UNIFACS
UNIVERSIDADE SALVADOR
LAUREATE INTERNATIONAL UNIVERSITIES

SISTEMAS DE INFORMAÇÃO

RELATÓRIO SISTEMAS DISTRBUÍDOS E MOBILE

GUILHERME ORNELLAS CARVALHO

JORGE HENRIQUE RAMOS GANDOLFI

JOÃO VICTOR MORAES LEITE

TACYO DOS SANTOS UZEDA

VICTOR OLIVEIRA CERQUEIRA

GUILHERME ORNELLAS CARVALHO

Salvador
2024

JORGE HENRIQUE RAMOS GANDOLFI

JOÃO VICTOR MORAES LEITE

TACYO DOS SANTOS UZEDA

VICTOR OLIVEIRA CERQUEIRA

ARQUITETURA EMPRESARIAL: ESTUDO DE CASO SOLUTIS

Trabalho apresentado à disciplina Sistemas Distribuídos e mobile Integrado ao curso Sistema de Informação pela Universidade de Salvador - Campus Tancredo Neves, como requisito parcial para a obtenção de nota da A3.

Orientador: Prof. Adailton de Jesus Cerqueira Junior

Introdução

Atividade Avaliativa: Rede de Loja

Com base no que foi ordenado pelo professor Adailton de Jesus Cerqueira Junior, docente da disciplina de Sistemas Distribuídos e Mobile, foi proposto uma oportunidade de

desenvolver uma aplicação para simular a captação de dados de vendas de uma rede de loja. Inicialmente, tal projeto consiste na criação de uma interface em HTML que permite o registro sobre as transações comerciais e no cadastro de clientes e fornecedores. Além disso, a escolha do HTML com complementação do Javascript para a construção do trabalho, foi selecionado por conta do conhecimento atribuído dessa disciplina.

Em primeira análise, a proposta do trabalho não exigia uma linguagem específica, e assim o HTML correspondia ao adequado para criar uma interface que permite que os dados de vendas fossem inseridos e visualizados. Ademais, foi necessário adquirir conhecimentos externos, por conta de o trabalho ter sido feito com métodos que não foram aplicados por Adailton e Marivaldo, a exemplo do CSS atrelado ao HTML para a construção do nosso front-end

Em segunda análise, aplicação fundamentada em aula foi Mysql Workbench e está mesma ferramenta será utilizada pela equipe, porque é um banco de dados ideal para armazenar e gerenciar dados de vendas de uma rede de lojas de forma persistente e estruturada. O mesmo é um tanto avançado em manipulação e consulta de dados em larga escala, possui uma maior eficiência e flexibilidade para o desenvolvimento de aplicação em dados distribuídos. Além disso, o Mysql é uma solução amplamente usada em ambientes de produção, proporcionando maior escalabilidade e integridade dos dados principalmente com a ferramenta utilizada para a criação de containers, como o Docker.

Fundamentação Teórica

O trabalho é composto pelo arquivo principal (index), e seus complementes quem fornecem utilidade nos setores de estoque, cliente e produto auxiliando no banco de dados.

- **Front e Back-End**

- **Index:** O arquivo principal (index) está responsável por cadastrar, e assim será pedido informações precisas, mas básicas, que são o CPF, nome, e-mail, telefone e endereço. Depois do cadastro, as informações estarão armazenadas no banco de dados, que poderá ser usada para listar todos os cadastros quando requerer.
- **Consultar Produto:** O código fornecido é uma aplicação web que permite aos usuários consultar produtos a partir de um formulário de busca. A interface inclui campos para inserir informações como ID, nome, marca, categoria e fornecedor do produto. Após o preenchimento dos campos, ao submeter o formulário, uma requisição HTTP é feita ao backend usando o método fetch, passando os parâmetros inseridos no formulário. O servidor retorna os produtos que correspondem aos critérios de pesquisa, e os resultados são exibidos dinamicamente em uma tabela na mesma página. O uso de async/await no JavaScript permite o tratamento assíncrono da requisição, garantindo uma experiência de usuário fluída. A tabela é atualizada com os dados dos produtos retornados sem recarregar a página, proporcionando uma interação eficiente. Esse tipo de consulta é útil para sistemas de estoque ou plataformas de e-commerce que precisam filtrar produtos com base em diferentes atributos.
- **Estoque:** O código é uma aplicação para a gestão de estoque de produtos, permitindo o cadastro, controle de movimentações e visualização dos produtos em estoque. A interface é dividida em seções, incluindo um formulário para cadastro de produtos com campos como nome, descrição, categoria, marca, modelo, especificações e fornecedor. Outro formulário permite registrar movimentações de estoque, como entradas e saídas de produtos, com informações sobre a quantidade e valor. As movimentações e produtos são exibidos dinamicamente em tabelas. O JavaScript gerencia os eventos de submissão dos formulários, realizando o armazenamento das movimentações em um array e atualizando a visualização na página sem a necessidade de recarregar. Há também funcionalidades de edição e exclusão de produtos registrados. O uso de alert informa o sucesso das operações e forEach é utilizado para renderizar os dados nas tabelas. A estrutura facilita a gestão e o controle de estoque em tempo real.
- **Clientes:** O código apresentado é uma aplicação web que permite o gerenciamento de dados de clientes, incluindo operações CRUD (Criar, Ler, Atualizar e Deletar). Ele é estruturado em HTML, com um formulário de cadastro que coleta informações como nome, CPF, endereço e email. Essas informações são enviadas para um backend através de requisições HTTP, utilizando o método fetch para as operações

