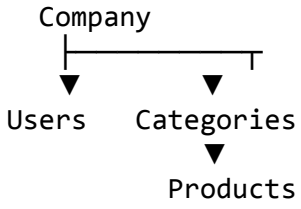
- `JwtUtil.java`
- `JwtFilter.java`
- `SecurityConfig.java`

📂 6. Modelo Entidade-Relacionamento (ER)

Estrutura Relacional

- **User** → pertence a uma `Company` (ManyToOne)
- **Company** → possui muitos `Users` e `Categories` (OneToMany)
- **Category** → pertence a uma `Company` (ManyToOne)
- **Product** → pertence a uma `Category` (ManyToOne)

Diagrama ER Visual:



🔐 7. Sistema de Autenticação JWT

O sistema utiliza **JWT (JSON Web Token)** para autenticação e autorização das rotas:

Componentes:

- `AuthRequestDTO.java` → recebe email/senha
- `AuthResponseDTO.java` → retorna tokens
- `AuthController.java` → endpoints `/login` e `/refresh`
- `AuthService.java` → lógica para autenticação e geração de tokens
- `JwtUtil.java` → geração, extração e verificação de JWT
- `JwtFilter.java` → intercepta requisições para validar token
- `SecurityConfig.java` → define políticas de segurança da API

Fluxo:

```
1. POST /auth/login
   → AuthController chama AuthService
   → Valida email/senha → Gera JWT
   → Retorna accessToken e refreshToken
```

```
2. Requisições com JWT
   → Cliente envia token no Header Authorization
   → JwtFilter valida e autentica

3. POST /auth/refresh
   → Gera novo accessToken a partir do refreshToken
```

🌐 8. Lista de rotas

Autenticação:

- `POST /auth/login`
- `POST /auth/refresh`

Usuários:

- `POST /register`
- `GET /users`
- `GET /users/{id}`
- `POST /users`
- `PUT /users/{id}`
- `DELETE /users/{id}`

Empresas:

- `POST /companies`
- `GET /companies`
- `PUT /companies/{id}`
- `DELETE /companies/{id}`

Categorias:

- `GET /categories`
- `GET /categories/{id}`
- `POST /categories`
- `PUT /categories/{id}`
- `DELETE /categories/{id}`

Produtos:

- `GET /products`
- `GET /products/{id}`
- `POST /products`
- `PUT /products/{id}`

- `DELETE /products/{id}`

Todas as rotas protegidas exigem token JWT no header:

```
Authorization: Bearer <seu-token>
```