<!DOCTYPE html>  
<html lang="pt-BR">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Catálogo Suquin - Filmes e Séries Moderno</title>  
 <script src="https://cdn.tailwindcss.com"></script>  
 <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700&display=swap" rel="stylesheet">  
 <style>  
 body {  
 font-family: 'Inter', sans-serif;  
 background-color: #000000; /\* AMOLED Black \*/  
 color: #e0e0e0;   
 }  
 /\* General Modal Styling \*/  
 .modal-base {  
 display: none;   
 position: fixed;  
 z-index: 1000;  
 left: 0;  
 top: 0;  
 width: 100%;  
 height: 100%;  
 overflow: auto;  
 background-color: rgba(0,0,0,0.92); /\* Even darker overlay \*/  
 padding: 20px;  
 align-items: center; /\* For vertical centering of modal-content \*/  
 justify-content: center; /\* For horizontal centering \*/  
 }  
 .modal-content-base {  
 background-color: #0a0a0a; /\* Near black \*/  
 margin: auto; /\* Used with flex align/justify on modal-base \*/  
 padding: 25px; /\* Increased padding \*/  
 border: 1px solid #581c87; /\* Darker Lilac/Purple border (violet-800) \*/  
 width: 90%;  
 border-radius: 1rem; /\* rounded-xl \*/  
 position: relative;  
 color: #d1d5db;   
 box-shadow: 0 10px 25px rgba(139, 92, 246, 0.1), 0 5px 10px rgba(139, 92, 246, 0.08); /\* Softer lilac shadow \*/  
 }  
 /\* Main Item Modal \*/  
 #itemModal .modal-content-base { max-width: 850px; }  
 /\* Episode Player Modal (Second Modal) \*/  
 #episodePlayerModal .modal-content-base { max-width: 700px; background-color: #080808; }  
 #episodePlayerModal { z-index: 1050; } /\* Ensure it's on top \*/  
  
