Universidade Salvador

Arley do Nascimento Vinagre - 12722132338

Eduardo Batista Araujo - 1272327420

Marcelo Silva do Carmo Filho - 1272323017

Tauan Santos Santana – 12722216126

Tiago Silva Coelho Maciel - 1272326567

Sistemas Distribuídos e Mobile

Relatório A3

Salvador– BA

2024

Arley do Nascimento Vinagre - 12722132338

Eduardo Batista Araujo - 1272327420

Marcelo Silva do Carmo Filho - 1272323017

Tauan Santos Santana – 12722216126

Tiago Silva Coelho Maciel - 1272326567

Sistemas Distribuídos e Mobile

Relatório A3

Trabalho apresentado à disciplina de Sistemas Distribuídos e Mobile destinado à criação de uma aplicação que simule a captação de dados de venda de uma rede de lojas, para a instituição Universidade Salvador no curso de Ciência da Computação – 3° Semestre.

Orientadores: Adailton de Jesus e Marivaldo Perreira.

Salvador– BA

2024

Introdução

A crescente complexidade dos processos empresariais demanda tecnologias que tornem as operações mais eficientes, especialmente em vendas, gestão de clientes e controle de estoque. A automação dessas áreas melhora a experiência do cliente, aumenta a produtividade e reduz erros humanos, tornando-se essencial para manter a competitividade no mercado. Soluções integradas que otimizam esses processos oferecem agilidade, precisão e melhores subsídios para a tomada de decisões estratégicas.

Este trabalho apresenta o desenvolvimento de uma aplicação de gestão modular baseada na arquitetura de microsserviços, integrando vendas, clientes e estoque. A utilização de Node.js com o framework Express permite uma API eficiente e de fácil manutenção, enquanto o banco de dados MySQL garante confiabilidade no armazenamento e manipulação de dados. A arquitetura modular proporciona escalabilidade, facilitando adaptações e melhorias no sistema conforme as necessidades do negócio evoluem.

Para garantir portabilidade e consistência, a aplicação será containerizada com Docker, permitindo a implantação uniforme em diferentes ambientes. Entre as principais funcionalidades estão o controle de estoque, cadastro de clientes e vendedores, registro de vendas e geração de relatórios estatísticos. Esses relatórios, focados em produtos mais vendidos e consumo médio, fornecem insights estratégicos que impulsionam a eficiência operacional e a melhoria contínua no desempenho empresarial.

Material e Métodos

Para a construção da aplicação, diversas tecnologias e conceitos foram utilizados. O principal objetivo foi implementar um sistema modular que pudesse ser escalado facilmente, utilizando a arquitetura de microsserviços, que é uma abordagem cada vez mais adotada no desenvolvimento de sistemas distribuídos.

Tecnologias Utilizadas

- **Node.js e Express**: Para a criação da API backend, utilizando o JavaScript para garantir agilidade e alto desempenho.
- **MySQL**: Sistema de gerenciamento de banco de dados relacional, escolhido pela sua robustez e capacidade de lidar com grandes volumes de dados.
- **Docker**: Utilizado para a containerização da aplicação, garantindo a portabilidade e consistência do ambiente de execução.
- **HeidiSQL**: Ferramenta de administração do banco de dados MySQL, facilitando a gestão visual e execução de consultas SQL.
- **Postman**: Usado para testar as APIs e garantir a integridade e o bom funcionamento das requisições.

Arquitetura do Sistema

A aplicação foi dividida em módulos funcionais, cada um responsável por uma parte específica do sistema: clientes, vendedores, produtos, pedidos e relatórios. A arquitetura de microsserviços foi adotada para separar as funcionalidades, permitindo escalabilidade e manutenção independentes. O banco de dados MySQL foi estruturado com cinco tabelas principais: clientes, vendedores, produtos, pedidos e itens_pedido, inter-relacionadas por chaves estrangeiras.

Docker

A aplicação e o banco de dados foram containerizados utilizando Docker e orquestrados com Docker Compose. A configuração envolveu a criação de containers isolados para os serviços da aplicação e o banco de dados, garantindo que ambos pudessem ser executados de maneira independente e replicável em diferentes ambientes.

