[ВВЕДЕНИЕ 4](#_Toc153187783)

[1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ 5](#_Toc153187784)

[2 ПРОГРАММНОЕ ПРОЕКТИРОВАНИЕ 6](#_Toc153187785)

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 9](#_Toc153187786)

[3.1 Детальная реализация функциональных частей ПО 9](#_Toc153187787)

[3.2 Сопроводительная документация 23](#_Toc153187788)

[3.3 Анализ ПО 23](#_Toc153187789)

[4 МЕТОДИКА И РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ 24](#_Toc153187790)

[ЗАКЛЮЧЕНИЕ 25](#_Toc153187791)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26](#_Toc153187792)

[ПРИЛОЖЕНИЕ А 27](#_Toc153187793)

[ПРИЛОЖЕНИЕ б 30](#_Toc153187794)

[ПРИЛОЖЕНИЕ В 31](#_Toc153187795)

# ВВЕДЕНИЕ

В современном мире, где интернет-технологии становятся неотъемлемой частью нашей повседневной жизни, роль цифровых платформ в развлечениях значительно возросла. Одной из наиболее динамично развивающихся сфер является индустрия онлайн-кинотеатров. Данный курсовой проект посвящен разработке клиент-серверного онлайн кинотеатра, который предоставляет пользователям возможность смотреть актуальные фильмы, выбирать кинопроизведения по жанрам, искать интересующие их фильмы, а также просматривать трейлеры.

Серверная часть прокта строится на Node JS и фреймворке Express JS. В качестве СУБД был выбран MySQL. Взаимодействие серверной и клиентской части строится на архитектуре REST API.

Проект онлайн-кинотеатра включает в себя браузерную и десктопную версию. Реализация десктопной версии осуществлялась с использованием JavaScript фреймворка Electron. Он является одним из мощных, доступных и популярных фреймворков для создания нативных приложений. Создание веб-версии осуществлялось с использованием современных веб-технологий: HTML, CSS и JavaScript.

Этот подход позволил создать гибкую и функциональную платформу, способную обрабатывать запросы пользователей в режиме реального времени и предоставлять им качественный и удобный интерфейс просмотра фильмов.

Целью проекта является не только предоставление доступа к широкому спектру кинопродукции, но и создание удобного и интуитивно понятного пользовательского интерфейса, который бы удовлетворял потребности современных пользователей в быстром и удобном доступе к информации о любых фильмах, а также новинкам киноиндустрии.

Проект рассчитан на широкую аудиторию пользователей, ценящих качественный и быстрый доступ к кинопродукции через интернет. Он включает в себя функциональные возможности, такие как поиск по названию, выбор фильмов по жанрам, просмотр актуальных новинок, а также возможность просмотра трейлеров.

# 1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ

На первом этапе разработки проекта был проведен тщательный анализ и поиск информации об аналогах. Изучение существующих онлайн кинотеатров, их функциональности и популярности – это важный этап разработки собственного приложения. Также был проведен анализ рыночных тенденций и потребительских предпочтений в области онлайн-просмотра фильмов.

Была дана оценка конкурентоспособности и выделение уникальных особенностей существующих платформ, таких как качество контента, скорость загрузки, удобство интерфейса.

Одним из важнейших этапов анализа исходных данных был анализ существующих технологий веб-разработки, их возможностей и ограничений для реализации проекта.

Также были изучены правовые аспекты: законодательство, касательно авторских прав и распространения медиаконтента в интернете, анализ требований к лицензированию контента и возможных правовых рисков, связанных с предоставлением фильмов и сериалов в онлайн кинотеатре.

Для пользовательской части был использовал HTML (Язык гипертекстовой разметки), CSS (Каскадные таблицы стилей) с препроцессором SASS, а также язык программирования JavaScript.

Для серверной было решено использовать Node.js, а также фреймворк Express. Это обосновано их актуальностью, наличием обширной документации, а так же наличием множества приложений с использованием данных технологий.

Для создания десктопного приложения был использован JavaScript фреймворк Electron.

# 2 ПРОГРАММНОЕ ПРОЕКТИРОВАНИЕ

Программное проектирование клиент-серверного онлайн кинотеатра охватывает комплексный подход к разработке архитектуры веб-приложения, дизайну пользовательского интерфейса и выбору технологий для реализации ключевых функций. Основной акцент проекта – обеспечение высокой производительности, удобства использования и эффективного управления медиаконтентом.

Клиентская часть разработана с использованием HTML, CSS и JavaScript, Electron JS, обеспечивая динамичный и отзывчивый пользовательский интерфейс. Серверная часть реализована на Node.js с использованием фреймворка Express, что позволяет эффективно обрабатывать запросы пользователей и управлять потоком данных. Структура приложения включает в себя следующие основные компоненты:

− API: этот модуль включает в себя функцию API запроса к серверу. После получения данных, с ними можно работать в сторонних функциях приложения.

− Detail: отображает детали фильма. Также отображает рекомендации к текущему фильму.

− Global: включает в себя функции, необходимые для разработки и сокращения кода.

− Index: отображает главную страницу приложения. Также показывает пользователю список самых популярных и просматриваемых фильмов.

− Movie-card: заполняет прокручиваемый блок фильмов на главной странице для удобной навигации пользователя по фильмам.

− Search: содержит функции для поиска фильмов по названию.

− Sidebar: содержит функции для бокового меню приложения.

Дизайн интерфейса пользователя разработан с акцентом на простоту и интуитивность. Также стоит отметить, что одной из важнейших составляющих современных приложений является доступность для различных экранов и устройств. Поэтому одной из главных задач в разработке было обеспечить кроссбраузерность, а также адаптивность для мобильных устройств, планшетов и телевизоров. Верстка приложения основана на принципах «гибкой верстки» и «адаптивной верстки». Основные характеристики дизайна включают

− Современный и минималистичный дизайн: Чистые линии, простота элементов управления и сдержанная цветовая гамма.

