Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**ОБЛАЧНОЕ ХРАНИЛИЩЕ С ПОДДЕРЖКОЙ ВЕРСИОНИРОВАНИЯ ФАЙЛОВ**

БГУИР КП 6-05-0612-01-018 ПЗ

Студент Моров Н.В.

Руководитель Болтак С.В.

Минск 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc199096055)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc199096056)

[1.1 Обзор аналогов 6](#_Toc199096057)

[1.2 Постановка задачи 8](#_Toc199096058)

[2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 9](#_Toc199096059)

[2.1 Структура программы 9](#_Toc199096060)

[2.2 Проектирование интерфейса программного средства 10](#_Toc199096061)

[2.3 Проектирование функционала программного средства 14](#_Toc199096062)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 19](#_Toc199096063)

[3.1 Управление кмонатами просмотра 19](#_Toc199096064)

[3.2 Работа с чатом и пользователями 21](#_Toc199096065)

[3.3 Механизм Long Polling 22](#_Toc199096066)

[3.4 Управление жизненным циклом коммнат 24](#_Toc199096067)

[4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 26](#_Toc199096068)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 27](#_Toc199096069)

[5.1 Интерфейс программного средства 27](#_Toc199096070)

[5.2 Управление просмотром 27](#_Toc199096071)

[5.3 Использование чата 28](#_Toc199096072)

[ЗАКЛЮЧЕНИЕ 31](#_Toc199096073)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 32](#_Toc199096074)

[ПРИЛОЖЕНИЕ А. Исходный код программы 33](#_Toc199096075)

# ВВЕДЕНИЕ

Технологии реального времени стали неотъемлемой частью современной цифровой экосистемы, предоставляя пользователям удобные инструменты для совместной работы и развлечений. Платформы для синхронизированного просмотра контента обеспечивают высокий уровень вовлеченности, позволяя участникам одновременно смотреть видео и обмениваться впечатлениями, независимо от их местоположения, при наличии подключения к интернету. В отличие от простого обмена ссылками на видео, такие решения предлагают синхронизацию воспроизведения, общий чат и управление состоянием просмотра для всех участников.

Одной из ключевых функций современных платформ для совместного просмотра является поддержка синхронизации состояния воспроизведения. Этот механизм позволяет автоматически передавать информацию о текущем состоянии видео (воспроизведение/пауза, текущее время) всем участникам комнаты, обеспечивая идентичный опыт просмотра. Синхронизация особенно полезна для групп, где важно, чтобы все участники видели одну и ту же сцену одновременно и могли обсуждать ее в реальном времени. Однако не все существующие сервисы предлагают простую настройку, требуют регистрации или ограничены определенными видеохостингами, что делает актуальной задачу создания собственного легковесного решения для синхронного просмотра.

Разработка индивидуального решения позволяет реализовать гибкое управление комнатами просмотра, простую интеграцию локальных видеофайлов и кастомизированный пользовательский интерфейс. В отличие от многих стандартных сервисов, собственная система может обеспечить автоматическую очистку неактивных комнат, контроль над данными сессии (сообщения чата, состояние плеера) и минималистичный дизайн без излишних функций. Это особенно важно для пользователей, которым требуется простое и быстрое решение для совместного просмотра без сложных настроек и зависимостей от внешних платформ.

Целью данного проекта является разработка веб-приложения "CineSync" для синхронизированного просмотра видеофайлов с поддержкой общего чата. В рамках работы будут изучены основные принципы синхронизации состояния между клиентами, методы организации обмена данными в реальном времени (long polling), механизмы работы с состоянием видеоплеера, а также способы идентификации пользователей в рамках сессии. Итогом станет функциональное приложение, которое обеспечит удобное создание комнат для просмотра, синхронизацию видео и обмен сообщениями между участниками.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор аналогов

На сегодняшний день существует множество платформ и сервисов, предоставляющих пользователям возможность синхронного просмотра видеоконтента. Каждое из них обладает собственными особенностями, функциональными возможностями и ограничениями. В рамках данного обзора рассмотрим общие типы таких сервисов.

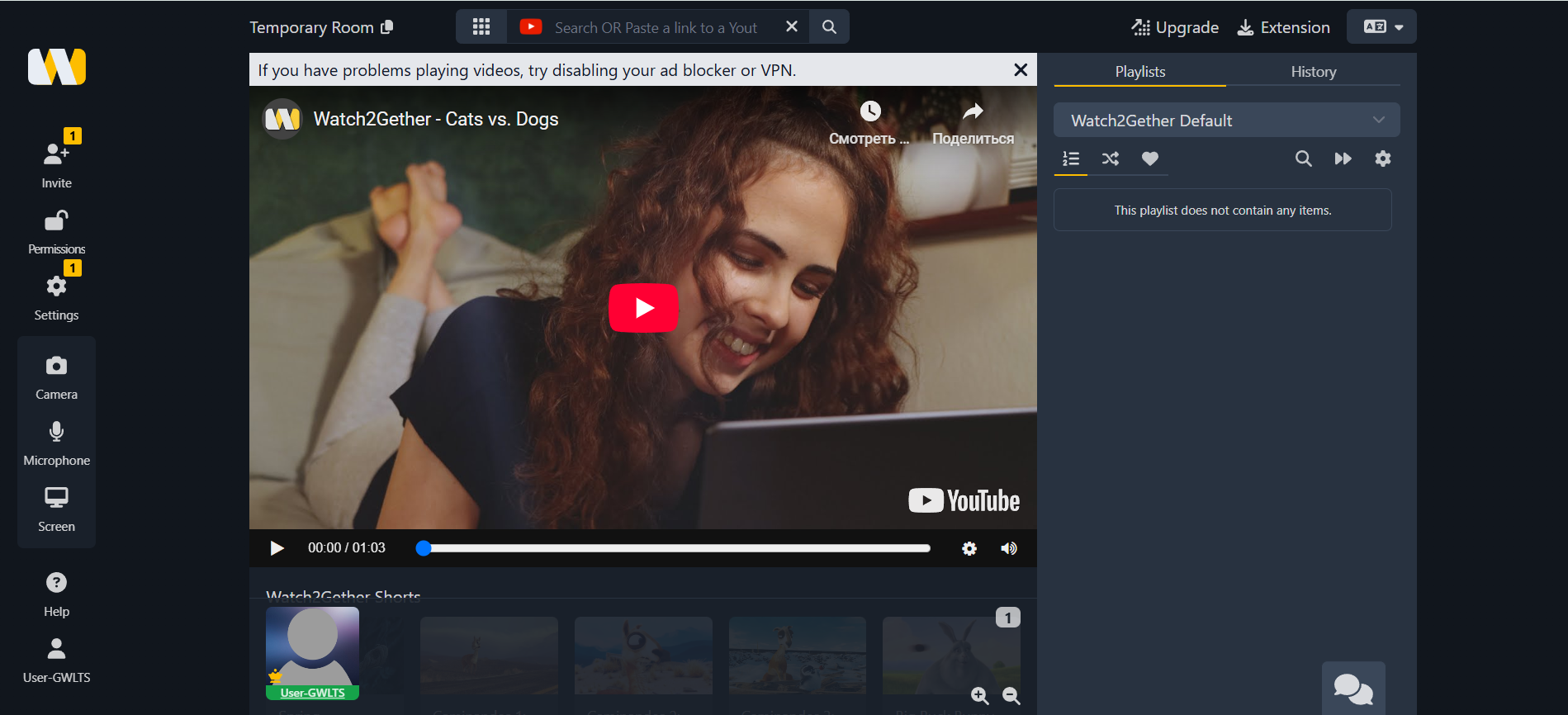


Рисунок 1.1 **–** Watch2Gether

В первую очередь стоит рассмотреть **веб-сервисы для совместного просмотра**, такие как Watch2Gether, Kast (ранее Eztv), Metastream или TwoSeven. Эти платформы обычно поддерживают интеграцию с популярными видеохостингами (YouTube, Vimeo и др.) и стриминговыми сервисами. Они предлагают создание "комнат", где участники могут добавлять видео в плейлист, смотреть их синхронно и общаться в чате. Некоторые из них также поддерживают демонстрацию экрана или передачу видео с локального компьютера. Однако такие сервисы часто требуют регистрации, могут содержать рекламу в бесплатных версиях или иметь ограничения по количеству участников.

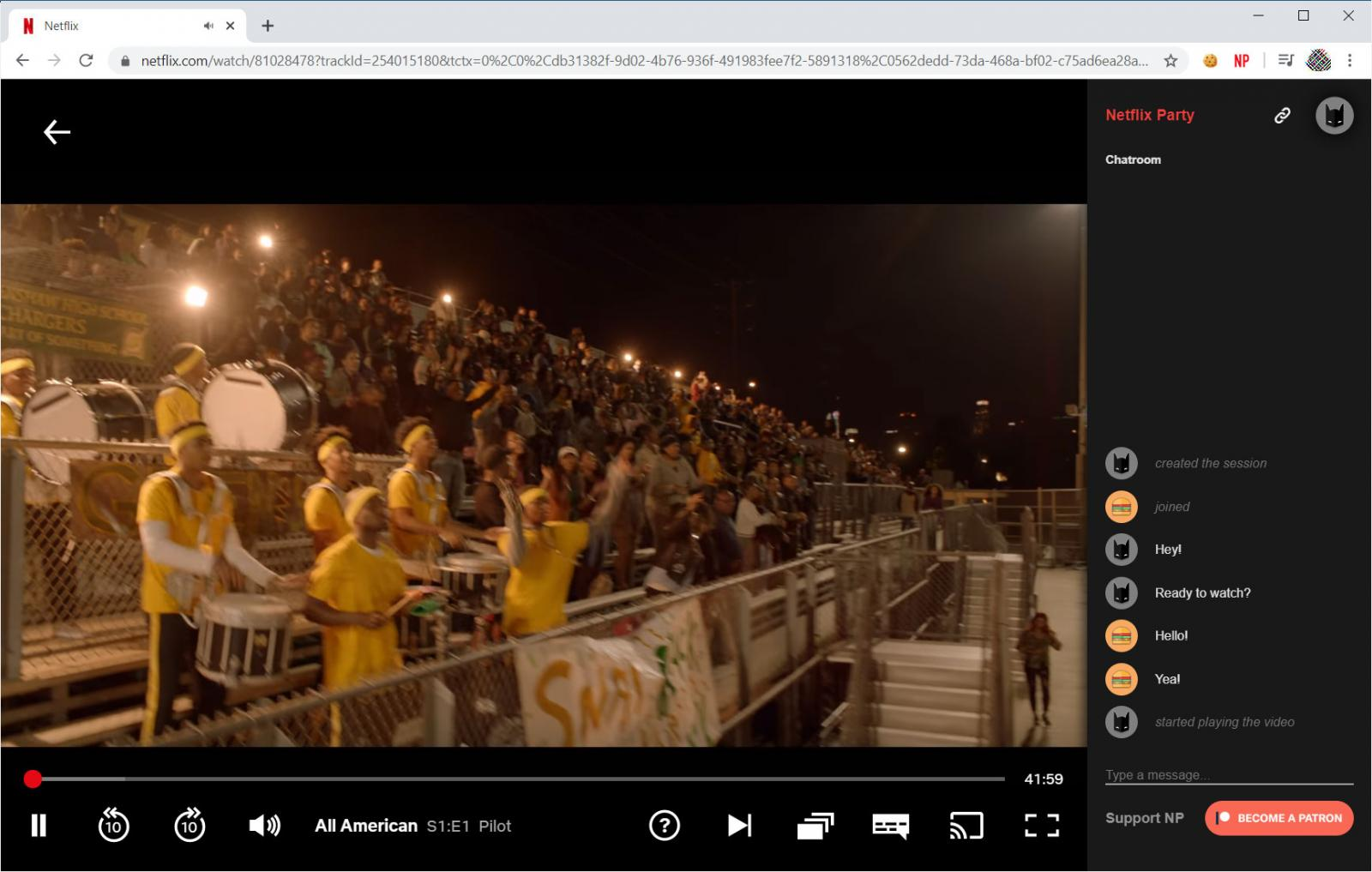


Рисунок 1.2 **–** Netflix Party

Следующим типом являются **браузерные расширения**, например, Netflix Party (теперь Teleparty) или Scener. Они встраиваются непосредственно в браузер и позволяют синхронизировать просмотр на конкретных стриминговых платформах (Netflix, HBO Max, Disney+ и т.д.). Преимуществом является тесная интеграция с интерфейсом платформы, но их функциональность ограничена поддерживаемыми сайтами и требует установки расширения всеми участниками.

Таким образом, существующие решения для синхронного просмотра обладают схожими базовыми возможностями (синхронизация видео, чат), но отличаются способом доступа (веб, расширение), поддержкой источников контента и моделью монетизации. На основе анализа аналогов можно выделить ключевые требования к разрабатываемому приложению «CineSync»: простота создания комнат, поддержка локальных видеофайлов (предопределенных на сервере), синхронизация состояния плеера (воспроизведение, пауза, перемотка), текстовый чат в реальном времени и легкий веб-интерфейс без необходимости сложной регистрации.

## Постановка задачи

В рамках данной курсовой работы необходимо разработать веб-приложение «CineSync» для синхронизированного просмотра видеофайлов с функцией чата. Данное программное средство должно обеспечивать надежную синхронизацию состояния видеоплеера между всеми участниками комнаты и удобный обмен сообщениями.

Система должна включать следующие основные функции:  
– Создание новой комнаты просмотра для выбранного видеофайла из предопределенного списка.  
– Присоединение пользователей к комнате по уникальной ссылке.  
– Ввод имени пользователя при входе в комнату для идентификации в чате.  
– Синхронизация состояния видеоплеера:  
– Воспроизведение/пауза.  
– Перемотка видео (синхронизация текущего времени).  
– Отображение видеофайлов, размещенных на сервере.  
– Обмен текстовыми сообщениями в чате комнаты в реальном времени.  
– Отображение количества активных пользователей в комнате.  
– Автоматическое обновление состояния видео и сообщений чата у всех клиентов (используя long polling).  
– "Версионирование" состояния комнаты (для видео и чата) для эффективного обновления клиентов.  
– Интерфейс для выбора видео, просмотра и общения через веб-приложение.  
– Автоматическая очистка неактивных комнат и пользователей.

Для реализации проекта будет использоваться язык программирования Python и веб-фреймворк FastAPI. В качестве механизма хранения данных о комнатах, их состоянии и сообщениях чата будет использоваться структура данных в оперативной памяти сервера (dict). Интерфейс взаимодействия с системой будет реализован в виде веб-приложения с использованием HTML, CSS и JavaScript (шаблонизатор Jinja2). Для обеспечения асинхронной обработки запросов и поддержки long polling будут использоваться возможности FastAPI и asyncio. Приложение будет имитировать работу с несколькими "узлами" (комнатами), каждая из которых имеет свое состояние и управляется независимо, с использованием блокировок (asyncio.Lock) для предотвращения состояния гонки.