Resultados

A aplicação foi implementada com sucesso, permitindo o cadastro e gerenciamento de clientes, vendedores, produtos e pedidos. Além disso, o sistema gerou relatórios estatísticos sobre o desempenho de vendas, produtos mais vendidos e o estoque de produtos. A containerização com Docker foi eficaz para garantir a portabilidade e facilitar o processo de implantação.

Funcionalidades Implementadas

- **Criação do Banco de Dados**

O MySQL foi escolhido para este projeto devido à sua confiabilidade e capacidade de lidar com grandes volumes de dados. No contexto deste projeto, o banco de dados foi configurado para interagir com o **HeidiSQL**, que é uma ferramenta de gerenciamento de banco de dados MySQL. O HeidiSQL facilita a administração visual do banco de dados, permitindo a execução de consultas SQL e o gerenciamento das tabelas de forma simplificada.

Para baixar e instalar o HeidiSQL é só acessar: https://www.heidisql.com/download.php

A conexão com o banco de dados foi realizada com a versão 5.7 do MySQL, pois é compatível com o HeidiSQL e oferece um bom equilíbrio entre desempenho e suporte a recursos. A configuração da conexão com o banco de dados foi realizada da seguinte forma:

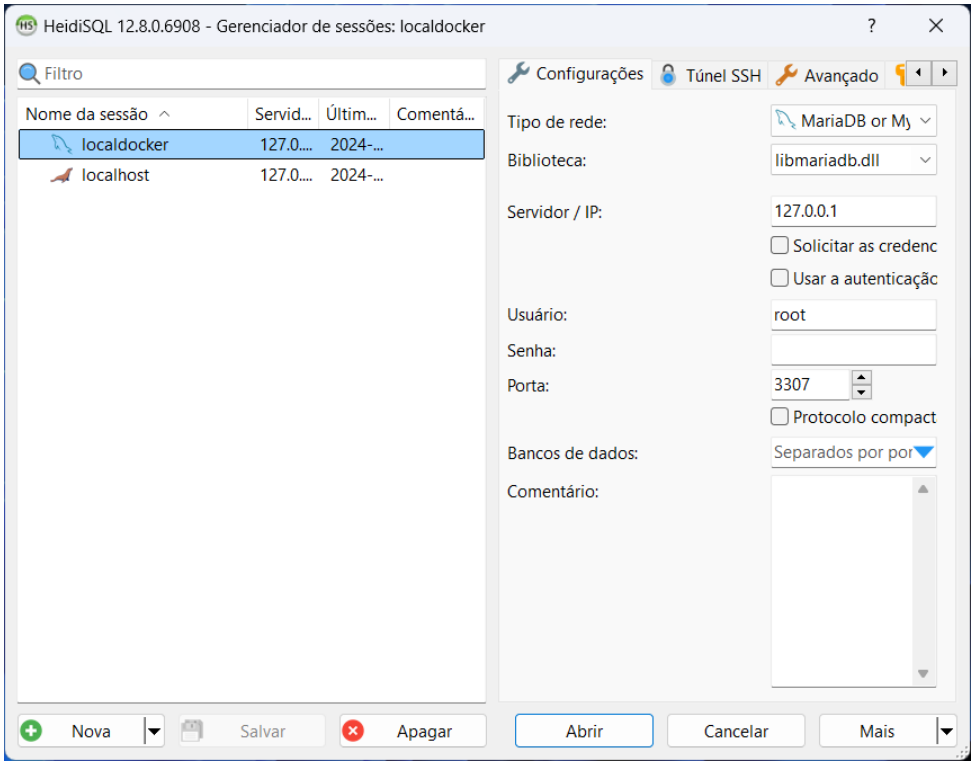
```
const mysql = require('mysql');
const db = mysql.createConnection({
    host: process.env.MYSQL_HOST || 'localhost',
    user: process.env.MYSQL_USERNAME || 'root',
    password: process.env.MYSQL_PASSWORD || '',
    database: process.env.MYSQL_DATABASE || 'api'
});
```

Este código realiza a conexão ao banco de dados MySQL, utilizando variáveis de ambiente para armazenar as credenciais de acesso, como o host, usuário, senha e o nome do banco de dados. A versão 5.7 do MySQL foi escolhida por sua

compatibilidade com o HeidiSQL, permitindo uma fácil visualização e gerenciamento do banco, bem como garantindo a consistência e a integridade dos dados.