de POST (criar), PUT (atualizar) e DELETE (deletar), e GET para ler os dados. A interação com o usuário é dinâmica, já que a tabela de clientes é atualizada automaticamente sem a necessidade de recarregar a página. O JavaScript manipula a exibição dos dados na tabela e permite a seleção de um cliente para atualização ou exclusão. A validação dos campos do formulário é feita através do atributo required no HTML, garantindo que os dados essenciais sejam preenchidos antes do envio. Esse modelo de gestão facilita a interação do usuário com a plataforma, proporcionando uma interface eficiente para a administração dos dados dos clientes.

- **Relatório de Produtos:** Esse código HTML permite a visualização de relatórios de vendas, consumo e estoque de produtos por meio de gráficos interativos gerados com a biblioteca Chart.js. Ele utiliza uma estrutura de dados dinâmica, onde informações são obtidas via uma solicitação HTTP (fetch) a um backend, retornando um objeto JSON com as métricas de desempenho dos produtos, como vendas, consumo por cliente e estoque. Cada seção do relatório é acompanhada de um gráfico correspondente, como gráficos de barras para vendas e estoque, e gráficos de linha para consumo médio. A interação com os dados ocorre em tempo real, permitindo a análise detalhada e a tomada de decisões baseada em dados atualizados, enquanto as tabelas exibem as mesmas informações de forma tabular.
- **Vendas:** Esse código HTML permite a gestão de vendas através de um formulário dinâmico, onde o usuário pode selecionar um cliente, vendedor, produto e informar a quantidade do produto para registrar uma venda. A página interage com uma API backend para buscar dados sobre clientes, vendedores e produtos, bem como registrar, atualizar e cancelar vendas. O formulário envia os dados por meio de requisições HTTP para a API, utilizando métodos como POST (para registrar), PUT (para atualizar) e DELETE (para cancelar). As vendas registradas são exibidas em uma tabela dinâmica, e o usuário pode interagir com as entradas, selecionando uma venda para ser editada ou cancelada. Esse processo facilita o gerenciamento de vendas de maneira prática e eficiente, com atualizações em tempo real.
- **Vendedores:** Esse código HTML foi desenvolvido para a gestão de vendedores em um sistema. Ele permite cadastrar, atualizar e excluir vendedores, além de exibir todos os vendedores cadastrados em uma tabela dinâmica. A interação com o usuário é realizada por meio de um formulário que coleta informações como nome, CPF e e-mail do vendedor. Essas informações são enviadas para a API backend usando métodos HTTP como POST (para cadastro), PUT (para atualização) e DELETE (para exclusão). Quando a página é carregada, a tabela de vendedores é automaticamente

preenchida com os dados provenientes de um banco de dados. O usuário pode clicar nas linhas da tabela para editar ou excluir um vendedor, e os dados do formulário serão atualizados conforme a seleção. As operações são realizadas de forma assíncrona com o uso de fetch(), garantindo que a interface seja atualizada sem necessidade de recarregar a página.

- **Banco de Dados**

No banco de dados foram criadas 5 tabelas de dados:

- A primeira é a tabela dos Clientes contendo o ID do cliente como chave primaria, o nome do cliente, CPF como chave única e o Endereço.
- Em seguida, a tabela dos Fornecedores, contém o ID como chave primaria, o nome da empresa, o CNPJ do como chave única, o Endereço e o nome do funcionário mediador.
- Na tabela Funcionários, contém o ID do funcionário como chave primaria, o Nome do funcionário, seu Cargo, e como chaves únicas o Login e a Senha de acesso.
- Na tabela de Produtos, possui as colunas ID como chave primaria, Nome, Categoria, Descrição com suas especificações, a Quantidade em estoque, a Localização e uma chave estrangeira com o ID do fornecedor.
- Na tabela de Movimentações, possui as colunas ID de movimentação como chave primaria, ID do produto como chave estrangeira da tabela produto, ID de funcionário como chave estrangeira da tabela funcionário, o Tipo de movimentação, a Quantidade movimentada, e o ID do cliente como chave estrangeira da tabela clientes.

