UNIFACS

SISTEMAS DISTRIBUÍDOS E MOBILE

AQUILA VIEIRA DE JESUS SILVA - 1272418353

GUSTAVO MATHEUS DE ALELUIA GALVÃO - 12724143932

HUDSON CASTRO FRANÇA - 12723211957

MILENA OLIVEIRA DE SOUZA - 1272326149

PROJETO DE SIMULAÇÃO DE VENDAS PARA LOJA DE TI

Salvador/BA
2025

AQUILA VIEIRA DE JESUS SILVA - 1272418353

GUSTAVO MATHEUS DE ALELUIA GALVÃO - 12724143932

HUDSON CASTRO FRANÇA - 12723211957

MILENA OLIVEIRA DE SOUZA - 1272326149

PROJETO DE SIMULAÇÃO DE VENDAS PARA LOJA DE TI

Relatório de projeto de Tecnologia da Informação apresentado para a matéria de Sistemas Distribuídos e Mobile para obtenção da nota da A3.

Professor: Adailton de Jesus Cerqueira Junior

Salvador/BA
2025

SUMÁRIO

Salvador/BA
2025

INTRODUÇÃO

Este relatório detalha o desenvolvimento de uma aplicação para simular o gerenciamento de vendas de uma loja de TI, abrangendo produtos como periféricos, acessórios, componentes e dispositivos eletrônicos. O projeto foi desenvolvido como parte da avaliação da disciplina de Sistemas Distribuídos e Mobile da UNIFACS. A motivação para a escolha do tema reside na relevância de sistemas de gerenciamento de vendas no contexto atual do comércio eletrônico e na oportunidade de aplicar conceitos de sistemas distribuídos na construção de uma solução escalável e modular.

A metodologia de abordagem consistiu na implementação de duas APIs distintas: uma API principal para gerenciar clientes, vendedores, produtos e vendas, e uma API secundária dedicada à geração de relatórios estatísticos. Ambas as aplicações foram desenvolvidas em JavaScript e executadas em containers Docker, utilizando um banco de dados MySQL para persistência dos dados. Os objetivos principais do projeto incluíram a criação de funcionalidades CRUD para clientes, vendedores, estoque e vendas, bem como a geração de relatórios de produtos mais vendidos, produtos por cliente, consumo médio do cliente e produtos com baixo estoque. A escolha da arquitetura de microsserviços visou promover a escalabilidade, modularidade e manutenção independente das funcionalidades.

FUNDAMENTAÇÃO TEÓRICA

Arquiteturas baseadas em microsserviços têm se destacado por sua capacidade de distribuir responsabilidades entre múltiplos serviços independentes, que se comunicam entre si. O presente projeto se insere nesse cenário, com a aplicação para simular o gerenciamento de vendas de uma loja de TI utilizando uma abordagem orientada a serviços.

O primeiro microsserviço (API-MAIN) é responsável pelo gerenciamento de entidades centrais do sistema, como clientes, vendedores, produtos e vendas, implementando funcionalidades CRUD (Create, Read, Update, Delete) para cada um desses recursos. O segundo microsserviço (API-REPORT) tem como foco a geração de relatórios estatísticos, incluindo métricas como produtos mais vendidos, produtos adquiridos por cliente, consumo médio por cliente e identificação de produtos com baixo estoque. Essa separação funcional reflete uma das principais vantagens dos sistemas distribuídos, que é a independência entre módulos, o que facilita tanto o desenvolvimento quanto a manutenção evolutiva do sistema.

A linguagem utilizada no desenvolvimento foi o JavaScript, executada no ambiente Node.js, que permite rodar código JavaScript fora do navegador, no servidor e usar JavaScript para tarefas de backend. O framework Express.js para estruturar e

facilitar a criação das rotas HTTP. Para a persistência dos dados, foi utilizado o MySQL, um sistema de gerenciamento de banco de dados. Ele armazena dados em tabelas organizadas por colunas e linhas, e usa a linguagem SQL (Structured Query Language) para gerenciar e consultar esses dados.

