<!DOCTYPE html>

<html lang="pt-BR">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Controle Financeiro Pessoal Completo</title>

<script src="https://cdn.tailwindcss.com"></script>

<script src="https://unpkg.com/xlsx/dist/xlsx.full.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<style>

/\* Define a fonte Inter para toda a página \*/

body {

font-family: 'Inter', sans-serif;

background-color: #f0f4f8; /\* Cor de fundo suave \*/

}

/\* Estilos personalizados para o modal de confirmação \*/

.modal {

display: none; /\* Escondido por padrão \*/

position: fixed; /\* Posição fixa na tela \*/

z-index: 1000; /\* Acima de outros elementos \*/

left: 0;

top: 0;

width: 100%;

height: 100%;

overflow: auto; /\* Habilita rolagem se o conteúdo for muito grande \*/

background-color: rgba(0,0,0,0.4); /\* Fundo semi-transparente \*/

display: flex; /\* Usa flexbox para centralizar \*/

justify-content: center; /\* Centraliza horizontalmente \*/

align-items: center; /\* Centraliza verticalmente \*/

}

.modal-content {

background-color: #ffffff;

margin: auto;

padding: 24px;

border-radius: 12px; /\* Cantos arredondados \*/

box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1); /\* Sombra suave \*/

width: 90%;

max-width: 400px;

text-align: center;

}

.modal-buttons {

display: flex;

justify-content: space-around;

margin-top: 24px;

}

.modal-buttons button {

padding: 10px 20px;

border-radius: 8px;

cursor: pointer;

font-weight: 600;

transition: background-color 0.3s ease;

}

.modal-buttons .confirm-btn {

background-color: #ef4444; /\* Vermelho para confirmar \*/

color: white;

border: none;

}

.modal-buttons .confirm-btn:hover {

background-color: #dc2626;

}

.modal-buttons .cancel-btn {

background-color: #e2e8f0; /\* Cinza para cancelar \*/

color: #334155;

border: 1px solid #cbd5e1;

}

.modal-buttons .cancel-btn:hover {

background-color: #cbd5e1;

}

/\* Estilos para a mensagem de feedback \*/

#feedback-message {

display: none;

padding: 12px;

margin-top: 16px;

border-radius: 8px;

font-weight: 500;

}

.feedback-success {

background-color: #d1fae5;

color: #065f46;

}

.feedback-error {

background-color: #fee2e2;

color: #991b1b;

}

.feedback-info {

background-color: #e0f2fe;

color: #075985;

}

/\* Estilos para ocultar/mostrar elementos \*/

.hidden {

display: none !important;

}

/\* Estilos para a seção colapsável \*/

.collapsible-header {

cursor: pointer;

display: flex;

justify-content: space-between;

align-items: center;

padding: 1rem 0;

border-bottom: 1px solid #e2e8f0;

margin-bottom: 1rem;

}

.collapsible-header:hover {

background-color: #f8fafc;

}

.collapsible-header h2 {

margin-bottom: 0;

}

.arrow-icon {

transition: transform 0.3s ease;

}

.arrow-icon.rotated {

transform: rotate(90deg);

}

/\* Estilo para transações pagas \*/

.paid-transaction {

text-decoration: line-through;

color: #6b7280; /\* gray-500 \*/

}

</style>

</head>

<body class="p-4 sm:p-6 md:p-8 lg:p-10">

<div class="max-w-5xl mx-auto bg-white rounded-xl shadow-lg p-6 sm:p-8">

<h1 class="text-3xl sm:text-4xl font-bold text-center text-gray-800 mb-8">Controle Financeiro Pessoal Completo</h1>

<div class="grid grid-cols-1 md:grid-cols-3 gap-4 mb-8">

<div class="bg-blue-100 p-4 rounded-lg shadow-md">

<h2 class="text-lg font-semibold text-blue-800">Total de Receitas</h2>

<p id="total-receitas" class="text-2xl font-bold text-blue-700">R$ 0,00</p>

</div>

<div class="bg-red-100 p-4 rounded-lg shadow-md">

<h2 class="text-lg font-semibold text-red-800">Total de Despesas</h2>

<p id="total-despesas" class="text-2xl font-bold text-red-700">R$ 0,00</p>

</div>

<div class="bg-green-100 p-4 rounded-lg shadow-md">

<h2 class="text-lg font-semibold text-green-800">Saldo Atual</h2>

<p id="saldo-atual" class="text-2xl font-bold text-green-700">R$ 0,00</p>

</div>

</div>

<div class="mb-8 border-b pb-6 border-gray-200">

<h2 class="text-2xl font-semibold text-gray-700 mb-4">Previsão de Despesas Próximo Mês</h2>

<div class="bg-yellow-50 p-4 rounded-lg shadow-md flex justify-between items-center">

<h3 class="text-lg font-semibold text-yellow-800">Total Previsto:</h3>

<p id="predicted-expenses" class="text-2xl font-bold text-yellow-700">R$ 0,00</p>

</div>

</div>

<div class="mb-8 border-b pb-6 border-gray-200">

<h2 class="text-2xl font-semibold text-gray-700 mb-4">Resumo Mensal</h2>

<div class="flex items-center space-x-4 mb-4">

<button id="prev-month-btn" class="px-4 py-2 bg-gray-200 rounded-md hover:bg-gray-300">Anterior</button>

<span id="current-month-display" class="text-lg font-semibold text-gray-800"></span>

<button id="next-month-btn" class="px-4 py-2 bg-gray-200 rounded-md hover:bg-gray-300">Próximo</button>

</div>

<div class="grid grid-cols-1 md:grid-cols-3 gap-4">

<div class="bg-blue-50 p-4 rounded-lg shadow-sm">

<h3 class="text-lg font-semibold text-blue-700">Receitas do Mês</h3>

<p id="monthly-income" class="text-xl font-bold text-blue-600">R$ 0,00</p>

</div>

<div class="bg-red-50 p-4 rounded-lg shadow-sm">

<h3 class="text-lg font-semibold text-red-700">Despesas do Mês</h3>

<p id="monthly-expenses" class="text-xl font-bold text-red-600">R$ 0,00</p>

</div>

<div class="bg-green-50 p-4 rounded-lg shadow-sm">

<h3 class="text-lg font-semibold text-green-700">Saldo do Mês</h3>

<p id="monthly-balance" class="text-xl font-bold text-green-600">R$ 0,00</p>

</div>

</div>

</div>

<div class="mb-8 border-b pb-6 border-gray-200">

<h2 class="text-2xl font-semibold text-gray-700 mb-4">Gerenciar Contas Bancárias / Dinheiro</h2>

<div class="p-4 bg-gray-50 rounded-lg shadow-sm mb-6">

<h3 class="text-xl font-semibold text-gray-700 mb-3">Adicionar Conta</h3>

<form id="add-account-form" class="flex flex-col sm:flex-row gap-3">

<input type="text" id="account-name" placeholder="Nome da Conta (ex: Banco X, Dinheiro)" required

class="flex-grow px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm">

<select id="account-type"

class="px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm">

<option value="bank">Conta Bancária</option>

<option value="cash">Dinheiro</option>

</select>

<input type="number" id="initial-balance" step="0.01" placeholder="Saldo Inicial (R$)" required

class="w-32 px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm">

<input type="number" id="overdraft-limit" step="0.01" placeholder="Cheque Especial (R$)"

class="w-32 px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm hidden">

<button type="submit"

class="px-5 py-2 bg-blue-500 text-white font-semibold rounded-md shadow-md hover:bg-blue-600 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-400 transition duration-150 ease-in-out">

Adicionar

</button>

</form>

</div>

<div>

<h3 class="text-lg font-semibold text-gray-700 mb-2">Minhas Contas</h3>

<ul id="accounts-list" class="bg-white rounded-md shadow-sm divide-y divide-gray-200">

</ul>

</div>

</div>

<div class="mb-8 border-b pb-6 border-gray-200">

<h2 class="text-2xl font-semibold text-gray-700 mb-4">Gerenciar Cartões de Crédito</h2>

<div class="p-4 bg-gray-50 rounded-lg shadow-sm mb-6">

<h3 class="text-xl font-semibold text-gray-700 mb-3">Adicionar Cartão de Crédito</h3>

