

**Relatório**

**Loja Esportiva**

Caique Silva dos Reis

1272322080

Salvador-BA

25.1-2025

## **1. Introdução**

Este relatório descreve o desenvolvimento de um sistema distribuído para simular as operações de uma rede de lojas especializadas na venda de roupas esportivas. Dada a crescente valorização do mercado fitness e o foco da população em bem-estar e saúde, esse setor é oportuno para a aplicação de conceitos de sistemas distribuídos e análise de dados.

O sistema cobre as principais operações de uma loja desse segmento: cadastro de clientes e vendedores, controle de produtos e estoque, processamento de vendas e geração de relatórios analíticos. Para isso, adotamos uma arquitetura baseada em microsserviços, onde cada funcionalidade é dividida em serviços independentes, executados em containers Docker e comunicando-se via APIs RESTful. Todos acessam um banco de dados PostgreSQL centralizado.

Além das funcionalidades básicas, o sistema inclui controle rigoroso de estoque, recebimento de pedidos, cancelamento de compras e geração de relatórios especializados. Essa abordagem simula um ambiente real de varejo, onde decisões estratégicas dependem diretamente da análise de dados operacionais.

## **2. Fundamentação Teórica**

Sistemas distribuídos são compostos por múltiplos serviços independentes que interagem para compor um sistema coeso. Esse modelo favorece escalabilidade, isolamento de falhas e facilidade de manutenção. A arquitetura adotada segue o padrão de microsserviços, em que cada serviço tem responsabilidades bem definidas.

A implementação utilizou JavaScript com Express (Node.js) para APIs REST, enquanto PostgreSQL foi escolhido por sua robustez e suporte a transações. A orquestração dos serviços é feita com Docker e docker-compose, simplificando o gerenciamento dos containers.

Destaca-se o módulo de relatórios analíticos, que realiza consultas agregadas nos dados de vendas e produtos para gerar insights como os produtos mais vendidos, consumo médio dos clientes e itens com baixo estoque. Essa separação permite escalar o serviço de análise conforme a demanda.

## **3. Projeto de Implementação**

### **3.1 Estrutura Geral da Aplicação**

A aplicação é composta pelos seguintes serviços:

* clientes-service: gerenciamento de clientes (CRUD).
* vendedores-service: gerenciamento de vendedores (CRUD).
* produtos-service: gerenciamento de produtos e controle de estoque.
* vendas-service: processamento de pedidos de compra e cancelamentos.
* relatorios-service: geração de relatórios estatísticos e operacionais.

Cada serviço roda isoladamente em containers Docker, comunicando-se via HTTP. Todos acessam um banco PostgreSQL comum, garantindo integridade e consistência dos dados entre os módulos.

### **3.2 Modelagem dos Dados**

As principais tabelas utilizadas no projeto são:

* clientes: armazena dados dos clientes.
* vendedores: armazena dados dos vendedores.
* produtos: armazena os produtos com informações e quantidade em estoque.
* vendas: registros das vendas realizadas, associando cliente e vendedor.
* itens\_venda: detalhes dos produtos vendidos em cada venda.

.

### **3.3 Tecnologias Utilizadas**

* Node.js e Express: construção das APIs REST.
* PostgreSQL: banco de dados relacional confiável.
* Docker e Docker Compose: containerização e orquestração dos serviços.

### **3.4 Funcionalidades Implementadas**

* Gerenciamento de Estoque: cadastro, edição e consulta de produtos com atualização em tempo real da quantidade disponível.
* Recebimento de Pedidos: validação do estoque antes de registrar vendas e atualização da quantidade em estoque. Cada venda registra os itens, cliente e vendedor envolvidos.
* Cancelamento de Pedido: rota específica para cancelar uma venda, removendo itens e restaurando o estoque dos produtos envolvidos.
* Geração de Relatórios: serviço dedicado para análises como:  
  + Produtos mais vendidos
  + Produtos comprados por cliente
  + Consumo médio por cliente
  + Produtos com baixo estoque
  + Total de vendas por cliente, vendedor e produto

### **3.4.1 Geração e Consulta de Relatórios Analíticos**

Para suporte à gestão, foi implementado o sistema de relatório, que gera dados analíticos via consultas SQL agregadas, retornando informações essenciais para o negócio.

