



Relatório do Projeto: Simulação de Captação de Dados de Venda de uma Rede de Lojas

UC: Sistemas Distribuídos e Mobile

Orientador: Prof. Adailton de Jesus Cerqueira Junior

Integrantes:

RA	NOME COMPLETO
12722211832	Amanda Kelly Brito Cerqueira
1272225024	Carolina Paiva Carvalho Brinço
1272227351	Cleisson Claudio Brasil Santana dos Santos
12722131007	Luís André Gomes de Oliveira Silva
12722116801	Paulo Henrique Pereira Araujo Piedade

SUMÁRIO

1.INTRODUÇÃO.....	3
2. FUNDAMENTAÇÃO TEÓRICA	4
3. PROJETO DE IMPLEMENTAÇÃO	4
4. INSTRUÇÕES PARA INSTALAÇÃO E EXECUÇÃO DA APLICAÇÃO	7
PASSO 1: PREPARAÇÃO DO AMBIENTE.....	7
5. DESCRIÇÃO DA ARQUITETURA DA APLICAÇÃO: API, RELATÓRIO E FRONT-END ESTÁTICO	13
7. CONSIDERAÇÕES FINAIS	20
8. BIBLIOGRAFIA	21

1.Introdução

Este relatório descreve o desenvolvimento de uma API CRUD utilizando Node.js, Express, MySQL e Docker, conforme apresentado no repositório no GitHub. O objetivo principal do projeto é implementar uma aplicação que permita operações básicas de CRUD (criação, leitura, atualização e exclusão) sobre dados armazenados em um banco de dados relacional, seguindo boas práticas de engenharia de software e utilizando tecnologias modernas para garantir eficiência e escalabilidade.

A escolha do tema foi motivada pela crescente demanda por aplicações web escaláveis e modulares, que possam atender às necessidades de empresas e instituições em ambientes digitais cada vez mais competitivos (1). O uso de tecnologias como Docker reflete a importância de práticas que facilitem a implantação e manutenção de sistemas, reduzindo a dependência de configurações específicas de ambiente. Além disso, o projeto proporciona uma aplicação prática das habilidades de desenvolvimento back-end e gerenciamento de bancos de dados adquiridas ao longo da disciplina.

Este projeto demonstra, de forma clara, a integração de diferentes componentes em uma solução coesa, abordando a criação de APIs RESTful com Express, a modelagem e manipulação de dados com MySQL e a orquestração de contêineres com Docker e Docker Compose (2). Ao longo do desenvolvimento, foi possível explorar os desafios da integração entre essas tecnologias e suas vantagens, como escalabilidade, modularidade e facilidade de implantação.

O objetivo final é oferecer uma aplicação funcional e intuitiva que simule um cenário de captação de dados de vendas de uma rede de lojas. Este sistema, ao permitir a gestão centralizada de dados de vendas, produtos e clientes, também serve como base para futuras implementações, como relatórios analíticos e autenticação de usuários. Além disso, os aprendizados adquiridos com este projeto poderão ser aplicados em contextos reais, contribuindo para a formação de soluções tecnológicas mais robustas e alinhadas com as demandas do mercado.

2. Fundamentação Teórica

O projeto utiliza o Node.js como ambiente de execução para JavaScript no lado do servidor, devido a necessidade de obter a eficiência em operações assíncronas, considerando sua escalabilidade, versatilidade e grande possibilidade de customização (2). A biblioteca Express, foi utilizada por ser um framework para construção de APIs RESTful, que facilita a criação de rotas e middleware, sem obscurecer os recursos do Node.js, sendo flexível e rápido (3). O MySQL foi escolhido como sistema de gerenciamento de banco de dados relacional, devido à sua compatibilidade com aplicações web, sendo o banco de dados de código aberto mais conhecido no mundo, além de apresentar alta escalabilidade, desempenho e confiabilidade (4). Docker foi empregado para criar contêineres, permitindo maior portabilidade e simplificação no deploy na aplicação, minimizando a divergência entre ambientes e não se preocupando com a necessidade de pré-requisitos (5).

3. Projeto de implementação

A aplicação foi desenvolvida para simular a captação de dados de vendas de uma rede de lojas, utilizando **Docker** para containerização, **Express.js** como framework para as APIs, **MySQL 8.0** como banco de dados relacional e um **front-end estático** em HTML e CSS para facilitar a interação do usuário.

Tecnologias e Ferramentas Utilizadas:

1. Linguagens de Programação:

JavaScript (Node.js): A aplicação foi construída utilizando JavaScript, rodando em Node.js. A escolha de JavaScript foi baseada na familiaridade da equipe com a linguagem e sua adequação para o desenvolvimento de APIs e integração com banco de dados.

2. Frameworks:

Express.js: Framework minimalista para Node.js, usado para criar as APIs RESTful. Express simplifica o roteamento e facilita a integração com outras bibliotecas, como a do MySQL.

mysql2: Biblioteca utilizada para conectar e interagir com o banco de dados MySQL de forma eficiente. O mysql2 oferece uma interface rápida e suporta promessas, o que facilita o uso assíncrono.

3. Banco de Dados:

MySQL 8.0: Banco de dados relacional utilizado para armazenar os dados de vendas. Escolhemos o MySQL por sua popularidade, confiabilidade e boa integração com o Node.js.

4. Docker & Docker Compose:

Docker: Utilizado para criar containers que encapsulam os diferentes componentes da aplicação (banco de dados, API e relatórios). Isso assegura que a aplicação será executada da mesma forma em qualquer ambiente.