<form id="add-credit-card-form" class="flex flex-col sm:flex-row gap-3">

<input type="text" id="card-name" placeholder="Nome do Cartão (ex: Visa, Master)" required

class="flex-grow px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-purple-500 focus:border-purple-500 sm:text-sm">

<input type="number" id="card-limit" step="0.01" placeholder="Limite (R$)" required

class="w-28 px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-purple-500 focus:border-purple-500 sm:text-sm">

<input type="number" id="card-due-day" min="1" max="31" placeholder="Dia Venc. Fatura" required

class="w-32 px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-purple-500 focus:border-purple-500 sm:text-sm">

<button type="submit"

class="px-5 py-2 bg-purple-500 text-white font-semibold rounded-md shadow-md hover:bg-purple-600 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-purple-400 transition duration-150 ease-in-out">

Adicionar

</button>

</form>

</div>

<div>

<h3 class="text-lg font-semibold text-gray-700 mb-2">Meus Cartões</h3>

<ul id="credit-cards-list" class="bg-white rounded-md shadow-sm divide-y divide-gray-200">

</ul>

</div>

</div>

<div class="mb-8 border-b pb-6 border-gray-200">

<h2 class="text-2xl font-semibold text-gray-700 mb-4">Adicionar Nova Transação Manualmente</h2>

<form id="transaction-form" class="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">

<div>

<label for="date" class="block text-sm font-medium text-gray-700 mb-1">Data</label>

<input type="date" id="date" required

class="mt-1 block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm">

</div>

<div>

<label for="description" class="block text-sm font-medium text-gray-700 mb-1">Descrição</label>

<input type="text" id="description" placeholder="Ex: Salário, Aluguel, Supermercado" required

class="mt-1 block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm">

</div>

<div>

<label for="value" class="block text-sm font-medium text-gray-700 mb-1">Valor (R$)</label>

<input type="number" id="value" step="0.01" placeholder="0.00" required

class="mt-1 block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm">

</div>

<div>

<label for="type" class="block text-sm font-medium text-gray-700 mb-1">Tipo</label>

<select id="type" required

class="mt-1 block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm">

<option value="receita">Receita</option>

<option value="despesa">Despesa</option>

</select>

</div>

<div>

<label for="category" class="block text-sm font-medium text-gray-700 mb-1">Categoria</label>

<input type="text" id="category" placeholder="Ex: Salário, Moradia, Alimentação" required

class="mt-1 block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm">

</div>

<div>

<label for="account-select" class="block text-sm font-medium text-gray-700 mb-1">Conta / Cartão</label>

<select id="account-select" required

class="mt-1 block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm">

<option value="">Selecione uma conta</option>

</select>

</div>

<div class="col-span-1 sm:col-span-2 lg:col-span-1">

<label for="installments" class="block text-sm font-medium text-gray-700 mb-1">Parcelas (total)</label>

<input type="number" id="installments" min="1" value="1"

class="mt-1 block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm">

</div>

<div class="col-span-1 sm:col-span-2 lg:col-span-1">

<label for="installments-paid" class="block text-sm font-medium text-gray-700 mb-1">Parcelas já pagas</label>

<input type="number" id="installments-paid" min="0" value="0"

class="mt-1 block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm">

</div>

<div class="sm:col-span-2 lg:col-span-full flex items-end justify-end">

<button type="submit"

class="w-full sm:w-auto px-6 py-2 bg-blue-600 text-white font-semibold rounded-md shadow-md hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500 transition duration-150 ease-in-out">

Adicionar Transação

</button>

</div>

</form>

</div>

<div class="mb-8 border-b pb-6 border-gray-200">

<div id="import-header" class="collapsible-header">

<h2 class="text-2xl font-semibold text-gray-700">Importar Extratos</h2>

<svg id="import-arrow" class="arrow-icon w-6 h-6 text-gray-600" fill="none" stroke="currentColor" viewBox="0 0 24 24" xmlns="http://www.w3.org/2000/svg">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M9 5l7 7-7 7"></path>

</svg>

</div>

<div id="import-section-content" class="hidden">

<p class="text-sm text-gray-600 mb-4">

Seu arquivo deve ser OFX, CSV ou Excel e conter informações de transação com Data, Descrição, Valor e Tipo.

<span class="font-bold text-red-600">Atenção: Integração direta com bancos (Open Finance) não é possível em uma aplicação web no navegador por motivos de segurança e privacidade.</span>

<span class="font-bold text-red-600">Importação por imagem não é suportada (requer OCR e processamento de servidor).</span>

</p>

<div class="p-4 bg-gray-50 rounded-lg shadow-sm">

<h3 class="text-xl font-semibold text-gray-700 mb-3">Carregar Arquivo de Extrato</h3>

<div class="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4 items-end">

<div>

<label for="file-input" class="block text-sm font-medium text-gray-700 mb-1">Selecione o Arquivo</label>

<input type="file" id="file-input" accept=".ofx, .csv, .xls, .xlsx"

class="block w-full text-sm text-gray-500 file:mr-4 file:py-2 file:px-4 file:rounded-md file:border-0 file:text-sm file:font-semibold file:bg-gray-100 file:text-gray-700 hover:file:bg-gray-200 cursor-pointer">

</div>

<div class="flex justify-end col-span-full sm:col-span-1">

<button id="import-file-btn"

class="w-full sm:w-auto px-6 py-2 bg-indigo-600 text-white font-semibold rounded-md shadow-md hover:bg-indigo-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500 transition duration-150 ease-in-out">

Carregar para Revisão

</button>

</div>

</div>

</div>

</div>

<div id="feedback-message" class="hidden"></div>

</div>

<div id="import-review-section" class="mb-8 border-b pb-6 border-gray-200 hidden">

<h2 class="text-2xl font-semibold text-gray-700 mb-4">Revisar Transações Importadas</h2>

<p class="text-sm text-gray-600 mb-4">

Vincule cada transação importada a uma de suas contas ou cartões.

</p>

<div class="overflow-x-auto rounded-lg shadow-md mb-4">

<table class="min-w-full divide-y divide-gray-200">

<thead class="bg-yellow-50">

<tr>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-700 uppercase tracking-wider">Data</th>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-700 uppercase tracking-wider">Descrição</th>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-700 uppercase tracking-wider">Valor</th>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-700 uppercase tracking-wider">Tipo</th>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-700 uppercase tracking-wider">Categoria</th>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-700 uppercase tracking-wider">Vincular a</th>

</tr>

</thead>

<tbody id="imported-transactions-table-body" class="bg-white divide-y divide-gray-200">

</tbody>

</table>

</div>

<div class="flex justify-end gap-4">

<button id="cancel-review-btn"

class="px-6 py-2 bg-gray-400 text-white font-semibold rounded-md shadow-md hover:bg-gray-500 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-gray-300 transition duration-150 ease-in-out">

Cancelar Importação

</button>

<button id="confirm-import-btn"

class="px-6 py-2 bg-blue-600 text-white font-semibold rounded-md shadow-md hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500 transition duration-150 ease-in-out">

Confirmar Importação

</button>

</div>

</div>

<div class="mb-8 border-b pb-6 border-gray-200">

<h2 class="text-2xl font-semibold text-gray-700 mb-4">Dashboard de Visualização (Despesas por Categoria)</h2>

<div class="bg-white p-4 rounded-lg shadow-md">

<canvas id="expenses-chart"></canvas>

</div>

</div>

<div class="mb-8 border-b pb-6 border-gray-200">

<h2 class="text-2xl font-semibold text-gray-700 mb-4">Baixar Planilha</h2>

<button id="download-xlsx-btn"

class="w-full sm:w-auto px-6 py-2 bg-gray-600 text-white font-semibold rounded-md shadow-md hover:bg-gray-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-gray-500 transition duration-150 ease-in-out">

Baixar Transações do Mês como Excel (.xlsx)

</button>

</div>

<div>

<h2 class="text-2xl font-semibold text-gray-700 mb-4">Minhas Transações</h2>

<div class="overflow-x-auto rounded-lg shadow-md">

<table class="min-w-full divide-y divide-gray-200">

<thead class="bg-gray-50">

<tr>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Data</th>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Descrição</th>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Valor</th>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Tipo</th>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Categoria</th>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Conta / Cartão</th>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Parcela</th>