− Адаптивность: Интерфейс оптимизирован для комфортного просмотра на различных устройствах, от мобильных телефонов до настольных компьютеров.

− Интуитивные элементы управления: Удобное управление воспроизведением, легкодоступные функции поиска и категоризации.

Для реализации проекта были выбраны следующие технологии:

− HTML, CSS и JavaScript: Используются для создания интерактивного и адаптивного фронтенда.

− Node.js и Express: Обеспечивают мощную и гибкую серверную платформу для обработки запросов и управления контентом.

− База данных MySQL: Для хранения информации о фильмах, и их жанрах.

В процессе разработки особое внимание уделялось визуализаци ключевых элементов интерфейса:

− Главная страница: Демонстрация актуальных и популярных фильмов, включая навигационные элементы и поиск. Пример дизайна главного экрана представлен на рисунке 2.1.

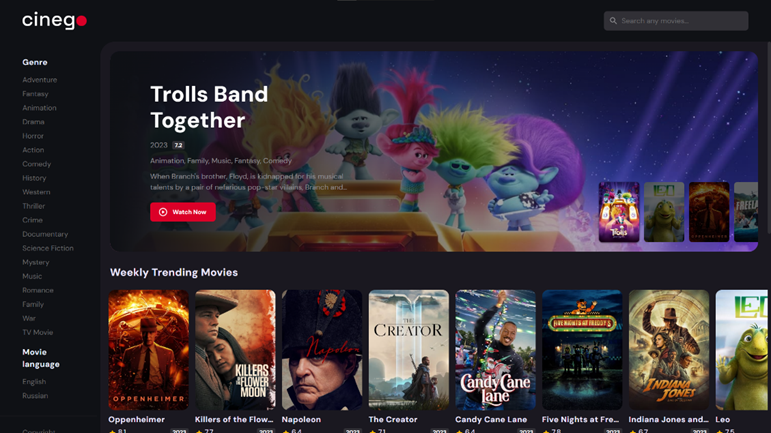


Рисунок 2.1 – Главный экран онлайн-кинотеатра «Cinego»

− Страницы категорий: Удобный интерфейс для фильтрации и выбора фильмов по различным критериям. Пример интерфейса страницы категорий представлен на рисунке 2.2.

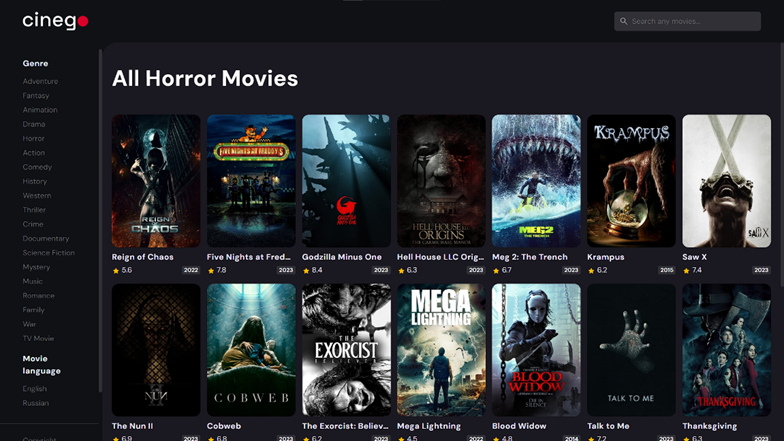


Рисунок 2.2 – Экран категорий онлайн-кинотеатра «Cinego»

− Страница фильма: Включает детальную информацию о фильме, варианты воспроизведения и трейлеры. Пример интерфейса страницы фильма представлен на рисунке 2.3

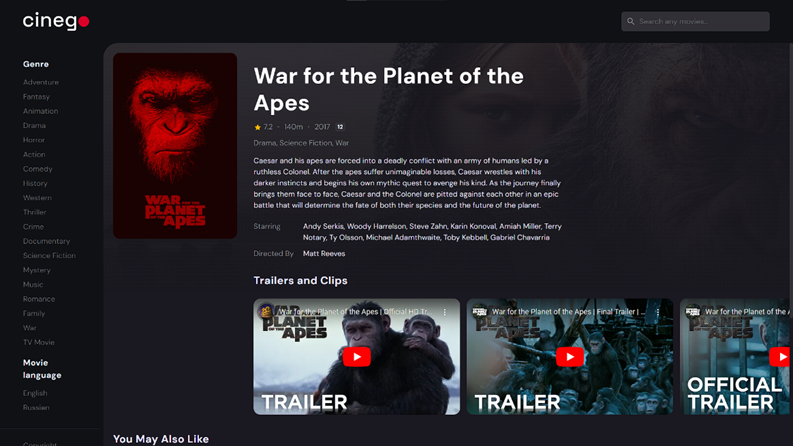


Рисунок 2.3 – Экран фильма онлайн-кинотеатра «Cinego»