Projeto de Implementação

- **Front-End**

- **Index:** Aplicado em HTML, o arquivo principal (index) cadastra o cliente em um form criado chamado de “addClienteForm” para os inputs CPF, nome, e-mail, telefone e endereço que são da marcação label, isso ocorre dentro do ‘Body’. Após

isso, já no script, há um chamado em get do “addClienteForm” para o addEventListener , no qual será responsável em enviar para o banco de dados e de imprimir a lista dos cadastros. Nessa passagem do form para o addEventListener transforma os inputs em constantes, envia para o banco de dados através await fetch, armazenando em “/clientes” quando precisar retornar a tabelado. A verificação dessas ações está ocorrendo corretamente será em If Else.

- **Clientes:** Este projeto visa criar um sistema web para o gerenciamento de clientes, permitindo o cadastro, atualização, exclusão e exibição dos dados dos clientes. A aplicação utiliza HTML, CSS, e JavaScript para estruturar e gerenciar a interação com os dados dos clientes de maneira dinâmica. O sistema tem um formulário onde é possível inserir as informações de um novo cliente, incluindo nome, CPF, endereço e e-mail. Após preencher os campos, o usuário pode cadastrar o cliente, enviando uma requisição POST para o backend. Se a operação for bem-sucedida, o cliente será adicionado ao banco de dados, e a lista de clientes será atualizada automaticamente na página. Caso contrário, o usuário será notificado sobre o erro. Além do cadastro, o sistema também oferece funcionalidades para atualizar e excluir dados de clientes. Para isso, ao clicar em um cliente na tabela de clientes, suas informações serão carregadas no formulário, permitindo ao usuário editá-las. Quando o botão "Atualizar Cliente" é pressionado, uma requisição PUT é enviada ao backend para atualizar os dados do cliente no banco de dados. Se o usuário precisar excluir um cliente, ele pode selecioná-lo na tabela e clicar no botão "Deletar Cliente", o que envia uma requisição DELETE para o backend, removendo o cliente selecionado. A tabela de clientes exibe todas as informações cadastradas, como o ID do cliente, nome, CPF, endereço e e-mail. Um campo adicional mostra os produtos adquiridos, se houverem. Cada linha da tabela é interativa, permitindo ao usuário clicar em um cliente para preencher automaticamente o formulário com seus dados, o que facilita a atualização ou exclusão.O processo de interação começa com a página sendo carregada, momento em que uma requisição GET é enviada ao backend para buscar todos os clientes cadastrados e preencher a tabela com os dados. O sistema então permite o cadastro de novos clientes, a atualização de dados existentes e a exclusão de registros, sempre atualizando a lista de clientes na tabela após cada operação. O frontend e o backend se comunicam através de requisições HTTP, garantindo que as informações sejam sempre atualizadas e consistentes.
- **Consultar Produtor:** O sistema desenvolvido tem como objetivo permitir a consulta de produtos em estoque, fornecendo ao usuário uma interface onde ele pode

pesquisar por diferentes critérios, como ID, nome, marca, categoria e fornecedor do produto. A aplicação utiliza HTML, CSS e JavaScript para criar a interface de consulta e se comunicar com o backend, mostrando os resultados sem a necessidade de recarregar a página. A estrutura da página é composta por um formulário de consulta, onde o usuário pode preencher os campos correspondentes aos diferentes parâmetros de pesquisa. O formulário contém cinco campos de entrada: ID do Produto, Nome do Produto, Marca do Produto, Categoria do Produto e Fornecedor do Produto. O usuário pode preencher qualquer combinação desses campos para realizar a busca. Após preencher os campos desejados, ele pode clicar no botão "Consultar", que dispara uma requisição ao backend para buscar os produtos com os critérios especificados. A consulta é feita por meio de uma requisição GET para o endpoint /estoque/produtos/consulta, onde os parâmetros de busca são passados na URL (como id, nome, marca, etc.). O backend responde com os produtos que correspondem aos critérios de pesquisa fornecidos. A resposta é no formato JSON, e os dados retornados são então processados no frontend para exibição na tabela. A tabela, inicialmente vazia, é preenchida com os resultados da consulta. Cada linha da tabela representa um produto, com colunas para o ID, nome, marca, categoria e fornecedor. Se nenhum produto for encontrado para os critérios de pesquisa, a tabela ficará vazia, mas a consulta ainda será processada normalmente. O script JavaScript, responsável por lidar com a lógica da consulta, é ativado quando o formulário é submetido. Ele impede o envio padrão do formulário e faz a requisição assíncrona usando a função fetch.