<th scope="col" class="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Ações</th>

</tr>

</thead>

<tbody id="transaction-list" class="bg-white divide-y divide-gray-200">

</tbody>

</table>

</div>

<p id="no-transactions-message" class="text-center text-gray-500 mt-4 hidden">Nenhuma transação registrada ainda.</p>

</div>

</div>

<div id="confirmation-modal" class="modal">

<div class="modal-content">

<p class="text-lg font-semibold text-gray-800 mb-4">Tem certeza que deseja excluir esta transação?</p>

<div class="modal-buttons">

<button id="confirm-delete" class="confirm-btn">Sim, Excluir</button>

<button id="cancel-delete" class="cancel-btn">Cancelar</button>

</div>

</div>

</div>

<script>

// Global data arrays

let accounts = [];

let creditCards = [];

let transactions = [];

let importedTransactionsToReview = []; // Temporary array for transactions pending review

let transactionToDeleteInfo = null; // Stores info about what to delete (id, type)

let currentMonth = new Date().getMonth(); // 0-indexed (Jan = 0)

let currentYear = new Date().getFullYear();

// Chart.js instance

let expensesChart = null;

// DOM Elements

const totalReceitasEl = document.getElementById('total-receitas');

const totalDespesasEl = document.getElementById('total-despesas');

const saldoAtualEl = document.getElementById('saldo-atual');

const predictedExpensesEl = document.getElementById('predicted-expenses'); // New element for prediction

const noTransactionsMessage = document.getElementById('no-transactions-message');

const confirmationModal = document.getElementById('confirmation-modal');

const confirmDeleteBtn = document.getElementById('confirm-delete');

const cancelDeleteBtn = document.getElementById('cancel-delete');

const feedbackMessageEl = document.getElementById('feedback-message');

// Account & Credit Card Management

const addAccountForm = document.getElementById('add-account-form');

const accountsList = document.getElementById('accounts-list');

const addCreditCardForm = document.getElementById('add-credit-card-form');

const creditCardsList = document.getElementById('credit-cards-list');

const accountSelect = document.getElementById('account-select'); // For manual transactions

const accountTypeSelect = document.getElementById('account-type'); // For showing/hiding overdraft limit

const initialBalanceInput = document.getElementById('initial-balance'); // For initial balance

const overdraftLimitInput = document.getElementById('overdraft-limit'); // For overdraft limit

const cardDueDayInput = document.getElementById('card-due-day'); // For credit card due date

// Transaction Form

const transactionForm = document.getElementById('transaction-form');

const transactionList = document.getElementById('transaction-list');

const installmentsInput = document.getElementById('installments');

const installmentsPaidInput = document.getElementById('installments-paid'); // NEW: Paid installments input

const transactionTypeSelect = document.getElementById('type');

// Import Sections

const importHeader = document.getElementById('import-header'); // New: for collapsible header

const importSectionContent = document.getElementById('import-section-content'); // New: for collapsible content

const importArrow = document.getElementById('import-arrow'); // New: for arrow icon

const importReviewSection = document.getElementById('import-review-section');

const importedTransactionsTableBody = document.getElementById('imported-transactions-table-body');

const confirmImportBtn = document.getElementById('confirm-import-btn');

const cancelReviewBtn = document.getElementById('cancel-review-btn');

// Unified Import

const fileInput = document.getElementById('file-input');

const importFileBtn = document.getElementById('import-file-btn');

// Monthly Summary Elements

const prevMonthBtn = document.getElementById('prev-month-btn');

const nextMonthBtn = document.getElementById('next-month-btn');

const currentMonthDisplay = document.getElementById('current-month-display');

const monthlyIncomeEl = document.getElementById('monthly-income');

const monthlyExpensesEl = document.getElementById('monthly-expenses');

const monthlyBalanceEl = document.getElementById('monthly-balance');

// Download

const downloadXlsxBtn = document.getElementById('download-xlsx-btn'); // Renamed from downloadCsvBtn

// --- Utility Functions ---

// Function to show feedback messages to the user

const showFeedback = (message, type) => {

feedbackMessageEl.textContent = message;

feedbackMessageEl.classList.remove('hidden', 'feedback-success', 'feedback-error', 'feedback-info');

feedbackMessageEl.classList.add(`feedback-${type}`);

feedbackMessageEl.style.display = 'block'; // Ensure it's visible

setTimeout(() => {

feedbackMessageEl.style.display = 'none'; // Hide after 5 seconds

}, 5000);

};

// Function to format values to BRL currency

const formatCurrency = (value) => {

return new Intl.NumberFormat('pt-BR', {

style: 'currency',

currency: 'BRL'

}).format(value);

};

// Function to generate a unique ID

const generateUniqueId = () => '\_' + Math.random().toString(36).substr(2, 9);

// --- Data Persistence (localStorage) ---

// Function to load all data from localStorage

const loadData = () => {

const storedAccounts = localStorage.getItem('financialAccounts');

if (storedAccounts) {

accounts = JSON.parse(storedAccounts);

}

const storedCreditCards = localStorage.getItem('financialCreditCards');

if (storedCreditCards) {

creditCards = JSON.parse(storedCreditCards);

}

const storedTransactions = localStorage.getItem('financialTransactions');

if (storedTransactions) {

transactions = JSON.parse(storedTransactions);

}

renderAccounts();

renderCreditCards();

populateAccountSelects(); // Populates all account/card selects

renderTransactions();

updateSummary();

calculateNextMonthExpensesPrediction(); // Calculate prediction on load

renderMonthlySummary(); // Initial render of monthly summary

renderExpensesChart(); // Initial render of the chart

};

// Function to save accounts to localStorage

const saveAccounts = () => {

localStorage.setItem('financialAccounts', JSON.stringify(accounts));

};

// Function to save credit cards to localStorage

const saveCreditCards = () => {

localStorage.setItem('financialCreditCards', JSON.stringify(creditCards));

};

// Function to save transactions to localStorage

const saveTransactions = () => {

localStorage.setItem('financialTransactions', JSON.stringify(transactions));

};

// --- Account and Credit Card Management ---

// Event listener for account type change to show/hide overdraft limit

accountTypeSelect.addEventListener('change', () => {

if (accountTypeSelect.value === 'bank') {

overdraftLimitInput.classList.remove('hidden');

} else {

overdraftLimitInput.classList.add('hidden');

overdraftLimitInput.value = ''; // Clear value if hidden

}

});

// Function to add a new account

addAccountForm.addEventListener('submit', (e) => {

e.preventDefault();

const name = document.getElementById('account-name').value.trim();

const type = document.getElementById('account-type').value;

const initialBalance = parseFloat(initialBalanceInput.value) || 0; // Get initial balance

const overdraftLimit = type === 'bank' ? parseFloat(overdraftLimitInput.value) || 0 : 0; // Get overdraft limit if bank account

if (name) {

const newAccount = {

id: generateUniqueId(),

name: name,

type: type,

balance: initialBalance, // Set initial balance

overdraftLimit: overdraftLimit // Store overdraft limit

};

accounts.push(newAccount);

saveAccounts();

renderAccounts();

populateAccountSelects(); // Update all selects

addAccountForm.reset();

overdraftLimitInput.classList.add('hidden'); // Hide after adding

showFeedback("Conta adicionada com sucesso!", "success");

} else {

showFeedback("Por favor, insira um nome para a conta.", "error");

}

});

// Function to render accounts list

const renderAccounts = () => {

accountsList.innerHTML = '';

if (accounts.length === 0) {

accountsList.innerHTML = '<li class="p-3 text-gray-500 text-center">Nenhuma conta adicionada.</li>';

return;

}

accounts.forEach(account => {

const li = document.createElement('li');

li.classList.add('flex', 'justify-between', 'items-center', 'p-3', 'hover:bg-gray-100', 'rounded-md');

const overdraftDisplay = account.type === 'bank' && account.overdraftLimit > 0 ? ` (+ ${formatCurrency(account.overdraftLimit)} Ch.Esp.)` : '';

li.innerHTML = `

<span class="font-medium text-gray-800">${account.name} (${account.type === 'bank' ? 'Bancária' : 'Dinheiro'})${overdraftDisplay}</span>

