

**UNIFACS - UNIVERSIDADE SALVADOR**

**SISTEMAS DISTRIBUÍDOS**

**TRABALHO A3**

SALVADOR

2024

**UNIFACS - UNIVERSIDADE SALVADOR**

**ALUNOS**

JAILSON RODRIGUES DE NEIVA – 12722131344

EDSON COSTA MONTEIRO NETO – 1272212361

VITOR HUGO ALMEIDA PASSOS BRAGA – 12724145826

VINÍCIUS ARIEL RIBEIRO DOS SANTOS – 1272318365

GABRIEL SANTANA DE ASSUNÇÃO - 12723211354

**RELATÓRIO A3**

Trabalho apresentado para fins avaliativos da Unidade Curricular SISTEMAS DISTRIBUÍDOS correspondente ao ano letivo 2024.2

Orientador: Adailton Cerqueira

SALVADOR

2024

Relatório

**Requerimentos de Software**

Para a execução e desenvolvimento da aplicação, foram utilizados os seguintes softwares, linguagens e bibliotecas:

1. **Linguagem de Programação**:
   * **JavaScript** com execução no ambiente **Node.js**.
2. **Dependências**:
   * **Express**: Framework minimalista para criação de servidores web.
   * **Sequelize**: ORM (Object-Relational Mapper) para interação com bancos de dados relacionais neste caso o PostgreSQL.
   * **CORS**: Middleware para habilitar o compartilhamento de recursos entre origens diferentes.
   * Todas as dependências são instaladas automaticamente utilizando as instruções abaixo da seção de Requirementos de Software.
3. **Ambiente de Desenvolvimento**:
   * **VSCode**: Editor de código utilizado para o desenvolvimento do projeto, e extensões para gerenciamento do Docker e opcionalmente realização de métodos HTTP com o ThunderClient
4. **Ambiente de Contêineres**:
   * **Docker**: Utilizado para containerizar a aplicação. O arquivo Dockerfile e/ou docker-compose.yml contém as configurações necessárias para a execução.
5. **Banco de Dados e Ferramentas de Gerenciamento**:
   * **DBeaver**: Ferramenta utilizada para gerenciar o banco de dados.
   * Banco de dados: PostgreSQL.

**Configuração do Ambiente**

1. **Instalação do Node.js**:
   * Certifique-se de que o Node.js (e npm) está instalado no sistema.
2. **Configuração do Banco de Dados**:
   * Configure as credenciais e o endereço do banco de dados.

* Conexão DB:
* Database: db-postgres
* Port: 5432
* Host: localhost
* Usuário: usuario
* Senha: usuario

**Instruções para execução:**

1. Estando na pasta 'codigo' e com o Docker inicializado, abra o terminal no Vscode e execute o comando 'docker-compose up --build -d' (Todas as bibliotecas serão baixadas automaticamente).
2. Instale o PostgreSQL no DBeaver ou outra ferramenta de sua preferência para acompanhar o CRUD em SQL.
3. Use a extensão do Vscode -> 'Thunder Client', Postman ou Insomnia REST para usar os métodos HTTP.

**Introdução:**

O desenvolvimento do projeto utiliza uma API REST para a administração de um sistema de supermercado, no qual integra funcionalidades que permitem fazer o gerenciamento dos clientes, vendedores, produtos e vendas de forma centralizada, promovendo a automação de processos e escalabilidade, garantindo eficiência para no negócio. Além disso, a implementação de uma API de geração de relatórios permite a extração e análise de dados importantes, auxiliando na tomada de decisões estratégicas. A escolha do tema se deve ao fato da relevância da digitalização no mercado de varejo e pela necessidade de ferramentas tecnológicas para aprimorar a eficiência operacional e a tomada de decisão baseada em dados.

