ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

СОГЛАСОВАНО

Старший преподаватель департамента программной инженерии факультета компьютерных наук

_____ А. В. Поповкин
«_____» _____ 2020 г.

УТВЕРЖДАЮ

Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук

_____ В. В. Шилов
«_____» _____ 2020 г.

ПРИЛОЖЕНИЕ ДЛЯ СОВМЕСТНОГО ПРОСМОТРА ФИЛЬМОВ

Пояснительная записка

Лист Утверждения

RU.17701729.02.07-01 81 01-1-ЛУ

Исполнитель: Студент группы БПИ-194
_____ В. А. Анненков
«_____» _____ 2020 г.

Москва 2020

Утверждён
RU.17701729.02.07-01 81 01-1-ЛУ

ПРИЛОЖЕНИЕ ДЛЯ СОВМЕСТНОГО ПРОСМОТРА ФИЛЬМОВ

Пояснительная записка

RU.17701729.02.07-01 81 01-1

Листов 33

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Москва 2020

Аннотация

В данном программном документе приведена пояснительная записка к серверному приложению «Приложение для совместного просмотра фильмов».

В данный документ внесены разделы «Введение», «Назначение и область применения программы», «Технические характеристики», «Ожидаемые технико-экономические показатели», «Источники, использованные при разработке».

В разделе «Введение» указано наименование программы и документы, на основании которых ведется разработка.

В разделе «Назначение и область применения» указано функциональное назначение программы и эксплуатационное назначение программы.

В разделе «Технические характеристики» указаны постановка задачи на разработку программы, описание и обоснование метода организации входных и выходных данных, описание и обоснование выбора состава технических и программных средств.

В разделе «Ожидаемые технико-экономические показатели» указана предполагаемая потребность и экономические преимущества разработки по сравнению с отечественными и зарубежными образцами или аналогами

Настоящий документ разработан в соответствии с требованиями:

1) ГОСТ 19.101-77 Виды программ и программных документов [3];
2) ГОСТ 19.102-77 Стадии разработки [4];
3) ГОСТ 19.103-77 Обозначения программ и программных документов [5];
4) ГОСТ 19.104-78 Основные надписи [6];
5) ГОСТ 19.105-78 Общие требования к программным документам [7];
6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом [8];
7) ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению [9].

Изменения к Пояснительной записке оформляются согласно ГОСТ 19.603-78 [10], ГОСТ 19.604-78 [11].

Содержание

1. Введение

1.1. Наименование программы

1.1.1. Наименование программы на русском языке

Приложение для совместного просмотра фильмов.

1.1.2. Наименование программы на английском языке

Application for Collective Movie Watching.

1.2. Основание для разработки

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 “Программная инженерия” и утвержденная академическим руководителем тема курсового проекта.

2. Назначение и область применения

2.1. Функциональное назначение

Функциональным назначением «приложения для совместного просмотра фильмов» является предоставление пользователям возможности смотреть фильмы на расстоянии. Пользователи могут загружать видео на сервер, воспроизводить видео, общаться с другими пользователями посредством сообщений и «реакций».

Сама программа реализует серверную часть сервиса для совместного просмотра фильмов. Основная задача программы – предоставить приложениям-клиентам всю необходимую информацию для реализации логики совместного просмотра фильмов.

Отдельный клиент может загрузить видео на сервер, после чего любой из посторонних клиентов сможет скачать это видео. Для синхронизации воспроизведения видео клиенты могут использовать WebSocket интерфейс сервера.

Каждое загруженное на сервер видео обрабатывается и разделяется на небольшие фрагменты по 10 секунд. Такой подход позволяет клиентам быстрее скачивать видео небольшими порциями, а также мгновенно переключаться между различными разрешениями в процессе просмотра.

Видео преобразовывается в формат HLS (HTTP Live Streaming). Особенность этого формата в том, что он поддерживается любыми операционными системами: Windows, macOS, Android, iOS и т. д. Несмотря на то, что обработка видео в формат HLS требует длительного времени, с текущим форматом это не является проблемой. Видео, которые преобразованы в HLS лишь частично, могут быть воспроизведены клиентами до момента последнего обработанного фрагмента. Таким образом клиенты могут воспроизводить фильмы мгновенно после загрузки на сервер, не дожидаясь полной обработки. Обработка идёт параллельно.

2.2. Эксплуатационное назначение

Сервер может быть использован frontend-клиентами для реализации логики совместного просмотра фильмов.

«Приложение для совместного просмотра фильмов» может быть использовано пользователями для совместного просмотра фильмов или других видео на расстоянии. Данное приложение становится особенно актуальным, когда требуется что-то посмотреть наиболее простым способом. Для этого в приложении реализован минималистичный интерфейс, возможность быстро загрузить собственный файл с фильмом и сразу же начать его просмотр. Приложение не ограничивает пользователя собственной базой фильмов, предоставляя пользователю возможность самостоятельно загружать файлы с фильмами.

3. Технические характеристики

3.1. Постановка задачи

Программа обеспечивает возможность выполнения следующих функций:

1) Загрузка видеофайла для его дальнейшей обработки в формат HLS.
2) Корректное получение обработанного видео в формате HLS.
3) Получение актуальных данных о перемотке видео с других клиентов. Поддержание видео в актуальном состоянии.
4) Получение актуальных данных о сообщениях в чате.
5) Получение актуальных данных о «реакциях».