<span class="text-gray-600">${formatCurrency(account.balance)}</span>

<div>

<button data-id="${account.id}" data-type="account" class="delete-item-btn text-red-500 hover:text-red-700 ml-2">Excluir</button>

</div>

`;

accountsList.appendChild(li);

});

addDeleteListeners();

};

// Function to add a new credit card

addCreditCardForm.addEventListener('submit', (e) => {

e.preventDefault();

const name = document.getElementById('card-name').value.trim();

const limit = parseFloat(document.getElementById('card-limit').value);

const dueDay = parseInt(cardDueDayInput.value);

if (name && !isNaN(limit) && limit > 0 && !isNaN(dueDay) && dueDay >= 1 && dueDay <= 31) {

const newCard = {

id: generateUniqueId(),

name: name,

limit: limit,

currentUsage: 0, // Current amount used (outstanding balance)

dueDay: dueDay // Due day of the month

};

creditCards.push(newCard);

saveCreditCards();

renderCreditCards();

populateAccountSelects(); // Update all selects

addCreditCardForm.reset();

showFeedback("Cartão de crédito adicionado com sucesso!", "success");

} else {

showFeedback("Por favor, insira um nome, um limite válido e um dia de vencimento entre 1 e 31 para o cartão.", "error");

}

});

// Function to render credit cards list

const renderCreditCards = () => {

creditCardsList.innerHTML = '';

if (creditCards.length === 0) {

creditCardsList.innerHTML = '<li class="p-3 text-gray-500 text-center">Nenhum cartão adicionado.</li>';

return;

}

creditCards.forEach(card => {

const li = document.createElement('li');

li.classList.add('flex', 'justify-between', 'items-center', 'p-3', 'hover:bg-gray-100', 'rounded-md');

const usagePercentage = card.limit > 0 ? (card.currentUsage / card.limit) \* 100 : 0;

const usageColor = usagePercentage > 80 ? 'text-red-600' : (usagePercentage > 50 ? 'text-orange-600' : 'text-green-600');

li.innerHTML = `

<div class="flex flex-col">

<span class="font-medium text-gray-800">${card.name} (Vencimento: Dia ${card.dueDay})</span>

<span class="text-sm text-gray-600">Limite: ${formatCurrency(card.limit)}</span>

<span class="text-sm ${usageColor}">Fatura Atual / Saldo Devedor: ${formatCurrency(card.currentUsage)} (${usagePercentage.toFixed(1)}%)</span>

</div>

<button data-id="${card.id}" data-type="credit-card" class="delete-item-btn text-red-500 hover:text-red-700 ml-2">Excluir</button>

`;

creditCardsList.appendChild(li);

});

addDeleteListeners();

};

// Function to populate all account/credit card select dropdowns

const populateAccountSelects = () => {

const selects = [accountSelect]; // Only populate manual transaction select initially

selects.forEach(selectElement => {

const currentValue = selectElement.value; // Store current value to preserve selection

selectElement.innerHTML = '<option value="">Selecione uma conta</option>';

accounts.forEach(account => {

const option = document.createElement('option');

option.value = `account\_${account.id}`;

option.textContent = `${account.name} (Conta)`;

selectElement.appendChild(option);

});

creditCards.forEach(card => {

const option = document.createElement('option');

option.value = `card\_${card.id}`;

option.textContent = `${card.name} (Cartão)`;

selectElement.appendChild(option);

});

selectElement.value = currentValue; // Restore selection

});

};

// Function to create a select dropdown for account/card linking in review table

const createAccountLinkSelect = (transactionId) => {

const select = document.createElement('select');

select.id = `link-account-${transactionId}`;

select.classList.add('mt-1', 'block', 'w-full', 'px-3', 'py-2', 'border', 'border-gray-300', 'rounded-md', 'shadow-sm', 'focus:outline-none', 'focus:ring-blue-500', 'focus:border-blue-500', 'sm:text-sm');

let defaultOption = document.createElement('option');

defaultOption.value = '';

defaultOption.textContent = 'Vincular...';

select.appendChild(defaultOption);

accounts.forEach(account => {

const option = document.createElement('option');

option.value = `account\_${account.id}`;

option.textContent = `${account.name} (Conta)`;

select.appendChild(option);

});

creditCards.forEach(card => {

const option = document.createElement('option');

option.value = `card\_${card.id}`;

option.textContent = `${card.name} (Cartão)`;

select.appendChild(option);

});

return select;

};

// --- Transaction Management ---

// Event listener for manual transaction form submission

transactionForm.addEventListener('submit', (e) => {

e.preventDefault();

const date = document.getElementById('date').value;

const description = document.getElementById('description').value.trim();

const value = parseFloat(document.getElementById('value').value); // This is the TOTAL value

const type = transactionTypeSelect.value;

const category = document.getElementById('category').value.trim();

const selectedAccountValue = accountSelect.value;

const installments = parseInt(installmentsInput.value);

const paidCount = parseInt(installmentsPaidInput.value) || 0; // NEW: Paid installments input

if (!date || !description || isNaN(value) || value <= 0 || !type || !category || !selectedAccountValue || installments < 1) {

showFeedback("Por favor, preencha todos os campos da transação corretamente.", "error");

return;

}

if (accounts.length === 0 && creditCards.length === 0) {

showFeedback("Por favor, adicione pelo menos uma conta ou cartão antes de adicionar transações.", "error");

return;

}

if (paidCount > installments) {

showFeedback("O número de parcelas pagas não pode ser maior que o total de parcelas.", "error");

return;

}

if (installments > 1 && type === 'receita') {

showFeedback("Receitas não podem ser parceladas. Defina parcelas como 1.", "error");

return;

}

const [accountTypePrefix, accountId] = selectedAccountValue.split('\_');

const valuePerInstallment = value / installments;

const originalTransactionId = generateUniqueId();

// Apply financial impact based on the TOTAL value of the transaction

if (accountTypePrefix === 'account') {

const account = accounts.find(acc => acc.id === accountId);

if (account) {

if (type === 'receita') {

account.balance += value; // Add total value for receipts

} else { // despesa

account.balance -= value; // Subtract total value for expenses

}

}

} else if (accountTypePrefix === 'card' && type === 'despesa') {

const card = creditCards.find(c => c.id === accountId);

if (card) {

card.currentUsage += value; // Add total purchase value to current usage

}

}

// For credit card receipts, they would reduce the currentUsage, but parcelled receipts are rare.

// Generate all installment records

for (let i = 0; i < installments; i++) {

const installmentDate = new Date(date);

installmentDate.setMonth(installmentDate.getMonth() + i);

const newTransaction = {

id: generateUniqueId(),

originalId: installments > 1 ? originalTransactionId : null,

date: installmentDate.toISOString().slice(0, 10),

description: installments > 1 ? `${description} (${i + 1}/${installments})` : description,

value: valuePerInstallment,

totalOriginalValue: value,

type: type,

category: category,

accountId: accountTypePrefix === 'account' ? accountId : null,

creditCardId: accountTypePrefix === 'card' ? accountId : null,

installment: installments > 1 ? `${i + 1}/${installments}` : 'Não',

isPaid: i < paidCount // Mark as paid if within paidCount

};

transactions.push(newTransaction);

}

saveTransactions();

saveAccounts(); // Save updated account balances

saveCreditCards(); // Save updated credit card usages

renderTransactions();

updateSummary();

calculateNextMonthExpensesPrediction(); // Recalculate prediction

renderMonthlySummary(); // Update monthly summary

renderExpensesChart(); // Update chart

transactionForm.reset();

installmentsPaidInput.value = 0; // Reset paid installments to 0

showFeedback("Transação(ões) adicionada(s) com sucesso!", "success");

});

// Function to update account balance (only for bank accounts/cash now)

const updateBalance = (transaction, isDeletion = false) => {

const multiplier = isDeletion ? -1 : 1; // Reverse effect if deleting

if (transaction.accountId) {

const account = accounts.find(acc => acc.id === transaction.accountId);

if (account) {

// For bank accounts, we affect the balance by the individual installment value

// when adding/deleting. The total impact for parcelled is handled by the loop.