Docker Compose: Usado para orquestrar a execução de múltiplos containers, simplificando o gerenciamento de serviços como o banco de dados, a API e os relatórios. Com o Compose, é possível iniciar todos os serviços necessários com um único comando.

5. Front-End Estático:

HTML & CSS: O front-end foi desenvolvido em HTML e CSS para criar uma interface de usuário simples, com inputs para facilitar a interação com o sistema. A decisão de manter o front-end como uma página estática foi tomada devido a dificuldades na integração com as APIs, o que simplificou a implementação e a execução.

6. Requisitos Adicionais:

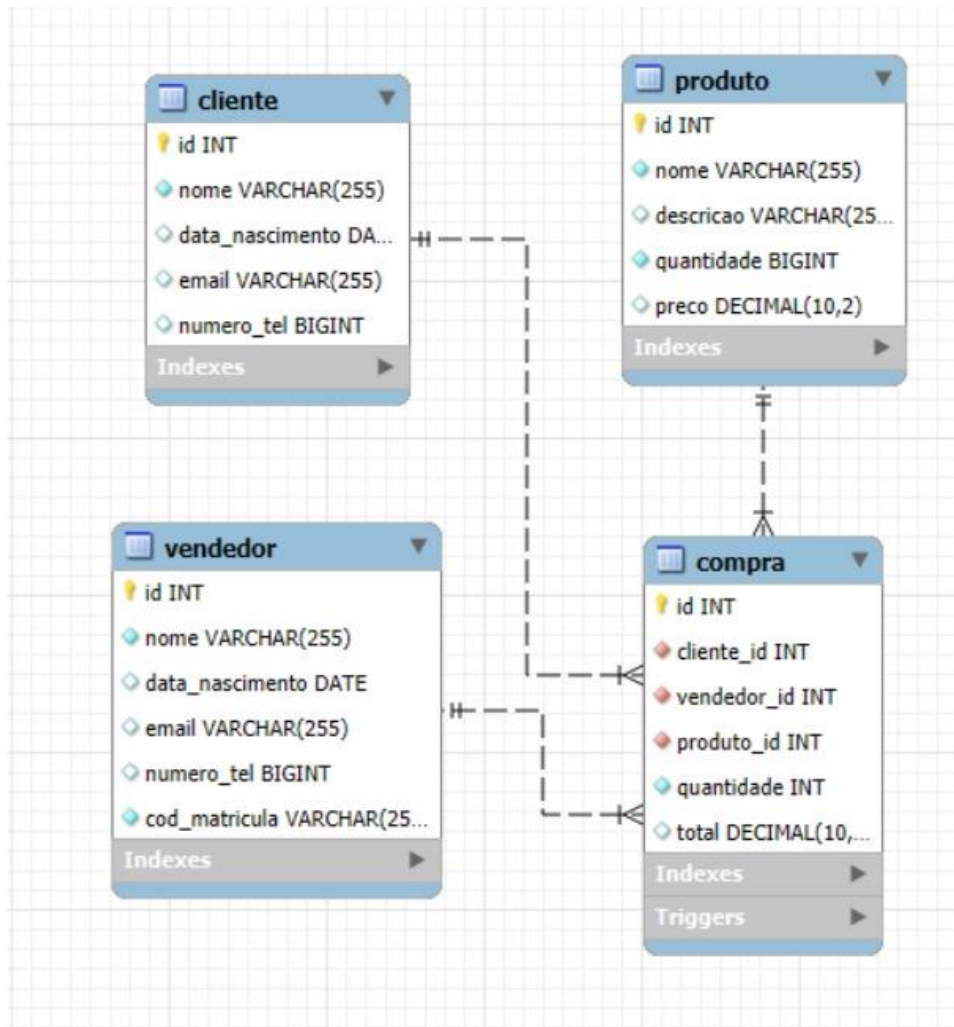
Sistema Operacional: A aplicação foi desenvolvida para ser executada em ambientes Linux e Windows (com Docker instalado).

Docker e Docker Compose Instalados: Ambos devem estar instalados no ambiente de desenvolvimento ou produção para o correto funcionamento da aplicação.

Navegador Web: A interface foi criada pensando na sua compatibilidade com o Google Chrome Versão 131.0.6778.87 (Versão oficial) 64 bits

7. Modelagem de dados:

Diagramas foram criados para modelar o sistema e facilitar o entendimento da arquitetura. Um diagrama Entidade-Relacionamento (ER) detalha a estrutura do banco, modelando o banco de dados relacional, além de determinar os requisitos a serem necessários em um projeto como este (7).



4. Instruções para Instalação e Execução da Aplicação

Passo 1: Preparação do Ambiente

1. Instalar Docker Desktop, Docker Compose e Git:

- **Docker Desktop:** Acesse o site oficial do Docker e siga as instruções para instalar o Docker Desktop.
- **Docker Compose:** O Docker Compose já vem incluído nas versões mais recentes do Docker Desktop, portanto, não é necessário instalá-lo separadamente.
- **Git:** Caso ainda não tenha o Git instalado, acesse o [site oficial do Git](https://git-scm.com/) e siga as instruções para instalação.

