// --- START OF FILE js/dataLoader.js ---

// Асинхронная функция для загрузки JSON данных из файла.

async function loadJSON(filePath) {

try {

// Выполняем запрос к указанному файлу.

const response = await fetch(filePath);

// Проверяем, успешен ли ответ.

if (!response.ok) {

// Если нет, выбрасываем ошибку.

throw new Error(`HTTP error! status: ${response.status}`);

}

// Парсим ответ как JSON.

const data = await response.json();

// Возвращаем полученные данные.

return data;

} catch (error) {

// В случае ошибки выводим ее в консоль.

console.error(`Could not load JSON from ${filePath}:`, error);

// Возвращаем null, чтобы указать на неудачу загрузки.

return null;

}

}

// --- END OF FILE js/dataLoader.js ---

Листинг 1 - Загрузка JSON данных

// --- START OF FILE js/commonLoader.js ---

// Событие DOMContentLoaded срабатывает, когда весь HTML был полностью загружен и разобран.

document.addEventListener('DOMContentLoaded', () => {

// Получаем элементы-плейсхолдеры для хедера и футера.

const headerPlaceholder = document.getElementById('header-placeholder');

const footerPlaceholder = document.getElementById('footer-placeholder');

// Асинхронная функция для загрузки HTML контента по URL и вставки его в указанный элемент.

const loadHTML = async (url, element) => {

// Если элемент-плейсхолдер не найден, ничего не делаем.

if (!element) return;

try {

// Загружаем HTML.

const response = await fetch(url);

if (!response.ok) {

throw new Error(`HTTP error! status: ${response.status} for ${url}`);

}

const html = await response.text();

// Вставляем загруженный HTML в элемент.

element.innerHTML = html;

// Если загружен хедер, инициализируем его логику.

// setTimeout используется для того, чтобы скрипты внутри \_header.html успели выполниться.

if (url.includes('\_header.html')) {

setTimeout(initializeHeaderLogic, 0);

}

} catch (error) {

// В случае ошибки выводим сообщение.

console.error(`Could not load HTML from ${url}:`, error);

element.innerHTML = `<p style="color:red; text-align: center; padding: 10px;">Не удалось загрузить компонент (${url}).</p>`;

}

};

// Параллельно загружаем хедер и футер.

Promise.all([

loadHTML('\_header.html', headerPlaceholder),

loadHTML('\_footer.html', footerPlaceholder)

]).catch(error => {

// Обработка общей ошибки загрузки компонентов.

console.error("Error loading header/footer components:", error);

});

});

// Функция для инициализации логики хедера (меню, скролл).

function initializeHeaderLogic() {

console.log('Attempting to initialize header logic...');

// Находим необходимые элементы хедера.

const header = document.getElementById('main-header');

const mobileMenuToggle = document.querySelector('.mobile-menu-toggle'); // Кнопка открытия/закрытия мобильного меню

const mobileMenuClose = document.querySelector('.mobile-menu-close'); // Кнопка закрытия внутри мобильного меню

const mobileMenu = document.querySelector('.mobile-menu'); // Само мобильное меню

// Проверки на наличие элементов, чтобы избежать ошибок.

if (!header) {

console.error("Header initialization failed: #main-header not found.");

return;

}

if (!mobileMenuToggle) {

console.error("Header initialization failed: .mobile-menu-toggle not found.");

}

if (!mobileMenu) {

console.error("Header initialization failed: .mobile-menu not found.");

}

if (!mobileMenuClose) {

console.error("Header initialization failed: .mobile-menu-close not found.");

}

console.log('Header elements:', { header, mobileMenuToggle, mobileMenu, mobileMenuClose });

// Логика изменения фона хедера при скролле.

function handleScroll() {

if (window.scrollY > 50) { // Если прокрутка больше 50px

header.classList.add('scrolled'); // Добавляем класс для стилизации

} else {

header.classList.remove('scrolled');

}

}

// Функция для переключения состояния мобильного меню.

function toggleMobileMenu() {

if (!mobileMenu || !mobileMenuToggle) return; // Если нет меню или кнопки - выходим

console.log('Toggling mobile menu');

mobileMenu.classList.toggle('open'); // Переключаем класс 'open' для показа/скрытия

// Блокируем/разблокируем скролл страницы при открытом/закрытом меню.

document.body.classList.toggle('no-scroll', mobileMenu.classList.contains('open'));

const isExpanded = mobileMenu.classList.contains('open');

mobileMenuToggle.setAttribute('aria-expanded', isExpanded); // Устанавливаем ARIA атрибут для доступности

}

// Добавляем слушатель события скролла.

window.addEventListener('scroll', handleScroll, { passive: true }); // passive: true для лучшей производительности

handleScroll(); // Вызываем сразу, чтобы применить стиль при загрузке, если страница уже прокручена.

// Если все элементы мобильного меню найдены, добавляем слушатели событий.

if (mobileMenuToggle && mobileMenu && mobileMenuClose) {

console.log('Adding menu toggle listeners.');

mobileMenuToggle.addEventListener('click', toggleMobileMenu); // Клик по "бургеру"

mobileMenuClose.addEventListener('click', toggleMobileMenu); // Клик по кнопке "закрыть"

// Добавляем слушатели на ссылки внутри мобильного меню, чтобы оно закрывалось при клике.

mobileMenu.querySelectorAll('a').forEach(link => {

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

if (mobileMenu.classList.contains('open')) {

console.log('Closing menu on link click');

toggleMobileMenu();

}

});

});

} else {

console.warn('Could not add menu toggle listeners because some elements were not found.');

}

console.log('Header logic initialization complete.');

}

// --- END OF FILE js/commonLoader.js ---

Листинг 2 - Загрузка шапки/подвала, инициализация шапки.