A comunicação entre os serviços e o banco de dados foi viabilizada por meio do driver *mysql2*, que oferece suporte a operações assíncronas com *promises*. Por meio da biblioteca Joi, garantimos que os dados recebidos de uma requisição API estão no formato correto, com os tipos certos e atendem a regras específicas antes de serem processados. Além disso, a ferramenta dotenv foi utilizada para o gerenciamento de variáveis de ambiente.

A utilização do Docker e Docker Compose para a conteinerização das aplicações facilita o processo de implantação em diferentes ambientes e padroniza os mesmos. Fazendo com que nosso projeto esteja de acordo com os princípios da disciplina de Sistemas Distribuídos, ao proporcionar isolamento, escalabilidade e portabilidade dos serviços desenvolvidos.

O projeto colocou em prática os principais conceitos de Sistemas Distribuídos, unindo teoria e prática de forma bem aplicada. A escolha pela arquitetura de microsserviços, junto com o uso de tecnologias modernas, mostra como os princípios de desmembramento, escalabilidade e independência entre serviços foram usados para criar uma base forte para sistemas mais robustos e confiáveis.

PROJETO DE IMPLEMENTAÇÃO

Visão Geral da Arquitetura

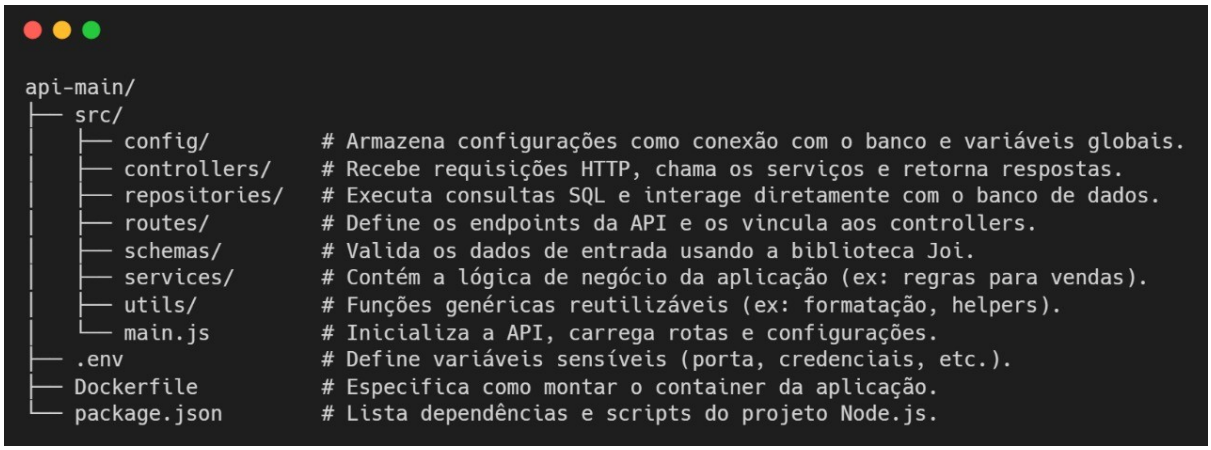
O projeto foi organizado com base na arquitetura de microsserviços. As **API Rest** foram divididas em dois serviços distintos.

- **api-main:** gerenciamento de entidades e operações principais.
- **api-report:** geração de relatórios analíticos.

Ambos os serviços compartilham um banco de dados MySQL, acessado de forma independente por cada API, conforme sua responsabilidade.