Разрабатываемое программное средство должно обеспечивать надежность синхронизации, удобство использования и интерактивность, что позволит пользователям эффективно совместно просматривать видео и общаться.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

При разработке приложения «CineSync» будет использована следующая структура файлов и модулей (основной код находится в manager.py):

– **manager.py** – основной модуль приложения, содержащий:  
– Инициализацию FastAPI приложения.  
– Определение маршрутов (API endpoints) для:  
– Отображения главной страницы (/).  
– Создания комнаты (/create\_room/{video\_name}).  
– Отображения страницы комнаты (/room/{room\_id}).  
– Обновления состояния видео в комнате (/room/{room\_id}/update).  
– Отправки сообщений в чат (/room/{room\_id}/chat).  
– Получения обновлений состояния и чата (long polling) (/room/{room\_id}/poll).  
– Получения истории сообщений (/room/{room\_id}/messages).  
– Удаления комнаты (административная функция) (/room/{room\_id}).  
– Получения статистики комнаты (/room/{room\_id}/stats).  
– Логику управления комнатами (создание, хранение состояния, сообщений, списка пользователей).  
– Класс ChatMessage для представления сообщений чата.  
– Хранилища данных в памяти: rooms (словарь для комнат) и room\_locks (словарь для блокировок комнат).  
– Фоновую задачу cleanup\_inactive\_rooms для очистки неактивных комнат.  
– Обработчики событий startup (запуск фоновой задачи) и shutdown.

– **run.py** – скрипт для удобного запуска uvicorn-сервера с приложением manager:app. Включает проверки наличия необходимых директорий и видеофайлов.

– **templates/** – директория, содержащая HTML-шаблоны Jinja2:  
– **index.html** – шаблон главной страницы, отображающий список доступных видео для создания комнат.  
– **room.html** – шаблон страницы комнаты просмотра, включающий видеоплеер, чат и элементы управления.

– **static/** – директория для статических файлов:  
– Видеофайлы (\*.mp4), которые будут доступны для просмотра.

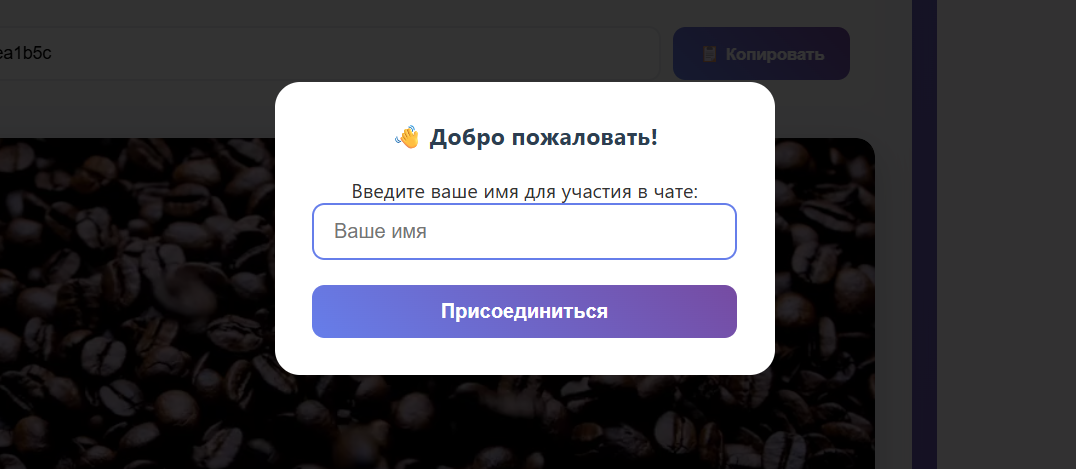
– **Переменные окружения и конфигурация:** В данном коде явного .env файла или отдельного config.py нет. Конфигурация (например, хост и порт для uvicorn) задается в run.py или при запуске. Настройки, такие как максимальная длина сообщения или время жизни неактивных пользователей/комнат, захардкожены в manager.py.

## 2.2 Проектирование интерфейса программного средства

При разработке интерфейса программного средства «CineSync» за основу будет взят минималистичный и интуитивно понятный дизайн, ориентированный на удобство совместного просмотра видео.

**2.2.1** **Вход в комнату**

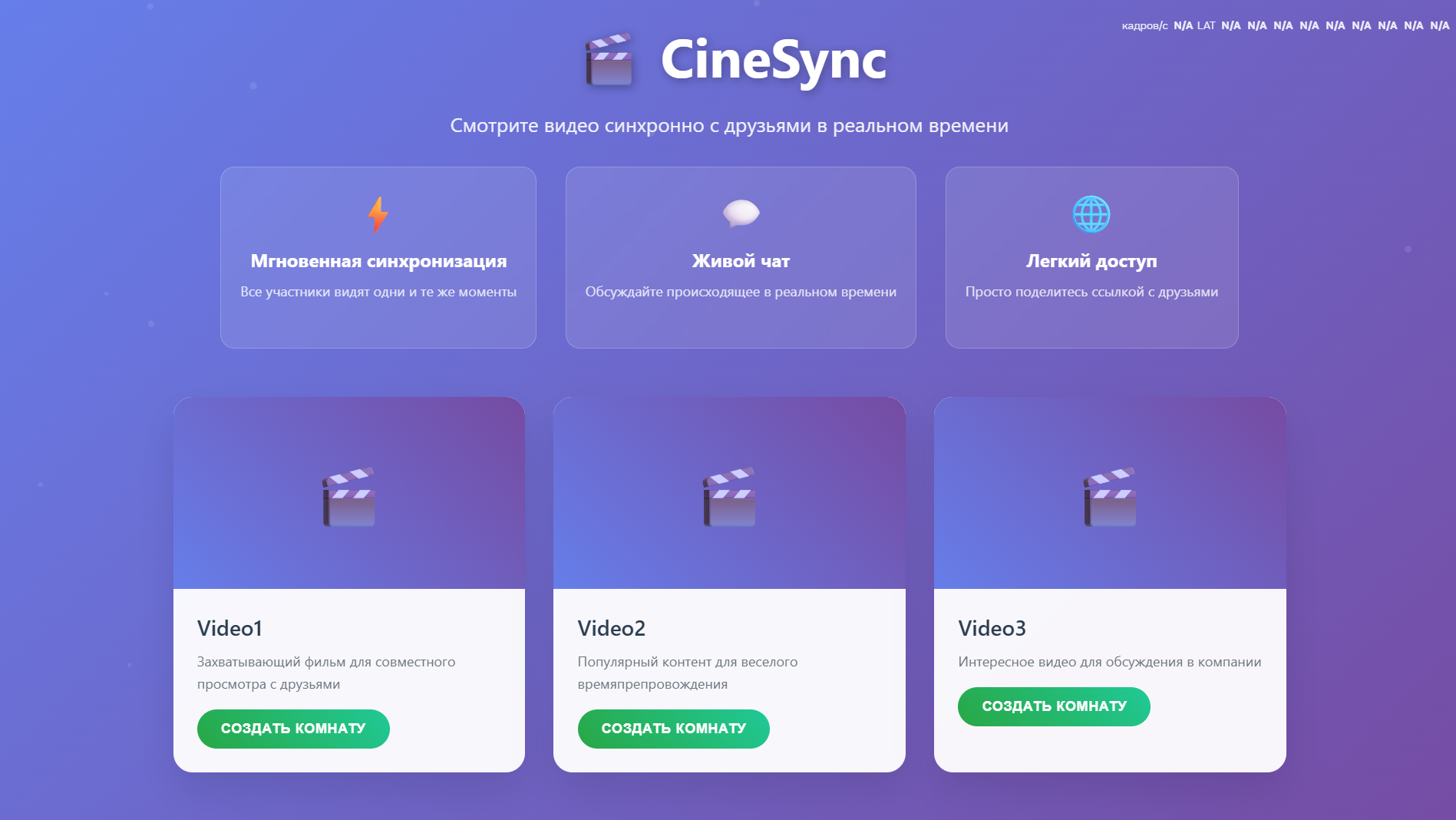
Перед полноценным присоединением к комнате просмотра и чату, пользователю будет предложено ввести свое имя. Это имя будет использоваться для идентификации в чате.  
Окно (модальное) для ввода имени содержит поле для ввода текста и кнопку "Присоединиться". Внешний вид макета окна ввода имени представлен на рисунке 2.1.

****

**Рисунок 2.1 – Ввод имени**

2.2.2 **Главное окно**

Главное окно веб-приложения (index.html) представляет собой страницу, где пользователи могут выбрать видео для совместного просмотра и создать новую комнату.  
– В верхней части страницы расположен заголовок "CineSync" и краткое описание сервиса.  
– Ниже представлены карточки доступных видео. Каждая карточка содержит:  
– Название видео.  
– Краткое описание или иконку.  
– Кнопку "Создать комнату".  
Внешний вид главного окна представлен на рисунке 2.2.

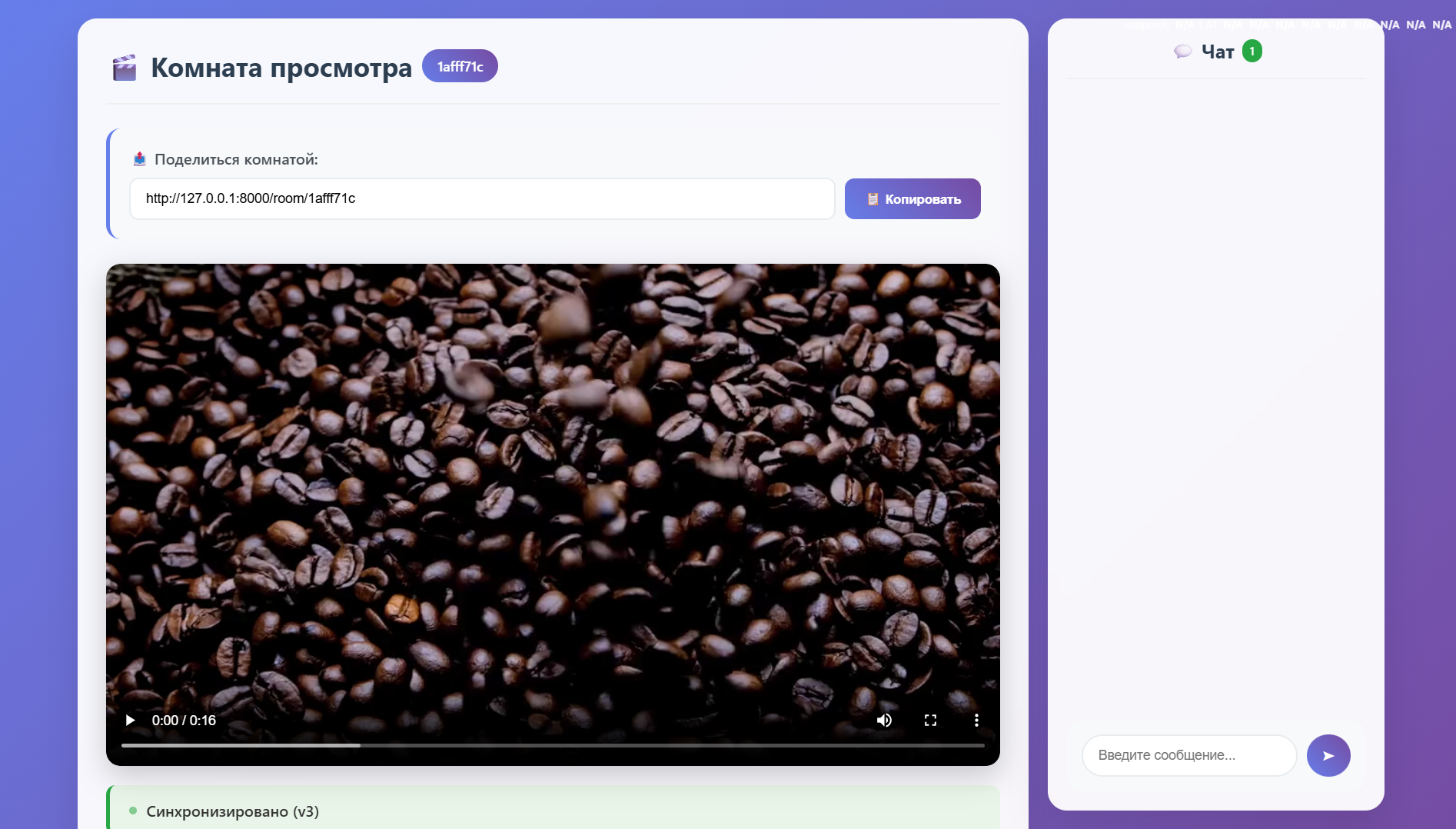
****

**Рисунок 2.2 – Главное окно приложения**

2.2.3 **Элементы управления приложения**

Основной интерфейс приложения, где происходит синхронизированный просмотр видео и общение в чате (room.html). Окно разделено на две основные части: область просмотра видео и панель чата.  
– **Область просмотра видео:**  
– Заголовок с ID комнаты.  
– Секция для копирования ссылки на комнату.  
– Встроенный HTML5 видеоплеер для отображения выбранного видео.  
– Индикатор статуса синхронизации.  
– **Панель чата:**  
– Заголовок чата с отображением количества онлайн-пользователей.  
– Область отображения сообщений чата (имя отправителя, текст сообщения, время).  
– Поле для ввода нового сообщения.  
– Кнопка для отправки сообщения.

Элементы управления:  
– **Видеокарточка на главной странице** (для видео) – при клике на кнопку "Создать комнату" происходит переход на страницу комнаты с выбранным видео.  
– **HTML5 видеоплеер**: стандартные элементы управления (play/pause, ползунок времени, громкость), действия с которыми синхронизируются.  
– **"Копировать" (для ссылки на комнату)** – копирует URL комнаты в буфер обмена.  
– **Поле ввода сообщения и кнопка "Отправить"** – для общения в чате.  
Внешний вид окна комнаты просмотра представлен на рисунке 2.3.



**Рисунок 2.3 – Окно комнаты просмотра**

2.2.4 **Навигация**

Навигация в приложении проста:  
– С главной страницы (index.html) пользователь выбирает видео, что приводит к созданию комнаты и перенаправлению на страницу комнаты (room.html/{room\_id}).  
– Пользователи могут напрямую заходить в комнату, если у них есть ссылка (/room/{room\_id}).  
– Возврат на главную страницу возможен через ручной ввод URL или кнопку "Назад" в браузере.

2.2.5 **Управление версиями файлов**

Программное средство не управляет версиями файлов в традиционном смысле. Вместо этого, оно использует "версии" для отслеживания изменений состояния видеоплеера и чата в комнате:  
– **state.version**: числовой идентификатор версии состояния видео (воспроизведение, время). Увеличивается при каждом изменении.  
– **message\_version**: числовой идентификатор версии списка сообщений. Увеличивается при добавлении нового сообщения.  
Клиенты при опросе сервера (long polling) передают свои текущие версии. Если версии на сервере новее, клиент получает обновления. Это обеспечивает эффективную синхронизацию без передачи избыточных данных.  
Визуального интерфейса для "управления версиями состояния" нет, это внутренний механизм синхронизации.

2.2.6 **Обработка сессии и безопасности**

– **Идентификация пользователя:** При входе в комнату пользователь указывает имя. Генерируется случайный user\_id на стороне клиента для его уникальной идентификации в рамках сессии комнаты.  
– **Безопасность:** Доступ к комнате осуществляется по уникальной, но публично известной ссылке. Специальных механизмов шифрования данных чата или сессии, кроме HTTPS (если сервер настроен соответствующим образом), не предусмотрено.  
– **Управление активностью:** Сервер отслеживает last\_seen для каждого пользователя в комнате. Неактивные пользователи удаляются из списка участников. Неактивные комнаты (без пользователей или без обновлений состояния) могут быть удалены фоновой задачей.  
– **Long Polling Timeout:** Механизм long polling (/room/{room\_id}/poll) имеет таймаут (30 секунд). Если за это время обновлений нет, сервер отвечает, и клиент немедленно делает новый запрос. Это поддерживает "живое" соединение.