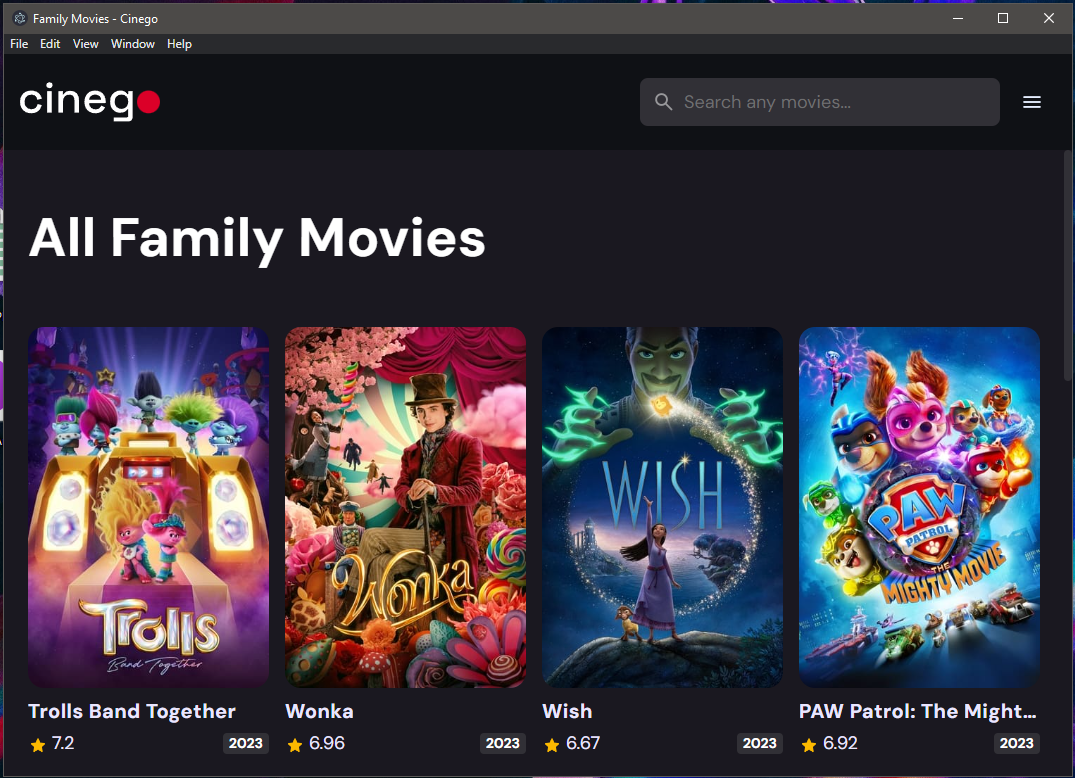
− Нативная версия онлайн-кинотеатра по функционалу ничем не отличается от браузерной версии. Пример интерфейса нативной (десктопной) версии приложения представлен на рисунке 2.4

Рисунок 2.4 – Десктопная версия онлайн-кинотеатра

# 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

## **3.1 Детальная реализация функциональных частей ПО**

API: этот модуль включает в себя функцию API запроса к серверу. После получения данных, с ними можно работать в сторонних функциях приложения.

− Detail: отображает детали фильма. Также отображает рекомендации к текущему фильму.

− Global: включает в себя функции, необходимые для разработки и сокращения кода.

− Index: отображает главную страницу приложения. Также показывает пользователю список самых популярных и просматриваемых фильмов.

− Movie-card: заполняет прокручиваемый блок фильмов на главной странице для удобной навигации пользователя по фильмам.

− Search: содержит функции для поиска фильмов по названию.

− Sidebar: содержит функции для бокового меню приложения.

В этом разделе представлено описание основных компонентов клиент-серверного приложения «Cinego», разработанного на базе фреймворка Express. Основные функциональные части включают модуль api запроса, отображения деталей фильма, реализации

Модуль воспроизведения медиа отвечает за загрузку и обработку аудио и видео файлов. Реализация этого модуля демонстрируется в листинге 3.1, где настроены основные параметры приложения и создан экземпляр QQmlApplicationEngine.

Листинг 3.1 – Функция API запроса (api.js)

'use strict';

const api\_key = 'd53cd63350aa29d1b3f4dad57de14485';

const imageBaseURL = '/t/p/';

const fetchDataFromServer = function (url, callback, optionalParam) {

fetch(url)

.then(response => response.json())

.then(data => {

callback(data, optionalParam);

});

}

export { imageBaseURL, api\_key, fetchDataFromServer };

Далее представлен пример модуля отображения деталей фильма, представлен в листинге 3.2.

Листинг 3.2 – Отображение деталей фильма (detail.js)

'use strict';

import { api\_key, imageBaseURL, fetchDataFromServer } from "./api.js";

import { sidebar } from "./sidebar.js";

import { createMovieCard } from "./movie-card.js";

import { search } from "./search.js";

const movieId = window.localStorage.getItem("movieId");

const pageContent = document.querySelector("[page-content]");

sidebar();

const getGenres = function (genreList) {

const newGenreList = [];

for (const { name } of genreList) newGenreList.push(name);

return newGenreList.join(", ");

}

const getCasts = function (castList) {

const newCastList = [];

for (let i = 0, len = castList.length; i < len && i < 10; i++) {

const { name } = castList[i];

newCastList.push(name);

}

return newCastList.join(", ");

}

const getDirectors = function (crewList) {

const directors = crewList.filter(({ job }) => job === "Director");

const directorList = [];

for (const { name } of directors) directorList.push(name);

return directorList.join(", ");

}

// returns only trailers and teasers as array

const filterVideos = function (videoList) {

return videoList.filter(({ type, site }) => (type === "Trailer" || type === "Teaser") && site === "YouTube");

}

fetchDataFromServer(`https://127.0.0.1/3/movie/${movieId}?api\_key=${api\_key}&append\_to\_response=casts,videos,images,releases`, function (movie) {

Продолжение листинга 3.2

const {

backdrop\_path,

poster\_path,

title,

release\_date,

runtime,

vote\_average,

releases: { countries: [{ certification } = { certification: "N/A" }] },

genres,

overview,

casts: { cast, crew },

videos: { results: videos }

} = movie;

document.title = `${title} - Cinego`;

const movieDetail = document.createElement("div");

movieDetail.classList.add("movie-detail");

movieDetail.innerHTML = `

<div class="backdrop-image" style="background-image: url('${imageBaseURL}${"w1280" || "original"}${backdrop\_path || poster\_path}')"></div>