// --- START OF FILE js/slider.js ---

// Функция для настройки и инициализации слайдера.

function setupSlider(options) {

// Деструктуризация опций слайдера с значениями по умолчанию.

const {

containerSelector, // Селектор контейнера слайдера

slidesSelector, // Селектор обертки для слайдов

dotsSelector, // Селектор контейнера для точек (индикаторов)

prevButtonSelector, // Селектор кнопки "назад"

nextButtonSelector, // Селектор кнопки "вперед"

itemsPerView: initialItemsPerView = 1, // Количество видимых элементов (может быть функцией)

itemsToScroll = 1, // Количество элементов для прокрутки за раз

loop = false, // Бесконечная прокрутка

onSlideChange // Функция обратного вызова при смене слайда

} = options;

// Поиск основных элементов слайдера.

const sliderContainer = document.querySelector(containerSelector);

if (!sliderContainer) { console.error(`Container not found: ${containerSelector}`); return null; }

const slidesWrapper = sliderContainer.querySelector(slidesSelector);

const dotsContainer = sliderContainer.querySelector(dotsSelector);

const prevButton = sliderContainer.querySelector(prevButtonSelector);

const nextButton = sliderContainer.querySelector(nextButtonSelector);

// Проверка наличия обязательных элементов.

if (!slidesWrapper || !prevButton || !nextButton) { console.error(`Essential elements missing for "${containerSelector}"`); return null; }

let slides = Array.from(slidesWrapper.children); // Получаем массив слайдов

let totalItems = slides.length;

if (totalItems === 0) { console.warn(`No slides found for "${containerSelector}"`); return null; }

// Переменные состояния слайдера.

let currentIndex = 0; // Текущий индекс активного слайда

let slideWidth = 0; // Ширина одного слайда

let gap = 0; // Промежуток между слайдами

let currentItemsPerView = 1; // Текущее количество видимых элементов

let currentItemsToScroll = itemsToScroll;

let disableControls = false; // Флаг для отключения управления, если мало слайдов

let currentLoop = false; // Текущее состояние бесконечной прокрутки

let maxIndex = 0; // Максимально возможный индекс для прокрутки

let isTransitioning = false; // Флаг, указывающий, идет ли анимация перехода

// Функция для перехода к указанному слайду.

function goToSlide(index, animate = true) {

if (isTransitioning && animate) return; // Если идет анимация, не прерываем

if (totalItems === 0) return;

let requestedIndex = index;

let newIndex;

// Логика для бесконечной прокрутки.

if (currentLoop) {

if (requestedIndex < 0) newIndex = maxIndex;

else if (requestedIndex > maxIndex) newIndex = 0;

else newIndex = requestedIndex;

} else {

// Ограничиваем индекс в пределах доступных слайдов.

newIndex = Math.max(0, Math.min(requestedIndex, maxIndex));

if (newIndex === currentIndex && requestedIndex !== currentIndex) return; // Если индекс не изменился

}

const previousIndex = currentIndex;

currentIndex = newIndex;

// Вызов колбэка при смене слайда.

if (newIndex !== previousIndex && typeof onSlideChange === 'function') {

try {

onSlideChange(currentIndex);

} catch (e) {

console.error("Error in onSlideChange callback:", e);

}

}

// Расчет смещения для transform: translateX.

let offset = 0;

if (isFinite(slideWidth) && slideWidth > 0 && isFinite(gap)) {

offset = -currentIndex \* (slideWidth + gap);

} else {

console.warn(`Invalid values for offset calculation in "${containerSelector}"`);

}

// Применение смещения к обертке слайдов.

if (animate) {

isTransitioning = true;

slidesWrapper.style.transition = 'transform 0.5s ease-in-out';

slidesWrapper.removeEventListener('transitionend', onTransitionEnd);

slidesWrapper.addEventListener('transitionend', onTransitionEnd, { once: true });

} else {

slidesWrapper.style.transition = 'none'; // Отключаем анимацию для мгновенного перехода

}

slidesWrapper.style.transform = `translateX(${offset}px)`;

updateDots(); // Обновляем активную точку

updateButtons(); // Обновляем состояние кнопок

}

// Обработчик завершения анимации перехода.

function onTransitionEnd() {

isTransitioning = false;

}

// Функция для обновления размеров и параметров слайдера (например, при изменении размера окна).

function updateSliderDimensions() {

slides = Array.from(slidesWrapper.children);

totalItems = slides.length;

if (totalItems === 0) return;

// Получаем актуальное количество видимых элементов.

currentItemsPerView = (typeof initialItemsPerView === 'function') ? (parseInt(initialItemsPerView()) || 1) : (parseInt(initialItemsPerView) || 1);

currentItemsToScroll = parseInt(itemsToScroll) || 1;

if (currentItemsPerView <= 0) currentItemsPerView = 1;

if (currentItemsToScroll <= 0) currentItemsToScroll = 1;

// Определяем, нужно ли отключать управление.

disableControls = totalItems <= currentItemsPerView;

currentLoop = loop && !disableControls; // Бесконечная прокрутка активна, если разрешена и есть достаточно слайдов

maxIndex = Math.max(0, totalItems - currentItemsPerView); // Максимальный индекс

// Расчет ширины слайда и промежутка.

const containerWidth = sliderContainer.offsetWidth;

const computedStyle = window.getComputedStyle(slidesWrapper);

gap = parseFloat(computedStyle.gap) || 0; // Получаем значение CSS-свойства gap

if (currentItemsPerView <= 0) { slideWidth = 0; } else { slideWidth = (containerWidth - (gap \* (currentItemsPerView - 1))) / currentItemsPerView; }

if (!isFinite(slideWidth) || slideWidth <= 0) { slideWidth = 0; } // Проверка на валидность ширины

// Применяем ширину к каждому слайду.

slides.forEach(slide => {

if (slideWidth > 0) slide.style.width = `${slideWidth}px`;

else slide.style.width = `calc(${100 / currentItemsPerView}% - ${gap \* (currentItemsPerView - 1) / currentItemsPerView}px)`; // Fallback, если slideWidth не рассчитан

slide.style.flexShrink = '0'; // Запрещаем сжатие слайдов

});

// Показываем/скрываем кнопки в зависимости от disableControls.

if (prevButton) prevButton.style.display = disableControls ? 'none' : 'flex';

if (nextButton) nextButton.style.display = disableControls ? 'none' : 'flex';

// Корректируем текущий индекс, если он выходит за пределы нового maxIndex.

currentIndex = Math.max(0, Math.min(currentIndex, maxIndex));

goToSlide(currentIndex, false); // Переходим к текущему слайду без анимации

createDots(); // Создаем/обновляем точки

updateButtons(); // Обновляем состояние кнопок

}

// Функция для создания точек-индикаторов.

function createDots() {

if (!dotsContainer) return; // Если контейнера для точек нет

dotsContainer.innerHTML = ''; // Очищаем старые точки

if (disableControls) { dotsContainer.style.display = 'none'; return; } // Скрываем, если управление отключено

const numDots = maxIndex + 1; // Количество точек равно количеству "экранов" слайдов

if (numDots <= 1) { dotsContainer.style.display = 'none'; return; } // Если всего один "экран", точки не нужны

dotsContainer.style.display = 'flex';

