Objetivo: Avaliar o conhecimento técnico em React JS, Tailwind CSS, Node JS, TypeORM e SQL, além da capacidade de aplicar os princípios de Orientação a Objetos, Clean Code e testes unitários na construção de uma aplicação web funcional e bem estruturada. Também será avaliada a atenção aos detalhes de código, interface do usuário e as decisões de arquitetura tomadas.

Tema: Gestor Financeiro Pessoal

Descrição do Desafio:

Você deverá desenvolver uma aplicação web para gerenciar finanças pessoais. A aplicação deve permitir o cadastro de contas bancárias e o registro de transações financeiras entre essas contas.

Conhecimentos necessários:

- **Frontend:**
    - React JS (desenvolvimento da interface do usuário, organização de componentes, validações básicas).
    - Tailwind CSS (estilização responsiva e visualmente agradável).
- **Backend:**
    - Node JS com Express (desenvolvimento da API).
    - TypeORM (mapeamento objeto-relacional e interação com banco de dados SQL).
    - SQL (SQLite será suficiente).
    - Princípios de Orientação a Objetos (estrutura do código do backend).
- **Geral:**
    - Princípios de Clean Code (legibilidade, manutenção e organização do código).
    - Testes Unitários (garantia da qualidade do código).
    - Sistema de Controle de Versão (Git).
- **Conhecimentos Funcionais:**
    - Cadastro e gerenciamento de contas bancárias.
    - Registro e gerenciamento de transações financeiras (débito, crédito, transferência).
    - Lógica para transferências entre contas.
- **Documentação:**
    - Habilidade de documentar decisões de arquitetura, lógica de desenvolvimento, estrutura do projeto e instruções de execução.
    - Habilidade de explicar a cobertura de testes

Requisitos Funcionais Mínimos:

1. **Cadastro de Contas:**

    - Permitir a criação de novas contas bancárias, com os seguintes atributos:
        - Nome da Conta (ex: Conta Corrente Banco X, Cartão de Crédito Y)
        - Tipo da Conta (ex: Corrente, Poupança, Crédito, Investimento)

        - Saldo Inicial (opcional)
    - Listar as contas cadastradas, exibindo seus nomes, tipos e saldos atuais.
    - Permitir a edição e exclusão de contas existentes.
2. **Registro de Transações:**

    - Permitir o registro de novas transações financeiras, com os seguintes atributos:
        - Tipo da Transação (Débito ou Crédito)
        - Conta de Origem (para Débito e Transferência)
        - Conta de Destino (para Crédito e Transferência)
        - Valor
        - Descrição (opcional)
        - Data da Transação
    - Listar as transações registradas, exibindo seus tipos, contas envolvidas, valores, descrições e datas.
    - Filtrar as transações por conta e período (opcional).
3. **Transferências entre Contas:**

    - Permitir a realização de transferências de valores entre contas cadastradas. Ao realizar uma transferência, o saldo da conta de origem deve ser debitado e o saldo da conta de destino deve ser creditado.

Requisitos Técnicos:

- **Frontend:**

    - Desenvolver a interface do usuário utilizando **React JS**.
    - Aplicar estilos utilizando **Tailwind CSS** para garantir uma interface responsiva e visualmente agradável.
    - Organizar os componentes de forma clara e reutilizável.
    - Implementar validações básicas nos formulários.
- **Backend:**

    - Desenvolver a API utilizando **Node JS** com **Express**.
    - Utilizar **TypeORM** para realizar o mapeamento objeto-relacional e interagir com o banco de dados **SQL** (SQLite será suficiente para este desafio, facilitando a configuração).
    - Implementar os endpoints necessários para atender aos requisitos funcionais do frontend.
    - Aplicar os princípios de **Orientação a Objetos** na estruturação do código do backend.
- **Geral:**

    - Aplicar os princípios de **Clean Code** em todo o projeto, tornando o código legível, manutenível e bem organizado (nomes de variáveis e funções descritivos, funções pequenas, etc.).

    - Implementar **testes unitários** para as principais funcionalidades do backend (modelos, serviços, controladores). Demonstre como você garante a qualidade do seu código através dos testes.
    - Utilizar um sistema de controle de versão (Git) e fornecer o link para o repositório ao final do desafio.

Entrega:

A entrega do desafio deverá conter os seguintes itens:

1. **Código Fonte Completo:** O código completo do frontend e do backend, organizado em um repositório Git público (GitHub, GitLab, Bitbucket, etc.).
2. **Documentação:** Um documento (em formato Markdown dentro do repositório ou um arquivo separado) contendo:
    - **Decisões de Arquitetura:** Explicações detalhadas sobre as principais decisões de arquitetura tomadas durante o desenvolvimento. Justifique as escolhas de bibliotecas e tecnologias (por exemplo, "Utilizei TypeORM pois ele oferece um bom suporte a TypeScript e facilita o gerenciamento do banco de dados relacional, diferente de outras opções como Sequelize que...").
    - **Lógica de Desenvolvimento:** Explicações sobre a lógica implementada nas principais funcionalidades, especialmente nas operações de transferência entre contas e no tratamento de erros.
    - **Estrutura do Projeto:** Uma breve descrição da estrutura de pastas e arquivos do projeto, tanto no frontend quanto no backend.
    - **Instruções de Execução:** Passo a passo detalhado de como executar o frontend e o backend da aplicação em um ambiente de desenvolvimento local.
    - **Cobertura de Testes:** Uma breve descrição de quais partes do código foram testadas e como os testes foram implementados.

Critérios de Avaliação:

Serão avaliados os seguintes aspectos:

- **Funcionalidade:** Cumprimento dos requisitos funcionais mínimos.
- **Conhecimento Técnico:** Demonstração de conhecimento e aplicação das tecnologias solicitadas (React JS, Tailwind CSS, Node JS, TypeORM, SQL).
- **Orientação a Objetos:** Aplicação dos princípios de orientação a objetos na estrutura do backend.
- **Clean Code:** Qualidade e organização do código (legibilidade, nomes significativos, modularização, etc.).
- **Testes Unitários:** Implementação e abrangência dos testes unitários.
- **Interface do Usuário:** Usabilidade, responsividade e atenção aos detalhes visuais da interface.
- **Atenção aos Detalhes:** Cuidado na implementação das funcionalidades e na apresentação da interface.
- **Lógica de Desenvolvimento:** Clareza e correção da lógica implementada.
- **Documentação:** Qualidade e clareza da documentação, incluindo as justificativas das decisões de arquitetura e da lógica de desenvolvimento.