<figure class="poster-box movie-poster">

<img src="${imageBaseURL}w342${poster\_path}" alt="${title} poster" class="img-cover">

</figure>

<div class="detail-box">

<div class="detail-content">

<h1 class="heading">${title}</h1>

<div class="meta-list">

<div class="meta-item">

<img src="./assets/images/star.png" width="20" height="20" alt="rating">

<span class="span">${vote\_average.toFixed(1)}</span>

</div>

<div class="separator"></div>

<div class="meta-item">${runtime}m</div>

Продолжение листинга 3.2

<div class="separator"></div>

<div class="meta-item">${release\_date?.split("-")[0] ?? "Not Released"}</div>

<div class="meta-item card-badge">${certification}</div>

</div>

<p class="genre">${getGenres(genres)}</p>

<p class="overview">${overview}</p>

<ul class="detail-list">

<div class="list-item">

<p class="list-name">Starring</p>

<p>${getCasts(cast)}</p>

</div>

<div class="list-item">

<p class="list-name">Directed By</p>

<p>${getDirectors(crew)}</p>

</div>

</ul>

</div>

<div class="title-wrapper">

<h3 class="title-large">Trailers and Clips</h3>

</div>

<div class="slider-list">

<div class="slider-inner"></div>

</div>

</div>

`;

for (const { key, name } of filterVideos(videos)) {

const videoCard = document.createElement("div");

videoCard.classList.add("video-card");

videoCard.innerHTML = `

<iframe width="500" height="294" src="https://www.youtube.com/embed/${key}?&theme=dark&color=white&rel=0"

Продолжение листинга 3.2

frameborder="0" allowfullscreen="1" title="${name}" class="img-cover" loading="lazy"></iframe>

`;

movieDetail.querySelector(".slider-inner").appendChild(videoCard);

}

pageContent.appendChild(movieDetail);

fetchDataFromServer(`https://127.0.0.1/3/movie/${movieId}/recommendations?api\_key=${api\_key}&page=1`, addSuggestedMovies);

});

const addSuggestedMovies = function ({ results: movieList }, title) {

const movieListElem = document.createElement("section");

movieListElem.classList.add("movie-list");

movieListElem.ariaLabel = "You May Also Like";

movieListElem.innerHTML = `

<div class="title-wrapper">

<h3 class="title-large">You May Also Like</h3>

</div>

<div class="slider-list">

<div class="slider-inner"></div>

</div>

`;

for (const movie of movieList) {

const movieCard = createMovieCard(movie); // called from movie\_card.js

movieListElem.querySelector(".slider-inner").appendChild(movieCard);

}

pageContent.appendChild(movieListElem);

}

search();

Модуль global предоставляет основные функции для разработки и сокращения кода. Пример реализации модуля global представлен в листинге 3.3.

Листинг 3.3 – Глобальный модуль (global.js)

'use strict';

const addEventOnElements = function (elements, eventType, callback) {

for (const elem of elements) elem.addEventListener(eventType, callback);

}

const searchBox = document.querySelector("[search-box]");

const searchTogglers = document.querySelectorAll("[search-toggler]");

addEventOnElements(searchTogglers, "click", function () {

searchBox.classList.toggle("active");

});

const getMovieDetail = function (movieId) {

window.localStorage.setItem("movieId", String(movieId));

}

const getMovieList = function (urlParam, genreName) {

window.localStorage.setItem("urlParam", urlParam);

window.localStorage.setItem("genreName", genreName);

}

Модуль index собирает главную страницу приложения. Пример реализации модуля index представлен в листинге 3.4.

Листинг 3.4 – Главная страница (index.js)

'use strict';

import { sidebar } from "./sidebar.js";

import { api\_key, imageBaseURL, fetchDataFromServer } from "./api.js";

import { createMovieCard } from "./movie-card.js";

import { search } from "./search.js";

const pageContent = document.querySelector("[page-content]");

sidebar();

const homePageSections = [

{

title: "Upcoming Movies",

Продолжение листинга 3.4

path: "/movie/upcoming"

},

{

title: "Weekly Trending Movies",

path: "/trending/movie/week"

},

{

title: "Top Rated Movies",

path: "/movie/top\_rated"

}

]

const genreList = {

// create genre string from genre\_id eg: [23, 43] -> "Action, Romance".

asString(genreIdList) {

let newGenreList = [];

for (const genreId of genreIdList) {

this[genreId] && newGenreList.push(this[genreId]); // this == genreList;

}

return newGenreList.join(", ");

}

};

fetchDataFromServer(`https://127.0.0.1/3/genre/movie/list?api\_key=${api\_key}`, function ({ genres }) {

for (const { id, name } of genres) {

genreList[id] = name;

}

fetchDataFromServer(`https://127.0.0.1/3/movie/popular?api\_key=${api\_key}&page=1`, heroBanner);

});

const heroBanner = function ({ results: movieList }) {

const banner = document.createElement("section");

banner.classList.add("banner");

banner.ariaLabel = "Popular Movies";

banner.innerHTML = `

<div class="banner-slider"></div>

<div class="slider-control">

<div class="control-inner"></div>

</div>

Продолжение листинга 3.4

let controlItemIndex = 0;

for (const [index, movie] of movieList.entries()) {

const {

backdrop\_path,

title,

release\_date,

genre\_ids,

overview,

poster\_path,

vote\_average,

id

} = movie;

const sliderItem = document.createElement("div");

sliderItem.classList.add("slider-item");

sliderItem.setAttribute("slider-item", "");

sliderItem.innerHTML = `

<img src="${imageBaseURL}w1280${backdrop\_path}" alt="${title}" class="img-cover" loading=${index === 0 ? "eager" : "lazy"

