UNIVERSIDADE SALVADOR
SISTEMAS DISTRIBUÍDOS E MOBILE

PROJETO A3: Sistemas distribuídos e mobile

TEMA

Criação de uma aplicação para simular a captação de dados de venda de uma rede de lojas de roupas em alfaiataria.

Integrantes:

Odinelson Leandro Ferreira dos Santos / 12722211571
Yan Caique Santos Muniz / 12722210600
Diego da Conceição Santos / 12722133121
Vinicius Ribeiro de Santana / 1272221567
João Pedro Ferreira Guimarães / 1272221138

Orientador: Prof. Adailton de Jesus Cerqueira Junior

SALVADOR - BA
2024

SUMÁRIO

1 INTRODUÇÃO

O modelo deste formulário seguirá a seguinte estrutura, para um maior entendimento. No **tópico 1** foi apresentada a introdução, com os objetivos propostos a serem realizados. No **tópico 2** temos a fundamentação teórica, onde vamos citar os temas abordados no trabalho de maneira abrangente e sólida. No **tópico 3** será o projeto de implementação, explicação dos algoritmos e outros elementos que ajudaram na compreensão para entender como nosso código se comporta. **No tópico 4** teremos as considerações finais, Informaremos a opinião da equipe em relação ao trabalho, objetivos alcançados e desafios enfrentados durante todo o desenvolvimento. E por fim, no **tópico 5** que é a bibliografia / referência, onde colocaremos todo material utilizado para o nosso conhecimento durante a parte teórica.

Uma API é uma interface de programação e aplicativos usada no desenvolvimento web e de software.Com ela é possível fazer com que diferentes aplicativos interajam um com o outro e troquem de informações e dados de maneira segura e correta **(5.1)**.

Portanto, a proposta desse trabalho é criar uma aplicação para uma loja onde o foco é a venda de roupas em alfaiataria. Utilizamos da linguagem JavaScript para fazer todo o back-end em todas as partes do trabalho. Ao final é esperado que o usuário consiga gerenciar três tipos de serviços, sendo eles clientes, produtos e funcionários. Construímos dois containers docker para executar a aplicação.

Acompanhando os tópicos de acordo com os objetivos pré estabelecidos.

• Existe um serviço de gerenciar cliente, vendedor, estoque, vendas e geração de relatórios estatísticos.

• O sistema deve iniciar com no mínimo 10 (dez) produtos cadastrados, 5 (cinco) clientes e 2 (dois) vendedores.

• A aplicação é desenvolvida em Javascript.

• A equipe deve escolher implementar a solução usando API e outras tecnologias que julgar necessárias.

• O banco de dados tem que ser relacional.

• Toda a aplicação deve ser executada em containers Docker.

• O serviço de geração de relatório deve ser executado em uma aplicação distinta do restante do sistema.

2 FUNDAMENTAÇÃO TEÓRICA.

A aplicação foi feita com a linguagem de programação em Javascript, com a biblioteca mysql 2.0

2.1 Aplicativos utilizados.

Visual Studio Code - Realização do código em Javascript e utilização do terminal .

VS Code é, basicamente, um editor que auxilia os programadores na criação de um código de software, sobretudo nas importantes fases de codificação e testes (**5.2**). O motivo pelo qual foi utilizado esse editor, é que as funções que o mesmo disponibiliza ajuda bastante na hora de desenvolver.

Docker - Para criação dos containers (Docker compose).

O Docker é uma plataforma open source que facilita a criação e administração de ambientes isolados. Ele possibilita o empacotamento de uma aplicação ou ambiente dentro de um container, se tornando portátil para qualquer outro host que contenha o Docker instalado (**5.3**). A utilização do docker foi obrigatória para a realização do trabalho.

Postman - Realizar teste do API.

O Postman é uma ferramenta que dá suporte à documentação das requisições feitas pela API e faz a conexão através da HTTP. Ele possui ambiente para a documentação, execução de testes de APIs e requisições em geral (**5.4**). O uso do postman foi preferencial, porque foi de conhecimento de todos durante a aula ministrada pelo professor.