Ao abrir o **HeidiSQL**, é necessário criar um novo banco de dados para integrar as informações diretamente ao Docker. Para isso, clique na opção **"Nova"** para criar uma nova instância de conexão. Renomeie-a para **"localdocker"** e configure a porta como **3307**, evitando conflitos com a porta padrão (3306) utilizada pelo banco de dados local, conforme é mostrado na Figura 1.

Figura1 – Tela inicial do HeidiSQL



Fonte: HeidiSQL

- **Estrutura do Banco de Dados**

O banco de dados MySQL é composto por cinco tabelas principais: clientes, vendedores, produtos, pedidos e itens_pedido. As tabelas são relacionadas entre si através de chaves estrangeiras. Por exemplo, a tabela pedidos tem referências para as tabelas clientes e vendedores, enquanto a tabela itens_pedido faz referência às tabelas pedidos e produtos. Isso garante a integridade dos dados e facilita a consulta e manipulação das informações.

Exemplo de código de criação de tabela:

```
CREATE TABLE produtos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    descricao TEXT,
    preco DECIMAL(10,2) NOT NULL,
    quantidade_estoque INT NOT NULL,
    data_criacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- **API de Clientes e Vendedores**

O módulo de clientes permite realizar operações CRUD (Create, Read, Update, Delete) sobre os dados dos clientes. O código a seguir exemplifica a criação de um novo cliente via API RESTful utilizando o framework Express:

```
app.post('/clientes', (req, res) => {
  const { nome, email, telefone } = req.body;
  const query = 'INSERT INTO clientes (nome, email, telefone) VALUES (?, ?, ?)';
  db.query(query, [nome, email, telefone], (err, result) => {
    if (err) {
      return res.status(500).json({ error: 'Erro ao adicionar cliente' });
    }
    res.status(201).json({ message: 'Cliente adicionado com sucesso' });
  });
});
```

- **API de Produtos e Estoque**

O gerenciamento de produtos é uma das funcionalidades essenciais da aplicação. O módulo de produtos permite cadastrar novos itens, atualizar informações de estoque e excluir produtos. A tabela de produtos contém informações como nome, preço e quantidade em estoque, sendo que o estoque é atualizado sempre que um pedido é realizado.

Exemplo de rota Express para API de Produtos:

```
app.post('/produtos', (req, res) => {
```

```
  const { nome, descricao, preco, quantidade_estoque } = req.body;
 const query = 'INSERT INTO produtos (nome, descricao, preco, quantidade_estoque) VALUES (?, ?, ?, ?)';
  db.query(query, [nome, descricao, preco, quantidade_estoque], (err, result) => {
   if (err) {
    return res.status(500).json({ error: 'Erro ao adicionar produto' });
   }
   res.status(201).json({ message: 'Produto adicionado com sucesso' });
  });
 });
```

- **Geração de Relatórios Estatísticos**

O sistema possui um módulo separado para geração de relatórios, que são essenciais para a análise de desempenho. Alguns dos relatórios implementados são:

- **Produtos mais vendidos**: Exibe os produtos com maior volume de vendas.
- **Relatório de consumo médio dos clientes**: Permite entender o comportamento de compra dos clientes.
- **Relatório de produtos com baixo estoque**: Ajuda a monitorar o estoque e identificar produtos com baixa disponibilidade.

Exemplo de código para relatório de produtos mais vendidos:

```
app.get('/relatorios/produtos-mais-vendidos', (req, res) => {
  const query =
` SELECT p.nome, SUM(ip.quantidade) AS total_vendas
   FROM produtos p
   JOIN itens_pedido ip ON p.id = ip.produto_id
   GROUP BY p.id
   ORDER BY total_vendas DESC
   LIMIT 10; `;
  db.query(query, (err, results) => {
   if (err) {
    return res.status(500).json({ error: 'Erro ao gerar relatório' });
   }
   res.json(results);
```