## 2.3 Проектирование функционала программного средства

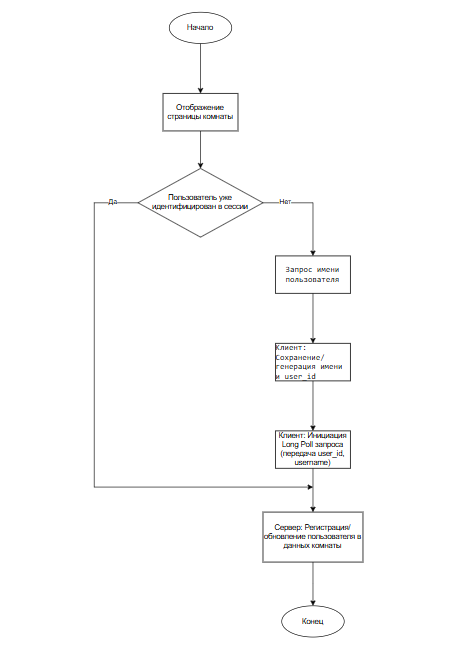
Для обеспечения удобного и эффективного совместного просмотра видео, приложение «CineSync» должно реализовывать следующий минимальный набор функций:

– Идентификация пользователя при входе в комнату.  
– Создание и присоединение к комнатам просмотра.  
– Синхронизация состояния видеоплеера.  
– Обмен сообщениями в чате.  
– Отслеживание и отображение онлайн-пользователей.  
– Обновление состояния через long polling.  
– Автоматическая очистка неактивных сущностей.

2.3.1 **Вход в комнату**

**П**ри первом входе на страницу комнаты (/room/{room\_id}), пользователю предлагается ввести свое имя (псевдоним). Это имя будет использоваться для его отображения в чате и для идентификации его действий (например, отправка сообщений). На стороне клиента (в JavaScript, не показано в manager.py, но подразумевается для взаимодействия с API) также генерируется уникальный user\_id (например, UUID), который передается на сервер при отправке сообщений или обновлении состояния плеера. Этот user\_id совместно с введенным именем помогает серверу отслеживать активных пользователей в комнате и корректно атрибутировать сообщения в чате. Состояние пользователя (имя, user\_id, время последней активности) хранится в оперативной памяти сервера в структуре данных комнаты (rooms[room\_id]["users"]).

**Блок-схема метода, который осуществляет вход пользователей представлена на рисунке 2.4.**

****

**Рисунок 2.4 – Блок-схема Вход пользователя**

2.3.2 **Навигация**

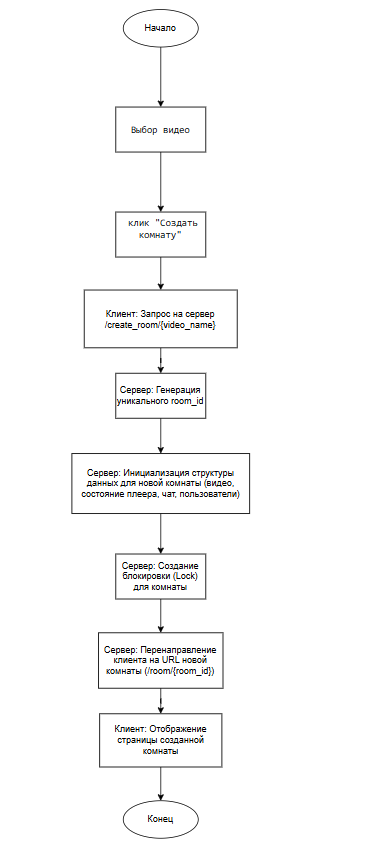
Навигация в приложении «CineSync» организована следующим образом:

**Главная страница (/):** Пользователь попадает на главную страницу, где отображается список предопределенных на сервере видеофайлов, доступных для совместного просмотра. Для каждого видео есть кнопка "Создать комнату".

**Создание комнаты:** При нажатии на кнопку "Создать комнату" для выбранного видео (/create\_room/{video\_name}), сервер генерирует уникальный идентификатор комнаты (room\_id), инициализирует ее состояние и перенаправляет пользователя на страницу этой комнаты (/room/{room\_id}).

**Присоединение к комнате по ссылке:** Пользователи могут присоединиться к уже существующей комнате, если у них есть прямая ссылка вида http://<адрес\_сервера>/room/{room\_id}. При переходе по такой ссылке, если комната существует, пользователь попадает на страницу просмотра.  
В приложении отсутствует концепция "папок" или файловой системы, доступной пользователю. Все взаимодействие происходит либо через главную страницу для создания новых сессий просмотра, либо через прямые ссылки на комнаты.

**Блок-схема навигации представлена на рисунке 2.5.**

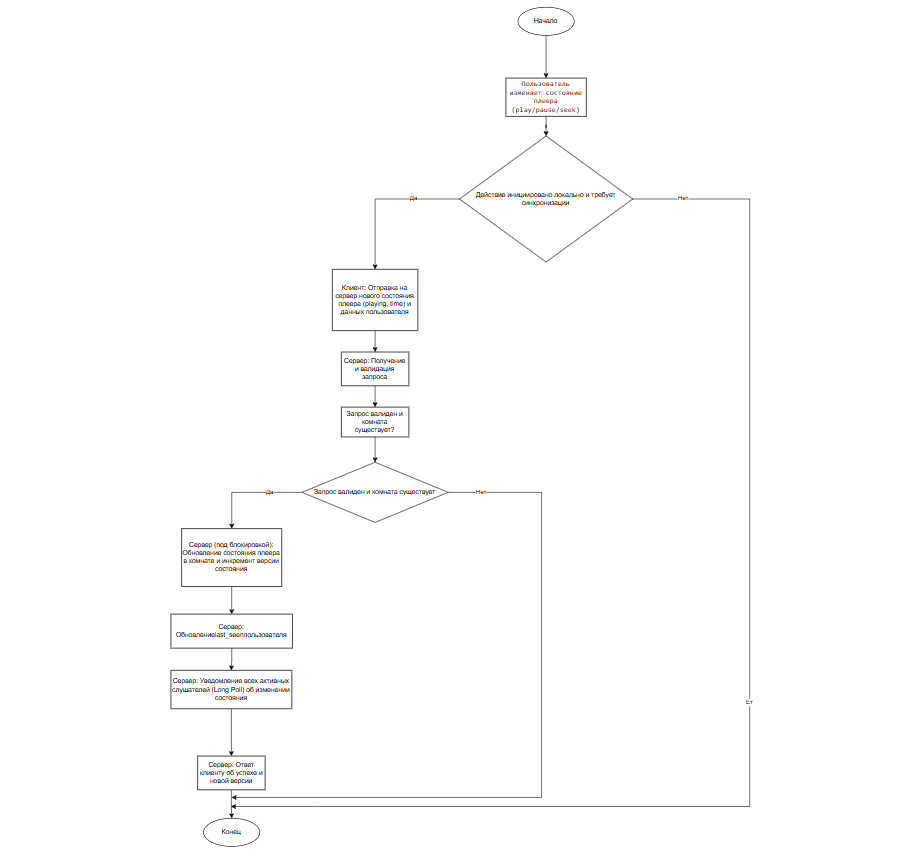
****

**Рисунок 2.5 – Блок-схема Навигация**

2.3.3 **Управление жизненным циклом комнат**

* Управление жизненным циклом комнат в приложении «CineSync» организована следующим образом:
* **Создание комнаты:** Комната создается автоматически сервером, когда пользователь выбирает видео на главной странице. Для каждой комнаты создается запись в словаре rooms на сервере, содержащая информацию о видео, состоянии плеера, списке пользователей и сообщениях чата. Также для каждой комнаты создается объект asyncio.Lock для безопасного конкурентного доступа.
* **Удаление комнаты:**
  + **Автоматическое:** Реализована фоновая задача (cleanup\_inactive\_rooms), которая периодически проверяет все активные комнаты. Если комната пуста (нет активных пользователей) или неактивна в течение определенного времени, она автоматически удаляется с сервера (запись удаляется из rooms и room\_locks).
  + **Административное:** Предусмотрен API эндпоинт (DELETE /room/{room\_id}), который позволяет принудительно удалить комнату. Эта функция не предназначена для обычных пользователей.  
    Управление жизненным циклом комнат направлено на эффективное использование ресурсов сервера.

**Блок-схема Управление жизненным циклом комнат представлена на рисунке 2.6.**

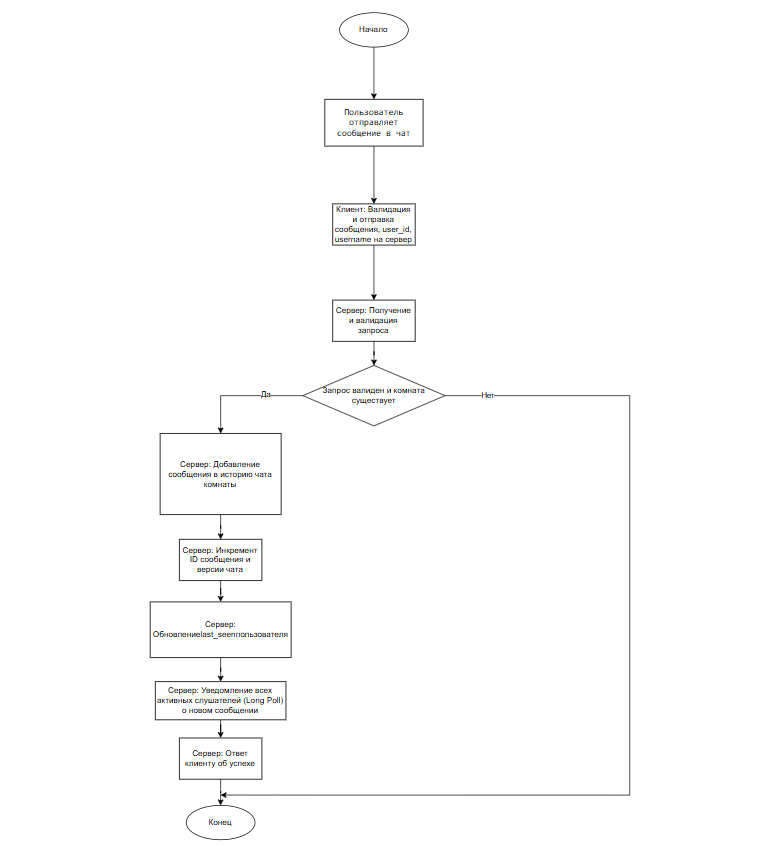
****

**Рисунок 2.6 – Блок-схема Управление жизненным циклом комнат**

2.3.4 **Взаимодействие с контентом и чатом**

* **Просмотр видео:** Видеофайлы (.mp4) заранее размещены на сервере в директории static/. Пользователи "стримят" (просматривают) эти видео через стандартный HTML5 видеоплеер, встроенный на странице комнаты. Состояние плеера (воспроизведение, пауза, время) синхронизируется между всеми участниками комнаты.
* **Отправка и получение сообщений чата:**
  + **Отправка:** Пользователь вводит текстовое сообщение в поле чата и отправляет его. Запрос (POST /room/{room\_id}/chat) содержит текст сообщения, user\_id и username отправителя. Сервер добавляет сообщение в историю чата комнаты, присваивает ему уникальный id и timestamp, обновляет message\_version комнаты и уведомляет всех слушателей (long poll) о новом сообщении.
  + **Получение:** Новые сообщения доставляются клиентам через механизм long polling (/room/{room\_id}/poll). Клиент также может запросить историю сообщений (GET /room/{room\_id}/messages).
* **Удаление контента:** Видеофайлы не удаляются пользователями. Сообщения чата хранятся в памяти сервера с ограничением на количество (например, последние 100 сообщений), старые сообщения могут быть отброшены при превышении лимита.

**Блок-схема Взаимодействие с контентом и чатом представлена на рисунке 2.7.**

****

**Рисунок 2.7 – Блок-схема Взаимодействие с контентом и чатом**

2.3.5 **Версионирование состояния для синхронизации**

Версионирование в «CineSync» это ключевой механизм для эффективной синхронизации клиентов через long polling:

**state.version**: Это числовой идентификатор версии состояния видеоплеера (воспроизведение/пауза, текущее время time). Он инкрементируется на сервере каждый раз, когда состояние плеера изменяется одним из участников (POST /room/{room\_id}/update).

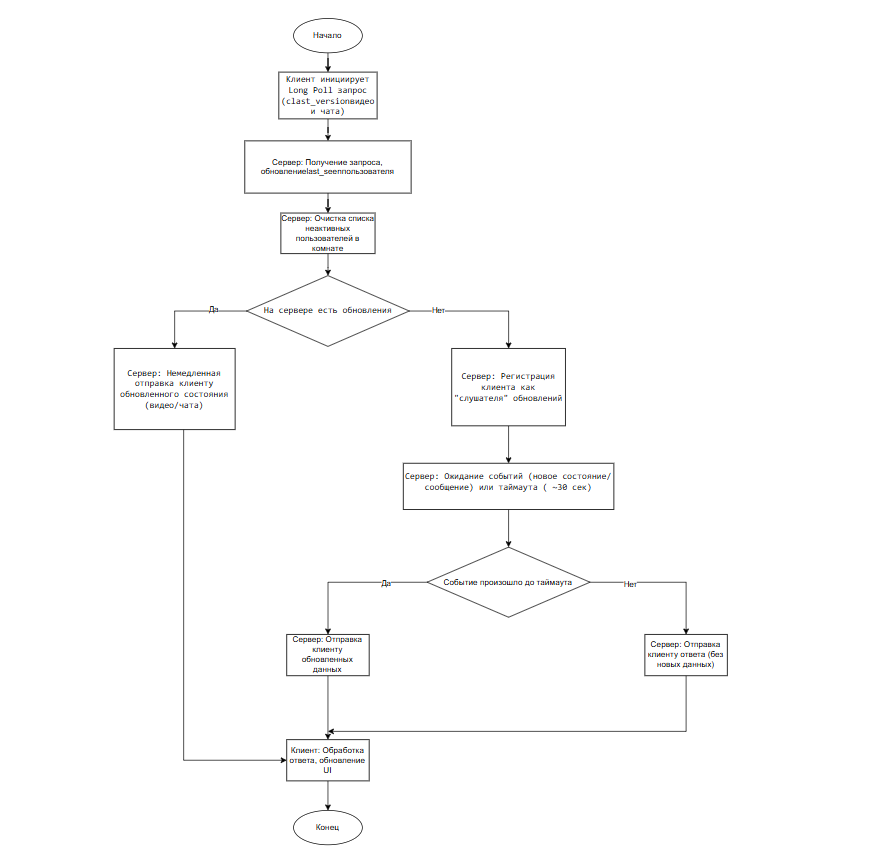
**message\_version**: Это числовой идентификатор версии списка сообщений чата. Он инкрементируется на сервере каждый раз, когда в чат добавляется новое сообщение (POST /room/{room\_id}/chat).

Когда клиент делает long-poll запрос (GET /room/{room\_id}/poll), он передает серверу last\_version (последнюю известную ему версию состояния видео) и last\_message\_version (последнюю известную ему версию сообщений).  
Сервер сравнивает эти версии с текущими версиями в комнате:

Если версии на сервере новее, сервер немедленно отправляет обновленное состояние видео и/или новые сообщения.