}>

<div class="banner-content">

<h2 class="heading">${title}</h2>

<div class="meta-list">

<div class="meta-item">${release\_date?.split("-")[0] ?? "Not Released"}</div>

<div class="meta-item card-badge">${vote\_average.toFixed(1)}</div>

</div>

<p class="genre">${genreList.asString(genre\_ids)}</p>

<p class="banner-text">${overview}</p>

<a href="./detail.html" class="btn" onclick="getMovieDetail(${id})">

<img src="./assets/images/play\_circle.png" width="24" height="24" aria-hidden="true" alt="play circle">

<span class="span">Watch Now</span>

</a>

</div>

`;

banner.querySelector(".banner-slider").appendChild(sliderItem);

Продолжение листинга 3.4

const controlItem = document.createElement("button");

controlItem.classList.add("poster-box", "slider-item");

controlItem.setAttribute("slider-control", `${controlItemIndex}`);

controlItemIndex++;

controlItem.innerHTML = `

<img src="${imageBaseURL}w154${poster\_path}" alt="Slide to ${title}" loading="lazy" draggable="false" class="img-cover">

`;

banner.querySelector(".control-inner").appendChild(controlItem);

}

pageContent.appendChild(banner);

addHeroSlide();

for (const { title, path } of homePageSections) {

fetchDataFromServer(`https://127.0.0.1/3${path}?api\_key=${api\_key}&page=1`, createMovieList, title);

}

}

const addHeroSlide = function () {

const sliderItems = document.querySelectorAll("[slider-item]");

const sliderControls = document.querySelectorAll("[slider-control]");

let lastSliderItem = sliderItems[0];

let lastSliderControl = sliderControls[0];

lastSliderItem.classList.add("active");

lastSliderControl.classList.add("active");

const sliderStart = function () {

lastSliderItem.classList.remove("active");

lastSliderControl.classList.remove("active");

// `this` == slider-control

sliderItems[Number(this.getAttribute("slider-control"))].classList.add("active");

this.classList.add("active");

lastSliderItem = sliderItems[Number(this.getAttribute("slider-control"))];

Продолжение листинга 3.4

lastSliderControl = this;

}

addEventOnElements(sliderControls, "click", sliderStart);

}

const createMovieList = function ({ results: movieList }, title) {

const movieListElem = document.createElement("section");

movieListElem.classList.add("movie-list");

movieListElem.ariaLabel = `${title}`;

movieListElem.innerHTML = `

<div class="title-wrapper">

<h3 class="title-large">${title}</h3>

</div>

<div class="slider-list">

<div class="slider-inner"></div>

</div>

`;

for (const movie of movieList) {

const movieCard = createMovieCard(movie); // called from movie\_card.js

movieListElem.querySelector(".slider-inner").appendChild(movieCard);

}

pageContent.appendChild(movieListElem);

}

search();

Модуль movie-list собирает страницу получения фильмов по какой-либо категории. Пример реализации модуля movie-list представлен в листинге 3.5.

Листинг 3.5 – Получение фильмов (movie-list.js)

'use strict';

import { api\_key, fetchDataFromServer } from "./api.js";

import { sidebar } from "./sidebar.js";

import { createMovieCard } from "./movie-card.js";

import { search } from "./search.js";

// collectc genre name & url parameter from local storage

const genreName = window.localStorage.getItem("genreName");

const urlParam = window.localStorage.getItem("urlParam");

const pageContent = document.querySelector("[page-content]");

sidebar();

let currentPage = 1;

let totalPages = 0;

Продолжение листинга 3.5

fetchDataFromServer(`https://127.0.0.1/3/discover/movie?api\_key=${api\_key}&sort\_by=popularity.desc&include\_adult=false&page=${currentPage}&${urlParam}`, function ({ results: movieList, total\_pages }) {

totalPages = total\_pages;

document.title = `${genreName} Movies - Cinego`;

const movieListElem = document.createElement("section");

movieListElem.classList.add("movie-list", "genre-list");

movieListElem.ariaLabel = `${genreName} Movies`;

movieListElem.innerHTML = `

<div class="title-wrapper">

<h1 class="heading">All ${genreName} Movies</h1>

</div>

<div class="grid-list"></div>

<button class="btn load-more" load-more>Load More</button>

`;

for (const movie of movieList) {

const movieCard = createMovieCard(movie);

movieListElem.querySelector(".grid-list").appendChild(movieCard);

}

pageContent.appendChild(movieListElem);

document.querySelector("[load-more]").addEventListener("click", function () {

if (currentPage >= totalPages) {

this.style.display = "none"; // this == loading-btn

return;

}

currentPage++;

this.classList.add("loading"); // this == loading-btn

fetchDataFromServer(`https://127.0.0.1/3/discover/movie?api\_key=${api\_key}&sort\_by=popularity.desc&include\_adult=false&page=${currentPage}&${urlParam}`, ({ results: movieList }) => {

this.classList.remove("loading"); // this == loading-btn

for (const movie of movieList) {

const movieCard = createMovieCard(movie);

movieListElem.querySelector(".grid-list").appendChild(movieCard);

Продолжение листинга 3.5

}

});

});

});

search();

Модуль search предоставляет функции для поиска фильма по названию. Пример реализации модуля search представлен в листинге 3.6.