```
  });
});
```

- **Containerização com Docker**

A aplicação foi containerizada utilizando Docker, uma plataforma que permite criar, implantar e executar aplicações em containers. Essa abordagem garante que tanto a aplicação quanto o banco de dados possam ser executados de maneira isolada, eficiente e replicável em diferentes ambientes. Para simplificar a gestão de múltiplos containers e configurar a comunicação entre eles, utilizou-se o Docker Compose, que facilita a orquestração de serviços.

Para baixar e instalar o Docker é só acessar: https://www.docker.com/get-started/

Configuração do Docker Compose

O arquivo docker-compose.yml define três serviços principais:

mysqldb: Um container baseado na imagem mysql:5.7, configurado para executar o banco de dados. A versão 5.7 foi escolhida por sua compatibilidade com a aplicação e ferramentas como o HeidiSQL.

app: O container da API principal, construída com Node.js, responsável pelo gerenciamento de clientes, produtos e pedidos.

sales-reports: Um módulo adicional que processa e gera relatórios estatísticos de vendas e estoque.

Configuração do Dockerfile

Os arquivos Dockerfile definem as instruções para construir os containers das aplicações. A escolha da imagem node:18-alpine garante uma base leve e eficiente para os serviços em Node.js.

Configuração de Variáveis de Ambiente

As variáveis de ambiente utilizadas no arquivo .env permitem ajustar facilmente configurações sem modificar os arquivos principais do projeto. Isso simplifica a configuração e a integração entre os serviços no ambiente de execução.

Testes e Validação

As APIs foram rigorosamente testadas utilizando o Postman, uma ferramenta que facilitou a validação dos endpoints e garantiu o correto funcionamento das operações. Esses testes confirmaram a comunicação eficaz entre a aplicação e o banco de dados, bem como a integridade das operações de CRUD (Create, Read, Update, Delete).

Exemplos de Testes Realizados

Cadastro de Produtos (POST)

Endpoint: http://localhost:3000/estoque/produtos

Corpo da requisição enviado pelo Postman:

```
{
  "nome": "Notebook Dell",
  "preco": 3500.99,
  "estoque": 15
}
```

Resposta esperada:

```
{
    "id": 1,
    "mensagem": "Produto cadastrado com sucesso."
}
```

Esse teste garantiu que os dados fornecidos eram armazenados corretamente no banco de dados configurado no serviço MySQL.

Listagem de Clientes (GET)

Endpoint: http://localhost:3000/clientes/clientes

Ao enviar uma requisição sem parâmetros, foi retornada uma lista de todos os clientes cadastrados. Para validar a integridade da comunicação, o Postman verificou que o servidor respondia com um status 200 OK e uma estrutura como esta:

```
[
   {
```

```
    "id": 1,
    "nome": "Eduardo",
    "email": "eduardo@gmail.com",
    "telefone": "71987848518",
    "endereco": "Rua da Paciência, 42"
},
{
    "id": 2,
    "nome": "Tiago",
    "email": "tiago@gmail.com",
    "telefone": "71984403622",
    "endereco": "Avenida Sete de Setembro, 211"
},
{
    "id": 3,
    "nome": "Tauan",
    "email": "tauan@gmail.com",
    "telefone": "71981941258",
    "endereco": "Rua Carlos Gomes, 58"
},
{
    "id": 4,
    "nome": "Arley",
    "email": "arley@gmail.com",
    "telefone": "71997136389",
    "endereco": "Rua das Pedras, 103"
},
{
    "id": 5,
    "nome": "Marcelo",
    "email": "marcelo@gmail.com",
    "telefone": "71999951812",
    "endereco": "Rua de Boa Viagem, 17"
}
```

]

Atualização de Estoque (PUT)

Endpoint: http://localhost:3000/estoque/produtos/{id}

Corpo da requisição para atualizar a quantidade em estoque:


```
{
  "nome": "Notebook Dell",
  "preco": 3500.99,
  "estoque": 50
}
```


Resposta esperada:


```
{
    "mensagem": "Estoque atualizado com sucesso."
}
```


O Postman confirmou que a tabela de produtos no banco de dados foi ajustada corretamente.

Relatorios (GET)

A aplicação inclui uma API dedicada à geração de relatórios para análise estratégica. Os endpoints disponíveis oferecem informações detalhadas, como:

- **Produtos Mais Vendidos:** Identifica os itens com maior volume de vendas.
  Endpoint: http://localhost:4000/relatorio/produtos-mais-vendidos
- **Produtos por Cliente:** Relaciona os produtos adquiridos por cada cliente.
  Endpoint: http://localhost:4000/relatorio/produtos-por-cliente
- **Consumo Médio:** Calcula o ticket médio por cliente, auxiliando na análise de comportamento de consumo.
  Endpoint: http://localhost:4000/relatorio/consumo-medio
- **Produtos com Baixo Estoque:** Aponta itens com estoques críticos, facilitando a reposição.
  Endpoint: http://localhost:4000/relatorio/baixo-estoque

Esses relatórios fornecem insights valiosos para a gestão de vendas, estoque e fidelização de clientes.

Exclusão de Pedidos (DELETE)

Endpoint: http://localhost:3000/pedidos/pedidos/{id}

Durante o teste, ao enviar uma requisição para excluir um pedido inexistente, o sistema respondeu corretamente com um status 404 Not Found. Para pedidos válidos, o status retornado foi 204 No Content, garantindo que a exclusão era realizada sem inconsistências.

Discussão

A implementação de uma arquitetura baseada em microsserviços trouxe benefícios significativos para o projeto, como a escalabilidade e a facilidade de manutenção do sistema. A utilização de Docker garantiu a portabilidade da aplicação, permitindo que fosse facilmente implantada em diferentes ambientes de desenvolvimento e produção. Além disso, o uso de ferramentas como o Postman e o HeidiSQL contribuiu para a validação da aplicação e a administração do banco de dados.

Porém, alguns desafios foram encontrados ao longo do desenvolvimento, principalmente no que diz respeito à integração entre os módulos da aplicação. A consistência dos dados também foi uma preocupação constante, sendo solucionada com a aplicação de boas práticas de desenvolvimento e o uso de transações no banco de dados para garantir a integridade dos registros.

Conclusões

O desenvolvimento da aplicação de gestão de vendas, clientes e estoque foi um sucesso, atendendo aos objetivos iniciais de automatizar processos empresariais e gerar relatórios de desempenho. A escolha das tecnologias, como Node.js, MySQL e Docker, provou-se eficiente para a construção de um sistema escalável, robusto e portável.

Como futuras melhorias, sugerem-se a implementação de autenticação e autorização para maior segurança, bem como a adição de uma interface gráfica (frontend) para tornar o sistema mais acessível aos usuários finais. O projeto representa uma solução viável para empresas que buscam melhorar o gerenciamento de vendas e estoque, e pode ser facilmente adaptado para diferentes tipos de negócios.

Referência

DOCKER. Using Docker Compose. Disponível em: <https://github.com/docker/getting-started/blob/master/docs/tutorial/using-docker-compose/index.md>.

DOCKER. Manuais. Disponível em: <https://docs.docker.com/manuals/>.

PERREIRA, Marivaldo. HeidiSQL. Aula ministrada no curso de Ciência da Computação, Universidade Salvador (UNIFACS), Salvador/BA. Disponível em: <https://d1s43sp1pfj89b.cloudfront.net/2024/10/03/2e5e0de3-7790-49f0-98ff-8a395599f4b8.mp4?AWSAccessKeyId=AKIARCAHYOUG644UP36K&Expires=1733366245&Signature=ypaEFBIlqTBN%2FwZDTKiyMsGKsFI%3D>. Aula do dia 03 de outubro 2024.

ORACLE CORPORATION. MySQL Stored Programs. Disponível em: <https://www.mysql.com/products/enterprise/storedprograms.html>.

BEZKODER. Exemplo Dockerize Node.js Express e MySQL – Docker Compose**.** Disponível em: <https://www.bezkoder.com/docker-compose-nodejs-mysql/>.

BEZKODER. Build Node.js Rest APIs with Express & MySQL. Disponível em: < https://www.bezkoder.com/node-js-rest-api-express-mysql/>.

MADSONAR. Projeto-Site-Portfólio-RESTFul-CRUD-JavaScript-NodeJs-Express-MySQL. Disponível em: <https://github.com/madsonar/Projeto-Site-Portfolio-RESTFul-CRUD-JavaScript-NodeJs-Express-MySQL>.