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

const dot = document.createElement('button'); dot.className = 'dot';

dot.setAttribute('aria-label', `Перейти к слайду ${i + 1}`);

dot.addEventListener('click', () => goToSlide(i)); // Переход по клику на точку

dotsContainer.appendChild(dot);

}

updateDots(); // Устанавливаем активную точку

}

// Функция для обновления активной точки.

function updateDots() {

if (!dotsContainer || disableControls) return;

const dots = dotsContainer.querySelectorAll('.dot');

if (dots.length === 0) return;

dots.forEach((dot, index) => {

dot.classList.toggle('active', index === currentIndex);

});

}

// Функция для обновления состояния кнопок "назад/вперед" (активны/неактивны).

function updateButtons() {

if (!prevButton || !nextButton || disableControls) return;

if (currentLoop) {

// В режиме loop кнопки всегда активны.

prevButton.disabled = false; nextButton.disabled = false;

} else {

prevButton.disabled = currentIndex <= 0;

nextButton.disabled = currentIndex >= maxIndex;

}

}

console.log(`[slider.js "${containerSelector}"] Initialization started.`);

updateSliderDimensions(); // Первоначальный расчет размеров

// Добавление слушателей на кнопки "назад/вперед".

if (prevButton && !disableControls) { prevButton.addEventListener('click', () => { goToSlide(currentIndex - currentItemsToScroll); }); }

if (nextButton && !disableControls) { nextButton.addEventListener('click', () => { goToSlide(currentIndex + currentItemsToScroll); }); }

// Добавление слушателя на изменение размера окна для адаптивности слайдера.

let resizeTimer;

window.addEventListener('resize', () => {

clearTimeout(resizeTimer); // Для оптимизации, чтобы не вызывать updateSliderDimensions слишком часто

resizeTimer = setTimeout(() => {

console.log(`[slider.js "${containerSelector}"] Resize detected. Updating.`);

updateSliderDimensions();

}, 250);

});

// Сохраняем экземпляр слайдера на DOM-элементе для возможного внешнего управления.

sliderContainer.sliderInstance = {

goTo: goToSlide,

update: updateSliderDimensions,

getCurrentIndex: () => currentIndex

};

console.log(`[slider.js "${containerSelector}"] Initialization complete.`);

return sliderContainer.sliderInstance; // Возвращаем API слайдера

}

// --- END OF FILE js/slider.js ---

Листинг 3 - Создание слайдеров/каруселей

// --- START OF FILE js/home.js ---

// Логика для главной страницы.

document.addEventListener('DOMContentLoaded', async () => {

console.log("[home.js NEW STRUCTURE] DOMContentLoaded");

// Получение элементов для слайдера рекомендаций.

const recContainer = document.querySelector('.recommendations-slider-container');

const recSlidesWrapper = recContainer?.querySelector('.recommendations-slides'); // Обертка слайдов рекомендаций

// Элементы для отображения контента РЕКОМЕНДАЦИЙ (текст, описание, кнопка) НА ОВЕРЛЕЕ.

const recContentOverlay = document.getElementById('rec-content-overlay');

const recTitleOverlay = document.getElementById('rec-title-overlay');

const recDescriptionOverlay = document.getElementById('rec-description-overlay');

const recDetailsButtonOverlay = document.getElementById('rec-details-button-overlay');

// Элементы для отображения контента РЕКОМЕНДАЦИЙ ПОД СЛАЙДЕРОМ (для мобильных).

const recContentBelow = document.getElementById('rec-content-below');

const recTitleBelow = document.getElementById('rec-title-below');

const recDescriptionBelow = document.getElementById('rec-description-below');

const recDetailsButtonBelow = document.getElementById('rec-details-button-below'); // Не используется, но объявлен

// Получение элементов для слайдера новостей.

const newsContainer = document.querySelector('.news-slider-container');

const newsSlidesWrapper = newsContainer?.querySelector('.news-slides'); // Обертка слайдов новостей

// Проверка наличия критически важных элементов для слайдера рекомендаций.

if (!recContainer || !recSlidesWrapper || !recContentOverlay || !recContentBelow ||

!recTitleOverlay || !recDescriptionOverlay || !recDetailsButtonOverlay ||

!recTitleBelow || !recDescriptionBelow /\*|| !recDetailsButtonBelow\*/) { // recDetailsButtonBelow не используется для контента под слайдером

console.error("[home.js NEW STRUCTURE] Missing critical recommendation elements.");

const recSection = document.querySelector('section.recommendations');

if(recSection) recSection.style.display = 'none'; // Скрываем секцию, если что-то пошло не так

}

console.log("[home.js NEW STRUCTURE] Loading data...");

// Асинхронная загрузка данных о фильмах и новостях.

const movies = await loadJSON('data/movies.json');

const news = await loadJSON('data/news.json');

console.log("[home.js NEW STRUCTURE] Data loaded:", { movies: movies?.length, news: news?.length });

if (!movies) { console.error("[home.js NEW STRUCTURE] Movies failed to load"); }

if (!news) { console.error("[home.js NEW STRUCTURE] News failed to load"); }

// Фильтрация рекомендованных фильмов.

const recommendedMovies = movies ? movies.filter(m => m && m.isRecommended) : [];

console.log(`[home.js NEW STRUCTURE] Found ${recommendedMovies.length} recommended movies.`);

// Если есть рекомендованные фильмы и найдены элементы слайдера.

if (recommendedMovies.length > 0 && recSlidesWrapper) {

recSlidesWrapper.innerHTML = ''; // Очищаем существующие слайды.

// Создаем слайды для каждого рекомендованного фильма.

recommendedMovies.forEach((movie, index) => {

if (!movie || !movie.id) return; // Пропускаем, если нет данных или ID

const slide = document.createElement('div');

slide.className = 'slide';

// Сохраняем данные фильма в data-атрибутах для последующего использования.

slide.dataset.title = movie.title || '';

slide.dataset.description = movie.description || '';

slide.dataset.detailsUrl = `movie-details.html?id=${movie.id}`;

slide.dataset.index = index;

slide.innerHTML = `

<img src="${movie.heroImageUrl || movie.posterUrl || 'images/placeholder\_hero.jpg'}" alt="${movie.title || 'Рекомендованный фильм'}">

`;

recSlidesWrapper.appendChild(slide);

});

let isUpdatingContent = false; // Флаг для предотвращения многократного обновления контента

// Функция для обновления текстового контента рекомендаций при смене слайда.

