минобрнауки россии

федеральное государственное бюджетное образовательное учреждение

высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

|  |  |
| --- | --- |
| Институт (факультет) | Институт Информационных Технологий |
| Кафедра | Математического и Программного Обеспечения ЭВМ |

**ПРОЕКТ**

|  |
| --- |
| по дисциплине Мобильное программирование |

|  |  |
| --- | --- |
| на тему | Разработка мобильного приложения «КиноВыбор» |

|  |
| --- |
| Выполнили студенты группы |
| 1ПИб-02-3оп-22 |
| направление подготовки (специальности) |
| 09.03.04, Программная инженерия |
| *шифр, наименование* |
| Маркелов Сергей Александрович |
| *фамилия, имя, отчество* |
| Подтелков Владислав Владимирович |
| *фамилия, имя, отчество* |

|  |
| --- |
| Руководитель |
| Селяничев Олег Леонидович |
| *фамилия, имя, отчество* |
| доцент, кандидат технических наук |
| *должность* |

|  |
| --- |
| Дата представления работы |
| «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г. |
|  |
| Заключение о допуске к защите |
|  |
|  |
|  |
|  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| количество баллов |
| Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Череповец, 2025

*Год*

Оглавление

[Введение 3](#_Toc213629418)

[1. Описание предметной области 4](#_Toc213629419)

[2. Архитектура приложения 5](#_Toc213629420)

[2.1. Общая архитектура 5](#_Toc213629421)

[2.2. Логика работы 5](#_Toc213629422)

[3. Средства разработки 11](#_Toc213629423)

[4. Мобильное приложение 13](#_Toc213629424)

[4.1. Разработка дизайна мобильного приложения 13](#_Toc213629425)

[4.2. Описание логики 19](#_Toc213629426)

[4.3. Реализация мобильного приложения 20](#_Toc213629427)

[5. Сервер 25](#_Toc213629428)

[5.1. Разработка сервера 25](#_Toc213629429)

[5.2. Соединение сервера и мобильного приложения 28](#_Toc213629430)

[6. Тестирование 31](#_Toc213629431)

[6.1. Тестирование сервера 31](#_Toc213629432)

[6.2. Тестирование мобильного приложения 33](#_Toc213629433)

[Заключение 36](#_Toc213629434)

[Список литературы 37](#_Toc213629435)

[Приложение 38](#_Toc213629436)

[П.1. Мобильное приложение 38](#_Toc213629437)

[П.2. Сервер 83](#_Toc213629438)

# Введение

В настоящее время совместный просмотр фильмов остаётся одним из самых популярных способов отдыха среди друзей и членов семьи. Но процесс выбора фильма часто превращается в затяжное обсуждение, когда у каждого свои предпочтения, а договориться бывает непросто. В итоге на сам просмотр остаётся меньше времени и желания.

В сфере развлечений выявлена проблема отсутствия удобных инструментов, позволяющих быстро и без споров выбрать фильм, который понравится всем участникам. Для её решения необходимо разработать мобильное приложение, которое сможет учитывать предпочтения пользователей, автоматически подбирать подходящие варианты и предоставлять удобный интерфейс для совместного выбора.

На данный момент существующие онлайн-кинотеатры и стриминговые сервисы не предоставляют возможности коллективного выбора контента. Пользователям приходится обсуждать варианты на ходу, что делает процесс длительным и неэффективным.

Решением данной проблемы является разработка мобильного приложения, предназначенного для выбора фильма в группе и без споров. Приложение должно предоставлять интуитивно понятный интерфейс, в котором пользователи смогут оценивать предлагаемые фильмы, а система с помощью алгоритма подбора определит наиболее подходящий вариант для всех участников.

Целью данного проекта является разработка мобильного приложения, предназначенного для автоматизации процесса выбора фильма для совместного просмотра. Приложение должно обеспечивать коллективное взаимодействие пользователей, хранение их предпочтений и справедливый алгоритм подбора фильма. Реализация данного проекта позволит сделать процесс выбора фильма быстрым, прозрачным и комфортным для всех участников.

Разработка приложения выполняется в команде из двух человек:

* Маркелов Сергей – frontend-разработчик. Отвечает за создание макета, разработку интерфейса, проведение тестирования, ведение документации
* Подтелков Владислав – backend-разработчик. Отвечает за сбор данных, разработку сервера и клиента, интеграцию API

# Описание предметной области

Совместный просмотр фильмов является одной из наиболее распространённых форм досуга среди молодёжи, друзей и семейных компаний. Он позволяет объединить общение и отдых, создавая позитивную атмосферу и усиливая социальные связи между участниками. Однако в процессе выбора фильма часто возникают трудности, связанные с различием вкусов и предпочтений. Обсуждения могут затягиваться, а иногда и приводить к спорам, из-за чего само удовольствие от просмотра снижается.

В последние годы всё больше людей используют онлайн-платформы и стриминговые сервисы, такие как Netflix, Кинопоиск, Okko, предоставляющие доступ к тысячам фильмов и сериалов. Несмотря на широкий выбор контента, большинство таких сервисов ориентировано на индивидуального пользователя и не предусматривает возможности коллективного выбора. Это создаёт потребность в специализированных решениях, позволяющих группе людей совместно принимать решение о том, что смотреть.

Предметной областью данного проекта является организация процесса коллективного выбора фильмов для совместного просмотра. В рамках этой области рассматриваются взаимодействия между пользователями, обмен информацией о предпочтениях и формирование общего решения на основе индивидуальных оценок.

Основными элементами предметной области являются:

* пользователи – участники группы, которые совместно выбирают фильм для просмотра;
* сессия выбора – временная группа, создаваемая для совместного голосования;
* список фильмов – набор предложенных вариантов, из которых пользователи делают выбор;
* оценки и предпочтения – данные, на основе которых система подбирает фильм, подходящий большинству;
* результат подбора – фильм, который максимально соответствует интересам всех участников.

# Архитектура приложения

Архитектура приложения представляет собой связку из трех уровней: клиент-сервер-API (рис. 1).

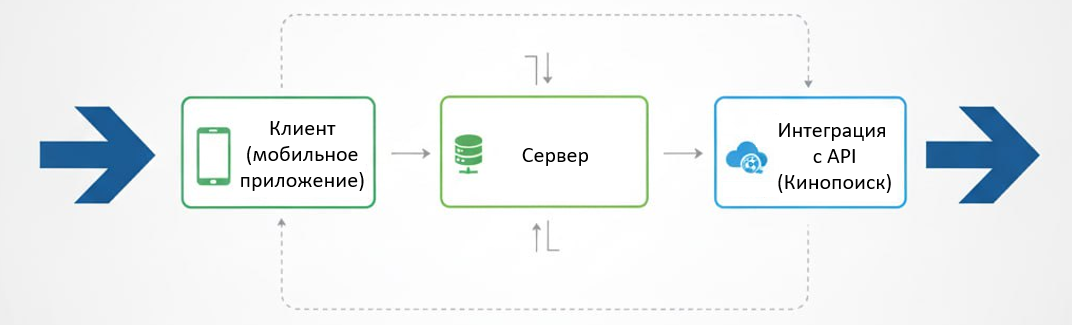


Рис. 1. Архитектура приложения

## Общая архитектура

Клиентом является часть приложения, с которой взаимодействует пользователь.

Основными функциями клиента являются:

* отображение пользовательского интерфейса;
* взаимодействие с сервером;
* отображение данных о фильмах, полученных через API;
* реализация динамических эффектов и плавных переходов.

Сервер отвечает за обработку данных и взаимодействие между клиентом и внешними источниками.

Основными функциями сервера являются:

* получение и обработка запросов от клиента;
* организация двустороннего взаимодействия в реальном времени через WebSocket [7];
* валидация данных и управление логикой приложения;
* интеграция с API для получения информации о фильмах.

В качестве внешнего источника данных используется API Кинопоиска, обеспечивающее доступ к информации о фильмах.

Основные данные, получаемые через API:

* список фильмов;
* постеры и описания;
* рейтинги;
* жанры;
* страны и годы выпуска.

## Логика работы

Логика работы приложения включает несколько последовательных этапов, каждый из которых отвечает за определённую часть процесса выбора фильма:

1. Создание группы:

* пользователь нажимает кнопку «Создать группу»;
* система генерирует уникальный 5-значный код, состоящий из букв и цифр;
* создается запись группы в базе данных;
* пользователь получает код для приглашения друзей.

Макеты интерфейса на этапе создания группы представлены на рис. 2.

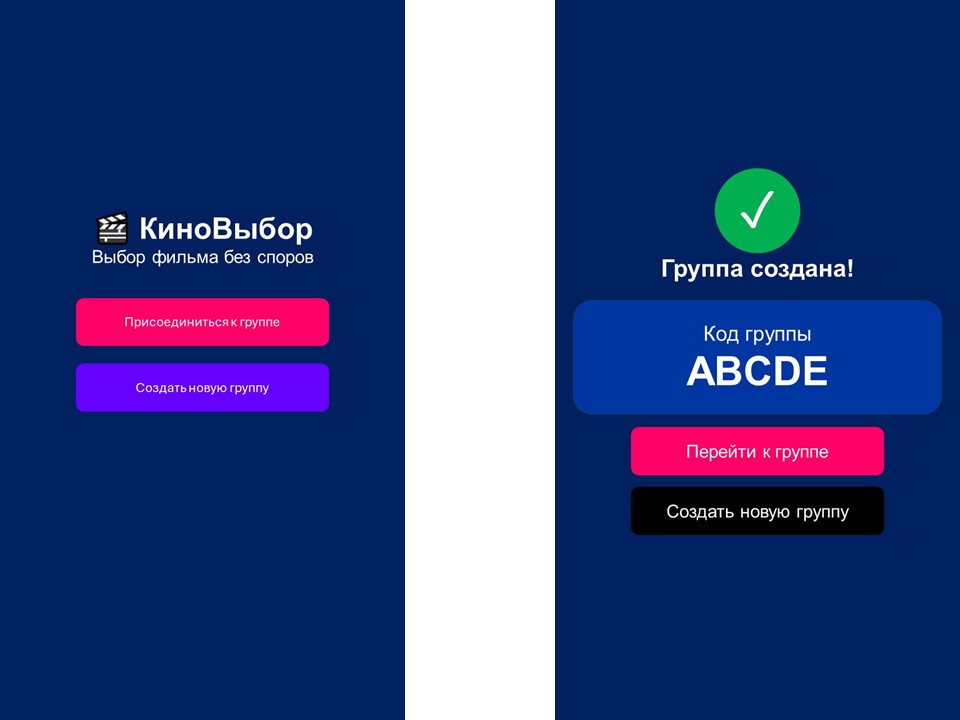


Рис. 2. Макеты интерфейса на этапе создания группы

1. Присоединение к группе:

* пользователь вводит код группы;
* система валидирует код (формат, существование, активность);
* проверяется количество участников (максимум 10);
* участник добавляется в группу.

Макеты интерфейса на этапе присоединения к группе представлены на рис. 3.

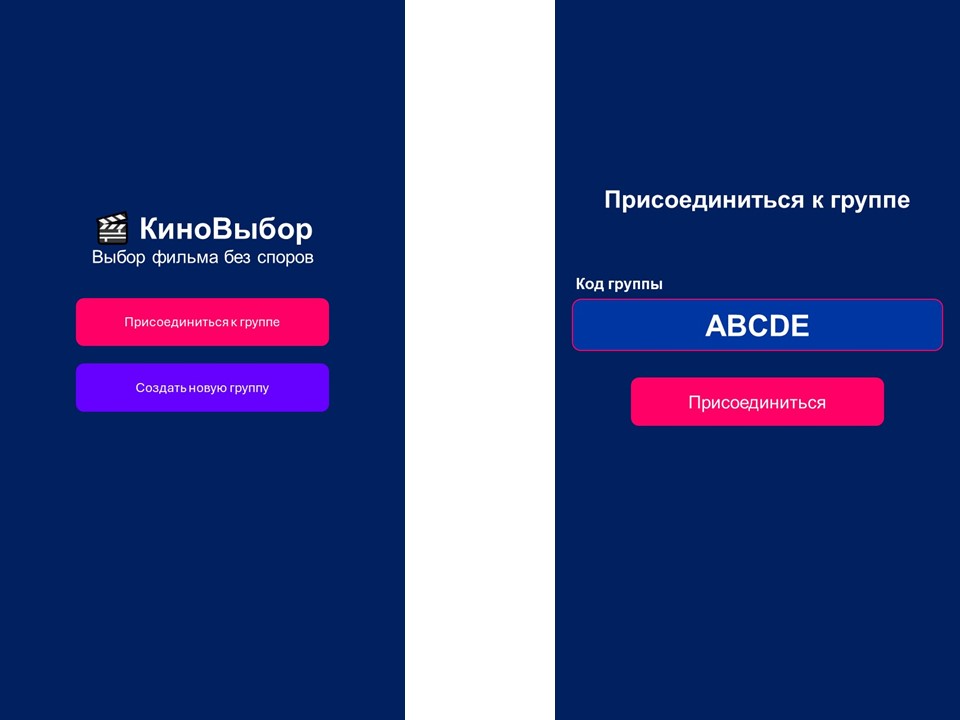


Рис. 3. Макеты интерфейса на этапе присоединения к группе

1. Добавление фильмов:

* участник ищет фильм через API Кинопоиска;
* участник выбирает фильм из результатов поиска;
* фильм добавляется в пул группы (максимум 5 фильмов на участника);
* все участники видят добавленный фильм в реальном времени.

Макет интерфейса на этапе выбора фильмов представлен на рис. 4.

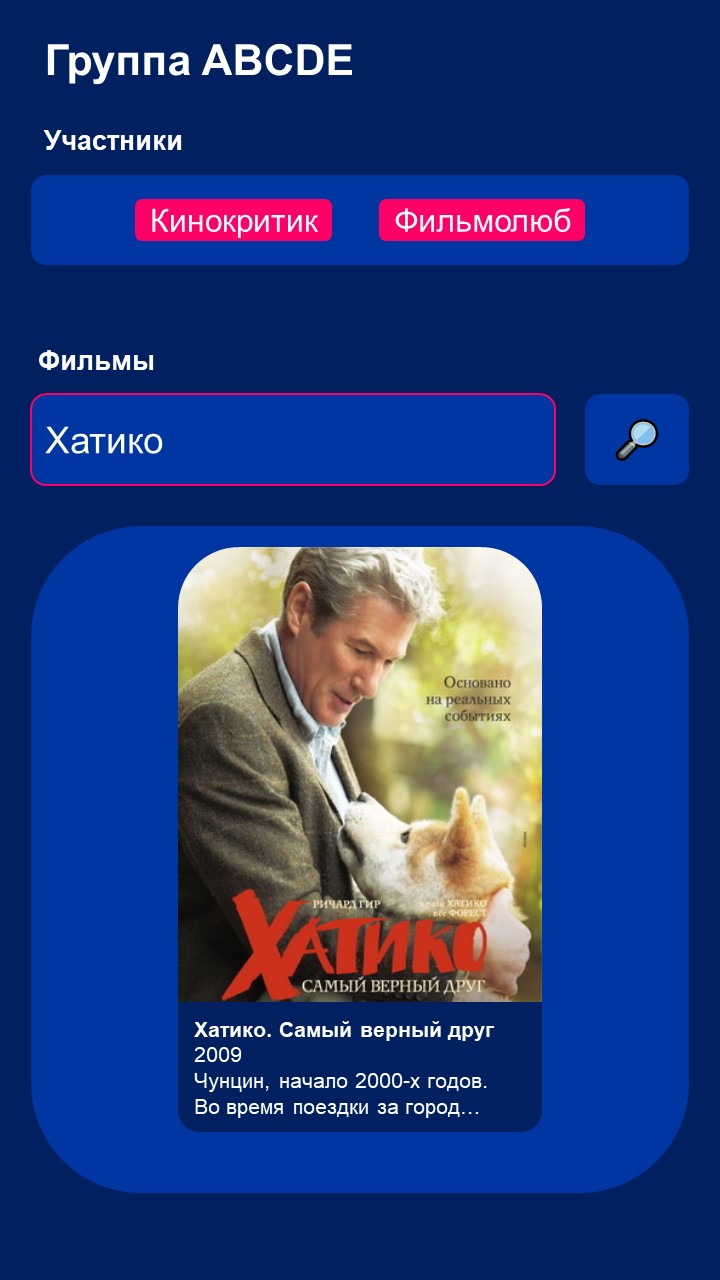


Рис. 4. Макет интерфейса на этапе выбора фильмов

1. Голосование:

* создатель группы нажимает кнопку «Начать выбор»;
* всем участникам показываются карточки фильмов;
* участники оценивают фильмы, нажимая на кнопки «Лайк» (нравится) или «Дизлайк» (не нравится), либо используя свайп – смахивание карточек влево или вправо;
* система отслеживает голоса всех участников.

Макет интерфейса на этапе голосования представлен на рис. 5.



Рис. 5. Макет интерфейса на этапе голосования

1. Поиск совпадений:

* начинается, когда все участники проголосовали за фильмы;
* система автоматически определяет совпадения;
* все участники получают уведомление о найденном фильме;
* отображается страница результатов с информацией о фильме.

Макет интерфейса на этапе поиска совпадений представлен на рис. 6.

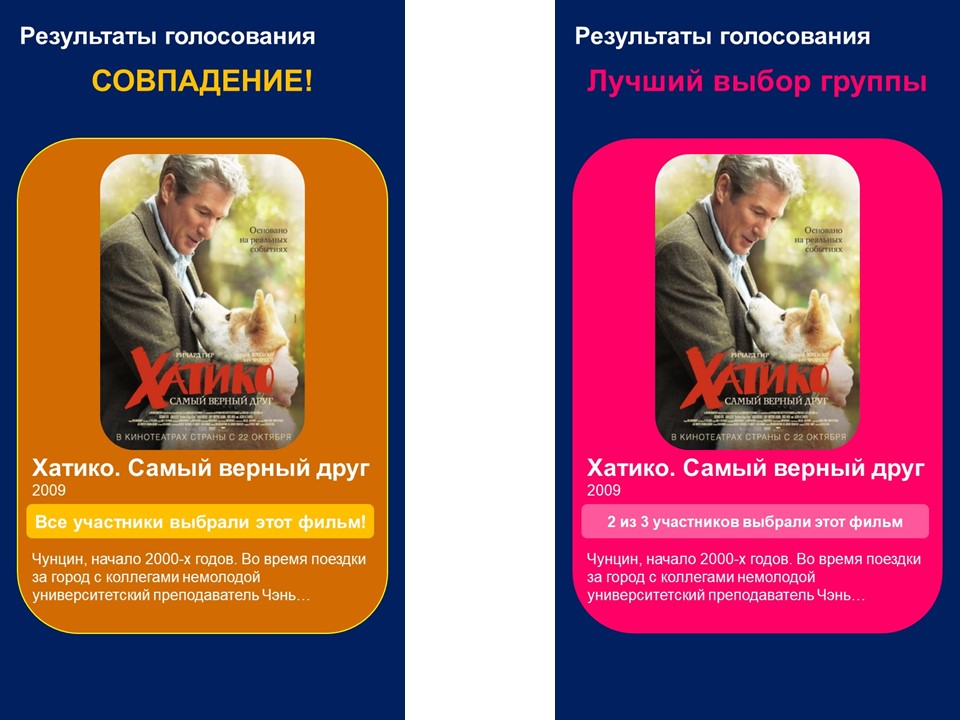


Рис. 6. Макет интерфейса на этапе поиска совпадений

# Средства разработки

Разработка мобильного приложения «КиноВыбор» осуществляется с использованием web-технологий, что позволяет обеспечить кроссплатформенность и доступность на различных устройствах. В качестве основы выбран фреймворк Next.js, предоставляющий современную архитектуру на базе библиотеки React и возможность серверного рендеринга [6]. Такое решение позволяет повысить производительность, улучшить скорость загрузки интерфейса и упростить адаптацию под мобильные устройства.

Пользовательский интерфейс реализован с использованием React, обеспечивающего компонентную структуру и гибкость при создании интерактивных элементов. Применение языка программирования TypeScript позволяет повысить надёжность и читаемость кода благодаря статической типизации [8]. Для быстрой и удобной разработки адаптивных интерфейсов используется фреймворк Tailwind CSS, что обеспечивает единый визуальный стиль и корректное отображение на экранах различного размера [2]. Дополнительно применяется библиотека Framer Motion, позволяющая реализовать плавные анимации и улучшить визуальное восприятие приложения пользователем [4].

Серверная логика реализована с использованием фреймворков Next.js API Routes и Node.js, что позволяет объединить клиентскую и серверную часть в рамках одного проекта [5]. Такой подход упрощает архитектуру и ускоряет процесс разработки. Node.js обеспечивает стабильную работу приложения и эффективную обработку запросов при взаимодействии с базой данных и внешними сервисами.

Для хранения данных и аутентификации пользователей используется облачная платформа Firebase, обеспечивающая надёжное хранение информации, простую интеграцию с frontend и поддержку работы в реальном времени. Это позволяет синхронизировать действия пользователей при совместном выборе фильмов и обеспечивать стабильную работу приложения без необходимости настройки отдельного сервера баз данных.

Для получения актуальной информации о фильмах используется Кинопоиск API, предоставляющий данные о названии, описании, жанрах, рейтингах и постерах [3]. Интеграция с этим сервисом позволяет сформировать удобный каталог фильмов и повысить качество рекомендаций.

В процессе разработки применяются инструменты, повышающие качество кода и эффективность работы команды:

* инструменты ESLint и Prettier используются для автоматической проверки и форматирования кода, обеспечивая его единый стиль и читаемость;
* система контроля версий Git используется для управления исходным кодом проекта, а его репозиторий хранится на GitHub, что обеспечивает удобство совместной работы, отслеживание изменений и ведение истории разработки;
* для проектирования интерфейсов и визуальных прототипов используется графический редактор Figma;
* управление задачами и этапами разработки осуществляется с помощью менеджера задач Trello, что способствует чёткому распределению обязанностей и контролю за выполнением проекта.

# Мобильное приложение

Разрабатываемое мобильное приложение позволит группе пользователей создавать сессии выбора фильма, добавлять варианты для голосования и получать результат автоматически. Все действия происходят в реальном времени, а интерфейс структурирован по ключевым функциям: создание группы, поиск фильмов, голосование и отображение результата.

## Разработка дизайна мобильного приложения

Дизайн мобильного приложения ориентирован на доступ с мобильных устройств и удобную навигацию между ключевыми пользовательскими сценариями.

На стартовом экране расположены 2 кнопки – «Присоединиться к группе» и «Создать новую группу» (рис. 7).

При выборе любого из вариантов программа откроет экран ввода никнейма, на котором расположено поле для ввода имени, кнопка подтверждения, а также несколько кнопок с вариантами готовых никнеймов (рис. 8).

При сценарии создания группы программа после ввода никнейма откроет экран подключения к группе, содержащий поле для ввода кода и кнопку подтверждения (рис. 9).

При сценарии создания группы программа после ввода никнейма откроет экран создания группы, содержащее сообщение об успешном создании группы, блок с кодом и 2 кнопки – «Перейти к группе» и «Создать новую группу» (рис. 10).

После входа в группу откроется основной экран, состоящий из 3 частей. В «шапке» располагается код группы, под ней – список участников, еще ниже – список фильмов. Добавление фильмов осуществляется по нажатию кнопки «Добавить фильм». У создателя группы под списком фильмов также располагается кнопка «Начать выбор фильма» (рис. 11).

При нажатии на кнопку «Добавить фильм» на том же экране появляется поле для поиска фильма и кнопка с иконкой лупы для подтверждения. После выполнения поиска приложение выдаст карточки с подходящими фильмами, под каждой из которых располагается кнопка «Добавить в группу» (рис. 12). При нажатии по самой карточки открывается описание фильма (рис. 13).

После начала голосования открывается экран выбора фильмов. В «шапке» располагается код группы и индикатор количества фильмов, за которые необходимо проголосовать, в виде полосы. Под «шапкой» располагается афиша фильма, под ней – 3 кнопки: красная кнопка с иконкой крестика («дизлайк»), серая кнопка с иконкой стрелки (открыть описание фильма) и зеленая кнопка с иконкой сердечка («лайк»). Для оценивания фильмов можно воспользоваться данными кнопками, либо смахиванием афиши влево или вправо. Внизу экрана располагается кнопка «Предыдущий» для возврата к прошлому фильму (рис. 14).

После голосования открывается экран результатов, в котором отображается афиша, название, описание фильма-победителя, а также информация о количестве лайков и дизлайков на этом фильме (рис. 15).