Estrutura de Pastas da API-MAIN

A estrutura de arquivos da api-main foi organizada da seguinte forma:



```
api-main/
├── src/
│   ├── config/          # Armazena configurações como conexão com o banco e variáveis globais.
│   ├── controllers/     # Recebe requisições HTTP, chama os serviços e retorna respostas.
│   ├── repositories/    # Executa consultas SQL e interage diretamente com o banco de dados.
│   ├── routes/          # Define os endpoints da API e os vincula aos controllers.
│   ├── schemas/         # Valida os dados de entrada usando a biblioteca Joi.
│   ├── services/        # Contém a lógica de negócio da aplicação (ex: regras para vendas).
│   ├── utils/           # Funções genéricas reutilizáveis (ex: formatação, helpers).
│   └── main.js          # Inicializa a API, carrega rotas e configurações.
├── .env                 # Define variáveis sensíveis (porta, credenciais, etc.).
├── Dockerfile           # Especifica como montar o container da aplicação.
└── package.json         # Lista dependências e scripts do projeto Node.js.
```

api-main

config/
Contém configurações gerais do sistema, como a conexão com o banco de dados e definições de variáveis globais da aplicação.

src/controllers/
Os controllers são responsáveis por receber as requisições HTTP, acionar os serviços apropriados e retornar as respostas.
Cada controller está isolado em seu próprio arquivo, promovendo organização e modularidade:

- ClientController.js: Gerencia operações de cadastro, listagem, atualização e exclusão de clientes.
- SellerController.js: Controla ações referentes aos vendedores.
- ProductController.js: Responsável pelo gerenciamento de produtos e estoque.
- SaleController.js: Realiza o controle do processo de vendas e consultas relacionadas.

src/repositories/
Os repositórios concentram as operações diretas com o banco de dados, organizadas por domínio.
Cada entidade possui um repositório específico:

- clientRepository.js: Executa consultas SQL voltadas ao gerenciamento de clientes.
- sellerRepository.js: Trata das operações com dados de vendedores.
- productRepository.js: Controla o acesso e a manipulação de produtos no banco de dados.
- saleRepository.js: Realiza leituras e escritas relacionadas às vendas.

src/routes/
As rotas da aplicação são organizadas por entidade, com cada conjunto de endpoints definido em um arquivo específico.
Essa abordagem facilita a manutenção e a escalabilidade da aplicação:

- clientRoutes.js: Define as rotas para clientes (CRUD).
- sellerRoutes.js: Endpoints relacionados aos vendedores.
- productRoutes.js: Rotas para operações com produtos e estoque.
- saleRoutes.js: Gerencia as rotas de vendas e transações.

src/schemas/
Os schemas definem regras de validação de dados, garantindo que as informações recebidas estejam no formato esperado.
Cada domínio possui um schema dedicado:

- clientSchema.js: Validação de dados de clientes.
- sellerSchema.js: Define as regras de entrada para vendedores.
- productSchema.js: Valida estrutura e tipo dos produtos.
- saleSchema.js: Regras para validação de vendas.

src/services/
A camada de serviços contém as regras de negócio da aplicação. Cada serviço é responsável por encapsular a lógica específica de sua entidade.
Arquivos por domínio:

- clientService.js: Processa a lógica de negócio relacionada aos clientes.
- sellerService.js: Implementa as regras que envolvem os vendedores.
- productService.js: Gerencia as regras de estoque e categorias de produtos.
- saleService.js: Executa a lógica de validação e processamento das vendas.

src/utils/
Utilitários de uso geral que apoiam outras partes do sistema.

- generateRegistration.js: Gera identificadores únicos para registros.
- validateID.js: Valida o formato e a legitimidade de identificadores utilizados na aplicação.

Integração com a API de Relatórios

A api-report conecta-se diretamente ao mesmo banco de dados para realizar consultas agregadas, como.