MySQL Workbench - Para trabalhar com o mysql.

MySQL Workbench é uma ferramenta visual de design, desenvolvimento e administração de bancos de dados MySQL. Com uma interface intuitiva, permite que desenvolvedores e administradores de banco de dados criem, modelam e gerenciem suas bases de dados de forma eficiente (**5.5**). Usado inicialmente para ver se os dados estavam sendo atribuídos corretamente ao banco de dados MySQL.

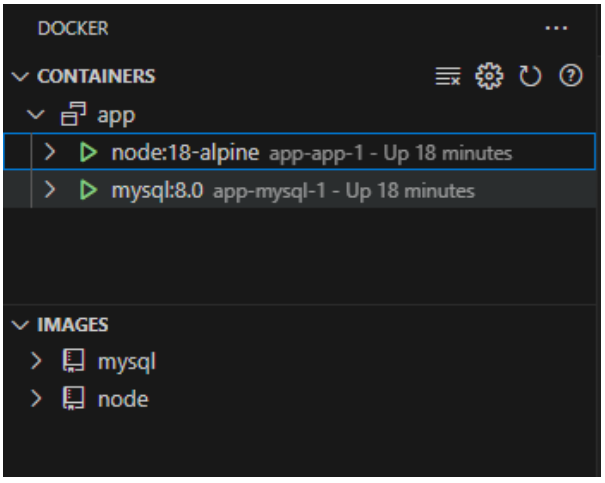
Banco de Dados - MySQL.

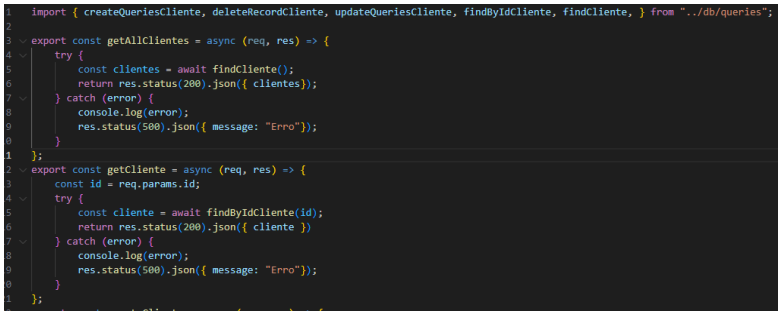
O MySQL é um sistema de gerenciamento de banco de dados relacional de código aberto (RDBMS) suportado pela Oracle e baseado em linguagem de consulta estruturada (SQL) (**5.6**) . O uso do MySQL foi para manipulação de dados usando o sistema de containers do docker.

2.2 Contextualização do código.

```
JS index.js > ...
1   import express from "express";
2   import "./db/index.js";
3   import { connectToDatabase } from "./db/index.js";
4   import appRouterCliente from "./routes/routercliente.js";
5   import appRouterVendedor from "./routes/routerVendedor.js";
6   import appRouterProduto from "./routes/routerProduto.js";
7   const app = express();
8
9
10  app.use(express.json());
11
12  app.use("/clientes", appRouterCliente);
13
14  app.use("/vendedores", appRouterVendedor);
15
16  app.use("/produtos", appRouterProduto);
17
18  const PORT = process.env.PORT || 3000;
19
20  connectToDatabase().then(() => {
21      app.listen(PORT, () => console.log("Servidor rodando na porta:", PORT));
22  })
23  .catch(err => {
24      console.log("Erro ao conectar o banco de dados. Error =", err);
25      process.exit(0);
26  });
```

Da linha 1 a 6 da figura acima, temos as importações referente a conexão com o banco de dados, e importações de rotas das tabelas. Declaração da variável “app.use” passando como parâmetro o “express.json”. Rodando o servidor na porta 3000.





```
import { createQueriesCliente, deleteRecordCliente, updateQueriesCliente, findByIdCliente, findCliente, } from "../db/queries";

export const getAllClientes = async (req, res) => {
    try {
        const clientes = await findCliente();
        return res.status(200).json({ clientes});
    } catch (error) {
        console.log(error);
        res.status(500).json({ message: "Erro"});
    }
};
export const getCliente = async (req, res) => {
    const id = req.params.id;
    try {
        const cliente = await findByIdCliente(id);
        return res.status(200).json({ cliente })
    } catch (error) {
        console.log(error);
        res.status(500).json({ message: "Erro"});
    }
};
```