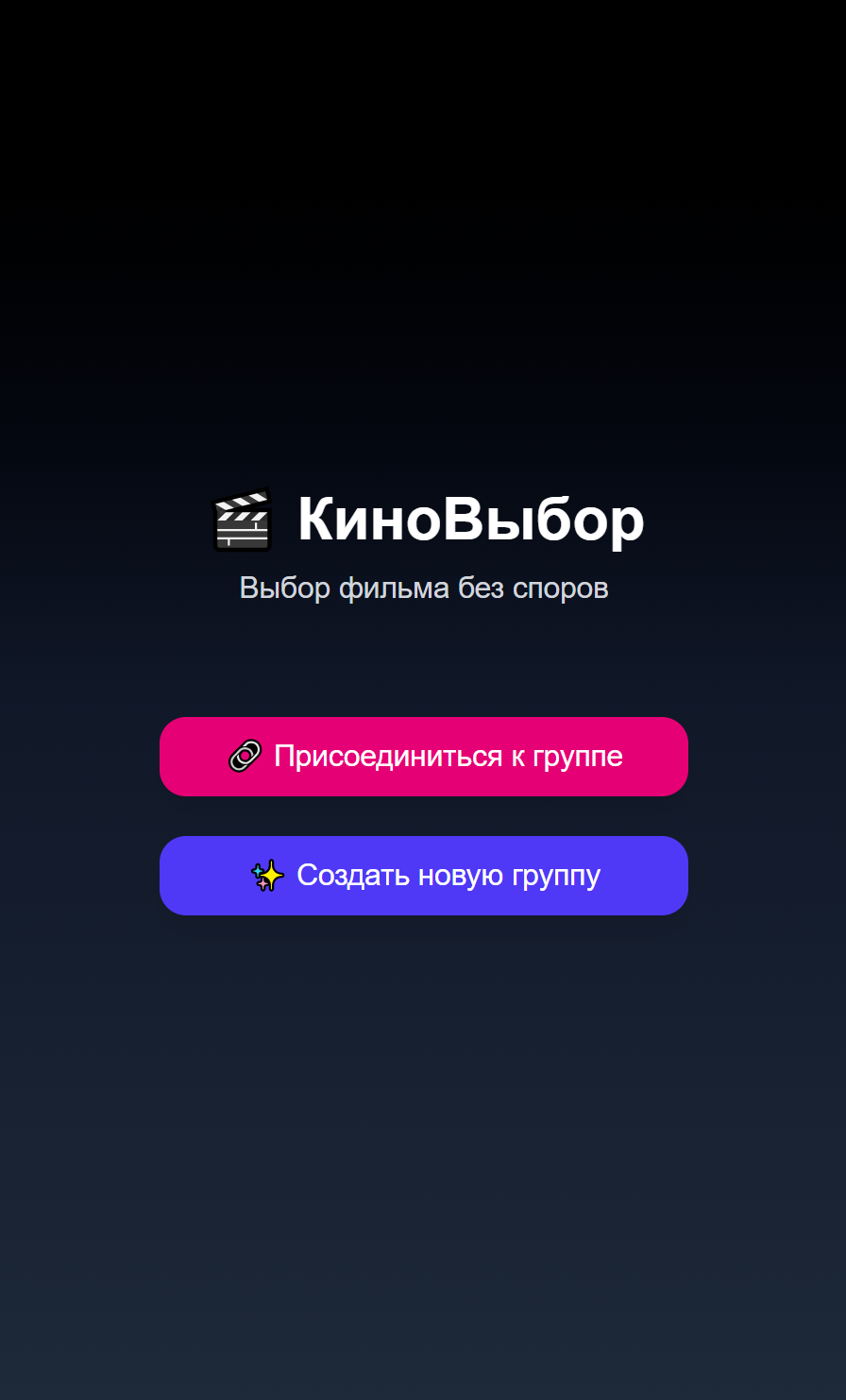


Рис. 7. Стартовый экран

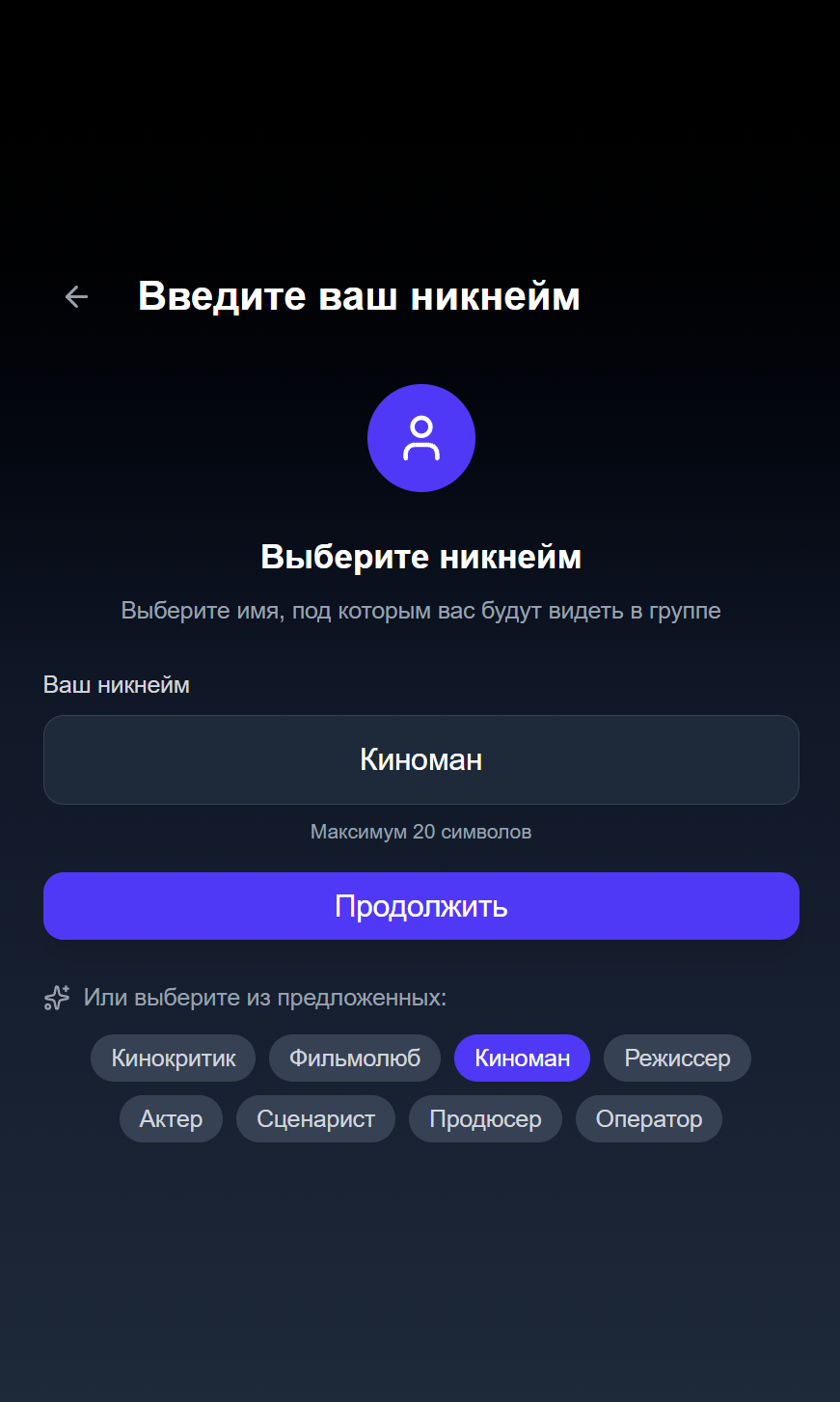


Рис. 8. Экран ввода никнейма

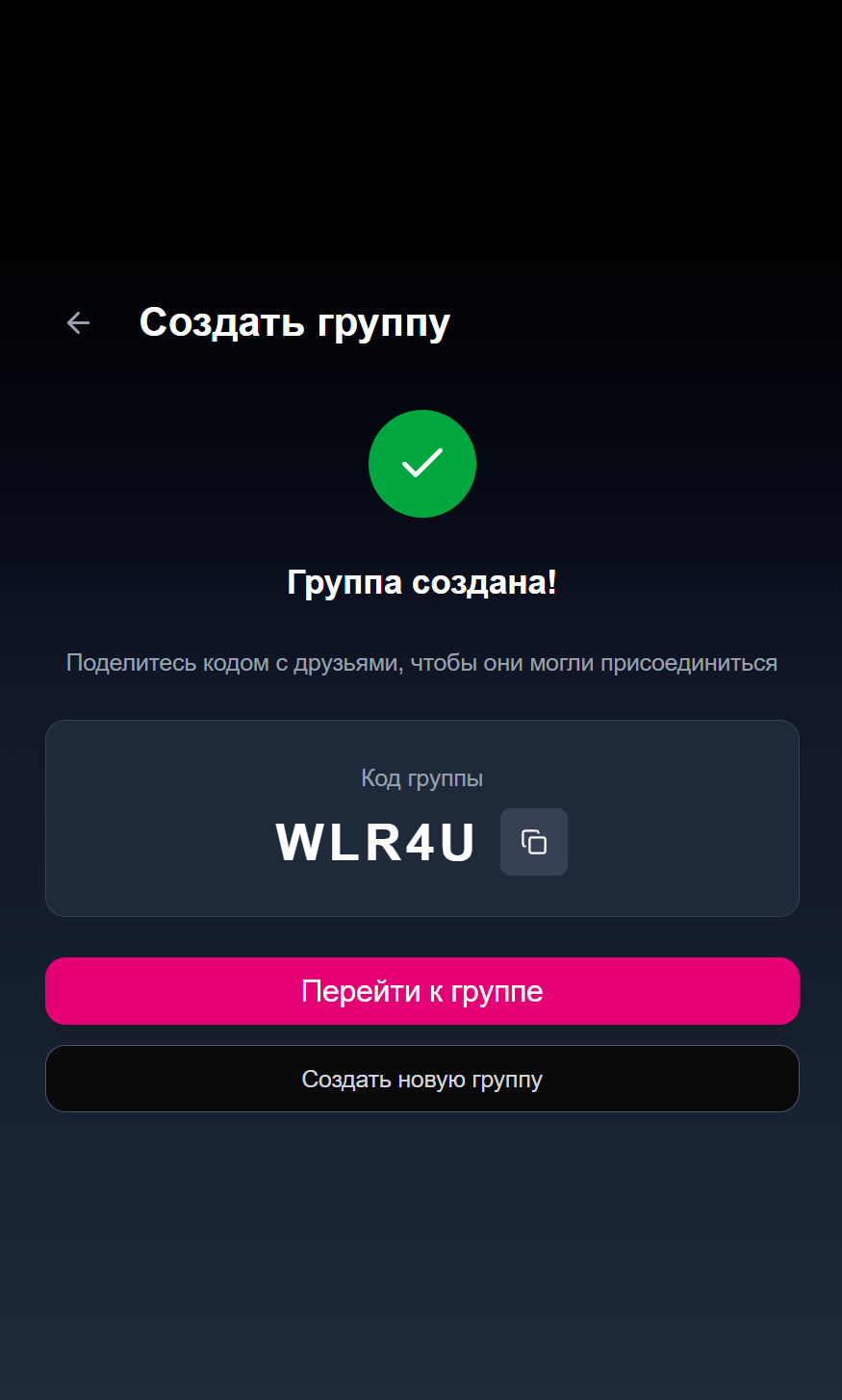


Рис. 9. Экран создания группы

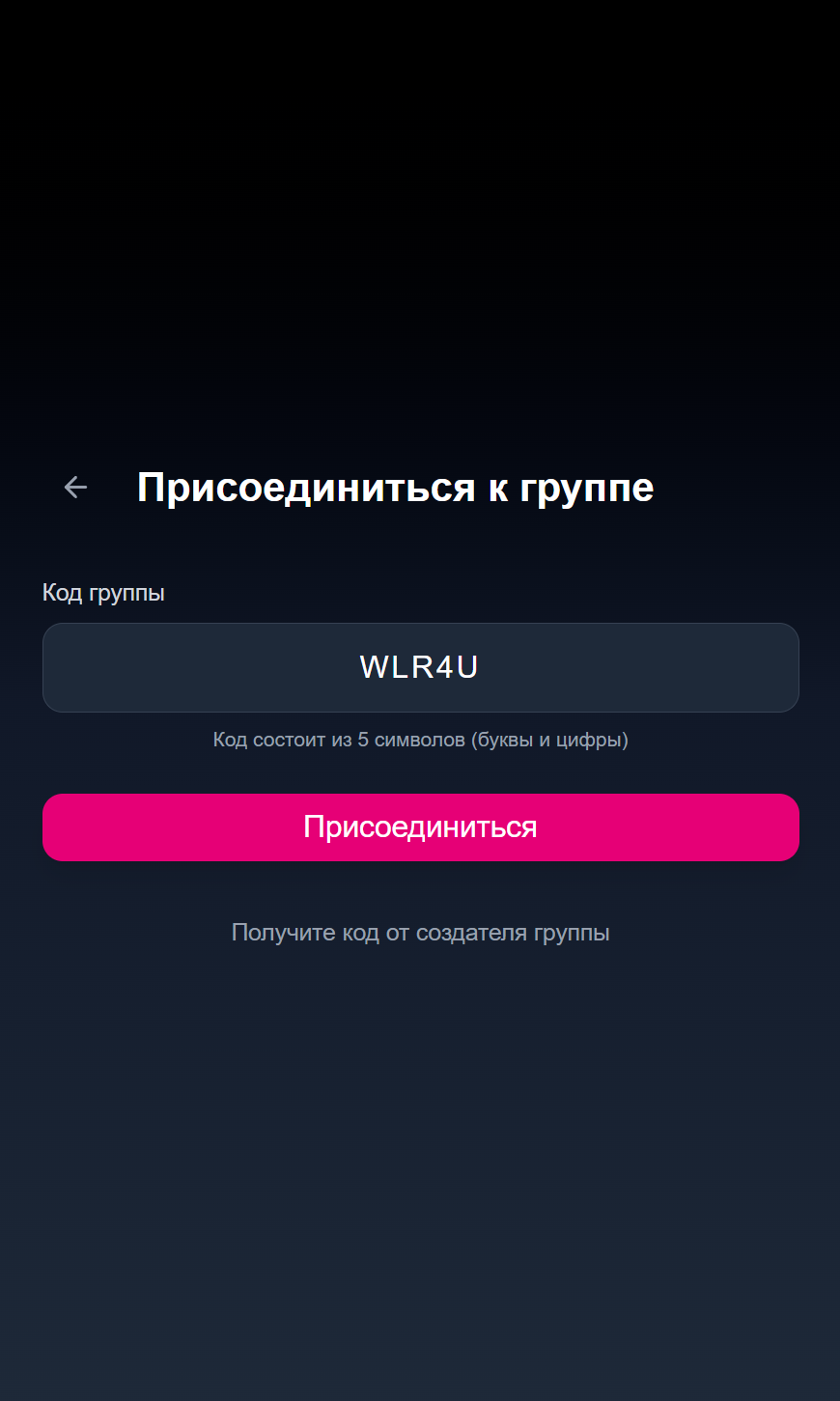


Рис. 10. Экран присоединения к группе

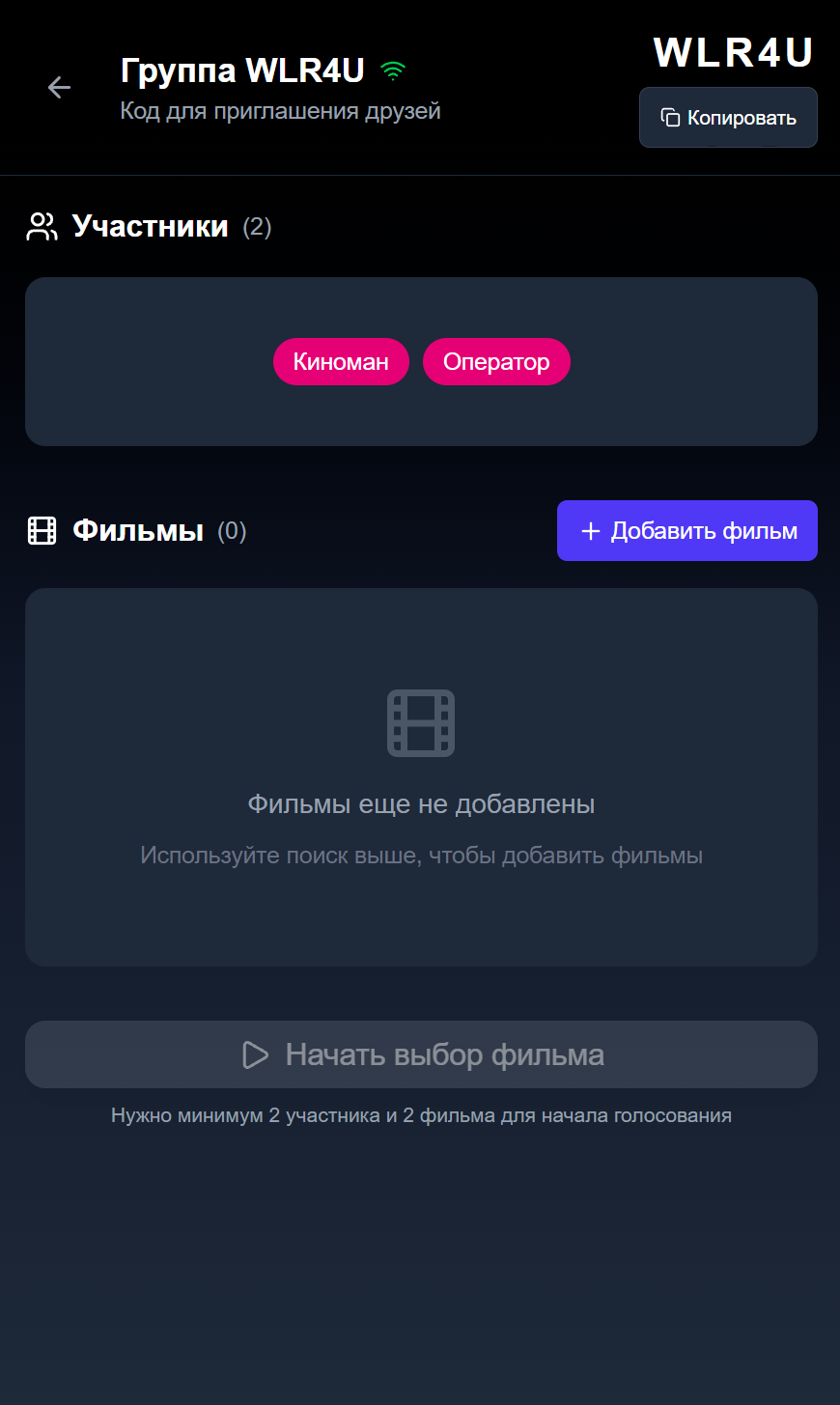


Рис. 11. Основной экран

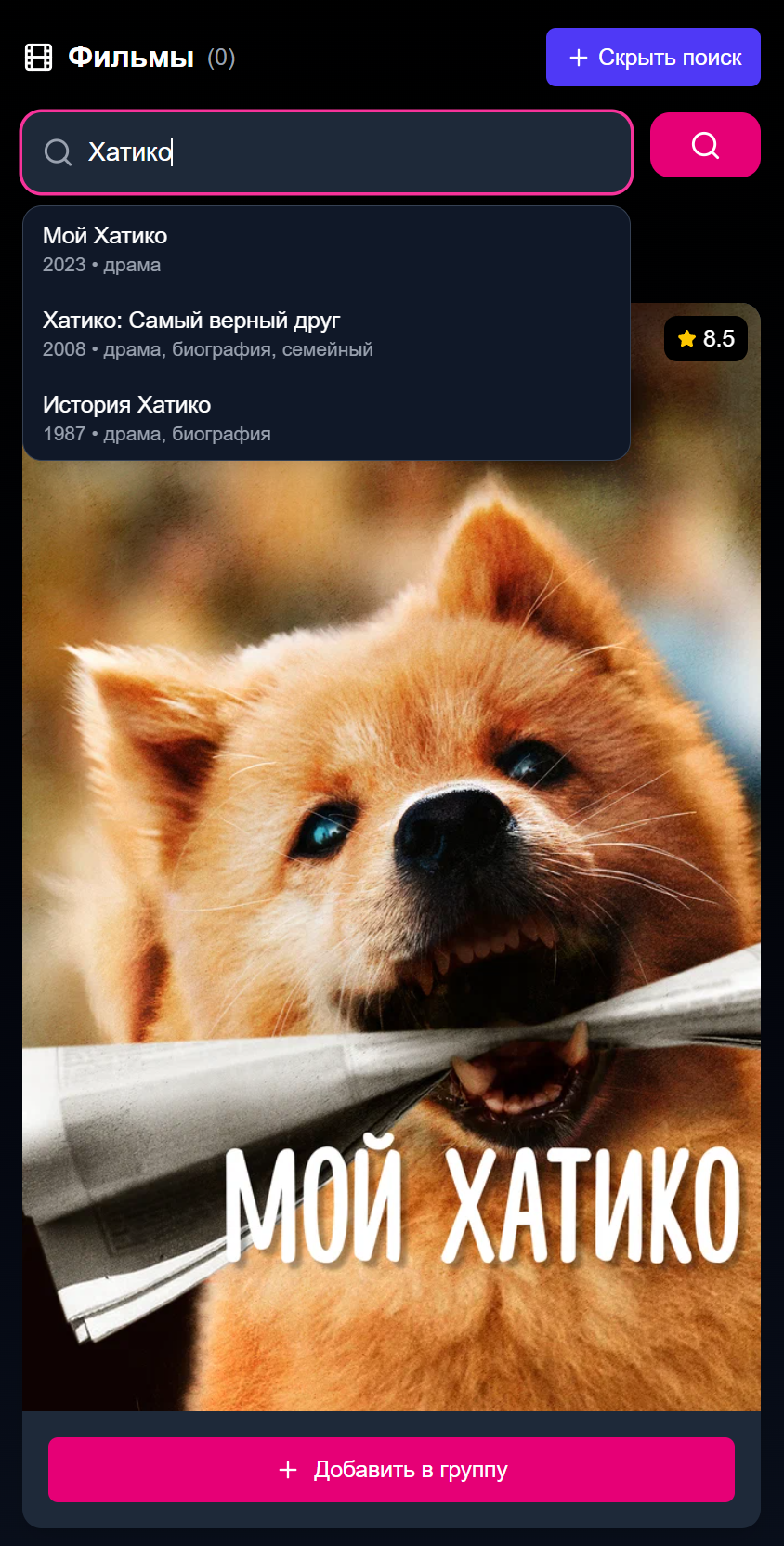


Рис. 12. Интерфейс поиска фильмов

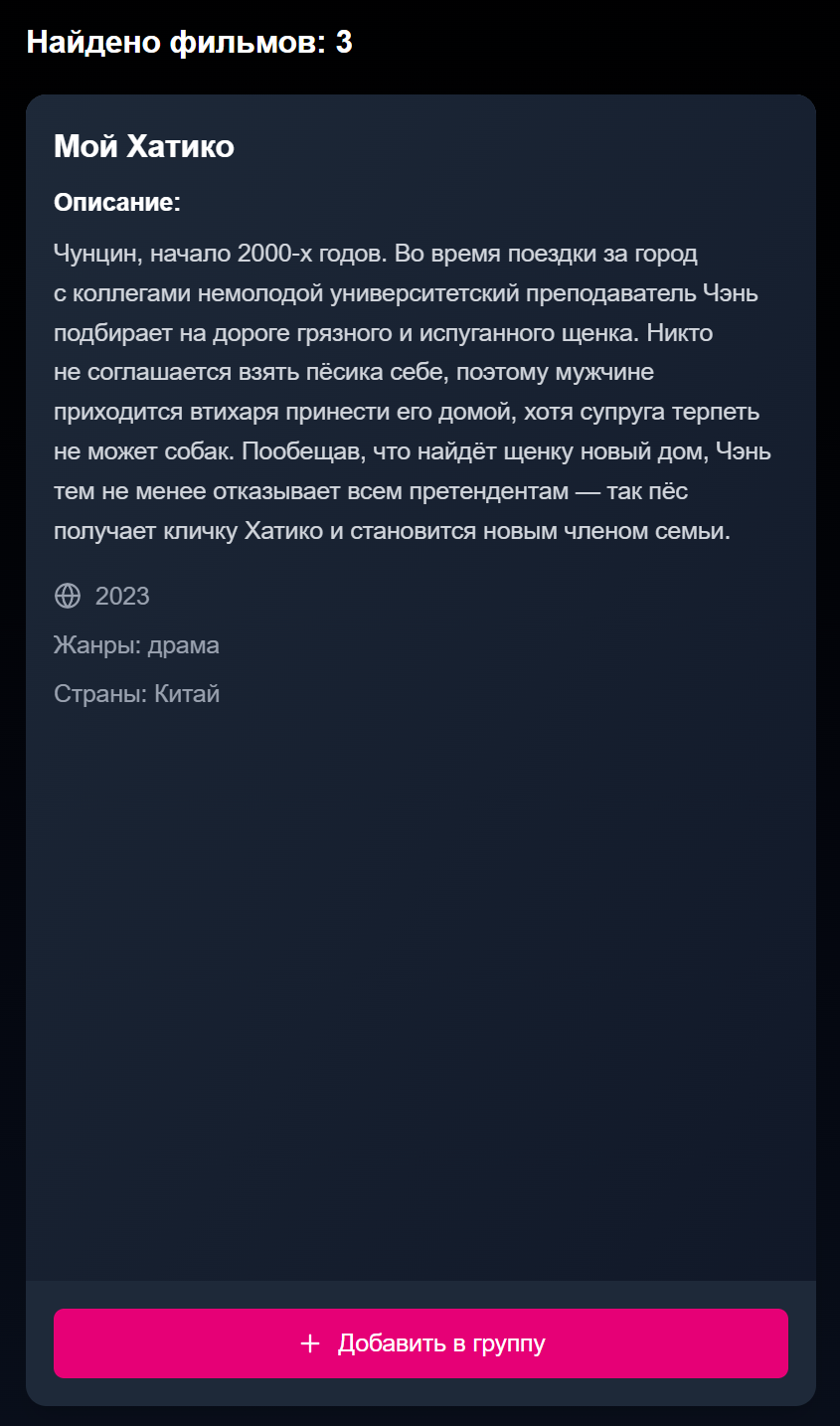


Рис. 13. Описание фильма

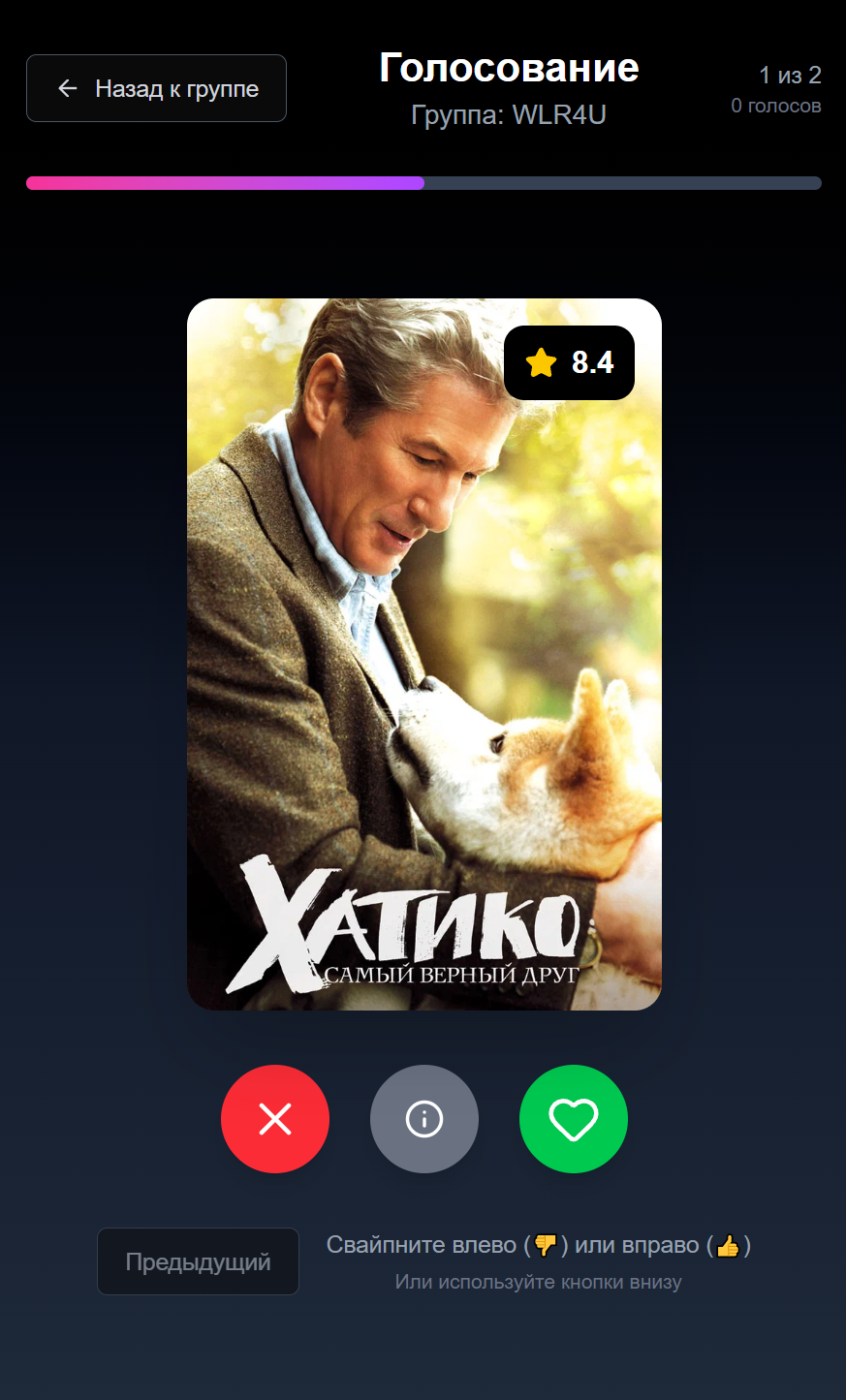


Рис. 14. Экран голосования

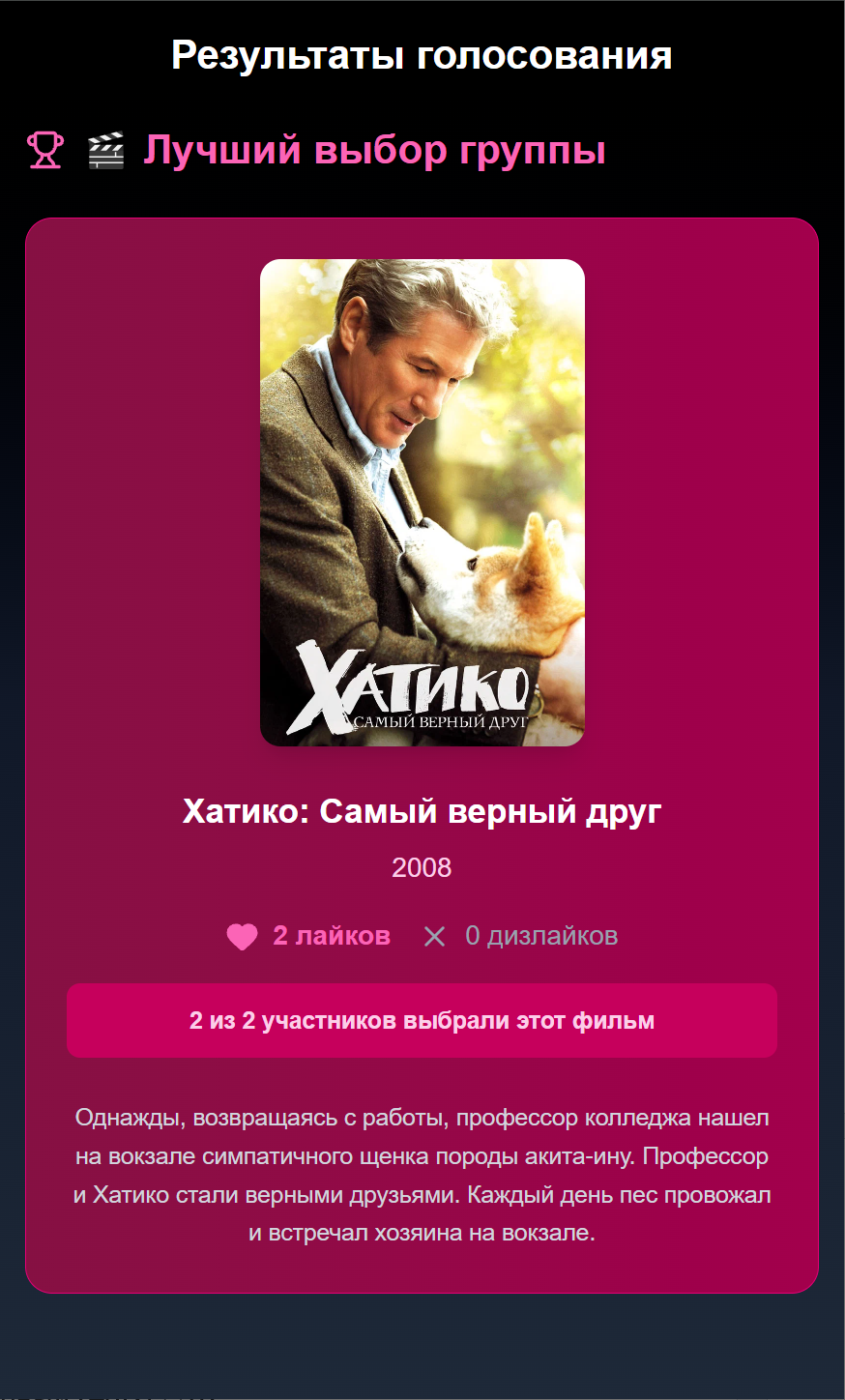


Рис. 15. Экран результатов голосования

При разработке дизайна использовалась следующая цветовая палитра:

* Фон экрана – угольно-черный (#0B1220);
* Фон карточек – графитовый (#111827);
* Основной текст (заголовки, названия), текст на кнопках – белый (#FFFFFF);
* Вторичный текст (год, рейтинг, подсказки) – светло‑серый (#CBD5E1);
* Разделители, границы – темно-серый (#1F2937);
* Главные кнопки действий – индиго (#4F46E5), розовый (#DB2777)
* Нажатое/активное состояние кнопок – насыщенный синий (#1D4ED8);
* Кнопка «Нравится» и подтверждающие метки – зелёный (#22C55E);
* Кнопка «Не нравится» и ошибки – красный (#EF4444);
* Предупреждающие заметки – оранжевый (#F59E0B);
* Неактивные элементы – пыльно-серый (#9CA3AF);
* Полоса прогресса и декоративные выделения – градиент от розового к фиолетовому (от #EC4899 до #A855F7)

## Описание логики

Логика приложения строится вокруг группового выбора фильмов.

Пользователь либо создаёт новую группу, указав своё имя, либо подключается к существующей по коду. Код генерируется автоматически и используется как «ключ» для входа всех участников. Код нормализуется (верхний регистр, удаление пробелов), что предотвращает ошибки при вводе.

После входа пользователь попадает в раздел группы, где видит текущий список фильмов. Список формируется из прикладной модели фильма: идентификатор из внешнего источника (API Кинопоиска), название, год, постер, краткое описание и рейтинг.

Добавление новых позиций выполняется через поиск. Пользователь вводит запрос, получает результаты с карточками фильмов и может добавить выбранную карточку в группу. Перед сохранением выполняется проверка на дубликаты, благодаря чему один и тот же фильм не появляется в списке повторно. При попытке добавления дубликата пользователь получает сообщение о том, что этот фильм уже добавлен.

Голосование организовано как двоичный выбор для каждой позиции – «Нравится» или «Не нравится». Пользователь выставляет оценки по списку без лишних переходов.

По завершении оценивания определяется один фильм‑победитель. Выбор осуществляется по максимальному числу положительных оценок. При равенстве применяется сравнение доли положительных голосов, затем – порядок добавления в группу как окончательный критерий. В разделе результата отображается постер и название выбранного фильма с пометкой о выборе, предоставляется действие для запуска нового раунда добавления и голосования.

Во всех ключевых точках предусматривается базовая валидация и сообщения об ошибках: обязательность имени при создании группы, корректность кода при подключении, минимальная длина поискового запроса, проверка формата голосов. В интерфейсе это выражается подсказками рядом с полями, чтобы пользователь быстро понимал, что нужно исправить и как продолжить сценарий.

## Реализация мобильного приложения

Мобильное приложение представляет собой набор компонентов, ориентированных на мобильные устройства и объединённых единым пользовательским сценарием: создание или подключение к группе, работа со списком фильмов, поиск и добавление, голосование и отображение результата.

Структура мобильного приложения представлена на рис. 16. Компоненты интерфейса выделены в отдельную область, прикладные операции с данными и инициализация клиента вынесены в библиотечные модули, а точки интеграции с сервером представлены маршрутами API (как внешние зависимости для клиента). Такая схема упрощает переиспользование компонентов, локализацию изменений и изоляцию клиентской логики от серверной.

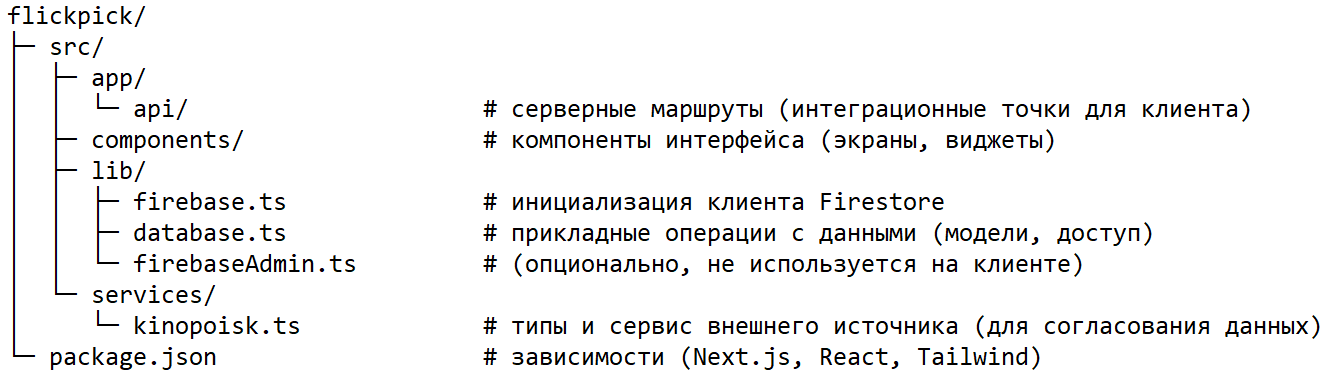


Рис. 16. Структура мобильного приложения

Переходы между ключевыми разделами реализуются через самостоятельные компоненты. Каждый раздел решает одну прикладную задачу и имеет собственные поля ввода, сообщения об ошибках и кнопки действий. Начальный раздел собирает необходимый минимум данных (имя или код), выполняет валидацию и инициирует переход в рабочее состояние — на раздел группы. Реализация начального раздела выполнена следующим образом:

'use client';

import { useState } from 'react';

export function GroupEntry() {

const [creatorName, setCreatorName] = useState('');

const [joinCode, setJoinCode] = useState('');

const [msg, setMsg] = useState<string | null>(null);

async function onCreate() {

setMsg(null);

const name = creatorName.trim();

if (!name) { setMsg('Укажите имя'); return; }

const res = await fetch('/api/groups-firebase', {

method: 'POST', headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ creatorName: name })

});

const json = await res.json();

if (!res.ok) { setMsg(json?.error || 'Не удалось создать группу'); return; }

// переход на экран группы с json.data.code

}

async function onJoin() {

setMsg(null);

const code = joinCode.toUpperCase().trim();

if (!code) { setMsg('Укажите код группы'); return; }

const res = await fetch(`/api/groups-firebase?code=${encodeURIComponent(code)}`);

const json = await res.json();

if (!res.ok) { setMsg(json?.error || 'Группа не найдена'); return; }

// переход на экран группы с данными json.data

}

return (

<main className="min-h-dvh bg-[#0B1220] text-white px-4 py-6">

{/\* поля ввода, кнопки, сообщения об ошибках по палитре из п.4.1 \*/}

</main>

);

}

Раздел группы служит центральной точкой сценария. При входе по коду происходит загрузка списка фильмов, отображение карточек в компактном мобильном виде и предложение действий: добавить новый фильм либо перейти к голосованию. Реализация раздела группы выполнена следующим образом:

'use client';

import { useEffect, useState } from 'react';

type Film = { id: string; kinopoiskId: number; title: string; year: number; poster?: string; rating?: number };

export function GroupScreen({ code }: { code: string }) {

const [films, setFilms] = useState<Film[]>([]);

const [error, setError] = useState<string | null>(null);

useEffect(() => {

let active = true;

(async () => {

const normalized = code.toUpperCase().trim();

const res = await fetch(`/api/groups-firebase?code=${encodeURIComponent(normalized)}`);

const json = await res.json();

if (!res.ok) { setError(json?.error || 'Ошибка загрузки'); return; }

if (active) setFilms((json.data?.films ?? []) as Film[]);

})();

return () => { active = false; };

}, [code]);

return (

<main className="min-h-dvh bg-[#0B1220] text-white">

{/\* шапка с кодом группы, список карточек по палитре из п.4.1 \*/}

</main>

);

}

В рамках выполнения поиска фильмов происходит валидация длины запроса, отображение результатов и добавление выбранного фильма одной кнопкой. При попытке добавить дубликат возвращается сообщение, при этом контекст поиска сохраняется. Реализация поиска и добавления выполнена следующим образом:

'use client';

import { useCallback, useMemo, useState } from 'react';

type SearchItem = { kinopoiskId: number; nameRu: string; year: number; posterUrl?: string; ratingKinopoisk?: number };

export function FilmSearch({ groupCode }: { groupCode: string }) {

const [q, setQ] = useState('');

const [items, setItems] = useState<SearchItem[]>([]);

const [msg, setMsg] = useState<string | null>(null);

const code = useMemo(() => groupCode.toUpperCase().trim(), [groupCode]);

const search = useCallback(async () => {

setMsg(null);

const query = q.trim();

if (query.length < 2) { setMsg('Введите не менее 2 символов'); return; }

const res = await fetch(`/api/films/search?q=${encodeURIComponent(query)}&page=1`);

const json = await res.json();

if (!res.ok) { setMsg(json?.error || 'Ошибка поиска'); return; }

setItems(json.data?.items ?? []);

}, [q]);

const add = useCallback(async (film: SearchItem) => {

setMsg(null);

const res = await fetch(`/api/groups-firebase/${encodeURIComponent(code)}/films`, {

method: 'POST', headers: { 'Content-Type': 'application/json' }, body: JSON.stringify({ film })

});

const json = await res.json();

if (!res.ok) { setMsg(json?.error || 'Не удалось добавить фильм'); return; }

setMsg('Фильм добавлен в группу');

}, [code]);

return (

<main className="min-h-dvh bg-[#0B1220] text-white px-4 py-4">

{/\* поле ввода, кнопка «Найти», список результатов с кнопкой «Добавить» \*/}

</main>

);

}

Раздел голосования содержит упрощённые карточки и три крупные кнопки. Оценки накапливаются локально и отправляются одним действием, пустая отправка предотвращается. Реализация раздела голосования выполнена следующим образом:

'use client';

import { useState } from 'react';

type VotePayload = { filmId: number; vote: 'like' | 'dislike'; participantId: string };

export function Voting({ groupCode, films, participantId }:{

groupCode: string; films: { kinopoiskId: number; title: string }[]; participantId: string;

}) {

const [votes, setVotes] = useState<VotePayload[]>([]);

const [msg, setMsg] = useState<string | null>(null);

function setVote(filmId: number, v: 'like' | 'dislike') {

setVotes(prev => {

const others = prev.filter(x => x.filmId !== filmId);

return [...others, { filmId, vote: v, participantId }];

});

}

async function submit() {

setMsg(null);

if (votes.length === 0) { setMsg('Нет оценок для отправки'); return; }

const code = groupCode.toUpperCase().trim();

const res = await fetch(`/api/groups-firebase/${encodeURIComponent(code)}/votes`, {

method: 'POST', headers: { 'Content-Type': 'application/json' }, body: JSON.stringify({ votes })

});

const json = await res.json();

if (!res.ok) { setMsg(json?.error || 'Ошибка отправки'); return; }

setMsg('Оценки учтены');

}

return (

<main className="min-h-dvh bg-[#0B1220] text-white px-4 py-4">

{/\* упрощённые карточки фильмов и кнопки «Нравится»/«Не нравится» по палитре из п.4.1 \*/}

</main>

);

}

Определение победителя выполняется по загруженным голосам и списку фильмов. Реализация вычисления результата выполнена следующим образом:

'use client';

import { useEffect, useMemo, useState } from 'react';

type Vote = { filmId: number; vote: 'like' | 'dislike' };

type Film = { id: string; kinopoiskId: number; title: string; addedAt?: string };

export function Winner({ groupCode, films }:{ groupCode: string; films: Film[] }) {

const [votes, setVotes] = useState<Vote[]>([]);

useEffect(() => {

(async () => {