const updateRecommendationContent = (index) => {

if (isUpdatingContent) return;

isUpdatingContent = true;

const currentSlide = recSlidesWrapper.children[index];

if (!currentSlide || !currentSlide.dataset) {

console.warn(`Slide data for index ${index} not found.`);

if(recContentOverlay) recContentOverlay.style.opacity = '0'; // Скрываем контент, если данных нет

if(recContentBelow) recContentBelow.style.opacity = '0';

isUpdatingContent = false;

return;

}

// Получаем данные из data-атрибутов текущего слайда.

const title = currentSlide.dataset.title || '';

const description = currentSlide.dataset.description || '';

const url = currentSlide.dataset.detailsUrl || '#';

// Плавное скрытие старого контента.

if(recContentOverlay) recContentOverlay.style.opacity = '0';

if(recContentBelow) recContentBelow.style.opacity = '0';

// Обновляем контент с небольшой задержкой для анимации.

setTimeout(() => {

// Обновляем контент на оверлее.

if(recTitleOverlay) recTitleOverlay.textContent = title;

if(recDescriptionOverlay) recDescriptionOverlay.textContent = description;

if(recDetailsButtonOverlay) recDetailsButtonOverlay.href = url;

// Обновляем контент под слайдером (для мобильных).

if(recTitleBelow) recTitleBelow.textContent = title;

if(recDescriptionBelow) recDescriptionBelow.textContent = description;

// if(recDetailsButtonBelow) recDetailsButtonBelow.href = url; // Кнопка "Подробнее" под слайдом не предусмотрена в HTML

// Плавное отображение нового контента.

if(recContentOverlay) recContentOverlay.style.opacity = '1';

if(recContentBelow) recContentBelow.style.opacity = '1';

console.log(`Updated content for slide ${index}: ${title}`);

isUpdatingContent = false;

}, 180); // Задержка соответствует transition в CSS

};

// Инициализация слайдера рекомендаций.

const sliderInstance = setupSlider({

containerSelector: '.recommendations-slider-container',

slidesSelector: '.recommendations-slides',

dotsSelector: '.recommendations-dots',

prevButtonSelector: '.rec-prev',

nextButtonSelector: '.rec-next',

itemsPerView: 1, // Всегда один слайд на весь экран

itemsToScroll: 1,

loop: recommendedMovies.length > 1, // Включаем loop, если фильмов больше одного

onSlideChange: updateRecommendationContent // Функция, вызываемая при смене слайда

});

// Обновляем контент для первого слайда при загрузке.

if (recommendedMovies.length > 0) {

updateRecommendationContent(0);

}

} else {

// Если нет рекомендованных фильмов, скрываем секцию.

console.log("[home.js NEW STRUCTURE] No recommended movies to display.");

const recSection = document.querySelector('section.recommendations');

if(recSection) recSection.style.display = 'none';

}

// Если есть новости и найдены элементы слайдера новостей.

if (news && news.length > 0 && newsSlidesWrapper) {

newsSlidesWrapper.innerHTML = ''; // Очищаем.

// Создаем слайды для каждой новости.

news.forEach(newsItem => {

if (!newsItem) return;

const slide = document.createElement('div');

slide.className = 'slide news-slide';

const imageUrl = newsItem.imageUrl || 'images/placeholder\_news.jpg';

const title = newsItem.title || 'Без заголовка';

const text = newsItem.text || '';

const date = newsItem.date ? new Date(newsItem.date).toLocaleDateString('ru-RU') : ''; // Форматируем дату

slide.innerHTML = `

<article class="news-item">

<img src="${imageUrl}" alt="${title}">

<div class="news-item-content">

<h3>${title}</h3>

<p>${text}</p>

<div class="news-item-date">${date}</div>

</div>

</article>`;

newsSlidesWrapper.appendChild(slide);

});

// Функция для определения количества отображаемых новостей в зависимости от ширины экрана.

const getNewsItemsToShow = () => {

const width = window.innerWidth;

if (width > 992) return 3; // Десктоп: 3 новости

if (width <= 768) return 1; // Мобильный: 1 новость

return 2; // Планшет: 2 новости

};

const itemsToShow = getNewsItemsToShow();

// Инициализация слайдера новостей.

setupSlider({

containerSelector: '.news-slider-container',

slidesSelector: '.news-slides',

dotsSelector: '.news-dots',

prevButtonSelector: '.news-prev',

nextButtonSelector: '.news-next',

itemsPerView: itemsToShow, // Количество видимых новостей

itemsToScroll: 1,

loop: news.length > itemsToShow // Loop, если новостей больше, чем отображается

});

// Обработчик изменения размера окна для переинициализации слайдера новостей

// с новым количеством видимых элементов.

let resizeTimerNews;

window.addEventListener('resize', () => {

clearTimeout(resizeTimerNews);

resizeTimerNews = setTimeout(() => {

const currentNewsContainer = document.querySelector('.news-slider-container');

// Проверяем, что слайдер был инициализирован

if (currentNewsContainer && currentNewsContainer.sliderInstance) {

console.log("[home.js NEW STRUCTURE] Resizing news slider");

const newItemsToShow = getNewsItemsToShow();

const currentNewsSlides = currentNewsContainer.querySelector('.news-slides');

// Сбрасываем transform, чтобы избежать проблем при переинициализации

if (currentNewsSlides) currentNewsSlides.style.transform = '';

// Переинициализируем слайдер с новыми параметрами

setupSlider({

containerSelector: '.news-slider-container',

slidesSelector: '.news-slides',

dotsSelector: '.news-dots',

prevButtonSelector: '.news-prev',

nextButtonSelector: '.news-next',

itemsPerView: newItemsToShow,

itemsToScroll: 1,

loop: news.length > newItemsToShow

});

}

}, 250);

});

} else {

// Если нет новостей, скрываем контейнер.

console.log("[home.js NEW STRUCTURE] No news to display.");

if (newsContainer) newsContainer.style.display = 'none';

}

console.log("[home.js NEW STRUCTURE] Script finished.");

});

// --- END OF FILE js/home.js ---

Листинг 4 - Логика главной страницы

// --- START OF FILE js/moviePage.js ---

// Логика для страницы детальной информации о фильме.

document.addEventListener('DOMContentLoaded', async () => {

// Получаем параметры из URL (ID фильма и, возможно, выбранную дату).

const urlParams = new URLSearchParams(window.location.search);

const movieId = urlParams.get('id'); // ID фильма

const selectedDateFromUrl = urlParams.get('date'); // Дата, если передана в URL

// Получение DOM-элементов страницы.