- **Estoque:** Através de um conjunto de formulários e tabelas, o sistema permite o cadastro de novos produtos, o registro de movimentações de estoque e a visualização do status atual dos itens no inventário. O sistema possui uma interface onde o usuário pode cadastrar produtos informando nome, descrição, categoria, marca, modelo, especificações técnicas e fornecedor. Esses dados são armazenados temporariamente no sistema, possibilitando o controle sobre os itens disponíveis no estoque. Além disso, o sistema permite registrar as movimentações de estoque, que podem ser de entrada (quando novos produtos são adicionados) ou de saída (quando produtos são retirados do estoque). O formulário de movimentação exige que o usuário informe o nome do produto, a quantidade e o valor, além de selecionar o tipo de movimentação. As informações de movimentação são salvas e exibidas em uma tabela, onde o usuário pode visualizar as transações realizadas, incluindo data, produto, quantidade e tipo de movimentação. Na interface, os produtos cadastrados aparecem em uma

tabela com o nome, quantidade, valor e as datas de entrada ou saída. Essa tabela é atualizada dinamicamente à medida que o estoque sofre alterações. O sistema também oferece a possibilidade de editar ou excluir produtos diretamente da tabela, garantindo flexibilidade na gestão do inventário. Além das funcionalidades de cadastro e movimentação, o sistema mantém um histórico completo das ações realizadas, possibilitando um acompanhamento detalhado das movimentações no estoque ao longo do tempo. O histórico é visualizado em outra tabela, onde o usuário pode verificar a data de cada movimentação, o produto envolvido, a quantidade movimentada e o responsável.

- **Relatório do Produto:** A implementação deste código visa desenvolver uma interface interativa para a geração de relatórios detalhados sobre os produtos em uma plataforma de vendas. Utilizando HTML, CSS, JavaScript e a biblioteca Chart.js para gerar gráficos dinâmicos, o projeto permite visualizar informações sobre os produtos mais vendidos, os produtos adquiridos por clientes, o consumo médio por cliente e o estoque de produtos. A página é organizada em várias seções, cada uma apresentando um relatório específico. Cada seção contém tanto gráficos interativos quanto tabelas, oferecendo uma visão detalhada e visualmente atraente dos dados. A implementação dos gráficos permite que os usuários analisem facilmente os dados de vendas, consumo e estoque, visualizando as informações de forma clara. A primeira seção exibe um gráfico de barras que representa os produtos mais vendidos, permitindo identificar quais itens estão gerando mais vendas. Os dados para o gráfico são extraídos de uma API que fornece as informações sobre os produtos e as vendas realizadas. Ao lado do gráfico, uma tabela é preenchida automaticamente com os nomes dos produtos e as respectivas quantidades vendidas. Na segunda seção, é exibido um gráfico de barras que mostra a quantidade de produtos adquiridos por cada cliente. Esses dados são importantes para entender o comportamento de compra dos clientes, permitindo segmentar os produtos mais procurados por cada um. A tabela ao lado do gráfico lista os clientes, os produtos comprados e a quantidade adquirida. A terceira seção mostra um gráfico de linha, que ilustra o consumo médio de cada cliente. O gráfico de linha é uma boa escolha para mostrar as tendências de consumo ao longo do tempo ou por cliente. A tabela associada mostra o cliente e o seu consumo médio, fornecendo uma visão quantitativa do comportamento de consumo. Por fim, a última seção contém um gráfico de barras que destaca os produtos com baixo estoque, permitindo aos administradores identificar rapidamente quais itens precisam ser reabastecidos. A tabela ao lado do gráfico lista os produtos

e suas quantidades em estoque, proporcionando um controle eficaz sobre o inventário. A interação entre os gráficos e as tabelas é dinâmica e feita via JavaScript. A coleta dos dados é realizada por meio de requisições para uma API, que fornece as informações necessárias para preencher os gráficos e tabelas. A utilização do fetch garante que os dados sejam recuperados de maneira assíncrona, permitindo uma atualização eficiente dos gráficos e tabelas sem a necessidade de recarregar a página. A biblioteca Chart.js é utilizada para renderizar gráficos interativos e responsivos, facilitando a visualização das informações.