Если версии совпадают, запрос "зависает" до тех пор, пока не произойдет изменение состояния (и, следовательно, инкремент версии) или не истечет таймаут.  
Этот механизм позволяет минимизировать передачу избыточных данных и обеспечивает быструю доставку только актуальных изменений. Визуального интерфейса для управления этими "версиями состояния" для пользователя нет, это внутренний механизм синхронизации.

**Блок-схема Версионирование состояния для синхронизации представлена на рисунке 2.8.**

****

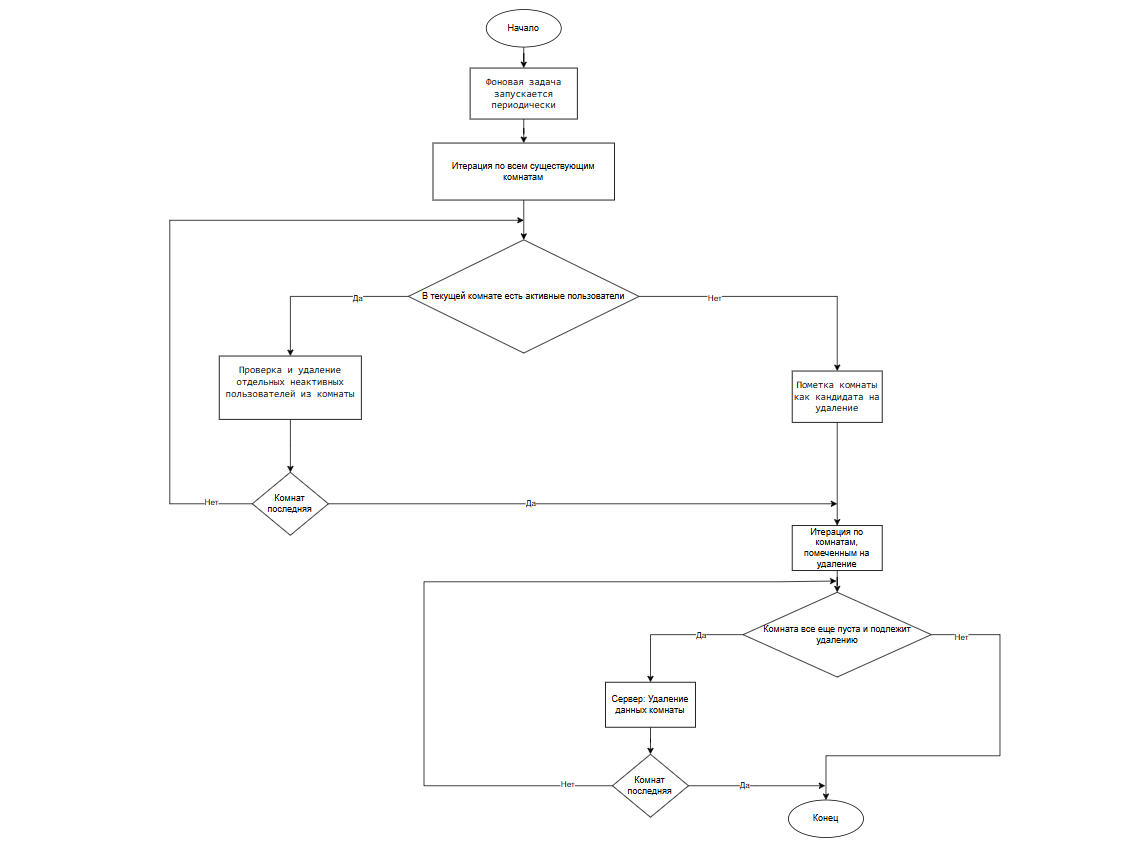
**Рисунок 2.8 – Блок-схема Версионирование состояния для синхронизации**

2.3.6 **Управление сессией пользователя в комнате**

Сессия пользователя в приложении «CineSync» привязана к его нахождению в конкретной комнате просмотра

* **Идентификация и активность:** Пользователь идентифицируется в комнате по user\_id и username. Его активность отслеживается сервером через поле last\_seen (временная метка последнего взаимодействия), которое обновляется при:
  + Отправке сообщения в чат (/room/{room\_id}/chat).
  + Обновлении состояния плеера (/room/{room\_id}/update).
  + Каждом long-poll запросе (/room/{room\_id}/poll).
* **Поддержание "живого" соединения:** Механизм long polling сам по себе служит для поддержания квази-активного соединения. Клиент постоянно (с небольшими перерывами или по таймауту long poll запроса) опрашивает сервер.
* **Очистка неактивных пользователей:** В рамках long-poll обработчика (poll\_state) и фоновой задачи (cleanup\_inactive\_rooms) сервер периодически проверяет last\_seen для каждого пользователя в комнате. Если пользователь не проявлял активность в течение заданного интервала (например, 30-60 секунд), он удаляется из списка активных пользователей комнаты (rooms[room\_id]["users"]). Это также влияет на счетчик онлайн-пользователей.  
  Таким образом, "сессия" пользователя в комнате активна, пока он взаимодействует с ней или пока его long-poll запросы поддерживают его last\_seen актуальным. При длительном бездействии пользователь "выпадает" из комнаты.

**Блок-схема Управление сессией пользователя в комнате представлена на рисунке 2.9.**

****

**Рисунок 2.9 – Блок-схема Управление сессией пользователя в комнате**

# РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 3.1 Управление комнатами просмотра

Основным функционалом системы является возможность создания комнат для синхронного просмотра видео, управление состоянием воспроизведения и обмен сообщениями в чате. Все операции осуществляются через REST API, реализованное с помощью FastAPI.

3.1.1 **Созданеи комнаты просмотра**

Пользователь может создать новую комнату просмотра, выбрав одно из доступных на сервере видео. Система генерирует уникальный идентификатор для комнаты и инициализирует её начальное состояние.

**# manager.py**

**@app.get("/create\_room/{video\_name}")**

**def create\_room(video\_name: str):**

**room\_id = str(uuid.uuid4())[:8] # Генерация уникального ID комнаты**

**rooms[room\_id] = {**

**"video": video\_name, # Связываем комнату с выбранным видео**

**"state": {"playing": False, "time": 0.0, "version": 0}, # Начальное состояние плеера**

**"listeners": {}, # Слушатели для long polling**

**"users": {}, # Активные пользователи в комнате**

**"messages": [], # История сообщений чата**

**"last\_message\_id": 0, # Счетчик ID сообщений**

**"message\_version": 0 # Версия состояния чата**

**}**

**room\_locks[room\_id] = asyncio.Lock() # Блокировка для конкурентного доступа**

**return RedirectResponse(f"/room/{room\_id}") # Перенаправление в созданную комнату**

Сначала генерируется короткий уникальный room\_id. Затем в глобальном словаре rooms создается запись для новой комнаты, содержащая имя видеофайла, начальное состояние видеоплеера (пауза, время 0.0, версия 0), пустые словари для слушателей long polling и пользователей, пустой список сообщений и начальные версии для сообщений. Также создается asyncio.Lock для этой комнаты для безопасного изменения её состояния. После успешного создания пользователь перенаправляется на страницу этой комнаты.

3.1.2 **Обновление состояния видеоплеера**

Любой участник комнаты может изменить состояние видеоплеера (воспроизведение, пауза, перемотка). Это действие отправляет запрос на сервер, который обновляет общее состояние комнаты и уведомляет других участников.

**# manager.py**

**@app.post("/room/{room\_id}/update")**

**async def update\_state(room\_id: str, request: Request):**

**if room\_id not in rooms:**

**raise HTTPException(status\_code=404, detail="Комната не найдена")**

**try:**

**data = await request.json()**

**playing = data.get("playing") # Получаем флаг воспроизведения**

**time\_pos = data.get("time") # Получаем текущее время видео**

**user\_id = data.get("user\_id")**

**username = data.get("username")**

**if playing is None or time\_pos is None: # Валидация входных данных**

**raise ValueError("Неверные данные")**

**except Exception as e:**

**raise HTTPException(status\_code=400, detail="Неверный формат данных")**

**async with room\_locks[room\_id]: # Используем блокировку для безопасного обновления**

**room = rooms[room\_id]**

**if user\_id and username: # Обновляем информацию о пользователе**

**room["users"][user\_id] = {**

**"username": username,**

**"last\_seen": time.time()**

**}**

**# Обновляем состояние видео и увеличиваем его версию**

**room["state"] = {**

**"playing": playing,**

**"time": time\_pos,**

**"version": room["state"]["version"] + 1**

**}**

**# Уведомляем всех ожидающих слушателей (long poll)**

**for event in room["listeners"].values():**

**event.set()**

**room["listeners"] = {} # Очищаем список слушателей после уведомления**

**return JSONResponse({"status": "success", "version": room["state"]["version"]})**

Метод получает room\_id и данные о новом состоянии плеера (playing, time), а также информацию о пользователе, инициировавшем изменение. Под блокировкой room\_locks[room\_id] обновляется состояние rooms[room\_id]["state"] и инкрементируется его версия. Затем все клиенты, ожидающие обновлений через long polling (хранящиеся в room["listeners"]), уведомляются о событии. Функция возвращает новый номер версии состояния видео.

3.1.3 **Удаление комнаты**

Предусмотрена возможность удаления комнаты, например, для администрирования или автоматической очистки.

**# manager.py**

**@app.delete("/room/{room\_id}")**

**async def delete\_room(room\_id: str):**

**if room\_id in rooms:**

**async with room\_locks[room\_id]:**

**# Уведомляем слушателей, если они есть, перед удалением**

**for event in rooms[room\_id].get("listeners", {}).values():**

**event.set()**

**del rooms[room\_id] # Удаляем данные комнаты**

**if room\_id in room\_locks:**

**del room\_locks[room\_id] # Удаляем блокировку комнаты**

**return JSONResponse({"status": "success", "message": "Комната удалена"})**

**raise HTTPException(status\_code=404, detail="Комната не найдена")**

Функция проверяет существование комнаты. Если комната найдена, она удаляется из словарей rooms и room\_locks. Перед удалением уведомляются все активные слушатели, чтобы их long poll запросы завершились.

3.1.4 **Версионирование состояния комнаты**

Для эффективной синхронизации клиентов система использует числовые "версии" для отслеживания изменений состояния видеоплеера и списка сообщений чата.

room["state"]["version"]: Увеличивается при каждом изменении состояния видео (play, pause, seek).

room["message\_version"]: Увеличивается при добавлении нового сообщения в чат.

Клиенты при запросе обновлений (/room/{room\_id}/poll) передают свои текущие версии. Сервер отвечает только теми данными, которые новее версий клиента, что минимизирует передаваемый трафик и нагрузку. Этот механизм подробно рассмотрен в функции poll\_state.

## 3.2 Работа с чатом и пользователями

Для обеспечения интерактивного общения во время просмотра реализован текстовый чат.

3.2.1 **отправка и получение сообщений чата**

Пользователи могут отправлять текстовые сообщения в чат комнаты. Сервер сохраняет сообщения и распространяет их среди участников.

**# manager.py**

**@app.post("/room/{room\_id}/chat")**

**async def send\_chat\_message(room\_id: str, request: Request):**

**# ... (проверка существования комнаты и валидация данных как в коде) ...**

**async with room\_locks[room\_id]:**

**room = rooms[room\_id]**

**room["last\_message\_id"] += 1 # Генерируем ID для нового сообщения**

**message = ChatMessage(user\_id, username, message\_text) # Создаем объект сообщения**

**message\_dict = message.to\_dict()**

**message\_dict["id"] = room["last\_message\_id"]**

**room["messages"].append(message\_dict) # Добавляем сообщение в историю**

**room["message\_version"] += 1 # Увеличиваем версию сообщений**

**if len(room["messages"]) > 100: # Ограничиваем историю чата**

**room["messages"] = room["messages"][-100:]**

**# Обновляем информацию о пользователе (last\_seen)**

**room["users"][user\_id] = {**

**"username": username,**

**"last\_seen": time.time()**

**}**

**# Уведомляем всех слушателей о новом сообщении**

**for event in room["listeners"].values():**

**event.set()**

**room["listeners"] = {}**

**return JSONResponse({"status": "success", "message\_id": room["last\_message\_id"]})**

**@app.get("/room/{room\_id}/messages")**

**async def get\_messages(room\_id: str, limit: int = 50, after\_id: int = 0):**

**# ... (проверка существования комнаты) ...**

**room = rooms[room\_id]**

**# Фильтрация и возврат сообщений (как в коде)**

**# ...**

**return JSONResponse({**

**"messages": messages,**

**"total": len(room["messages"]),**

**"message\_version": room["message\_version"]**

**})**

При отправке сообщения (POST /room/{room\_id}/chat), оно добавляется в список room["messages"], присваивается уникальный id, и message\_version инкрементируется. При запросе истории сообщений (GET /room/{room\_id}/messages) возвращается часть истории, возможно, отфильтрованная по after\_id.

3.2.2 **Управление пользователем в комнате**

Система отслеживает активных пользователей в каждой комнате. Пользователь считается активным, если он недавно отправлял запросы на обновление состояния или сообщения.  
Информация о пользователях (user\_id, username, last\_seen) хранится в room["users"].  
Обновление last\_seen происходит при вызове update\_state, send\_chat\_message и poll\_state.  
Функция poll\_state также включает логику очистки неактивных пользователей из room["users"] если current\_time - user\_info["last\_seen"] > 30 секунд.  
Количество активных пользователей (online\_count) передается клиентам через poll\_state.

**3.3** **Механизм Long Polling**

Для обеспечения синхронизации состояния видео и чата без использования WebSockets, применяется техника long polling.

**@app.get("/room/{room\_id}/poll")**

**asyncdefpoll\_state(room\_id: str, last\_version: int=0, last\_message\_version: int=0, user\_id: str=None, username: str=None):**

**ifroom\_idnotinrooms:**

**raiseHTTPException(status\_code=404, detail="Комната не найдена")**

**client\_id=str(uuid.uuid4())**

**room=rooms[room\_id]**

**# Регистрируем пользователя**

**ifuser\_idandusername:**

**asyncwithroom\_locks[room\_id]:**

**room["users"][user\_id] ={**

**"username": username,**

**"last\_seen": time.time()**

**}**

**# Очистка неактивных пользователей (не активны более 30 сек)**

**current\_time=time.time()**

**asyncwithroom\_locks[room\_id]:**

**inactive\_users=[**

**uidforuid, user\_infoinroom["users"].items()**

**ifcurrent\_time-user\_info["last\_seen"] >30**

**]**

**foruidininactive\_users:**

**delroom["users"][uid]**

**# Получаем текущее состояние**

**current\_state={**

**"state": room["state"],**

**"version": room["state"]["version"],**

**"message\_version": room["message\_version"],**

**"online\_count": len(room["users"]),**

**"messages": []**

**}**

**# Обрабатываем сообщения**

**iflast\_message\_version==0:**

**# Первое подключение - отправляем последние 10 сообщений**

**current\_state["messages"] =room["messages"][-10:] ifroom["messages"] else[]**

**elifroom["message\_version"] >last\_message\_version:**

**# Есть новые сообщения - отправляем только новые**

**# Находим новые сообщения, сравнивая по ID**

**ifroom["messages"]:**

**# Простой способ: отправляем последние несколько сообщений**

**# В реальном приложении лучше отслеживать по timestamp или ID**

**current\_state["messages"] =room["messages"][-3:] iflen(room["messages"]) >=3elseroom["messages"]**

**# Проверяем, нужно ли ждать обновлений**

**need\_to\_wait=(last\_version>=room["state"]["version"] and**

**last\_message\_version>=room["message\_version"])**

**ifneed\_to\_wait:**

**event=asyncio.Event()**

**asyncwithroom\_locks[room\_id]:**

**room["listeners"][client\_id] =event**

**try:**

**awaitasyncio.wait\_for(event.wait(), timeout=30.0)**

**exceptasyncio.TimeoutError:**

**pass**

**finally:**

**asyncwithroom\_locks[room\_id]:**

**ifclient\_idinroom["listeners"]:**

**delroom["listeners"][client\_id]**

**# Получаем обновленное состояние после ожидания**

**asyncwithroom\_locks[room\_id]:**

**# Снова очищаем неактивных пользователей**

**current\_time=time.time()**

**inactive\_users=[**

**uidforuid, user\_infoinroom["users"].items()**

**ifcurrent\_time-user\_info["last\_seen"] >30**

**]**

**foruidininactive\_users:**

**delroom["users"][uid]**

**# Формируем финальный ответ**

**current\_state={**

**"state": room["state"],**

**"version": room["state"]["version"],**

**"message\_version": room["message\_version"],**

**"online\_count": len(room["users"]),**

**"messages": []**

**}**

**# Отправляем новые сообщения только если версия изменилась**

**ifroom["message\_version"] >last\_message\_version:**

**current\_state["messages"] =room["messages"][-3:] iflen(room["messages"]) >=3elseroom["messages"]**

**returnJSONResponse(current\_state)**

Клиент отправляет GET-запрос на /room/{room\_id}/poll, передавая свои последние известные версии состояния видео и чата.