 .close-button {  
 color: #a78bfa; /\* Lilac (violet-400) \*/  
 position: absolute;  
 top: 18px; /\* Adjusted for padding \*/  
 right: 25px; /\* Adjusted for padding \*/  
 font-size: 38px; /\* Slightly larger \*/  
 font-weight: bold;  
 cursor: pointer;  
 transition: color 0.3s, transform 0.2s;  
 }  
 .close-button:hover,  
 .close-button:focus {  
 color: #c4b5fd; /\* Lighter Lilac (violet-300) \*/  
 transform: scale(1.1);  
 }  
 .search-input {  
 background-color: #181818; /\* Darker input bg \*/  
 border: 1px solid #4a0072; /\* Lilac/Purple border \*/  
 color: #e0e0e0;  
 border-radius: 0.75rem; /\* rounded-lg \*/  
 padding: 0.9rem 1.1rem; /\* Increased padding \*/  
 }  
 .search-input:focus {  
 outline: none;  
 box-shadow: 0 0 0 3px rgba(139, 92, 246, 0.5); /\* Lilac focus ring (violet-500) \*/  
 border-color: #8b5cf6;  
 }  
 .item-card {  
 background-color: #101010; /\* Darker gray for cards \*/  
 border: 1px solid #252525;   
 transition: transform 0.3s ease, box-shadow 0.3s ease, border-color 0.3s ease;  
 overflow: hidden;  
 border-radius: 0.75rem; /\* rounded-lg \*/  
 }  
 .item-card:hover {  
 transform: translateY(-6px); /\* Increased hover effect \*/  
 box-shadow: 0 12px 20px -4px rgba(167, 139, 250, 0.2), 0 5px 10px -3px rgba(167, 139, 250, 0.15);   
 border-color: #8b5cf6;   
 }  
 .item-card img {  
 transition: transform 0.3s ease;  
 border-radius: 0.75rem 0.75rem 0 0; /\* Rounded top corners for image \*/  
 }  
 .item-card:hover img {  
 transform: scale(1.05);  
 }  
 .loader, .gemini-loader {  
 border: 4px solid #4a0072;   
 border-top: 4px solid #a78bfa;   
 border-radius: 50%;  
 width: 30px;  
 height: 30px;  
 animation: spin 1s linear infinite;  
 }  
 .loader {   
 width: 50px;  
 height: 50px;  
 margin: 25px auto; /\* Increased margin \*/  
 border-width: 5px;  
 }  
 .gemini-loader {  
 display: inline-block;  
 vertical-align: middle;  
 margin-left: 10px; /\* Increased margin \*/  
 }  
 @keyframes spin {  
 0% { transform: rotate(0deg); }  
 100% { transform: rotate(360deg); }  
 }  
 .modal-body::-webkit-scrollbar, .episode-list-container::-webkit-scrollbar {  
 width: 10px; /\* Slightly thicker scrollbar \*/  
 }  
 .modal-body::-webkit-scrollbar-track, .episode-list-container::-webkit-scrollbar-track {  
 background: #1a1a1a;   
 border-radius: 10px;  
 }  
 .modal-body::-webkit-scrollbar-thumb, .episode-list-container::-webkit-scrollbar-thumb {  
 background: #8b5cf6;   
 border-radius: 10px;  
 border: 2px solid #1a1a1a; /\* Border around thumb \*/  
 }  
 .modal-body::-webkit-scrollbar-thumb:hover, .episode-list-container::-webkit-scrollbar-thumb:hover {  
 background: #7c3aed;   
 }  
 #EmbedderIframeContainer iframe, #EpisodeEmbedderIframeContainer iframe {  
 width: 100%;  
 height: 450px;   
 border-radius: 0.75rem; /\* rounded-lg \*/  
 background-color: #000;   
 }  
 .gemini-section, .player-section, .seasons-section {  
 margin-top: 25px; /\* Increased margin \*/  
 padding-top: 25px; /\* Increased padding \*/  
 border-top: 1px solid #4a0072;   
 }  
 .gemini-output {  
 background-color: #1f1f1f;   
 border: 1px solid #4a0072;  
 padding: 15px; /\* Increased padding \*/  
 border-radius: 0.75rem; /\* rounded-lg \*/  
 margin-top: 12px;   
 font-size: 0.9em;  
 line-height: 1.6;  
 white-space: pre-wrap;   
 color: #c4b5fd;   
 }  
 .action-button, .gemini-button, .category-button, .episode-button, .season-button {   
 background-color: #7c3aed;   
 color: white;  
 padding: 12px 20px; /\* Larger padding \*/  
 border-radius: 0.75rem; /\* rounded-lg \*/  
 font-size: 0.95em;   
 font-weight: 500;  
 transition: background-color 0.2s, box-shadow 0.2s, transform 0.1s;  
 cursor: pointer;  
 display: inline-flex;  
 align-items: center;  
 margin-bottom: 10px;   
 border: none;  
 box-shadow: 0 3px 6px rgba(0,0,0,0.25);  
 }  
 .action-button:hover, .gemini-button:hover, .category-button:hover, .episode-button:hover, .season-button:hover {  
 background-color: #6d28d9;   
 box-shadow: 0 5px 10px rgba(0,0,0,0.3);  
 transform: translateY(-1px);  
 }  
 .action-button:active, .gemini-button:active, .category-button:active, .episode-button:active, .season-button:active {  
 transform: translateY(0px);  
 box-shadow: 0 2px 4px rgba(0,0,0,0.2);  
 }  
 .action-button:disabled, .gemini-button:disabled, .category-button:disabled, .episode-button:disabled, .season-button:disabled {  
 background-color: #581c87;   
 cursor: not-allowed;  
 box-shadow: none;  
 transform: translateY(0px);  
 }  
 .category-button.active {  
 background-color: #8b5cf6;   
 color: #fff;  
 font-weight: 600;  
 box-shadow: inset 0 2px 4px rgba(0,0,0,0.15), 0 1px 2px rgba(139,92,246,0.6);  
 }  
 .season-button {  
 width: 100%;  
 justify-content: space-between;  
 background-color: #2a0045; /\* Darker base for season buttons \*/  
 margin-bottom: 6px;  
 }  
 .season-button.active { /\* Style for expanded season \*/  
 background-color: #8b5cf6;  
 }  
 .episode-list-container {  
 max-height: 300px;  
 overflow-y: auto;  
 padding-right: 10px; /\* Space for scrollbar \*/  
 margin-top: 8px;  
 background-color: rgba(255,255,255,0.03);  
 border-radius: 0.5rem; /\* rounded-md \*/  
 padding: 8px;  
 }  
 .episode-button {  
 width: 100%;  
 text-align: left;  
 padding: 10px 15px;  
 font-size: 0.9em;  
 background-color: #3c1a56; /\* Base for episode buttons \*/  
 margin-bottom: 5px;  
 }  
 .episode-button:hover {  
 background-color: #4a0072;  
 }  
 .toast-message {  
 position: fixed;  
 bottom: 25px; /\* Increased offset \*/  
 left: 50%;  
 transform: translateX(-50%);  
 background-color: #a78bfa;   
 color: #0d0d0d;   
 padding: 12px 22px; /\* Increased padding \*/  
 border-radius: 0.75rem; /\* rounded-lg \*/  
 z-index: 2050; /\* Above episode modal \*/  
 font-size: 0.95em;  
 box-shadow: 0 4px 15px rgba(0,0,0,0.35);  
 }  
 .section-title {  
 font-size: 2rem; /\* text-3xl equivalent \*/  
 font-weight: 700; /\* font-bold \*/  
 color: #c4b5fd;   
 margin-bottom: 1.25rem; /\* mb-5 \*/  
 padding-bottom: 0.75rem; /\* pb-3 \*/  
 border-bottom: 2px solid #581c87;   
 }  
 #episodePlayerTitle { /\* Styling for episode player modal title \*/  
 color: #c4b5fd;  
 font-size: 1.5rem; /\* text-2xl \*/  
 font-weight: 600; /\* font-semibold \*/  
 margin-bottom: 1rem;  
 }  
 </style>  
</head>  
<body>  
 <div class="container mx-auto px-4 py-10">  
 <header class="mb-12 text-center">  
 <h1 class="text-5xl font-bold text-violet-400 tracking-tight">Catálogo Suquin</h1>  
 <p class="text-xl text-gray-400 mt-3">Seu universo de filmes e séries.</p>  
 </header>  
  
 <nav class="mb-10 flex justify-center space-x-4 sm:space-x-5">  
 <button id="btnExplore" class="category-button active">Explorar</button>  
 <button id="btnMovies" class="category-button">Filmes</button>  
 <button id="btnSeries" class="category-button">Séries</button>  
 </nav>  
   
 <div class="mb-10 max-w-2xl mx-auto">  
 <input type="text" id="searchInput"  
 class="search-input w-full placeholder-gray-500 text-lg"  
 placeholder="Buscar por filmes ou séries...">  
 </div>  
   
 <h2 id="currentViewTitle" class="section-title text-center">Explorando o que está em alta</h2>  
  
 <div id="loader" class="loader hidden"></div>  
   
 <div id="resultsGrid" class="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 xl:grid-cols-5 gap-x-8 gap-y-10 mt-8">  
 </div>  
 <p id="noResults" class="text-center text-gray-500 hidden mt-12 text-lg">Nenhum resultado encontrado.</p>  
 </div>  
  
 <div id="itemModal" class="modal-base">  
 <div class="modal-content-base">  
 <span class="close-button" id="closeModalButton">×</span>  
 <div id="modalBody" class="modal-body max-h-[80vh] overflow-y-auto pr-3">  
 </div>  
 </div>  
 </div>  
  