2. Clonar o Repositório do Projeto:

- Abra o PowerShell, estando dentro de um local que desejar e execute o seguinte comando para clonar o repositório do projeto:

```
git clone --branch EntregaA3 https://github.com/luis-a-silva/Entrega-A3
```

3. Navegar até o Diretório do Projeto:

- Após o repositório ser clonado, navegue até o diretório do código-fonte com o comando:

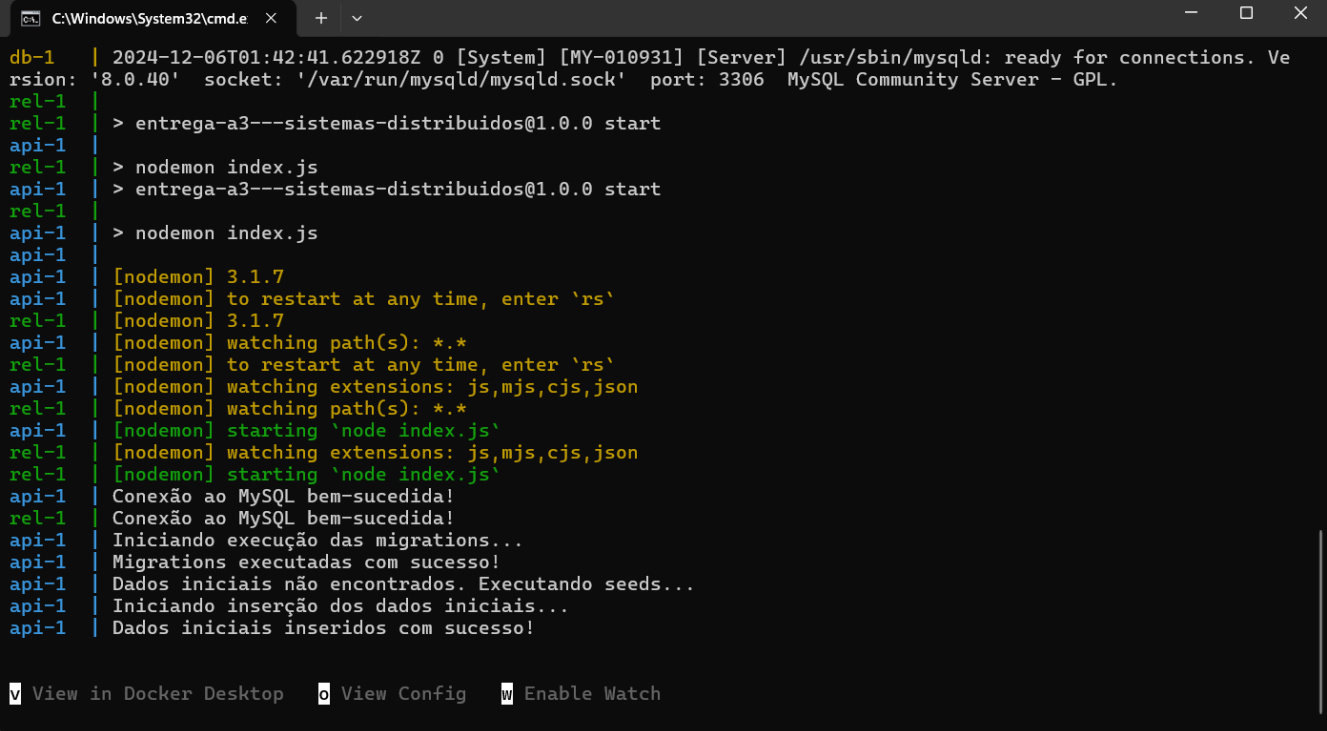
```
cd Entrega-A3\codigofonte  
cd codigofonte
```

4. Subir os Containers com Docker Compose:

- Dentro do diretório do código-fonte, execute o seguinte comando para subir os containers. O arquivo [docker-compose.yml](#) já contém as configurações necessárias para os serviços da aplicação, incluindo o banco de dados, API e relatórios.:

```
docker compose up --build
```

Demonstração de uma execução bem-sucedida do comando **docker compose up --build**. O PowerShell processará o comando e, quando a execução for finalizada, você poderá visualizar o container "codigofonte" no Docker Desktop, na aba **Containers**, conforme ilustrado nas imagens abaixo:



```
db-1 | 2024-12-06T01:42:41.622918Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Ve
rsion: '8.0.40' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL.
rel-1 |
rel-1 | > entrega-a3---sistemas-distribuidos@1.0.0 start
api-1 |
rel-1 | > nodemon index.js
api-1 | > entrega-a3---sistemas-distribuidos@1.0.0 start
rel-1 |
api-1 | > nodemon index.js
api-1 |
api-1 | [nodemon] 3.1.7
api-1 | [nodemon] to restart at any time, enter `rs`
rel-1 | [nodemon] 3.1.7
api-1 | [nodemon] watching path(s): *.*
rel-1 | [nodemon] to restart at any time, enter `rs`
api-1 | [nodemon] watching extensions: js,mjs,cjs,json
rel-1 | [nodemon] watching path(s): *.*
api-1 | [nodemon] starting `node index.js`
rel-1 | [nodemon] watching extensions: js,mjs,cjs,json
rel-1 | [nodemon] starting `node index.js`
api-1 | Conexão ao MySQL bem-sucedida!
rel-1 | Conexão ao MySQL bem-sucedida!
api-1 | Iniciando execução das migrations...
api-1 | Migrations executadas com sucesso!
api-1 | Dados iniciais não encontrados. Executando seeds...
api-1 | Iniciando inserção dos dados iniciais...
api-1 | Dados iniciais inseridos com sucesso!

V View in Docker Desktop  O View Config  W Enable Watch
```

Imagem 1: Tela do PowerShell após todos os containers serem executados com sucesso.