Сервер разработан на базе микросервисной архитектуры со следующими сервисами:

1) *Main Service* – Отвечает за базовое взаимодействие с клиентами. В основном, с помощью протокола WebSocket. Передаёт сообщения в чат, передаёт «реакции», синхронизирует видео между клиентами.
2) *Video Service* – Отвечает за обработку входящих видеофайлов и их хранение. Преобразует в формат HLS. Предоставляет API для работы с файлами видео.
3) *Eureka Server* – Отвечает за взаимодействие сервисов друг с другом, помогает различным сервисам находить друг друга.
4) *Gateway Server* – Каждый сервис работает на собственном порте. Клиентам было бы неудобно искать сервисы по различным адресам, поэтому этот сервис берёт на себя такую задачу. Точка входа для клиента. Сервис, который перенаправляет на другие сервисы.

3.2. Описание алгоритма и функционирования программы

3.2.1. Описание алгоритма загрузки и обработки видеофайла

На серверной части реализован алгоритм по преобразованию единого видеоролика в набор видеороликов меньшей длительности (сегментов). Также добавлена конвертацию видеоролика в видеоролики с меньшим качеством (Рис. 1).

Алгоритм конвертации видеоролика следующий:

1) Клиент загружает видеофайл на сервер.

2) Сервер конвертирует видеофайл в видеофайлы меньших разрешений.

$$video1080 \Rightarrow videos = \{video1080, video720, video480, video360, video240, video144\}$$

Данное действие требуется сделать для обеспечения возможности менять качество воспроизводимых роликов в плеере клиента, а также для сжатия самих файлов с целью ускорения кэширования видеороликов клиентом.

3) Сервер разделяет каждый видеоролик из множества видеороликов *videos* на небольшие фрагменты (1 сек.).

$$video = part1 + part2 + \cdots, \quad video \in videos$$

Данное действие требуется сделать для обеспечения возможности измененять качество и настройки видео в процессе его воспроизведения (без повторного кэширования и приостановки воспроизведения).

Реализовано API, с помощью которого клиенты могут загружать видео на обработку (Рис. 2), а затем скачивать в обработанном формате. Формат получаемого видео – *HLS*. Данный формат представляет из себя набор файлов: один файл формата *.m3u8*, несколько файлов формата *.ts*. Файл *.m3u8* содержит в себе информацию о нахождении сегментов. На основе информации из файла *.m3u8* клиенты делают запросы на сервер для получения требуемых сегментов.

Сам процесс обработки реализован с помощью инструмента *ffmpeg*. *ffmpeg* – мощный консольный инструмент, который позволяет осуществлять различные действия с видеофайлами. В том числе осуществлять конвертацию в различные форматы.

Листинг 1 — Пример команды ffmpeg для генерации файла в формате HLS

```
ffmpeg -i /path/ \
    -c:a aac \
    -strict experimental \
    -c:v libx264 \
    -s filename \
    -aspect 16:9 \
    -f hls \
    -hls_list_size 0 \
    -hls_time 10 \
    -threads 0 \
    {some}p/video.m3u8
```

ffmpeg это консольный инструмент, а взаимодействие с ним требуется осуществлять с помощью Java. Для реализации взаимодействия с bash-консолью используется класс *ProcessBuilder* в

Java. В отдельном потоке происходит чтение прогресса обработки из консоли, результат парсится с помощью регулярных выражений и преобразуется в удобный для использования формат. В процессе обработки клиент уже может загружать обработанные фрагменты и показывать фильм для пользователей.

