Projeto Final - Lógica de Programação com JS

Tema: Internet Banking (FutureBank)
Prazo de Entrega: 30/01
Formato: Individual ou Dupla

1. O Desafio

Você foi contratado por uma Fintech para desenvolver a lógica do **FutureBank**, um painel financeiro onde o usuário pode ver seu saldo, realizar transações e consultar seu extrato detalhado.
O Front-end (HTML e CSS) já foi desenhado pela equipe de design. Sua missão é **dar vida** ao sistema usando JavaScript.

2. Requisitos Técnicos (O que será avaliado)

Seu código deve obrigatoriamente conter:

1. **Variáveis e Constantes:** Para armazenar estado (saldo atual, nome do usuário).
2. **Objetos e Arrays:**
    - Um Objeto para representar o usuário.
    - Um Array de Objetos para armazenar o histórico de transações (o extrato).
3. **Funções:**
    - Funções com parâmetros (ex: depositar(valor)).
    - Funções com retorno.
4. **Laços de Repetição (Loops):**
    - Uso de for, for...of ou forEach para renderizar a tabela de extrato na tela.
5. **Condicionais e Operadores Lógicos:** Para validar se o usuário tem saldo suficiente, se a senha está correta, etc.
6. **Tratamento de Erros:** Uso de try/catch para impedir operações ilegais (ex: sacar valor negativo).
7. **Manipulação do DOM:** Atualizar o saldo e a lista de extrato na tela sem recarregar a página.

3. Funcionalidades do Sistema

A. Painel Principal (Dashboard)

- Exibir o nome do usuário e o saldo atualizado.
- Botões para realizar operações: Depositar, Sacar, Pagar Boleto.

B. Extrato (Histórico)

- Deve listar todas as operações realizadas.
- Cada linha do extrato deve ter: Data, Descrição, Valor e Tipo (Entrada/Saída).
- **Desafio Extra:** Colorir de verde as entradas e vermelho as saídas.

C. Funcionalidades de Lógica

1. **Depositar:** Soma valor ao saldo e adiciona registro no extrato.
2. **Sacar/Pagar:** Subtrai valor do saldo e adiciona registro no extrato.
    - *Regra:* Não permitir se o saldo for insuficiente.
3. **Filtrar Extrato (Opcional):** Um botão para ver "Tudo", apenas "Entradas" ou apenas "Saídas".

4. Código Base (Starter Kit)

Para começar, crie um arquivo index.html e cole o código abaixo. Ele já contém a estrutura visual e os IDs necessários para você manipular com JS.

O HTML + CSS (Copie e Cole)

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>FutureBank - Seu Banco Digital</title>
    <style>
        :root { --primary: #6c5ce7; --bg: #dfe6e9; --text: #2d3436; }
        body { font-family: 'Segoe UI', sans-serif; background-color: var(--bg); color: var(--text); display: flex; justify-content: center; padding: 20px; }
        .app-container { width: 100%; max-width: 800px; background: white; border-radius: 15px; box-shadow: 0 10px 30px rgba(0,0,0,0.1); overflow: hidden; }

        /* Header */
        header { background-color: var(--primary); color: white; padding: 20px; display: flex; justify-content: space-between; align-items: center; }
        .saldo-box { text-align: right; }
        .saldo-valor { font-size: 2rem; font-weight: bold; }

        /* Actions */
        .actions { padding: 20px; display: flex; gap: 10px; border-bottom: 1px solid #eee; }
        button { flex: 1; padding: 15px; border: none; border-radius: 8px; font-weight: bold; cursor: pointer; transition: 0.2s; color: white; }
        .btn-deposit { background-color: #00b894; }
```

```
        .btn-pay { background-color: #d63031; }
        button:hover { opacity: 0.9; transform: translateY(-2px); }

        /* Extract Area */
        .extract { padding: 20px; }
        .extract-header { display: flex; justify-content: space-between; align-items: center;
margin-bottom: 15px; }

        table { width: 100%; border-collapse: collapse; }
        th, td { text-align: left; padding: 12px; border-bottom: 1px solid #eee; }
        th { color: #636e72; font-size: 0.9rem; }

        .entrada { color: #00b894; font-weight: bold; }
        .saida { color: #d63031; font-weight: bold; }
    </style>
</head>
<body>

    <div class="app-container">
        <header>
            <div>
                <small>Olá,</small>
                <h2 id="nome-usuario">Carregando...</h2>
            </div>
            <div class="saldo-box">
                <small>Saldo Atual</small>
                <div class="saldo-valor">R$ <span id="saldo">0,00</span></div>
            </div>
        </header>

        <section class="actions">
            <button class="btn-deposit" onclick="novaTransacao('deposito')">💰
Depositar</button>
            <button class="btn-pay" onclick="novaTransacao('pagamento')">💸 Pagar
Boleto</button>
        </section>

        <section class="extract">
            <div class="extract-header">
                <h3>Extrato Recente</h3>
                <!-- Desafio: Fazer estes filtros funcionarem -->
                <select id="filtro-tipo" onchange="filtrarExtrato()">
                    <option value="todos">Todos</option>
```

```
            <option value="entrada">Entradas</option>
            <option value="saida">Saídas</option>
          </select>
        </div>

        <table>
          <thead>
            <tr>
              <th>Data</th>
              <th>Descrição</th>
              <th>Valor</th>
            </tr>
          </thead>
          <tbody id="lista-transacoes">
            <!-- As linhas serão inseridas aqui via JS -->
          </tbody>
        </table>
      </section>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

5. Roteiro de Desenvolvimento (Sugestão)

Não tente fazer tudo de uma vez. Siga este roteiro conforme as aulas forem acontecendo:

Fase 1: Estrutura de Dados (Já pode começar!)

- No arquivo script.js, crie um Objeto para o usuário (com nome e saldo inicial).
- Crie um Array de Objetos chamado transacoes para guardar o histórico.
- Exiba o nome e o saldo inicial na tela assim que a página carregar.

Fase 2: Funções e Operações (Próximas aulas)

- Crie uma função atualizarSaldo() que recalcula o total somando/subtraindo o array de transações.
- Crie a função novaTransacao(tipo) que:
    1. Pede o valor ao usuário (prompt).
    2. Pede a descrição.
    3. Valida os dados (não pode ser negativo, não pode ser texto).
    4. Adiciona o novo objeto ao array transacoes.

Fase 3: Renderização e Loops (Aula de Laços)

- Crie a função renderizarExtrato().
- Ela deve limpar o HTML da tabela (innerHTML = "").
- Deve usar um laço (for ou for...of) para percorrer o array transacoes.
- Para cada item, cria uma <tr> (linha da tabela) e joga na tela.

Fase 4: Proteção e Filtros (Aulas Finais)

- Use try/catch para tratar erros (ex: tentar pagar uma conta maior que o saldo).
- Implemente a lógica do <select> para filtrar o laço de repetição e mostrar apenas o que o usuário pediu.

Boa sorte! Transforme lógica em dinheiro (virtual)! 💸

Prazo para entrega (30/01 23:59)

Formas de entrega:

- Arquivo ou link do Github no LMS.
- Arquivo ou link do Github pelo WhatsApp

  Darei os feedbacks até 07/02