O sinal verde indicará que a aplicação está em execução como na imagem abaixo:

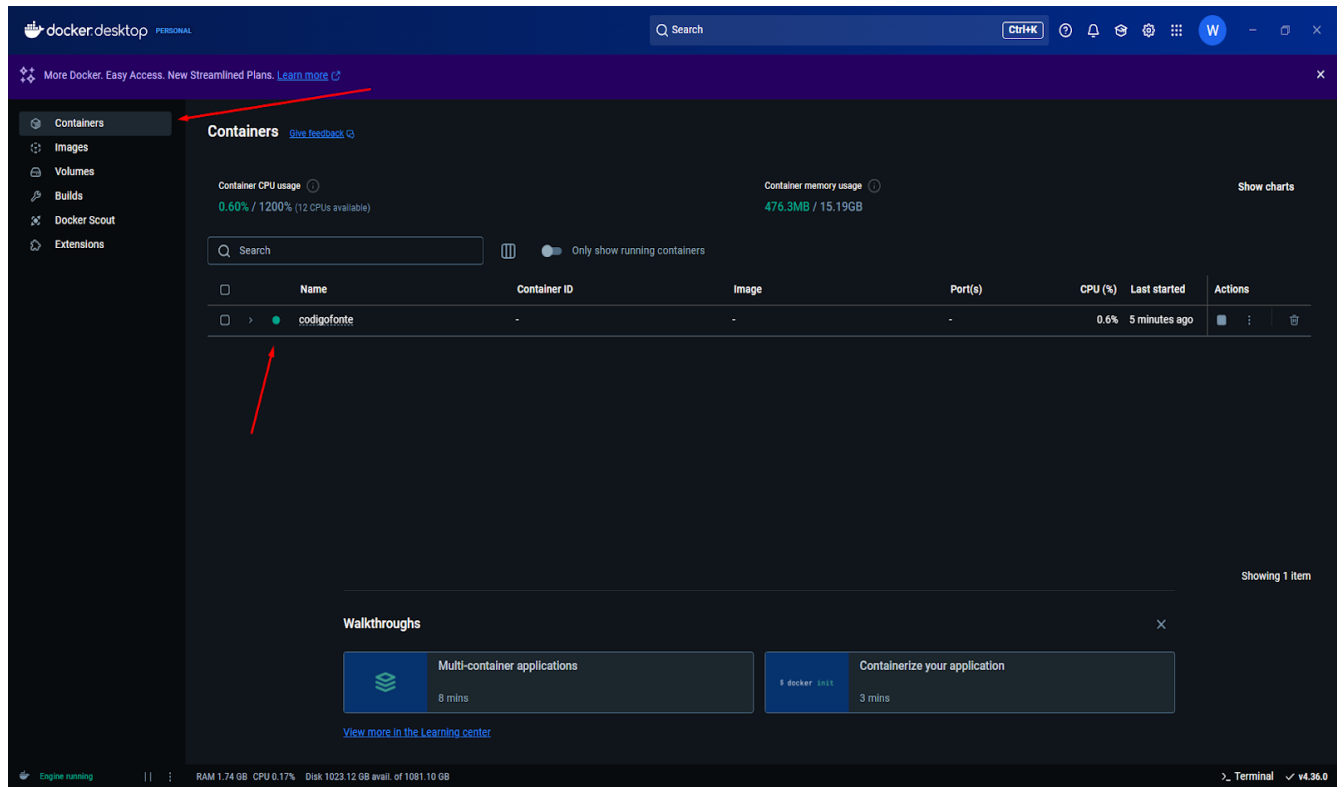


Imagem 2: Tela do Docker para desktop após todos os containers serem executados com sucesso.