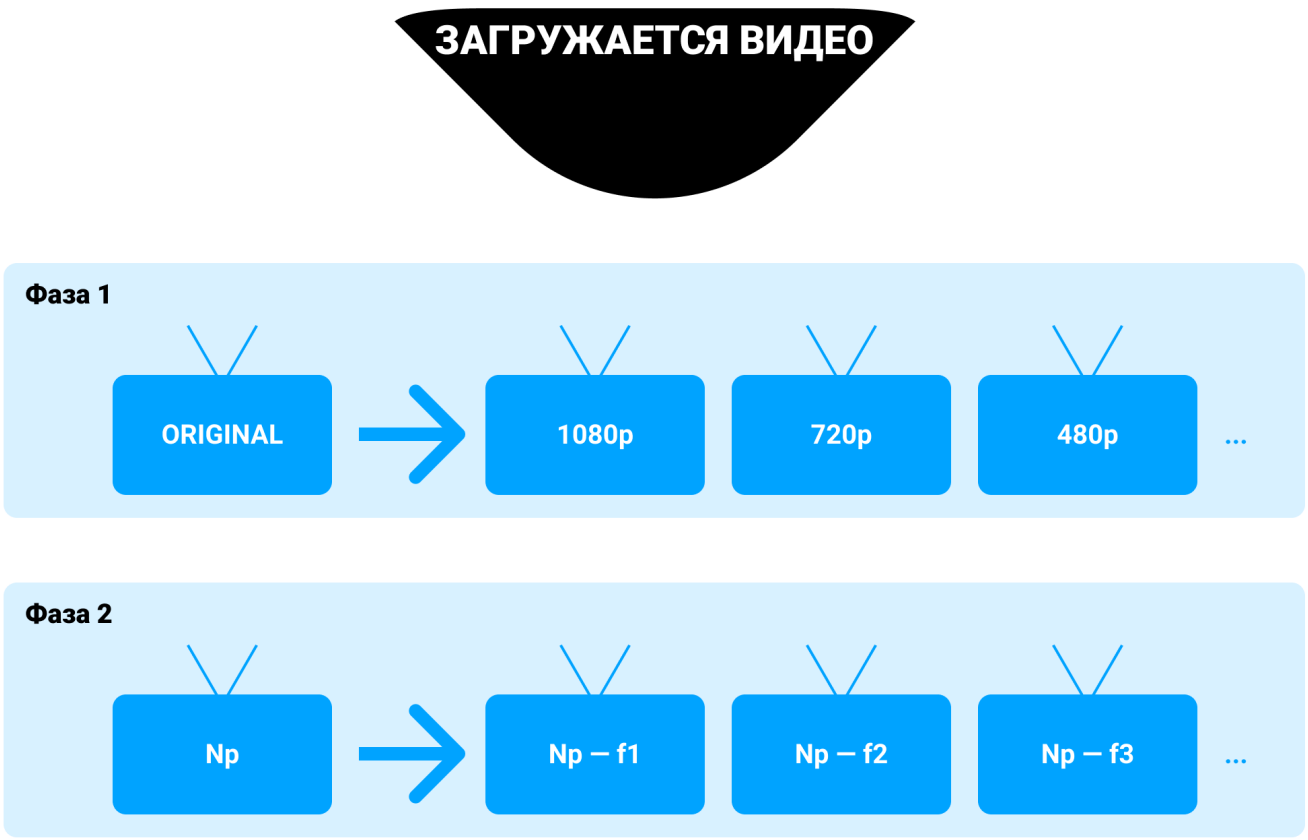

Рис. 1 — Конвертация видео в формат HLS

3.2.2. Описание алгоритма синхронизации видеопотока

3.2.2.1. Общий алгоритм

Для синхронизации видеопотока между разными клиентами используется протокол WebSocket. Каждому клиенту требуется установить соединение с WebSocket сервером по адресу «/main/ws». В процессе просмотра видео клиенты могут отправлять следующие запросы на сервер: воспроизведение, пауза, перемотка. В ответ сервер отправляет данный запрос всем клиентам, просматривающим видео в текущей комнате.

Когда сервер отправляет команду на воспроизведение, паузу или перемотку, он дополнительно передаёт информацию о времени, когда было осуществлено действие. Клиент может использовать эту информацию, чтобы рассчитать разницу между текущим и полученным временем, тем самым ещё сильнее уменьшив задержку при получении команды.

В процессе перемотки видео может возникнуть ситуация, когда у разных клиентов видео загружается с различной скоростью. В таком случае произойдёт рассинхронизация видео, ведь некоторые клиенты запустят воспроизведение раньше остальных. Для решения этой ситуации была реализована перемотка с подтверждением. Алгоритм следующий:

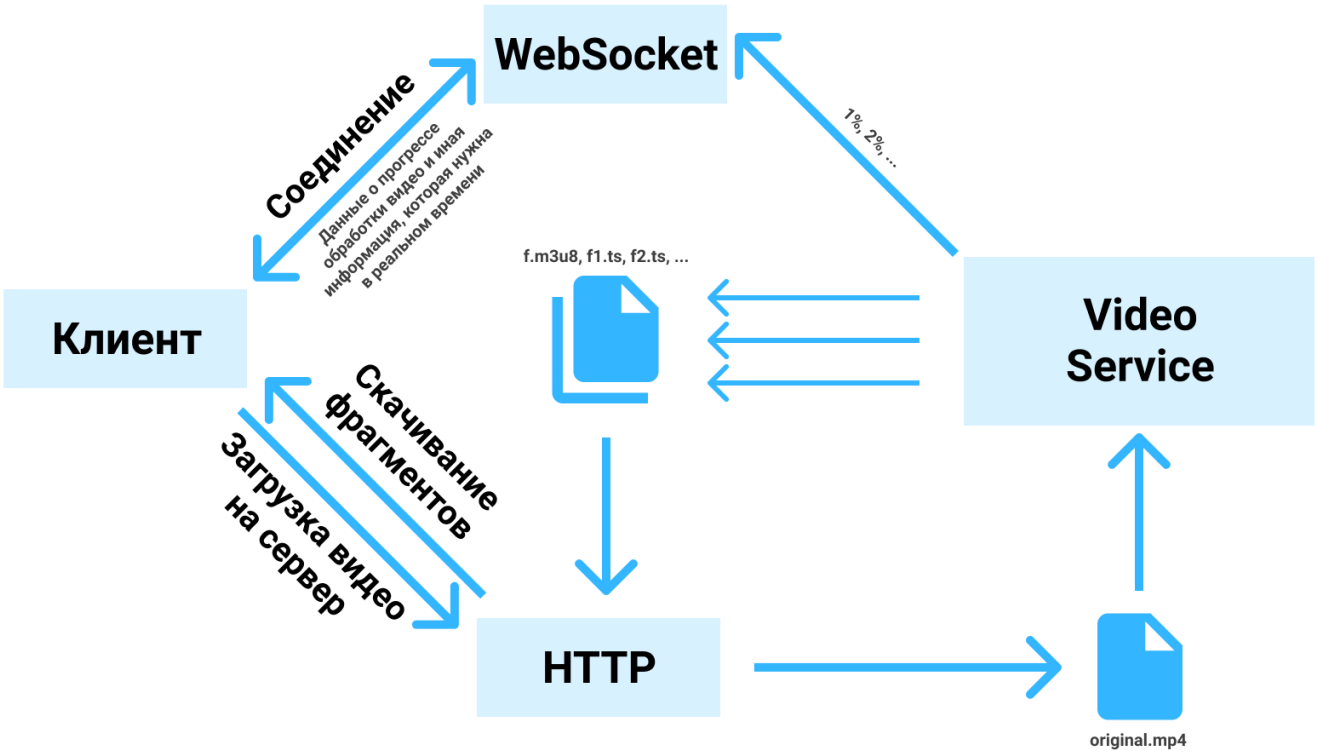

Рис. 2 — Взаимодействие клиента с сервером

1) Один из клиентов осуществляет перемотку видео на определённую позицию и отправляет команду для перемотки на сервер.

2) Эту команду получают остальные клиенты и тоже осуществляют перемотку на полученную позицию.

3) Как только некий клиент загрузил фрагмент видео и способен его воспроизвести, он отправляет на сервер команду *ready*, но воспроизведение видео не начинает. Так поступают все клиенты.