const mainContent = document.querySelector('.movie-detail-page');

const heroImage = document.getElementById('movie-hero-image');

const movieTitle = document.getElementById('movie-title');

const movieGenres = document.getElementById('movie-genres');

const movieRelease = document.getElementById('movie-release');

const movieDuration = document.getElementById('movie-duration');

const movieAge = document.getElementById('movie-age');

const directorInfoElement = document.querySelector('.director-info'); // Контейнер для информации о режиссере

const movieDirectorSpan = document.getElementById('movie-director'); // Элемент для имени режиссера

const movieDescription = document.getElementById('movie-description');

const playButton = document.getElementById('play-trailer-btn'); // Кнопка "Play" для трейлера

const trailerModal = document.getElementById('trailer-modal'); // Модальное окно для трейлера

const closeModalButton = document.getElementById('close-trailer-modal'); // Кнопка закрытия модального окна

const trailerIframe = document.getElementById('trailer-iframe'); // Iframe для встраивания трейлера

const scheduleSection = document.getElementById('movie-schedule'); // Секция с расписанием

const dateSelector = document.getElementById('movie-date-selector'); // Выпадающий список для выбора даты

const showtimesListContainer = document.getElementById('movie-showtimes-list'); // Контейнер для отображения сеансов

const scheduleControls = scheduleSection?.querySelector('.schedule-controls'); // Управление расписанием (селектор даты)

const noShowtimesMessage = scheduleSection?.querySelector('.no-showtimes-message'); // Сообщение "Нет сеансов на эту дату"

const noScheduleMessage = scheduleSection?.querySelector('.no-schedule-message'); // Сообщение "Для этого фильма нет расписания"

// Проверка наличия критически важных элементов.

if (!mainContent || !heroImage || !movieTitle || !movieGenres || !movieRelease || !movieDuration || !movieAge || !movieDescription ||

!scheduleSection || !dateSelector || !showtimesListContainer || !scheduleControls || !noShowtimesMessage || !noScheduleMessage) {

console.error("Критическая ошибка: Не найдены необходимые элементы на странице фильма или в секции расписания!");

if (mainContent) {

mainContent.innerHTML = `<p style="text-align: center; padding: 50px; color: red;">Ошибка загрузки страницы фильма. Пожалуйста, попробуйте позже.</p>`;

} else {

alert("Критическая ошибка загрузки страницы фильма.");

}

return;

}

// Проверка элементов трейлера (они могут отсутствовать, если трейлера нет).

if (!playButton || !trailerModal || !trailerIframe || !closeModalButton) {

console.warn("Элементы для трейлера могут отсутствовать.");

}

// Загрузка данных о фильмах.

const movies = await loadJSON('data/movies.json');

if (!movies) {

showError("Не удалось загрузить данные о фильмах. Расписание недоступно.");

return;

}

// Поиск текущего фильма по ID.

const movie = movies.find(m => m && m.id === movieId);

if (!movie) {

showError(`Ошибка: Фильм с ID "${movieId}" не найден.`);

return;

}

// Вспомогательные функции для установки текста и src изображений.

const setText = (element, text) => { if (element) element.textContent = text || ''; };

const setSrc = (element, src, alt) => {

if (element) {

const defaultSrc = element.dataset.defaultSrc || 'images/placeholder\_hero.jpg'; // Путь к заглушке

element.src = src || defaultSrc;

element.alt = alt || 'Изображение';

// Обработчик ошибки загрузки изображения.

element.onerror = () => {

console.warn(`Не удалось загрузить изображение: ${src}. Используется заглушка: ${defaultSrc}`);

element.src = defaultSrc;

};

}

};

// Функция для форматирования даты в удобочитаемый вид.

function formatDate(dateString) {

if (!dateString) return 'Неизвестно';

try {

// Проверка формата YYYY-MM-DD

if (!/^\d{4}-\d{2}-\d{2}$/.test(dateString)) return dateString; // Если формат неверный, возвращаем как есть

const date = new Date(dateString + 'T00:00:00'); // Добавляем время, чтобы избежать проблем с часовыми поясами

if (isNaN(date.getTime())) return dateString; // Проверка на валидность даты

return date.toLocaleDateString('ru-RU', { year: 'numeric', month: 'long', day: 'numeric' });

} catch (e) {

console.error("Ошибка форматирования даты:", e);

return dateString; // Возвращаем исходную строку в случае ошибки

}

}

// Функция для отображения сообщения об ошибке на странице.

function showError(message) {

console.error(message);

if (mainContent) {

mainContent.innerHTML = ''; // Очищаем основной контент

const errorP = document.createElement('p');

errorP.style.textAlign = 'center';

errorP.style.padding = '50px';

errorP.style.color = 'red';

errorP.textContent = message;

mainContent.appendChild(errorP);

if (scheduleSection) scheduleSection.style.display = 'none'; // Скрываем секцию расписания

} else {

alert(message);

}

}

// Функция для форматирования даты в короткий вид (например, "15 июля").

function formatShortDate(dateString) {

if (!dateString) return '';

try {

if (!/^\d{4}-\d{2}-\d{2}$/.test(dateString)) return dateString;

const date = new Date(dateString + 'T00:00:00');

if (isNaN(date.getTime())) return dateString;

return date.toLocaleDateString('ru-RU', { month: 'long', day: 'numeric' });

} catch (e) {

console.error("Ошибка форматирования короткой даты:", e);

return dateString;

}

}

// Заполнение информации о фильме на странице.

document.title = `${movie.title || 'Фильм'} - Сеть Кинотеатров`;

setSrc(heroImage, movie.heroImageUrl || movie.posterUrl, `Кадр из фильма ${movie.title || ''}`);

setText(movieTitle, movie.title);

setText(movieGenres, Array.isArray(movie.genres) ? movie.genres.join(', ') : 'Жанр не указан');

setText(movieRelease, `Премьера: ${formatDate(movie.releaseDate)}`);

setText(movieDuration, `Продолжительность: ${movie.duration ? movie.duration + ' мин.' : 'н/д'}`);

setText(movieAge, movie.ageRating || 'н/д');

// Отображение информации о режиссере, если она есть.

if (directorInfoElement) {

if (movieDirectorSpan && movie.director) {

setText(movieDirectorSpan, movie.director);

directorInfoElement.style.display = 'block';

} else {

directorInfoElement.style.display = 'none';

}

}

setText(movieDescription, movie.description);

let isUpdatingShowtimes = false; // Флаг для предотвращения одновременного обновления сеансов

// Функция для отображения сеансов на выбранную дату.