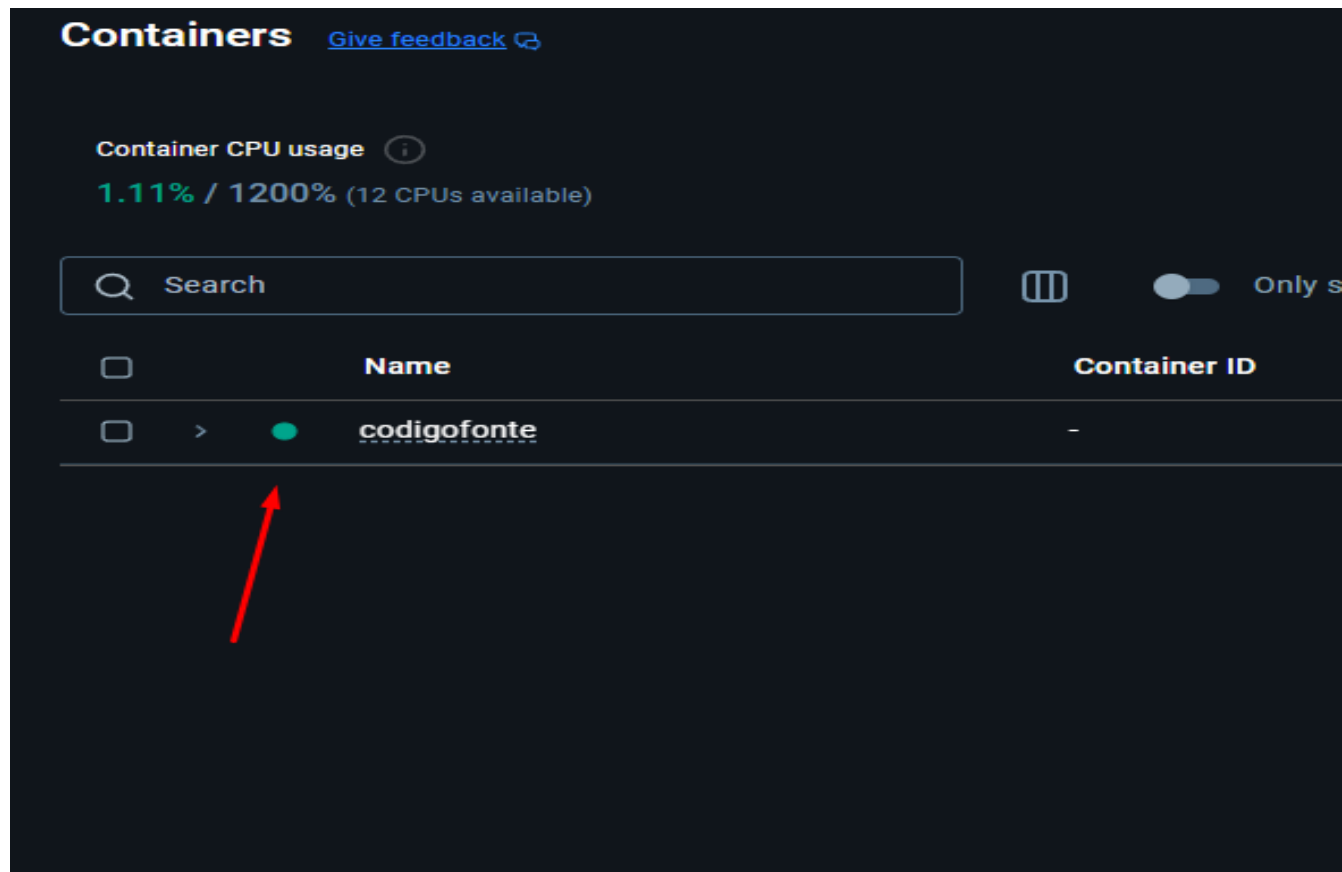
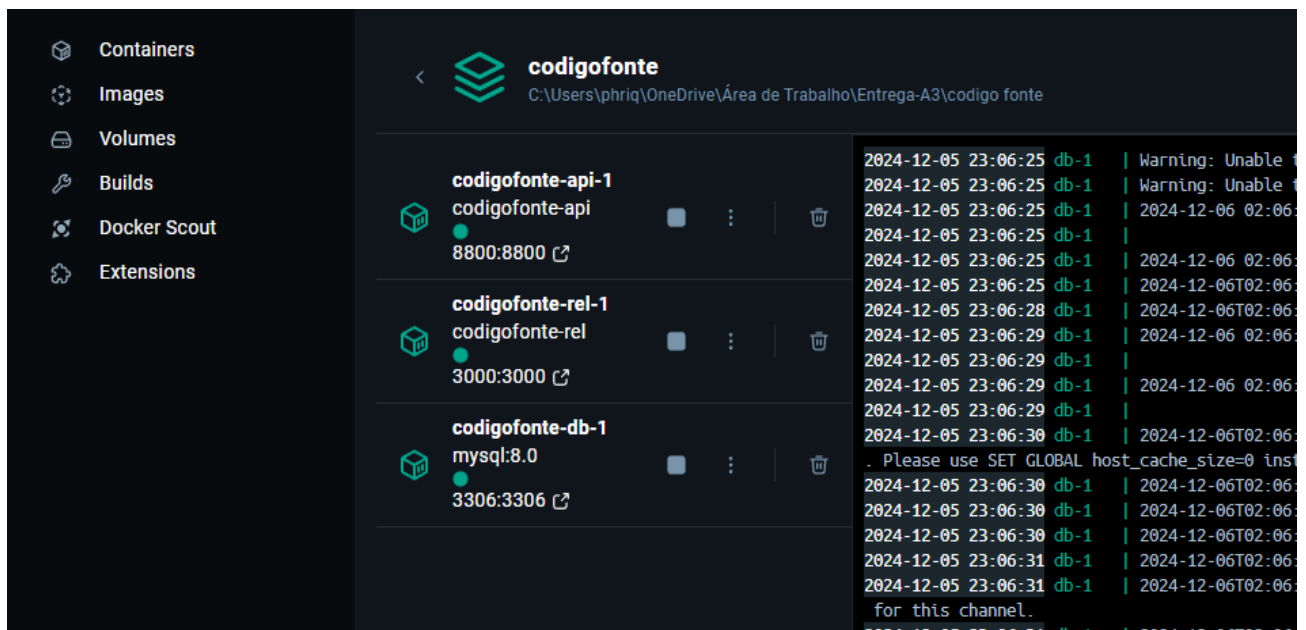