const code = groupCode.toUpperCase().trim();

const res = await fetch(`/api/groups-firebase/${encodeURIComponent(code)}/votes`);

const json = await res.json();

if (res.ok) setVotes((json.data?.votes ?? []) as Vote[]);

})();

}, [groupCode]);

const winner = useMemo(() => {

if (!votes.length) return null;

const byFilm = new Map<number, { likes: number; total: number; addedAt?: number }>();

for (const f of films) byFilm.set(f.kinopoiskId, { likes: 0, total: 0, addedAt: f.addedAt ? Date.parse(f.addedAt) : undefined });

for (const v of votes) { const a = byFilm.get(v.filmId); if (!a) continue; a.total++; if (v.vote === 'like') a.likes++; }

const arr = [...byFilm.entries()];

arr.sort((a, b) => {

const A = a[1], B = b[1];

if (B.likes !== A.likes) return B.likes - A.likes;

const rA = A.total ? A.likes / A.total : 0, rB = B.total ? B.likes / B.total : 0;

if (rB !== rA) return rB - rA;

return (A.addedAt ?? 0) - (B.addedAt ?? 0);

});

return films.find(f => f.kinopoiskId === arr[0]?.[0]) ?? null;

}, [votes, films]);

return (

<main className="min-h-dvh bg-[#0B1220] text-white px-4 py-4">

{/\* показ карточки победителя: постер, название, пометка о выборе \*/}

</main>

);

}

Для всех операций предусмотрены состояния ожидания, пустых данных и ошибок. Реализация вспомогательного компонента сообщений выполнена следующим образом:

function Hint({ text, tone }:{ text: string; tone: 'neutral'|'error'|'warn' }) {

const color = tone === 'error' ? '#EF4444' : tone === 'warn' ? '#F59E0B' : '#CBD5E1';

return <p className="mt-2 text-sm" style={{ color }}>{text}</p>;

}

# Сервер

Серверная часть обеспечивает прикладную логику и доступ к данным для мобильного интерфейса: создание и получение групп по коду, поиск и добавление фильмов, фиксацию голосов и выдачу результата.

## Разработка сервера

Создание группы и выдача кода доступа реализованы с генерацией фиксированного 5‑символьного кода и проверкой обязательности имени создателя. Реализация данной операции выполнена следующим образом:

// src/app/api/groups-firebase/route.ts — POST /api/groups-firebase

export async function POST(request: NextRequest) {

const body = await request.json();

const { creatorName } = body;

if (!creatorName || typeof creatorName !== 'string') {

return NextResponse.json({ error: 'Имя создателя группы обязательно' }, { status: 400 });

}

const code = generateGroupCode().toUpperCase().trim();

const groupId = await createGroup({

code, participants: [creatorName], isActive: true, createdBy: creatorName

});

return NextResponse.json({

success: true,

data: { id: groupId, code, participants: [creatorName], films: [], votes: [], createdAt: new Date(), isActive: true }

});

}

// Генерация 5‑символьного кода

function generateGroupCode(): string {

const chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';

let result = '';

for (let i = 0; i < 5; i++) result += chars.charAt(Math.floor(Math.random() \* chars.length));

return result;

}

Получение сведений о группе по коду выполняется с нормализацией строки (верхний регистр, обрезка пробелов) и подгрузкой фильмов группы. Реализация данной операции выполнена следующим образом:

// src/app/api/groups-firebase/route.ts — GET /api/groups-firebase?code=XXXXX

export async function GET(request: NextRequest) {

const { searchParams } = new URL(request.url);

const code = searchParams.get('code');

if (!code) {

return NextResponse.json({ error: 'Код группы обязателен' }, { status: 400 });

}

const normalizedCode = code.toUpperCase().trim();

const group = await getGroupByCode(normalizedCode);

if (!group) {

return NextResponse.json({ error: 'Группа не найдена' }, { status: 404 });

}

const films = await getFilmsByGroup(group.id);

return NextResponse.json({ success: true, data: { ...group, films } });

}

Нормализация кода и выборка данных группы на уровне доступа к базе предотвращают ошибки ввода и обеспечивают единообразие поиска. Реализация данной операции выполнена следующим образом:

// src/lib/database.ts — нормализация и выборка группы по коду

export async function getGroupByCode(code: string): Promise<Group | null> {

const normalizedCode = code?.toUpperCase().trim();

if (!normalizedCode) return null;

const q = query(collection(db, 'groups'), where('code', '==', normalizedCode));

const querySnapshot = await getDocs(q);

if (querySnapshot.empty) return null;

const docSnap = querySnapshot.docs[0];

const data = docSnap.data();

return {

id: docSnap.id,

code: data.code,

participants: data.participants || [],

films: [],

votes: [],

createdAt: data.createdAt?.toDate() || new Date(),

isActive: data.isActive ?? true,

createdBy: data.createdBy || ''

};

}

Добавление фильма в состав группы выполняется с предварительной проверкой отсутствия дубликатов и сопоставлением полей из внешнего источника к прикладной модели. Реализация данной операции выполнена следующим образом:

// src/app/api/groups-firebase/[id]/films/route.ts — POST /api/groups-firebase/[code]/films

export async function POST(request: NextRequest, { params }: { params: Promise<{ id: string }> }) {

const { id: groupCode } = await params;

const { film } = await request.json();

if (!film) {

return NextResponse.json({ error: 'Фильм не указан' }, { status: 400 });

}

const group = await getGroupByCode(groupCode);

if (!group) {

return NextResponse.json({ error: 'Группа не найдена' }, { status: 404 });

}

// Сопоставление полей на стороне сервера

await addFilmToGroup({

groupId: group.id,

kinopoiskId: film.kinopoiskId, // идентификатор Кинопоиска

title: film.nameRu, // название (рус.)

year: film.year, // год выпуска

poster: film.posterUrl, // постер (изображение)

description: film.description, // краткое описание

rating: film.ratingKinopoisk, // рейтинг Кинопоиска

addedBy: 'Пользователь'

});

const films = await getFilmsByGroup(group.id);

return NextResponse.json({ success: true, data: { ...group, films } });

}

Поиск фильмов во внешнем источнике и валидация параметров запроса выполняются в отдельном маршруте. Для части результатов дополнительно запрашиваются подробные сведения, что повышает информативность без усложнения клиента. Реализация данной операции выполнена следующим образом:

// src/app/api/films/search/route.ts — GET /api/films/search?q=...&page=...

export async function GET(request: NextRequest) {

const { searchParams } = new URL(request.url);

const query = searchParams.get('q');

const page = parseInt(searchParams.get('page') || '1');

if (!query || query.trim().length === 0) {

return NextResponse.json({ error: 'Поисковый запрос обязателен' }, { status: 400 });

}

if (query.trim().length < 2) {

return NextResponse.json({ error: 'Поисковый запрос должен содержать минимум 2 символа' }, { status: 400 });

}

const results = await kinopoiskService.searchFilms(query.trim(), page);

const filmsWithDetails = await Promise.allSettled(

results.items.slice(0, 5).map(async (film) => {

try {

const details = await kinopoiskService.getFilmDetails(film.kinopoiskId);

return { ...film, ...details };

} catch { return film; }

})

);

const enriched = filmsWithDetails.map((r, i) => r.status === 'fulfilled' ? r.value : results.items[i]);

const remaining = results.items.slice(5);

return NextResponse.json({ success: true, data: { ...results, items: [...enriched, ...remaining] } });

}

Приём, проверка и сохранение голосов реализованы с явной валидацией формата, а также с возможностью выборки и очистки голосов для конкретной группы. Реализация соответствующих операций выполнена следующим образом:

// src/app/api/groups-firebase/[id]/votes/route.ts — POST / GET / DELETE

export async function POST(request: NextRequest, { params }: { params: Promise<{ id: string }> }) {

const { id: groupCode } = await params;

const { votes } = await request.json();

if (!votes || !Array.isArray(votes)) {

return NextResponse.json({ error: 'Неверный формат голосов' }, { status: 400 });

}

const group = await getGroupByCode(groupCode.toUpperCase().trim());

if (!group) {

return NextResponse.json({ error: 'Группа с указанным кодом не найдена' }, { status: 404 });

}

const validVotes = votes.filter((v: { filmId: number; vote: string }) =>

v && typeof v.filmId === 'number' && (v.vote === 'like' || v.vote === 'dislike')

);

if (validVotes.length === 0) {

return NextResponse.json({ error: 'Нет валидных голосов' }, { status: 400 });

}

await addVotes(validVotes.map(v => ({

groupId: group.id,

filmId: v.filmId,

vote: v.vote,

participantId: v.participantId

})));

return NextResponse.json({ success: true, message: 'Голоса успешно сохранены' });

}

export async function GET(\_req: NextRequest, { params }: { params: Promise<{ id: string }> }) {

const { id: groupCode } = await params;

const group = await getGroupByCode(groupCode);

if (!group) return NextResponse.json({ error: 'Группа не найдена' }, { status: 404 });

const votes = await getVotesByGroup(group.id);

return NextResponse.json({ success: true, data: { votes } });

}

export async function DELETE(\_req: NextRequest, { params }: { params: Promise<{ id: string }> }) {

const { id: groupCode } = await params;

const normalized = groupCode?.toUpperCase().trim();

if (!normalized) return NextResponse.json({ error: 'Код группы не указан' }, { status: 400 });

const group = await getGroupByCode(normalized);

if (!group) return NextResponse.json({ error: 'Группа не найдена' }, { status: 404 });

await deleteVotesByGroup(group.id);

return NextResponse.json({ success: true, message: 'Все голоса успешно удалены' });

}

Конфигурация внешнего источника (Кинопоиск) и проверка ключа API защищают от некорректных вызовов и позволяют диагностировать ошибки конфигурации. Реализация соответствующей операции выполнена следующим образом:

// src/services/kinopoisk.ts — проверки наличия ключа и HTTP‑ошибок

const KINOPOISK\_API\_KEY = process.env.KINOPOISK\_API\_KEY || '';

async searchFilms(query: string, page = 1) {

if (!this.apiKey) {

throw new Error('API ключ Кинопоиска не настроен');

}

const response = await fetch(`${KINOPOISK\_API\_BASE\_URL}/films?keyword=${encodeURIComponent(query)}&page=${page}`, {

headers: { 'X-API-KEY': this.apiKey, 'Content-Type': 'application/json' },

});

if (!response.ok) {

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

}

return await response.json();

}

## Соединение сервера и мобильного приложения

Соединение клиентской части с сервером осуществляется по HTTP с обменом данными в формате JSON. Клиент инициирует операции (создание/получение группы, поиск фильмов, добавление фильмов, отправка/получение голосов), серверные обработчики валидируют входные параметры, нормализуют значения (например, код группы) и возвращают структурированные ответы.