4) Когда сервер получает команду *ready* от всех клиентов, он снова посылает команду всем клиентам с пометкой, что можно запускать видео.

5) В итоге: клиенты синхронизированы и запускают видео одновременно.

3.2.2.2. Определение задержки

В целях обеспечения наименьшей рассинхронизации видеопотока между клиентами, требуется хранить переменные $diff\_sc$ и $diff\_cs$ для каждого клиента. Данные переменные будут содержать в себе информацию о количестве затрачиваемого времени при передаче данных от сервера к клиенту или от клиента к серверу.

Формула для определения переменных $diff\_sc$ и $diff\_cs$: $\text{diff\_sc} = \text{time\_sc}_2 - time\_sc_1$ $diff\_cs = time\_cs_2 - time\_cs_1$, где:

– $time\_sc_1$ — время отправки сообщения сервером;
– $time\_sc_2$ — время получения сообщения клиентом;

– $diff\_sc$ — разница между временем отправки и временем получения при передаче сообщения от сервера к клиенту;
– $time\_cs_1$ — время отправки сообщения клиентом;
– $time\_cs_2$ — время получения сообщения сервером;
– $diff\_cs$ — разница между временем отправки и временем получения при передаче сообщения от клиента к серверу.

Так как возможно подключение нескольких клиентов, требуется хранить содержимое значений $diff\_sc_i$ и $diff\_cs_i$ для всех n клиентов.

Алгоритм (Рис. 3) синхронизации видео единый:

1) Клиент k отправляет серверу запрос на действие d (перемотку/приостановку/возобновление) видео.

2) Сервер отправляет всем клиентам сообщение с текущим серверным временем. Клиенты принимают значение и считают разницу $diff\_sc_i$ с учётом своего времени. Затем клиенты отправляют посчитанную разницу времени в миллисекундах обратно серверу, а также отправляют текущее клиентское время. С учётом полученной информации сервер считает разницу $diff\_cs_i$.

3) Сервер отправляет всем клиентам команду выполнить действие d и передаёт каждому клиенту задержку $delay_i$, которая считается по формуле:

$$delay_i = \max(diff\_sc_1, \ldots, diff\_sc_n) - diff\_sc_i + diff\_cs_k, \quad i \in [1, \ldots, n]$$

Значение $delay_i$ требуется по-разному использовать в различных ситуациях. При организации совместного просмотра фильма возможны следующие сценарии:

– **Перемотка** видео на позицию t мс одним из клиентов.

  При получении такой команды все клиенты (кроме клиента-инициатора) выполняют перемотку видео на позицию $(t + delay_i)$ мс.

– **Приостановка** видео одним из клиентов.

  При получении такой команды все клиенты (кроме клиента-инициатора) приостанавливают воспроизведение и выполняют перемотку видео на $delay_i$ мс назад.

– **Возобновление** видео одним из клиентов.

  При получении такой команды все клиенты (кроме клиента-инициатора) возобновляют воспроизведение и выполняют перемотку видео на $delay_i$ мс вперёд.

3.2.3. Описание алгоритма и функционирования чата и «реакций»

Реализации чата и «реакций» работают на основе протокола WebSocket. Каждому клиенту требуется установить соединение с WebSocket сервером по адресу «/main/ws». После этого клиенты могут подписаться на специальные «топики» комнат. Таким образом, когда в «топике» будет появляться новое действие (напр., новое сообщение или новая «реакция»), это действие получит любой клиент, подписанный на этот «топик».

Формат топика следующий: «/topic/room/{**ID комнаты**}/{**команда**}», где:

1) ID комнаты – уникальный идентификатор комнаты, в которой клиенты просматривают фильм;