 <div id="episodePlayerModal" class="modal-base">  
 <div class="modal-content-base">  
 <span class="close-button" id="closeEpisodeModalButton">×</span>  
 <div id="episodeModalBody" class="max-h-[80vh] overflow-y-auto">  
 <h3 id="episodePlayerTitle" class="text-center"></h3>  
 <div id="EpisodeEmbedderIframeContainer" class="my-4"></div>  
 <div class="text-center">  
 <button id="copyEpisodeLinkBtn" class="action-button" title="Copiar link do episódio">  
 Copiar Link do Episódio  
 <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" viewBox="0 0 24 24" fill="none" stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-linejoin="round" class="ml-2"><rect x="9" y="9" width="13" height="13" rx="2" ry="2"></rect><path d="M5 15H4a2 2 0 0 1-2-2V4a2 2 0 0 1 2-2h9a2 2 0 0 1 2 2v1"></path></svg>  
 </button>  
 </div>  
 </div>  
 </div>  
 </div>  
  
 <footer class="text-center py-12 mt-20 border-t border-gray-800">  
 <p class="text-gray-500">Catálogo Suquin © 2024 - Powered by TMDb, Embedder.net & Gemini AI</p>  
 </footer>  
  
 <script>  
 const TMDB\_API\_KEY = '5e5da432e96174227b25086fe8637985';  
 const TMDB\_BASE\_URL = 'https://api.themoviedb.org/3';  
 const IMAGE\_BASE\_URL = 'https://image.tmdb.org/t/p/w500';  
 const GEMINI\_API\_KEY = "";   
  
 const searchInput = document.getElementById('searchInput');  
 const resultsGrid = document.getElementById('resultsGrid');  
   
 const itemModal = document.getElementById('itemModal');  
 const modalBody = document.getElementById('modalBody');  
 const closeModalButton = document.getElementById('closeModalButton');  
   
 const episodePlayerModal = document.getElementById('episodePlayerModal');  
 // const episodeModalBody = document.getElementById('episodeModalBody'); // Already declared above, but ensure used correctly  
 const closeEpisodeModalButton = document.getElementById('closeEpisodeModalButton');  
 const episodePlayerTitleEl = document.getElementById('episodePlayerTitle');  
 const copyEpisodeLinkBtn = document.getElementById('copyEpisodeLinkBtn');  
  
 const loader = document.getElementById('loader');  
 const noResultsMessage = document.getElementById('noResults');  
 const currentViewTitleEl = document.getElementById('currentViewTitle');  
   
 const btnExplore = document.getElementById('btnExplore');  
 const btnMovies = document.getElementById('btnMovies');  
 const btnSeries = document.getElementById('btnSeries');  
 const categoryButtons = [btnExplore, btnMovies, btnSeries];  
  
 let currentItemModalEmbedderUrl = '';   
 let currentEpisodeEmbedderUrl = '';  
 let isItemModalOpen = false;  
 let isEpisodeModalOpen = false;  
  
  
 function showToast(message) {  
 const existingToast = document.querySelector('.toast-message');  
 if (existingToast) existingToast.remove();  
 const toast = document.createElement('div');  
 toast.className = 'toast-message';  
 toast.textContent = message;  
 document.body.appendChild(toast);  
 setTimeout(() => toast.remove(), 3000);  
 }  
  
 function embedderPlugin(type, imdb, season, episode, targetContainerId = 'EmbedderIframeContainer') {  
 var frameContainer = document.getElementById(targetContainerId);  
 if (!frameContainer) {  
 console.error(`Elemento '${targetContainerId}' não encontrado.`);  
 if (targetContainerId === 'EmbedderIframeContainer') currentItemModalEmbedderUrl = '';  
 else if (targetContainerId === 'EpisodeEmbedderIframeContainer') currentEpisodeEmbedderUrl = '';  
 return;  
 }  
 // The original plugin script creates a div with ID "EmbedderIframe" inside the target.  
 // We ensure our container is clean and then the plugin's logic (if it were external) would find this.  
 // For our integrated version, we directly create and manage the iframe.  
 frameContainer.innerHTML = ''; // Clear previous content  
   
 let seasonPath = "", episodePath = "";  
 if (type === "serie") {  
 if (season) seasonPath = "/" + season;  
 if (episode) episodePath = "/" + episode;  
 }  
 var ref = document.referrer || window.location.origin;  
 const embedUrl = `https://embedder.net/e/${imdb}${seasonPath}${episodePath}?ref=${encodeURIComponent(ref)}`;  
   
 if (targetContainerId === 'EmbedderIframeContainer') currentItemModalEmbedderUrl = embedUrl;  
 else if (targetContainerId === 'EpisodeEmbedderIframeContainer') currentEpisodeEmbedderUrl = embedUrl;  
   
 const iframe = document.createElement('iframe');  
 iframe.src = embedUrl;  
 iframe.setAttribute('scrolling', 'no');  
 iframe.setAttribute('frameborder', '0');  
 iframe.setAttribute('allowfullscreen', '');  
 iframe.setAttribute('webkitallowfullscreen', '');  
 iframe.setAttribute('mozallowfullscreen', '');  
 iframe.style.width = '100%';  
 iframe.style.height = '450px';  
 iframe.style.borderRadius = '0.75rem';  
 iframe.style.backgroundColor = '#000';  
 frameContainer.appendChild(iframe);  
 }  
  