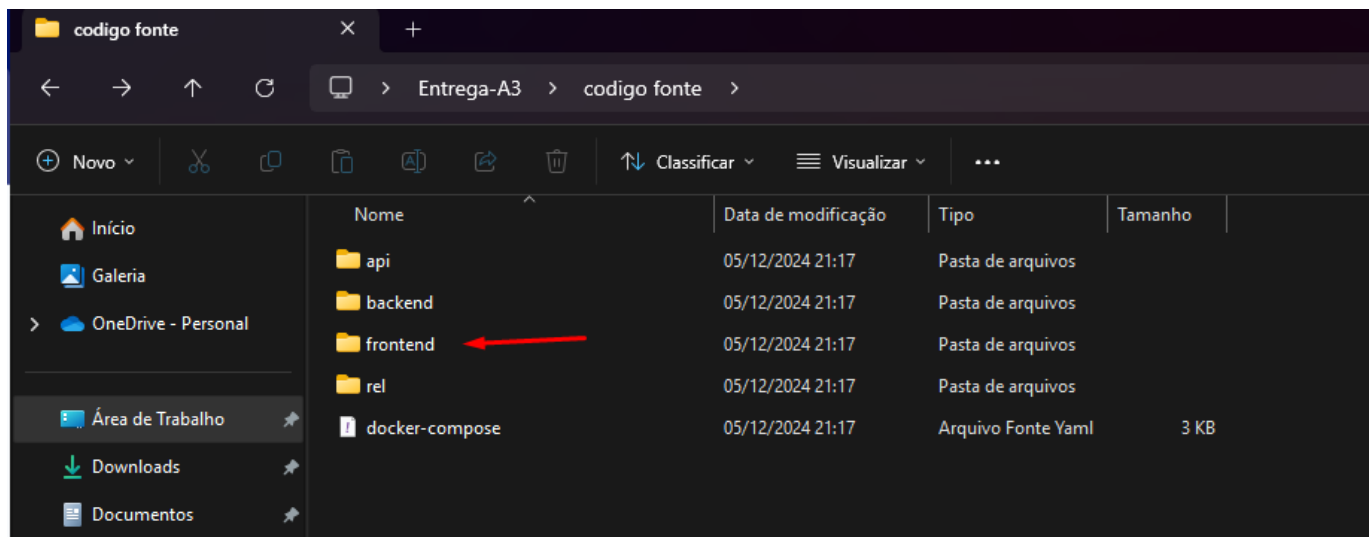
Imagem 3: Tela do Docker Desktop

Utilizamos a versão (Docker Compose version v2.29.2-desktop.2);
Isso irá construir as imagens Docker e iniciar os containers para o MySQL,
API e Relatório.

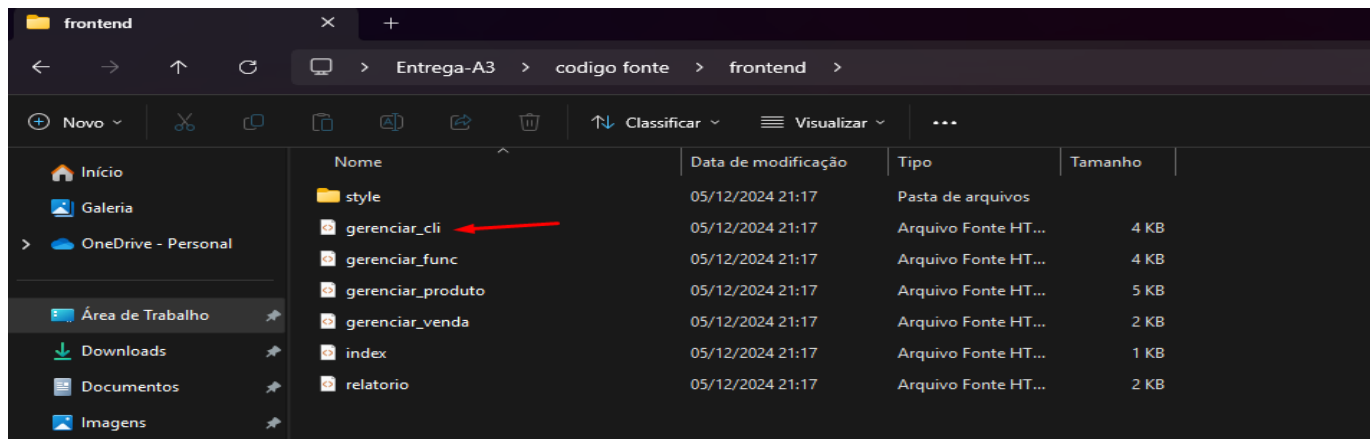


Passo 2: Acessar a Aplicação

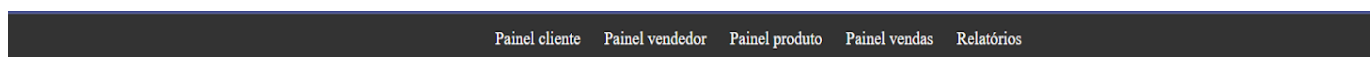
Retorne ao diretório <codigofonte>, abra o diretório <frontend>



Localize o arquivo “*gerenciar_cli*”, clique com o botão direito e
selecione a opção “*Abrir com*” e selecione um Navegador de sua
preferência



Após isso, o nosso sistema de **Simulação de Captação de Dados de Venda de uma Rede de Lojas** estará apto para o uso, conforme a imagem abaixo:



Cadastrar cliente

Nome

Data nascimento

E-mail

Número telefone

5. Descrição da Arquitetura da Aplicação: API, Relatório e Front-End Estático

API: A API estará disponível na porta **8800** do localhost:

<http://localhost:8800>

Métodos disponíveis da API:

```
getCompra,      addCompra,      deleteCompra,    getCompraById,  
getProduto,     addProduto,     updateProduto,   deleteProduto,  
getProdutoById, getUsers,    addUser,         updateUser,     deleteUser,  
getUserById,    getVendedor,    addVendedor,     updateVendedor,  
deleteVendedor, getVendedorById.
```

getCompra: Retorna uma lista de todas as compras, incluindo o nome do cliente, vendedor, produto, quantidade e total da compra.

getCompraById: Retorna os detalhes de uma compra específica, filtrando pelo ID fornecido, incluindo o cliente, vendedor, produto, quantidade e total.

addCompra: Insere uma nova compra no banco de dados, registrando o cliente, vendedor, produto, quantidade e total da compra.

deleteCompra: Deleta uma compra com base no ID fornecido.

getProduto: Retorna todos os produtos cadastrados, com informações como nome, descrição, quantidade e preço.

getProdutoById: Retorna os detalhes de um produto específico, filtrando pelo ID fornecido.