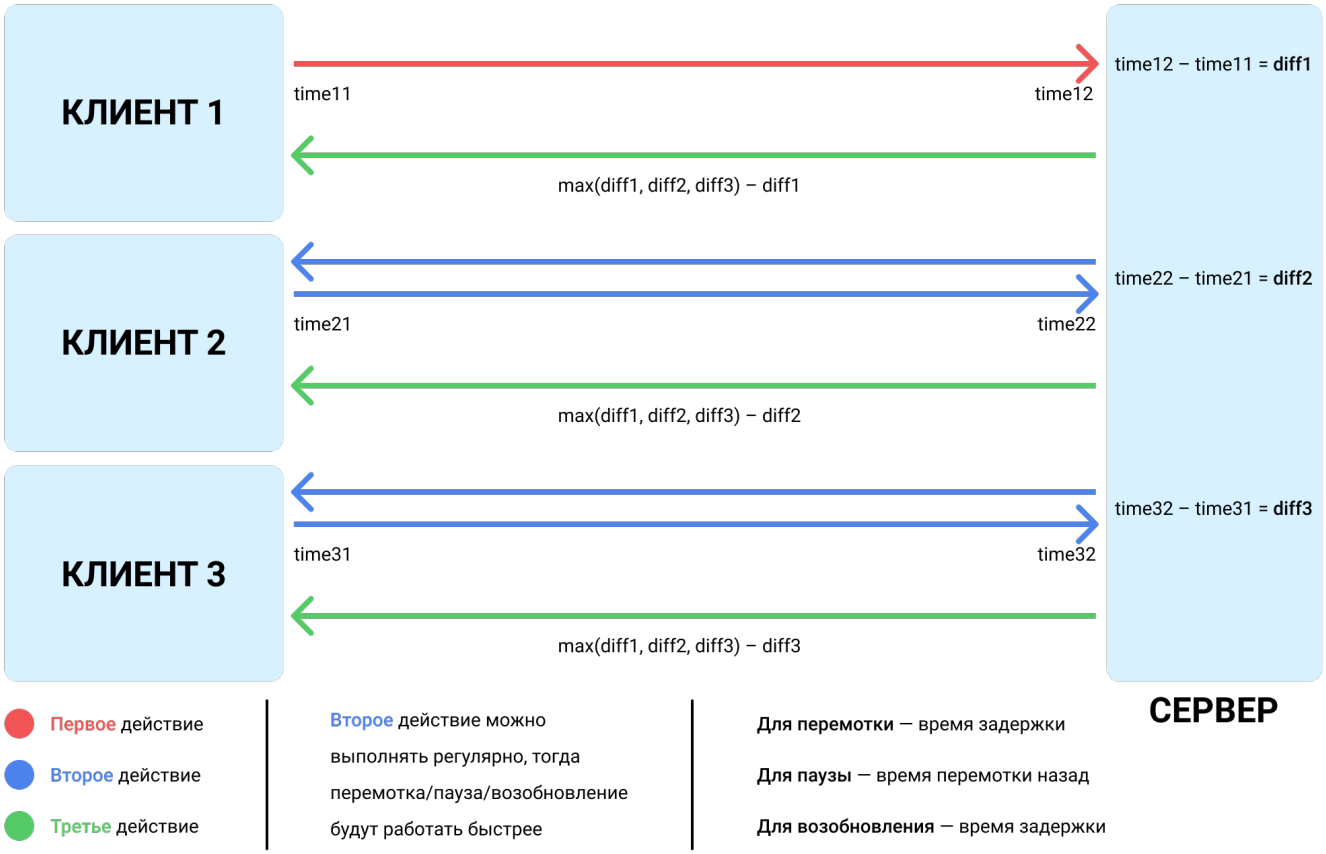

Рис. 3 — Взаимодействие клиентов с сервером

2) команда – тип события, на которое требуется подписаться:

   – messages – новые сообщения;
   – reactions – новые «реакции».

У каждого сообщения или «реакции» есть информация об отправителе, которую клиенты также могут использовать.

3.3. Описание и обоснование метода организации входных и выходных данных

3.3.1. Описание и обоснование метода организации входных данных

Взаимодействие с клиентами организовано в виде HTTP и WebSocket подключений.
При подключении через HTTP формат входных данных зависит от типа HTTP-метода:

1) GET – входные данные задаются в url.

2) POST, PUT, DELETE – параметры задаются в теле запроса.

При передаче входных данных через WebSocket формат входных данных представляет из себя JSON-объект.

3.3.2. Описание и обоснование метода организации выходных данных

При взаимодействии с любым протоколом формат выходных данных – JSON объект.

3.3.3. Примеры запросов

Отправлен корректный запрос с видеофайлом на адрес `https://comnata.tv/video/upload`;

Получен ответ:

```
{
    "status": "SUCCESS",
    "videoId": "322e60eedf4245a08ebc1d9bfc23a5bd",
    "videoUrl": "/video/getVideo/322e60eedf4245a08ebc1d9bfc23a5bd/video.m3u8"
}
```

3.4. Описание и обоснование выбора состава технических и программных средств

3.4.1. Состав технических и программных средств

Исходные коды для сервера должны реализованы с использованием следующих языков и технологий:

1) Kotlin, Java – использовать в качестве основных языков программирования;
2) Spring Boot – для реализации базовой архитектуры сервера, а также REST API методов и панели администрирования;
3) Spring Flyway – для миграций и версионирования базы данных;
4) Hibernate, Spring Data – для работы с базой данных, для связки таблиц базы данных с классами Java;
5) ffmpeg – для обработки видеофайлов: нарезка и изменение качества;
6) PostgreSQL – использовать в качестве СУБД;
7) Docker, docker-compose – для обеспечения переносимости сервера.

Требования к информационным и программным характеристикам сервера:

1) Наличие доступа по SSH;
2) Открытые порты: 22, 40, 80, 443, 3000, 5432, 5454, 8080-8100;
3) Установленные утилиты: nano, apt-get;
4) Установленные: docker, docker-compose, docker-machine;
5) Установленный с настройками по-умолчаию: Nginx, пользователь с доступом по SSH должен иметь права на редактирование конфигурации;
6) Операционная система Ubuntu 16.04.

3.4.2. Обоснование выбора состава технических и программных средств

За основу веб-сервера были выбраны языки программирования Kotlin/Java и фреймворк Spring Boot.

Kotlin и Java – богатые по функционалу и производительные языки программирования, которые удобно использовать для написания больших приложений. Основным преимуществом этих языков является хорошая реализация ООП парадигмы и статическая типизация. Таким образом, в масштабных проектах легче отлаживать ошибки с использованием этих языков программирования.

Spring Boot – основной веб-фреймворк для языков программирования Kotlin и Java. Данный фреймворк предоставляет собственную удобную реализацию процесса IoC (Inversion of Control), благодаря которому мы можем логично выстраивать зависимости между классами внутри нашего приложения.

Для версионирования базы данных был выбран инструмент Flyway. Благодаря Flyway в проекте есть возможность изменять базу данных и осуществлять переносимость её структуры.

Spring Data позволяет удобно взаимодействовать с базой данных. Данный инструмент позволяет помечать аннотацией @Entity классы данных, чтобы связать их с сущностью в базе данных. Взаимодействие с базой производится с помощью самописных интерфейсов, реализующих интерфейс JpaRepository.

ffmpeg – мощный консольный инструмент для обработки видео. ffmpeg позволяет конвертировать видео в любой формат, добавлять аудиодорожки, менять разрешение и многое другое. Данный инструмент лежит в основе реализации конвертации видео в формат HLS.

В качестве СУБД была выбрана система PostgreSQL. Такое решение было принято по нескольким причинам:

1) открытый исходный код СУБД, что делает её бесплатной;

2) адаптированность для хранения и обработки больших объёмов данных;

3) хорошая поддержка фреймворком Spring Data;

4) возможность получить бесплатную базу данных PostgreSQL в админ-панели Heroku.