- **Vendas:** O sistema permitirá registrar, atualizar e excluir vendas, além de listar todas as vendas realizadas. Ele também integra informações sobre clientes, vendedores e produtos, mas o foco principal é a interação simples e eficaz com o usuário, por meio de formulários dinâmicos e atualizações em tempo real. As tecnologias utilizadas incluem o HTML para estruturação, CSS para estilo e JavaScript para manipulação de dados, API REST que disponibiliza endpoints para interagir com os dados de clientes, vendedores, produtos e vendas. As requisições HTTP assíncronas são realizadas usando a Fetch API para acessar e modificar informações armazenadas no backend. O sistema permite que o usuário registre uma nova venda, selecionando o cliente, vendedor e produto, além de informar a quantidade do produto. A venda é registrada ao clicar no botão "Registrar Venda", que envia os dados para o backend. Caso o usuário precise atualizar ou excluir uma venda, ele pode selecionar uma venda na tabela e clicar nos botões "Atualizar Venda" ou "Cancelar Venda", que permitem editar ou remover o item da base de dados. A tabela de vendas exibe uma lista de todas as vendas registradas, mostrando o ID da venda, o nome do cliente, do vendedor, do produto e a quantidade vendida. Ao clicar em uma venda, os dados são carregados automaticamente no formulário, permitindo que o usuário modifique a venda ou a exclua. A página carrega as listas de clientes, vendedores e produtos dinamicamente, utilizando chamadas para a API, que retornam os dados em formato JSON. Esses dados são então preenchidos automaticamente nos campos de seleção do formulário. O fluxo de dados começa quando a página é carregada, realizando requisições GET para obter informações sobre clientes, vendedores e produtos e exibindo as vendas na tabela. Quando o usuário registra uma nova venda, os dados são enviados via requisição POST para o backend, que registra a venda no banco de dados. Se o usuário deseja atualizar ou cancelar uma venda, ele seleciona a venda desejada, faz as modificações necessárias e clica em "Atualizar Venda" ou "Cancelar Venda", respectivamente, enviando as requisições PUT ou DELETE para a API. A

lista de vendas é atualizada a cada operação para refletir as mudanças. No backend, a API oferece endpoints para listar, criar, atualizar e excluir informações relacionadas a clientes, vendedores, produtos e vendas. A comunicação entre o frontend e o backend ocorre através de requisições HTTP utilizando os métodos GET, POST, PUT e DELETE, garantindo que o sistema seja dinâmico e capaz de responder rapidamente às interações do usuário.

- **Vendedores:** O sistema permitirá cadastrar, atualizar e excluir vendedores, além de listar todos os vendedores registrados. As tecnologias utilizadas incluem o HTML é utilizado para definir a estrutura do conteúdo, CSS para o estilo visual da página e JavaScript para a manipulação dos dados e da interação com o backend, API REST que fornece endpoints para interagir com os dados dos vendedores, permitindo realizar operações de cadastro, atualização e exclusão. O sistema permite que o usuário cadastre um novo vendedor, preenchendo os campos de nome, CPF e e-mail. A requisição de cadastro é realizada via uma requisição POST, que envia os dados para o backend e os armazena. O sistema também permite atualizar as informações de um vendedor. Para isso, o usuário seleciona um vendedor da tabela e modifica os dados no formulário. Ao clicar no botão "Atualizar Vendedor", os dados modificados são enviados para o backend via requisição PUT. Caso o usuário precise excluir um vendedor, ele pode selecionar o vendedor desejado na tabela e clicar em "Deletar Vendedor", o que realiza uma requisição DELETE para remover o vendedor do sistema. A tabela de vendedores exibe uma lista de todos os vendedores cadastrados, mostrando o ID do vendedor, seu nome, CPF e e-mail. Cada linha da tabela é interativa, permitindo ao usuário clicar em um vendedor para carregar seus dados no formulário. O formulário pode então ser usado para realizar modificações ou exclusões. A página carrega automaticamente os vendedores ao ser inicializada, fazendo uma requisição GET para o backend que retorna os dados de todos os vendedores cadastrados. Esses dados são preenchidos na tabela e mantidos atualizados à medida que operações de cadastro, atualização e exclusão são realizadas. O fluxo de dados começa quando a página é carregada, realizando uma requisição GET para listar todos os vendedores e preenchendo a tabela com as informações. Quando o usuário deseja cadastrar um novo vendedor, ele preenche o formulário e envia os dados via requisição POST. Se o usuário precisar atualizar ou excluir um vendedor, ele seleciona um vendedor na tabela, faz as modificações necessárias e envia as alterações ou exclui o vendedor através das requisições PUT ou DELETE. Cada operação resulta na atualização automática da lista de vendedores

na página. No backend, a API oferece endpoints para listar, cadastrar, atualizar e excluir vendedores. As interações entre o frontend e o backend são feitas por meio de requisições HTTP usando os métodos GET, POST, PUT e DELETE.

