Sistema de Compras - Versão Piloto

Trabalho Prático – Sistema de Compras WEB - Desktop

Objetivo Geral

Desenvolver uma aplicação web fullstack com **Node.js**, **Sequelize**, **MySQL**, e **HTML/CSS/JavaScript**, com foco em consumo de dados externos, estruturação de banco de dados relacional e execução de operações de cadastro (CRUD), consultas e relatórios gerenciais.

Descrição da Aplicação

O sistema deve consumir dados das APIs públicas abaixo, utilizando **`fetch`**:

- **Produtos**: https://dummyjson.com/products
- **Usuários**: https://dummyjson.com/users

Esses dados deverão ser armazenados no banco de dados relacional **MySQL**, nas tabelas `usuários` e `produtos` `com nomes em português conforme o modelo de dados`.

Através da tabela intermediária de compras, o sistema deve registrar as transações entre usuários e produtos.

As tabelas de usuários e compras devem ser “populadas” a partir da API DummyJson a partir do comando fetch e posteriormente as tabelas são “populadas” pela operação do sistema pelos usuários do sistema.

As configurações sensíveis do sistema (como dados de acesso ao banco de dados) devem ser gerenciadas por meio de variáveis de ambiente, utilizando a biblioteca `dotenv.`

Exemplos típicos de variáveis:
No `.env`:

```
DB_HOST=localhost

DB_USER=root

DB_PASS=senai

DB_NAME=compras_db

PORT=3000
```

Estrutura do Banco de Dados

1. Tabela `usuarios`

Contém os seguintes campos:

- ID (id)
- Primeiro nome (firstName)
- Sobrenome (lastName)
- Idade (age)
- E-mail (email)
- Telefone (phone)
- Endereço (address)
- Cidade (city)
- Estado (state)
- Data de nascimento (birthDate)

2. Tabela `produtos`

Contém os seguintes campos:

- ID (id)
- Título (title)
- Descrição (description)
- Categoria (category)
- Preço (price)
- Percentual de desconto (discountPercentage)
- Estoque (stock)
- Marca (brand)

- Imagem (thumbnail) - URL da imagem

3. Tabela `compras`

Responsável por registrar cada compra realizada por usuários, com os seguintes campos:

- ID da compra (chave primária)
- ID do usuário (chave estrangeira)
- ID do produto (chave estrangeira)
- Quantidade
- Data da compra
- Preço unitário
- Desconto aplicado (%)
- Preço final (calculado)
- Forma de pagamento
- Status da compra

Funcionalidades Obrigatórias

Operações CRUD (Criar, Ler, Atualizar e Deletar) para:

- **Usuários**
- **Produtos**
- **Compras**

Cada entidade deve permitir:

- **Cadastro** de novos registros
- **Listagem** de todos os registros
- **Consulta por ID** (usuário ou produto)

- **Consulta por nome** (usuário ou produto)
- **Atualização** de registros existentes pelo código (id)
- **Remoção** de registros pelo código(id)

Gráficos com Chart.js

A aplicação deverá gerar dois gráficos com **Chart.js**: https://www.chartjs.org/

1. **Produto x Estoque**
    - Exibe os títulos dos produtos e a quantidade em estoque
2. **Usuário x Idade**
    - Exibe o nome completo dos usuários e suas idades

Filtro:

- O usuário poderá selecionar um **intervalo de ID (inicial e final)** para ambos os gráficos
- Os gráficos devem exibir **até 10 itens** no máximo

Relatórios Gerenciais

O sistema deverá exibir **5 relatórios gerenciais** em **tabelas HTML**, com foco em dados relevantes para um gestor.

1. Relatório de Usuários

- Listagem de todos os usuários cadastrados, com nome completo, idade, e-mail, cidade e estado.

2. Relatório de Produtos

- Listagem dos produtos com título, categoria, preço original, desconto aplicado e valor final estimado.

3. Relatório de Compras

- Listagem de todas as compras com ID, nome do usuário, nome do produto, quantidade, data e preço final.

4. Relatório de Estoque Crítico

- Produtos com estoque inferior a 10 unidades. Mostra título, estoque e categoria.

5. Relatório Consolidado

- Relatório completo com nome do usuário, nome do produto comprado, quantidade, data da compra, valor final, forma de pagamento e status.

Tecnologias e Ferramentas

- **Backend:** Node.js, Express, Sequelize, CORS
- **Banco de Dados:** MySQL
- **Frontend:** HTML, CSS, JavaScript
- **Gráficos:** Chart.js
- **Consumo de APIs:** `fetch`
- Uso da biblioteca **dotenv** para variáveis de ambiente

Entrega Esperada

- Projeto organizado por camadas usando a arquitetura MVC;
- Diagrama de caso de uso geral em UML;
- Diagrama de Classe UML com os relacionamentos;
- Diagrama de sequência do cadastro do produto, listagem do usuario
- Diagrama Lógico do Banco de Dados gerado a partir da Engenharia Reversa;
- Script de criação do banco de dados ou dump `.sql em arquivo único`
- Código-fonte funcional e comentado
- link do github do projeto, versionado. Pelo menos 7 versões;
- link do vercel do front end publicado;
- O sistema do backend deve conversar com o front end do vercel em máquina local
- Instruções de instalação e execução no **`README.md`**

Apresentação

A apresentação deve ser feita com o sistema funcionando (vídeo de até 7 minutos)