Так как сервер написан на микросервисной архитектуре, то полезно настроить контейнер Docker, чтобы переносимость сервера была безболезненной. К каждому сервису написан Dockerfile, который можно запустить одной командой для запуска его контейнера Docker. Для связки всех контейнеров Docker используется файл docker-compose, который объединяет все Dockerfile и задаёт правила для их взаимодействия.

4. Ожидаемые технико-экономические показатели

4.1. Предполагаемая потребность

Приложение могут использовать пользователи для совместного просмотра фильмов дистанционно.

4.2. Экономические преимущества по сравнению с отечественными и зарубежными аналогами

Большинство существующих сервисов поддерживают воспроизведение видео из существующей базы, не предоставляя пользователям возможность смотреть любые желаемые видео. Кроме того, сервисы, которые всё-таки имеют возможность загрузки видео, не позволяют просматривать видео в процессе его обработки – для просмотра требуется дождаться полной обработки, а этот процесс может быть длительным.

5. Источники, использованные при разработке

1) ГОСТ 19.101-77 Виды программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

2) ГОСТ 19.102-77 Стадии разработки. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

3) ГОСТ 19.103-77 Обозначения программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001

4) ГОСТ 19.104-78 Основные надписи. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

5) ГОСТ 19.105-78 Общие требования к программным документам. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001

7) ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

8) Загрузка файлов [Электронный ресурс] / Spring-Projects. Режим доступа: `https://spring-proj ru/guides/uploading-files/`, свободный. (Дата обращения: 15.05.2020)

9) Как установить и настроить PostgreSQL в MacOS [Электронный ресурс] / 900913. Режим доступа: `https://900913.ru/note/b/postgresql-macos-9da176/`, свободный. (Дата обращения: 15.05.2020)

10) Курс по Spring / ВТБ // [Электронный ресурс]: Google Drive. Режим доступа: `https:// drive.google.com/drive/folders/1PQspMs1gm8aIFNua9l-wDsjgPxidqfhC`, свободный. (Дата обращения: 15.05.2021)

11) Тянем ролик с Youtube и раздаем по WebRTC в реалтайме [Электронный ресурс] / Flashphoner. Режим доступа: `https://flashphoner.com/tyanem-rolik-s-youtube-i-razdaem-po-webrtc-v ?lang=ru`, свободный. (Дата обращения: 15.05.2020)

12) Adaptive HTTP Streaming Technologies: HLS vs. DASH / Tech Blog // [Электронный ресурс]: StriveCast. Режим доступа: `https://strivecast.com/hls-vs-mpeg-dash/`, свободный. (Дата обращения: 15.05.2021)

13) Create Multiple tables [Электронный ресурс] / LaunchSchool. Режим доступа: `https:// launchschool.com/books/sql_first_edition/read/multi_tables`, свободный. (Дата обращения: 15.05.2020)

14) Docker документация [Электронный ресурс] / Docker. Режим доступа: `https://docs. docker.com/`, свободный. (Дата обращения: 15.05.2020)

15) ffmpeg документация [Электронный ресурс] / ffmpeg. Режим доступа: `https://ffmpeg. org/ffmpeg.html`, свободный. (Дата обращения: 15.05.2020)

16) GitHub [Электронный ресурс]. Режим доступа: https://github.com/, свободный. (Дата обращения: 15.05.2020)

17) How do I use ffmpeg to get the video resolution? / Vladimir Stazhilov // [Электронный ресурс]: SuperUser. Режим доступа: https://superuser.com/questions/841235/how-do-i-use-ffmpeg- свободный. (Дата обращения: 15.05.2021)

18) How to create .mpd or .m3u8 video file on the server using FFMPEG for Adaptive Streaming / Mayur Solanki // [Электронный ресурс]: Medium. Режим доступа: https://mayur-solanki.medium.com/how-to-create-mpd-or-m3u8-video-file-from-server-using-ffmpeg-97e9e1fbf свободный. (Дата обращения: 15.05.2021)

19) How to read ffmpeg response from java and use it to create a progress bar? / shalki // [Электронный ресурс]: StackOverflow. Режим доступа: https://stackoverflow.com/questions/10927718/how-to-read-ffmpeg-response-from-java-and-use-it-to-create-a-progress-bar свободный. (Дата обращения: 15.05.2021)

20) How video streaming works on the web: An introduction / Paul Berberian // [Электронный ресурс]: Medium. Режим доступа: https://medium.com/canal-tech/how-video-streaming-works- свободный. (Дата обращения: 15.05.2021)

21) Kotlin документация [Электронный ресурс] / JetBrains. Режим доступа: https://kotlinlang.org/docs/, свободный. (Дата обращения: 15.05.2020)

22) PostgreSQL документация [Электронный ресурс] / PostgreSQL. Режим доступа: https://www.postgresql.org/docs/, свободный. (Дата обращения: 15.05.2020)

23) Spring документация [Электронный ресурс] / Spring. Режим доступа: https://spring.io/, свободный. (Дата обращения: 15.05.2020)

24) Spring Cloud Demo Project / gonwan // [Электронный ресурс]: GitHub. Режим доступа: https://github.com/gonwan/spring-cloud-demo, свободный. (Дата обращения: 15.05.2021)

25) Spring Cloud Netflix Microservices – start project (серия статей) – часть 4 / Kirill Sereda // [Электронный ресурс]: Medium. Режим доступа: https://medium.com/@kirill.sereda/spring-cloud-netflix-microservices-start-project-%D1%81%D0%B5%D1%80%D0%B8%D1%8F-%D1%81%D1%82%D0%B0%D1%82%D0%B5%D0%B9-%D1%87%D0%B0%D1%81%D1%82%D1%8C-4-d2137d19d783, свободный. (Дата обращения: 15.05.2021)

26) StackOverflow [Электронный ресурс]. Режим доступа: https://stackoverflow.com/, свободный. (Дата обращения: 15.05.2020)

27) streaming an mkv file while processing with ffmpeg / Phani Rithvij // [Электронный ресурс]: StackOverflow. Режим доступа: https://stackoverflow.com/questions/55460359/streaming-an-mkv-file-while-processing-with-ffmpeg, свободный. (Дата обращения: 15.05.2021)

ПРИЛОЖЕНИЕ 1

Терминология

IoC (Inversion of Control) – архитектурный принцип объектно-ориентированного программирования, используемый для уменьшения зацепления (связанности) в компьютерных программах.