Os relatórios são acessados por APIs REST específicas, retornando dados em JSON para fácil consumo. Os principais endpoints são:

* GET /relatorios/mais-vendidos
* GET /relatorios/por-cliente
* GET /relatorios/consumo-medio
* GET /relatorios/baixo-estoque

Essa estrutura permite escalabilidade e manutenção independente do serviço de relatórios.

### **3.5 Fluxo de uma Venda**

1. O cliente escolhe os produtos e envia um pedido.
2. O serviço de vendas valida o estoque disponível.
3. A venda é registrada no banco de dados.
4. O estoque dos produtos vendidos é atualizado.
5. O serviço de relatórios coleta as informações para geração das estatísticas.

## **4. Considerações Finais**

O sistema desenvolvido simula fielmente as operações essenciais de uma loja de roupas esportivas, adotando boas práticas de modularidade, escalabilidade e comunicação entre microsserviços. Docker e PostgreSQL proporcionaram estabilidade e flexibilidade, enquanto APIs REST facilitaram a integração entre módulos.

Como diferencial, foi implementado um módulo robusto de relatórios e uma funcionalidade de cancelamento de vendas com recuperação de estoque, alinhando o projeto com situações reais do varejo.

Os principais desafios foram a orquestração de containers, o gerenciamento transacional entre tabelas relacionadas e a padronização de rotas e respostas para integração com o serviço de relatórios — que, confesso, foi uma tarefa insuportável mas necessária.

Para futuras melhorias, podem ser incluídas autenticação JWT para proteção, interface gráfica para facilitar o uso, e uma camada de cache com Redis para otimizar os relatórios. Apesar das dificuldades, a solução atende aos requisitos propostos e simula bem o ambiente de uma aplicação real para o mercado de roupas esportivas.

(Confesso que fazer este trabalho me deu vontade de trabalhar em uma loja para aplicar tudo isso, pena que o mercado está difícil, mas vai dar certo!)

## 5. Bibliografia

* BROWN, Ethan Wilson. *Desenvolvendo com Node.js*. São Paulo: Novatec Editora, 2015.
* COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. *Sistemas Distribuídos: Conceitos e Projeto*. 5. ed. Porto Alegre: Bookman, 2012.
* Docker Inc. *Docker Documentation*. Disponível em:<https://docs.docker.com/>. Acesso em: 11 jun. 2025.
* Express.js. *Express – API Reference*. Disponível em:<https://expressjs.com/>. Acesso em: 11 jun. 2025.
* FERREIRA, Iago. *Aprenda Cloud e DevOps do ZERO*. YouTube, 2023. Disponível em:<https://www.youtube.com/watch?v=N3tNmoLbKwg>. Acesso em: 11 jun. 2025.
* GODOY, Leandro. *PostgreSQL: Guia do Programador*. São Paulo: Novatec Editora, 2019.
* KIPPER, Fernanda. *Docker e Docker Compose na Prática*. YouTube, 2022. Disponível em:<https://www.youtube.com/watch?v=DdoncfOdru8>. Acesso em: 11 jun. 2025.
* NEWMAN, Sam. *Building Microservices*. 2nd ed. Sebastopol: O’Reilly Media, 2021.
* Node.js Foundation. *Node.js Documentation*. Disponível em:<https://nodejs.org/en/docs>. Acesso em: 11 jun. 2025.
* OPENAI. ChatGPT. São Francisco, 2025. Disponível em:<https://chat.openai.com>. Acesso em: 11 jun. 2025. Utilizado como apoio para sugestões e aprimoramento textual.
* PostgreSQL Global Development Group. *PostgreSQL Documentation*. Disponível em:<https://www.postgresql.org/docs/>. Acesso em: 11 jun. 2025.
* Postman, Inc. *Postman Learning Center*. Disponível em:<https://learning.postman.com/>. Acesso em: 11 jun. 2025.
* RICHARDSON, Leonard; RUBY, Mike. *RESTful Web Services*. Sebastopol: O’Reilly Media, 2007.
* TANENBAUM, Andrew S.; VAN STEEN, Maarten. *Sistemas Distribuídos: Princípios e Paradigmas*. 2. ed. São Paulo: Pearson, 2007.
* TURNBULL, James. *The Docker Book: Containerization is the New Virtualization*. Independently published, 2014.