  
 async function fetchMultiplePages(endpoint, baseParams = {}, pagesToFetch = 2) {  
 let allResults = [];  
 const promises = [];  
 for (let i = 1; i <= pagesToFetch; i++) {  
 promises.push(  
 fetch(`${TMDB\_BASE\_URL}/${endpoint}?${new URLSearchParams({ api\_key: TMDB\_API\_KEY, language: 'pt-BR', page: i, ...baseParams })}`)  
 .then(response => {  
 if (!response.ok) throw new Error(`HTTP error! status: ${response.status} for page ${i}`);  
 return response.json();  
 })  
 );  
 }  
 try {  
 const pageDataArray = await Promise.all(promises);  
 pageDataArray.forEach(pageData => {  
 if (pageData?.results) allResults = allResults.concat(pageData.results);  
 });  
 allResults = allResults.filter((item, index, self) => index === self.findIndex((t) => (t.id === item.id)));  
 return { results: allResults };  
 } catch (error) {  
 console.error(`Failed to fetch multiple pages for ${endpoint}:`, error);  
 throw error;  
 }  
 }  
  
 async function fetchAndDisplay(endpoint, title, activeBtn, baseParams = {}, pages = 2) {  
 if (currentViewTitleEl) currentViewTitleEl.textContent = title; // Set title at the beginning  
 setActiveButton(activeBtn);  
 if (loader) loader.classList.remove('hidden');  
 if (resultsGrid) resultsGrid.innerHTML = '';   
 if (noResultsMessage) noResultsMessage.classList.add('hidden');  
 try {  
 const data = await fetchMultiplePages(endpoint, baseParams, pages);  
 displayResults(data?.results || [], title); // Pass title for context in noResultsMessage  
 } catch (error) {  
 if (currentViewTitleEl) currentViewTitleEl.textContent = "Erro ao Carregar Dados";  
 if (noResultsMessage) {  
 noResultsMessage.textContent = "Não foi possível carregar. Tente novamente.";  
 noResultsMessage.classList.remove('hidden');  
 }  
 displayResults([], title); // Pass title for context  
 } finally {  
 if (loader) loader.classList.add('hidden');  
 }  
 }  
   
 function setActiveButton(activeBtn) {  
 categoryButtons.forEach(btn => btn.classList.remove('active'));  
 if (activeBtn) activeBtn.classList.add('active');  
 }  
  
 async function fetchTrending() { await fetchAndDisplay('trending/all/week', "Explorando o que está em alta", btnExplore); }  
 async function fetchMovies() { await fetchAndDisplay('discover/movie', "Filmes Populares", btnMovies, { sort\_by: 'popularity.desc' }); }  
 async function fetchSeries() { await fetchAndDisplay('discover/tv', "Séries Populares", btnSeries, { sort\_by: 'popularity.desc' }); }  
 async function searchItems(query) {  
 if (!query.trim()) { fetchTrending(); return; }  
 await fetchAndDisplay('search/multi', `Resultados da Busca por: "${query}"`, null, { query }, 2);  
 }  
  
 async function fetchItemDetails(itemId, mediaType) {  
 if(loader) loader.classList.remove('hidden');  
 const endpoint = `${mediaType}/${itemId}`;  
 const paramsToAppend = mediaType === 'tv' ? 'videos,external\_ids,aggregate\_credits' : 'videos,external\_ids';  
 const urlParams = new URLSearchParams({ api\_key: TMDB\_API\_KEY, language: 'pt-BR', append\_to\_response: paramsToAppend });  
 try {  
 const response = await fetch(`${TMDB\_BASE\_URL}/${endpoint}?${urlParams}`);  
 if (!response.ok) throw new Error(`HTTP error! status: ${response.status}`);  
 const data = await response.json();  
 if (data) await displayModal(data, mediaType);  
 } catch (error) {  
 console.error(`Failed to fetch item details for ${mediaType}/${itemId}:`, error);  
 showToast("Erro ao carregar detalhes. Tente novamente.");  
 } finally {  
 if(loader) loader.classList.add('hidden');  
 }  
 }  
  
 async function fetchSeasonDetails(tvId, seasonNumber) {  
 const endpoint = `tv/${tvId}/season/${seasonNumber}`;  
 const urlParams = new URLSearchParams({ api\_key: TMDB\_API\_KEY, language: 'pt-BR' });  
 try {  
 const response = await fetch(`${TMDB\_BASE\_URL}/${endpoint}?${urlParams}`);  
 if (!response.ok) throw new Error(`HTTP error! status: ${response.status}`);  
 return await response.json();  
 } catch (error) {  
 console.error(`Failed to fetch season ${seasonNumber} for TV ID ${tvId}:`, error);  
 showToast(`Erro ao carregar episódios da temporada ${seasonNumber}.`);  
 return null;  
 }  
 }  
  
 async function fetchGeminiApi(prompt, buttonElement, outputElementId) {  
 const buttonLoader = buttonElement.querySelector('.gemini-loader');  
 buttonElement.disabled = true;  
 if (buttonLoader) buttonLoader.classList.remove('hidden');  
 const outputElement = document.getElementById(outputElementId);  
 if(outputElement) outputElement.innerHTML = '<p class="text-violet-300">✨ Pensando...</p>';  
 let chatHistory = [{ role: "user", parts: [{ text: prompt }] }];  
 const payload = { contents: chatHistory };  
 const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=${GEMINI\_API\_KEY}`;  
 try {  
 const response = await fetch(apiUrl, { method: 'POST', headers: { 'Content-Type': 'application/json' }, body: JSON.stringify(payload) });  
 if (!response.ok) {  
 const errorData = await response.json();  
 throw new Error(`Gemini API error: ${errorData.error?.message || response.status}`);  
 }  
 const result = await response.json();  
 if (result.candidates?.[0]?.content?.parts?.[0]?.text) {  
 if(outputElement) outputElement.innerHTML = result.candidates[0].content.parts[0].text.replace(/\n/g, '<br>');  
 } else { throw new Error('Resposta inesperada da API Gemini.'); }  
 } catch (error) {  
 console.error('Failed to fetch from Gemini API:', error);  
 if(outputElement) outputElement.innerHTML = `<p class="text-red-400">Oops! Não consegui gerar a informação. (${error.message})</p>`;  
 } finally {  
 buttonElement.disabled = false;  
 if (buttonLoader) buttonLoader.classList.add('hidden');  
 }  
 }  
  