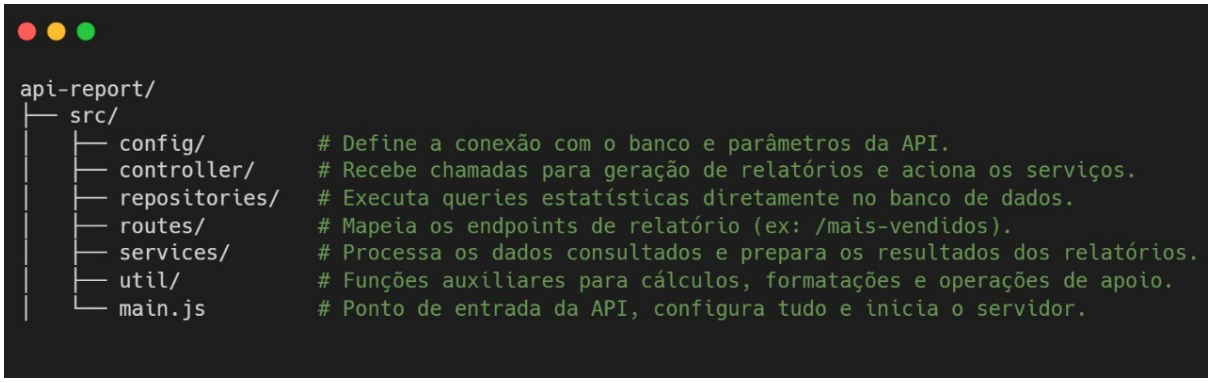
- Produtos mais vendidos.
- Produtos adquiridos por cliente.
- Consumo médio por cliente.

- Produtos com baixo estoque.

As consultas foram otimizadas via SQL, e o código seguiu o padrão de organização semelhante ao da api-main, prezando pela clareza e modularização.

Estrutura de Pastas da API-report

A estrutura de arquivos da api-report foi organizada da seguinte forma:



api-report

src/config/
Define os parâmetros de configuração da aplicação de relatórios, incluindo conexão com o banco de dados.

src/controllers/
Controladores responsáveis por receber requisições de relatórios e encaminhá-las para os serviços apropriados.

- reportController.js: Centraliza a lógica de entrada da API de relatórios.

src/repositories/ -> Repositórios responsáveis pela execução de queries específicas para geração de relatórios estatísticos.

- reportRepository.js: Implementa as consultas necessárias para análises e relatórios.

src/routes/
Define os endpoints específicos da API de relatórios, organizando os acessos e chamadas ao controller.

- reportRoutes.js: Mapeia rotas de relatórios para o reportController.

src/services/
Responsável por processar a lógica de geração dos relatórios com base nos dados recuperados.

- reportService.js: Aplica regras de análise e formatação de relatórios.

src/utils/
Funções utilitárias de apoio à API de relatórios.
- validateID.js: Reutilização da função de validação de identificadores.

main.js
Ponto de entrada da aplicação api-report, inicializando os serviços e registrando as rotas.

Conteinerização

O docker-compose.yml define três serviços

- api-main
- api-report
- mysql

Com isso, o sistema completo pode ser iniciado com um único comando, garantindo reprodutibilidade e escalabilidade para diferentes ambientes.

Banco de Dados

O sistema utiliza o MySQL como sistema gerenciador de banco de dados relacional (SGBDR), responsável por armazenar todas as informações persistentes da aplicação, como dados de clientes, vendedores, produtos e vendas. A estrutura do banco foi projetada para refletir as entidades principais da aplicação, promovendo integridade referencial e facilitando operações transacionais.

A base de dados, *salesdb*, é inicializada automaticamente por meio de um script SQL (init.sql) durante a execução do ambiente via Docker Compose. Esse script cria as tabelas necessárias e insere registros iniciais para viabilizar os testes e validações do sistema.

As duas APIs — api-main e api-report — compartilham o mesmo banco de dados, acessando-o de forma independente conforme suas responsabilidades. Enquanto a api-main realiza operações de **CRUD** (Create, Read, Update, Delete) sobre as tabelas principais, a api-report se concentra em consultas analíticas com uso de funções agregadas e cláusulas SQL otimizadas para relatórios estatísticos.

A comunicação entre as APIs e o MySQL é feita com o uso do driver **mysql2**, que oferece suporte a promessas e operações assíncronas no ambiente Node.js. Isso

garante um desempenho mais eficiente na manipulação de dados, especialmente em aplicações com múltiplas requisições simultâneas.

Além disso, o banco de dados é executado em um contêiner Docker, isolado dos demais serviços, e configurado para escutar na porta 3309. A conectividade entre os serviços ocorre dentro da rede virtual sales-network, gerenciada pelo Docker Compose, garantindo segurança e eficiência na comunicação.