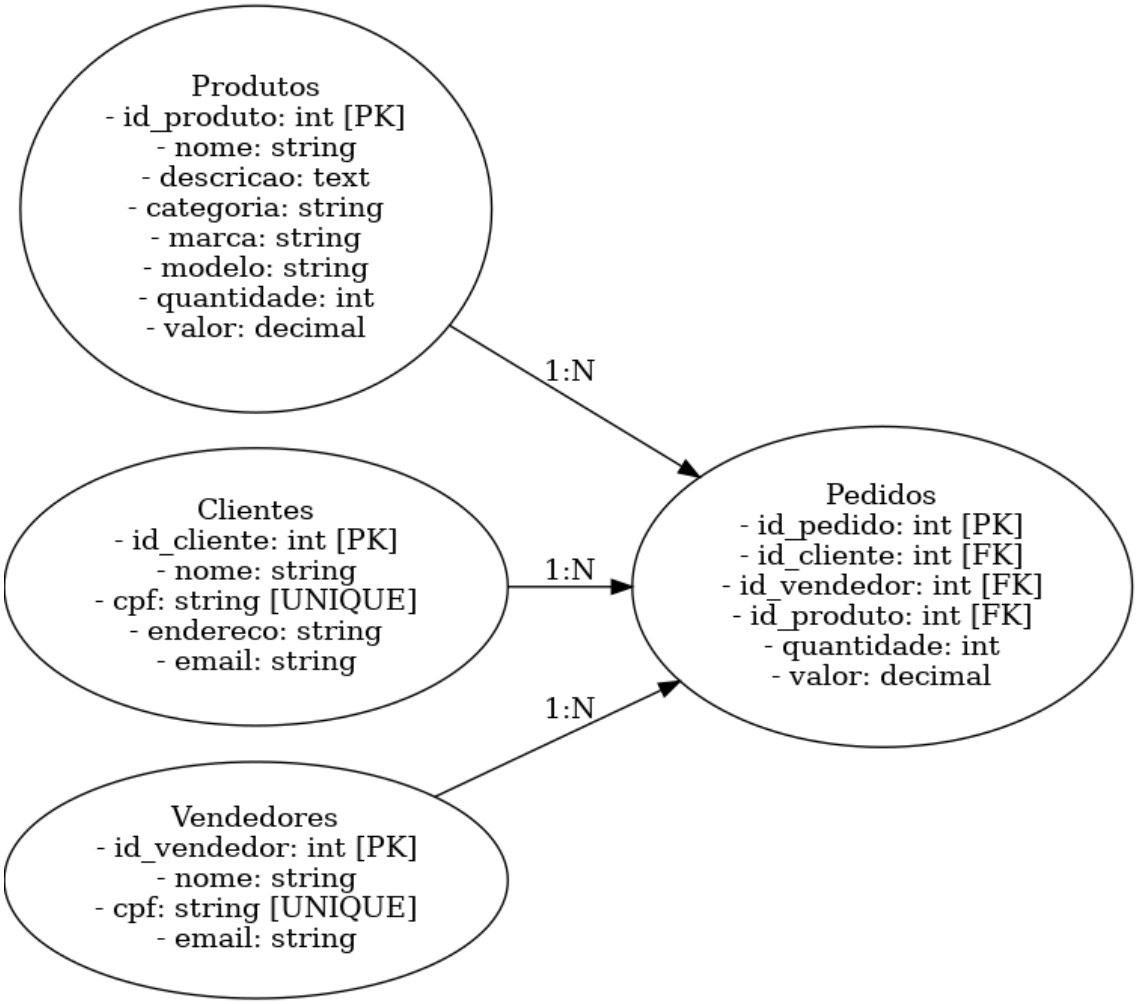
- **Banco de Dados**

Este projeto estabelece um sistema de gestão de loja utilizando MySQL e Docker Compose, permitindo o gerenciamento eficiente de fornecedores, produtos, clientes, vendedores, pedidos e movimentações de estoque. A infraestrutura é composta por dois serviços principais: um backend, denominado App, configurado para acessar o banco de dados e um serviço MySQL, que contém o banco relacional e suporta persistência de dados por meio de volumes Docker. Ambos os serviços estão conectados a uma rede interna para facilitar a comunicação.
O banco de dados é estruturado para atender às necessidades de uma loja, contendo tabelas que representam fornecedores, produtos, clientes, vendedores, pedidos e movimentações. A tabela de produtos inclui detalhes como especificações, categoria, preço, estoque e vinculação a fornecedores. Já as tabelas de clientes e vendedores armazenam informações pessoais e de contato, enquanto a tabela de pedidos conecta clientes, vendedores e produtos vendidos. A tabela de movimentações registra entradas e saídas de estoque, garantindo controle detalhado.