function displayShowtimes(selectedDate) {

if (isUpdatingShowtimes) return;

isUpdatingShowtimes = true;

console.log("Обновление времени для даты:", selectedDate);

if (!showtimesListContainer || !noShowtimesMessage) {

console.error("Не найдены элементы контейнера времени или сообщения 'нет сеансов'.");

isUpdatingShowtimes = false;

return;

}

// Плавное скрытие текущих сеансов/сообщения.

showtimesListContainer.classList.add('fade-out');

noShowtimesMessage.classList.add('fade-out');

setTimeout(() => { // Задержка для анимации скрытия

showtimesListContainer.innerHTML = ''; // Очищаем контейнер сеансов

// Получаем сеансы для выбранной даты.

const times = movie.showtimes && selectedDate && movie.showtimes[selectedDate];

console.log("Найденное время:", times);

if (times && Array.isArray(times) && times.length > 0) {

// Если сеансы есть, отображаем их.

noShowtimesMessage.style.display = 'none';

times.forEach(time => {

const timeElement = document.createElement('span');

timeElement.classList.add('time-slot');

timeElement.textContent = time;

showtimesListContainer.appendChild(timeElement);

});

showtimesListContainer.style.display = 'flex';

// requestAnimationFrame для плавной анимации появления

requestAnimationFrame(() => { requestAnimationFrame(() => {

showtimesListContainer.classList.remove('fade-out');

}); });

} else {

// Если сеансов нет, показываем соответствующее сообщение.

console.log("Сеансы не найдены, показываем сообщение.");

showtimesListContainer.style.display = 'none';

noShowtimesMessage.style.display = 'block';

requestAnimationFrame(() => { requestAnimationFrame(() => {

noShowtimesMessage.classList.remove('fade-out');

}); });

}

setTimeout(() => { isUpdatingShowtimes = false; }, 50); // Небольшая задержка перед сбросом флага

}, 300); // Длительность анимации fade-out

}

// Логика работы с расписанием.

if (movie.showtimes && typeof movie.showtimes === 'object' && Object.keys(movie.showtimes).length > 0) {

// Если у фильма есть расписание.

scheduleSection.style.display = 'block';

noScheduleMessage.style.display = 'none'; // Скрываем сообщение "нет расписания"

scheduleControls.style.display = 'flex'; // Показываем селектор дат

// Получаем и сортируем доступные даты.

const availableDates = Object.keys(movie.showtimes).sort((a, b) => new Date(a) - new Date(b));

console.log("Доступные даты:", availableDates);

dateSelector.innerHTML = ''; // Очищаем селектор.

// Заполняем селектор доступными датами.

availableDates.forEach(date => {

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

option.value = date;

option.textContent = formatShortDate(date); // Форматируем дату для отображения

dateSelector.appendChild(option);

});

// Определяем начальную дату для отображения сеансов.

let initialDate = '';

console.log("Дата из URL:", selectedDateFromUrl);

if (selectedDateFromUrl && availableDates.includes(selectedDateFromUrl)) {

// Если дата передана в URL и она доступна, используем ее.

initialDate = selectedDateFromUrl;

console.log("Используем дату из URL:", initialDate);

} else if (availableDates.length > 0) {

// Иначе используем первую доступную дату.

initialDate = availableDates[0];

console.log("Используем первую доступную дату:", initialDate);

} else {

console.log("Не удалось определить начальную дату.");

}

if (initialDate) {

dateSelector.value = initialDate; // Устанавливаем значение селектора

console.log("Установлено значение селектора:", dateSelector.value);

displayShowtimes(initialDate); // Отображаем сеансы для начальной даты

} else {

// Если не удалось определить дату (например, пустой объект showtimes), скрываем управление

scheduleControls.style.display = 'none';

displayShowtimes(null); // Это покажет "Нет сеансов"

}

// Добавляем слушатель на изменение даты в селекторе.

dateSelector.addEventListener('change', (event) => {

displayShowtimes(event.target.value);

});

} else {

// Если у фильма нет расписания.

scheduleSection.style.display = 'block';

scheduleControls.style.display = 'none'; // Скрываем селектор дат

showtimesListContainer.style.display = 'none'; // Скрываем контейнер сеансов

noScheduleMessage.style.display = 'block'; // Показываем сообщение "нет расписания"

console.log("Расписание для фильма отсутствует.");

}

// Функция для закрытия модального окна трейлера.

function closeModal() {

if (trailerModal && trailerIframe) {

trailerModal.classList.remove('open');

trailerIframe.src = ""; // Останавливаем воспроизведение видео

document.body.classList.remove('no-scroll'); // Разблокируем скролл страницы

}

}

// Логика работы с трейлером.

if (playButton && trailerModal && closeModalButton && trailerIframe && movie.trailerUrl) {

// Если все элементы трейлера и URL трейлера существуют.

playButton.style.display = 'flex'; // Показываем кнопку Play

playButton.style.opacity = '1';

playButton.style.visibility = 'visible';

// Клик по кнопке Play.

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

// Добавляем autoplay=1 к URL трейлера для автоматического воспроизведения.

const separator = movie.trailerUrl.includes('?') ? '&' : '?';

trailerIframe.src = `${movie.trailerUrl}${separator}autoplay=1`;

trailerModal.classList.add('open'); // Открываем модальное окно

document.body.classList.add('no-scroll'); // Блокируем скролл

});

// Клик по кнопке закрытия модального окна.

closeModalButton.addEventListener('click', closeModal);

// Закрытие модального окна при клике на фон (вне контента).

trailerModal.addEventListener('click', (event) => {

if (event.target === trailerModal) {

closeModal();

}

});

} else {

// Если функционал трейлера не может быть активирован.

if (playButton) {

playButton.style.display = 'none'; // Скрываем кнопку Play

}

console.log("Функционал трейлера отключен.");

}

});

// --- END OF FILE js/moviePage.js ---

Листинг 5 - Логика страницы фильма

// --- START OF FILE schedule.js ---

// Логика для страницы "Афиша" (расписание фильмов).

document.addEventListener('DOMContentLoaded', async () => {

console.log("[Schedule NEW] DOMContentLoaded");

// Получение DOM-элементов фильтров и сетки постеров.

const filtersContainer = document.querySelector('.filters'); // Общий контейнер фильтров

const toggleDesktopBtn = document.getElementById('filter-toggle-desktop-btn'); // Кнопка открытия/скрытия фильтров на десктопе

const panelDesktop = document.getElementById('filters-panel-desktop'); // Панель с фильтрами для десктопа

const toggleMobileBtn = document.querySelector('.filter-toggle-mobile'); // Кнопка открытия/скрытия фильтров на мобильных

const optionsMobile = document.getElementById('filters-options-mobile'); // Панель с фильтрами для мобильных