Criamos dois containers docker(Figura 1) diferentes como demonstra na figura acima para manipulação dos dados, onde um é da aplicação e o outro do banco de dados. Na pasta “controllers” criamos três classes com os seguintes nomes: ControllerCliente, ControllerProduto, e ControllerVendedor. Todas essas classes são responsáveis por processar solicitações e manipular dados como cadastrar, listar, deletar e editar uma informação salva no banco de dados. E importamos do arquivo “queries.js” para realizar os métodos programados(Figura 2), isso sucessivamente para todas as classes criadas.

db > JS index.js > ...

```js
import { createPool } from "mysql2/promise";
import { config } from "dotenv";
config();

const pool = createPool({
    port: process.env.MYSQL_PORT,
    password: process.env.MYSQL_PASSWORD,
    host: process.env.MYSQL_HOST,
    database: process.env.MYSQL_DATABASE,
    user: process.env.MYSQL_USER
});

const connectToDatabase = async() => {
    try {
        await pool.getConnection();
        console.log("Banco de dados conectado!");
    } catch (error) {
        console.log("Erro ao conectar banco de dados!");
        console.log(error);
        throw error;


    }
};

export { connectToDatabase, pool };
```

Figura 1

db > JS queries.js > findCliente

```js
import { pool } from "./index.js";

export const findCliente = async () => {
    const QUERY = "SELECT * FROM clientes";
    try {
        const cliente = await pool.getConnection();
        const result = await cliente.query(QUERY);
        return result;
    } catch (error) {
        console.log("Erro", error);
        throw error;
    }
};

export const findByIdCliente = async (id) => {
    const QUERY = "SELECT * FROM clientes WHERE id = ?";
    try {
        const cliente = await pool.getConnection();
        const result = await cliente.query(QUERY, [id]);
        return result[0];
    } catch (error) {
        console.log("Erro", error);
        throw error;
    }
};
```

Figura 2

Dentro da pasta “db” criamos dois arquivos index.js (figura 1), onde fazemos a importação do mysql 2 e configuramos o acesso atribuindo nome, senha e conexão. Na figura 2, temos o arquivo “queries.js” que criamos os métodos de gerenciamento das variáveis cliente, produto, e vendedores. Além de realizar a importação do método “pool”, que serve para criar a conexão com o banco de dados utilizando dotenv.

{} package.json > ...

```json
{
  "name": "app",
  "version": "1.0.0",
  "main": "index.js",
  "type": "module",
  ▷ Debug
  "scripts": {
    "dev": "nodemon index.js",
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "dotenv": "^16.4.7",
    "express": "^4.21.2",
    "mysql2": "^3.11.5"
  },
  "devDependencies": {
    "nodemon": "^3.1.7"
  }
}
```

No arquivo “Package.json” atribuímos propriedades básicas para o funcionamento do docker, como definir nome, versão, script, e dependência com o banco de dados.

app > docker-compose.yml

```yaml
version: '3'

services:
  app:
    image: node:18-alpine
    ports:
      - 127.0.0.1:3000:3000
    working_dir: /app
    volumes:
      - ./:/app
    command: ["node", "index.js"]
    environment:
      MYSQL_HOST: "mysql"
      MYSQL_USER: "app_user"
      MYSQL_PASSWORD: "app_password"
      MYSQL_DB: "lojaderoupa"
    depends_on:
      - mysql
    networks:
      - dockernet