Relatórios gerenciais são gerados automaticamente por meio de views SQL, que incluem dados como produtos mais vendidos, consumo médio de clientes e produtos com estoque crítico (≤ 10 unidades). Essas views fornecem insights valiosos para tomada de decisão e otimização do estoque. O banco foi inicialmente populado com dados fictícios para testes e validação, incluindo fornecedores e produtos de diferentes categorias, além de movimentações e pedidos simulados. A infraestrutura Docker garante portabilidade e fácil configuração do ambiente, utilizando volumes para persistência dos dados e scripts de inicialização para criação automática das tabelas e população inicial do banco. Um sistema de healthcheck assegura que o banco de dados esteja funcional antes do backend ser iniciado, aumentando a confiabilidade do ambiente. Com um design modular e expansível, o sistema oferece uma base sólida para operações

de loja, permitindo controle detalhado das atividades e suporte a decisões estratégicas. O projeto pode ser ampliado para incluir um frontend ou integração com APIs REST, melhorando a interface com os usuários e potencializando as funcionalidades já existentes.

- **UML Banco de Dados:**



O diagrama apresentado descreve a estrutura do banco de dados para uma aplicação de gerenciamento de produtos, clientes, vendedores e pedidos. Aqui estão os pontos principais:

Entidades e Atributos

1. **Produtos**:
    a. id_produto (PK)
    b. nome
    c. descricao
    d. categoria
    e. marca
    f. modelo
    g. quantidade
    h. Valor

2. **Clientes**:
    a. id_cliente (PK)
    b. nome
    c. cpf (UNIQUE)
    d. endereco
    e. email
3. **Vendedores**:
    a. id_vendedor (PK)
    b. nome
    c. cpf (UNIQUE)
    d. email
4. **Pedidos**:
    a. id_pedido (PK)
    b. id_cliente (FK)
    c. id_vendedor (FK)
    d. id_produto (FK)
    e. quantidade
    f. valor

Relacionamentos

- Cada **Produto** pode estar associado a múltiplos **Pedidos** (1:N).
- Cada **Cliente** pode fazer múltiplos **Pedidos** (1:N).
- Cada **Vendedor** pode gerenciar múltiplos **Pedidos** (1:N).

- **Docker**

Docker para uma aplicação Node.js e um banco de dados MySQL. A aplicação é construída a partir de uma imagem oficial do Node.js (versão 18), com o diretório de trabalho definido em /usr/src/app. As dependências do projeto são instaladas usando npm install, e o servidor Node.js é iniciado com o comando node app.js, expondo a porta 3000.
No arquivo docker-compose.yml, o serviço da aplicação (app) depende do serviço MySQL (mysql), que deve estar saudável antes de iniciar. O serviço da aplicação também configura variáveis de ambiente com informações de conexão ao banco de dados, como host, usuário, senha e nome do banco. Os volumes mapeiam o código da aplicação no host para o container, permitindo desenvolvimento dinâmico. O serviço MySQL utiliza a imagem oficial do MySQL 8 e é configurado com uma senha para o usuário root e o banco de dados inicial mysqlloja. O MySQL também usa volumes para persistir os dados e um arquivo SQL de inicialização. Além disso, é configurado um healthcheck para monitorar a disponibilidade do banco de dados com um teste de ping.

Ambos os serviços compartilham a rede backend, permitindo a comunicação entre a aplicação e o banco de dados. O volume db_data é utilizado para garantir a persistência dos dados do MySQL. Em resumo, a configuração oferece um ambiente isolado e escalável para o desenvolvimento, com fácil integração entre a aplicação Node.js e o MySQL, além de garantir persistência e confiabilidade através de volumes e healthchecks