При создании группы клиент передаёт имя создателя, сервер генерирует пятисимвольный код, создаёт запись и возвращает реквизиты новой группы. Реализация запроса выполнена следующим образом:

// Создание группы: клиент отправляет имя создателя и получает код группы

async function createGroup(creatorName: string) {

const res = await fetch('/api/groups-firebase', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ creatorName: creatorName.trim() })

});

const json = await res.json();

if (!res.ok) throw new Error(json?.error || 'Не удалось создать группу');

// json.data: { id, code, participants, ... }

return json.data;

}

Для получения группы по коду клиент передаёт этот код, сервер нормализует значение (верхний регистр, удаление пробелов), ищет запись и возвращает сведения о группе вместе с текущим списком фильмов. Реализация запроса выполнена следующим образом:

// Получение группы по коду: нормализация на клиенте + запрос на сервер

async function fetchGroup(rawCode: string) {

const code = rawCode.toUpperCase().trim();

const res = await fetch(`/api/groups-firebase?code=${encodeURIComponent(code)}`);

const json = await res.json();

if (!res.ok) throw new Error(json?.error || 'Группа не найдена');

// json.data: { id, code, participants, films: Film[], ... }

return json.data;

}

Для осуществления поиска фильма клиент отправляет текст запроса, сервер валидирует минимальную длину, обращается к внешнему источнику и возвращает результаты, дополняя часть позиций подробными сведениями. Реализация запроса выполнена следующим образом:

// Поиск фильмов: валидация запроса на клиенте и обращение к серверу

async function searchFilms(query: string, page = 1) {

const q = query.trim();

if (q.length < 2) throw new Error('Введите не менее 2 символов');

const res = await fetch(`/api/films/search?q=${encodeURIComponent(q)}&page=${page}`);

const json = await res.json();

if (!res.ok) throw new Error(json?.error || 'Ошибка поиска');

// json.data.items: массив фильмов (часть с расширенными деталями)

return json.data.items;

}

Для добавления фильма в состав группы клиент передаёт данные выбранной позиции из результатов поиска, сервер сопоставляет поля с внутренней моделью и проверяет отсутствие дубликатов перед сохранением. Реализация запроса выполнена следующим образом:

// Добавление фильма: клиент передаёт фильм из поиска, сервер сопоставляет поля и сохраняет

async function addFilmToGroup(groupCode: string, film: {

kinopoiskId: number; nameRu: string; year: number; posterUrl?: string; description?: string; ratingKinopoisk?: number;

}) {

const code = groupCode.toUpperCase().trim();

const res = await fetch(`/api/groups-firebase/${encodeURIComponent(code)}/films`, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ film })

});

const json = await res.json();

if (!res.ok) throw new Error(json?.error || 'Не удалось добавить фильм');

// json.data.films: обновлённый список фильмов группы

return json.data.films;

}

При голосовании клиент формирует набор оценок для фильмов группы, сервер проверяет формат и допустимые значения и фиксирует оценки. Реализация голосования выполняется следующим образом:

// Отправка голосов: массив оценок для фильмов группы

type VotePayload = { filmId: number; vote: 'like' | 'dislike'; participantId: string };

async function submitVotes(groupCode: string, votes: VotePayload[]) {

if (!votes.length) throw new Error('Нет оценок для отправки');

const code = groupCode.toUpperCase().trim();

const res = await fetch(`/api/groups-firebase/${encodeURIComponent(code)}/votes`, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ votes })

});

const json = await res.json();

if (!res.ok) throw new Error(json?.error || 'Ошибка отправки голосов');

// { success: true, message: ... }

return true;

}

Клиент получает актуальные оценки по группе и на их основе определяет выбранный фильм согласно правилам (максимум положительных, при равенстве – доля положительных, далее – порядок добавления). Реализация запроса выполняется следующим образом:

// Получение голосов: последующее вычисление победителя на клиенте

type Vote = { filmId: number; vote: 'like' | 'dislike' };

async function fetchVotes(groupCode: string): Promise<Vote[]> {

const code = groupCode.toUpperCase().trim();

const res = await fetch(`/api/groups-firebase/${encodeURIComponent(code)}/votes`);

const json = await res.json();

if (!res.ok) throw new Error(json?.error || 'Ошибка загрузки голосов');

// json.data.votes: массив голосов

return json.data.votes as Vote[];

}

Клиент проверяет результаты выполнения запросов и отображает краткие сообщения (по месту ввода и в составе разделов интерфейса). Сервер возвращает согласованные статусы и понятные тексты ошибок, что обеспечивает предсказуемую реакцию интерфейса и облегчает повторный ввод:

// Унифицированная обработка ошибок на клиенте

async function requestJSON(input: RequestInfo, init?: RequestInit) {

const res = await fetch(input, init);

const json = await res.json().catch(() => ({}));

if (!res.ok) {

const msg = json?.error || `Ошибка запроса (${res.status})`;

throw new Error(msg);

}

return json;

}

# Тестирование

Цель тестирования – подтвердить корректность выполнения ключевых сценариев (создание/подключение к группе, поиск и добавление фильмов, голосование, определение результата), устойчивость к ошибкам ввода и сбоям внешнего источника данных, а также предсказуемость поведения интерфейса на мобильных устройствах.

## Тестирование сервера

Наборы тестовых данных для проведения тестирования сервера представлены в табл. 1.

Таблица 1

Тестовые данные для сервера

|  |  |  |  |
| --- | --- | --- | --- |
| № | Исходные данные | Тестируемый модуль/подпрограмма | Ожидаемый результат |
| 1 | 2 | 3 | 4 |
| 1 | {"creatorName":"Киноман"} | route.ts | 200 OK; создана группа с 5-символьным кодом; ответ содержит id, code |
| 2 | {"creatorName":""} | route.ts | 400 Bad Request; сообщение: «Имя создателя группы обязательно» |
| 3 | code=" ab12c " | route.ts | 200 OK (если группа существует); код нормализуется в «AB12C»; возвращены данные группы и films[] |
| 4 | code="ZZZZZ" (несуществующая группа) | route.ts | 404 Not Found; сообщение: «Группа не найдена» |
| 5 | q="Хa" | route.ts | 200 OK; items[] не пуст; часть элементов дополнена деталями |
| 6 | q="A" | route.ts | 400 Bad Request; сообщение о минимальной длине запроса (≥ 2 символов) |
| 7 | {film:{kinopoiskId: 100, nameRu:"Film", year:2020, posterUrl:"..."}} | route.ts | 200 OK; фильм добавлен; в ответе обновлённый films[] |
| 7 | {film:{kinopoiskId: 100, nameRu:"Film", year:2020, posterUrl:"..."}} | route.ts | 200 OK; фильм добавлен; в ответе обновлённый films[] |
| 8 | Повтор запроса № 7 | route.ts | 400 Bad Request; сообщение: «Этот фильм уже добавлен в группу» |

Продолжение табл. 1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 9 | {votes:[{filmId:100, vote:"like", participantId:"u1"},{filmId:101, vote:"dislike", participantId:"u1"}]} | route.ts | 200 OK; сообщение об успешном сохранении |
| 10 | {votes:"not-an-array"} | route.ts | 400 Bad Request; сообщение: «Неверный формат голосов» |
| 11 | Корректный код группы | route.ts | 200 OK; data.votes[] – массив голосов по группе |
| 12 | Корректный код группы | route.ts | 200 OK; сообщение об успешном удалении голосов |
| 13 | q="Matrix" при отсутствии API‑ключа | kinopoisk.ts | 503/500; сообщение об ошибке конфигурации внешнего источника |
| 14 | code отсутствует | route.ts | 400 Bad Request; сообщение: «Код группы обязателен» |
| 15 | film отсутствует | route.ts | 400 Bad Request; сообщение: «Фильм не указан» |

Результаты тестирования сервера представлены в табл. 2.

Таблица 2

Результаты тестирования сервера

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата тестирования | Тестируемый модуль | Тестирование проводил | Описание теста | Результат тестирования |
| 1 | 2 | 3 | 4 | 5 |
| 11.10.2025 | route.ts | Маркелов С. А. | Тестирование с набором данных № 1 | Успех |
| 11.10.2025 | route.ts | Маркелов С. А. | Тестирование с набором данных № 2 | Успех |
| 13.10.2025 | route.ts | Подтелков В. В. | Тестирование с набором данных № 3 | Ошибка: получен код 404 вместо 200 (тестовая группа отсутствовала) |
| 14.10.2025 | route.ts | Подтелков В. В. | Повторное тестирование с набором данных № 3 | Успех |
| 15.10.2025 | route.ts | Подтелков В. В. | Тестирование с набором данных № 4 | Успех |
| 16.10.2025 | route.ts | Маркелов С. А. | Тестирование с набором данных № 5 | Успех |
| 16.10.2025 | route.ts | Маркелов С. А. | Тестирование с набором данных № 6 | Ошибка: получен код 200 вместо 400 (клиент отправил кэшированный запрос) |

Продолжение табл. 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 17.10.2025 | route.ts | Подтелков В. В. | Повторное тестирование с набором данных № 6 | Успех |
| 19.10.2025 | route.ts | Подтелков В. В. | Тестирование с набором данных № 7 | Успех |
| 19.10.2025 | route.ts | Подтелков В. В. | Тестирование с набором данных № 8 | Успех |
| 21.10.2025 | route.ts | Маркелов С. А. | Тестирование с набором данных № 9 | Успех |
| 21.10.2025 | route.ts | Маркелов С. А. | Тестирование с набором данных № 10 | Успех |
| 23.10.2025 | route.ts | Подтелков В. В. | Тестирование с набором данных № 11 | Ошибка: пустой массив голосов (ожидался непустой) |
| 24.10.2025 | route.ts | Подтелков В. В. | Повторное тестирование с набором данных № 11 | Успех |
| 24.10.2025 | route.ts | Подтелков В. В. | Тестирование с набором данных № 12 | Успех |
| 26.10.2025 | kinopoisk.ts | Маркелов С. А | Тестирование с набором данных № 13 | Успех |
| 27.10.2025 | route.ts | Подтелков В. В. | Тестирование с набором данных № 14 | Успех |
| 29.10.2025 | route.ts | Маркелов С. А. | Тестирование с набором данных № 15 | Ошибка: получен код 200 вместо 400 (некорректная валидация film) |
| 30.10.2025 | route.ts | Маркелов С. А. | Повторное тестирование с набором данных № 15 | Успех |

## Тестирование мобильного приложения

Наборы тестовых данных для проведения тестирования мобильного приложения представлены в табл. 3.

Таблица 3

Тестовые данные для мобильного приложения

|  |  |  |  |
| --- | --- | --- | --- |
| № | Исходные данные | Тестируемый модуль/подпрограмма | Ожидаемый результат |
| 1 | 2 | 3 | 4 |
| 1 | Поле «Имя» заполнено: «Киноман» | GroupEntry.tsx | Кнопка «Создать группу» активна, сообщение об ошибке не показывается |
| 2 | Поле «Имя» пустое | GroupEntry.tsx | Появляется сообщение «Укажите имя», переход не инициируется |

Продолжение табл. 3

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 3 | Поле «Код группы»: « ab12c » | GroupEntry.tsx | Значение нормализуется до «AB12C», отправка без локальной ошибки |
| 4 | Поле «Код группы»: пусто | GroupEntry.tsx | Появляется сообщение «Укажите код группы», отправка не выполняется |
| 5 | Поиск: запрос «Хa» | FilmSearch.tsx | Отображаются карточки результатов, ошибок нет |
| 6 | Поиск: запрос «A» | FilmSearch.tsx | Появляется сообщение «Введите не менее 2 символов»; запрос не отправляется |
| 7 | Добавление фильма из результатов | FilmSearch.tsx | Появляется сообщение «Фильм добавлен в группу», контекст поиска сохраняется |
| 8 | Повторное добавление того же фильма | FilmSearch.tsx | Появляется сообщение о наличии фильма, состав списка не меняется |
| 9 | Пустой список группы | GroupScreen.tsx | Текст «Список пуст…», кнопка «Добавить фильм» доступна |
| 10 | Список группы из 5 фильмов | GroupScreen.tsx | Карточки с постером, названием, годом и рейтингом, корректные отступы |
| 11 | Выбор «Нравится» | Voting.tsx | Кнопка подсвечивается зелёным, выбор фиксируется |
| 12 | Выбор «Не нравится» | Voting.tsx | Кнопка подсвечивается красным, выбор фиксируется |
| 13 | Отправка без выбранных оценок | Voting.tsx | Сообщение «Нет оценок для отправки», отправка не выполняется |
| 14 | Отправка с заполненными оценками | Voting.tsx | Сообщение «Оценки учтены», интерфейс в текущем состоянии |
| 15 | Показ результата при наличии голосов | Winner.tsx | Отображается карточка одного выбранного фильма |

Результаты тестирования мобильного приложения представлены в табл. 4.

Таблица 4

Результаты тестирования мобильного приложения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата тестирования | Тестируемый модуль | Тестирование проводил | Описание теста | Результат тестирования |
| 1 | 2 | 3 | 4 | 5 |
| 10.10.2025 | GroupEntry.tsx | Маркелов С. А. | Тестирование с набором данных № 1 | Успех |
| 10.10.2025 | GroupEntry.tsx | Маркелов С. А. | Тестирование с набором данных № 2 | Успех |
| 12.10.2025 | GroupEntry.tsx | Подтелков В. В. | Тестирование с набором данных № 3 | Успех |
| 12.10.2025 | GroupEntry.tsx | Подтелков В. В. | Тестирование с набором данных № 4 | Ошибка: сообщение «Укажите код группы» не отображено |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 13.10.2025 | GroupEntry.tsx | Подтелков В. В. | Повторное тестирование с набором данных № 4 | Успех |
| 14.10.2025 | FilmSearch.tsx | Маркелов С. А. | Тестирование с набором данных № 5 | Успех |
| 14.10.2025 | FilmSearch.tsx | Маркелов С. А. | Тестирование с набором данных № 6 | Успех |
| 16.10.2025 | FilmSearch.tsx | Подтелков В. В. | Тестирование с набором данных № 7 | Успех |
| 16.10.2025 | FilmSearch.tsx | Подтелков В. В. | Тестирование с набором данных № 8 | Ошибка: не показано сообщение о дубле |
| 17.10.2025 | FilmSearch.tsx | Подтелков В. В. | Повторное тестирование с набором данных № 8 | Успех |
| 18.10.2025 | GroupScreen.tsx | Маркелов С. А. | Тестирование с набором данных № 9 | Успех |
| 18.10.2025 | GroupScreen.tsx | Маркелов С. А. | Тестирование с набором данных № 10 | Успех |
| 20.10.2025 | Voting.tsx | Подтелков В. В. | Тестирование с набором данных № 11 | Успех |
| 20.10.2025 | Voting.tsx | Подтелков В. В. | Тестирование с набором данных № 12 | Успех |
| 21.10.2025 | Voting.tsx | Маркелов С. А. | Тестирование с набором данных № 13 | Успех |
| 21.10.2025 | Voting.tsx | Маркелов С. А. | Тестирование с набором данных № 14 | Ошибка: кнопка отправки активна при пустом наборе голосов |
| 22.10.2025 | Voting.tsx | Маркелов С. А. | Повторное тестирование с набором данных № 14 | Успех |
| 23.10.2025 | Winner.tsx | Подтелков В. В. | Тестирование с набором данных № 15 | Успех |

# Заключение

В ходе выполнения проекта было разработано мобильное web-приложение «КиноВыбор», которое представляет собой современное web-приложение, решающее реальную проблему выбора фильма в группе. Приложение построено с использованием технологий Next.js, React TypeScript, Tailwind CSS, Framer Motion, Next.js API Routes, Node.js и Firebase.

Архитектура приложения обеспечивает:

* масштабируемость — легко добавлять новые функции;
* производительность — быстрая загрузка и отзывчивый интерфейс;
* надежность — валидация данных и обработка ошибок;
* удобство использования — интуитивный интерфейс с анимациями.

Приложение интегрировано с API Кинопоиска для получения актуальной информации о фильмах, что позволяет формировать полный каталог с постерами, рейтингами и описаниями.

# Список литературы

1. Методика и организация самостоятельной работы студентов: учебно-методическое пособие / Е.В. Ершов, Л.Н. Виноградова, В.В. Селивановских [и др.]. – Череповец: ФГБОУ ВПО ЧГУ, 2015. – 243 с.
2. Get started with Tailwind CSS — Tailwind CSS [электр.ресурс] https://tailwindcss.com/docs/installation/using-vite. Дата обращения: 08.10.2025.
3. Kinopoisk Unofficial API Documentation — Kinopoisk API Unofficial [электр.ресурс] https://kinopoiskapiunofficial.tech/documentation/api/. Дата обращения: 08.10.2025.
4. Motion x Framer integration guide — Motion [электр.ресурс] https://motion.dev/docs/framer. Дата обращения: 08.10.2025.
5. Next.js Docs — Next.js [электр.ресурс] https://nextjs.org/docs. Дата обращения: 08.10.2025.
6. Quick Start — React [электр.ресурс] https://react.dev/learn. Дата обращения: 08.10.2025.
7. The WebSocket API (WebSockets) — Mozilla Developer Network [электр.ресурс] https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\_API. Дата обращения: 08.10.2025.
8. TypeScript Documentation — TypeScript [электр.ресурс] https://www.typescriptlang.org/docs/. Дата обращения: 08.10.2025.
9. What is REST? — REST API Tutorial [электр.ресурс] https://restfulapi.net/. Дата обращения: 08.10.2025.

# Приложение

Текст программы

## П.1. Мобильное приложение

Файл FilmCard.tsx:

'use client';

import React, { useState } from 'react';

import { motion } from 'framer-motion';

import { Button } from '@/components/ui/button';

import { Plus, Star, Clock, Globe, RotateCcw } from 'lucide-react';

import { KinopoiskFilm } from '@/services/kinopoisk';

import { filmUtils } from '@/services/kinopoisk';

interface FilmCardProps {

film: KinopoiskFilm;

onAdd?: (film: KinopoiskFilm) => void;

showAddButton?: boolean;

className?: string;

}

export default function FilmCard({

film,

onAdd,

showAddButton = true,

className = ''

}: FilmCardProps) {

const [isFlipped, setIsFlipped] = useState(false);

const handleAdd = () => {

if (onAdd) {

onAdd(film);

}

};

const handleFlip = () => {

setIsFlipped(!isFlipped);

};

return (

<motion.div

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ duration: 0.3 }}

className={`bg-gray-800 rounded-xl overflow-hidden shadow-lg hover:shadow-xl transition-shadow ${className}`}

>

{/\* Контейнер для переворота \*/}

<div className="relative aspect-[2/3] bg-gray-700 perspective-1000">

<motion.div

className="relative w-full h-full preserve-3d"

animate={{ rotateY: isFlipped ? 180 : 0 }}

transition={{ duration: 0.6, ease: "easeInOut" }}

>

{/\* Передняя сторона - постер \*/}

<div className="absolute inset-0 w-full h-full backface-hidden">

<div

className="relative w-full h-full cursor-pointer group"

onClick={handleFlip}

>

<img

src={filmUtils.getPosterUrl(film)}

alt={film.nameRu}

className="w-full h-full object-cover transition-transform group-hover:scale-105"

onError={(e) => {

const target = e.target as HTMLImageElement;

target.src = '/placeholder-poster.jpg';

}}

/>

{/\* Рейтинг \*/}

{film.ratingKinopoisk && (

<div className="absolute top-2 right-2 bg-black bg-opacity-70 text-white px-2 py-1 rounded-lg flex items-center gap-1 text-sm">

<Star className="h-3 w-3 fill-yellow-400 text-yellow-400" />

{filmUtils.formatRating(film.ratingKinopoisk)}

</div>

)}

{/\* Индикатор переворота \*/}

<div className="absolute bottom-2 left-2 bg-black bg-opacity-70 text-white px-2 py-1 rounded-lg text-xs opacity-0 group-hover:opacity-100 transition-opacity">

<RotateCcw className="h-3 w-3 inline mr-1" />

Нажмите для описания

</div>

</div>

</div>

{/\* Задняя сторона - описание \*/}

<div

className="absolute inset-0 w-full h-full backface-hidden rotate-y-180 bg-gradient-to-br from-gray-800 to-gray-900 p-4 flex flex-col justify-between cursor-pointer group"

onClick={handleFlip}

>

<div className="flex-1 overflow-y-auto">

<h3 className="font-bold text-white text-lg mb-2 line-clamp-2">

{film.nameRu}

</h3>

{film.nameEn && film.nameEn !== film.nameRu && (

<p className="text-gray-400 text-sm mb-3 line-clamp-1">

{film.nameEn}

</p>

)}

{/\* Описание \*/}

<div className="mb-4">

<h4 className="text-white text-sm font-semibold mb-2">Описание:</h4>

<p className="text-gray-300 text-sm leading-relaxed">

{film.description || 'Описание не доступно'}

</p>

</div>

{/\* Дополнительная информация \*/}

<div className="space-y-2 text-sm">

<div className="flex items-center gap-2 text-gray-400">

<Globe className="h-4 w-4" />

<span>{film.year}</span>

{film.duration && (

<>

<span>•</span>

<Clock className="h-4 w-4" />

<span>{filmUtils.formatDuration(film.duration)}</span>

</>

)}

</div>

{film.genres && film.genres.length > 0 && (

<div className="text-gray-400">

<span className="font-medium">Жанры:</span> {filmUtils.formatGenres(film.genres)}

</div>

)}

{film.countries && film.countries.length > 0 && (

<div className="text-gray-400">

<span className="font-medium">Страны:</span> {filmUtils.formatCountries(film.countries)}

</div>

)}

</div>

</div>

{/\* Подсказка для возврата \*/}

<div className="flex justify-center mt-4">

<div className="bg-black bg-opacity-50 text-white px-3 py-2 rounded-lg text-sm opacity-0 group-hover:opacity-100 transition-opacity">

<RotateCcw className="h-4 w-4 inline mr-2" />

Кликните для возврата к постеру

</div>

</div>

</div>

</motion.div>

</div>

{/\* Кнопка добавления (всегда видима) \*/}

{showAddButton && onAdd && (

<div className="p-4">

<Button

onClick={handleAdd}

className="w-full bg-pink-600 hover:bg-pink-700 text-white"

>

<Plus className="h-4 w-4 mr-2" />

Добавить в группу

</Button>

</div>

)}

</motion.div>

);

}

Файл FilmSearch.tsx:

'use client';

import React, { useEffect, useMemo, useRef, useState } from 'react';

import { motion, AnimatePresence } from 'framer-motion';

import { Button } from '@/components/ui/button';

import { Search, Loader2, AlertCircle, CornerDownLeft } from 'lucide-react';

import FilmCard from '@/components/FilmCard';

import { KinopoiskFilm } from '@/services/kinopoisk';

interface FilmSearchProps {

onAddFilm?: (film: KinopoiskFilm) => void;

className?: string;

}

export default function FilmSearch({ onAddFilm, className = '' }: FilmSearchProps) {

const [query, setQuery] = useState('');

const [films, setFilms] = useState<KinopoiskFilm[]>([]);

const [isLoading, setIsLoading] = useState(false);

const [error, setError] = useState<string | null>(null);

const [hasSearched, setHasSearched] = useState(false);

// Typeahead state

const [suggestions, setSuggestions] = useState<KinopoiskFilm[]>([]);

const [showSuggestions, setShowSuggestions] = useState(false);

const [highlightIndex, setHighlightIndex] = useState<number>(-1);

const suggestionsRef = useRef<HTMLDivElement | null>(null);

const inputRef = useRef<HTMLInputElement | null>(null);

const abortRef = useRef<AbortController | null>(null);

const minChars = 2;

const maxSuggestions = 8;

const isAbortError = (e: unknown): e is DOMException => {

return e instanceof DOMException && e.name === 'AbortError';

};

const fetchSuggestions = useMemo(

() =>

async (q: string) => {

if (abortRef.current) {

abortRef.current.abort();

}

abortRef.current = new AbortController();

try {

const res = await fetch(`/api/films/search?q=${encodeURIComponent(q)}`, {

signal: abortRef.current.signal,

});

const data = await res.json();

if (data.success) {

setSuggestions((data.data.items || []).slice(0, maxSuggestions));

setShowSuggestions(true);

setHighlightIndex(-1);

} else {

setSuggestions([]);

setShowSuggestions(false);

}

} catch (e) {

if (!isAbortError(e)) {

setSuggestions([]);

setShowSuggestions(false);

}

}

},

[]

);

// Debounce user input for typeahead

useEffect(() => {

if (query.trim().length < minChars) {

setSuggestions([]);

setShowSuggestions(false);

return;

}

const id = setTimeout(() => {

fetchSuggestions(query.trim());

}, 300);

return () => clearTimeout(id);

}, [query, fetchSuggestions]);

// Close suggestions on outside click

useEffect(() => {

function onClickOutside(e: MouseEvent) {

if (

suggestionsRef.current &&

!suggestionsRef.current.contains(e.target as Node) &&

inputRef.current &&

e.target !== inputRef.current

) {

setShowSuggestions(false);

}

}

document.addEventListener('mousedown', onClickOutside);

return () => document.removeEventListener('mousedown', onClickOutside);

}, []);

const handleSearch = async () => {

if (!query.trim()) return;

setIsLoading(true);

setError(null);

setHasSearched(true);

setShowSuggestions(false);

try {

const response = await fetch(`/api/films/search?q=${encodeURIComponent(query.trim())}`);

const data = await response.json();

if (data.success) {

setFilms(data.data.items || []);

} else {

setError(data.error || 'Произошла ошибка при поиске');

setFilms([]);

}

} catch (error) {

console.error('Ошибка поиска фильмов:', error);

setError('Произошла ошибка при поиске фильмов');

setFilms([]);

} finally {

setIsLoading(false);

}

};

const selectSuggestion = (idx: number) => {

if (idx < 0 || idx >= suggestions.length) return;

const film = suggestions[idx];

setQuery(film.nameRu || '');

setShowSuggestions(false);

// Покажем результаты по выбранному названию

void handleSearch();

};

const onInputKeyDown = (e: React.KeyboardEvent<HTMLInputElement>) => {

if (!showSuggestions || suggestions.length === 0) {

if (e.key === 'Enter') {

e.preventDefault();

handleSearch();

}

return;

}

if (e.key === 'ArrowDown') {

e.preventDefault();

setHighlightIndex((prev) => (prev + 1) % suggestions.length);

} else if (e.key === 'ArrowUp') {

e.preventDefault();

setHighlightIndex((prev) => (prev - 1 + suggestions.length) % suggestions.length);

} else if (e.key === 'Enter') {

e.preventDefault();

selectSuggestion(highlightIndex === -1 ? 0 : highlightIndex);

} else if (e.key === 'Escape') {

setShowSuggestions(false);

}

};

const handleAddFilm = (film: KinopoiskFilm) => {

if (onAddFilm) {

onAddFilm(film);

}

};

return (

<div className={`space-y-6 ${className}`}>

{/\* Поисковая строка \*/}

<div className="flex gap-3">

<div className="flex-1 relative" ref={suggestionsRef}>

<Search className="absolute left-3 top-1/2 transform -translate-y-1/2 h-5 w-5 text-gray-400" />

<input

ref={inputRef}

type="text"

value={query}

onChange={(e) => setQuery(e.target.value)}

onKeyDown={onInputKeyDown}

placeholder="Поиск фильмов..."

className="w-full pl-10 pr-4 py-3 bg-gray-800 border border-gray-700 rounded-xl text-white placeholder-gray-400 focus:outline-none focus:ring-2 focus:ring-pink-500 focus:border-transparent"

/>

{/\* Подсказки \*/}

<AnimatePresence>

{showSuggestions && suggestions.length > 0 && (

<motion.ul

initial={{ opacity: 0, y: -4 }}

animate={{ opacity: 1, y: 0 }}

exit={{ opacity: 0, y: -4 }}

className="absolute z-20 mt-2 w-full max-h-80 overflow-auto rounded-xl border border-gray-700 bg-gray-900 shadow-xl"

role="listbox"

>

{suggestions.map((s, idx) => (

<li

key={s.kinopoiskId}

role="option"

aria-selected={idx === highlightIndex}

onMouseEnter={() => setHighlightIndex(idx)}

onMouseDown={(e) => {

// предотвратить blur до onClick

e.preventDefault();

}}

onClick={() => selectSuggestion(idx)}

className={

`flex items-center gap-3 px-3 py-2 cursor-pointer ` +

(idx === highlightIndex ? 'bg-gray-800' : 'hover:bg-gray-800')

}

>

<div className="flex-1 min-w-0">

<div className="text-sm text-white truncate">{s.nameRu}</div>

<div className="text-xs text-gray-400 truncate">

{(s.year ? `${s.year} • ` : '') + (s.genres?.map(g => g.genre).join(', ') || '')}

</div>

</div>

<kbd className="hidden sm:flex items-center gap-1 text-[10px] text-gray-400">

<CornerDownLeft className="h-3 w-3" /> Enter

</kbd>

</li>

))}

</motion.ul>

)}

</AnimatePresence>

</div>

<Button

onClick={handleSearch}

disabled={!query.trim() || isLoading}

className="px-6 py-3 bg-pink-600 hover:bg-pink-700 disabled:bg-gray-600 disabled:cursor-not-allowed rounded-xl"

>

{isLoading ? (

<Loader2 className="h-5 w-5 animate-spin" />

) : (

<Search className="h-5 w-5" />

)}

</Button>

</div>

{/\* Сообщение об ошибке \*/}

<AnimatePresence>

{error && (

<motion.div

initial={{ opacity: 0, y: -10 }}

animate={{ opacity: 1, y: 0 }}

exit={{ opacity: 0, y: -10 }}

className="flex items-center gap-2 p-4 bg-red-900 bg-opacity-50 border border-red-700 rounded-xl text-red-200"

>

<AlertCircle className="h-5 w-5" />

<span>{error}</span>

</motion.div>

)}

</AnimatePresence>

{/\* Результаты поиска \*/}

<AnimatePresence>

{hasSearched && !isLoading && (

<motion.div

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

exit={{ opacity: 0 }}

className="space-y-4"

>

{films.length > 0 ? (

<>

<h3 className="text-lg font-semibold text-white">

Найдено фильмов: {films.length}

</h3>

<div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-4">

{films.map((film) => (

<FilmCard

key={film.kinopoiskId}

film={film}

onAdd={handleAddFilm}

showAddButton={!!onAddFilm}

/>

))}

</div>

</>

) : (

<div className="text-center py-8">

<p className="text-gray-400 text-lg">

Фильмы не найдены. Попробуйте другой запрос.