Листинг 3.6 – Поиск фильма по названию (search.js)

'use strict';

import { api\_key, fetchDataFromServer } from "./api.js";

import { createMovieCard } from "./movie-card.js";

export function search() {

const searchWrapper = document.querySelector("[search-wrapper]");

const searchField = document.querySelector("[search-field]");

const searchResultModal = document.createElement("div");

searchResultModal.classList.add("search-modal");

document.querySelector("main").appendChild(searchResultModal);

let searchTimeout;

searchField.addEventListener("input", function () {

if (!searchField.value.trim()) {

searchResultModal.classList.remove("active");

searchWrapper.classList.remove("searching");

clearTimeout(searchTimeout);

return;

}

searchWrapper.classList.add("searching");

clearTimeout(searchTimeout);

searchTimeout = setTimeout(function () {

fetchDataFromServer(`https://127.0.0.1/3/search/movie?api\_key=${api\_key}&query=${searchField.value}&page=1&include\_adult=false`, function ({ results: movieList }) {

searchWrapper.classList.remove("searching");

searchResultModal.classList.add("active");

searchResultModal.innerHTML = ""; // remove old results

searchResultModal.innerHTML = `

<p class="label">Results for</p>

Продолжение листинга 3.6

<h1 class="heading">${searchField.value}</h1>

<div class="movie-list">

<div class="grid-list"></div>

</div>

`;

for (const movie of movieList) {

const movieCard = createMovieCard(movie);

searchResultModal.querySelector(".grid-list").appendChild(movieCard);

}

});

}, 500);

});

}

Модуль sidebar предоставляет основные функции для бокового меню приложения. Пример реализации модуля sidebar представлен в листинге 3.7.

Листинг 3.7 – Боковое меню (sidebar.js)

'use strict';

import { api\_key, fetchDataFromServer } from "./api.js";

export function sidebar() {

const genreList = {};

fetchDataFromServer(`https://127.0.0.1/3/genre/movie/list?api\_key=${api\_key}`, function ({ genres }) {

for (const { id, name } of genres) {

genreList[id] = name;

}

genreLink();

});

const sidebarInner = document.createElement("div");

sidebarInner.classList.add("sidebar-inner");

sidebarInner.innerHTML = `

<div class="sidebar-list">

<p class="title">Genre</p>

</div>

<div class="sidebar-list">

<p class="title">Movie language</p>

<a href="./movie-list.html" menu-close class="sidebar-link"

onclick='getMovieList("with\_original\_language=en", "English")'>English</a>

<a href="./movie-list.html" menu-close class="sidebar-link"

Продолжение листинга 3.7

onclick='getMovieList("with\_original\_language=ru", "Russian")'>Russian</a>

</div>

<div class="sidebar-footer">

<p class="copyright">

Copyright 2023 <a href="https://www.instagram.com/aucuo/" target="\_blank">aucuo</a>

</p>

<img src="./assets/images/logo.svg" width="80" alt="the movie database logo">

</div>

`;

const genreLink = function () {

for (const [genreId, genreName] of Object.entries(genreList)) {

const link = document.createElement("a");

link.classList.add("sidebar-link");

link.setAttribute("href", "./movie-list.html");

link.setAttribute("menu-close", "");

link.setAttribute("onclick", `getMovieList("with\_genres=${genreId}", "${genreName}")`);

link.textContent = genreName;

sidebarInner.querySelectorAll(".sidebar-list")[0].appendChild(link);

}

const sidebar = document.querySelector("[sidebar]");

sidebar.appendChild(sidebarInner);

toggleSidebar(sidebar);

}

const toggleSidebar = function (sidebar) {

const sidebarMenuBtn = document.querySelector("[menu-btn]");

const sidebarMenuTogglers = document.querySelectorAll("[menu-toggler]");

const sidebarMenuClose = document.querySelectorAll("[menu-close]");

const overlay = document.querySelector("[overlay]");

addEventOnElements(sidebarMenuTogglers, "click", function () {

sidebar.classList.toggle("active");

sidebarMenuBtn.classList.toggle("active");

overlay.classList.toggle("active");

});

addEventOnElements(sidebarMenuClose, "click", function () {

sidebar.classList.remove("active");

sidebarMenuBtn.classList.remove("active");

overlay.classList.remove("active");

});

}

## **3.2 Сопроводительная документация**

Сопроводительная документация по разработанному программному продукту предоставляется в составе технического задания (приложение А) согласно ГОСТ 19.201-78.

Требования к сопроводительной документации устанавливаются государственными стандартами ЕСПД.

## **3.3 Анализ ПО**

Анализ программного обеспечения клиент-серверного онлайн кинотеатра уделяет внимание следующим ключевым аспектам:

− Производительность и оптимизация: оценка эффективности использования серверных и клиентских ресурсов. Оптимизация обеспечение быстрой загрузки контента и плавного воспроизведения видео в различных качествах без значительных задержек. Совместимость с различными браузерами и операционными системами.

− Интуитивный пользовательский интерфейс: исследование показало, что использование HTML, CSS и JavaScript способствует созданию гибкого и понятного интерфейса, делая приложение доступным для разных категорий пользователей. Оценка удобства навигации и доступности основных функций, таких как поиск фильмов, выбор жанров и просмотр трейлеров.

− Функциональность платформы: анализ возможностей поиска и категоризации фильмов, проверка эффективности алгоритмов фильтрации. Оценка качества передачи данных.

# 4 МЕТОДИКА И РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Тестирование программного обеспечения включало следующие ключевые аспекты:

* Запуск приложения: проверка инициализации и запуска приложения.
* Проверка получения запросов и его правильного отображения на страницах приложения.
* Проверка работоспособности всех страниц и функций приложения.
* Работа интерфейса управления: проверка функционирования элементов управления: выполнение поиска, корректность отображения всех элементов и их правильность, актуальность всех данных.
* Тестирование адаптивности интерфейса: проверка корректного отображения приложения на всех типах экранов.
* Стабильность приложения: проверка работы приложения в различных условиях.