A abordagem utilizada no desenvolvimento do projeto envolve a aplicação de conceitos de arquitetura de software moderna, como a separação de responsabilidades em módulos bem definidos e a integração de um banco de dados relacional. Para o desenvolvimento do software, optou-se por tecnologias que são amplamente utilizadas no mercado, como o Node.js para a implementação da API, o PostgreSQL para gerenciar dados e o Docker para deixar a aplicação em contêiners, garantindo uma portabilidade e facilidade de implementação.

Os objetivos do projeto buscam projetar um sistema modular e escalável que permita futuras expansões sem grandes alterações estruturais, além de otimizar a gestão de dados no supermercado, permitindo o controle detalhado de estoque, acompanhamento de vendas e registro de informações de clientes e vendedores e a geração de relatórios que forneçam insights cruciais para o gerenciamento do negócio.

**Fundamentação Teórica:**

O projeto segue o modelo de sistemas distribuídos, no qual são um modelo computacional onde componentes de um sistema são distribuídos por várias máquinas interligadas, mas que trabalham de forma colaborativa para alcançar um objetivo comum. Essa abordagem permite maior escalabilidade, disponibilidade e tolerância a falhas. As seguintes características estão presentes no projeto:

1. Transparência:

* O acesso aos dados do banco de dados (PostgreSQL) é mediado pela API, garantindo que os detalhes da comunicação estejam ocultos ao cliente.

1. Escalabilidade:

* Usando Docker, é possível adicionar novas instâncias da API ou do banco de dados para atender a maiores volumes de requisições ou dados.
* Como o banco de dados e a API são serviços distintos, cada um pode ser ajustado de acordo com as necessidades específicas

1. Confiabilidade e tolerância a falhas:

* Caso algum serviço falhe, ele pode ser reiniciado automaticamente devido à política de restart: always configurada.
* A API utiliza o comando pg\_isready para garantir que o banco de dados esteja disponível antes de iniciar, minimizando problemas de conexão.

**API REST:**

Outro modelo utilizado no projeto é o da API REST, modelo este que é amplamente utilizado em sistemas distribuídos devido à sua simplicidade e escalabilidade. APIs REST adotam o paradigma cliente-servidor, onde o cliente faz solicitações HTTP para acessar ou manipular recursos gerenciados pelo servidor. A API utilizada no projeto permite gerenciar recursos como clientes, produtos, vendedores e vendas, que podem ser acessados por meio de URLs dedicadas, utilizando operações com os métodos HTTP padrão como GET, POST, PUT e DELETE, garantindo conformidade com o padrão REST.

**Banco de Dados Relacional (PostgreSQL):**

O banco de dados utilizado no projeto é o PostgreSQL, um sistema de gerenciamento de banco de dados relacional que oferece robustez, segurança e conformidade com padrões ACID (Atomicidade, Consistência, Isolamento e Durabilidade). Ele é ideal para sistemas distribuídos que requerem alta integridade de dados e suporte a grandes volumes de transações.

**Contêineres e Docker:**

A utilização do Docker no projeto exemplifica o papel dos contêineres em sistemas distribuídos. Docker permite empacotar aplicações e seus ambientes em contêineres isolados, garantindo portabilidade e consistência entre ambientes de desenvolvimento, teste e produção.

1. Dockerfile:

* Define o ambiente da API, incluindo dependências, variáveis e comandos de inicialização.

1. Docker Compose:

* Coordena múltiplos serviços, como a API e o banco de dados, criando uma infraestrutura local que simula um sistema distribuído.

**Projeto de implementação:**

Esta seção se dedica a demonstrar elementos que demonstraram papel fundamental para construção e funcionamento do projeto.

Diagrama UML: Utilizado para ter uma visão geral de como seria estruturado a arquitetura e os dados das entidades, Clientes, Vendedores, Produtos, Vendas, e assim poder ter uma melhor organização na estruturação do código e do banco de dados. (figura 1).

Diagrama

Descrição gerada automaticamente com confiança média

Figura 1: Diagrama UML[1](#UML) demostrando os dados e os relacionamentos das entidades, Clientes, Vendedores, Vendas, Produtos.