</p>

</div>

)}

</motion.div>

)}

</AnimatePresence>

{/\* Состояние загрузки \*/}

<AnimatePresence>

{isLoading && (

<motion.div

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

exit={{ opacity: 0 }}

className="flex items-center justify-center py-8"

>

<div className="flex items-center gap-3 text-gray-400">

<Loader2 className="h-6 w-6 animate-spin" />

<span>Поиск фильмов...</span>

</div>

</motion.div>

)}

</AnimatePresence>

</div>

);

}

Файл GroupPageClient.tsx:

'use client';

import React, { useState, useEffect } from "react";

import { motion } from "framer-motion";

import { Button } from "@/components/ui/button";

import { ArrowLeft, Users, Plus, Play, Film as FilmIcon, Wifi, WifiOff, Copy, Check } from "lucide-react";

import FilmSearch from "@/components/FilmSearch";

import { KinopoiskFilm } from "@/services/kinopoisk";

// import { Film } from "@/lib/database"; // Используется через WebSocket

import { useGroupSocket } from "@/hooks/useSocket";

interface GroupPageClientProps {

groupCode: string;

}

export default function GroupPageClient({ groupCode }: GroupPageClientProps) {

const [isCreator, setIsCreator] = useState(false);

const [groupCreator, setGroupCreator] = useState<string | null>(null);

const [copied, setCopied] = useState(false);

const [showFilmSearch, setShowFilmSearch] = useState(false);

// Получаем никнейм из URL параметров, sessionStorage или используем дефолтный

const [participantName] = useState(() => {

if (typeof window !== 'undefined') {

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

const nicknameFromUrl = urlParams.get('nickname');

if (nicknameFromUrl) {

// Сохраняем никнейм в sessionStorage для последующего использования

sessionStorage.setItem(`nickname\_${groupCode}`, nicknameFromUrl);

return nicknameFromUrl;

}

// Пробуем получить из sessionStorage

const nicknameFromStorage = sessionStorage.getItem(`nickname\_${groupCode}`);

if (nicknameFromStorage) {

return nicknameFromStorage;

}

return 'Участник';

}

return 'Участник';

});

// Используем WebSocket для реального времени

const {

isConnected,

participants,

films,

addFilm: addFilmRealtime,

startVoting: startVotingRealtime

} = useGroupSocket(groupCode, participantName);

// Загружаем данные группы для определения создателя

useEffect(() => {

const loadGroup = async () => {

try {

const response = await fetch(`/api/groups-firebase?code=${groupCode}`);

const data = await response.json();

if (data.success) {

const creator = data.data.createdBy;

setGroupCreator(creator);

setIsCreator(participantName === creator);

} else {

console.error('Ошибка загрузки группы:', data.error);

}

} catch (error) {

console.error('Ошибка загрузки группы:', error);

}

};

if (groupCode) {

loadGroup();

}

}, [groupCode, participantName]);

const handleAddFilm = async (film: KinopoiskFilm) => {

try {

// Используем WebSocket для добавления фильма в реальном времени

if (isConnected) {

addFilmRealtime(film);

setShowFilmSearch(false);

} else {

// Fallback на обычный API, если WebSocket не подключен

const response = await fetch(`/api/groups-firebase/${groupCode}/films`, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ film }),

});

const data = await response.json();

if (data.success) {

setShowFilmSearch(false);

} else {

alert(data.error || 'Ошибка добавления фильма');

}

}

} catch (error) {

console.error('Ошибка добавления фильма:', error);

alert('Ошибка добавления фильма');

}

};

const handleStartVoting = () => {

if (films.length === 0) {

alert('Добавьте хотя бы один фильм перед началом голосования');

return;

}

// Используем WebSocket для начала голосования (все участники перейдут автоматически)

if (isConnected && startVotingRealtime) {

startVotingRealtime(films);

} else {

// Fallback: переход только для текущего пользователя

window.location.href = `/vote/${groupCode}?nickname=${encodeURIComponent(participantName)}`;

}

};

return (

<div className="flex flex-col min-h-screen bg-gradient-to-b from-black via-gray-900 to-gray-800 text-white">

{/\* Header \*/}

<div className="flex items-center justify-between p-4 border-b border-gray-800">

<div className="flex items-center">

<Button

variant="ghost"

size="icon"

onClick={() => window.location.href = '/'}

className="mr-4 text-gray-400 hover:text-white"

>

<ArrowLeft className="h-5 w-5" />

</Button>

<div>

<div className="flex items-center gap-2">

<h1 className="text-xl font-bold">Группа {groupCode}</h1>

{isConnected ? (

<Wifi className="h-4 w-4 text-green-500" />

) : (

<WifiOff className="h-4 w-4 text-red-500" />

)}

</div>

<p className="text-sm text-gray-400">Код для приглашения друзей</p>

</div>

</div>

<div className="text-right">

<div className="text-2xl font-bold tracking-widest">{groupCode}</div>

<Button

size="sm"

className="mt-1 text-xs bg-gray-800 border border-gray-700 text-white hover:bg-gray-700 hover:border-gray-600 transition-all duration-200"

onClick={async () => {

try {

await navigator.clipboard.writeText(groupCode);

setCopied(true);

setTimeout(() => setCopied(false), 2000);

} catch (err) {

console.error('Ошибка копирования:', err);

}

}}

>

{copied ? (

<span className="flex items-center gap-1">

<Check className="h-3 w-3" />

Скопировано

</span>

) : (

<span className="flex items-center gap-1">

<Copy className="h-3 w-3" />

Копировать

</span>

)}

</Button>

</div>

</div>

<div className="flex-1 p-4">

{/\* Participants \*/}

<motion.div

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ duration: 0.6 }}

className="mb-8"

>

<div className="flex items-center mb-4">

<Users className="h-5 w-5 mr-2" />

<h2 className="text-lg font-semibold">Участники</h2>

<span className="ml-2 text-sm text-gray-400">({participants.length})</span>

</div>

<div className="bg-gray-800 rounded-xl p-4 min-h-[100px] flex items-center justify-center">

{participants.length === 0 ? (

<p className="text-gray-400 text-center">

Пока никто не присоединился к группе

</p>

) : (

<div className="flex flex-wrap gap-2">

{participants.map((participant, index) => (

<div

key={index}

className="bg-pink-600 px-3 py-1 rounded-full text-sm"

>

{participant}

</div>

))}

</div>

)}

</div>

</motion.div>

{/\* Films \*/}

<motion.div

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.2, duration: 0.6 }}

className="mb-8"

>

<div className="flex items-center justify-between mb-4">

<div className="flex items-center">

<FilmIcon className="h-5 w-5 mr-2" />

<h2 className="text-lg font-semibold">Фильмы</h2>

<span className="ml-2 text-sm text-gray-400">({films.length})</span>

</div>

<Button

onClick={() => setShowFilmSearch(!showFilmSearch)}

size="sm"

className="bg-indigo-600 hover:bg-indigo-700"

>

<Plus className="h-4 w-4 mr-1" />

{showFilmSearch ? 'Скрыть поиск' : 'Добавить фильм'}

</Button>

</div>

{/\* Поиск фильмов \*/}

{showFilmSearch && (

<motion.div

initial={{ opacity: 0, height: 0 }}

animate={{ opacity: 1, height: 'auto' }}

exit={{ opacity: 0, height: 0 }}

className="mb-6"

>

<FilmSearch onAddFilm={handleAddFilm} />

</motion.div>

)}

{/\* Список фильмов \*/}

<div className="bg-gray-800 rounded-xl p-4 min-h-[200px]">

{films.length === 0 ? (

<div className="flex items-center justify-center h-48">

<div className="text-center">

<FilmIcon className="h-12 w-12 text-gray-600 mx-auto mb-3" />

<p className="text-gray-400 mb-2">Фильмы еще не добавлены</p>

<p className="text-sm text-gray-500">

Используйте поиск выше, чтобы добавить фильмы

</p>

</div>

</div>

) : (

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

{films.map((film) => (

<div key={film.id} className="max-w-xs mx-auto">

<div className="bg-gray-700 rounded-xl overflow-hidden shadow-lg">

{/\* Постер \*/}

<div className="relative aspect-[2/3] bg-gray-600">

<img

src={film.poster || '/placeholder-poster.jpg'}

alt={film.title}

className="w-full h-full object-cover"

onError={(e) => {

const target = e.target as HTMLImageElement;

target.src = '/placeholder-poster.jpg';

}}

/>

{/\* Рейтинг \*/}

{film.rating && (

<div className="absolute top-2 right-2 bg-black bg-opacity-70 text-white px-2 py-1 rounded-lg flex items-center gap-1 text-sm">

<span className="text-yellow-400">★</span>

{film.rating.toFixed(1)}

</div>

)}

</div>

{/\* Информация о фильме \*/}

<div className="p-4">

<h3 className="font-bold text-white text-lg mb-2 line-clamp-2">

{film.title}

</h3>

<div className="flex items-center gap-2 text-sm text-gray-400 mb-2">

<span>{film.year}</span>

</div>

{/\* Описание \*/}

{film.description && (

<p className="text-gray-400 text-sm line-clamp-3">

{film.description}

</p>

)}

</div>

</div>

</div>

))}

</div>

)}

</div>

</motion.div>

{/\* Actions \*/}

{isCreator && (

<motion.div

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.4, duration: 0.6 }}

>

<Button

onClick={handleStartVoting}

disabled={films.length < 2 || participants.length < 2}

className="w-full py-4 text-lg bg-pink-600 hover:bg-pink-700 disabled:bg-gray-600 disabled:cursor-not-allowed rounded-xl shadow-lg transition"

>

<Play className="h-5 w-5 mr-2" />

Начать выбор фильма

</Button>

<p className="text-xs text-gray-400 mt-2 text-center">

{films.length < 2 || participants.length < 2

? 'Нужно минимум 2 участника и 2 фильма для начала голосования'

: 'Готово к началу голосования'}

</p>

</motion.div>

)}

</div>

</div>

);

}

Файл NicknameInput.tsx:

'use client';

import React, { useState } from 'react';

import { motion } from 'framer-motion';

import { Button } from '@/components/ui/button';

import { ArrowLeft, User, Sparkles } from 'lucide-react';

interface NicknameInputProps {

onNicknameSubmit: (nickname: string) => void;

onBack: () => void;

title?: string;

subtitle?: string;

}

export default function NicknameInput({

onNicknameSubmit,

onBack,

title = "Введите ваш никнейм",

subtitle = "Выберите имя, под которым вас будут видеть другие участники"

}: NicknameInputProps) {

const [nickname, setNickname] = useState('');

const [isSubmitting, setIsSubmitting] = useState(false);

// Генерируем случайные предложения никнеймов

const suggestedNicknames = [

'Кинокритик', 'Фильмолюб', 'Киноман', 'Режиссер', 'Актер',

'Сценарист', 'Продюсер', 'Оператор', 'Монтажер', 'Звукорежиссер',

'Костюмер', 'Гример', 'Каскадер', 'Дублер', 'Статист',

'Критик', 'Обозреватель', 'Блогер', 'Журналист', 'Редактор'

];

const handleSubmit = async (e: React.FormEvent) => {

e.preventDefault();

if (!nickname.trim()) return;

setIsSubmitting(true);

// Небольшая задержка для анимации

setTimeout(() => {

onNicknameSubmit(nickname.trim());

setIsSubmitting(false);

}, 500);

};

const handleSuggestedClick = (suggested: string) => {

setNickname(suggested);

};

return (

<div className="flex flex-col items-center justify-center min-h-screen bg-gradient-to-b from-black via-gray-900 to-gray-800 text-white px-4">

<motion.div

initial={{ opacity: 0, y: -20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ duration: 0.6 }}

className="w-full max-w-md"

>

{/\* Header \*/}

<div className="flex items-center mb-8">

<Button

variant="ghost"

size="icon"

onClick={onBack}

className="mr-4 text-gray-400 hover:text-white"

>

<ArrowLeft className="h-5 w-5" />

</Button>

<h1 className="text-2xl font-bold">{title}</h1>

</div>

<motion.div

initial={{ opacity: 0, y: 30 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.2, duration: 0.6 }}

className="space-y-6"

>

{/\* Иконка \*/}

<div className="w-16 h-16 bg-indigo-600 rounded-full flex items-center justify-center mx-auto">

<User className="h-8 w-8" />

</div>

{/\* Описание \*/}

<div className="text-center">

<h2 className="text-xl font-semibold mb-2">Выберите никнейм</h2>

<p className="text-gray-400 text-sm">{subtitle}</p>

</div>

{/\* Форма ввода \*/}

<form onSubmit={handleSubmit} className="space-y-4">

<div>

<label htmlFor="nickname" className="block text-sm font-medium text-gray-300 mb-2">

Ваш никнейм

</label>

<input

id="nickname"

type="text"

value={nickname}

onChange={(e) => setNickname(e.target.value)}

placeholder="Введите ваш никнейм"

maxLength={20}

className="w-full px-4 py-3 bg-gray-800 border border-gray-700 rounded-xl text-white placeholder-gray-400 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-transparent text-center text-lg"

required

/>

<p className="text-xs text-gray-400 mt-2 text-center">

Максимум 20 символов

</p>

</div>

<Button

type="submit"

disabled={!nickname.trim() || isSubmitting}

className="w-full py-4 text-lg bg-indigo-600 hover:bg-indigo-700 disabled:bg-gray-600 disabled:cursor-not-allowed rounded-xl shadow-lg transition"

>

{isSubmitting ? (

<div className="flex items-center">

<div className="animate-spin rounded-full h-5 w-5 border-b-2 border-white mr-2"></div>

Подключение...

</div>

) : (

'Продолжить'

)}

</Button>

</form>

{/\* Предложенные никнеймы \*/}

<div className="space-y-3">

<div className="flex items-center gap-2 text-gray-400 text-sm">

<Sparkles className="h-4 w-4" />

<span>Или выберите из предложенных:</span>

</div>

<div className="flex flex-wrap gap-2 justify-center">

{suggestedNicknames.slice(0, 8).map((suggested, index) => (

<motion.button

key={suggested}

type="button"

onClick={() => handleSuggestedClick(suggested)}

whileHover={{ scale: 1.05 }}

whileTap={{ scale: 0.95 }}

className={`px-3 py-1 rounded-full text-sm transition-colors ${

nickname === suggested

? 'bg-indigo-600 text-white'

: 'bg-gray-700 text-gray-300 hover:bg-gray-600'

}`}

>

{suggested}

</motion.button>

))}

</div>

</div>

</motion.div>

</motion.div>

</div>

);

}

Файл ResultsPageClient.tsx:

'use client';

import React, { useState, useEffect } from 'react';

import { motion } from 'framer-motion';

import { ArrowLeft, Users, Trophy, Heart, X, RotateCcw, CheckCircle, AlertCircle } from 'lucide-react';

import { Button } from '@/components/ui/button';

import { findMatches, MatchingStats, Vote } from '@/lib/matching';

import { Film } from '@/lib/groups';

import { Film as DatabaseFilm } from '@/lib/database';

import { useGroupSocket } from '@/hooks/useSocket';

interface ResultsPageClientProps {

groupCode: string;

}

export default function ResultsPageClient({ groupCode }: ResultsPageClientProps) {

const [group, setGroup] = useState<{ participants: string[]; films: Film[]; createdBy?: string } | null>(null);

const [votes, setVotes] = useState<Vote[]>([]);

const [matchingStats, setMatchingStats] = useState<MatchingStats | null>(null);

const [isLoading, setIsLoading] = useState(true);

const [error, setError] = useState<string | null>(null);

// Получаем никнейм из URL или sessionStorage

const getParticipantName = (): string => {

if (typeof window !== 'undefined') {

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

const nicknameFromUrl = urlParams.get('nickname');

if (nicknameFromUrl) {

sessionStorage.setItem(`nickname\_${groupCode}`, nicknameFromUrl);

return nicknameFromUrl;

}

const nicknameFromStorage = sessionStorage.getItem(`nickname\_${groupCode}`);

return nicknameFromStorage || 'Участник';

}

return 'Участник';

};

// Загрузка данных группы и голосов

useEffect(() => {

const loadData = async () => {

try {

setIsLoading(true);

// Нормализуем код группы

const normalizedCode = groupCode?.toUpperCase().trim() || groupCode;

// Загружаем данные группы

const groupResponse = await fetch(`/api/groups-firebase?code=${encodeURIComponent(normalizedCode)}`);

const groupData = await groupResponse.json();

if (!groupData.success) {

setError(groupData.error || 'Группа не найдена');

return;

}

setGroup(groupData.data);

// Загружаем голоса

const votesResponse = await fetch(`/api/groups-firebase/${normalizedCode}/votes`);

const votesData = await votesResponse.json();

if (votesData.success) {

setVotes(votesData.data.votes || []);

}

// Вычисляем совпадения и результаты

if (groupData.data.films && groupData.data.participants) {

const votes = votesData.data.votes || [];

const films = groupData.data.films;

const participants = groupData.data.participants;

// Проверяем SUPER MATCH: все участники выбрали один фильм

const votesByFilm = new Map<number, { likes: number; dislikes: number; voters: string[] }>();

votes.forEach((vote: Vote) => {

const filmId = Number(vote.filmId);

if (!votesByFilm.has(filmId)) {

votesByFilm.set(filmId, { likes: 0, dislikes: 0, voters: [] });

}

const filmVotes = votesByFilm.get(filmId)!;

if (vote.vote === 'like') {

filmVotes.likes++;

} else {

filmVotes.dislikes++;

}

if (!filmVotes.voters.includes(vote.participantId)) {

filmVotes.voters.push(vote.participantId);

}

});

// Находим SUPER MATCH и лучший фильм

let superMatch: any = null;

let bestMatch: any = null;

films.forEach((film: DatabaseFilm) => {

const filmVotes = votesByFilm.get(Number(film.kinopoiskId));

if (!filmVotes) return;

// SUPER MATCH: все участники лайкнули

if (filmVotes.likes === participants.length && filmVotes.dislikes === 0) {

superMatch = { film, voters: filmVotes.voters, likes: filmVotes.likes };

}

// Лучший фильм по количеству лайков

if (!bestMatch || filmVotes.likes > bestMatch.likes) {

bestMatch = { film, likes: filmVotes.likes, dislikes: filmVotes.dislikes, voters: filmVotes.voters };

}

});

// Создаем расширенную статистику

const stats = findMatches(votes, films, participants);

setMatchingStats({

...(stats as any),

superMatch,

bestMatch

} as any);

}

} catch (err) {

setError('Ошибка загрузки данных');

console.error('Ошибка загрузки данных:', err);

} finally {

setIsLoading(false);

}

};

loadData();

}, [groupCode]);

// Удаляем фильмы и голоса после загрузки результатов (с задержкой, чтобы результаты успели отобразиться)

useEffect(() => {

if (!isLoading && matchingStats) {

// Удаляем через 30 секунд после показа результатов (достаточно времени для просмотра)

const cleanupTimer = setTimeout(async () => {

try {

const response = await fetch(`/api/groups-firebase/${groupCode}/cleanup`, {

method: 'POST',

});

if (!response.ok) {

console.error('Ошибка очистки данных группы:', await response.text());

} else {

console.log('Данные группы (фильмы и голоса) успешно удалены');

}

} catch (error) {

console.error('Ошибка при очистке данных группы:', error);

}

}, 30000); // 30 секунд задержка

return () => clearTimeout(cleanupTimer);

}

}, [isLoading, matchingStats, groupCode]);

// Инициализируем сокет для возможности широковещательного сброса

const { resetGroup } = useGroupSocket(groupCode, getParticipantName());

const handleNewVoting = async () => {

// Создатель запускает сброс через сокет — сервер очистит фильмы/голоса и сообщит всем

resetGroup();

};

if (isLoading) {

return (

<div className="min-h-screen bg-gradient-to-b from-black via-gray-900 to-gray-800 flex items-center justify-center">

<div className="text-center">

<div className="animate-spin rounded-full h-12 w-12 border-b-2 border-pink-500 mx-auto mb-4"></div>

<p className="text-white text-lg">Анализ результатов...</p>

</div>

</div>

);

}

if (error) {

return (

<div className="min-h-screen bg-gradient-to-b from-black via-gray-900 to-gray-800 flex items-center justify-center">

<div className="text-center max-w-md mx-auto px-4">

<AlertCircle className="h-16 w-16 text-red-500 mx-auto mb-4" />

<h1 className="text-2xl font-bold text-white mb-4">Ошибка</h1>

<p className="text-gray-300 mb-6">{error}</p>

<Button

onClick={() => window.location.href = '/'}

className="bg-pink-600 hover:bg-pink-700"

>

<ArrowLeft className="h-4 w-4 mr-2" />

На главную

</Button>

</div>

</div>

);

}

if (!matchingStats) {

return (

<div className="min-h-screen bg-gradient-to-b from-black via-gray-900 to-gray-800 flex items-center justify-center">

<div className="text-center max-w-md mx-auto px-4">

<AlertCircle className="h-16 w-16 text-yellow-500 mx-auto mb-4" />

<h1 className="text-2xl font-bold text-white mb-4">Нет данных</h1>

<p className="text-gray-300 mb-6">

Не удалось загрузить результаты голосования.

</p>

<Button

onClick={() => window.location.href = '/'}

className="bg-pink-600 hover:bg-pink-700"

>

<ArrowLeft className="h-4 w-4 mr-2" />

На главную

</Button>

</div>

</div>

);

}

const hasPerfectMatch = matchingStats.matches.length > 0;

const hasPartialMatch = matchingStats.partialMatches.length > 0;

return (

<div className="min-h-screen bg-gradient-to-b from-black via-gray-900 to-gray-800 text-white">

{/\* Заголовок \*/}

<div className="container mx-auto px-4 py-4 md:py-6">

<div className="flex flex-col md:flex-row md:items-center md:justify-center gap-4 mb-6 md:mb-8">

<div className="text-center md:text-left md:flex-1 md:px-4">

<h1 className="text-2xl md:text-3xl font-bold">Результаты голосования</h1>

</div>

</div>

{/\* Результаты \*/}

{matchingStats && (matchingStats as any).superMatch ? (

<motion.div

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

className="mb-8"

>

<h2 className="text-2xl md:text-4xl font-bold text-yellow-400 mb-6 flex flex-col md:flex-row items-center justify-center gap-2 md:gap-3">

<Trophy className="h-6 w-6 md:h-8 md:w-8" />

<span className="text-center whitespace-nowrap">🎉 СУПЕР СОВПАДЕНИЕ! 🎉</span>

</h2>

<div className="bg-gradient-to-r from-yellow-900 via-yellow-800 to-yellow-900 rounded-2xl p-6 md:p-8 border-4 border-yellow-500 shadow-2xl">

{/\* Постер \*/}

<div className="flex justify-center mb-6">

<img

src={(matchingStats as any).superMatch.film.poster || '/placeholder-poster.jpg'}

alt={(matchingStats as any).superMatch.film.title}

className="w-48 md:w-40 h-72 md:h-60 object-cover rounded-xl shadow-lg"

onError={(e) => {

const target = e.target as HTMLImageElement;

target.src = '/placeholder-poster.jpg';

}}

/>

</div>

{/\* Информация о фильме \*/}

<div className="text-center md:text-left">

<h3 className="text-2xl md:text-3xl font-bold text-white mb-2">{(matchingStats as any).superMatch.film.title}</h3>

<p className="text-yellow-200 text-lg md:text-xl mb-6">{(matchingStats as any).superMatch.film.year}</p>

<div className="bg-yellow-700 bg-opacity-70 rounded-lg p-4 mb-6">

<p className="text-yellow-100 text-base md:text-lg font-semibold">

✨ Все {(matchingStats as any).superMatch.voters.length} участников выбрали этот фильм!