Пример адаптивности интерфейса представлен на рисунке 3.1.

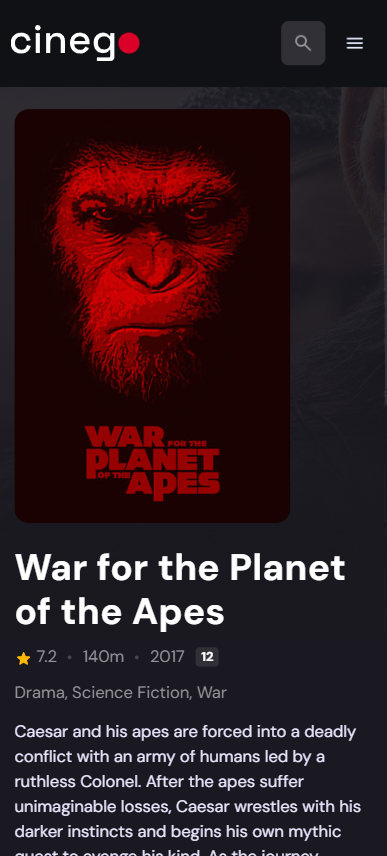


Рисунок 3.1 – Пример адаптации интерфейса под экран мобильного устройства

# ЗАКЛЮЧЕНИЕ

В ходе работы над курсовым проектом "Клиент-серверный онлайн кинотеатр", было создано полноценное приложение, предназначенное для просмотра медиаконтента онлайн. Проект был реализован с использованием современных веб-технологий, включая HTML, CSS, JavaScript для фронтенда Node.js и Express для серверной части, , а также Electron для создания нативной части приложения, что обеспечило высокую функциональность и удобство использования.

В процессе разработки были освоены важные навыки программирования, а также глубокое понимание клиент-серверных взаимодействий и передачи медиаданных. Была достигнута цель создания интуитивно понятного, отзывчивого и адаптивного пользовательского интерфейса, который обеспечивает удобный доступ к широкому спектру фильмов и сериалов.

Ключевой особенностью проекта стало создание эффективной системы поиска и категоризации контента, собственная нативная версия, а также интеграция возможности просмотра трейлеров, что значительно улучшило пользовательский опыт. Приложение поддерживает различные форматы видео и обеспечивает высококачественное воспроизведение контента.

В процессе тестирования приложения были выявлены и успешно исправлены некоторые технические недостатки, что способствовало повышению его стабильности и надежности. В результате, приложение демонстрирует высокую производительность и надежность, соответствуя всем заявленным требованиям.

Таким образом, разработанный клиент-серверный онлайн кинотеатр представляет собой эффективное и удобное решение, сочетающее в себе передовые технологии веб-разработки и функциональный интерфейс. Это делает его привлекательным для широкого круга пользователей, которые ищут качественное и удобное средство для просмотра фильмов онлайн.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Express - Node.js web application framework [Электронный ресурс] – Режим доступа: https://expressjs.com/. Дата доступа: 01.11.2023.

2 Node.js v21.4.0 documentation [Электронный ресурс] – Режим доступа: https://nodejs.org/docs/latest/api/. Дата доступа: 05.11.2023.

3 SASS Docs [Электронный ресурс] – Режим доступа: https://sass-lang.com/documentation/. Дата доступа: 02.10.2023.

4 MDN web docs JavaScript [Электронный ресурс] – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/JavaScript. Дата доступа: 02.10.2023.

5 IBM API article about [Электронный ресурс] – Режим доступа: https://www.ibm.com/topics/api. Дата доступа: 01.10.2023.

6 Client-server model [Электронный ресурс] – Режим доступа: https://www.techtarget.com/searchnetworking/definition/client-server#:~:text=Client%2Dserver%20is%20a%20relationship,computing%20model%20used%20by%20mainframes. Дата доступа: 02.10.2023.

7 Electron JS Docs [Электронный ресурс] – Режим доступа: https://www.electronjs.org/ru/docs/latest/. Дата доступа: 14.10.2023.

# ПРИЛОЖЕНИЕ А

(обязательное)

**Техническое задание**

**Введение**

Наименование программного продукта – «Онлайн-кинотеатр». Данное приложение разработано для получения актуальной информации о фильмах.

**А.1 Основание для разработки**

Программа разрабатывается в рамках курсового проекта студента учебного заведения «Полоцкий государственный университет имени Ефросинии Полоцкой». Основанием для разработки является задание курсового проекта.

**А.2 Назначение разработки**

Функциональное и эксплуатационное назначение программы – обеспечение пользователей удобным и функциональным средством для поиска фильмов и просмотра медиаконтента.

**А.3 Требования к программному продукту**

**А.3.1 Требования к функциональным характеристикам**

Для программы «Онлайн-кинотеатр» выдвигаются следующие требования:

− Адаптивность: Интерфейс оптимизирован для комфортного просмотра на различных устройствах, от мобильных телефонов до настольных компьютеров.

− Интуитивные элементы управления: Удобное управление воспроизведением, легкодоступные функции поиска и категоризации.

**А.3.2 Требования к надежности**

* Устойчивость к возможным сбоям и ошибкам.
* Стабильность отображения контента

**А.3.3 Условия эксплуатации**

Эксплуатация программы должна осуществляться на серверах, поддерживающих использование MySQL и Node.js. Пользователи должны иметь базовые навыки работы с компьютером, операционной системой и веб-браузерами.