 function displayResults(items, operationTitle) { // Added operationTitle for context  
 if (resultsGrid) resultsGrid.innerHTML = '';  
 const validItems = items ? items.filter(item => item.poster\_path) : [];   
   
 if (noResultsMessage) { // Check if element exists  
 if (validItems.length === 0) {  
 if (operationTitle && operationTitle.toLowerCase().startsWith("resultados da busca por:")) {  
 const searchTermMatch = operationTitle.match(/Resultados da Busca por: "([^"]\*)"/);  
 const searchTerm = searchTermMatch ? searchTermMatch[1] : searchInput.value.trim();  
 noResultsMessage.textContent = `Nenhum resultado encontrado para "${searchTerm}".`;  
 } else {  
 noResultsMessage.textContent = "Nenhum item encontrado nesta categoria.";  
 }  
 noResultsMessage.classList.remove('hidden');  
 return;  
 }  
 noResultsMessage.classList.add('hidden');  
 }  
  
 validItems.forEach(item => {  
 const effectiveMediaType = item.media\_type || (item.title ? 'movie' : 'tv');  
 const card = document.createElement('div');  
 card.className = 'item-card rounded-lg shadow-lg cursor-pointer overflow-hidden flex flex-col';  
 card.addEventListener('click', () => fetchItemDetails(item.id, effectiveMediaType));  
 const img = document.createElement('img');  
 img.src = `${IMAGE\_BASE\_URL}${item.poster\_path}`;  
 img.alt = item.title || item.name;  
 img.className = 'w-full h-auto object-cover aspect-[2/3]';  
 img.onerror = () => {   
 img.src = `https://placehold.co/500x750/101010/581c87?text=Indispon%C3%ADvel`;  
 img.alt = 'Imagem Indisponível';  
 };  
 const infoDiv = document.createElement('div');  
 infoDiv.className = 'p-3 flex flex-col flex-grow';   
 const titleEl = document.createElement('h3');  
 titleEl.className = 'text-md font-semibold text-violet-300 truncate mb-1';   
 titleEl.textContent = item.title || item.name;  
 titleEl.title = item.title || item.name;  
 const dateText = item.release\_date || item.first\_air\_date;  
 const dateEl = document.createElement('p');  
 dateEl.className = 'text-xs text-gray-500';   
 dateEl.textContent = dateText ? new Date(dateText).getFullYear() : 'Ano N/A';   
 infoDiv.appendChild(titleEl);  
 infoDiv.appendChild(dateEl);  
 card.appendChild(img);  
 card.appendChild(infoDiv);  
 if (resultsGrid) resultsGrid.appendChild(card);  
 });  
 }  
  
 async function displayModal(item, mediaType) { // Made async  
 const title = item.title || item.name;  
 const overview = item.overview || 'Sinopse não disponível.';  
 const releaseDate = item.release\_date || item.first\_air\_date;  
 const rating = item.vote\_average ? item.vote\_average.toFixed(1) : 'N/A';  
 const posterPath = item.poster\_path ? `${IMAGE\_BASE\_URL}${item.poster\_path}` : `https://placehold.co/400x600/0a0a0a/581c87?text=Poster`;  
 currentItemModalEmbedderUrl = '';   
  
 let playerSectionHTML = '';  
 let seasonsSectionHTML = '';  
  
 if (mediaType === 'movie') {  
 playerSectionHTML = `  
 <div class="player-section">  
 <div class="flex justify-between items-center mb-2">  
 <h4 class="text-xl font-semibold text-violet-200">Player (Embedder.net)</h4>  
 <button id="copyMainPlayerLinkBtn" class="action-button text-xs py-1 px-3 rounded-md" title="Copiar link do player">  
 Copiar Link  
 <svg xmlns="http://www.w3.org/2000/svg" width="14" height="14" viewBox="0 0 24 24" fill="none" stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-linejoin="round" class="ml-2"><rect x="9" y="9" width="13" height="13" rx="2" ry="2"></rect><path d="M5 15H4a2 2 0 0 1-2-2V4a2 2 0 0 1 2-2h9a2 2 0 0 1 2 2v1"></path></svg>  
 </button>  
 </div>  
 <div id="embedderPlayerMessage" class="text-gray-400 mb-2">Verificando disponibilidade...</div>  
 <div id="EmbedderIframeContainer"></div>  
 </div>`;  
 } else if (mediaType === 'tv') {  
 seasonsSectionHTML = `  
 <div class="seasons-section">  
 <h4 class="text-xl font-semibold text-violet-200 mb-3">Temporadas e Episódios</h4>  
 <div id="seasonsAccordion"></div>  
 </div>`;  
 }  
  