if (transaction.type === 'receita') {

account.balance += transaction.value \* multiplier;

} else { // despesa

account.balance -= transaction.value \* multiplier;

}

}

}

// Credit card currentUsage is now managed in transaction form submission and updateSummary

};

// Function to render the list of transactions in the table

const renderTransactions = () => {

transactionList.innerHTML = ''; // Clear existing list

if (transactions.length === 0) {

noTransactionsMessage.classList.remove('hidden');

return;

} else {

noTransactionsMessage.classList.add('hidden');

}

// Sort transactions by date, most recent first

const sortedTransactions = [...transactions].sort((a, b) => new Date(b.date) - new Date(a.date));

sortedTransactions.forEach((transaction, index) => {

const row = document.createElement('tr');

// Add 'paid-transaction' class if isPaid is true

row.classList.add(index % 2 === 0 ? 'bg-white' : 'bg-gray-50', 'hover:bg-gray-100');

if (transaction.isPaid) {

row.classList.add('paid-transaction');

}

const valueColorClass = transaction.type === 'receita' ? 'text-green-600' : 'text-red-600';

const accountOrCardName = transaction.accountId

? accounts.find(acc => acc.id === transaction.accountId)?.name || 'Conta Desconhecida'

: (transaction.creditCardId

? creditCards.find(card => card.id === transaction.creditCardId)?.name || 'Cartão Desconhecido'

: 'N/A');

row.innerHTML = `

<td class="px-6 py-4 whitespace-nowrap text-sm text-gray-900">${transaction.date}</td>

<td class="px-6 py-4 whitespace-nowrap text-sm text-gray-900">${transaction.description}</td>

<td class="px-6 py-4 whitespace-nowrap text-sm font-medium ${valueColorClass}">${formatCurrency(transaction.value)}</td>

<td class="px-6 py-4 whitespace-nowrap text-sm text-gray-900">${transaction.type === 'receita' ? 'Receita' : 'Despesa'}</td>

<td class="px-6 py-4 whitespace-nowrap text-sm text-gray-900">${transaction.category}</td>

<td class="px-6 py-4 whitespace-nowrap text-sm text-gray-900">${accountOrCardName}</td>

<td class="px-6 py-4 whitespace-nowrap text-sm text-gray-900">${transaction.installment || 'Não'}</td>

<td class="px-6 py-4 whitespace-nowrap text-right text-sm font-medium">

<button data-id="${transaction.id}" data-type="transaction" class="delete-item-btn text-red-600 hover:text-red-900 transition duration-150 ease-in-out rounded-md p-2">

Excluir

</button>

</td>

`;

transactionList.appendChild(row);

});

addDeleteListeners();

};

// Function to update the financial summary and recalculate credit card usage

const updateSummary = () => {

let totalReceitas = 0;

let totalDespesas = 0;

transactions.forEach(transaction => {

if (transaction.type === 'receita') {

totalReceitas += transaction.value;

} else if (transaction.type === 'despesa') {

totalDespesas += transaction.value;

}

});

const saldoAtual = totalReceitas - totalDespesas;

totalReceitasEl.textContent = formatCurrency(totalReceitas);

totalDespesasEl.textContent = formatCurrency(totalDespesas);

saldoAtualEl.textContent = formatCurrency(saldoAtual);

if (saldoAtual < 0) {

saldoAtualEl.classList.remove('text-green-700');

saldoAtualEl.classList.add('text-red-700');

} else {

saldoAtualEl.classList.remove('text-red-700');

saldoAtualEl.classList.add('text-green-700');

}

// Recalculate currentUsage for each credit card

creditCards.forEach(card => {

let cardTotalUsage = 0;

const processedOriginalIds = new Set(); // To avoid double counting totalOriginalValue for parcelled transactions

transactions.forEach(t => {

if (t.creditCardId === card.id && t.type === 'despesa') {

if (t.originalId === null || !t.originalId) { // Single transaction

cardTotalUsage += t.value;

} else { // Parcelled transaction

// Only add the totalOriginalValue once for the entire parcelled series

if (!processedOriginalIds.has(t.originalId)) {

cardTotalUsage += t.totalOriginalValue;

processedOriginalIds.add(t.originalId);

}

}

}

// Handle receipts on credit card if they reduce usage

if (t.creditCardId === card.id && t.type === 'receita') {

cardTotalUsage -= t.value; // Receipts reduce usage

}

});

card.currentUsage = cardTotalUsage;

});

saveAccounts(); // Save updated account balances

saveCreditCards(); // Save updated credit card usages

renderAccounts(); // Re-render accounts to show updated balances

renderCreditCards(); // Re-render credit cards to show updated usages

};

// --- Unified Import Logic ---

importFileBtn.addEventListener('click', () => {

const file = fileInput.files[0];

if (!file) {

showFeedback("Por favor, selecione um arquivo para importar.", "error");

return;

}

if (accounts.length === 0 && creditCards.length === 0) {

showFeedback("Por favor, adicione pelo menos uma conta ou cartão antes de importar extratos.", "error");

return;

}

const fileName = file.name.toLowerCase();

const reader = new FileReader();

reader.onload = (e) => {

if (fileName.endsWith('.ofx')) {

parseOFX(e.target.result);

} else if (fileName.endsWith('.csv')) {

parseCSV(e.target.result);

} else if (fileName.endsWith('.xls') || fileName.endsWith('.xlsx')) {

const data = new Uint8Array(e.target.result);

parseExcel(data);

} else {

showFeedback("Formato de arquivo não suportado. Por favor, use .ofx, .csv, .xls ou .xlsx", "error");

}

};

reader.onerror = () => {

showFeedback("Erro ao ler o arquivo.", "error");

};

if (fileName.endsWith('.xls') || fileName.endsWith('.xlsx')) {

reader.readAsArrayBuffer(file);

} else {

reader.readAsText(file);

}

});

// Function to parse OFX content

const parseOFX = (ofxContent) => {

let parser = new DOMParser();

let xmlDoc;

try {

const xmlStart = ofxContent.indexOf('<OFX>');

if (xmlStart === -1) {

showFeedback("Formato OFX inválido: Não encontrado a tag <OFX>.", "error");

return;

}

const cleanOfxContent = ofxContent.substring(xmlStart);

xmlDoc = parser.parseFromString(cleanOfxContent, "text/xml");

if (xmlDoc.getElementsByTagName("parsererror").length > 0) {

console.warn("XML parsing error, attempting regex-based parsing.");

parseOFXWithRegex(ofxContent);

return;

}

} catch (e) {

console.error("Error parsing OFX as XML:", e);

showFeedback("Erro ao parsear o arquivo OFX. Verifique se é um OFX válido.", "error");

return;

}

const transactionsNodes = xmlDoc.getElementsByTagName('STMTTRN');

importedTransactionsToReview = []; // Clear previous review data

if (transactionsNodes.length === 0) {

showFeedback("Nenhuma transação encontrada no arquivo OFX.", "info");

return;

}

Array.from(transactionsNodes).forEach(node => {

const dateNode = node.getElementsByTagName('DTPOSTED')[0];

const amountNode = node.getElementsByTagName('TRNAMT')[0];

const memoNode = node.getElementsByTagName('MEMO')[0];

const typeNode = node.getElementsByTagName('TRNTYPE')[0];

if (dateNode && amountNode && memoNode && typeNode) {

let dateRaw = dateNode.textContent.substring(0, 8); //YYYYMMDD

let description = memoNode.textContent.trim();

let value = parseFloat(amountNode.textContent);

let typeOfx = typeNode.textContent.trim().toLowerCase();

const year = dateRaw.substring(0, 4);

const month = dateRaw.substring(4, 6);

const day = dateRaw.substring(6, 8);

const formattedDate = `${year}-${month}-${day}`;

let type = 'despesa';

if (typeOfx === 'credit' || typeOfx === 'xferin' || value >= 0) {

type = 'receita';

} else if (typeOfx === 'debit' || typeOfx === 'xferout' || value < 0) {

type = 'despesa';