Algoritmos essenciais: Estruturas de código que garantem integridade (figura 2), sincronização (figura 3) e tolerância a falhas (figura 4), conforme ensinado em aula e permitindo que o projeto tenha os pilares da computação distribuída.

Texto

Descrição gerada automaticamente

Figura 2: Estrutura de código que captura erros de sintaxe JSON e impede que o sistema pare de funcionar devido a um erro de digitação, exibindo apenas uma mensagem de erro programada, garantindo a integridade[2](#Middleware).

Texto

Descrição gerada automaticamente

Figura 3: Para garantir que o ambiente de desenvolvimento fosse consistente, foi implementada uma rotina para verificar a conexão com o banco de dados e fazer a sincronização.

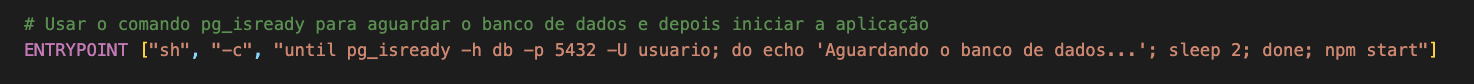


Figura 4: O uso de “pg\_isready” no Docker garante que a aplicação só inicie quando o banco estiver pronto.

**Considerações Finais:**

Neste projeto, a implementação de uma API REST para o sistema de supermercado foi uma oportunidade para aplicar conceitos fundamentais de desenvolvimento e modelagem de software, arquitetura de sistemas e integração com banco de dados, utilizando tecnologias como Node.js, PostgreSQL, Sequelize e Docker. A utilização de ferramentas de modelagem como a construção do diagrama UML foi crucial para uma modelagem clara e eficiente do sistema, facilitando a visualização das ideias e estrutura do projeto.

Um dos maiores desafios enfrentados durante o projeto foi garantir que todas as partes do sistema (API, banco de dados e containers Docker) funcionassem corretamente em conjunto, especialmente considerando a interdependência dos serviços. Além de erros de execução em solicitações HTTP que estavam sendo ocasionados pelo throw no arquivo “validações.js”, no qual ocasionava redundâncias.

Com relação aos resultados, a API de Supermercado foi implementada com sucesso, com rotas bem definidas para manipulação de clientes, vendedores, produtos e vendas. As operações de CRUD funcionam de maneira eficiente, atendendo aos requisitos do sistema.

Além disso, foi desenvolvida e implementada com sucesso a API de geração de relatórios, capaz de realizar operações **GET** para receber, gerar e atualizar relatórios, bem como a funcionalidade de deletá-los. Essa API complementa o sistema ao oferecer uma maneira prática e eficiente de gerir informações analíticas, atendendo às necessidades de acompanhamento e tomada de decisão do negócio.

**PROCEDIMENTOS USO DA API**

1. Estando na pasta 'codigo' execute 'docker-compose up --build -d'

2. Opcional -> Instale o postgres no dbeaver ou outra ferramenta de sua preferência para acompanhar o crud no banco de dados.

3. Use a extensão do vscode -> 'thunder client', postman ou o insomnia para usar os métodos http

4. Obs. Estes procedimentos encontram-se também no arquivo README.md

**# MÉTODOS HTTP**

**1. CLIENTES - 'CRUD'..........................................................**

    \* 'GET'

      - (tudo)....... http://localhost:3000/clientes

      - (por nome)... http://localhost:3000/clientes/nome/jailson

      - (por cpf) ... http://localhost:3000/clientes/cpf/111.222.333-44

    \* 'POST'

      - http://localhost:3000/clientes

        {

          "nome":"jailson",

          "cpf":"111.222.333-44",

          "endereco":"rua A bairro X"

        }

    \* 'PUT/PATCH'

      - (por cpf).... http://localhost:3000/clientes/cpf/111.222.333-44

      - (por Id)..... http://localhost:3000/clientes/id/1

    \* 'DELETE'