```

Figura 1

app > init.sql

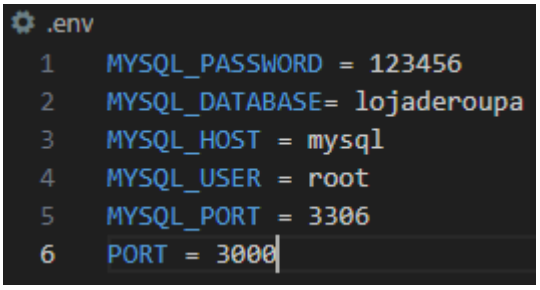
```sql
CREATE TABLE clientes (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nome VARCHAR(255),
  email VARCHAR(255),
  telefone VARCHAR(20)
);

CREATE TABLE vendedores (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nome VARCHAR(255),
  matricula VARCHAR(20)
);

CREATE TABLE produtos (
  id INT PRIMARY KEY AUTO_INCREMENT,
  tipo VARCHAR(255),
  cor VARCHAR(255),
  preco DECIMAL(10, 2)
);
```

Figura 2

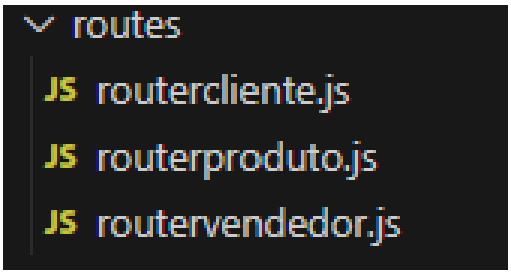
```sql
INSERT INTO clientes (nome, email, telefone) VALUES
('João', 'joao@example.com', '11912345678'),
('Maria', 'maria@example.com', '11998765432'),
('Pedro', 'pedro@example.com', '11911111111'),
('Ana', 'ana@example.com', '11922222222'),
('Luiz', 'luiz@example.com', '11933333333');

INSERT INTO vendedores (nome, matricula) VALUES
('Antonio', '1194'),
('Neide', '1195');

INSERT INTO produtos (tipo, cor, preco) VALUES
('Camisa', 'Branca', 49.99),
('Calça', 'Jeans', 79.99),
('Vestido', 'Preto', 99.99),
('Terno', 'Preto', 999.99),
('Bermuda', 'Marrom', 49.99),
('Camiseta', 'Cinza', 199.99),
('Casaco', 'Vermelho', 99.99),
('Cinto', 'Cinto preto', 89.99),
('Meia', 'Meia branca', 49.99),
('Calça', 'Preta', 139.99);
```

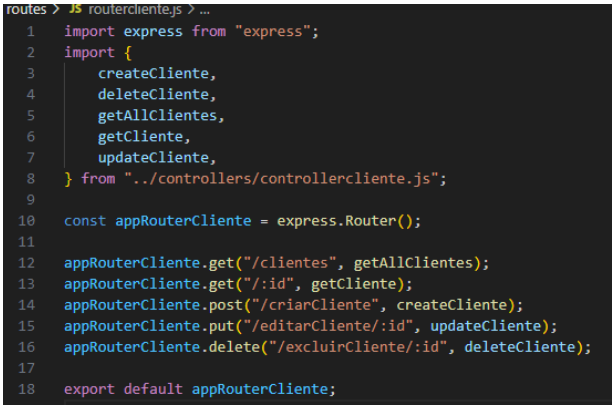
Figura 3

Usamos o Docker Compose para manipular mais de um container, utilizando a versão 3(Figura 1 na linha 1), para o banco de dados estamos usando o mysql 2.0. (Figura 1 na linha 5). Criamos um script de inicialização com o init.sql, onde atribuímos instruções que são executadas imediatamente após a conexão com um banco de dados ser estabelecida, la definimos os dados salvos, que devem ser iniciados já com a aplicação citado na requisição do trabalho(Figura 1 e 2). Na parte de volume da estrutura docker compose colocamos o diretório para esse arquivo init.sql (Figura 1 na linha 10).



No arquivo “.Env” usado para armazenar variáveis de ambiente que configuram e personalizam a aplicação sem precisar modificar diretamente o código-fonte. atribuímos a senha, o número da porta, nome de usuário, e data base.