}

const finalValue = Math.abs(value);

importedTransactionsToReview.push({

tempId: generateUniqueId(), // Temporary ID for review table

date: formattedDate,

description: description,

value: finalValue,

type: type,

category: 'OFX Importado', // Default category

totalOriginalValue: finalValue, // For imported, single transaction = total original value

isPaid: false // Default for imported transactions

});

}

});

renderImportReviewTable();

fileInput.value = ''; // Clear the file input

};

// Fallback for OFX parsing with Regex (less robust, but can work for some cases)

const parseOFXWithRegex = (ofxContent) => {

const transactionsRegex = /<STMTTRN>([\s\S]\*?)<\/STMTTRN>/g;

let match;

importedTransactionsToReview = []; // Clear previous review data

while ((match = transactionsRegex.exec(ofxContent)) !== null) {

const transactionBlock = match[1];

const dateMatch = /<DTPOSTED>(\d{8})/.exec(transactionBlock);

const amountMatch = /<TRNAMT>([\d.-]+)/.exec(transactionBlock);

const memoMatch = /<MEMO>([\s\S]\*?)<\/(MEMO|NAME)>/i.exec(transactionBlock); // MEMO or NAME

const typeMatch = /<TRNTYPE>([A-Z]+)/.exec(transactionBlock);

if (dateMatch && amountMatch && memoMatch && typeMatch) {

let dateRaw = dateMatch[1];

let description = memoMatch[1].trim();

let value = parseFloat(amountMatch[1]);

let typeOfx = typeMatch[1].trim().toLowerCase();

const year = dateRaw.substring(0, 4);

const month = dateRaw.substring(4, 6);

const day = dateRaw.substring(6, 8);

const formattedDate = `${year}-${month}-${day}`;

let type = 'despesa';

if (typeOfx === 'credit' || typeOfx === 'xferin' || value >= 0) {

type = 'receita';

} else if (typeOfx === 'debit' || typeOfx === 'xferout' || value < 0) {

type = 'despesa';

}

const finalValue = Math.abs(value);

importedTransactionsToReview.push({

tempId: generateUniqueId(),

date: formattedDate,

description: description,

value: finalValue,

type: type,

category: 'OFX Importado',

totalOriginalValue: finalValue,

isPaid: false // Default for imported transactions

});

}

}

if (importedTransactionsToReview.length > 0) {

renderImportReviewTable();

fileInput.value = '';

} else {

showFeedback("Nenhuma transação válida encontrada no arquivo OFX (tentativa via Regex).", "error");

}

};

// Function to parse CSV content

const parseCSV = (csvContent) => {

const lines = csvContent.split('\n').filter(line => line.trim() !== ''); // Remove empty lines

if (lines.length === 0) {

showFeedback("O arquivo CSV está vazio ou mal formatado.", "error");

return;

}

// Try to detect delimiter (comma or semicolon)

const firstLine = lines[0];

const commaCount = (firstLine.match(/,/g) || []).length;

const semicolonCount = (firstLine.match(/;/g) || []).length;

const delimiter = semicolonCount > commaCount ? ';' : ',';

const headers = lines[0].split(delimiter).map(header => header.trim().toLowerCase());

const dataRows = lines.slice(1);

const dateColIndex = headers.indexOf('data');

const descriptionColIndex = headers.indexOf('descricao') > -1 ? headers.indexOf('descricao') : headers.indexOf('descrição');

const valueColIndex = headers.indexOf('valor');

const typeColIndex = headers.indexOf('tipo');

const categoryColIndex = headers.indexOf('categoria');

if (dateColIndex === -1 || descriptionColIndex === -1 || valueColIndex === -1) {

showFeedback("O CSV deve conter as colunas 'Data', 'Descricao' e 'Valor'.", "error");

return;

}

importedTransactionsToReview = []; // Clear previous review data

dataRows.forEach(row => {

const columns = row.split(delimiter);

if (columns.length <= Math.max(dateColIndex, descriptionColIndex, valueColIndex)) {

console.warn("Linha do CSV ignorada devido a colunas insuficientes:", row);

return;

}

let date = columns[dateColIndex]?.trim();

let description = columns[descriptionColIndex]?.trim();

let valueStr = columns[valueColIndex]?.trim();

let type = typeColIndex !== -1 ? columns[typeColIndex]?.trim().toLowerCase() : '';

let category = categoryColIndex !== -1 ? columns[categoryColIndex]?.trim() : 'Não Categorizado';

valueStr = valueStr.replace(/\./g, '').replace(/,/g, '.'); // Remove thousand separators and replace comma with dot for parsing

const value = parseFloat(valueStr);

// Date format conversion (DD/MM/YYYY toYYYY-MM-DD)

if (date.includes('/')) {

const parts = date.split('/');

if (parts.length === 3) {

date = `${parts[2]}-${parts[1].padStart(2, '0')}-${parts[0].padStart(2, '0')}`;

}

}

if (!date || !description || isNaN(value) || value === 0) {

console.warn("Linha do CSV ignorada devido a dados inválidos:", row);

return;

}

if (type === '') {

type = value >= 0 ? 'receita' : 'despesa';

}

const finalValue = Math.abs(value);

importedTransactionsToReview.push({

tempId: generateUniqueId(),

date: date,

description: description,

value: finalValue,

type: type,

category: 'OFX Importado',

totalOriginalValue: finalValue, // For imported, single transaction = total original value

isPaid: false // Default for imported transactions

});

});

renderImportReviewTable();

fileInput.value = '';

};

// Function to parse Excel content

const parseExcel = (data) => {

const workbook = XLSX.read(data, { type: 'array' });

const firstSheetName = workbook.SheetNames[0];

const worksheet = workbook.Sheets[firstSheetName];

const json = XLSX.utils.sheet\_to\_json(worksheet, { header: 1 }); // Get data as array of arrays

if (json.length === 0) {

showFeedback("O arquivo Excel está vazio.", "error");

return;

}

const headers = json[0].map(h => String(h).trim().toLowerCase());

const dataRows = json.slice(1);

const dateColIndex = headers.indexOf('data');

const descriptionColIndex = headers.indexOf('descricao') > -1 ? headers.indexOf('descricao') : headers.indexOf('descrição');

const valueColIndex = headers.indexOf('valor');

const typeColIndex = headers.indexOf('tipo');

const categoryColIndex = headers.indexOf('categoria');

if (dateColIndex === -1 || descriptionColIndex === -1 || valueColIndex === -1) {

showFeedback("O Excel deve conter as colunas 'Data', 'Descricao' e 'Valor'.", "error");

return;

}

importedTransactionsToReview = []; // Clear previous review data

dataRows.forEach(row => {

if (row.length <= Math.max(dateColIndex, descriptionColIndex, valueColIndex)) {

console.warn("Linha do Excel ignorada devido a colunas insuficientes:", row);

return;

}

let date = row[dateColIndex];

let description = String(row[descriptionColIndex] || '').trim();

let value = parseFloat(row[valueColIndex]);

let type = typeColIndex !== -1 ? String(row[typeColIndex] || '').trim().toLowerCase() : '';

let category = categoryColIndex !== -1 ? String(row[categoryColIndex] || '').trim() : 'Não Categorizado';

// Handle Excel date formats (numeric date toYYYY-MM-DD)

if (typeof date === 'number') {

const excelDate = new Date(Math.round((date - 25569) \* 86400 \* 1000)); // Convert Excel serial date to JS Date

date = excelDate.toISOString().slice(0, 10);

} else {

// Try to parse string date (DD/MM/YYYY orYYYY-MM-DD)

if (String(date).includes('/')) {

const parts = String(date).split('/');

if (parts.length === 3) {

date = `${parts[2]}-${parts[1].padStart(2, '0')}-${parts[0].padStart(2, '0')}`;

}

} else if (String(date).includes('-') && String(date).length >= 10) {

date = String(date).slice(0, 10); // EnsureYYYY-MM-DD format

} else {

// Fallback for other date formats, might need more robust parsing

date = new Date().toISOString().slice(0, 10); // Default to today if date is unparseable

console.warn("Could not parse date from Excel, defaulting to today:", row[dateColIndex]);

}

}

if (!date || !description || isNaN(value) || value === 0) {

console.warn("Linha do Excel ignorada devido a dados inválidos:", row);

return;