      - (por cpf).... http://localhost:3000/clientes/cpf/111.222.333-44

      - (por Id)..... http://localhost:3000/clientes/id/1

      - (todos)...... http://localhost:3000/clientes/todos

**2. VENDEDORES - 'CRUD'.......................................................**

    \* 'GET'

      - (tudo)....... http://localhost:3000/vendedores

      - (por nome)... http://localhost:3000/vendedores/nome/jair

      - (por cpf) ... http://localhost:3000/vendedores/cpf/111.222.333-44

    \* 'POST'

      - http://localhost:3000/vendedores

        {

          "nome":"jailson",

          "cpf":"111.222.333-44",

          "endereco":"Endereço A"

        }

    \* 'PUT/PATCH'

      - (por cpf).... http://localhost:3000/vendedores/cpf/111.222.333-44

      - (por Id)..... http://localhost:3000/vendedores/id/1

        {

          "nome":"jailson",

          "cpf":"111.222.333-44",

          "endereco":"Endereço A"

        }

    \* 'DELETE'

      - (por cpf).... http://localhost:3000/vendedores/cpf/111.222.333-44

      - (por Id)..... http://localhost:3000/vendedores/id/1

      - (todos)...... http://localhost:3000/vendedores/todos

**3. PRODUTOS - 'CRUD'.......................................................**

    \* GET

      - (tudo)....... http://localhost:3000/produtos

      - (por nome)... http://localhost:3000/produtos/nome/farinha

      - (por id) .... http://localhost:3000/produtos/id/1

    \* POST

      - http://localhost:3000/produtos

        {

          "nome":"farinha",

          "preco":3.50,

          "estoque":50

        }

    \* 'PUT/PATCH' Atualiza ou "nome", ou "preco", ou "estoqe

      - (por nome).... http://localhost:3000/produtos/nome/farinha

      - (por Id)...... http://localhost:3000/produtos/id/1

    \* 'DELETE'

      - (por nome).... http://localhost:3000/produtos/nome/farinha

      - (por Id)...... http://localhost:3000/produtos/id/1

      - (todos)....... http://localhost:3000/produtos/todos

**4. PEDIDOS - 'CREATE/DELETE' (COMPRAR E CANCELAR).......................**

   \* Receber pedido:

     POST http://localhost:3000/pedidos/comprar

        {

          "idProduto": 1,

          "quantidade": 222

        }

   \* Cancelar pedido:

     POST http://localhost:3000/pedidos/cancelar

        {

          "idProduto": 1,

          "quantidade": 5

        }

**5. VENDAS - 'CREATE'....................................................**

   POST http://localhost:3000/vendas

       {

          "clienteNome": "Jailson",

          "vendedorNome": "Alberto",

          "produtoNome": "sal",

          "quantidade": 3

      }

**6. ESTOQUE - 'READ' - POR PRODUTO....................................**

   GET http://localhost:3000/produtos/feijão/estoque

**7. CRUD RELATÓRIOS.......................................................**

    \* GET

      - (baixo-estoque)... http://localhost:3030/relatorios/baixo-estoque/

      - (mais-vendidos) .... http://localhost:3030/relatorios/mais-vendidos/

      - (consumo-medio) .... http://localhost:3030/relatorios/consumo-medio/

      - (consumo-medio-id) .... http://localhost:3030/relatorios/consumo-medio/:id

      - (produto-cliente) .... http://localhost:3030/relatorios/produto-cliente/:id

      - (tudo)....... http://localhost:3030/relatorios/

    \* DELETE

      - ..... http://localhost:3030/relatorios/:id

**Bibliografia**

<https://www.lucidchart.com/pages/pt/o-que-e-uml>

<https://learn.microsoft.com/pt-br/aspnet/core/fundamentals/middleware/?view=aspnetcore-9.0>

1. <https://www.postgresql.org/docs/current/app-pg-isready.html>