Já na pasta “routes” criamos os arquivos com módulo do Express que define as rotas relacionadas à funcionalidade de todas as classes criadas de uma aplicação. Junto com controladores que implementam as lógica para as rotas.

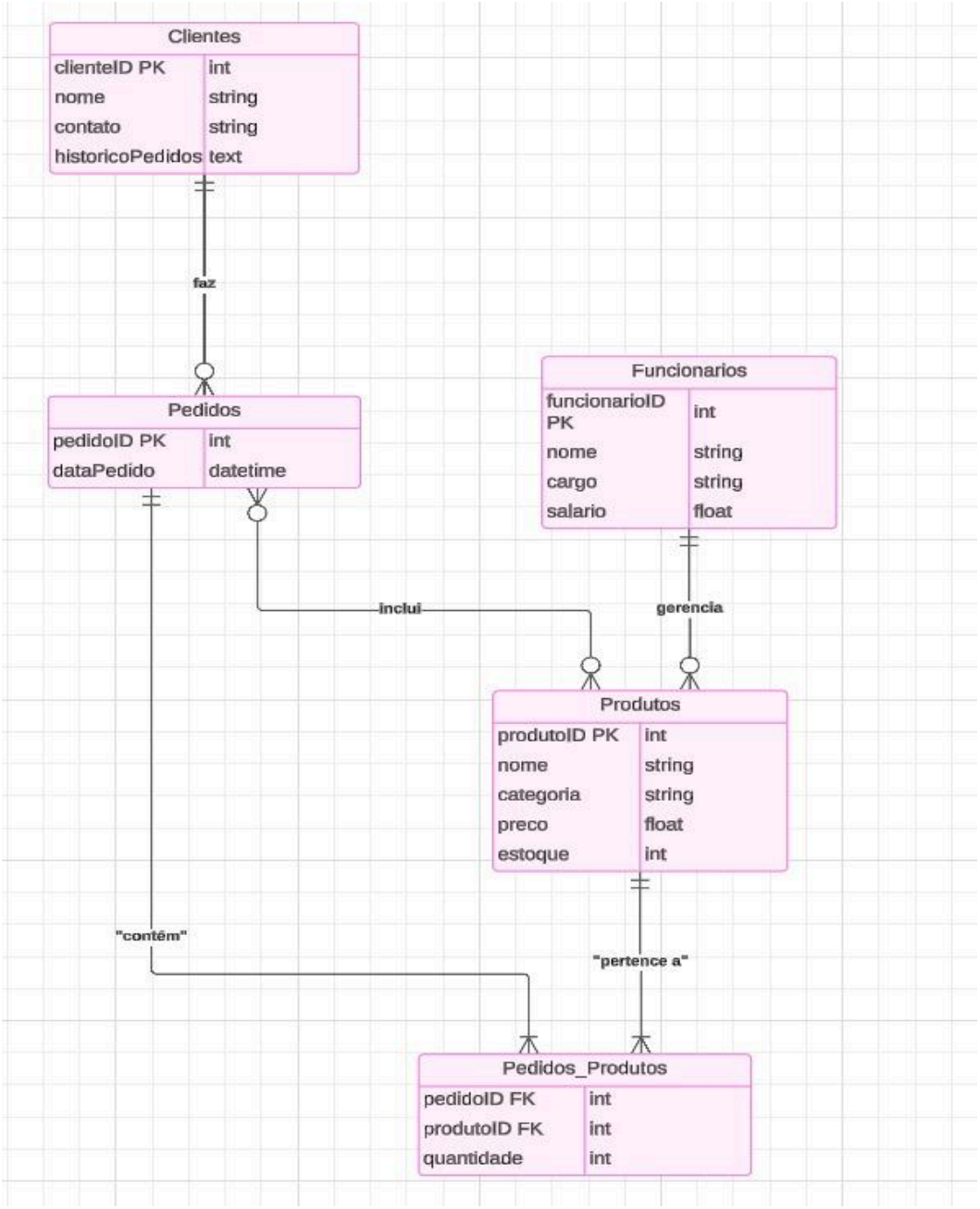
3 PROJETO DE IMPLEMENTAÇÃO

Neste projeto, o foco foi desenvolver uma aplicação para uma loja especializada em roupas de alfaiataria, utilizando JavaScript para implementar todo o back-end. A solução foi projetada para permitir o gerenciamento de três tipos de serviços essenciais para a operação da loja: **clientes**, **produtos** e **funcionários**.