 if (modalBody) modalBody.innerHTML = `  
 <div class="flex flex-col md:flex-row gap-8">  
 <div class="md:w-1/3 flex-shrink-0">  
 <img src="${posterPath}" alt="${title}" class="w-full rounded-xl shadow-lg" onerror="this.src='https://placehold.co/400x600/0a0a0a/581c87?text=Poster'; this.alt='Poster Indisponível';">  
 </div>  
 <div class="md:w-2/3">  
 <h2 class="text-3xl lg:text-4xl font-bold mb-3 text-violet-300">${title}</h2>  
 <p class="text-sm text-gray-400 mb-1"><strong>Tipo:</strong> ${mediaType === 'movie' ? 'Filme' : 'Série de TV'}</p>  
 <p class="text-sm text-gray-400 mb-1"><strong>Lançamento:</strong> ${releaseDate ? new Date(releaseDate).toLocaleDateString('pt-BR') : 'Não informado'}</p>  
 <p class="text-sm text-gray-400 mb-4"><strong>Avaliação:</strong> ${rating} ⭐</p>  
 <h4 class="text-xl font-semibold mb-2 text-violet-200">Sinopse Padrão</h4>  
 <p class="text-gray-300 mb-6 text-sm leading-relaxed">${overview}</p>  
 </div>  
 </div>  
 ${playerSectionHTML}  
 ${seasonsSectionHTML}  
 <div class="gemini-section">  
 <h3 class="text-2xl font-semibold mb-4 text-violet-300">✨ Explore com IA Gemini</h3>  
 <div class="mb-3">   
 <button id="generateCreativeSummaryBtn" class="gemini-button">  
 Gerar Resumo Criativo <span class="gemini-loader hidden"></span>  
 </button>  
 <div id="creativeSummaryOutput" class="gemini-output mt-2 hidden"></div>  
 </div>  
 <div class="mb-3">   
 <button id="generateMovieNightBtn" class="gemini-button">  
 Ideias para Noite Temática <span class="gemini-loader hidden"></span>  
 </button>  
 <div id="movieNightOutput" class="gemini-output mt-2 hidden"></div>  
 </div>  
 <div>   
 <button id="generateTriviaBtn" class="gemini-button">  
 Gerar Curiosidades <span class="gemini-loader hidden"></span>  
 </button>  
 <div id="triviaOutput" class="gemini-output mt-2 hidden"></div>  
 </div>  
 </div>  
 `;  
 if (itemModal) itemModal.style.display = 'flex';  
 document.body.style.overflow = 'hidden';  
 isItemModalOpen = true;  
  
 if (mediaType === 'movie') {  
 const imdbId = item.external\_ids?.imdb\_id;  
 const tmdbId = item.id;   
 const embedderPlayerMessageEl = document.getElementById('embedderPlayerMessage');  
 const copyMainPlayerLinkBtn = document.getElementById('copyMainPlayerLinkBtn');  
  
 if (copyMainPlayerLinkBtn) {  
 copyMainPlayerLinkBtn.disabled = true;   
 copyMainPlayerLinkBtn.addEventListener('click', () => {  
 if (currentItemModalEmbedderUrl) {  
 const textarea = document.createElement('textarea');  
 textarea.value = currentItemModalEmbedderUrl;  
 document.body.appendChild(textarea);  
 textarea.select();  
 try { document.execCommand('copy'); showToast('Link do player copiado!'); }   
 catch (err) { showToast('Falha ao copiar.'); console.error('Falha ao copiar:', err); }  
 document.body.removeChild(textarea);  
 } else { showToast('Link do player não disponível.'); }  
 });  
 }  
  
 if (document.getElementById('EmbedderIframeContainer')) document.getElementById('EmbedderIframeContainer').innerHTML = '';  
 if (embedderPlayerMessageEl) embedderPlayerMessageEl.textContent = 'Verificando disponibilidade...';  
  
 if (!imdbId) {  
 if (embedderPlayerMessageEl) embedderPlayerMessageEl.textContent = 'ID IMDB não encontrado. Player não disponível.';  
 if (copyMainPlayerLinkBtn) copyMainPlayerLinkBtn.disabled = true;  
 currentItemModalEmbedderUrl = '';  
 } else {  
 const statusUrl = `https://embedder.net/api/status?tmdb=${tmdbId}&type=movie`;  
 try {  
 const statusResponse = await fetch(statusUrl);   
 if (!statusResponse.ok) throw new Error(`Erro HTTP ${statusResponse.status}`);  
 const statusText = await statusResponse.text();  
 const isAvailable = statusText.trim().toLowerCase() === 'true';  
 if (isAvailable) {  
 if (embedderPlayerMessageEl) embedderPlayerMessageEl.textContent = `Disponível! Carregando player...`;  
 embedderPlugin('filme', imdbId, null, null, 'EmbedderIframeContainer');  
 if (copyMainPlayerLinkBtn) copyMainPlayerLinkBtn.disabled = false;  
 } else {  
 if (embedderPlayerMessageEl) embedderPlayerMessageEl.textContent = `Conteúdo não disponível no Embedder.net.`;  
 if (copyMainPlayerLinkBtn) copyMainPlayerLinkBtn.disabled = true;  
 currentItemModalEmbedderUrl = '';  
 }  
 } catch (error) {   
 console.error("Falha ao verificar status (Movie):", statusUrl, error);   
 if (embedderPlayerMessageEl) embedderPlayerMessageEl.innerHTML = `Falha ao verificar (Erro: ${error.message}). <br>Tentando carregar player...`;  
 embedderPlugin('filme', imdbId, null, null, 'EmbedderIframeContainer');   
 if (copyMainPlayerLinkBtn && currentItemModalEmbedderUrl) copyMainPlayerLinkBtn.disabled = false; else if (copyMainPlayerLinkBtn) copyMainPlayerLinkBtn.disabled = true;  
 }  
 }  
 } else if (mediaType === 'tv') {  
 const seasonsAccordion = document.getElementById('seasonsAccordion');  
 if (item.seasons && seasonsAccordion) {  
 const validSeasons = item.seasons.filter(s => s.season\_number > 0 && s.episode\_count > 0);  
 validSeasons.forEach(async (season, index) => {  
 const seasonDiv = document.createElement('div');  
 const seasonButton = document.createElement('button');  
 seasonButton.className = 'season-button';  
 seasonButton.innerHTML = `${season.name || `Temporada ${season.season\_number}`} <span class="text-xs text-gray-400">(${season.episode\_count} episódios)</span> <span class="text-violet-300 ml-auto text-xl">${index === 0 ? '▾' : '▸'}</span>`; // Adjusted arrow  
   