Если на сервере есть более новые данные (версии выше), сервер немедленно отвечает ими.

Если данные на сервере не новее, запрос "зависает" на сервере до 30 секунд. Сервер добавляет asyncio.Event для этого клиента в room["listeners"].

Если в течение этих 30 секунд происходит событие (обновление видео или новое сообщение), соответствующий event.set() вызывается, long poll запрос "просыпается" и немедленно отвечает клиенту новыми данными.

Если за 30 секунд событий не произошло (таймаут), сервер отвечает текущим (неизмененным) состоянием.

Клиент, получив ответ (или по таймауту), немедленно делает новый long poll запрос.  
Этот цикл обеспечивает квази-реальное время доставки обновлений.

**3.4** **Управление жизненным циклом комнат**

Для предотвращения накопления неактивных комнат в памяти сервера, реализована фоновая задача.

**# manager.py**

**async def cleanup\_inactive\_rooms():**

**while True:**

**try:**

**await asyncio.sleep(300) # Пауза 5 минут**

**current\_time = time.time()**

**rooms\_to\_delete = []**

**for room\_id, room\_data in list(rooms.items()): # list() для безопасного удаления во время итерации**

**# Если в комнате нет активных пользователей (проверяется по users)**

**# и/или комната неактивна определенное время**

**if not room\_data.get("users"):**

**# Можно добавить дополнительную проверку на время последнего обновления состояния,**

**# но в текущей логике, если нет пользователей, то и обновлений быть не должно.**

**# Например, если комната пуста более 10 минут.**

**# Для этого нужно хранить `last\_activity\_timestamp` для комнаты.**

**# В текущей реализации, если users пуст, комната считается кандидатом на удаление.**

**# Более строгая проверка:**

**# last\_activity = room\_data.get("last\_activity\_time", 0)**

**# if not room\_data.get("users") and (current\_time - last\_activity > 600):**

**rooms\_to\_delete.append(room\_id)**

**else:**

**# Внутри `poll\_state` уже есть очистка неактивных пользователей.**

**# Здесь можно добавить дополнительную, более агрессивную очистку,**

**# если `poll\_state` по какой-то причине не справляется,**

**# или для пользователей, которые давно не делали poll-запросы.**

**async with room\_locks.get(room\_id, asyncio.Lock()): # Получаем блокировку или создаем временную, если ее нет**

**if room\_id in rooms: # Проверяем, что комната все еще существует**

**room\_users = rooms[room\_id].get("users", {})**

**inactive\_users = [**

**uid for uid, user\_info in room\_users.items()**

**if current\_time - user\_info.get("last\_seen", 0) > 600 # 10 минут неактивности**

**]**

**for uid in inactive\_users:**

**if uid in room\_users:**

**del room\_users[uid]**

**for room\_id in rooms\_to\_delete:**

**if room\_id in rooms and not rooms[room\_id].get("users"): # Повторная проверка, что комната все еще пуста**

**print(f"Удаляем неактивную комнату: {room\_id}")**

**# Уведомить оставшихся слушателей (хотя их не должно быть, если нет юзеров)**

**if room\_id in room\_locks:**

**async with room\_locks[room\_id]:**

**for event in rooms.get(room\_id, {}).get("listeners", {}).values():**

**event.set()**

**if room\_id in rooms: del rooms[room\_id]**

**if room\_id in room\_locks: del room\_locks[room\_id]**

**except Exception as e:**

**print(f"Ошибка в cleanup\_inactive\_rooms: {e}")**

**@app.on\_event("startup")**

**async def startup\_event():**

**asyncio.create\_task(cleanup\_inactive\_rooms()) # Запуск задачи при старте приложения**

Эта асинхронная функция периодически (каждые 5 минут) проверяет все существующие комнаты. Если комната пуста (нет активных пользователей) в течение определенного времени (в текущей реализации упрощено: если users пусто, комната кандидат на удаление), она удаляется из rooms и room\_locks. Также предусмотрена очистка отдельных неактивных пользователей внутри комнат, если они долго не проявляли активность. Задача запускается при старте FastAPI приложения.

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В процессе тестирования приложения для синхронизированного просмотра видео «CineSync» были выявлены и устранены различные ошибки и недочёты, касающиеся как клиентской (JavaScript в room.html), так и серверной части (Python в manager.py).

Одной из основных проблем, обнаруженных на раннем этапе тестирования, была рассинхронизация состояния видеоплеера при одновременных действиях нескольких пользователей или при нестабильном интернет-соединении. Это могло приводить к тому, что разные пользователи видели видео на разных временных отметках или с разным состоянием воспроизведения/паузы. Для решения этой проблемы был усовершенствован механизм long polling и обработка версий состояния:

Клиентская логика (room.html) была настроена так, чтобы при получении нового состояния от сервера (data.version > currentVersion), она принудительно применяла серверное состояние (video.currentTime = data.state.time, video.play() или video.pause()), подавляя на короткое время отправку собственных обновлений (suppress = true).

Блокировки asyncio.Lock на сервере (async with room\_locks[room\_id]:) обеспечили атомарность операций обновления состояния комнаты, предотвращая гонки данных при одновременных запросах.

Другой важный аспект касался корректной работы чата:

Проблема: Сообщения могли дублироваться или приходить не в том порядке.

Решение: Каждому сообщению на сервере присваивается уникальный last\_message\_id и инкрементируется message\_version. Клиент отслеживает lastMessageTimestamp и displayedMessages (Set из ID сообщений), чтобы отображать только действительно новые и уникальные сообщения. При первоначальной загрузке запрашивается история последних сообщений.

Проблема: Длинные сообщения могли ломать верстку чата.

Решение: На клиенте (maxlength="500" для chatInput) и сервере (проверка len(message\_text) > 500) было добавлено ограничение на длину сообщения.

В процессе тестирования приложения «CineSync» была проведена проверка основных функций и поведения системы при различных пользовательских сценариях:  
– **Создание и присоединение к комнатам:** Проверена корректность генерации ссылок, отображение выбранного видео, работа модального окна для ввода имени.  
– **Синхронизация видеоплеера:** Тестировались сценарии одновременного нажатия play/pause несколькими пользователями, перемотка, восстановление синхронизации после временной потери соединения одним из клиентов.  
– **Работа чата:** Проверена отправка и получение сообщений, отображение имени отправителя, корректное отображение времени, автоматическая прокрутка к новым сообщениям, загрузка истории.  
– **Отображение количества онлайн-пользователей:** Проверено, что счетчик корректно обновляется при входе/выходе пользователей и при очистке неактивных.  
– **Работа long polling:** Отслеживалась нагрузка на сервер, корректность таймаутов и немедленных ответов при наличии обновлений.  
– **Автоматическая очистка:** Проверена работа фоновой задачи cleanup\_inactive\_rooms по удалению пустых комнат и неактивных пользователей.  
– **Интерфейс пользователя:** Проверена отзывчивость интерфейса, работа кнопок (копирование ссылки, отправка сообщения), отображение статуса подключения.

Большинство проблем возникало из-за особенностей асинхронной обработки и необходимости точной синхронизации состояния между несколькими клиентами. Они были успешно устранены в процессе итеративного тестирования (ручного и логического анализа кода). Финальная версия программного средства обеспечивает устойчивую работу, корректную синхронизацию и обработку основных пользовательских сценариев.

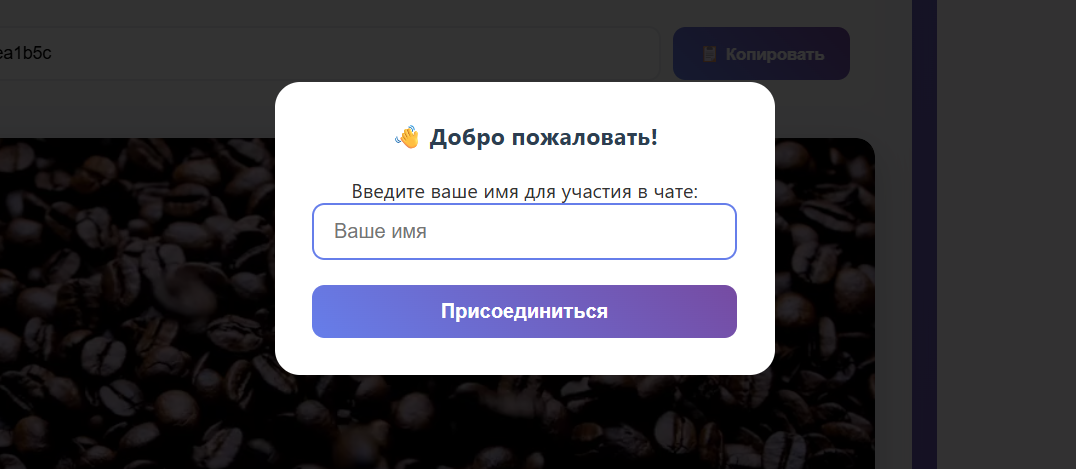
# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 5.1 Интерфейс программного средства

5.1.1 **Окно регистрации и авторизации**

При первом запуске веб-приложения «CineSync» пользователю отображается главная страница (index.html).

* **Заголовок:** "🎬 CineSync" и слоган "Смотрите видео синхронно с друзьями в реальном времени".
* **Блок "Возможности":** Краткое описание ключевых функций (Мгновенная синхронизация, Живой чат, Легкий доступ).
* **Список видео для просмотра:** Отображаются карточки доступных видеофайлов. Каждая карточка содержит:
  + Иконку и название видео.
  + Краткое описание.
  + Кнопку "Создать комнату".  
    При нажатии на кнопку "Создать комнату" для выбранного видео, пользователь будет перенаправлен в новую комнату просмотра. Внешний вид главной страницы показан на Рисунке 5.1.

****

**Рисунок 2.1 – Ввод имени**

## 5.2 Главное окно пользователя

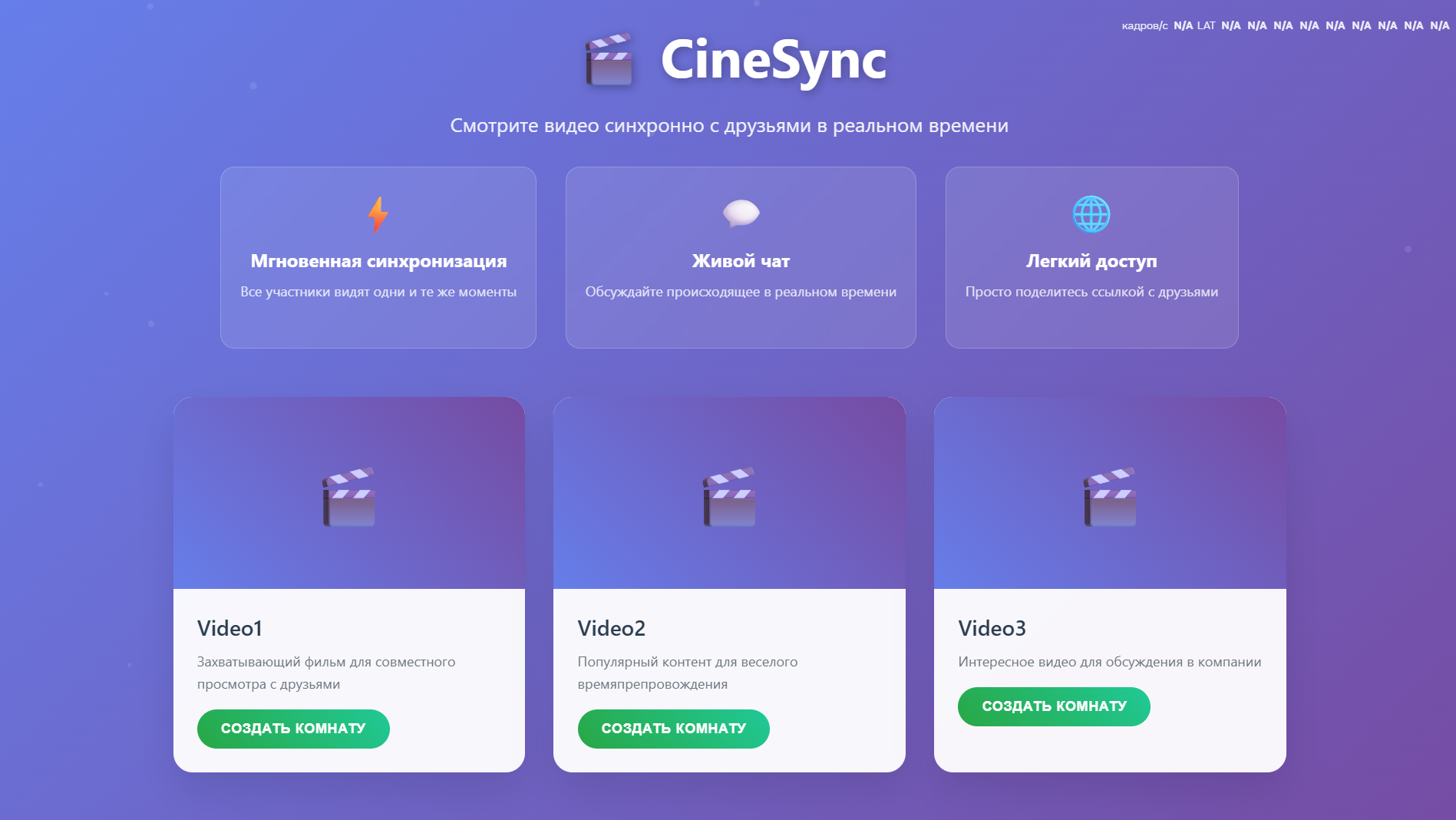
После создания комнаты или перехода по прямой ссылке на комнату, пользователь попадает в основное окно приложения.

