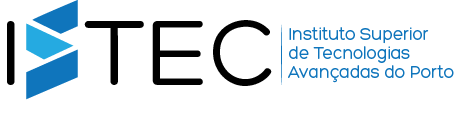
**StockMaster**

Tanjil Khan, Luís Martins, Ricardo Magalhães

Desenvolvimento de Software

MSS



**Índice**

# 

[**Tema 1**](#_abgh9wvlq1qf)

[**Arquitetura 2**](#_6h0fc2ujkbjm)

[**Estrutura do Servidor 3**](#_m1yf4gdog5ga)

[**Razão da escolha da arquitetura 4**](#_u51t1w1ucxb3)

# Tema

O projeto é um sistema de gestão de armazéns e produtos com modelo b2b. Empresas utilizam a plataforma para organizar e gerir os seus armazéns e os produtos contidos neles de forma eficiente. O sistema oferece uma interface amigável para que cada empresa possa controlar suas operações internas, incluindo a adição, edição e exclusão de armazéns e produtos.

O objetivo principal é fornecer uma solução digital para empresas gerirem seus próprios recursos de forma autônoma, prática e segura. Isso inclui:

Gestão de Armazéns**:** Cada empresa pode criar, editar, visualizar e deletar os armazéns que pertencem a ela.

Gestão de Produtos**:** Dentro de cada armazém, a empresa pode cadastrar os produtos que estão armazenados, além de editar e controlar os detalhes de cada item (nome, quantidade, etc.).

Isolamento de Dados: Cada empresa tem acesso apenas aos seus próprios armazéns e produtos. Nenhuma empresa consegue visualizar ou interagir com os dados de outras empresas.

# Diagrama de classes



# 

# Arquitetura

A arquitetura deste projeto irá ser baseado em cliente-servidor. Esta arquitetura separa as responsabilidades entre o cliente (frontend) e o servidor (backend). Essa divisão proporciona maior facilitamento do desenvolvimento de cada camada separadamente.

Cliente (Frontend):

* + Desenvolvido em ReactJS.
  + Comunica-se com o servidor através de APIs REST, enviando e recebendo dados em formato JSON.

Servidor (Backend):

* + Construído com Node.js e ExpressJS.
  + Estrutura organizada em controllers, middlewares, models e routes.
  + Usa Mongoose para interagir com a base de dados MongoDB.

Base de Dados (MongoDB):

* + Armazena dados estruturados em coleções (empresas, armazéns e produtos).
  + Relacionamentos definidos nos modelos (ex.: produtos pertencem a armazéns e empresas).

# 

## 

## Estrutura do Servidor

O servidor foi projetado com uma estrutura em camadas, que separa as responsabilidades e melhora a organização do código. As principais camadas são:

1. **Controllers**:
   * Local: controllers/
   * Cada controller é responsável por uma funcionalidade específica, como gerir empresas, armazéns e produtos.
   * Exemplos:
     + armazemController.js: gere as operações relacionadas aos armazéns.
     + authController.js: gere autenticação e segurança.
     + empresaController.js: gere os dados das empresas.
     + produtoController.js: gere as informações dos produtos.
2. **Middlewares**:
   * Local: middlewares/
   * Função: Os middlewares são funções auxiliares que interceptam as requisições antes de chegarem aos controllers. Eles podem ser usados para validações, autenticação, tratamento de erros e outras operações.
   * Exemplos: Middleware para autenticação de utilizadores, tratamento de erros globais.
3. **Models**:
   * Local: models/
   * Modelos:
     + Armazem.js: Representa os armazéns.
     + Empresa.js: Representa as empresas registadas.
     + Produto.js: Representa os produtos armazenados.
4. **Routes**:
   * Local: routes/
   * Exemplos:
     + empresaRoutes.js: Define as rotas relacionadas às empresas (ex.: /api/empresas).

## 

## Razão da escolha da arquitetura

Escolhemos a arquitetura cliente-servidor porque separa bem o que cada parte faz: no cliente é onde nós tratamos do que o utilizador vê e interage, enquanto o servidor trata dos dados. Além disso, podemos acrescentar novas funcionalidades no servidor sem impactar no cliente. Também o modelo cliente-servidor permite implementar mecanismos robustos de segurança, como JWT e Isolamento dos dados no servidor.

Por fim, escolhemos a arquitetura cliente-servidor principalmente pela comunicação Padronizada (HTTP/REST), a arquitetura permite que o cliente e o servidor se comuniquem com o uso de APIs REST