HLS (HTTP Live Streaming) – коммуникационный протокол для потоковой передачи медиа на основе HTTP, разработанный компанией Apple.

База данных – совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных.

Реакция – всплывающий смайлик для быстрой передачи эмоций во время просмотра.

Терминал – разновидность текстового интерфейса между человеком и компьютером, в котором инструкции компьютеру даются в основном путём ввода с клавиатуры текстовых строк (команд).

Комната – виртуальное пространство, в котором воспроизводится видео и к которому могут подключаться пользователи.

Клиент – приложение, которое подключается к серверу.

Сегмент – небольшой отрывок из исходного видео.

Сервер – выделенный или специализированный компьютер для выполнения сервисного программного обеспечения.

Серверное приложение – программа на сервере.

СУБД – система управления базами данных.

Фреймворк – дополнение к программе, расширяющее функционал или упрощающее процесс разработки.

ПРИЛОЖЕНИЕ 2

Описание формата HLS

Формат HLS представляет из себя набор файлов с расширениями *.m3u8* (один файл) и *.ts* (несколько файлов).

– *.m3u8* – Обычный текстовый файл, который содержит информацию о расположении файлов с расширением *.ts*. Может ссылаться как на медиа-файлы, так и на другие файлы с расширением *.m3u8*. Например, для указания разрешений видео.

– *.ts* – Медиа-файл, небольшой отрывок из исходного видео.