</p>

</div>

{(matchingStats as any).superMatch.film.description && (

<div className="mt-6">

<p className="text-gray-200 text-sm md:text-base leading-relaxed">{(matchingStats as any).superMatch.film.description}</p>

</div>

)}

</div>

</div>

</motion.div>

) : matchingStats && (matchingStats as any).bestMatch ? (

<motion.div

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

className="mb-8"

>

<h2 className="text-2xl font-bold text-pink-400 mb-6 flex items-center">

<Trophy className="h-6 w-6 mr-2" />

🎬 Лучший выбор группы

</h2>

<div className="bg-gradient-to-r from-pink-900 to-pink-800 rounded-2xl p-6 border border-pink-600">

{/\* Постер \*/}

<div className="flex justify-center mb-6">

<img

src={(matchingStats as any).bestMatch.film.poster || '/placeholder-poster.jpg'}

alt={(matchingStats as any).bestMatch.film.title}

className="w-48 md:w-32 h-72 md:h-48 object-cover rounded-xl shadow-lg"

onError={(e) => {

const target = e.target as HTMLImageElement;

target.src = '/placeholder-poster.jpg';

}}

/>

</div>

{/\* Информация о фильме \*/}

<div className="text-center md:text-left">

<h3 className="text-xl md:text-2xl font-bold text-white mb-2">{(matchingStats as any).bestMatch.film.title}</h3>

<p className="text-pink-200 text-base md:text-lg mb-4">{(matchingStats as any).bestMatch.film.year}</p>

<div className="flex items-center justify-center md:justify-start gap-4 mb-4">

<div className="flex items-center gap-2 text-pink-400">

<Heart className="h-5 w-5 fill-current" />

<span className="font-semibold">{(matchingStats as any).bestMatch.likes} лайков</span>

</div>

<div className="flex items-center gap-2 text-gray-400">

<X className="h-5 w-5" />

<span>{(matchingStats as any).bestMatch.dislikes} дизлайков</span>

</div>

</div>

<div className="bg-pink-700 bg-opacity-50 rounded-lg p-3 mb-6">

<p className="text-pink-200 text-sm">

<strong>{(matchingStats as any).bestMatch.likes} из {matchingStats.totalParticipants} участников выбрали этот фильм</strong>

</p>

</div>

{(matchingStats as any).bestMatch.film.description && (

<div className="mt-6">

<p className="text-gray-300 text-sm md:text-base leading-relaxed">{(matchingStats as any).bestMatch.film.description}</p>

</div>

)}

</div>

</div>

</motion.div>

) : hasPerfectMatch ? (

<motion.div

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

className="mb-8"

>

<h2 className="text-2xl font-bold text-green-400 mb-6 flex items-center">

<Trophy className="h-6 w-6 mr-2" />

🎉 Идеальные совпадения!

</h2>

<div className="space-y-6">

{matchingStats.matches.map((match, index) => (

<div key={match.film.id} className="bg-gradient-to-r from-green-900 to-green-800 rounded-2xl p-6 border border-green-600">

{/\* Постер фильма \*/}

<div className="flex justify-center mb-6">

<img

src={match.film.poster || '/placeholder-poster.jpg'}

alt={match.film.title}

className="w-48 md:w-32 h-72 md:h-48 object-cover rounded-xl shadow-lg"

onError={(e) => {

const target = e.target as HTMLImageElement;

target.src = '/placeholder-poster.jpg';

}}

/>

</div>

{/\* Информация о фильме \*/}

<div className="text-center md:text-left">

<h3 className="text-xl md:text-2xl font-bold text-white mb-2">{match.film.title}</h3>

<p className="text-green-200 text-base md:text-lg mb-4">{match.film.year}</p>

<div className="flex items-center justify-center md:justify-start gap-4 mb-4">

<div className="flex items-center gap-2 text-green-400">

<Heart className="h-5 w-5 fill-current" />

<span className="font-semibold">{match.likes} лайков</span>

</div>

<div className="flex items-center gap-2 text-gray-400">

<X className="h-5 w-5" />

<span>{match.dislikes} дизлайков</span>

</div>

</div>

<div className="bg-green-700 bg-opacity-50 rounded-lg p-3 mb-6">

<p className="text-green-200 text-sm">

<strong>

{match.participants.length === 1

? 'Вы выбрали этот фильм!'

: `Все участники (${match.participants.length}) выбрали этот фильм!`}

</strong>

</p>

</div>

{match.film.description && (

<div className="mt-6">

<p className="text-gray-300 text-sm md:text-base leading-relaxed">{match.film.description}</p>

</div>

)}

</div>

</div>

))}

</div>

</motion.div>

) : hasPartialMatch ? (

<motion.div

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

className="mb-8"

>

<h2 className="text-2xl font-bold text-yellow-400 mb-6 flex items-center">

<Trophy className="h-6 w-6 mr-2" />

🤝 Частичные совпадения

</h2>

<div className="space-y-4">

{matchingStats.partialMatches.map((match, index) => (

<div key={match.film.id} className="bg-gradient-to-r from-yellow-900 to-yellow-800 rounded-2xl p-6 border border-yellow-600">

<div className="flex items-start gap-6">

<img

src={match.film.poster || '/placeholder-poster.jpg'}

alt={match.film.title}

className="w-24 h-36 object-cover rounded-xl shadow-lg"

onError={(e) => {

const target = e.target as HTMLImageElement;

target.src = '/placeholder-poster.jpg';

}}

/>

<div className="flex-1">

<h3 className="text-xl font-bold text-white mb-2">{match.film.title}</h3>

<p className="text-yellow-200 mb-2">{match.film.year}</p>

<div className="flex items-center gap-4 mb-3">

<div className="flex items-center gap-2 text-yellow-400">

<Heart className="h-4 w-4 fill-current" />

<span className="font-semibold">{match.likes} лайков</span>

</div>

<div className="flex items-center gap-2 text-gray-400">

<X className="h-4 w-4" />

<span>{match.dislikes} дизлайков</span>

</div>

</div>

<div className="bg-yellow-700 bg-opacity-50 rounded-lg p-3">

<p className="text-yellow-200 text-sm">

<strong>{Math.round(match.matchPercentage)}%</strong> участников выбрали этот фильм

</p>

</div>

</div>

</div>

</div>

))}

</div>

</motion.div>

) : (

<motion.div

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

className="text-center py-12"

>

<AlertCircle className="h-16 w-16 text-gray-500 mx-auto mb-4" />

<h2 className="text-2xl font-bold text-white mb-4">Совпадений не найдено</h2>

<p className="text-gray-400 mb-6">

{matchingStats.totalParticipants === 1

? 'Вы не выбрали ни одного фильма, который вам понравился.'

: 'К сожалению, не удалось найти фильм, который понравился бы всем участникам.'}

</p>

<p className="text-gray-500 text-sm">

Попробуйте добавить больше фильмов или начать новое голосование.

</p>

</motion.div>

)}

{/\* Кнопки действий \*/}

<motion.div

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.2 }}

className="flex justify-center"

>

{group && getParticipantName() === (group.createdBy || '') && (

<Button

onClick={handleNewVoting}

className="bg-pink-600 hover:bg-pink-700 text-white px-8 py-3"

>

<RotateCcw className="h-5 w-5 mr-2" />

Начать новое голосование

</Button>

)}

</motion.div>

</div>

</div>

);

}

Файл VotingCard.tsx:

'use client';

import React, { useState, useRef, useEffect } from 'react';

import { motion, PanInfo } from 'framer-motion';

import { Heart, X, Star, Globe, Clock, Info } from 'lucide-react';

import { KinopoiskFilm } from '@/services/kinopoisk';

import { filmUtils } from '@/services/kinopoisk';

interface VotingCardProps {

film: KinopoiskFilm;

onVote: (filmId: number, vote: 'like' | 'dislike') => void;

onSkip?: () => void;

className?: string;

}

export default function VotingCard({

film,

onVote,

onSkip,

className = ''

}: VotingCardProps) {

const [showDescription, setShowDescription] = useState(false);

const [dragDirection, setDragDirection] = useState<'left' | 'right' | null>(null);

const descriptionRef = useRef<HTMLDivElement>(null);

const handleDrag = (event: unknown, info: PanInfo) => {

const threshold = 50;

if (info.offset.x > threshold) {

setDragDirection('right');

} else if (info.offset.x < -threshold) {

setDragDirection('left');

} else {

setDragDirection(null);

}

};

const handleDragEnd = (event: unknown, info: PanInfo) => {

const threshold = 100;

if (info.offset.x > threshold) {

onVote(film.kinopoiskId, 'like');

} else if (info.offset.x < -threshold) {

onVote(film.kinopoiskId, 'dislike');

}

setDragDirection(null);

};

const handleLike = () => {

onVote(film.kinopoiskId, 'like');

};

const handleDislike = () => {

onVote(film.kinopoiskId, 'dislike');

};

// Прокрутка к описанию на мобильных устройствах

useEffect(() => {

if (showDescription && descriptionRef.current) {

// Проверяем, что это мобильное устройство

const isMobile = window.innerWidth < 768;

if (isMobile) {

// Небольшая задержка для завершения анимации появления

setTimeout(() => {

descriptionRef.current?.scrollIntoView({

behavior: 'smooth',

block: 'start',

});

}, 350); // Немного больше, чем длительность анимации (300ms)

}

}

}, [showDescription]);

return (

<motion.div

className={`relative w-full max-w-[280px] md:max-w-sm mx-auto ${className}`}

drag="x"

dragConstraints={{ left: 0, right: 0 }}

dragElastic={0.2}

onDrag={handleDrag}

onDragEnd={handleDragEnd}

animate={{

x: dragDirection === 'left' ? -20 : dragDirection === 'right' ? 20 : 0,

rotate: dragDirection === 'left' ? -5 : dragDirection === 'right' ? 5 : 0,

}}

transition={{ type: "spring", stiffness: 300, damping: 30 }}

>

{/\* Карточка фильма \*/}

<div className="relative aspect-[2/3] bg-gray-800 rounded-2xl overflow-hidden shadow-2xl max-h-[60vh] md:max-h-none">

<div className="relative w-full h-full">

<img

src={filmUtils.getPosterUrl(film)}

alt={film.nameRu}

className="w-full h-full object-cover"

onError={(e) => {

const target = e.target as HTMLImageElement;

target.src = '/placeholder-poster.jpg';

}}

/>

{/\* Рейтинг \*/}

{film.ratingKinopoisk && (

<div className="absolute top-4 right-4 bg-black bg-opacity-70 text-white px-3 py-2 rounded-xl flex items-center gap-2 text-lg font-bold">

<Star className="h-5 w-5 fill-yellow-400 text-yellow-400" />

{filmUtils.formatRating(film.ratingKinopoisk)}

</div>

)}

{/\* Индикаторы свайпа \*/}

<div className="absolute inset-0 pointer-events-none">

{/\* Лайк индикатор \*/}

<motion.div

className="absolute top-1/2 left-8 transform -translate-y-1/2"

animate={{

opacity: dragDirection === 'right' ? 1 : 0,

scale: dragDirection === 'right' ? 1.2 : 1,

}}

transition={{ duration: 0.2 }}

>

<div className="bg-green-500 text-white px-6 py-3 rounded-2xl text-2xl font-bold flex items-center gap-3 shadow-lg">

<Heart className="h-8 w-8 fill-white" />

ЛАЙК!

</div>

</motion.div>

{/\* Дизлайк индикатор \*/}

<motion.div

className="absolute top-1/2 right-8 transform -translate-y-1/2"

animate={{

opacity: dragDirection === 'left' ? 1 : 0,

scale: dragDirection === 'left' ? 1.2 : 1,

}}

transition={{ duration: 0.2 }}

>

<div className="bg-red-500 text-white px-6 py-3 rounded-2xl text-2xl font-bold flex items-center gap-3 shadow-lg">

<X className="h-8 w-8" />

ПАСС!

</div>

</motion.div>

</div>

</div>

</div>

{/\* Кнопки действий \*/}

<div className="flex justify-center gap-6 mt-8">

<motion.button

onClick={handleDislike}

className="w-16 h-16 bg-red-500 hover:bg-red-600 text-white rounded-full flex items-center justify-center shadow-lg transition-colors"

whileHover={{ scale: 1.1 }}

whileTap={{ scale: 0.9 }}

>

<X className="h-8 w-8" />

</motion.button>

<motion.button

onClick={() => setShowDescription(!showDescription)}

className="w-16 h-16 bg-gray-500 hover:bg-gray-600 text-white rounded-full flex items-center justify-center shadow-lg transition-colors"

whileHover={{ scale: 1.1 }}

whileTap={{ scale: 0.9 }}

title="Информация о фильме"

>

<Info className="h-6 w-6" />

</motion.button>

<motion.button

onClick={handleLike}

className="w-16 h-16 bg-green-500 hover:bg-green-600 text-white rounded-full flex items-center justify-center shadow-lg transition-colors"

whileHover={{ scale: 1.1 }}

whileTap={{ scale: 0.9 }}

>

<Heart className="h-8 w-8" />

</motion.button>

</div>

{/\* Описание фильма \*/}

{showDescription && (

<motion.div

ref={descriptionRef}

initial={{ opacity: 0, height: 0 }}

animate={{ opacity: 1, height: 'auto' }}

exit={{ opacity: 0, height: 0 }}

transition={{ duration: 0.3 }}

className="mt-4 bg-gray-800 rounded-xl p-4"

>

<h3 className="font-bold text-white text-xl mb-2">

{film.nameRu}

</h3>

{film.nameEn && film.nameEn !== film.nameRu && (

<p className="text-gray-400 text-sm mb-3">

{film.nameEn}

</p>

)}

{/\* Описание \*/}

<p className="text-gray-200 text-sm md:text-base leading-relaxed mb-4">

{film.description || 'Описание не доступно'}

</p>

{/\* Дополнительная информация \*/}

<div className="space-y-2 text-sm">

<div className="flex items-center gap-3 text-gray-400">

<Globe className="h-4 w-4" />

<span>{film.year}</span>

{film.duration && (

<>

<span>•</span>

<Clock className="h-4 w-4" />

<span>{filmUtils.formatDuration(film.duration)}</span>

</>

)}

</div>

{film.genres && film.genres.length > 0 && (

<div className="text-gray-400">

<span className="font-medium">Жанры:</span> {filmUtils.formatGenres(film.genres)}

</div>

)}

{film.countries && film.countries.length > 0 && (

<div className="text-gray-400">

<span className="font-medium">Страны:</span> {filmUtils.formatCountries(film.countries)}

</div>

)}

</div>

</motion.div>

)}

</motion.div>

);

}

Файл VotingPageClient.tsx:

'use client';

import React, { useState, useEffect } from 'react';

import { motion, AnimatePresence } from 'framer-motion';

import { ArrowLeft, Users, AlertCircle } from 'lucide-react';

import { Button } from '@/components/ui/button';

import VotingCard from '@/components/VotingCard';

import { KinopoiskFilm } from '@/services/kinopoisk';

import { Vote } from '@/lib/groups';

import { useGroupSocket } from '@/hooks/useSocket';

import WaitingForOthers from '@/components/WaitingForOthers';

interface VotingPageClientProps {

groupCode: string;

}

export default function VotingPageClient({ groupCode }: VotingPageClientProps) {

const [group, setGroup] = useState<{ participants: string[]; films: KinopoiskFilm[] } | null>(null);

const [films, setFilms] = useState<KinopoiskFilm[]>([]);

const [currentFilmIndex, setCurrentFilmIndex] = useState(0);

const [votes, setVotes] = useState<Vote[]>([]);

const [isLoading, setIsLoading] = useState(true);

const [error, setError] = useState<string | null>(null);

const [isVotingComplete, setIsVotingComplete] = useState(false);

const [votesSaved, setVotesSaved] = useState(false);

const [completedParticipants, setCompletedParticipants] = useState<string[]>([]);

// Получаем никнейм из URL или sessionStorage

const participantName = typeof window !== 'undefined'

? (() => {

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

const nicknameFromUrl = urlParams.get('nickname');

if (nicknameFromUrl) {

// Сохраняем никнейм в sessionStorage для последующего использования

sessionStorage.setItem(`nickname\_${groupCode}`, nicknameFromUrl);

return nicknameFromUrl;

}

// Пробуем получить из sessionStorage

const nicknameFromStorage = sessionStorage.getItem(`nickname\_${groupCode}`);

return nicknameFromStorage || 'Участник';

})()

: 'Участник';

// Подключаемся к WebSocket для отслеживания завершения голосования

const { completeVoting, isConnected, participants, completedParticipants: socketCompletedParticipants } = useGroupSocket(groupCode, participantName);

// Загрузка данных группы

useEffect(() => {

const loadGroupData = async () => {

try {

setIsLoading(true);

// Нормализуем код группы

const normalizedCode = groupCode?.toUpperCase().trim();

const response = await fetch(`/api/groups-firebase?code=${encodeURIComponent(normalizedCode)}`);

const data = await response.json();

if (data.success) {

setGroup(data.data);

// Преобразуем Film в KinopoiskFilm для голосования

const filmsData = data.data.films || [];

console.log('Загружено фильмов:', filmsData.length, filmsData);

const kinopoiskFilms = filmsData.map((film: { id?: string; kinopoiskId: number; title: string; year: number; poster?: string; description?: string; rating?: number }) => ({

kinopoiskId: film.kinopoiskId,

nameRu: film.title,

nameEn: film.title,

year: film.year,

posterUrl: film.poster,

posterUrlPreview: film.poster,

description: film.description,

ratingKinopoisk: film.rating,

ratingImdb: film.rating,

genres: [],

countries: [],

duration: 0

}));

setFilms(kinopoiskFilms);

} else {

setError(data.error || 'Группа не найдена');

}

} catch (err) {

setError('Ошибка загрузки группы');

console.error('Ошибка загрузки группы:', err);

} finally {

setIsLoading(false);

}

};

loadGroupData();

}, [groupCode]);

// Автоматическое сохранение голосов при завершении

useEffect(() => {

if (isVotingComplete && !votesSaved && votes.length > 0) {

saveVotesAutomatically();

}

// eslint-disable-next-line react-hooks/exhaustive-deps

}, [isVotingComplete, votesSaved, votes.length]);

// Обновляем список завершивших из WebSocket

useEffect(() => {

if (socketCompletedParticipants.length > 0) {

// Объединяем с текущим пользователем, если его там нет

const allCompleted = Array.from(new Set([participantName, ...socketCompletedParticipants]));

setCompletedParticipants(allCompleted);

} else if (votesSaved) {

// Если голоса сохранены, добавляем текущего пользователя

setCompletedParticipants([participantName]);

}

}, [socketCompletedParticipants, votesSaved, participantName]);

// Автоматическое сохранение голосов

const saveVotesAutomatically = async () => {

if (votesSaved) return; // Предотвращаем повторное сохранение

try {

setVotesSaved(true);

const normalizedGroupCode = groupCode?.toUpperCase().trim() || groupCode;

console.log('Auto-saving votes:', {

groupCode: normalizedGroupCode,

participantName,

votesCount: votes.length

});

// Сохраняем голоса через API

const response = await fetch(`/api/groups-firebase/${normalizedGroupCode}/votes`, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

votes: votes.map(v => ({

filmId: v.filmId,

vote: v.vote,

participantId: participantName

}))

}),

});

if (!response.ok) {

const errorData = await response.json().catch(() => ({ error: 'Ошибка сервера' }));

console.error('HTTP error saving votes:', response.status, errorData);

setError(errorData.error || 'Ошибка сохранения голосов');

setVotesSaved(false); // Разрешаем повторную попытку

return;

}

const data = await response.json();

if (data.success) {

console.log('Votes saved successfully');

// Отправляем событие о завершении голосования через WebSocket

if (isConnected && completeVoting) {

completeVoting();

}

} else {

console.error('Error saving votes:', data.error);

setError(data.error || 'Ошибка сохранения голосов');

setVotesSaved(false);

}

} catch (err) {

console.error('Exception saving votes:', err);

setError('Ошибка сохранения голосов: ' + (err instanceof Error ? err.message : 'Неизвестная ошибка'));

setVotesSaved(false);

}

};

// Обработка голоса

const handleVote = (filmId: number, vote: 'like' | 'dislike') => {

const newVote: Vote = {

filmId,

vote,

timestamp: Date.now(),

participantId: participantName // Используем никнейм из URL

};

setVotes(prev => [...prev, newVote]);

// Переход к следующему фильму

if (currentFilmIndex < films.length - 1) {

setCurrentFilmIndex(prev => prev + 1);

} else {

// Голосование завершено - автоматически сохраним

setIsVotingComplete(true);

}

};

// Пропуск фильма (добавляем как dislike)

const handleSkip = () => {

if (currentFilmIndex < films.length) {

const currentFilm = films[currentFilmIndex];

if (currentFilm) {

// При пропуске считаем как dislike

handleVote(currentFilm.kinopoiskId, 'dislike');

}

}

};

// Возврат к предыдущему фильму

const handlePrevious = () => {

if (currentFilmIndex > 0) {

setCurrentFilmIndex(prev => prev - 1);

}

};

if (isLoading) {

return (

<div className="min-h-screen bg-gradient-to-b from-black via-gray-900 to-gray-800 flex items-center justify-center">

<div className="text-center">

<div className="animate-spin rounded-full h-12 w-12 border-b-2 border-pink-500 mx-auto mb-4"></div>

<p className="text-white text-lg">Загрузка голосования...</p>

</div>

</div>

);

}

if (error) {

return (

<div className="min-h-screen bg-gradient-to-b from-black via-gray-900 to-gray-800 flex items-center justify-center">

<div className="text-center max-w-md mx-auto px-4">

<AlertCircle className="h-16 w-16 text-red-500 mx-auto mb-4" />

<h1 className="text-2xl font-bold text-white mb-4">Ошибка</h1>

<p className="text-gray-300 mb-6">{error}</p>

<Button

onClick={() => window.location.href = '/'}

className="bg-pink-600 hover:bg-pink-700"

>

<ArrowLeft className="h-4 w-4 mr-2" />

На главную

</Button>

</div>

</div>

);

}

// Показываем ошибку только после завершения загрузки

if (!isLoading && (!group || films.length === 0)) {

return (

<div className="min-h-screen bg-gradient-to-b from-black via-gray-900 to-gray-800 flex items-center justify-center">

<div className="text-center max-w-md mx-auto px-4">

<Users className="h-16 w-16 text-gray-500 mx-auto mb-4" />

<h1 className="text-2xl font-bold text-white mb-4">Нет фильмов для голосования</h1>

<p className="text-gray-300 mb-6">

{error

? `Ошибка: ${error}`

: 'В группе пока нет добавленных фильмов. Попросите участников добавить фильмы.'}

</p>

<Button

onClick={() => {

const nickname = sessionStorage.getItem(`nickname\_${groupCode}`) || participantName;

window.location.href = `/group/${groupCode}?nickname=${encodeURIComponent(nickname)}`;

}}

className="bg-pink-600 hover:bg-pink-700"

>

<ArrowLeft className="h-4 w-4 mr-2" />

Вернуться к группе

</Button>

</div>

</div>

);

}

if (isVotingComplete) {

// Показываем красивый экран ожидания вместо кнопки сохранения

return (

<WaitingForOthers

completedCount={completedParticipants.length || 1} // Начинаем с 1, так как текущий пользователь завершил

totalCount={participants.length || group?.participants?.length || 1}

/>

);

}

const currentFilm = films[currentFilmIndex];

const progress = ((currentFilmIndex + 1) / films.length) \* 100;

return (

<div

className="h-screen overflow-hidden bg-gradient-to-b from-black via-gray-900 to-gray-800 text-white flex flex-col"

style={{

overscrollBehavior: 'none',

WebkitOverscrollBehavior: 'none'

}}

>

{/\* Заголовок - фиксированный \*/}

<div className="container mx-auto px-4 py-6 flex-shrink-0">

<div className="flex items-center justify-between mb-6">

<Button

onClick={() => {

const nickname = sessionStorage.getItem(`nickname\_${groupCode}`) || participantName;

window.location.href = `/group/${groupCode}?nickname=${encodeURIComponent(nickname)}`;

}}

variant="outline"

className="text-gray-300 border-gray-600 hover:bg-gray-700"

>

<ArrowLeft className="h-4 w-4 mr-2" />

Назад к группе

</Button>

<div className="text-center">

<h1 className="text-2xl font-bold">Голосование</h1>

<p className="text-gray-400">Группа: {groupCode}</p>

</div>

<div className="text-right">

<p className="text-sm text-gray-400">

{currentFilmIndex + 1} из {films.length}

</p>

<p className="text-xs text-gray-500">

{votes.length} голосов

</p>

</div>

</div>

{/\* Прогресс-бар \*/}

<div className="w-full bg-gray-700 rounded-full h-2 mb-4">

<motion.div

className="bg-gradient-to-r from-pink-500 to-purple-500 h-2 rounded-full"

initial={{ width: 0 }}

animate={{ width: `${progress}%` }}

transition={{ duration: 0.5 }}

/>

</div>

</div>

{/\* Карточка фильма и навигация - скроллируемая область \*/}

<div className="flex-1 overflow-y-auto">

<div className="container mx-auto px-4 py-6">

<div className="flex justify-center">

<AnimatePresence mode="wait">

<motion.div

key={currentFilmIndex}

initial={{ opacity: 0, scale: 0.8 }}

animate={{ opacity: 1, scale: 1 }}

exit={{ opacity: 0, scale: 0.8 }}

transition={{ duration: 0.3 }}

>

<VotingCard

film={currentFilm}

onVote={handleVote}

onSkip={handleSkip}

/>

</motion.div>

</AnimatePresence>

</div>

{/\* Навигация \*/}

<div className="flex justify-center gap-4 mt-8">

<Button

onClick={handlePrevious}

disabled={currentFilmIndex === 0}

variant="outline"

className="text-gray-300 border-gray-600 hover:bg-gray-700 disabled:opacity-50"

>

Предыдущий

</Button>

<div className="text-center">

<p className="text-sm text-gray-400">

Свайпните влево (👎) или вправо (👍)

</p>

<p className="text-xs text-gray-500 mt-1">

Или используйте кнопки внизу

</p>

</div>

</div>

</div>

</div>

</div>

);

}

Файл WaitingForOthers.tsx:

'use client';

import React from 'react';

import { motion } from 'framer-motion';

import { Users } from 'lucide-react';

interface WaitingForOthersProps {

  completedCount: number;

  totalCount: number;

}

export default function WaitingForOthers({ completedCount, totalCount }: WaitingForOthersProps) {

  const progress = totalCount > 0 ? (completedCount / totalCount) \* 100 : 0;

  // Анимация пульсации для иконок

  const pulseVariants = {

    animate: {

      scale: [1, 1.2, 1],

      opacity: [0.7, 1, 0.7],

    },

    transition: {

      duration: 2,

      repeat: Infinity,

      ease: "easeInOut"

    }

  };

  // Анимация для счетчика

  const countVariants = {

    initial: { scale: 0 },

    animate: { scale: 1 },

    transition: { type: "spring", stiffness: 200, damping: 15 }

  };

  return (

    <div className="min-h-screen bg-gradient-to-b from-black via-gray-900 to-gray-800 flex items-center justify-center px-4">

      <div className="max-w-md w-full text-center">

        {/\* Главная иконка с пульсацией \*/}

        <motion.div

          className="flex justify-center mb-8"

          variants={pulseVariants}

          animate="animate"

        >

          <Users className="h-24 w-24 text-pink-500" />

        </motion.div>

        {/\* Заголовок \*/}

        <motion.h1

          initial={{ opacity: 0, y: -20 }}

          animate={{ opacity: 1, y: 0 }}

          transition={{ delay: 0.2 }}

          className="text-3xl font-bold text-white mb-4"

        >

          Отличная работа! 🎉

        </motion.h1>

        {/\* Сообщение \*/}

        <motion.p

          initial={{ opacity: 0, y: 20 }}

          animate={{ opacity: 1, y: 0 }}

          transition={{ delay: 0.3 }}

          className="text-gray-300 text-lg mb-8"

        >

          Вы завершили голосование! Ожидаем других участников...