 const episodesContainer = document.createElement('div');  
 episodesContainer.className = 'episode-list-container hidden pl-4';  
 if (index === 0) {  
 seasonButton.classList.add('active');  
 episodesContainer.classList.remove('hidden');  
 await loadSeasonEpisodes(item.id, season.season\_number, episodesContainer, item.external\_ids?.imdb\_id);  
 }  
  
 seasonButton.addEventListener('click', async () => {  
 const isActive = seasonButton.classList.toggle('active');  
 seasonButton.querySelector('span:last-child').innerHTML = isActive ? '▾' : '▸'; // Adjusted arrow  
 episodesContainer.classList.toggle('hidden');  
 if (isActive && episodesContainer.innerHTML.includes('Carregando episódios...')) {   
 await loadSeasonEpisodes(item.id, season.season\_number, episodesContainer, item.external\_ids?.imdb\_id);  
 } else if (isActive && episodesContainer.innerHTML === '') { // Also load if empty  
 await loadSeasonEpisodes(item.id, season.season\_number, episodesContainer, item.external\_ids?.imdb\_id);  
 }  
 });  
 seasonDiv.appendChild(seasonButton);  
 seasonDiv.appendChild(episodesContainer);  
 seasonsAccordion.appendChild(seasonDiv);  
 });  
 }  
 }  
   
 const creativeSummaryBtn = document.getElementById('generateCreativeSummaryBtn');  
 if (creativeSummaryBtn) {  
 const creativeSummaryOutput = document.getElementById('creativeSummaryOutput');  
 creativeSummaryBtn.addEventListener('click', () => {  
 if (creativeSummaryOutput) creativeSummaryOutput.classList.remove('hidden');  
 fetchGeminiApi( `Você é um crítico de cinema divertido e criativo. Crie um resumo super original e envolvente para ${mediaType === 'movie' ? 'o filme' : 'a série'} "${title}". A sinopse oficial é: "${overview}". Surpreenda-me com sua criatividade, usando um tom leve e que desperte muita curiosidade! Não inclua saudações ou despedidas, vá direto ao resumo.`, creativeSummaryBtn, 'creativeSummaryOutput');  
 });  
 }  
 const movieNightBtn = document.getElementById('generateMovieNightBtn');  
 if (movieNightBtn) {  
 const movieNightOutput = document.getElementById('movieNightOutput');  
 movieNightBtn.addEventListener('click', () => {  
 if (movieNightOutput) movieNightOutput.classList.remove('hidden');  
 fetchGeminiApi( `Planeje uma noite temática incrível para assistir ${mediaType === 'movie' ? 'ao filme' : 'à série'} "${title}" (sinopse: "${overview}"). Sugira: 1. Comidas e petiscos temáticos. 2. Bebidas (alcoólicas e não alcoólicas) que combinem. 3. Ideias para criar uma atmosfera especial (decoração simples, música ambiente antes de começar, etc.). Formate a resposta com títulos claros para cada seção (Ex: 🍿 Petiscos Estelares, 🍹 Bebidas Galácticas, ✨ Atmosfera Mágica). Seja criativo e prático!`, movieNightBtn, 'movieNightOutput');  
 });  
 }  
 const triviaBtn = document.getElementById('generateTriviaBtn');   
 if (triviaBtn) {  
 const triviaOutput = document.getElementById('triviaOutput');  
 triviaBtn.addEventListener('click', () => {  
 if (triviaOutput) triviaOutput.classList.remove('hidden');  
 fetchGeminiApi( `Liste 3 a 5 curiosidades ou fatos interessantes sobre ${mediaType === 'movie' ? 'o filme' : 'a série'} "${title}" (cuja sinopse é: "${overview}"). Formate como uma lista de tópicos (bullet points). Seja conciso, informativo e divertido. Se não encontrar curiosidades específicas, pode mencionar aspectos notáveis da produção, recepção ou elenco.`, triviaBtn, 'triviaOutput');  
 });  
 }  
 }  
  
 async function loadSeasonEpisodes(tvId, seasonNumber, container, seriesImdbId) {  
 container.innerHTML = '<p class="text-sm text-gray-400 p-2">Carregando episódios...</p>';  
 const seasonData = await fetchSeasonDetails(tvId, seasonNumber);  
 if (seasonData && seasonData.episodes) {  
 container.innerHTML = '';   
 if (seasonData.episodes.length === 0) {  
 container.innerHTML = '<p class="text-sm text-gray-400 p-2">Nenhum episódio encontrado para esta temporada.</p>';  
 return;  
 }  
 seasonData.episodes.forEach(ep => {  
 const epButton = document.createElement('button');  
 epButton.className = 'episode-button';  
 epButton.textContent = `EP ${ep.episode\_number}: ${ep.name || 'Episódio sem título'}`;  
 epButton.addEventListener('click', () => {  
 openEpisodePlayerModal(seriesImdbId, seasonNumber, ep.episode\_number, tvId, `S${seasonNumber} E${ep.episode\_number}: ${ep.name || 'Episódio'}`);  
 });  
 container.appendChild(epButton);  
 });  
 } else {  
 container.innerHTML = '<p class="text-sm text-red-400 p-2">Falha ao carregar episódios.</p>';  
 }  
 }  
  
 async function openEpisodePlayerModal(seriesImdbId, seasonNumber, episodeNumber, seriesTmdbId, episodeTitleText) {  
 if (!seriesImdbId) {  
 showToast("ID da série não encontrado para o player.");  
 return;  
 }  
 if (episodePlayerTitleEl) episodePlayerTitleEl.textContent = episodeTitleText || `Temporada ${seasonNumber}, Episódio ${episodeNumber}`;  
   