O uso de **JavaScript** no back-end se mostrou adequado devido à sua flexibilidade e à vasta gama de bibliotecas e frameworks disponíveis, o que facilitou a implementação de funcionalidades complexas. Além disso, **dois containers Docker** foram construídos para isolar e executar a aplicação de forma eficiente e segura, garantindo a escalabilidade e a facilidade de gerenciamento do ambiente de execução.

Embora a codificação em si tenha sido central, optamos por focar na demonstração de algoritmos e na modelagem da aplicação. Diagrama de classes em **UML** foi utilizado para representar a estrutura do sistema, destacando as interações entre clientes, produtos e

funcionários. Além disso, um **Diagrama Entidade-Relacionamento (ER)** foi construído para modelar o banco de dados relacional, demonstrando como as tabelas de clientes, produtos e funcionários se relacionam entre si, garantindo a consistência e a integridade dos dados.



Essa abordagem modular, combinada com o uso de containers, assegura que a aplicação seja facilmente escalável e possa ser mantida e expandida conforme a demanda da loja.

4 CONSIDERAÇÕES FINAIS

Tivemos alguns desafios enfrentados durante toda a construção do trabalho, mas destacamos alguns:

- Como fazer o docker funcionar com o servidor, inicialmente entendendemos que o banco de dados deveria ser feito pelo mysql e depois implementado no docker, mas ao tirar dúvida em sala, descobrimos que o banco de dados era pra ser feita já em um container do docker, com isso tivemos que mudar a estrutura do trabalho.

- Como fazer o CRUD responder corretamente. Ao fazer a chamada da função para realizar as ações como listar, deletar e outras, percebemos que o resultado sempre parava no tratamento do erro “catch” e não realizava a busca nem listava os dados já cadastrados no banco de dados do servidor.

- Tivemos dificuldade para criarmos os dois containers docker, pois quando conseguimos subir um o outro dava erro, quando ajeitava o que estava com problema o que estava funcionando dava problema.

- Estávamos com problemas na pasta index, quaisquer modificações que realizamos ela dava erro.
- Enfrentamos problemas em relação a nossa base de dados e a visualização da mesma por conta de conflitos das versões do mysql.

Embora não tenhamos conseguido concluir o trabalho, é importante reconhecer o esforço e a dedicação que colocamos em cada etapa. Foi um processo desafiador e exaustivo, mas também uma oportunidade incrível de aprendizado e crescimento, pois conseguimos construir os containers docker e construir a estrutura docker compose. O empenho demonstrado por todos foi exemplar, e mesmo sem o resultado final esperado, o caminho trilhado nos preparou para enfrentar desafios futuros com ainda mais determinação e experiência. Cada passo dado foi um avanço em direção ao nosso aprimoramento.

5 REFERÊNCIAS

Aqui estão os links que utilizamos para geração do relatório.

5.1 O que é API e pra que serve?

https://ebaconline.com.br/blog/o-que-e-uma-api-seo#:~:text=Uma%20API%20

5.2 Remessa Online, 2021 - Visual Studio Code.

https://www.remessaonline.com.br/blog/visual-studio-code-confira-as-principais-funcoes-da-ferramenta/

5.3 Marylene-guedes, 2018 - Docker

https://www.treinaweb.com.br/blog/no-final-das-contas-o-que-e-o-docker-e-como-ele-funciona

5.4 enotas Blog, 2024 - Postman

https://enotas.com.br/blog/postman/\

5.5 LBODEV, 2024 - MySQL Workbench

https://lbodev.com.br/glossario/o-que-e-mysql-workbench/

5.6 Aléxis Cerqueira Góis, 2021 - MySQL

https://www.tecmundo.com.br/software/223924-mysql-usar-o-sistema.htm