5.2.1 **Структура интерфейса**

Перед тем как присоединиться к просмотру и чату, появится модальное окно (Рисунок 5.2), где необходимо ввести свое имя. Это имя будет отображаться в чате.

Поле для ввода "Ваше имя".

Кнопка "Присоединиться".

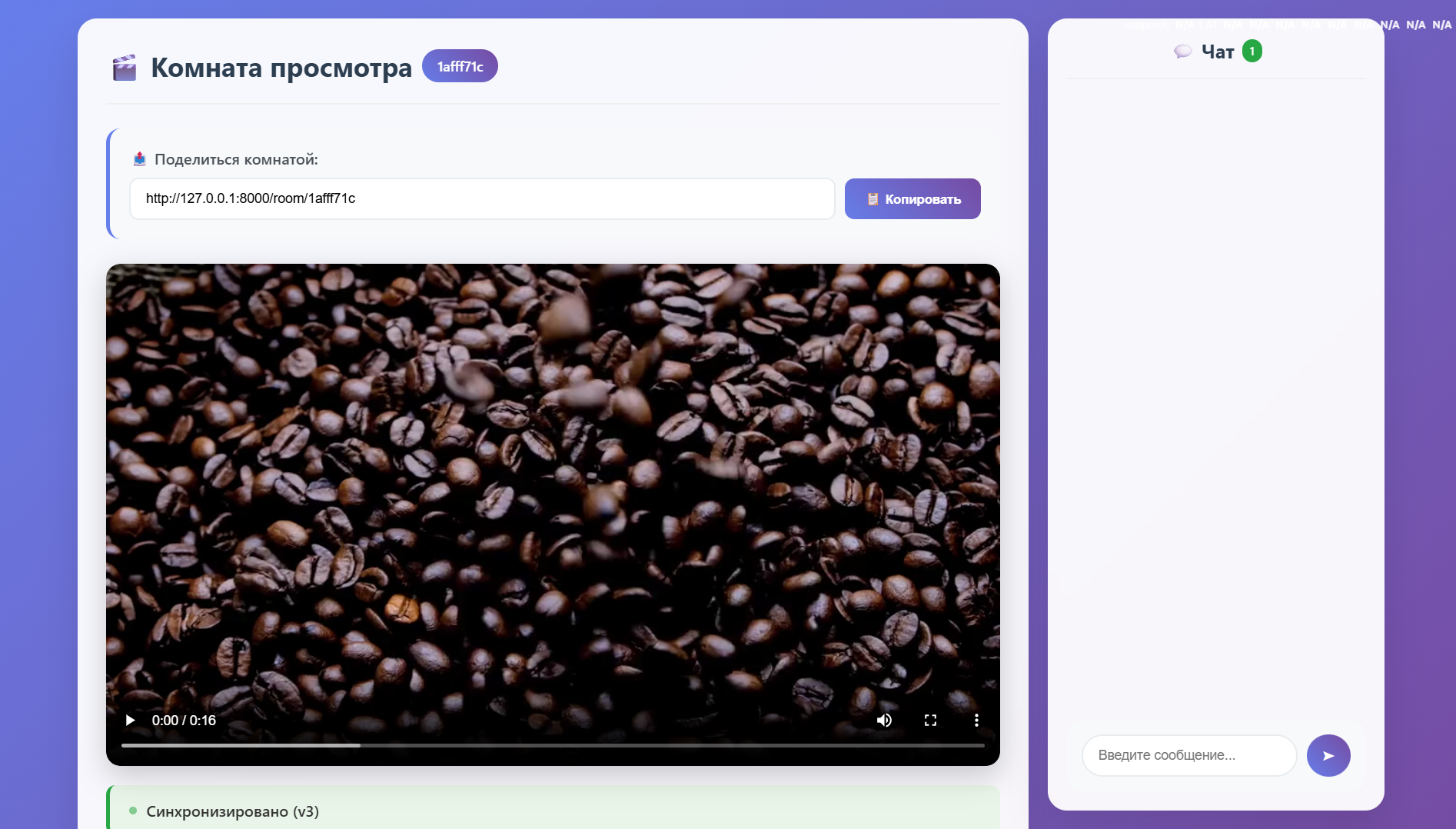
****

**Рисунок 2.2 – Главное окно приложения**

## 5.2.2 Управление папками

После ввода имени, пользователь видит основной интерфейс (Рисунок 5.3), который состоит из следующих элементов:

* **Область видеоплеера (слева):**
  + **Заголовок:** "🎬 Комната просмотра" и уникальный ID комнаты.
  + **Секция "Поделиться комнатой":** Содержит URL текущей комнаты и кнопку "📋 Копировать" для копирования ссылки.
  + **Видеоплеер:** Стандартный HTML5 видеоплеер с элементами управления (воспроизведение/пауза, громкость, перемотка, полноэкранный режим). Действия с плеером синхронизируются со всеми участниками.
  + **Статус синхронизации:** Отображает текущий статус подключения и синхронизации (например, "Синхронизировано (vX)").
* **Панель чата (справа):**
  + **Заголовок чата:** "💬 Чат" и счетчик активных пользователей в комнате (<span id="onlineCount"></span>).
  + **Окно сообщений (chatMessages):** Здесь отображаются сообщения от всех участников, включая имя отправителя (если это не вы), текст сообщения и время отправки. Ваши сообщения выделяются другим цветом и выравниваются по правому краю.
  + **Поле ввода сообщения (chatInput):** Текстовое поле для набора сообщения.
  + **Кнопка отправки сообщения:** Кнопка с иконкой "➤" для отправки введенного сообщения.



**Рисунок 2.3 – Окно комнаты просмотра**

5.3 Управление файлами

5.3.2 **Удаление**

**Воспроизведение/Пауза:** Любой участник комнаты может нажать кнопку play/pause на видеоплеере. Это действие будет синхронизировано, и видео начнет/прекратит воспроизведение у всех остальных участников.

**Перемотка:** Перемотка видео с помощью ползунка также синхронизируется. Видео у всех участников переключится на выбранный момент времени.

**Громкость и полноэкранный режим:** Эти настройки плеера являются локальными и не синхронизируются.

**Рисунок 5.4 – Сообщение об удалении папки**

5.4.1 **Загрузка**

**Отправка сообщения:** Введите текст вашего сообщения в поле "Введите сообщение..." в нижней части панели чата и нажмите кнопку "➤" или клавишу Enter. Ваше сообщение появится в окне чата.

**Чтение сообщений:** Новые сообщения от других участников будут автоматически появляться в окне чата. Окно прокручивается вниз по мере поступления новых сообщений.

**Идентификация:** Ваше имя, введенное при входе в комнату, будет отображаться рядом с вашими сообщениями (для других участников).

5.4.2 **Скачивание**

Чтобы пригласить друзей в комнату:

Найдите секцию "Поделиться комнатой" над видеоплеером.

Нажмите кнопку "📋 Копировать". Ссылка на текущую комнату будет скопирована в буфер обмена.

Отправьте эту ссылку своим друзьям. Перейдя по ней, они попадут в вашу комнату просмотра.

5.4.2 **Удаление**

Для выхода из комнаты просто закройте вкладку браузера. Ваша сессия в комнате завершится, и через некоторое время (если вы были последним активным пользователем) комната может быть автоматически удалена с сервера.

# ЗАКЛЮЧЕНИЕ

В современном цифровом мире потребность в совместном потреблении контента и разделении эмоций в реальном времени неуклонно растет. Одним из эффективных решений данной задачи являются платформы для синхронизированного просмотра видео, позволяющие пользователям вместе смотреть фильмы, сериалы или любые другие видеоролики, находясь в разных местах. В рамках данного проекта было разработано веб-приложение «CineSync», реализующее ключевые функции такого сервиса.

В процессе разработки были успешно достигнуты все поставленные цели и выполнены основные задачи. Пользователям доступен удобный веб-интерфейс для выбора видео и создания комнат просмотра. Внутри комнаты обеспечивается синхронизация состояния видеоплеера (воспроизведение, пауза, текущее время) между всеми участниками, а также возможность общения в реальном времени через интегрированный текстовый чат. Особое внимание было уделено механизму long polling для обеспечения эффективной и быстрой доставки обновлений состояния без необходимости использования WebSocket.

«CineSync» позволяет пользователям указывать свое имя для идентификации в чате, отображает количество активных участников и предоставляет простую возможность поделиться ссылкой на комнату. Система также включает фоновую задачу для автоматической очистки неактивных пользователей и пустых комнат, что способствует эффективному использованию ресурсов сервера. Для управления конкурентным доступом к состоянию комнат используются асинхронные блокировки.

Разработанное приложение обладает простой архитектурой и может быть легко развернуто. Оно демонстрирует практическое применение современных веб-технологий, таких как FastAPI, asyncio для асинхронного программирования, Jinja2 для шаблонизации, и клиентского JavaScript для интерактивности. В дальнейшем возможно расширение функциональности: добавление поддержки большего числа источников видео (например, по URL с популярных хостингов), создание приватных комнат с паролями, расширенные настройки чата или интеграция видеозвонков.

Таким образом, проект успешно решает задачу построения функционального и интерактивного приложения для синхронизированного просмотра видео, предоставляя пользователям простой и удобный инструмент для совместного времяпрепровождения онлайн.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Sebastián Ramírez. FastAPI Documentation [Электронный ресурс]. – Режим доступа: [https://fastapi.tiangolo.com/](https://www.google.com/url?sa=E&q=https%3A%2F%2Ffastapi.tiangolo.com%2F) (Основной фреймворк)  
[2] Python asyncio — Asynchronous I/O. Python Documentation [Электронный ресурс]. – Режим доступа: [https://docs.python.org/3/library/asyncio.html](https://www.google.com/url?sa=E&q=https%3A%2F%2Fdocs.python.org%2F3%2Flibrary%2Fasyncio.html) (Для асинхронных операций и long polling)  
[3] Jinja2 Documentation [Электронный ресурс]. – Режим доступа: [https://jinja.palletsprojects.com/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fjinja.palletsprojects.com%2F) (Шаблонизатор HTML)  
[4] MDN Web Docs. HTML, CSS, JavaScript [Электронный ресурс]. – Режим доступа: [https://developer.mozilla.org/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fdeveloper.mozilla.org%2F) (Для клиентской части)  
[5] Uvicorn Documentation [Электронный ресурс]. – Режим доступа: [https://www.uvicorn.org/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fwww.uvicorn.org%2F) (ASGI сервер для FastAPI)  
[6] Long Polling. Wikipedia [Электронный ресурс]. – Режим доступа: [https://en.wikipedia.org/wiki/Push\_technology#Long\_polling](https://www.google.com/url?sa=E&q=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FPush_technology%23Long_polling) (Описание техники long polling)

# ПРИЛОЖЕНИЕ А. Исходный код программы

    manager.py

**importuuid**

**importasyncio**

**importtime**

**fromdatetime importdatetime**

**fromfastapi importFastAPI, Request, HTTPException**

**fromfastapi.responses importRedirectResponse, JSONResponse**

**fromfastapi.staticfiles importStaticFiles**

**fromfastapi.templating importJinja2Templates**

**fromtyping importDict, List, Any**

**app =FastAPI()**

**app.mount("/static", StaticFiles(directory="static"), name="static")**

**templates =Jinja2Templates(directory="templates")**

**# Хранилище данных**

**rooms: Dict[str, Dict[str, Any]] ={}**

**room\_locks: Dict[str, asyncio.Lock] ={}**

**classChatMessage:**

**def\_\_init\_\_(self, user\_id: str, username: str, message: str):**

**self.user\_id =user\_id**

**self.username =username**

**self.message =message**

**self.timestamp =datetime.now().isoformat()**

**defto\_dict(self):**

**return{**

**"user\_id": self.user\_id,**

**"username": self.username,**

**"message": self.message,**

**"timestamp": self.timestamp**

**}**

**@app.get("/")**

**defindex(request: Request):**

**videos =["video1.mp4", "video2.mp4", "video3.mp4"]**

**returntemplates.TemplateResponse("index.html", {"request": request, "videos": videos})**

**@app.get("/create\_room/{video\_name}")**

**defcreate\_room(video\_name: str):**

**room\_id =str(uuid.uuid4())[:8]**

**rooms[room\_id] ={**

**"video": video\_name,**

**"state": {"playing": False, "time": 0.0, "version": 0},**

**"listeners": {},**

**"users": {}, # Активные пользователи**

**"messages": [], # История сообщений**

**"last\_message\_id": 0,**

**"message\_version": 0 # Отдельная версия для сообщений**

**}**

**room\_locks[room\_id] =asyncio.Lock()**

**returnRedirectResponse(f"/room/{room\_id}")**

**@app.get("/room/{room\_id}")**

**defget\_room(request: Request, room\_id: str):**

**ifroom\_id notinrooms:**

**returnRedirectResponse("/")**

**returntemplates.TemplateResponse("room.html", {**

**"request": request,**

**"room\_id": room\_id,**

**"video\_name": rooms[room\_id]["video"]**

**})**

**@app.post("/room/{room\_id}/update")**

**asyncdefupdate\_state(room\_id: str, request: Request):**

**ifroom\_id notinrooms:**

**raiseHTTPException(status\_code=404, detail="Комната не найдена")**

**try:**

**data =awaitrequest.json()**

**playing =data.get("playing")**

**time\_pos =data.get("time")**

**user\_id =data.get("user\_id")**

**username =data.get("username")**

**ifplaying isNoneortime\_pos isNone:**

**raiseValueError("Неверные данные")**

**exceptExceptionase:**

**raiseHTTPException(status\_code=400, detail="Неверный формат данных")**

**asyncwithroom\_locks[room\_id]:**

**room =rooms[room\_id]**

**# Обновляем информацию о пользователе**

**ifuser\_id andusername:**

**room["users"][user\_id] ={**

**"username": username,**

**"last\_seen": time.time()**

**}**

**# Обновляем состояние видео**

**room["state"] ={**

**"playing": playing,**

**"time": time\_pos,**

**"version": room["state"]["version"] +1**

**}**

**# Уведомляем всех слушателей**

**forevent inroom["listeners"].values():**

**event.set()**

**room["listeners"] ={}**

**returnJSONResponse({"status": "success", "version": room["state"]["version"]})**

**@app.post("/room/{room\_id}/chat")**

**asyncdefsend\_chat\_message(room\_id: str, request: Request):**

**ifroom\_id notinrooms:**

**raiseHTTPException(status\_code=404, detail="Комната не найдена")**

**try:**

**data =awaitrequest.json()**

**message\_text =data.get("message", "").strip()**

**user\_id =data.get("user\_id")**

**username =data.get("username")**

**ifnotmessage\_text ornotuser\_id ornotusername:**

**raiseValueError("Неверные данные")**

**# Ограничение длины сообщения**

**iflen(message\_text) >500:**

**raiseValueError("Сообщение слишком длинное")**

**exceptExceptionase:**

**raiseHTTPException(status\_code=400, detail="Неверный формат данных")**

**asyncwithroom\_locks[room\_id]:**

**room =rooms[room\_id]**

**# Создаем новое сообщение с уникальным ID**

**room["last\_message\_id"] +=1**

**message =ChatMessage(user\_id, username, message\_text)**

**message\_dict =message.to\_dict()**

**message\_dict["id"] =room["last\_message\_id"] # Добавляем уникальный ID**

**room["messages"].append(message\_dict)**

**room["message\_version"] +=1 # Увеличиваем версию сообщений**

**# Ограничиваем количество сообщений в истории**

**iflen(room["messages"]) >100:**

**room["messages"] =room["messages"][-100:]**

**# Обновляем информацию о пользователе**

**room["users"][user\_id] ={**

**"username": username,**

**"last\_seen": time.time()**

**}**

**# Уведомляем всех слушателей о новом сообщении**

**forevent inroom["listeners"].values():**

**event.set()**

**room["listeners"] ={}**

**returnJSONResponse({"status": "success", "message\_id": room["last\_message\_id"]})**

**@app.get("/room/{room\_id}/poll")**

**asyncdefpoll\_state(room\_id: str, last\_version: int=0, last\_message\_version: int=0, user\_id: str=None, username: str=None):**