        </motion.p>

        {/\* Прогресс \*/}

        <motion.div

          initial={{ opacity: 0, y: 20 }}

          animate={{ opacity: 1, y: 0 }}

          transition={{ delay: 0.4 }}

          className="mb-8"

        >

          <div className="flex items-center justify-center gap-4 mb-4">

            <motion.div

              key={completedCount}

              variants={countVariants}

              initial="initial"

              animate="animate"

              className="text-4xl font-bold text-pink-400"

            >

              {completedCount}

            </motion.div>

            <span className="text-2xl text-gray-400">из</span>

            <span className="text-4xl font-bold text-gray-400">{totalCount}</span>

          </div>

          {/\* Прогресс-бар \*/}

          <div className="w-full bg-gray-700 rounded-full h-3 overflow-hidden">

            <motion.div

              className="h-full bg-gradient-to-r from-pink-500 via-purple-500 to-pink-500"

              initial={{ width: 0 }}

              animate={{ width: `${progress}%` }}

              transition={{ duration: 0.5, ease: "easeOut" }}

            />

          </div>

        </motion.div>

        {/\* Анимированные точки \*/}

        <motion.div

          initial={{ opacity: 0 }}

          animate={{ opacity: 1 }}

          transition={{ delay: 0.5 }}

          className="flex justify-center gap-2"

        >

          {[0, 1, 2].map((i) => (

            <motion.div

              key={i}

              className="w-3 h-3 bg-pink-500 rounded-full"

              animate={{

                y: [0, -10, 0],

                opacity: [0.5, 1, 0.5],

              }}

              transition={{

                duration: 1.5,

                repeat: Infinity,

                delay: i \* 0.2,

                ease: "easeInOut",

              }}

            />

          ))}

        </motion.div>

        {/\* Сообщение о следующем шаге \*/}

        <motion.p

          initial={{ opacity: 0 }}

          animate={{ opacity: 1 }}

          transition={{ delay: 0.6 }}

          className="text-gray-400 text-sm mt-8"

        >

          Как только все участники завершат голосование, вы автоматически увидите результаты!

        </motion.p>

      </div>

    </div>

  );

}

Файл button.tsx:

import React from 'react';

import { cn } from '@/lib/utils';

interface ButtonProps extends React.ButtonHTMLAttributes<HTMLButtonElement> {

variant?: 'default' | 'destructive' | 'outline' | 'secondary' | 'ghost' | 'link';

size?: 'default' | 'sm' | 'lg' | 'icon';

children: React.ReactNode;

}

const Button = React.forwardRef<HTMLButtonElement, ButtonProps>(

({ className, variant = 'default', size = 'default', ...props }, ref) => {

return (

<button

className={cn(

'inline-flex items-center justify-center whitespace-nowrap rounded-md text-sm font-medium ring-offset-background transition-colors focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2 disabled:pointer-events-none disabled:opacity-50',

{

'bg-primary text-primary-foreground hover:bg-primary/90': variant === 'default',

'bg-destructive text-destructive-foreground hover:bg-destructive/90': variant === 'destructive',

'border border-input bg-background hover:bg-accent hover:text-accent-foreground': variant === 'outline',

'bg-secondary text-secondary-foreground hover:bg-secondary/80': variant === 'secondary',

'hover:bg-accent hover:text-accent-foreground': variant === 'ghost',

'text-primary underline-offset-4 hover:underline': variant === 'link',

},

{

'h-10 px-4 py-2': size === 'default',

'h-9 rounded-md px-3': size === 'sm',

'h-11 rounded-md px-8': size === 'lg',

'h-10 w-10': size === 'icon',

},

className

)}

ref={ref}

{...props}

/>

);

}

);

Button.displayName = 'Button';

export { Button };

Файл NicknameContext.tsx:

'use client';

import React, { createContext, useContext, useState, useEffect } from 'react';

interface NicknameContextType {

nickname: string | null;

setNickname: (nickname: string) => void;

clearNickname: () => void;

}

const NicknameContext = createContext<NicknameContextType | undefined>(undefined);

export function NicknameProvider({ children }: { children: React.ReactNode }) {

const [nickname, setNicknameState] = useState<string | null>(null);

// Загружаем никнейм из localStorage при инициализации

useEffect(() => {

if (typeof window !== 'undefined') {

const stored = localStorage.getItem('flickpick-nickname');

if (stored) {

setNicknameState(stored);

}

}

}, []);

const setNickname = (newNickname: string) => {

setNicknameState(newNickname);

if (typeof window !== 'undefined') {

localStorage.setItem('flickpick-nickname', newNickname);

}

};

const clearNickname = () => {

setNicknameState(null);

if (typeof window !== 'undefined') {

localStorage.removeItem('flickpick-nickname');

}

};

return (

<NicknameContext.Provider value={{ nickname, setNickname, clearNickname }}>

{children}

</NicknameContext.Provider>

);

}

export function useNickname() {

const context = useContext(NicknameContext);

if (context === undefined) {

throw new Error('useNickname must be used within a NicknameProvider');

}

return context;

}

Файл useSocket.ts:

/\* eslint-disable @typescript-eslint/no-explicit-any \*/

import { useEffect, useState } from 'react';

import { io, Socket } from 'socket.io-client';

import { ServerToClientEvents, ClientToServerEvents } from '@/lib/socket';

type SocketType = Socket<ServerToClientEvents, ClientToServerEvents>;

export function useSocket() {

const [socket, setSocket] = useState<SocketType | null>(null);

const [isConnected, setIsConnected] = useState(false);

useEffect(() => {

let socketInstance: SocketType | null = null;

let isMounted = true;

// Функция инициализации Socket.IO

const initializeSocket = async () => {

try {

// Сначала делаем HTTP-запрос для инициализации Socket.IO сервера

const socketUrl = typeof window !== 'undefined'

? window.location.origin

: (process.env.NEXT\_PUBLIC\_SOCKET\_URL || 'http://localhost:3000');

// Принудительно инициализируем Socket.IO сервер через HTTP-запрос

await fetch(`${socketUrl}/api/socket`, {

method: 'GET',

}).catch(() => {

// Игнорируем ошибки, так как это просто триггер инициализации

console.log('Socket server initialization trigger sent');

});

// Небольшая задержка для инициализации сервера

await new Promise(resolve => setTimeout(resolve, 100));

if (!isMounted) return;

// Создаем Socket.IO подключение

socketInstance = io(socketUrl, {

path: '/api/socket',

transports: ['websocket', 'polling'],

reconnection: true,

reconnectionDelay: 1000,

reconnectionAttempts: 5,

timeout: 10000,

});

socketInstance.on('connect', () => {

console.log('✅ Socket connected:', socketInstance?.id);

if (isMounted) {

setIsConnected(true);

}

});

socketInstance.on('disconnect', (reason) => {

console.log('❌ Socket disconnected:', reason);

if (isMounted) {

setIsConnected(false);

}

});

socketInstance.on('connect\_error', (error) => {

console.error('⚠️ Socket connection error:', error.message);

if (isMounted) {

setIsConnected(false);

}

});

if (isMounted) {

setSocket(socketInstance);

}

} catch (error) {

console.error('Error initializing socket:', error);

}

};

initializeSocket();

return () => {

isMounted = false;

if (socketInstance) {

socketInstance.close();

socketInstance = null;

}

};

}, []);

return { socket, isConnected };

}

export function useGroupSocket(groupCode: string, participantName: string) {

const { socket, isConnected } = useSocket();

const [participants, setParticipants] = useState<string[]>([]);

const [films, setFilms] = useState<any[]>([]);

const [completedParticipants, setCompletedParticipants] = useState<string[]>([]);

useEffect(() => {

if (!socket || !isConnected) return;

// Подключаемся к группе

socket.emit('group:join', { groupCode, participantName });

// Слушаем события группы

socket.on('group:participant-joined', (data) => {

setParticipants(data.participants);

});

socket.on('group:participant-left', (data) => {

setParticipants(data.participants);

});

socket.on('group:film-added', (data) => {

setFilms(data.films);

});

socket.on('group:film-removed', (data) => {

setFilms(data.films);

});

socket.on('voting:started', () => {

console.log('Voting started, redirecting to voting page');

// Автоматически перенаправляем на страницу голосования

window.location.href = `/vote/${groupCode}?nickname=${encodeURIComponent(participantName)}`;

});

socket.on('voting:vote-cast', (data) => {

console.log(`${data.participant} voted ${data.vote} for film ${data.filmId}`);

});

socket.on('voting:completed', (data: { participant: string }) => {

console.log('Participant completed voting:', data.participant);

setCompletedParticipants(prev => {

if (!prev.includes(data.participant)) {

return [...prev, data.participant];

}

return prev;

});

});

socket.on('voting:all-completed', () => {

console.log('All participants completed voting, redirecting to results');

// Получаем никнейм из sessionStorage для передачи в результаты

const nickname = typeof window !== 'undefined'

? sessionStorage.getItem(`nickname\_${groupCode}`) || participantName

: participantName;

// Перенаправляем на страницу результатов с никнеймом

window.location.href = `/results/${groupCode}?nickname=${encodeURIComponent(nickname)}`;

});

socket.on('notification:error', (data) => {

console.error('Socket error:', data.message);

});

socket.on('group:closed', (data) => {

console.log('Group closed:', data.message);

// Перенаправляем пользователя на главную страницу, если группа закрыта

if (typeof window !== 'undefined') {

alert('Группа была закрыта создателем');

window.location.href = '/';

}

});

socket.on('group:reset', () => {

// Очищаем локально и возвращаемся в группу

setFilms([]);

const nickname = typeof window !== 'undefined'

? (sessionStorage.getItem(`nickname\_${groupCode}`) || participantName)

: participantName;

window.location.href = `/group/${groupCode}?nickname=${encodeURIComponent(nickname)}`;

});

socket.on('group:creator-changed', (data) => {

console.log('Creator changed:', data.message);

// Обновляем страницу, чтобы отобразить нового создателя

if (typeof window !== 'undefined') {

const nickname = sessionStorage.getItem(`nickname\_${groupCode}`) || participantName;

// Перезагружаем страницу для обновления данных группы

window.location.reload();

}

});

// Очистка при размонтировании

return () => {

socket.emit('group:leave', { groupCode, participantName });

socket.off('group:participant-joined');

socket.off('group:participant-left');

socket.off('group:film-added');

socket.off('group:film-removed');

socket.off('voting:started');

socket.off('voting:vote-cast');

socket.off('voting:completed');

socket.off('voting:all-completed');

socket.off('notification:error');

socket.off('group:closed');

socket.off('group:reset');

};

}, [socket, isConnected, groupCode, participantName]);

const addFilm = (film: any) => {

if (socket) {

socket.emit('film:add', { groupCode, film });

}

};

const castVote = (filmId: number, vote: 'like' | 'dislike') => {

if (socket) {

socket.emit('voting:vote', { groupCode, filmId, vote });

}

};

const startVoting = (filmsToVote: any[]) => {

if (socket) {

socket.emit('voting:start', { groupCode, films: filmsToVote });

}

};

const completeVoting = () => {

if (socket) {

socket.emit('voting:completed', { groupCode, participantName });

}

};

const resetGroup = () => {

if (socket) {

socket.emit('group:reset', { groupCode });

}

};

return {

socket,

isConnected,

participants,

films,

completedParticipants,

addFilm,

castVote,

startVoting,

completeVoting

, resetGroup

};

}

Файл firebase.ts:

import { initializeApp } from 'firebase/app';

import { getFirestore } from 'firebase/firestore';

// Конфигурация Firebase (в реальном проекте эти данные должны быть в .env)

const firebaseConfig = {

    apiKey: "AIzaSyCU\_u9jfTn6XHwnDshftOy7zB7--Q9JaL8",

    authDomain: "flickpick-5f4ab.firebaseapp.com",

    projectId: "flickpick-5f4ab",

    storageBucket: "flickpick-5f4ab.firebasestorage.app",

    messagingSenderId: "564529143651",

    appId: "1:564529143651:web:449588b431681880493d56",

    measurementId: "G-NZXQS3GRPR"

  };

// Инициализация Firebase

const app = initializeApp(firebaseConfig);

// Инициализация Firestore

export const db = getFirestore(app);

export default app;

Файл page.tsx:

'use client';

import React, { useState } from "react";

import { motion } from "framer-motion";

import { Button } from "@/components/ui/button";

import { ArrowLeft } from "lucide-react";

import NicknameInput from "@/components/NicknameInput";

export default function JoinPage() {

const [code, setCode] = useState("");

const [isLoading, setIsLoading] = useState(false);

const [showNicknameInput, setShowNicknameInput] = useState(false);

const [participantNickname, setParticipantNickname] = useState("");

const handleNicknameSubmit = (newNickname: string) => {

setParticipantNickname(newNickname);

setShowNicknameInput(false);

// После ввода никнейма присоединяемся к группе

joinGroup(newNickname);

};

const joinGroup = async (nickname: string) => {

setIsLoading(true);

try {

const response = await fetch('/api/groups-firebase/join', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

code: code.toUpperCase(),

participantName: nickname,

}),

});

const data = await response.json();

if (data.success) {

// Перенаправляем на страницу группы с никнеймом в URL

window.location.href = `/group/${data.data.code}?nickname=${encodeURIComponent(nickname)}`;

} else {

alert(`Ошибка присоединения: ${data.error}`);

}

} catch (error) {

console.error('Ошибка присоединения к группе:', error);

alert('Произошла ошибка при присоединении к группе');

} finally {

setIsLoading(false);

}

};

const handleJoin = () => {

if (!code.trim()) return;

setShowNicknameInput(true);

};

if (showNicknameInput) {

return (

<NicknameInput

onNicknameSubmit={handleNicknameSubmit}

onBack={() => setShowNicknameInput(false)}

title="Введите ваш никнейм"

subtitle="Выберите имя, под которым вас будут видеть в группе"

/>

);

}

return (

<div className="flex flex-col items-center justify-center min-h-screen bg-gradient-to-b from-black via-gray-900 to-gray-800 text-white px-4">

<motion.div

initial={{ opacity: 0, y: -20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ duration: 0.6 }}

className="w-full max-w-md"

>

{/\* Header \*/}

<div className="flex items-center mb-8">

<Button

variant="ghost"

size="icon"

onClick={() => window.history.back()}

className="mr-4 text-gray-400 hover:text-white"

>

<ArrowLeft className="h-5 w-5" />

</Button>

<h1 className="text-2xl font-bold">Присоединиться к группе</h1>

</div>

{/\* Form \*/}

<motion.div

initial={{ opacity: 0, y: 30 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.2, duration: 0.6 }}

className="space-y-6"

>

<div>

<label htmlFor="code" className="block text-sm font-medium text-gray-300 mb-2">

Код группы

</label>

<input

id="code"

type="text"

value={code}

onChange={(e) => setCode(e.target.value.toUpperCase())}

placeholder="Введите 5-значный код"

maxLength={5}

className="w-full px-4 py-3 bg-gray-800 border border-gray-700 rounded-xl text-white placeholder-gray-400 focus:outline-none focus:ring-2 focus:ring-pink-500 focus:border-transparent text-center text-lg tracking-widest"

/>

<p className="text-xs text-gray-400 mt-2 text-center">

Код состоит из 5 символов (буквы и цифры)

</p>

</div>

<Button

onClick={handleJoin}

disabled={code.length !== 5 || isLoading}

className="w-full py-4 text-lg bg-pink-600 hover:bg-pink-700 disabled:bg-gray-600 disabled:cursor-not-allowed rounded-xl shadow-lg transition"

>

{isLoading ? "Подключение..." : "Присоединиться"}

</Button>

</motion.div>

{/\* Info \*/}

<motion.div

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

transition={{ delay: 0.4, duration: 0.6 }}

className="mt-8 text-center"

>

<p className="text-sm text-gray-400">

Получите код от создателя группы

</p>

</motion.div>

</motion.div>

</div>

);

}

Файл Home.tsx:

'use client';

import React from "react";

import { motion } from "framer-motion";

import { Button } from "@/components/ui/button";

export default function Home() {

return (

<div className="flex flex-col items-center justify-center min-h-screen bg-gradient-to-b from-black via-gray-900 to-gray-800 text-white px-4">

<motion.h1

initial={{ opacity: 0, y: -20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ duration: 0.6 }}

className="text-4xl font-bold mb-16 text-center"

>

🎬 FlickPick

<span className="block text-lg font-normal mt-2 text-gray-300">

Выбор фильма без споров

</span>

</motion.h1>

<motion.div

initial={{ opacity: 0, y: 30 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.3, duration: 0.6 }}

className="flex flex-col w-full max-w-xs gap-6"

>

<Button

className="w-full py-6 text-lg bg-pink-600 hover:bg-pink-700 rounded-2xl shadow-lg transition"

onClick={() => (window.location.href = "/join")}

>

🔗 Присоединиться к группе

</Button>

<Button

className="w-full py-6 text-lg bg-indigo-600 hover:bg-indigo-700 rounded-2xl shadow-lg transition"

onClick={() => (window.location.href = "/create")}

>

✨ Создать новую группу

</Button>

</motion.div>

</div>

);

}

Файл global.css:

@import "tailwindcss";

:root {

  --background: #ffffff;

  --foreground: #171717;

}

@theme inline {

  --color-background: var(--background);

  --color-foreground: var(--foreground);

  --font-sans: var(--font-geist-sans);

  --font-mono: var(--font-geist-mono);

}

@media (prefers-color-scheme: dark) {

  :root {

    --background: #0a0a0a;

    --foreground: #ededed;

  }

}

body {

  background: var(--background);

  color: var(--foreground);

  font-family: Arial, Helvetica, sans-serif;

}

/\* 3D Flip Card Styles \*/

.perspective-1000 {

  perspective: 1000px;

}

.preserve-3d {

  transform-style: preserve-3d;

}

.backface-hidden {

  backface-visibility: hidden;

}

.rotate-y-180 {

  transform: rotateY(180deg);

}

/\* Line clamp utilities \*/

.line-clamp-1 {

  overflow: hidden;

  display: -webkit-box;

  -webkit-box-orient: vertical;

  -webkit-line-clamp: 1;

}

.line-clamp-2 {

  overflow: hidden;

  display: -webkit-box;

  -webkit-box-orient: vertical;

  -webkit-line-clamp: 2;

}

.line-clamp-3 {

  overflow: hidden;

  display: -webkit-box;

  -webkit-box-orient: vertical;

  -webkit-line-clamp: 3;

}

## П.2. Сервер

Файл search/route.ts:

import { NextRequest, NextResponse } from 'next/server';

import { kinopoiskService } from '@/services/kinopoisk';

// GET /api/films/search?q=query&page=1 - поиск фильмов

export async function GET(request: NextRequest) {

try {

const { searchParams } = new URL(request.url);

const query = searchParams.get('q');

const page = parseInt(searchParams.get('page') || '1');

if (!query || query.trim().length === 0) {

return NextResponse.json(

{ error: 'Поисковый запрос обязателен' },

{ status: 400 }

);

}

if (query.trim().length < 2) {

return NextResponse.json(

{ error: 'Поисковый запрос должен содержать минимум 2 символа' },

{ status: 400 }

);

}

const results = await kinopoiskService.searchFilms(query.trim(), page);

// Загружаем детали для первых 5 фильмов параллельно

const filmsWithDetails = await Promise.allSettled(

results.items.slice(0, 5).map(async (film) => {

try {

const details = await kinopoiskService.getFilmDetails(film.kinopoiskId);

return { ...film, ...details };

} catch {

// Если не удалось загрузить детали, возвращаем исходный фильм

return film;

}

})

);

// Объединяем результаты

const enrichedItems = filmsWithDetails.map((result, index) => {

if (result.status === 'fulfilled') {

return result.value;

} else {

// Если детали не загрузились, используем исходный фильм

return results.items[index];

}

});

// Добавляем остальные фильмы без деталей

const remainingItems = results.items.slice(5);

return NextResponse.json({

success: true,

data: {

...results,

items: [...enrichedItems, ...remainingItems]

},

});

} catch (error) {

console.error('Ошибка поиска фильмов:', error);

// Если ошибка связана с API ключом

if (error instanceof Error && error.message.includes('API ключ')) {

return NextResponse.json(

{ error: 'API ключ Кинопоиска не настроен. Обратитесь к администратору.' },

{ status: 503 }

);

}

return NextResponse.json(

{ error: 'Произошла ошибка при поиске фильмов' },

{ status: 500 }

);

}

}

Файл details/route.ts:

import { NextRequest, NextResponse } from 'next/server';

import { kinopoiskService } from '@/services/kinopoisk';

// GET /api/films/details/[id] - получение деталей фильма

export async function GET(

request: NextRequest,

{ params }: { params: Promise<{ id: string }> }

) {

try {

const { id } = await params;

const kinopoiskId = parseInt(id);

if (isNaN(kinopoiskId) || kinopoiskId <= 0) {

return NextResponse.json(

{ error: 'Неверный ID фильма' },

{ status: 400 }

);

}

const filmDetails = await kinopoiskService.getFilmDetails(kinopoiskId);

return NextResponse.json({

success: true,

data: filmDetails,

});

} catch (error) {

console.error('Ошибка получения деталей фильма:', error);

// Если ошибка связана с API ключом

if (error instanceof Error && error.message.includes('API ключ')) {

return NextResponse.json(

{ error: 'API ключ Кинопоиска не настроен. Обратитесь к администратору.' },

{ status: 503 }

);

}

// Если фильм не найден

if (error instanceof Error && error.message.includes('404')) {

return NextResponse.json(

{ error: 'Фильм не найден' },

{ status: 404 }

);

}

return NextResponse.json(

{ error: 'Произошла ошибка при получении информации о фильме' },

{ status: 500 }

);

}

}

Файл groups/route.ts:

import { NextRequest, NextResponse } from 'next/server';

import { createGroup, getGroup } from '@/lib/groups';

// POST /api/groups - создание новой группы

export async function POST(request: NextRequest) {

try {

const body = await request.json();

const { creatorName } = body;

const result = createGroup(creatorName);

if (!result.success) {

return NextResponse.json(

{ error: result.error },

{ status: 400 }

);

}

return NextResponse.json({

success: true,

group: {

id: result.group!.id,

code: result.group!.code,

participants: result.group!.participants,

createdAt: result.group!.createdAt,

},

});

} catch (error) {

console.error('Ошибка создания группы:', error);

return NextResponse.json(

{ error: 'Внутренняя ошибка сервера' },

{ status: 500 }

);

}

}

// GET /api/groups?code=ABC123 - получение информации о группе

export async function GET(request: NextRequest) {

try {

const { searchParams } = new URL(request.url);

const code = searchParams.get('code');

if (!code) {

return NextResponse.json(

{ error: 'Код группы обязателен' },

{ status: 400 }

);

}

const group = getGroup(code);

if (!group) {

return NextResponse.json(

{ error: 'Группа не найдена' },

{ status: 404 }

);

}

return NextResponse.json({

success: true,

data: {

id: group.id,

code: group.code,

participants: group.participants,

films: group.films,

createdAt: group.createdAt,

isActive: group.isActive,

},

});

} catch (error) {

console.error('Ошибка получения группы:', error);

return NextResponse.json(

{ error: 'Внутренняя ошибка сервера' },

{ status: 500 }

);

}

}

Файл groups/join/route.ts:

import { NextRequest, NextResponse } from 'next/server';

import { joinGroup } from '@/lib/groups';

// POST /api/groups/join - присоединение к группе

export async function POST(request: NextRequest) {

try {

const body = await request.json();

const { code, participantName } = body;

const result = joinGroup(code, participantName);

if (!result.success) {

return NextResponse.json(

{ error: result.error },

{ status: result.error === 'Группа с таким кодом не найдена' ? 404 : 400 }

);

}

return NextResponse.json({

success: true,

message: 'Успешно присоединились к группе',

group: {

id: result.group!.id,

code: result.group!.code,

participants: result.group!.participants,

films: result.group!.films,

createdAt: result.group!.createdAt,

isActive: result.group!.isActive,

},

});

} catch (error) {

console.error('Ошибка присоединения к группе:', error);

return NextResponse.json(

{ error: 'Внутренняя ошибка сервера' },

{ status: 500 }

);

}

}

Файл groups-firebase/route.ts:

import { NextRequest, NextResponse } from 'next/server';

import { createGroup, getGroupByCode, getFilmsByGroup, cleanupOldGroups } from '@/lib/database';

// POST /api/groups-firebase - создание новой группы

export async function POST(request: NextRequest) {

try {

const body = await request.json();

const { creatorName } = body;

if (!creatorName || typeof creatorName !== 'string') {

return NextResponse.json(

{ error: 'Имя создателя группы обязательно' },

{ status: 400 }

);