}

if (type === '') {

type = value >= 0 ? 'receita' : 'despesa';

}

const finalValue = Math.abs(value);

importedTransactionsToReview.push({

tempId: generateUniqueId(),

date: date,

description: description,

value: finalValue,

type: type,

category: 'OFX Importado',

totalOriginalValue: finalValue,

isPaid: false // Default for imported transactions

});

});

renderImportReviewTable();

fileInput.value = '';

};

// --- Import Review Section Logic ---

const renderImportReviewTable = () => {

importedTransactionsTableBody.innerHTML = ''; // Clear previous review table

if (importedTransactionsToReview.length === 0) {

importReviewSection.classList.add('hidden');

showFeedback("Nenhuma transação válida para revisão.", "info");

return;

}

importReviewSection.classList.remove('hidden');

showFeedback(`Pronto para revisar ${importedTransactionsToReview.length} transações.`, "info");

importedTransactionsToReview.forEach(transaction => {

const row = document.createElement('tr');

row.classList.add('bg-white', 'hover:bg-gray-50');

const valueColorClass = transaction.type === 'receita' ? 'text-green-600' : 'text-red-600';

row.innerHTML = `

<td class="px-6 py-4 whitespace-nowrap text-sm text-gray-900">${transaction.date}</td>

<td class="px-6 py-4 whitespace-nowrap text-sm text-gray-900">${transaction.description}</td>

<td class="px-6 py-4 whitespace-nowrap text-sm font-medium ${valueColorClass}">${formatCurrency(transaction.value)}</td>

<td class="px-6 py-4 whitespace-nowrap text-sm text-gray-900">${transaction.type === 'receita' ? 'Receita' : 'Despesa'}</td>

<td class="px-6 py-4 whitespace-nowrap text-sm text-gray-900">${transaction.category}</td>

<td class="px-6 py-4 whitespace-nowrap text-sm text-gray-900"></td>

`;

const selectCell = row.querySelector('td:last-child');

const accountLinkSelect = createAccountLinkSelect(transaction.tempId);

selectCell.appendChild(accountLinkSelect);

importedTransactionsTableBody.appendChild(row);

});

};

// Event listener for Confirm Import button

confirmImportBtn.addEventListener('click', () => {

let allLinked = true;

importedTransactionsToReview.forEach(tempTransaction => {

const selectElement = document.getElementById(`link-account-${tempTransaction.tempId}`);

if (!selectElement || !selectElement.value) {

allLinked = false;

}

});

if (!allLinked) {

showFeedback("Por favor, vincule TODAS as transações importadas a uma conta ou cartão antes de confirmar.", "error");

return;

}

let successfullyImportedCount = 0;

importedTransactionsToReview.forEach(tempTransaction => {

const selectElement = document.getElementById(`link-account-${tempTransaction.tempId}`);

const selectedAccountValue = selectElement.value; // Guaranteed to have a value now

const [accountTypePrefix, accountId] = selectedAccountValue.split('\_');

const finalTransaction = {

id: generateUniqueId(),

originalId: null, // Imported transactions are treated as single for now

date: tempTransaction.date,

description: tempTransaction.description,

value: tempTransaction.value,

type: tempTransaction.type,

category: tempTransaction.category,

accountId: accountTypePrefix === 'account' ? accountId : null,

creditCardId: accountTypePrefix === 'card' ? accountId : null,

installment: 'Não',

totalOriginalValue: tempTransaction.totalOriginalValue,

isPaid: tempTransaction.isPaid || false // Ensure isPaid is carried over or defaults to false

};

transactions.push(finalTransaction);

// Update account balance for bank accounts

if (accountTypePrefix === 'account') {

updateBalance(finalTransaction);

}

// For credit cards, update currentUsage with the total value of the imported transaction

else if (accountTypePrefix === 'card' && finalTransaction.type === 'despesa') {

const card = creditCards.find(c => c.id === accountId);

if (card) {

card.currentUsage += finalTransaction.totalOriginalValue;

}

}

// For receipts on credit card, reduce usage

else if (accountTypePrefix === 'card' && finalTransaction.type === 'receita') {

const card = creditCards.find(c => c.id === accountId);

if (card) {

card.currentUsage -= finalTransaction.totalOriginalValue;

}

}

successfullyImportedCount++;

});

showFeedback(`${successfullyImportedCount} transações importadas e vinculadas com sucesso!`, "success");

importedTransactionsToReview = []; // Clear review data

importReviewSection.classList.add('hidden'); // Hide review section

saveTransactions();

saveAccounts();

saveCreditCards();

renderTransactions();

updateSummary();

calculateNextMonthExpensesPrediction(); // Recalculate prediction

renderMonthlySummary(); // Update monthly summary

renderExpensesChart(); // Update chart

});

// Event listener for Cancel Import button

cancelReviewBtn.addEventListener('click', () => {

importedTransactionsToReview = []; // Clear review data

importReviewSection.classList.add('hidden'); // Hide review section

showFeedback("Importação cancelada.", "info");

});

// --- Download XLSX ---

// Event listener for XLSX download button

downloadXlsxBtn.addEventListener('click', () => {

// Filter transactions for the current month and year

const transactionsToExport = transactions.filter(t => {

const transactionDate = new Date(t.date);

return transactionDate.getMonth() === currentMonth &&

transactionDate.getFullYear() === currentYear;

});

if (transactionsToExport.length === 0) {

showFeedback(`Não há transações para baixar para ${monthNames[currentMonth]} de ${currentYear}.`, "info");

return;

}

const dataToExport = transactionsToExport.map(t => {

const accountOrCardName = t.accountId

? accounts.find(acc => acc.id === t.accountId)?.name || 'Conta Desconhecida'

: (t.creditCardId

? creditCards.find(card => card.id === t.creditCardId)?.name || 'Cartão Desconhecido'

: 'N/A');

return {

Data: t.date,

Descrição: t.description,

Valor: t.value,

Tipo: t.type === 'receita' ? 'Receita' : 'Despesa',

Categoria: t.category,

'Conta/Cartão': accountOrCardName,

Parcela: t.installment || 'Não',

'Já Paga': t.isPaid ? 'Sim' : 'Não' // Include isPaid status

};

});

const worksheet = XLSX.utils.json\_to\_sheet(dataToExport);

const workbook = XLSX.utils.book\_new();

XLSX.utils.book\_append\_sheet(workbook, worksheet, "Transações");

const fileName = `controle\_financeiro\_${monthNames[currentMonth].toLowerCase()}\_${currentYear}.xlsx`;

XLSX.writeFile(workbook, fileName);

showFeedback(`Planilha do mês de ${monthNames[currentMonth]} de ${currentYear} baixada com sucesso como Excel (.xlsx)!`, "success");

});

// --- Deletion Logic (for accounts, credit cards, and transactions) ---

// Add event listeners for all delete buttons

const addDeleteListeners = () => {

document.querySelectorAll('.delete-item-btn').forEach(button => {

button.onclick = null; // Remove previous listeners to prevent duplicates

button.onclick = (e) => {

const itemId = e.target.dataset.id;

const itemType = e.target.dataset.type;

transactionToDeleteInfo = { id: itemId, type: itemType }; // Store info about what to delete

confirmationModal.style.display = 'flex'; // Show the modal

};

});

};

// Event listeners for the confirmation modal

confirmDeleteBtn.addEventListener('click', () => {

if (transactionToDeleteInfo.type === 'transaction') {

const transactionIdToDelete = transactionToDeleteInfo.id;

const transactionIndex = transactions.findIndex(t => t.id === transactionIdToDelete);

if (transactionIndex !== -1) {

const transaction = transactions[transactionIndex];

// Reverse balance for bank accounts (individual installment value)

if (transaction.accountId) {

updateBalance(transaction, true);

}

// Reverse usage for credit cards (total original value if last installment deleted, or individual value if single)

else if (transaction.creditCardId) {

const card = creditCards.find(c => c.id === transaction.creditCardId);

if (card) {

if (transaction.originalId) { // It's a parcelled transaction

// Find all remaining installments for this original purchase

const remainingInstallments = transactions.filter(t =>

t.originalId === transaction.originalId && t.id !== transactionIdToDelete

);

// If this is the LAST installment of the original purchase being deleted,

// then reverse the TOTAL original value from currentUsage.

if (remainingInstallments.length === 0) {

if (transaction.type === 'despesa') {

card.currentUsage -= transaction.totalOriginalValue;

} else if (transaction.type === 'receita') {

card.currentUsage += transaction.totalOriginalValue; // Reverse receipt

}

}

// If there are other installments remaining, currentUsage (which reflects total purchase)

// should not change. Only the individual installment record is removed.