// Элементы фильтров для десктопа.

const dateFilterDesktop = document.getElementById('date-filter');

const genreFilterDesktop = document.getElementById('genre-filter');

const ageFilterDesktop = document.getElementById('age-filter');

const applyButtonDesktop = document.getElementById('apply-filters-desktop'); // Кнопка "Применить" для десктопа

// Элементы фильтров для мобильных.

const dateFilterMobile = document.getElementById('date-filter-mobile');

const genreFilterMobile = document.getElementById('genre-filter-mobile');

const ageFilterMobile = document.getElementById('age-filter-mobile');

const applyButtonMobile = document.getElementById('apply-filters-mobile'); // Кнопка "Применить" для мобильных

const posterGrid = document.querySelector('.poster-grid'); // Контейнер для отображения постеров фильмов

let lastAppliedDate = ''; // Хранит последнюю примененную дату для использования в ссылках на постеры

// Проверка наличия критически важных элементов.

if (!filtersContainer || !toggleDesktopBtn || !panelDesktop || !toggleMobileBtn || !optionsMobile || !applyButtonDesktop || !applyButtonMobile || !posterGrid) {

console.error("[Schedule NEW] Critical elements not found. Aborting.");

if(posterGrid) posterGrid.innerHTML = "<p>Ошибка инициализации фильтров.</p>";

return;

}

let allMovies = []; // Массив для хранения всех загруженных фильмов.

// Основная функция инициализации.

async function initialize() {

console.log("[Schedule NEW] Initializing...");

// Загрузка данных о фильмах.

const loadedMovies = await loadJSON('data/movies.json');

if (!loadedMovies) {

if(posterGrid) posterGrid.innerHTML = "<p>Не удалось загрузить список фильмов.</p>";

return;

}

allMovies = loadedMovies;

console.log(`[Schedule NEW] ${allMovies.length} movies loaded.`);

// Заполнение фильтров (жанры, дата по умолчанию).

if (genreFilterDesktop && genreFilterMobile) populateGenreFilter(allMovies);

if (dateFilterDesktop && dateFilterMobile) setDefaultDate();

// Устанавливаем lastAppliedDate из значения десктопного или мобильного фильтра даты

if (dateFilterDesktop) {

lastAppliedDate = dateFilterDesktop.value;

} else if (dateFilterMobile) {

lastAppliedDate = dateFilterMobile.value;

}

applyFilters(); // Первоначальное применение фильтров для отображения всех фильмов (или по дате по умолчанию).

setupListeners(); // Установка обработчиков событий.

console.log("[Schedule NEW] Initialization complete.");

}

// Установка обработчиков событий для кнопок и фильтров.

function setupListeners() {

toggleDesktopBtn.addEventListener('click', handleDesktopToggle); // Клик по кнопке десктопных фильтров

toggleMobileBtn.addEventListener('click', handleMobileToggle); // Клик по кнопке мобильных фильтров

applyButtonDesktop.addEventListener('click', () => handleApply('desktop')); // Применить десктопные фильтры

applyButtonMobile.addEventListener('click', () => handleApply('mobile')); // Применить мобильные фильтры

// Закрытие десктопной панели фильтров при клике вне ее.

document.addEventListener('click', handleClickOutside);

panelDesktop.addEventListener('click', (event) => event.stopPropagation()); // Предотвращаем закрытие при клике внутри панели

console.log("[Schedule NEW] Event listeners attached.");

}

// Обработчик для кнопки переключения десктопных фильтров.

function handleDesktopToggle(event) {

event.stopPropagation(); // Предотвращаем всплытие события, чтобы не сработал handleClickOutside

console.log("[Schedule NEW] Desktop toggle clicked.");

if (optionsMobile.classList.contains('is-visible')) closeMobileFilters(); // Закрываем мобильные, если открыты

const isOpen = panelDesktop.classList.toggle('is-visible'); // Переключаем видимость панели

toggleDesktopBtn.classList.toggle('active', isOpen); // Стиль для активной кнопки

toggleDesktopBtn.setAttribute('aria-expanded', String(isOpen)); // ARIA атрибут

}

// Обработчик для кнопки переключения мобильных фильтров.

function handleMobileToggle() {

console.log("[Schedule NEW] Mobile toggle clicked.");

if (panelDesktop.classList.contains('is-visible')) closeDesktopFilters(); // Закрываем десктопные, если открыты

const isMobileOpen = optionsMobile.classList.toggle('is-visible'); // Переключаем видимость

toggleMobileBtn.setAttribute('aria-expanded', String(isMobileOpen));

}

// Обработчик для кнопок "Применить фильтры".

function handleApply(source) { // source: 'desktop' или 'mobile'

console.log(`[Schedule NEW] Apply clicked from ${source}`);

const dateInput = (source === 'mobile') ? dateFilterMobile : dateFilterDesktop;

if (dateInput) {

lastAppliedDate = dateInput.value; // Обновляем последнюю примененную дату

console.log("Updating lastAppliedDate to:", lastAppliedDate);

}

applyFilters(source); // Применяем фильтры

// Закрываем панель фильтров после применения (если она была открыта).

if (source === 'desktop' && panelDesktop.classList.contains('is-visible')) {

// Можно не закрывать десктопную панель, чтобы пользователь мог изменить фильтры еще раз

// closeDesktopFilters();

} else if (source === 'mobile' && optionsMobile.classList.contains('is-visible')) {

closeMobileFilters();

}

}

// Обработчик для закрытия десктопной панели фильтров при клике вне ее.

function handleClickOutside(event) {

if (panelDesktop.classList.contains('is-visible') &&

!panelDesktop.contains(event.target) && // Клик не внутри панели

!toggleDesktopBtn.contains(event.target)) // Клик не по кнопке открытия

{

console.log("[Schedule NEW] Closing desktop panel on outside click.");

closeDesktopFilters();

}

}

// Функции для закрытия панелей фильтров.

function closeDesktopFilters() {

panelDesktop.classList.remove('is-visible');

toggleDesktopBtn.classList.remove('active');

toggleDesktopBtn.setAttribute('aria-expanded', 'false');

}

function closeMobileFilters() {

optionsMobile.classList.remove('is-visible');

toggleMobileBtn.setAttribute('aria-expanded', 'false');

}

// Заполнение выпадающего списка жанров.