}

// Генерируем уникальный код группы (уже в верхнем регистре)

const code = generateGroupCode().toUpperCase().trim();

const groupId = await createGroup({

code,

participants: [creatorName],

isActive: true,

createdBy: creatorName

});

return NextResponse.json({

success: true,

data: {

id: groupId,

code,

participants: [creatorName],

films: [],

votes: [],

createdAt: new Date(),

isActive: true

}

});

} catch (error) {

console.error('Ошибка создания группы:', error);

return NextResponse.json(

{ error: 'Внутренняя ошибка сервера' },

{ status: 500 }

);

}

}

// GET /api/groups-firebase?code=ABC123 - получение информации о группе

export async function GET(request: NextRequest) {

try {

const { searchParams } = new URL(request.url);

const code = searchParams.get('code');

// Автоочистка старых групп (выполняется периодически при запросах)

// В продакшене лучше делать через cron job, но для простоты делаем здесь

if (Math.random() < 0.1) { // 10% вероятность при каждом запросе

cleanupOldGroups().catch(err => console.error('Auto-cleanup error:', err));

}

if (!code) {

return NextResponse.json(

{ error: 'Код группы обязателен' },

{ status: 400 }

);

}

// Нормализуем код группы (верхний регистр, без пробелов)

const normalizedCode = code.toUpperCase().trim();

const group = await getGroupByCode(normalizedCode);

if (!group) {

return NextResponse.json(

{ error: 'Группа не найдена' },

{ status: 404 }

);

}

// Загружаем фильмы группы

const films = await getFilmsByGroup(group.id);

return NextResponse.json({

success: true,

data: {

...group,

films

}

});

} catch (error) {

console.error('Ошибка получения группы:', error);

return NextResponse.json(

{ error: 'Внутренняя ошибка сервера' },

{ status: 500 }

);

}

}

// Генерация уникального 5-значного кода

function generateGroupCode(): string {

const chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';

let result = '';

for (let i = 0; i < 5; i++) {

result += chars.charAt(Math.floor(Math.random() \* chars.length));

}

return result;

}

Файл groups-firebase/join/route.ts:

import { NextRequest, NextResponse } from 'next/server';

import { getGroupByCode, updateGroup } from '@/lib/database';

// POST /api/groups-firebase/join - присоединение к группе

export async function POST(request: NextRequest) {

  try {

    const body = await request.json();

    const { code, participantName } = body;

    if (!code || !participantName) {

      return NextResponse.json(

        { error: 'Код группы и имя участника обязательны' },

        { status: 400 }

      );

    }

    // Получаем группу

    const group = await getGroupByCode(code);

    if (!group) {

      return NextResponse.json(

        { error: 'Группа с таким кодом не найдена' },

        { status: 404 }

      );

    }

    // Проверяем, не является ли участник уже членом группы

    if (group.participants.includes(participantName)) {

      return NextResponse.json(

        { error: 'Вы уже являетесь участником этой группы' },

        { status: 400 }

      );

    }

    // Добавляем участника

    const updatedParticipants = [...group.participants, participantName];

    await updateGroup(group.id, { participants: updatedParticipants });

    return NextResponse.json({

      success: true,

      data: {

        ...group,

        participants: updatedParticipants

      }

    });

  } catch (error) {

    console.error('Ошибка присоединения к группе:', error);

    return NextResponse.json(

      { error: 'Внутренняя ошибка сервера' },

      { status: 500 }

    );

  }

}

Файл groups-firebase/[id]/films/route.ts:

import { NextRequest, NextResponse } from 'next/server';

import { getGroupByCode, addFilmToGroup, getFilmsByGroup } from '@/lib/database';

// POST /api/groups-firebase/[id]/films - добавление фильма в группу

export async function POST(

request: NextRequest,

{ params }: { params: Promise<{ id: string }> }

) {

try {

const { id: groupCode } = await params;

const body = await request.json();

const { film } = body;

if (!film) {

return NextResponse.json(

{ error: 'Фильм не указан' },

{ status: 400 }

);

}

// Получаем группу

const group = await getGroupByCode(groupCode);

if (!group) {

return NextResponse.json(

{ error: 'Группа не найдена' },

{ status: 404 }

);

}

// Добавляем фильм в группу

await addFilmToGroup({

groupId: group.id,

kinopoiskId: film.kinopoiskId,

title: film.nameRu,

year: film.year,

poster: film.posterUrl,

description: film.description,

rating: film.ratingKinopoisk,

addedBy: 'Пользователь' // TODO: Получать реальное имя

});

// Получаем обновленный список фильмов

const films = await getFilmsByGroup(group.id);

return NextResponse.json({

success: true,

data: {

...group,

films

}

});

} catch (error) {

console.error('Ошибка добавления фильма:', error);

// Более детальная обработка ошибок

if (error instanceof Error) {

if (error.message.includes('уже добавлен')) {

return NextResponse.json(

{ error: 'Этот фильм уже добавлен в группу' },

{ status: 400 }

);

}

if (error.message.includes('index')) {

return NextResponse.json(

{ error: 'Ошибка базы данных: требуется настройка индексов' },

{ status: 500 }

);

}

return NextResponse.json(

{ error: `Ошибка: ${error.message}` },

{ status: 500 }

);

}

return NextResponse.json(

{ error: 'Внутренняя ошибка сервера' },

{ status: 500 }

);

}

}

Файл groups-firebase/[id]/votes/route.ts:

import { NextRequest, NextResponse } from 'next/server';

import { getGroupByCode, addVotes, getVotesByGroup, deleteVotesByGroup } from '@/lib/database';

// POST /api/groups-firebase/[id]/votes - сохранение голосов

export async function POST(

request: NextRequest,

{ params }: { params: Promise<{ id: string }> }

) {

try {

const { id: groupCode } = await params;

const body = await request.json();

const { votes } = body;

// Нормализуем код группы (верхний регистр, без пробелов)

const normalizedCode = groupCode?.toUpperCase().trim();

console.log('Saving votes for group:', {

original: groupCode,

normalized: normalizedCode,

votesCount: votes?.length

});

console.log('Votes data:', votes);

if (!votes || !Array.isArray(votes)) {

console.error('Invalid votes format:', votes);

return NextResponse.json(

{ error: 'Неверный формат голосов' },

{ status: 400 }

);

}

if (!normalizedCode) {

console.error('Group code is missing or invalid');

return NextResponse.json(

{ error: 'Код группы не указан' },

{ status: 400 }

);

}

// Получаем группу

const group = await getGroupByCode(normalizedCode);

if (!group) {

console.error('Group not found:', normalizedCode);

// Пробуем найти все группы для отладки

console.error('Attempting to find group with code:', normalizedCode);

return NextResponse.json(

{ error: `Группа с кодом "${normalizedCode}" не найдена` },

{ status: 404 }

);

}

console.log('Group found:', group.id, group.code);

// Валидация голосов

const validVotes = votes.filter((vote: { filmId: number; vote: string }) =>

vote.filmId &&

typeof vote.filmId === 'number' &&

(vote.vote === 'like' || vote.vote === 'dislike')

);

if (validVotes.length === 0) {

return NextResponse.json(

{ error: 'Нет валидных голосов' },

{ status: 400 }

);

}

// Сохраняем голоса

await addVotes(validVotes.map(vote => ({

groupId: group.id,

filmId: vote.filmId,

vote: vote.vote,

participantId: vote.participantId

})));

return NextResponse.json({

success: true,

message: 'Голоса успешно сохранены'

});

} catch (error) {

console.error('Ошибка сохранения голосов:', error);

return NextResponse.json(

{ error: 'Внутренняя ошибка сервера' },

{ status: 500 }

);

}

}

// GET /api/groups-firebase/[id]/votes - получение голосов

export async function GET(

request: NextRequest,

{ params }: { params: Promise<{ id: string }> }

) {

try {

const { id: groupCode } = await params;

const group = await getGroupByCode(groupCode);

if (!group) {

return NextResponse.json(

{ error: 'Группа не найдена' },

{ status: 404 }

);

}

const votes = await getVotesByGroup(group.id);

return NextResponse.json({

success: true,

data: { votes }

});

} catch (error) {

console.error('Ошибка получения голосов:', error);

return NextResponse.json(

{ error: 'Внутренняя ошибка сервера' },

{ status: 500 }

);

}

}

// DELETE /api/groups-firebase/[id]/votes - удаление всех голосов группы

export async function DELETE(

request: NextRequest,

{ params }: { params: Promise<{ id: string }> }

) {

try {

const { id: groupCode } = await params;

const normalizedCode = groupCode?.toUpperCase().trim();

if (!normalizedCode) {

return NextResponse.json(

{ error: 'Код группы не указан' },

{ status: 400 }

);

}

const group = await getGroupByCode(normalizedCode);

if (!group) {

return NextResponse.json(

{ error: 'Группа не найдена' },

{ status: 404 }

);

}

await deleteVotesByGroup(group.id);

return NextResponse.json({

success: true,

message: 'Все голоса успешно удалены'

});

} catch (error) {

console.error('Ошибка удаления голосов:', error);

return NextResponse.json(

{ error: 'Внутренняя ошибка сервера' },

{ status: 500 }

);

}

}

Файл database.ts:

import {

collection,

doc,

addDoc,

getDocs,

updateDoc,

deleteDoc,

query,

where,

orderBy,

serverTimestamp

} from 'firebase/firestore';

import { db } from './firebase';

import { getAdminDb } from './firebaseAdmin';

import { Group, Vote } from './groups';

// Интерфейс для фильма из Firebase

export interface Film {

id: string;

kinopoiskId: number;

title: string;

year: number;

poster?: string;

description?: string;

rating?: number;

addedBy: string;

addedAt: Date;

}

// Коллекции Firestore

const GROUPS\_COLLECTION = 'groups';

const FILMS\_COLLECTION = 'films';

const VOTES\_COLLECTION = 'votes';

// ===== ГРУППЫ =====

export interface GroupData {

code: string;

participants: string[];

createdAt: unknown; // serverTimestamp

isActive: boolean;

createdBy: string;

}

export async function createGroup(groupData: Omit<GroupData, 'createdAt'>): Promise<string> {

try {

const docRef = await addDoc(collection(db, GROUPS\_COLLECTION), {

...groupData,

createdAt: serverTimestamp()

});

return docRef.id;

} catch (error) {

console.error('Ошибка создания группы:', error);

throw new Error('Не удалось создать группу');

}

}

export async function getGroupByCode(code: string): Promise<Group | null> {

try {

// Нормализуем код группы (верхний регистр, без пробелов)

const normalizedCode = code?.toUpperCase().trim();

if (!normalizedCode) {

console.error('Empty or invalid group code');

return null;

}

console.log('Searching for group with code:', normalizedCode);

const q = query(collection(db, GROUPS\_COLLECTION), where('code', '==', normalizedCode));

const querySnapshot = await getDocs(q);

if (querySnapshot.empty) {

return null;

}

const doc = querySnapshot.docs[0];

const data = doc.data();

return {

id: doc.id,

code: data.code,

participants: data.participants || [],

films: [], // Будем загружать отдельно

votes: [], // Будем загружать отдельно

createdAt: data.createdAt?.toDate() || new Date(),

isActive: data.isActive ?? true,

createdBy: data.createdBy || ''

};

} catch (error) {

console.error('Ошибка получения группы:', error);

throw new Error('Не удалось получить группу');

}

}

export async function updateGroup(groupId: string, updates: Partial<GroupData>): Promise<void> {

try {

const groupRef = doc(db, GROUPS\_COLLECTION, groupId);

await updateDoc(groupRef, updates);

} catch (error) {

console.error('Ошибка обновления группы:', error);

throw new Error('Не удалось обновить группу');

}

}

// ===== ФИЛЬМЫ =====

export interface FilmData {

groupId: string;

kinopoiskId: number;

title: string;

year: number;

poster?: string;

description?: string;

rating?: number;

addedBy: string;

addedAt: unknown; // serverTimestamp

}

export async function addFilmToGroup(filmData: Omit<FilmData, 'addedAt'>): Promise<string> {

try {

// Проверяем, не существует ли уже такой фильм в группе

const existingFilms = await getFilmsByGroup(filmData.groupId);

const isDuplicate = existingFilms.some(film =>

film.kinopoiskId === filmData.kinopoiskId

);

if (isDuplicate) {

throw new Error('Этот фильм уже добавлен в группу');

}

const docRef = await addDoc(collection(db, FILMS\_COLLECTION), {

...filmData,

addedAt: serverTimestamp()

});

return docRef.id;

} catch (error) {

console.error('Ошибка добавления фильма:', error);

if (error instanceof Error) {

throw error; // Передаем оригинальную ошибку

}

throw new Error('Не удалось добавить фильм');

}

}

export async function getFilmsByGroup(groupId: string): Promise<Film[]> {

try {

// Упрощенный запрос без orderBy для избежания необходимости в индексах

const q = query(

collection(db, FILMS\_COLLECTION),

where('groupId', '==', groupId)

);

const querySnapshot = await getDocs(q);

const films = querySnapshot.docs.map(doc => {

const data = doc.data();

return {

id: doc.id,

kinopoiskId: data.kinopoiskId,

title: data.title,

year: data.year,

poster: data.poster,

description: data.description,

rating: data.rating,

addedBy: data.addedBy,

addedAt: data.addedAt?.toDate() || new Date()

};

});

// Сортируем на клиенте

return films.sort((a, b) => b.addedAt.getTime() - a.addedAt.getTime());

} catch (error) {

console.error('Ошибка получения фильмов:', error);

throw new Error('Не удалось получить фильмы');

}

}

// ===== ГОЛОСА =====

export interface VoteData {

groupId: string;

filmId: number;

vote: 'like' | 'dislike';

participantId: string;

timestamp: unknown; // serverTimestamp

}

export async function addVotes(votes: Omit<VoteData, 'timestamp'>[]): Promise<void> {

try {

const batch = votes.map(vote =>

addDoc(collection(db, VOTES\_COLLECTION), {

...vote,

timestamp: serverTimestamp()

})

);

await Promise.all(batch);

} catch (error) {

console.error('Ошибка добавления голосов:', error);

throw new Error('Не удалось добавить голоса');

}

}

export async function getVotesByGroup(groupId: string): Promise<Vote[]> {

try {

const q = query(collection(db, VOTES\_COLLECTION), where('groupId', '==', groupId));

const querySnapshot = await getDocs(q);

return querySnapshot.docs.map(doc => {

const data = doc.data();

return {

filmId: data.filmId,

vote: data.vote,

timestamp: data.timestamp?.toDate() || new Date(),

participantId: data.participantId

};

});

} catch (error) {

console.error('Ошибка получения голосов:', error);

throw new Error('Не удалось получить голоса');

}

}

export async function deleteVotesByGroup(groupId: string): Promise<void> {

try {

let deletedCount = 0;

const admin = getAdminDb();

if (admin) {

const snap = await admin.collection(VOTES\_COLLECTION).where('groupId', '==', groupId).get();

deletedCount = snap.docs.length;

const batch = admin.batch();

snap.docs.forEach((d: any) => batch.delete(d.ref));

await batch.commit();

} else {

const q = query(collection(db, VOTES\_COLLECTION), where('groupId', '==', groupId));

const querySnapshot = await getDocs(q);

deletedCount = querySnapshot.docs.length;

const deletePromises = querySnapshot.docs.map(doc => deleteDoc(doc.ref));

await Promise.all(deletePromises);

}

console.log(`Удалено ${deletedCount} голосов для группы ${groupId}`);

} catch (error) {

console.error('Ошибка удаления голосов:', error);

throw new Error('Не удалось удалить голоса');

}

}

export async function deleteFilmsByGroup(groupId: string): Promise<void> {

try {

let deletedCount = 0;

const admin = getAdminDb();

if (admin) {

const snap = await admin.collection(FILMS\_COLLECTION).where('groupId', '==', groupId).get();

deletedCount = snap.docs.length;

const batch = admin.batch();

snap.docs.forEach((d: any) => batch.delete(d.ref));

await batch.commit();

} else {

const q = query(collection(db, FILMS\_COLLECTION), where('groupId', '==', groupId));

const querySnapshot = await getDocs(q);

deletedCount = querySnapshot.docs.length;

const deletePromises = querySnapshot.docs.map(doc => deleteDoc(doc.ref));

await Promise.all(deletePromises);

}

console.log(`Удалено ${deletedCount} фильмов для группы ${groupId}`);

} catch (error) {

console.error('Ошибка удаления фильмов:', error);

throw new Error('Не удалось удалить фильмы');

}

}

export async function deleteGroup(groupId: string): Promise<void> {

try {

// Сначала удаляем все связанные данные (фильмы и голоса)

await deleteFilmsByGroup(groupId);

await deleteVotesByGroup(groupId);

// Затем удаляем саму группу

const admin = getAdminDb();

if (admin) {

await admin.collection(GROUPS\_COLLECTION).doc(groupId).delete();

} else {

const groupRef = doc(db, GROUPS\_COLLECTION, groupId);

await deleteDoc(groupRef);

}

console.log(`Группа ${groupId} и все связанные данные успешно удалены`);

} catch (error) {

console.error('Ошибка удаления группы:', error);

throw new Error('Не удалось удалить группу');

}

}

// Автоматическая очистка заброшенных групп (старше 24 часов)

export async function cleanupOldGroups(): Promise<number> {

try {

const now = new Date();

const dayAgo = new Date(now.getTime() - 24 \* 60 \* 60 \* 1000); // 24 часа назад

const q = query(collection(db, GROUPS\_COLLECTION));

const querySnapshot = await getDocs(q);

let deletedCount = 0;

const deletePromises: Promise<void>[] = [];

querySnapshot.docs.forEach(docSnap => {

const data = docSnap.data();

const createdAt = data.createdAt?.toDate();

if (createdAt && createdAt < dayAgo) {

deletePromises.push(deleteGroup(docSnap.id));

deletedCount++;

}

});

await Promise.all(deletePromises);

if (deletedCount > 0) {

console.log(`Автоочистка: удалено ${deletedCount} старых групп`);

}

return deletedCount;

} catch (error) {

console.error('Ошибка автоочистки групп:', error);

return 0;

}

}

// ===== УТИЛИТЫ =====

export async function getGroupWithData(code: string): Promise<Group | null> {

try {

const group = await getGroupByCode(code);

if (!group) {

return null;

}

// Загружаем фильмы и голоса

const [films, votes] = await Promise.all([

getFilmsByGroup(group.id),

getVotesByGroup(group.id)

]);

return {

...group,

films,

votes

};

} catch (error) {

console.error('Ошибка получения полных данных группы:', error);

throw new Error('Не удалось получить данные группы');

}

}

Файл firebaseAdmin.ts:

/\* eslint-disable @typescript-eslint/no-explicit-any \*/

let adminApp: any | null = null;

let adminDb: any | null = null;

export function getAdminDb() {

try {

if (adminDb) return adminDb;

// Динамически подключаем firebase-admin, чтобы не требовать зависимость в браузере

// и не падать, если пакет не установлен

// eslint-disable-next-line @typescript-eslint/no-var-requires

const admin = require('firebase-admin');

if (!admin.apps || admin.apps.length === 0) {

const projectId = process.env.FIREBASE\_PROJECT\_ID;

const clientEmail = process.env.FIREBASE\_CLIENT\_EMAIL;

const privateKey = (process.env.FIREBASE\_PRIVATE\_KEY || '').replace(/\\n/g, '\n');

if (!projectId || !clientEmail || !privateKey) {

console.warn('[firebase-admin] Не заданы переменные окружения FIREBASE\_PROJECT\_ID / FIREBASE\_CLIENT\_EMAIL / FIREBASE\_PRIVATE\_KEY');

return null;

}

adminApp = admin.initializeApp({

credential: admin.credential.cert({

projectId,

clientEmail,

privateKey,

}),

});

} else {

adminApp = admin.app();

}

adminDb = admin.firestore(adminApp);

return adminDb;

} catch (e) {

console.warn('[firebase-admin] Не удалось инициализировать admin SDK:', (e as Error).message);

return null;

}

}

export function hasAdminCredentials(): boolean {

return Boolean(process.env.FIREBASE\_PROJECT\_ID && process.env.FIREBASE\_CLIENT\_EMAIL && process.env.FIREBASE\_PRIVATE\_KEY);

}

Файл kinopoisk.ts:

// Типы для работы с Кинопоиск API

export interface KinopoiskFilm {

kinopoiskId: number;

nameRu: string;

nameEn?: string;

year: number;

posterUrl: string;

posterUrlPreview: string;

description?: string;

ratingKinopoisk?: number;

ratingImdb?: number;

genres: Array<{ genre: string }>;

countries: Array<{ country: string }>;

duration?: number;

}

export interface KinopoiskSearchResponse {

total: number;

totalPages: number;

items: KinopoiskFilm[];

}

export interface KinopoiskFilmDetails extends KinopoiskFilm {

shortDescription?: string;

slogan?: string;

type: string;

ratingAgeLimits?: string;

distributors?: string;

premiere?: {

world: string;

russia?: string;

};

}

// Конфигурация API

const KINOPOISK\_API\_BASE\_URL = 'https://kinopoiskapiunofficial.tech/api/v2.2';

const KINOPOISK\_API\_KEY = process.env.KINOPOISK\_API\_KEY || '';

// Сервис для работы с Кинопоиск API

export class KinopoiskService {

private static instance: KinopoiskService;

private apiKey: string;

private constructor() {

this.apiKey = KINOPOISK\_API\_KEY;

}

public static getInstance(): KinopoiskService {

if (!KinopoiskService.instance) {

KinopoiskService.instance = new KinopoiskService();

}

return KinopoiskService.instance;

}

// Поиск фильмов по названию

async searchFilms(query: string, page: number = 1): Promise<KinopoiskSearchResponse> {

if (!this.apiKey) {

throw new Error('API ключ Кинопоиска не настроен');

}

try {

const response = await fetch(

`${KINOPOISK\_API\_BASE\_URL}/films?keyword=${encodeURIComponent(query)}&page=${page}`,

{

headers: {

'X-API-KEY': this.apiKey,

'Content-Type': 'application/json',

},

}

);

if (!response.ok) {

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

}

const data = await response.json();

return data;

} catch (error) {

console.error('Ошибка поиска фильмов:', error);

throw new Error('Не удалось найти фильмы');

}

}

// Получение детальной информации о фильме

async getFilmDetails(kinopoiskId: number): Promise<KinopoiskFilmDetails> {

if (!this.apiKey) {

throw new Error('API ключ Кинопоиска не настроен');

}

try {

const response = await fetch(

`${KINOPOISK\_API\_BASE\_URL}/films/${kinopoiskId}`,

{

headers: {

'X-API-KEY': this.apiKey,

'Content-Type': 'application/json',

},

}

);

if (!response.ok) {

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

}

const data = await response.json();

return data;

} catch (error) {

console.error('Ошибка получения деталей фильма:', error);

throw new Error('Не удалось получить информацию о фильме');

}

}

// Получение топ фильмов

async getTopFilms(page: number = 1): Promise<KinopoiskSearchResponse> {

if (!this.apiKey) {

throw new Error('API ключ Кинопоиска не настроен');

}

try {

const response = await fetch(

`${KINOPOISK\_API\_BASE\_URL}/films/top?type=TOP\_250\_BEST\_FILMS&page=${page}`,

{

headers: {

'X-API-KEY': this.apiKey,

'Content-Type': 'application/json',

},

}

);

if (!response.ok) {

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

}

const data = await response.json();

return data;

} catch (error) {

console.error('Ошибка получения топ фильмов:', error);

throw new Error('Не удалось получить топ фильмы');

}

}

// Проверка доступности API

async checkApiAvailability(): Promise<boolean> {

try {

await this.getTopFilms(1);

return true;

} catch {

return false;

}

}

}

// Экспорт экземпляра сервиса

export const kinopoiskService = KinopoiskService.getInstance();

// Утилиты для работы с данными фильмов

export const filmUtils = {

// Форматирование рейтинга

formatRating(rating?: number): string {

if (!rating) return 'Нет рейтинга';

return rating.toFixed(1);

},

// Форматирование продолжительности

formatDuration(duration?: number): string {

if (!duration) return 'Не указано';

const hours = Math.floor(duration / 60);

const minutes = duration % 60;

return `${hours}ч ${minutes}м`;

},

// Форматирование жанров

formatGenres(genres: Array<{ genre: string }>): string {

return genres.map(g => g.genre).join(', ');

},

// Форматирование стран

formatCountries(countries: Array<{ country: string }>): string {

return countries.map(c => c.country).join(', ');

},

// Получение URL постера с fallback

getPosterUrl(film: KinopoiskFilm): string {

return film.posterUrl || film.posterUrlPreview || '/placeholder-poster.jpg';

},

// Создание краткого описания

getShortDescription(film: KinopoiskFilm): string {

if (film.description) {

return film.description.length > 150

? film.description.substring(0, 150) + '...'

: film.description;

}

return 'Описание не доступно';

}

};

Файл layout.tsx:

import type { Metadata } from "next";

import { Geist, Geist\_Mono } from "next/font/google";

import "./globals.css";

const geistSans = Geist({

variable: "--font-geist-sans",

subsets: ["latin"],

});

const geistMono = Geist\_Mono({

variable: "--font-geist-mono",

subsets: ["latin"],

});

export const metadata: Metadata = {

title: "КиноВыбор - Выбор фильма без споров",

description: "Веб-приложение для выбора фильма в группе друзей без споров",

};

export default function RootLayout({

children,

}: Readonly<{

children: React.ReactNode;

}>) {

return (

<html lang="ru">

<body

className={`${geistSans.variable} ${geistMono.variable} antialiased`}

>

{children}

</body>

</html>

);

}

Файл page.tsx:

import HomePage from "@/app/Home";

export default function Home() {

return <HomePage />;

}

Файл group/[id]/page.tsx:

import React from "react";

import GroupPageClient from "@/components/GroupPageClient";

interface GroupPageProps {

params: Promise<{

id: string;

}>;

}

export default async function GroupPage({ params }: GroupPageProps) {

const { id: groupCode } = await params;

return <GroupPageClient groupCode={groupCode} />;

}

Файл results/[id]/page.tsx:

import React from "react";

import ResultsPageClient from "@/components/ResultsPageClient";

interface ResultsPageProps {

params: Promise<{

id: string;

}>;

}

export default async function ResultsPage({ params }: ResultsPageProps) {

const { id: groupCode } = await params;

// Нормализуем код группы (верхний регистр, без пробелов)

const normalizedCode = groupCode?.toUpperCase().trim() || groupCode;

return <ResultsPageClient groupCode={normalizedCode} />;

}

Файл vote/[id]/page.tsx:

import React from "react";

import VotingPageClient from "@/components/VotingPageClient";

interface VotingPageProps {

params: Promise<{

id: string;

}>;

}

export default async function VotingPage({ params }: VotingPageProps) {

const { id: groupCode } = await params;

// Нормализуем код группы (верхний регистр, без пробелов)

const normalizedCode = groupCode?.toUpperCase().trim() || groupCode;

return <VotingPageClient groupCode={normalizedCode} />;

}