**А.3.4 Требования к составу и параметрам технических средств**

Минимальные системные требования для стабильной работы приложения:

1. Операционная система: Windows 7, Windows 8/8.1, Windows 10 или более поздняя версия
2. Операционная система: macOS 10.10 или более поздняя версия
3. Операционная система: популярные дистрибутивы Linux, такие как Ubuntu, Fedora, и др.
4. Оперативная память: 2 ГБ или более
5. Процессор: Intel Pentium 4 или выше
6. Свободное место на жестком диске: около 200 МБ

**А.3.5 Требования к информационной и программной совместимости**

Программное средство должно удовлетворять следующему требованию: ОС Windows 7 и выше.

Компьютер должен иметь доступ к сети.

**А.3.6 Требования к маркировке и упаковке**

Требования к маркировке и упаковке отсутствуют.

**А.3.7 Требования к транспортированию и хранению**

Программное средство должно храниться на электронном носителе в

виде исполняемого файла.

**А.4 Требования к программной документации**

Программная документация по приложению должна быть предоставлена в следующем составе:

1 техническое задание. Согласно ГОСТ 19.201-78;

2 пояснительная записка. Согласно ГОСТ 19.101-77.

Требования к перечисленным программным документам устанавли-ваются государственными стандартами ЕСПД.

**А.5 Стадии и этапы разработки**

1. Анализ исходных данных, постановка задачи проектирования, разработка технического задания.
2. Разработка архитектуры, интерфейса и структуры программы.
3. Реализация, тестирование программы.
4. Разработка программной документации.

**А.6 Порядок контроля и приемки**

Контроль и приемка программы осуществляются по результатам тестирования функциональности и стабильности работы. Методы испытаний включают визуальный контроль и выполнение тестовых сценариев для проверки соответствия функциональности техническому заданию.

# ПРИЛОЖЕНИЕ б

(обязательное)

**Диаграмма вариантов использования**

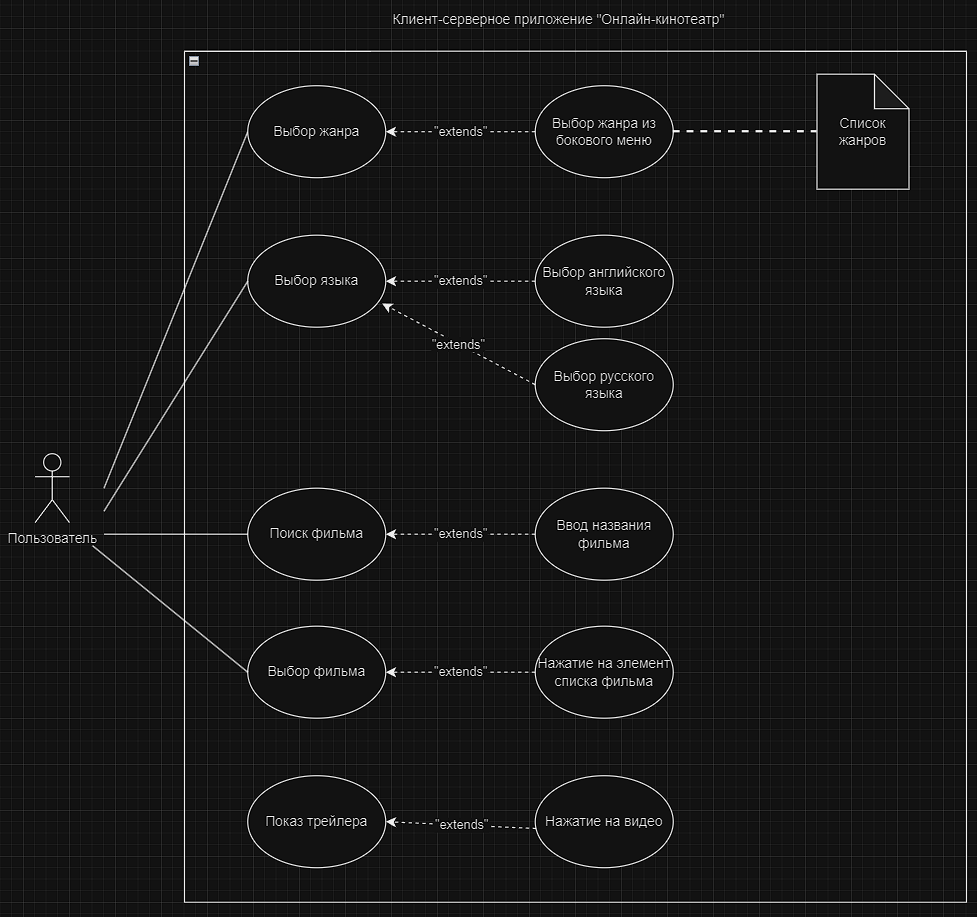
****

Рисунок Б.1 – Диаграмма вариантов использования

# ПРИЛОЖЕНИЕ В

(обязательное)

**Таблица тестирования**

Таблица В.1 – Способы проверок с указанием ожидаемых результатов испытаний

|  |  |  |  |
| --- | --- | --- | --- |
| Тестовый вариант | Входные данные | Ожидаемый результат | Результат тестирования |
| Запуск онлайн-кинотеатра | Вход в программу | Открытие главной страницы кинотеатра с доступными функциями управления | Тест пройден успешно |
| Показ деталей выбранного фильма | Нажатие на картинку фильма | Перенаправление на страницу нужного фильма и получение данных | Тест пройден успешно |
| Выбор категории фильма | Нажатие на название категории | Перенаправление на страницу нужной категории и получение списка фильмов | Тест пройден успешно |
| Поиск фильма | Нажатие на поле поиска и ввод данных | Выполнение поиска нужного фильма по имени и вывод результата | Тест пройден успешно |
| Показ трейлера | Нажатие на видео | Проигрывание видео-материала | Тест пройден успешно |
| Выход из программы | Закрытие вкладки приложения | Закрытие программы | Тест пройден успешно |