addProduto: Insere um novo produto no banco de dados, incluindo nome, descrição, quantidade e preço.

updateProduto: Atualiza os dados de um produto específico, com base no ID fornecido, incluindo nome, descrição, quantidade e preço.

deleteProduto: Deleta um produto com base no ID fornecido.

getUsers: Retorna todos os clientes cadastrados no sistema.

getUserById: Retorna os detalhes de um cliente específico, filtrando pelo ID fornecido.

addUser: Insere um novo cliente no banco de dados, com informações como nome, data de nascimento, email e número de telefone.

updateUser: Atualiza os dados de um cliente específico, com base no ID fornecido, incluindo nome, data de nascimento, email e número de telefone.

deleteUser: Deleta um cliente com base no ID fornecido.

getVendedor: Retorna todos os vendedores cadastrados no sistema.

getVendedorById: Retorna os detalhes de um vendedor específico, filtrando pelo ID fornecido.

addVendedor: Insere um novo vendedor no banco de dados, com informações como nome, data de nascimento, email, número de telefone e código de matrícula.

updateVendedor: Atualiza os dados de um vendedor específico, com base no ID fornecido, incluindo nome, data de nascimento, email, número de telefone e código de matrícula.

deleteVendedor: Deleta um vendedor com base no ID fornecido.

Relatório: A aplicação de geração de relatórios estará disponível na porta 3000 do localhost:

<http://localhost:3000>

Métodos disponíveis do Relatório:

`getProdutoByQuantidade, getConsumoMedioByCliente, getProdutoMaisVendido, getTotalCompraByCliente`

getProdutoMaisVendido: Retorna os produtos mais vendidos, classificados pela quantidade de unidades vendidas, incluindo o nome e descrição de cada produto.

getTotalCompraByCliente: Retorna o total de unidades compradas por cada cliente, com o id e nome do cliente.

getConsumoMedioByCliente: Retorna a média de consumo (total gasto) por cliente, incluindo o id e nome do cliente.

getProdutoByQuantidade: Retorna produtos com a quantidade de estoque inferior ou igual ao valor especificado na requisição.

Front-End Estático: A página HTML estática estará disponível ao abrir com o navegador qualquer um dos ficheiros .html da pasta ./frontend

Explicação de uso da página:

Página painel do cliente

Painel clientePainel vendedorPainel produtoPainel vendasRelatórios

Cadastrar cliente

Nome

Data nascimento

dd/mm/aaaa

E-mail

Número telefone

Enviar

Limpar

Preencha os inputs com os dados solicitados

Após preenchido clique **Enviar**

Cientes cadastrados na base

Nome	Data de nascimento	E-mail	Número de telefone	Ações
Maria Silva	14/05/1990	maria.silva@example.com	(11) 9 8765-4321	<div><div></div><div></div></div>
João Santos	19/03/1985	joao.santos@example.com	(11) 9 9988-7766	<div><div></div><div></div></div>
Carlos Oliveira	24/08/1992	carlos.oliveira@example.com	(11) 9 5555-4444	<div><div></div><div></div></div>
Patricia Lima	29/11/1988	patricia.lima@example.com	(11) 9 4444-3333	<div><div></div><div></div></div>
Roberto Alves	11/04/1995	roberto.alves@example.com	(11) 9 3333-2222	<div><div></div><div></div></div>

Clique no ícone de lixeira para deletar o cliente

Clique no ícone de lápis para editar os dados do cliente

2024 - Todos os direitos reservados ©

Página painel do vendedor

Painel cliente

Painel vendedor

Painel produto

Painel vendas

Relatórios

Cadastrar vendedor

Nome

Código de matrícula

Data nascimento

dd/mm/aaaa

E-mail

Número telefone

Enviar

Limpar

Preencha os inputs com os dados solicitados

Após preenchido clique **Enviar**

Vendedores cadastrados na base

Cód. matrícula	Nome	Data de nascimento	E-mail	Número de telefone	Ações
001	José Luiz	14/05/1990	jose.luiz@example.com	(11) 9 8888-7070	<div><div></div><div></div></div>
002	Ana Paula	09/07/1988	ana.paula@example.com	(11) 9 7777-6666	<div><div></div><div></div></div>

Clique no ícon de lixeira para deletar o vendedor

Clique no ícon de lápis para editar os dados do vendedor

2024 - Todos os direitos reservados ©

Página painel do produto

Painel clientePainel vendedorPainel produtoPainel vendasEstadísticas

Cadastrar novo produto:

Nome:

Descrição:

Quantidade:

Preço:

Enviar

Limpar

Preencha os inputs com os dados solicitados

Após preenchido clique **Enviar**

Tabela de preços e produtos

Produto	Descrição	Quantidade	Preço (R\$)	Ações
Guirama Clássico	Guirama tradicional para jardins	50	89	<div><div><div></div></div><div><div></div></div></div>
Guirama Pescador	Guirama com vaso de peixe	30	129	<div><div><div></div></div><div><div></div></div></div>
Guirama Doméstico	Guirama decorado para decoração	40	99	<div><div><div></div></div><div><div></div></div></div>
Guirama Jardineiro	Guirama com ferramentas de jardim	25	119	<div><div><div></div></div><div><div></div></div></div>
Guirama Místico	Guirama tocado acrílico	20	139	<div><div><div></div></div><div><div></div></div></div>
Guirama Casal	Par de guiramas realísticos	15	179	<div><div><div></div></div><div><div></div></div></div>
Guirama Luminoso	Guirama com lâmpada solar	35	159	<div><div><div></div></div><div><div></div></div></div>
Guirama Festivo	Guirama montado em copanete	45	109	<div><div><div></div></div><div><div></div></div></div>
Guirama Mágico	Guirama com varinha mágica	30	149	<div><div><div></div></div><div><div></div></div></div>
Guirama Guerreiro	Guirama com escudo e espada	25	169	<div><div><div></div></div><div><div></div></div></div>