**ifroom\_id notinrooms:**

**raiseHTTPException(status\_code=404, detail="Комната не найдена")**

**client\_id =str(uuid.uuid4())**

**room =rooms[room\_id]**

**# Регистрируем пользователя**

**ifuser\_id andusername:**

**asyncwithroom\_locks[room\_id]:**

**room["users"][user\_id] ={**

**"username": username,**

**"last\_seen": time.time()**

**}**

**# Очистка неактивных пользователей (не активны более 30 сек)**

**current\_time =time.time()**

**asyncwithroom\_locks[room\_id]:**

**inactive\_users =[**

**uid foruid, user\_info inroom["users"].items()**

**ifcurrent\_time -user\_info["last\_seen"] >30**

**]**

**foruid ininactive\_users:**

**delroom["users"][uid]**

**# Получаем текущее состояние**

**current\_state ={**

**"state": room["state"],**

**"version": room["state"]["version"],**

**"message\_version": room["message\_version"],**

**"online\_count": len(room["users"]),**

**"messages": []**

**}**

**# Обрабатываем сообщения**

**iflast\_message\_version ==0:**

**# Первое подключение - отправляем последние 10 сообщений**

**current\_state["messages"] =room["messages"][-10:] ifroom["messages"] else[]**

**elifroom["message\_version"] >last\_message\_version:**

**# Есть новые сообщения - отправляем только новые**

**# Находим новые сообщения, сравнивая по ID**

**ifroom["messages"]:**

**# Простой способ: отправляем последние несколько сообщений**

**# В реальном приложении лучше отслеживать по timestamp или ID**

**current\_state["messages"] =room["messages"][-3:] iflen(room["messages"]) >=3elseroom["messages"]**

**# Проверяем, нужно ли ждать обновлений**

**need\_to\_wait =(last\_version >=room["state"]["version"] and**

**last\_message\_version >=room["message\_version"])**

**ifneed\_to\_wait:**

**event =asyncio.Event()**

**asyncwithroom\_locks[room\_id]:**

**room["listeners"][client\_id] =event**

**try:**

**awaitasyncio.wait\_for(event.wait(), timeout=30.0)**

**exceptasyncio.TimeoutError:**

**pass**

**finally:**

**asyncwithroom\_locks[room\_id]:**

**ifclient\_id inroom["listeners"]:**

**delroom["listeners"][client\_id]**

**# Получаем обновленное состояние после ожидания**

**asyncwithroom\_locks[room\_id]:**

**# Снова очищаем неактивных пользователей**

**current\_time =time.time()**

**inactive\_users =[**

**uid foruid, user\_info inroom["users"].items()**

**ifcurrent\_time -user\_info["last\_seen"] >30**

**]**

**foruid ininactive\_users:**

**delroom["users"][uid]**

**# Формируем финальный ответ**

**current\_state ={**

**"state": room["state"],**

**"version": room["state"]["version"],**

**"message\_version": room["message\_version"],**

**"online\_count": len(room["users"]),**

**"messages": []**

**}**

**# Отправляем новые сообщения только если версия изменилась**

**ifroom["message\_version"] >last\_message\_version:**

**current\_state["messages"] =room["messages"][-3:] iflen(room["messages"]) >=3elseroom["messages"]**

**returnJSONResponse(current\_state)**

**@app.get("/room/{room\_id}/messages")**

**asyncdefget\_messages(room\_id: str, limit: int=50, after\_id: int=0):**

**"""Получить историю сообщений"""**

**ifroom\_id notinrooms:**

**raiseHTTPException(status\_code=404, detail="Комната не найдена")**

**room =rooms[room\_id]**

**# Фильтруем сообщения по ID, если указан after\_id**

**ifafter\_id >0:**

**messages =[msg formsg inroom["messages"] ifmsg.get("id", 0) >after\_id]**

**else:**

**messages =room["messages"][-limit:] ifroom["messages"] else[]**

**returnJSONResponse({**

**"messages": messages,**

**"total": len(room["messages"]),**

**"message\_version": room["message\_version"]**

**})**

**@app.delete("/room/{room\_id}")**

**asyncdefdelete\_room(room\_id: str):**

**"""Удалить комнату (для администрирования)"""**

**ifroom\_id inrooms:**

**asyncwithroom\_locks[room\_id]:**

**# Уведомляем всех слушателей о закрытии комнаты**

**forevent inrooms[room\_id]["listeners"].values():**

**event.set()**

**delrooms[room\_id]**

**delroom\_locks[room\_id]**

**returnJSONResponse({"status": "success", "message": "Комната удалена"})**

**raiseHTTPException(status\_code=404, detail="Комната не найдена")**

**@app.get("/room/{room\_id}/stats")**

**asyncdefget\_room\_stats(room\_id: str):**

**"""Получить статистику комнаты"""**

**ifroom\_id notinrooms:**

**raiseHTTPException(status\_code=404, detail="Комната не найдена")**

**room =rooms[room\_id]**

**# Очистка неактивных пользователей**

**current\_time =time.time()**

**asyncwithroom\_locks[room\_id]:**

**inactive\_users =[**

**uid foruid, user\_info inroom["users"].items()**

**ifcurrent\_time -user\_info["last\_seen"] >30**

**]**

**foruid ininactive\_users:**

**delroom["users"][uid]**

**returnJSONResponse({**

**"room\_id": room\_id,**

**"video": room["video"],**

**"online\_users": len(room["users"]),**

**"total\_messages": len(room["messages"]),**

**"video\_state": room["state"],**

**"message\_version": room["message\_version"],**

**"users": [**

**{**

**"username": user\_info["username"],**

**"last\_seen": user\_info["last\_seen"]**

**}**

**foruser\_info inroom["users"].values()**

**]**

**})**

**# Фоновая задача для очистки неактивных комнат**

**asyncdefcleanup\_inactive\_rooms():**

**"""Очистка неактивных комнат каждые 5 минут"""**

**whileTrue:**

**try:**

**awaitasyncio.sleep(300) # 5 минут**

**current\_time =time.time()**

**rooms\_to\_delete =[]**

**forroom\_id, room inrooms.items():**

**# Если в комнате нет активных пользователей более 10 минут**

**ifnotroom["users"]:**

**# Проверяем время последнего обновления состояния**

**# Если комната не использовалась более 10 минут, помечаем для удаления**

**rooms\_to\_delete.append(room\_id)**

**else:**

**# Очищаем неактивных пользователей**

**asyncwithroom\_locks[room\_id]:**

**inactive\_users =[**

**uid foruid, user\_info inroom["users"].items()**

**ifcurrent\_time -user\_info["last\_seen"] >60**

**]**

**foruid ininactive\_users:**

**delroom["users"][uid]**

**# Удаляем неактивные комнаты**

**forroom\_id inrooms\_to\_delete:**

**ifroom\_id inrooms andnotrooms[room\_id]["users"]:**

**print(f"Удаляем неактивную комнату: {room\_id}")**

**delrooms[room\_id]**

**ifroom\_id inroom\_locks:**

**delroom\_locks[room\_id]**

**exceptExceptionase:**

**print(f"Ошибка в cleanup\_inactive\_rooms: {e}")**

**# Запуск фоновой задачи при старте приложения**

**@app.on\_event("startup")**

**asyncdefstartup\_event():**

**asyncio.create\_task(cleanup\_inactive\_rooms())**

**@app.on\_event("shutdown")**

**asyncdefshutdown\_event():**

**print("Завершение работы сервера...")**

**if\_\_name\_\_=="\_\_main\_\_":**

**importuvicorn**

**uvicorn.run(app, host="127.0.0.1", port=8000, reload=True)**

**run.py**

**#!/usr/bin/env python3**

**"""**

**Скрипт для запуска CineSync - приложения синхронного просмотра видео**

**"""**

**importuvicorn**

**importos**

**importsys**

**defmain():**

**print("🎬 Запуск CineSync...")**

**print("="\*50)**

**# Проверяем, существуют ли необходимые директории**

**required\_dirs=['static', 'templates']**

**fordir\_nameinrequired\_dirs:**

**ifnotos.path.exists(dir\_name):**

**print(f"❌ Директория '{dir\_name}' не найдена!")**

**print(f"Создайте директорию '{dir\_name}' и поместите в неё соответствующие файлы.")**

**return**

**# Проверяем наличие видеофайлов**

**static\_files=os.listdir('static') ifos.path.exists('static') else[]**

**video\_files=[fforfinstatic\_filesiff.endswith('.mp4')]**

**ifnotvideo\_files:**

**print("⚠️ В директории 'static' не найдены видеофайлы (.mp4)")**

**print("Поместите ваши видеофайлы в директорию 'static' для тестирования.")**

**else:**

**print(f"✅ Найдено видеофайлов: {len(video\_files)}")**

**forvideoinvideo\_files:**

**print(f" - {video}")**

**print("\n🚀 Запуск сервера...")**

**print("📍 Откройте браузер и перейдите по адресу: http://127.0.0.1:8000")**

**print("⏹️ Для остановки сервера нажмите Ctrl+C")**

**print("="\*50)**

**try:**

**uvicorn.run(**

**"manager:app",**

**host="127.0.0.1",**

**port=8000,**

**reload=True,**

**log\_level="info"**

**)**

**exceptKeyboardInterrupt:**

**print("\n👋 Сервер остановлен. До свидания!")**

**exceptExceptionase:**

**print(f"\n❌ Ошибка запуска сервера: {e}")**

**sys.exit(1)**

**if\_\_name\_\_=="\_\_main\_\_":**

**main()**

**templates\index.html**

**<!DOCTYPEhtml>**

**<html>**

**<head>**

**<title>Синхронизированный просмотр видео</title>**

**<metacharset="UTF-8">**

**<metaname="viewport"content="width=device-width, initial-scale=1.0">**

**<style>**

**\*{**

**margin: 0;**

**padding: 0;**

**box-sizing: border-box;**

**}**

**body{**

**font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;**

**background: linear-gradient(135deg, #667eea0%, #764ba2100%);**

**min-height: 100vh;**

**display: flex;**

**align-items: center;**

**justify-content: center;**

**color: #333;**

**}**

**.container{**

**max-width: 1200px;**

**width: 100%;**

**padding: 20px;**

**}**

**.header{**

**text-align: center;**

**margin-bottom: 50px;**

**color: white;**

**}**

**.headerh1{**

**font-size: 3.5rem;**

**font-weight: 700;**

**margin-bottom: 20px;**

**text-shadow: 2px2px10pxrgba(0, 0, 0, 0.3);**

**}**

**.headerp{**

**font-size: 1.3rem;**

**opacity: 0.9;**

**margin-bottom: 30px;**

**}**

**.features{**

**display: flex;**

**justify-content: center;**

**gap: 30px;**

**margin-bottom: 50px;**

**flex-wrap: wrap;**

**}**

**.feature{**

**background: rgba(255, 255, 255, 0.1);**

**padding: 20px;**

**border-radius: 15px;**

**text-align: center;**

**color: white;**

**backdrop-filter: blur(10px);**

**border: 1pxsolidrgba(255, 255, 255, 0.2);**

**min-width: 200px;**

**}**

**.feature-icon{**

**font-size: 2.5rem;**

**margin-bottom: 10px;**

**}**

**.featureh3{**

**margin-bottom: 10px;**

**font-size: 1.2rem;**

**}**

**.featurep{**

**font-size: 0.9rem;**

**opacity: 0.8;**

**}**

**.video-grid{**

**display: grid;**

**grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));**

**gap: 30px;**

**margin-bottom: 50px;**

**}**

**.video-card{**

**background: rgba(255, 255, 255, 0.95);**

**border-radius: 20px;**

**overflow: hidden;**

**transition: all0.3sease;**

**cursor: pointer;**

**backdrop-filter: blur(10px);**

**box-shadow: 020px40pxrgba(0, 0, 0, 0.1);**

**text-decoration: none;**

**color: inherit;**

**}**

**.video-card:hover{**

**transform: translateY(-10px) scale(1.02);**

**box-shadow: 030px60pxrgba(0, 0, 0, 0.2);**

**}**

**.video-thumbnail{**

**width: 100%;**

**height: 200px;**

**background: linear-gradient(45deg, #667eea, #764ba2);**

**display: flex;**

**align-items: center;**

**justify-content: center;**

**font-size: 4rem;**

**color: white;**

**position: relative;**

**overflow: hidden;**

**}**

**.video-thumbnail::before{**

**content: '';**

**position: absolute;**

**top: 0;**

**left: 0;**

**right: 0;**

**bottom: 0;**

**background: rgba(0, 0, 0, 0.2);**

**opacity: 0;**

**transition: opacity 0.3sease;**

**}**

**.video-card:hover.video-thumbnail::before{**

**opacity: 1;**

**}**

**.play-button{**

**position: absolute;**

**background: rgba(255, 255, 255, 0.9);**

**color: #667eea;**

**border-radius: 50%;**

**width: 60px;**

**height: 60px;**

**display: flex;**

**align-items: center;**

**justify-content: center;**

**font-size: 1.5rem;**

**transform: scale(0);**

**transition: transform 0.3sease;**

**}**

**.video-card:hover.play-button{**

**transform: scale(1);**

**}**

**.video-info{**

**padding: 25px;**

**}**

**.video-title{**

**font-size: 1.4rem;**

**font-weight: 600;**

**margin-bottom: 10px;**

**color: #2c3e50;**

**}**

**.video-description{**

**color: #6c757d;**

**font-size: 0.9rem;**

**line-height: 1.6;**

**}**

**.create-room-btn{**

**background: linear-gradient(45deg, #28a745, #20c997);**

**color: white;**

**padding: 12px25px;**

**border: none;**

**border-radius: 25px;**

**font-weight: 600;**

**font-size: 0.9rem;**

**margin-top: 15px;**

**cursor: pointer;**

**transition: all0.3sease;**

**text-transform: uppercase;**

**letter-spacing: 0.5px;**

**}**

**.create-room-btn:hover{**

**transform: translateY(-2px);**

**box-shadow: 05px15pxrgba(40, 167, 69, 0.4);**

**}**

**.footer{**

**text-align: center;**

**color: rgba(255, 255, 255, 0.8);**

**margin-top: 50px;**

**}**

**.footerp{**

**margin-bottom: 20px;**

**}**

**.footer-links{**

**display: flex;**

**justify-content: center;**

**gap: 30px;**

**flex-wrap: wrap;**

**}**

**.footer-link{**

**color: rgba(255, 255, 255, 0.8);**

**text-decoration: none;**

**transition: color0.3sease;**

**}**

**.footer-link:hover{**

**color: white;**

**}**

**/\* Анимации \*/**

**@keyframesfadeInUp{**

**from {**

**opacity: 0;**

**transform: translateY(30px);**

**}**

**to {**

**opacity: 1;**

**transform: translateY(0);**

**}**

**}**

**.video-card{**

**animation: fadeInUp 0.6seaseforwards;**

**}**

**.video-card:nth-child(1) { animation-delay: 0.1s; }**

**.video-card:nth-child(2) { animation-delay: 0.2s; }**

**.video-card:nth-child(3) { animation-delay: 0.3s; }**

**.video-card:nth-child(4) { animation-delay: 0.4s; }**

**/\* Адаптивность \*/**