- **Back-End**

- **Clientes:** cria uma API RESTful com Express.js para gerenciar clientes em um banco de dados. As funções CRUD (Create, Read, Update, Delete) são implementadas para buscar todos os clientes, buscar um cliente por ID, criar um novo cliente, atualizar os dados de um cliente existente e deletar um cliente. As rotas são configuradas com o Express para mapear as funções aos métodos HTTP apropriados: GET / para buscar todos, GET /:id para buscar por ID, POST / para criar, PUT /:id para atualizar e DELETE /:id para deletar. O código também inclui tratamento de erros e retorna respostas em formato JSON.
- **Pedidos:** define um conjunto de funções para gerenciar pedidos em um sistema utilizando Express.js e MySQL. Ele inclui: getAllPedidos, que busca todos os pedidos e os dados relacionados de clientes, vendedores e produtos; createPedido, que cria um novo pedido no banco de dados; updatePedido, que atualiza um pedido existente; e deletePedido, que deleta um pedido específico. As funções estão associadas a rotas no Express para permitir manipulação de pedidos por meio de requisições GET, POST, PUT e DELETE. O código usa a conexão com o banco de dados para realizar as operações e retornar respostas apropriadas.
- **Produtos:** Define operações para gerenciar produtos em um sistema usando Express.js e MySQL. Ele possui funções para buscar todos os produtos, buscar um produto por ID, criar, atualizar e deletar produtos, e realizar consultas com filtros como nome, categoria, marca e fornecedor. As consultas são feitas utilizando uma conexão com o banco de dados pool. As rotas são configuradas no Express para lidar com requisições HTTP como GET, POST, PUT e DELETE. As respostas incluem os dados dos produtos ou mensagens de erro, dependendo do sucesso ou falha das operações.
- **Relatório:** Este código define um controlador para gerar relatórios sobre produtos e clientes utilizando Express.js e MySQL. Ele possui quatro funções: produtosMaisVendidos, que retorna os produtos mais vendidos baseados nas movimentações de saída; produtosBaixoEstoque, que lista os produtos com estoque abaixo ou igual a 10 unidades; consumoMedioCliente, que calcula o consumo médio de cada cliente com base nas quantidades compradas; e produtoPorCliente, que exibe quais produtos foram comprados por cada cliente. Cada função executa uma consulta SQL e retorna os resultados ou uma mensagem de erro. As rotas são configuradas para permitir acesso a esses relatórios.
- **Vendedores:** Este código define funções para gerenciar vendedores usando Express.js e MySQL. As funções incluem buscar todos os vendedores, buscar um

vendedor por ID, criar um novo vendedor, atualizar um vendedor existente e deletar um vendedor. Cada função realiza operações no banco de dados e retorna resultados em formato JSON ou mensagens de erro conforme necessário. As funções são mapeadas para rotas HTTP (GET, POST, PUT, DELETE) no Express.

Considerações Finais

O trabalho desenvolvido para a disciplina de Sistemas Distribuídos e Móbiles sob a orientação dos professores Adailton de Jesus Cerqueira Junior e Marivaldo Pereira dos Santos, teve como objetivo a construção de uma aplicação capaz de simular a captação de dados de vendas em uma rede de lojas. O grupo optou por utilizar a linguagem HTML e Javascript para o desenvolvimento da aplicação. Tal aplicação essa foi projetada para simular a captura dos dados de vendas de uma loja e fluxos de dados distribuídos. Durante a execução do projeto, foi aplicado diversos conceitos aprendidos na disciplina, como comunicação entre servidores e clientes coletando os dados em tempo real. Por isso essa experiência foi enriquecedora por proporcionar um desafio que aplica aspectos técnicos e práticos dos sistemas distribuídos em um contexto real de negócios. Por fim, agradecimentos aos professores Adailton de Jesus Cerqueira Junior e Marivaldo Pereira dos Santos pela orientação e pelos conhecimentos compartilhados ao longo da disciplina. Além do mais, o projeto permitiu uma visão mais profunda sobre a importância da integração de sistemas e do processamento de dados de forma distribuída, aspectos fundamentais em um mundo cada vez mais conectado.

Bibliografia

DOCKER. *MySQL Official Image*. Disponível em: https://hub.docker.com/_/mysql. Acesso em: 6 dez. 2024.

YOUTUBE. *JavaScript Crud Application - Crud Operations With JavaScript*. Disponível em: https://www.youtube.com/watch?v=sDPw2Yp4JwE. Acesso em: 6 dez. 2024.

YOUTUBE. *Build a Crud API with Docker, Node.js, Express.js & PostgreSQL*. Disponível em: https://www.youtube.com. Acesso em: 6 dez. 2024.

YOUTUBE. *Build a Rest API with Node.js and Express*. Disponível em: https://www.youtube.com/watch?v=l8WPWK9mS5M. Acesso em: 6 dez. 2024.

YOUTUBE. *Node.js MySQL Crud - Get, Post, Put and Delete*. Disponível em: https://www.youtube.com/watch?v=YkBOkV0s5eQ. Acesso em: 6 dez. 2024.

YOUTUBE. *JavaScript Crud Application - Crud Operations With JavaScript*. Disponível em: https://www.youtube.com/watch?v=KiRKUTDYlG8. Acesso em: 6 dez. 2024.