Clique no ícone de lixeira para deletar o produto

Clique no ícone de lápis para editar os dados do produto

Clique no ícone do carrinho de compras para comprar produtos

2023 - Todos os direitos reservados ©

Modal do carrinho de compras

Vendedor:
José Luiz

Cliente:
João Santos

Produto:
1
Produto: Gnome Clássico
Valor(un): R\$ 89

Quantidade:
10

Total: R\$ 890.00

Finalizar
Cancelar

Selecione o vendedor e o cliente

Insira uma quantidade do produto

Clique em Finalizar



6. Apresentação e Detalhamento sobre a Arquitetura, Estratégia e Algoritmos Utilizados

Arquitetura da Aplicação

A arquitetura foi projetada de forma modular e escalável, com a separação das responsabilidades entre os diferentes serviços. Utilizamos a abordagem de **micro serviços**, o que nos permite:

1. **Separar a API principal da API de relatórios**, o que facilita o desenvolvimento, manutenção e escalabilidade.
2. **Banco de Dados MySQL** para armazenar dados de vendas, produtos e clientes, com tabelas e relacionamentos bem definidos.

A arquitetura consiste nos seguintes componentes principais:

- **API Principal:** Responsável pela captação e manipulação dos dados de vendas. Ela recebe as requisições do frontend, interage com o banco de dados e retorna os dados em formato JSON.
- **API de Relatórios:** Um serviço separado que consulta o banco de dados e gera relatórios com base nos dados capturados pela API principal.
- **Banco de Dados MySQL:** Armazena as informações de vendas, clientes e outros dados necessários para a aplicação.
- **Front-End Estático:** Uma página HTML simples que permite ao usuário interagir com a aplicação sem a necessidade de integração com as APIs.

Estratégia Utilizada

A estratégia de desenvolvimento foi baseada na **separação de responsabilidades** entre os serviços. Isso facilita a manutenção e a escalabilidade do sistema:

1. **API Principal e Relatório Independentes:** Ao dividir as responsabilidades, a API de relatórios pode ser mantida e escalada separadamente da API de vendas. Ambas as APIs são construídas com Express.js, utilizando rotas e controllers para manter o código modular e reutilizável.
2. **Docker como Facilitador:** A escolha de usar Docker garante que todos os serviços da aplicação (API, banco de dados e relatórios) sejam executados em containers isolados, com todas as dependências necessárias, garantindo que a aplicação rode da mesma forma em qualquer ambiente.

Algoritmos e Fluxo de Dados

- **API de Vendas:** A API principal realiza operações CRUD (Create, Read, Update, Delete) no banco de dados MySQL para gerenciar os dados de vendas.
- **API de Relatórios:** Realiza consultas agregadas no banco de dados e gera relatórios baseados nos dados de vendas capturados pela API principal. A lógica de geração de relatórios pode incluir a filtragem de dados, ordenação e agregação de vendas por data, produto ou vendedor.

Front-End Estático

A equipe optou por construir um **front-end estático** em HTML e CSS para facilitar o uso da aplicação. A página contém os seguintes componentes:

- **Inputs:** Campos de entrada para capturar dados relacionados às vendas.
- **Botões:** Botões para enviar os dados para a API e gerar relatórios.

O **front-end** não está integrado diretamente com as APIs, mas serve como uma interface simples para que o usuário interaja com os dados de forma intuitiva.

7. Considerações Finais

O desenvolvimento deste projeto possibilitou a aplicação de diversas tecnologias: como Node.js, MySQL e Docker. Além de mostrar a viabilidade de criar uma API funcional e escalável, foi possível descobrir sobre a integração eficiente entre esses componentes.

Entre os principais desafios, destacam-se a configuração do ambiente com Docker e a otimização do fluxo de desenvolvimento. A resolução desses problemas envolveu pesquisa e experimentação. Outro aprendizado foi a necessidade de manter um código bem documentado e organizado, que é uma das partes fundamentais para a manutenção e evolução do projeto.

Os resultados alcançados foram positivos, com uma aplicação plenamente funcional que pode ser facilmente usável, adaptada e expandida. Como trabalhos futuros, seria interessante implementar autenticação, melhorias no front-end e testes automatizados. Este projeto serviu não apenas como um aprendizado técnico, mas também como uma oportunidade de explorar a nova ferramenta de aprendizado, o Docker, não deixando de lado as soluções criativas e os desafios no desenvolvimento de software.

8. Bibliografia

- (1) [Desenvolvimento de aplicativos da Web em 2024 | AppMaster](#)

[Como está o mercado de desenvolvimento web?](#)

- (2) [Node.JS: definição, características, vantagens e usos possíveis | Alura](#)

[Node.js Documentation](#)

- (3) [Express - framework de aplicativo da web Node.js](#)

[Express Documentation](#)

- (3) [O que é o MySQL| Oracle Brasil](#)

[MySQL Documentation](#)

- (4) [O que é o Docker e quais as Vantagens de usá-lo? | Alura](#)

[Docker Documentation](#)

- (5) [SQL: O que é, quais os comandos e como utilizar? | Alura](#)

[O que é um ORM – o significado das ferramentas de mapeamento relacional de objetos de banco de dados](#)

- (6) [O que é um diagrama entidade relacionamento? | Lucidchart](#)