**@media(max-width: 768px) {**

**.headerh1{**

**font-size: 2.5rem;**

**}**

**.headerp{**

**font-size: 1.1rem;**

**}**

**.features{**

**flex-direction: column;**

**align-items: center;**

**}**

**.feature{**

**width: 100%;**

**max-width: 300px;**

**}**

**.video-grid{**

**grid-template-columns: 1fr;**

**}**

**.container{**

**padding: 15px;**

**}**

**}**

**@media(max-width: 480px) {**

**.headerh1{**

**font-size: 2rem;**

**}**

**.video-info{**

**padding: 20px;**

**}**

**}**

**/\* Particle animation background \*/**

**.particles{**

**position: fixed;**

**top: 0;**

**left: 0;**

**width: 100%;**

**height: 100%;**

**overflow: hidden;**

**z-index: -1;**

**}**

**.particle{**

**position: absolute;**

**background: rgba(255, 255, 255, 0.1);**

**border-radius: 50%;**

**animation: float 6sease-in-outinfinite;**

**}**

**@keyframesfloat{**

**0%, 100% { transform: translateY(0px) rotate(0deg); }**

**50% { transform: translateY(-20px) rotate(180deg); }**

**}**

**</style>**

**</head>**

**<body>**

**<divclass="particles"id="particles"></div>**

**<divclass="container">**

**<divclass="header">**

**<h1>🎬 CineSync</h1>**

**<p>Смотрите видео синхронно с друзьями в реальном времени</p>**

**<divclass="features">**

**<divclass="feature">**

**<divclass="feature-icon">⚡</div>**

**<h3>Мгновенная синхронизация</h3>**

**<p>Все участники видят одни и те же моменты</p>**

**</div>**

**<divclass="feature">**

**<divclass="feature-icon">💬</div>**

**<h3>Живой чат</h3>**

**<p>Обсуждайте происходящее в реальном времени</p>**

**</div>**

**<divclass="feature">**

**<divclass="feature-icon">🌐</div>**

**<h3>Легкий доступ</h3>**

**<p>Просто поделитесь ссылкой с друзьями</p>**

**</div>**

**</div>**

**</div>**

**<divclass="video-grid">**

**{% for video in videos %}**

**<ahref="/create\_room/{{ video }}"class="video-card">**

**<divclass="video-thumbnail">**

**🎬**

**<divclass="play-button">▶</div>**

**</div>**

**<divclass="video-info">**

**<h3class="video-title">{{ video.replace('.mp4', '').replace('\_', ' ').title() }}</h3>**

**<pclass="video-description">**

**{% if 'video1' in video %}**

**Захватывающий фильм для совместного просмотра с друзьями**

**{% elif 'video2' in video %}**

**Популярный контент для веселого времяпрепровождения**

**{% else %}**

**Интересное видео для обсуждения в компании**

**{% endif %}**

**</p>**

**<buttonclass="create-room-btn">Создать комнату</button>**

**</div>**

**</a>**

**{% endfor %}**

**</div>**

**<divclass="footer">**

**<p>CineSync - лучший способ смотреть видео вместе, находясь в разных местах</p>**

**<divclass="footer-links">**

**<ahref="#"class="footer-link">О проекте</a>**

**<ahref="#"class="footer-link">Как использовать</a>**

**<ahref="#"class="footer-link">Поддержка</a>**

**<ahref="#"class="footer-link">Контакты</a>**

**</div>**

**</div>**

**</div>**

**<script>**

**// Создание анимированных частиц на фоне**

**functioncreateParticles() {**

**constparticlesContainer= document.getElementById('particles');**

**constparticleCount= 20;**

**for(leti= 0; i< particleCount; i++) {**

**constparticle= document.createElement('div');**

**particle.className= 'particle';**

**// Случайные размеры и позиции**

**constsize= Math.random() \* 6+ 2;**

**particle.style.width= size+ 'px';**

**particle.style.height= size+ 'px';**

**particle.style.left= Math.random() \* 100+ '%';**

**particle.style.top= Math.random() \* 100+ '%';**

**// Случайная задержка анимации**

**particle.style.animationDelay= Math.random() \* 6+ 's';**

**particle.style.animationDuration= (Math.random() \* 4+ 4) + 's';**

**particlesContainer.appendChild(particle);**

**}**

**}**

**// Инициализация при загрузке страницы**

**window.addEventListener('load', () =>{**

**createParticles();**

**});**

**// Добавление интерактивности к карточкам**

**document.querySelectorAll('.video-card').forEach(card=>{**

**card.addEventListener('mouseenter', () =>{**

**card.style.zIndex= '10';**

**});**

**card.addEventListener('mouseleave', () =>{**

**card.style.zIndex= '1';**

**});**

**});**

**</script>**

**</body>**

**</html>**

**templates\room.html**

**<!DOCTYPEhtml>**

**<html>**

**<head>**

**<title>Синхронизированный просмотр видео</title>**

**<metacharset="UTF-8">**

**<metaname="viewport"content="width=device-width, initial-scale=1.0">**

**<style>**

**\*{**

**margin: 0;**

**padding: 0;**

**box-sizing: border-box;**

**}**

**body{**

**font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;**

**background: linear-gradient(135deg, #667eea0%, #764ba2100%);**

**min-height: 100vh;**

**display: flex;**

**align-items: center;**

**justify-content: center;**

**color: #333;**

**}**

**.container{**

**max-width: 1200px;**

**width: 100%;**

**padding: 20px;**

**}**

**.header{**

**text-align: center;**

**margin-bottom: 50px;**

**color: white;**

**}**

**.headerh1{**

**font-size: 3.5rem;**

**font-weight: 700;**

**margin-bottom: 20px;**

**text-shadow: 2px2px10pxrgba(0, 0, 0, 0.3);**

**}**

**.headerp{**

**font-size: 1.3rem;**

**opacity: 0.9;**

**margin-bottom: 30px;**

**}**

**.features{**

**display: flex;**

**justify-content: center;**

**gap: 30px;**

**margin-bottom: 50px;**

**flex-wrap: wrap;**

**}**

**.feature{**

**background: rgba(255, 255, 255, 0.1);**

**padding: 20px;**

**border-radius: 15px;**

**text-align: center;**

**color: white;**

**backdrop-filter: blur(10px);**

**border: 1pxsolidrgba(255, 255, 255, 0.2);**

**min-width: 200px;**

**}**

**.feature-icon{**

**font-size: 2.5rem;**

**margin-bottom: 10px;**

**}**

**.featureh3{**

**margin-bottom: 10px;**

**font-size: 1.2rem;**

**}**

**.featurep{**

**font-size: 0.9rem;**

**opacity: 0.8;**

**}**

**.video-grid{**

**display: grid;**

**grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));**

**gap: 30px;**

**margin-bottom: 50px;**

**}**

**.video-card{**

**background: rgba(255, 255, 255, 0.95);**

**border-radius: 20px;**

**overflow: hidden;**

**transition: all0.3sease;**

**cursor: pointer;**

**backdrop-filter: blur(10px);**

**box-shadow: 020px40pxrgba(0, 0, 0, 0.1);**

**text-decoration: none;**

**color: inherit;**

**}**

**.video-card:hover{**

**transform: translateY(-10px) scale(1.02);**

**box-shadow: 030px60pxrgba(0, 0, 0, 0.2);**

**}**

**.video-thumbnail{**

**width: 100%;**

**height: 200px;**

**background: linear-gradient(45deg, #667eea, #764ba2);**

**display: flex;**

**align-items: center;**

**justify-content: center;**

**font-size: 4rem;**

**color: white;**

**position: relative;**

**overflow: hidden;**

**}**

**.video-thumbnail::before{**

**content: '';**

**position: absolute;**

**top: 0;**

**left: 0;**

**right: 0;**

**bottom: 0;**

**background: rgba(0, 0, 0, 0.2);**

**opacity: 0;**

**transition: opacity 0.3sease;**

**}**

**.video-card:hover.video-thumbnail::before{**

**opacity: 1;**

**}**

**.play-button{**

**position: absolute;**

**background: rgba(255, 255, 255, 0.9);**

**color: #667eea;**

**border-radius: 50%;**

**width: 60px;**

**height: 60px;**

**display: flex;**

**align-items: center;**

**justify-content: center;**

**font-size: 1.5rem;**

**transform: scale(0);**

**transition: transform 0.3sease;**

**}**

**.video-card:hover.play-button{**

**transform: scale(1);**

**}**

**.video-info{**

**padding: 25px;**

**}**

**.video-title{**

**font-size: 1.4rem;**

**font-weight: 600;**

**margin-bottom: 10px;**

**color: #2c3e50;**

**}**

**.video-description{**

**color: #6c757d;**

**font-size: 0.9rem;**

**line-height: 1.6;**

**}**

**.create-room-btn{**

**background: linear-gradient(45deg, #28a745, #20c997);**

**color: white;**

**padding: 12px25px;**

**border: none;**

**border-radius: 25px;**

**font-weight: 600;**

**font-size: 0.9rem;**

**margin-top: 15px;**

**cursor: pointer;**

**transition: all0.3sease;**

**text-transform: uppercase;**

**letter-spacing: 0.5px;**

**}**

**.create-room-btn:hover{**

**transform: translateY(-2px);**

**box-shadow: 05px15pxrgba(40, 167, 69, 0.4);**

**}**

**.footer{**

**text-align: center;**

**color: rgba(255, 255, 255, 0.8);**

**margin-top: 50px;**

**}**

**.footerp{**

**margin-bottom: 20px;**

**}**

**.footer-links{**

**display: flex;**

**justify-content: center;**

**gap: 30px;**

**flex-wrap: wrap;**

**}**

**.footer-link{**

**color: rgba(255, 255, 255, 0.8);**

**text-decoration: none;**

**transition: color0.3sease;**

**}**

**.footer-link:hover{**

**color: white;**

**}**

**/\* Анимации \*/**

**@keyframesfadeInUp{**

**from {**

**opacity: 0;**

**transform: translateY(30px);**

**}**

**to {**

**opacity: 1;**

**transform: translateY(0);**

**}**

**}**

**.video-card{**

**animation: fadeInUp 0.6seaseforwards;**

**}**

**.video-card:nth-child(1) { animation-delay: 0.1s; }**

**.video-card:nth-child(2) { animation-delay: 0.2s; }**

**.video-card:nth-child(3) { animation-delay: 0.3s; }**

**.video-card:nth-child(4) { animation-delay: 0.4s; }**

**/\* Адаптивность \*/**

**@media(max-width: 768px) {**

**.headerh1{**

**font-size: 2.5rem;**

**}**

**.headerp{**

**font-size: 1.1rem;**

**}**

**.features{**

**flex-direction: column;**

**align-items: center;**

**}**

**.feature{**

**width: 100%;**

**max-width: 300px;**

**}**

**.video-grid{**

**grid-template-columns: 1fr;**

**}**

**.container{**

**padding: 15px;**

**}**

**}**

**@media(max-width: 480px) {**

**.headerh1{**

**font-size: 2rem;**

**}**

**.video-info{**

**padding: 20px;**

**}**

**}**

**/\* Particle animation background \*/**

**.particles{**

**position: fixed;**

**top: 0;**

**left: 0;**

**width: 100%;**

**height: 100%;**

**overflow: hidden;**

**z-index: -1;**

**}**

**.particle{**

**position: absolute;**

**background: rgba(255, 255, 255, 0.1);**

**border-radius: 50%;**

**animation: float 6sease-in-outinfinite;**

**}**

**@keyframesfloat{**

**0%, 100% { transform: translateY(0px) rotate(0deg); }**

**50% { transform: translateY(-20px) rotate(180deg); }**

**}**

**</style>**

**</head>**

**<body>**

**<divclass="particles"id="particles"></div>**

**<divclass="container">**

**<divclass="header">**

**<h1>🎬 CineSync</h1>**

**<p>Смотрите видео синхронно с друзьями в реальном времени</p>**

**<divclass="features">**

**<divclass="feature">**

**<divclass="feature-icon">⚡</div>**

**<h3>Мгновенная синхронизация</h3>**

**<p>Все участники видят одни и те же моменты</p>**

**</div>**

**<divclass="feature">**

**<divclass="feature-icon">💬</div>**

**<h3>Живой чат</h3>**

**<p>Обсуждайте происходящее в реальном времени</p>**

**</div>**

**<divclass="feature">**

**<divclass="feature-icon">🌐</div>**

**<h3>Легкий доступ</h3>**

**<p>Просто поделитесь ссылкой с друзьями</p>**

**</div>**

**</div>**

**</div>**

**<divclass="video-grid">**

**{% for video in videos %}**

**<ahref="/create\_room/{{ video }}"class="video-card">**

**<divclass="video-thumbnail">**

**🎬**

**<divclass="play-button">▶</div>**

**</div>**

**<divclass="video-info">**

**<h3class="video-title">{{ video.replace('.mp4', '').replace('\_', ' ').title() }}</h3>**

**<pclass="video-description">**

**{% if 'video1' in video %}**

**Захватывающий фильм для совместного просмотра с друзьями**

**{% elif 'video2' in video %}**

**Популярный контент для веселого времяпрепровождения**

**{% else %}**

**Интересное видео для обсуждения в компании**

**{% endif %}**

**</p>**

**<buttonclass="create-room-btn">Создать комнату</button>**

**</div>**

**</a>**

**{% endfor %}**

**</div>**

**<divclass="footer">**

**<p>CineSync - лучший способ смотреть видео вместе, находясь в разных местах</p>**

**<divclass="footer-links">**

**<ahref="#"class="footer-link">О проекте</a>**

**<ahref="#"class="footer-link">Как использовать</a>**

**<ahref="#"class="footer-link">Поддержка</a>**

**<ahref="#"class="footer-link">Контакты</a>**

**</div>**

**</div>**

**</div>**

**<script>**

**// Создание анимированных частиц на фоне**

**functioncreateParticles() {**

**constparticlesContainer= document.getElementById('particles');**

**constparticleCount= 20;**

**for(leti= 0; i< particleCount; i++) {**

**constparticle= document.createElement('div');**

**particle.className= 'particle';**

**// Случайные размеры и позиции**

**constsize= Math.random() \* 6+ 2;**

**particle.style.width= size+ 'px';**

**particle.style.height= size+ 'px';**

**particle.style.left= Math.random() \* 100+ '%';**

**particle.style.top= Math.random() \* 100+ '%';**

**// Случайная задержка анимации**

**particle.style.animationDelay= Math.random() \* 6+ 's';**

**particle.style.animationDuration= (Math.random() \* 4+ 4) + 's';**

**particlesContainer.appendChild(particle);**

**}**

**}**

**// Инициализация при загрузке страницы**

**window.addEventListener('load', () =>{**

**createParticles();**

**});**

**// Добавление интерактивности к карточкам**

**document.querySelectorAll('.video-card').forEach(card=>{**

**card.addEventListener('mouseenter', () =>{**

**card.style.zIndex= '10';**

**});**

**card.addEventListener('mouseleave', () =>{**

**card.style.zIndex= '1';**

**});**

**});**

**</script>**

**</body>**

**</html>**