 const episodeIframeContainer = document.getElementById('EpisodeEmbedderIframeContainer');  
 if (!episodeIframeContainer) return;  
  
 const episodePlayerMessageEl = document.createElement('div');   
 episodePlayerMessageEl.id = 'tempEpisodeMessage'; // Give it an ID to remove later  
 episodePlayerMessageEl.className = 'text-gray-400 mb-2 text-center';  
 episodeIframeContainer.innerHTML = '';   
 episodeIframeContainer.appendChild(episodePlayerMessageEl);  
   
 episodePlayerMessageEl.textContent = 'Verificando disponibilidade do episódio...';  
 currentEpisodeEmbedderUrl = '';   
  
 if (episodePlayerModal) episodePlayerModal.style.display = 'flex';  
 isEpisodeModalOpen = true;  
 if(isItemModalOpen && itemModal) itemModal.style.zIndex = '900';   
  
 const statusUrl = `https://embedder.net/api/status?tmdb=${seriesTmdbId}&sea=${seasonNumber}&epi=${episodeNumber}&type=tv`;  
 try {  
 const statusResponse = await fetch(statusUrl);  
 if (!statusResponse.ok) throw new Error(`Erro HTTP ${statusResponse.status}`);  
 const statusText = await statusResponse.text();  
 const isAvailable = statusText.trim().toLowerCase() === 'true';  
  
 // Remove temporary message before adding player or final message  
 const tempMsg = document.getElementById('tempEpisodeMessage');  
 if (tempMsg) tempMsg.remove();  
  
 if (isAvailable) {  
 // If message element was removed, re-create or just proceed to embed  
 embedderPlugin('serie', seriesImdbId, seasonNumber, episodeNumber, 'EpisodeEmbedderIframeContainer');  
 if(copyEpisodeLinkBtn) copyEpisodeLinkBtn.disabled = false;  
 } else {  
 episodeIframeContainer.innerHTML = `<p class="text-center text-gray-400 py-4">Episódio não disponível no Embedder.net.</p>`;  
 if(copyEpisodeLinkBtn) copyEpisodeLinkBtn.disabled = true;  
 }  
 } catch (error) {  
 console.error("Falha ao verificar status do episódio:", statusUrl, error);  
 const tempMsg = document.getElementById('tempEpisodeMessage');  
 if (tempMsg) tempMsg.remove(); // Ensure temp message is removed on error too  
  
 episodeIframeContainer.innerHTML = `<p class="text-center text-gray-400 py-4">Falha ao verificar (Erro: ${error.message}). <br>Tentando carregar player...</p>`;  
 embedderPlugin('serie', seriesImdbId, seasonNumber, episodeNumber, 'EpisodeEmbedderIframeContainer');  
 if(copyEpisodeLinkBtn) copyEpisodeLinkBtn.disabled = !(!currentEpisodeEmbedderUrl);  
 }  
 }  
   
 if(copyEpisodeLinkBtn) {  
 copyEpisodeLinkBtn.addEventListener('click', () => {  
 if (currentEpisodeEmbedderUrl) {  
 const textarea = document.createElement('textarea');  
 textarea.value = currentEpisodeEmbedderUrl;  
 document.body.appendChild(textarea);  
 textarea.select();  
 try { document.execCommand('copy'); showToast('Link do episódio copiado!'); }   
 catch (err) { showToast('Falha ao copiar.'); console.error('Falha ao copiar:', err); }  
 document.body.removeChild(textarea);  
 } else { showToast('Link do episódio não disponível.'); }  
 });  
 }  
  
 // --- Global Event Listeners ---  
 let searchTimeout;  
 if (searchInput) searchInput.addEventListener('input', (e) => {  
 clearTimeout(searchTimeout);  
 const query = e.target.value;  
 if (query.trim()) { searchTimeout = setTimeout(() => searchItems(query), 500); }   
 else { fetchTrending(); }  
 });  
  
 if (closeModalButton) closeModalButton.addEventListener('click', () => {  
 if (itemModal) itemModal.style.display = 'none';  
 if (modalBody) modalBody.innerHTML = '';   
 document.body.style.overflow = 'auto';  
 isItemModalOpen = false;  
 });  
 if (closeEpisodeModalButton) closeEpisodeModalButton.addEventListener('click', () => {  
 if (episodePlayerModal) episodePlayerModal.style.display = 'none';  
 const episodeIframeContainer = document.getElementById('EpisodeEmbedderIframeContainer');  
 if (episodeIframeContainer) episodeIframeContainer.innerHTML = '';   
 isEpisodeModalOpen = false;  
 if(isItemModalOpen && itemModal) itemModal.style.zIndex = '1000';   
 });  
  
 window.addEventListener('click', (event) => {  
 if (isEpisodeModalOpen && event.target === episodePlayerModal) {  
 if (closeEpisodeModalButton) closeEpisodeModalButton.click();  
 } else if (isItemModalOpen && event.target === itemModal && !isEpisodeModalOpen) {  
 if (closeModalButton) closeModalButton.click();  
 }  
 });  
 document.addEventListener('keydown', (event) => {  
 if (event.key === 'Escape') {  
 if (isEpisodeModalOpen) {  
 if (closeEpisodeModalButton) closeEpisodeModalButton.click();  
 } else if (isItemModalOpen) {  
 if (closeModalButton) closeModalButton.click();  
 }  
 }  
 });  
  
 if (btnExplore) btnExplore.addEventListener('click', fetchTrending);  
 if (btnMovies) btnMovies.addEventListener('click', fetchMovies);  
 if (btnSeries) btnSeries.addEventListener('click', fetchSeries);  
  
 document.addEventListener('DOMContentLoaded', fetchTrending);  
 </script>  
</body>  
</html>