A estrutura das tabelas foi organizada de forma a permitir escalabilidade e clareza na modelagem dos dados. Entre os principais relacionamentos definidos, destacam-se:

- **Clientes e Vendas**: relação um-para-muitos, onde cada cliente pode ter diversas vendas associadas.

- **Vendedores e Vendas**: cada venda é associada a um vendedor específico.

- **Produtos e Vendas**: relação muitos-para-muitos, resolvida por meio de uma tabela intermediária que armazena os itens vendidos, suas quantidades e preços unitários no momento da transação.

```sql
CREATE TABLE clients (
    id INT AUTO_INCREMENT PRIMARY KEY,
    full_name VARCHAR(80) NOT NULL,
    email VARCHAR(150) UNIQUE NOT NULL,
    phone VARCHAR(15) UNIQUE NOT NULL,
    address VARCHAR(80),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

INSERT INTO clients (full_name, email, phone, address) VALUES
    ('Clara Guimarães Rosa',
    'clara_G_rosa@gmail.com',
    '71987654321',
    'Caminho das Árvores, 985 - Salvador');
```

Esse modelo relacional fornece uma base sólida tanto para as funcionalidades operacionais da api-main quanto para os relatórios analíticos gerados pela api-report, promovendo consistência, desempenho e escalabilidade no gerenciamento de dados.

Endpoints (Exemplos)

Response API-MAIN
/api/clients/:id - Operação GET

Esta requisição recupera os dados de um cliente específico com base no seu ID. É utilizado para exibir informações detalhadas de um cliente já cadastrado no sistema.

```
{
    "id": 3,
    "full_name": "Mateus Bispo Lima",
    "email": "limabispo@yahoo.com",
    "phone": "31976543210",
    "address": "Travessa da Paz, 85 - Belo Horizonte",
    "created_at": "2025-06-08T17:03:39.000Z",
    "updated_at": "2025-06-08T17:03:39.000Z"
}
```

Descrição:
O campo **id** é gerado automaticamente pelo banco.
created_at e **updated_at** são timestamps que indicam quando o registro foi criado e modificado.

Response API-MAIN
/api/clients/ - Operação POST

Esta r equisição realiza o cadastro de um novo cliente no sistema. Os dados devem ser enviados no corpo da requisição no formato JSON.

```
{
    "id": 6,
    "full_name": "Milena Oliveira",
    "email": "milena.oliveira@gmail.com",
    "phone": "71988889999",
    "address": "Rua das Mangueiras, nº 127, Jardim Armação, Salvador - BA, CEP 41750-540"
}
```

Descrição:

Todos os campos são obrigatórios no cadastro.
O campo **id** é retornado após a criação, indicando o identificador do novo cliente no banco.

CONSIDERAÇÕES FINAIS

O desenvolvimento do projeto permitiu aplicar os conceitos estudados na disciplina de Sistemas Distribuídos de forma prática. A separação de responsabilidades em diferentes serviços, o uso de conteinerização e a estruturação modular demonstraram ser eficientes na construção de uma aplicação escalável e de fácil manutenção.

Entre os desafios enfrentados estão o versionamento da base de dados, o controle de dependências entre APIs e a necessidade de consultas SQL otimizadas para relatórios. Todos esses pontos foram superados com a adoção de boas práticas de desenvolvimento e pesquisa sobre padrões de projeto modernos.

Salvador/BA
2025

BIBLIOGRAFIA

TANENBAUM, Andrew S.; VAN STEEN, Maarten. *Sistemas Distribuídos: Princípios e Paradigmas*. Pearson, 2014.

NEWMAN, Sam. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015.

Node.js. Disponível em: https://nodejs.org

Express.js. Disponível em: https://expressjs.com

MySQL. Disponível em: https://www.mysql.com

Docker. Disponível em: https://www.docker.com

Joi Documentation. Disponível em: https://joi.dev

dotenv. Disponível em: https://www.npmjs.com/package/dotenv

Salvador/BA
2025