function populateGenreFilter(movies) {

const genreSelects = [genreFilterDesktop, genreFilterMobile];

if (!genreSelects[0] || !genreSelects[1]) return; // Если селекторы не найдены

const genres = new Set(); // Используем Set для получения уникальных жанров.

movies.forEach(movie => { if(movie && Array.isArray(movie.genres)) movie.genres.forEach(genre => genres.add(genre)); });

genreSelects.forEach(select => {

while (select.options.length > 1) select.remove(1); // Очищаем старые опции (кроме "Все жанры")

Array.from(genres).sort().forEach(genre => { // Сортируем жанры по алфавиту

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

option.value = genre; option.textContent = genre;

select.appendChild(option);

});

});

console.log("[Schedule NEW] Genres populated.");

}

// Установка даты по умолчанию (сегодня) и ограничений для выбора даты.

function setDefaultDate() {

const dateInputs = [dateFilterDesktop, dateFilterMobile];

if (!dateInputs[0] || !dateInputs[1]) return;

const today = new Date();

const year = today.getFullYear();

const month = String(today.getMonth() + 1).padStart(2, '0'); // Месяцы от 0 до 11

const day = String(today.getDate()).padStart(2, '0');

const todayStr = `${year}-${month}-${day}`; // Формат YYYY-MM-DD

// Устанавливаем максимальную дату (например, на месяц вперед).

const maxDate = new Date();

maxDate.setMonth(maxDate.getMonth() + 1);

const maxYear = maxDate.getFullYear();

const maxMonth = String(maxDate.getMonth() + 1).padStart(2, '0');

const maxDay = String(maxDate.getDate()).padStart(2, '0');

const maxDateStr = `${maxYear}-${maxMonth}-${maxDay}`;

dateInputs.forEach(input => {

input.value = todayStr; // Дата по умолчанию - сегодня

input.min = todayStr; // Минимальная дата - сегодня

input.max = maxDateStr; // Максимальная дата

});

lastAppliedDate = todayStr; // Устанавливаем начальное значение

console.log("[Schedule NEW] Date set. Initial lastAppliedDate:", lastAppliedDate);

}

// Функция для отображения отфильтрованных постеров.

function renderPosters(moviesToRender) {

if (!posterGrid) return;

posterGrid.innerHTML = ''; // Очищаем сетку.

const currentDateForLink = lastAppliedDate; // Используем последнюю примененную дату для ссылок

console.log("Using date for links:", currentDateForLink);

if (!Array.isArray(moviesToRender) || moviesToRender.length === 0) {

// Если фильмов по запросу нет.

posterGrid.innerHTML = '<p>По вашему запросу фильмов не найдено.</p>';

return;

}

moviesToRender.forEach(movie => {

if (!movie || !movie.id) return;

const posterElement = document.createElement('a'); // Постер - это ссылка

posterElement.classList.add('poster-item');

// Формируем URL для страницы фильма, добавляя выбранную дату, если она есть.

let movieUrl = `movie-details.html?id=${movie.id}`;

if (currentDateForLink) {

movieUrl += `&date=${currentDateForLink}`;

}

posterElement.href = movieUrl;

posterElement.innerHTML = `

<img src="${movie.posterUrl || 'images/placeholder\_poster.jpg'}" alt="Постер фильма ${movie.title || ''}" loading="lazy">

<div class="poster-overlay">

<span class="poster-details-button">Подробнее</span>

</div>

`;

posterGrid.appendChild(posterElement);

});

console.log(`[Schedule NEW] Rendered ${moviesToRender.length} posters.`);

}

// Функция применения фильтров.

function applyFilters(source = 'initial') { // source: 'desktop', 'mobile', или 'initial' (при первой загрузке)

console.log(`--- Applying filters (source: ${source}) ---`);

let selectedDate, selectedGenre, selectedAge;

// Получаем значения из соответствующих фильтров в зависимости от источника.

if (source !== 'initial') {

const dateInput = (source === 'mobile') ? dateFilterMobile : dateFilterDesktop;

const genreSelect = (source === 'mobile') ? genreFilterMobile : genreFilterDesktop;

const ageSelect = (source === 'mobile') ? ageFilterMobile : ageFilterDesktop;

if (!dateInput || !genreSelect || !ageSelect) {

console.error("Filter elements not found for source:", source);

return;

}

selectedDate = dateInput.value;

selectedGenre = genreSelect.value;

selectedAge = ageSelect.value;

lastAppliedDate = selectedDate; // Обновляем здесь, так как это фактическое применение

} else {

// При начальной загрузке используем значения по умолчанию или из lastAppliedDate.

selectedDate = lastAppliedDate;

selectedGenre = genreFilterDesktop ? genreFilterDesktop.value : 'all'; // Значение по умолчанию 'all'

selectedAge = ageFilterDesktop ? ageFilterDesktop.value : 'all'; // Значение по умолчанию 'all'

}

console.log("Filtering with values:", { date: selectedDate, genre: selectedGenre, age: selectedAge });

if (!Array.isArray(allMovies)) {

console.error("Movies data is not available for filtering.");

return;

}

// Фильтрация массива всех фильмов.

const filteredMovies = allMovies.filter(movie => {

if (!movie) return false;

// Проверка наличия сеансов на выбранную дату.

const hasShowtimeOnDate = selectedDate

? movie.showtimes && movie.showtimes[selectedDate] && movie.showtimes[selectedDate].length > 0

: true; // Если дата не выбрана (хотя у нас всегда выбрана), считаем, что подходит

// Проверка соответствия жанра.

const genreMatch = selectedGenre === 'all' || (movie.genres && movie.genres.includes(selectedGenre));

// Проверка соответствия возрастного рейтинга.

const parseAge = (ageString) => parseInt(String(ageString).replace(/\D/g, ''), 10) || 0; // Извлекаем число из строки типа "12+"

const movieAgeNum = movie.ageRating ? parseAge(movie.ageRating) : 0;

const selectedAgeNum = selectedAge === 'all' ? 999 : parseAge(selectedAge); // 'all' соответствует любому возрасту

const ageMatch = movieAgeNum <= selectedAgeNum; // Фильм подходит, если его рейтинг не выше выбранного

return hasShowtimeOnDate && genreMatch && ageMatch;

});

console.log(`Found ${filteredMovies.length} movies after filtering.`);

renderPosters(filteredMovies); // Отображаем отфильтрованные фильмы.

}

initialize(); // Запускаем инициализацию.

});

// --- END OF FILE schedule.js ---

Листинг 6 - Логика афиши