} else { // It's a single transaction (not parcelled)

if (transaction.type === 'despesa') {

card.currentUsage -= transaction.value;

} else if (transaction.type === 'receita') {

card.currentUsage += transaction.value;

}

}

}

}

transactions.splice(transactionIndex, 1); // Remove the transaction from the array

saveTransactions(); // Save changes

}

} else if (transactionToDeleteInfo.type === 'account') {

const accountId = transactionToDeleteInfo.id;

// Remove account and all associated transactions, reversing their impact on balance

// For accounts, we reverse the impact of each individual transaction associated with it.

transactions.filter(t => t.accountId === accountId).forEach(t => updateBalance(t, true));

accounts = accounts.filter(acc => acc.id !== accountId);

transactions = transactions.filter(t => t.accountId !== accountId);

saveAccounts();

saveTransactions();

} else if (transactionToDeleteInfo.type === 'credit-card') {

const cardId = transactionToDeleteInfo.id;

// When deleting a credit card, we need to reverse the \*total original values\* of all associated parcelled transactions

// and the values of single transactions.

const transactionsToRemove = transactions.filter(t => t.creditCardId === cardId);

const processedOriginalIdsForDeletion = new Set();

transactionsToRemove.forEach(t => {

if (t.type === 'despesa') {

if (t.originalId) {

if (!processedOriginalIdsForDeletion.has(t.originalId)) {

const card = creditCards.find(c => c.id === cardId);

if (card) card.currentUsage -= t.totalOriginalValue;

processedOriginalIdsForDeletion.add(t.originalId);

}

} else {

const card = creditCards.find(c => c.id === cardId);

if (card) card.currentUsage -= t.value;

}

} else if (t.type === 'receita') {

const card = creditCards.find(c => c.id === cardId);

if (card) card.currentUsage += t.value;

}

});

creditCards = creditCards.filter(card => card.id !== cardId);

transactions = transactions.filter(t => t.creditCardId !== cardId);

saveCreditCards();

saveTransactions();

}

renderAccounts();

renderCreditCards();

populateAccountSelects();

renderTransactions();

updateSummary();

calculateNextMonthExpensesPrediction(); // Recalculate prediction

renderMonthlySummary(); // Update monthly summary

renderExpensesChart(); // Update chart

transactionToDeleteInfo = null; // Reset info

confirmationModal.style.display = 'none'; // Hide the modal

});

cancelDeleteBtn.addEventListener('click', () => {

transactionToDeleteInfo = null; // Reset info

confirmationModal.style.display = 'none'; // Hide the modal

});

// --- Prediction Logic ---

const calculateNextMonthExpensesPrediction = () => {

let predictedExpenses = 0;

const today = new Date();

const nextMonthStart = new Date(today.getFullYear(), today.getMonth() + 1, 1);

const nextMonthEnd = new Date(today.getFullYear(), today.getMonth() + 2, 0); // Last day of next month

transactions.forEach(t => {

const transactionDate = new Date(t.date);

// Only consider future expenses (type 'despesa') that are not yet paid

// and fall within the next month

if (t.type === 'despesa' && !t.isPaid &&

transactionDate >= nextMonthStart && transactionDate <= nextMonthEnd) {

predictedExpenses += t.value;

}

});

predictedExpensesEl.textContent = formatCurrency(predictedExpenses);

};

// --- Initial Load ---

window.onload = loadData;

// Event listener for type change to enable/disable installments

transactionTypeSelect.addEventListener('change', () => {

if (transactionTypeSelect.value === 'receita') {

installmentsInput.value = 1;

installmentsInput.disabled = true;

installmentsPaidInput.value = 0;

installmentsPaidInput.disabled = true;

} else {

installmentsInput.disabled = false;

installmentsPaidInput.disabled = false;

}

});

// Toggle import section visibility

importHeader.addEventListener('click', () => {

importSectionContent.classList.toggle('hidden');

importArrow.classList.toggle('rotated');

});

// --- Monthly Summary Logic ---

const monthNames = [

"Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho",

"Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro"

];

const renderMonthlySummary = () => {

currentMonthDisplay.textContent = `${monthNames[currentMonth]} de ${currentYear}`;

let monthlyIncome = 0;

let monthlyExpenses = 0;

transactions.forEach(transaction => {

const transactionDate = new Date(transaction.date);

// Only count transactions for the current month and year, and if they are "paid" or not parcelled

// For monthly summary, we count the individual installment value.

if (transactionDate.getMonth() === currentMonth && transactionDate.getFullYear() === currentYear) {

if (transaction.type === 'receita') {

monthlyIncome += transaction.value;

} else if (transaction.type === 'despesa') {

monthlyExpenses += transaction.value;

}

}

});

const monthlyBalance = monthlyIncome - monthlyExpenses;

monthlyIncomeEl.textContent = formatCurrency(monthlyIncome);

monthlyExpensesEl.textContent = formatCurrency(monthlyExpenses);

monthlyBalanceEl.textContent = formatCurrency(monthlyBalance);

if (monthlyBalance < 0) {

monthlyBalanceEl.classList.remove('text-green-600');

monthlyBalanceEl.classList.add('text-red-600');

} else {

monthlyBalanceEl.classList.remove('text-red-600');

monthlyBalanceEl.classList.add('text-green-600');

}

renderExpensesChart(); // Update chart when month changes

};

prevMonthBtn.addEventListener('click', () => {

currentMonth--;

if (currentMonth < 0) {

currentMonth = 11;

currentYear--;

}

renderMonthlySummary();

});

nextMonthBtn.addEventListener('click', () => {

currentMonth++;

if (currentMonth > 11) {

currentMonth = 0;

currentYear++;

}

renderMonthlySummary();

});

// --- Chart.js Visualization ---

const renderExpensesChart = () => {

const ctx = document.getElementById('expenses-chart').getContext('2d');

// Filter transactions for the current month and type 'despesa'

const monthlyExpenses = transactions.filter(t => {

const transactionDate = new Date(t.date);

return t.type === 'despesa' &&

transactionDate.getMonth() === currentMonth &&

transactionDate.getFullYear() === currentYear;

});

// Aggregate expenses by category

const expensesByCategory = {};

monthlyExpenses.forEach(t => {

const category = t.category || 'Não Categorizado';

expensesByCategory[category] = (expensesByCategory[category] || 0) + t.value;

});

const labels = Object.keys(expensesByCategory);

const data = Object.values(expensesByCategory);

// Generate random colors for the bars

const backgroundColors = labels.map(() => `rgba(${Math.floor(Math.random() \* 255)}, ${Math.floor(Math.random() \* 255)}, ${Math.floor(Math.random() \* 255)}, 0.6)`);

const borderColors = labels.map(color => color.replace('0.6', '1'));

if (expensesChart) {

expensesChart.destroy(); // Destroy previous chart instance

}

expensesChart = new Chart(ctx, {

type: 'bar',

data: {

labels: labels,

datasets: [{

label: 'Despesas por Categoria',

data: data,

backgroundColor: backgroundColors,

borderColor: borderColors,

borderWidth: 1

}]

},

options: {

responsive: true,

maintainAspectRatio: false,

scales: {

y: {

beginAtZero: true,

title: {

display: true,

text: 'Valor (R$)'

}

},

x: {

title: {

display: true,

text: 'Categoria'

}

}

},

plugins: {

legend: {

display: false // No need for legend if only one dataset

},

tooltip: {

callbacks: {

label: function(context) {

let label = context.dataset.label || '';

if (label) {

label += ': ';

}

if (context.parsed.y !== null) {

label += formatCurrency(context.parsed.y);

}

return label;

}

}

}

}

}

});

};

</script>

</body>

</html>