Листинг 2 — Пример содержимого файла с расширением .m3u8

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:17
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:16.683333,
video0.ts
#EXTINF:8.341667,
video1.ts
#EXTINF:8.341667,
video2.ts
#EXTINF:8.341667,
video3.ts
...
```

ПРИЛОЖЕНИЕ 3

Структура базы данных

Приведена схема базы данных (Рис. 4).
В базе данных:

– таблица *app_user* хранит информацию о пользователях;

– таблица *room* хранит информацию о комнатах;

– таблица *flyway_schema_history* хранит информацию о применённых миграциях к базе данных.

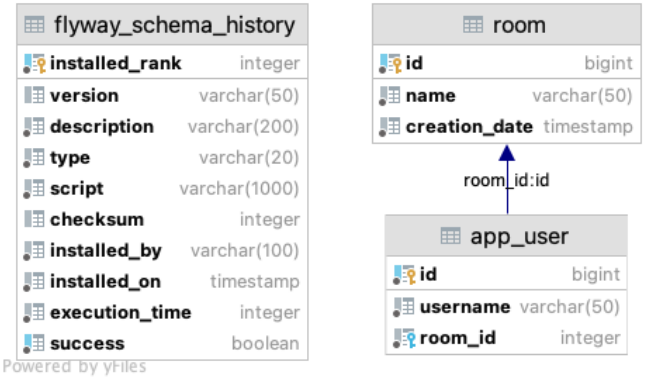

Рис. 4 — Схема базы данных

ПРИЛОЖЕНИЕ 4

Описание и функциональное назначение классов

Таблица 1 — Описание классов модуля MainService

Класс	**Назначение**
MainController	Контроллер с основными методами
WebsocketController	Контроллер с методами для взаимодействия через протокол WebSocket
RoomService	Сервис работы с комнатами
WebsocketService	Сервис для взаимодействия с WebSocket
RoomRepository	Репозиторий для получения информации о комнатах
UserRepository	Репозиторий для получения информации о пользователях
WebsocketEventListener	Слушатель для определения событий подключения/отключения пользователей
Room	Модель комнаты
User	Модель пользователя
RoomActionRequest	Модель запроса для синхронизации видео
RoomChatMessageRequest	Модель запроса для сообщений в чате
RoomReactionRequest	Модель запроса для реакций
Response	Абстрактная модель ответа
RoomActionResponse	Модель ответа для синхронизации видео
RoomChatMessageResponse	Модель ответа для сообщений в чате
RoomJoinResponse	Модель ответа для подключений к комнате
RoomLeftResponse	Модель ответа для выходов из комнаты
RoomReactionResponse	Модель ответа для реакции
WebsocketEnums	Enum классы для WebSocket

Таблица 2 — Описание классов модуля VideoService

Класс	**Назначение**
VideoController	Контроллер для взаимодействия с видеофайлами
VideoService	Сервис для работы с видеофайлами
FfmpegManager	Класс для обработки видеофайлов с помощью ffmpeg
VideoResolution	Модель разрешения видео
MainClient	Клиент для получения информации из MainService

ПРИЛОЖЕНИЕ 5

Описание и функциональное назначение полей, методов и свойств модуля MainService

Таблица 3 — Описание полей, методов и свойств класса MainController

MainController				
Поля				
Имя	**М. дост.**	**Тип**	**Назначение**	
logger	private	Logger	Объект для логгирования	
roomService	private	RoomService	Сервис для взаимодействия с комнатами	
Методы				
Имя	**М. дост.**	**Тип**	**Арг.**	**Назначение**
createVideo	public	void	String videoUuid	Метод для добавления видео в базу данных
setVideoProgress	public	void	String videoUuid	Метод для установки прогресса обработки видеофайла

Таблица 4 — Описание полей, методов и свойств класса WebsocketController

WebsocketController				
Поля				
Имя	**М. дост.**	**Тип**	**Назначение**	
URL_ROOM _JOIN	private	String	Сервис для взаимодействия с комнатами	
URL_ROOM _VIDEO _ACTION	private	String	Сервис для взаимодействия с комнатами	
URL_ROOM _VIDEO _ACTION _READY	private	String	Сервис для взаимодействия с комнатами	
URL_ROOM _CHAT _MESSAGE	private	String	Сервис для взаимодействия с комнатами	
URL_ROOM _REACTION	private	String	Сервис для взаимодействия с комнатами	
logger	private	Logger	Объект для логгирования	
roomService	private	RoomService	Сервис для взаимодействия с комнатами	
Методы				
Имя	**М. дост.**	**Тип**	**Арг.**	**Назначение**
processRoomJoin	public	void	Principal principal	Метод для добавления видео в базу данных
processRoom VideoAction	public	void	Principal principal	Метод для установки прогресса обработки видеофайла
processRoom VideoAction Ready	public	void	Principal principal	Метод для установки прогресса обработки видеофайла
processRoom ChatMessage	public	void	Principal principal	Метод для установки прогресса обработки видеофайла
processRoom Reaction	public	void	Principal principal	Метод для установки прогресса обработки видеофайла

Таблица 5 — Описание полей, методов и свойств класса RoomService

RoomService				
Поля				
Имя	**М. дост.**	**Тип**	**Назначение**	
logger	private	Logger	Объект для логгирования	
websocketService	private	RoomService	Сервис для взаимодействия с протоколом WebSocket	
userRepository	private	RoomService	Сервис для взаимодействия с пользователями	
roomRepository	private	RoomService	Сервис для взаимодействия с комнатами	
actionsStore	private	RoomService	Компонент для хранения локальных данных	
URL_ROOM _JOINS	private	RoomService	Константа url входов в комнату	
URL_ROOM _LEFTS	private	RoomService	Константа url выходов из комнаты	
URL_ROOM _ACTIONS	private	RoomService	Константа url действий с видео	
URL_ROOM _CHAT _MESSAGES	private	RoomService	Константа url сообщений в чате	
URL_ROOM _REACTIONS	private	RoomService	Константа url реакций	
Методы				
Имя	**М. дост.**	**Тип**	**Арг.**	**Назначение**
checkIsRoomExist	public	void	Principal principal	Метод для проверки наличия комнаты
createRoom	public	void	Principal principal	Метод для создания комнаты
processVideoJoin	public	void	Principal principal	Метод для реализации входа
processVideoLeft	public	void	Principal principal	Метод для реализации выхода
processRoom VideoAction	public	void	Principal principal	Метод для реализации действия с видео
processRoom VideoActionSeek	public	void	Principal principal	Метод для реализации перемотки
processRoom VideoActionReady	public	void	Principal principal	Метод для реализации готовности
processRoom ChatMessage	public	void	Principal principal	Метод для реализации сообщения в чате
processRoom ChatReaction	public	void	Principal principal	Метод для реализации реакции в чате

Таблица 6 — Описание полей, методов и свойств класса WebsocketService

WebsocketService				
Поля				
Имя	**М. дост.**	**Тип**	**Назначение**	
logger	private	Logger	Объект для логгирования	
messagingTemplate	private	SimpMessagingTemplate	Объект для отправки WebSocket сообщений	
simpUserRegistry	private	SimpUserRegistry	Объект для получения количества WebSocket сессий	
Методы				
Имя	**М. дост.**	**Тип**	**Арг.**	**Назначение**
send	public	void	url: String, obj: Any	Метод для отправки сообщений в топик
sendToUser	public	void	url: String, user: String, obj: Any	Метод для отправки сообщений пользователю
getNumberOfSessions	public	int	-	Метод для получения списка сессий

Таблица 7 — Описание полей, методов и свойств класса RoomRepository

RoomRepository				
Методы				
Имя	**М. дост.**	**Тип**	**Арг.**	**Назначение**
saveAndFlush	public	Room	room: Room	Метод для сохранения комнаты в базу
findRoomByName	public	Room?	name: String	Метод для поиска комнаты в базе по имени

Таблица 8 — Описание полей, методов и свойств класса UserRepository

UserRepository				
Методы				
Имя	**М. дост.**	**Тип**	**Арг.**	**Назначение**
findUserByUsername	public	User	name: String	Метод для поиска пользователя в базе по имени
findAllByRoomName	public	List<User>	roomName: String	Метод для поиска всех участников комнаты

Таблица 9 — Описание полей, методов и свойств класса WebsocketEventListener

WebsocketEventListener				
Поля				
Имя	**М. дост.**	**Тип**	**Назначение**	
logger	private	Logger	Объект для логгирования	
roomService	private	RoomService	Сервис для взаимодействия с комнатами	
Методы				
Имя	**М. дост.**	**Тип**	**Арг.**	**Назначение**
handleSessionConnect	public	void	event: SessionConnectEvent	Метод для определения события подключения
handleSessionDisconnect	public	void	event: SessionDisconnectEvent	Метод для определения события отключения

Таблица 10 — Описание полей, методов и свойств класса Action

Action				
Поля				
Имя	**М. дост.**	**Тип**	**Назначение**	
id	private	Long	ID события	
type	private	String	Тип события	
seekTime	private	Double	Время в видео для перемотки	
author	private	String	Автор события	
users	private	MutableSet<String>	Пользователи, задействованные в событии	
Методы				
Имя	**М. дост.**	**Тип**	**Арг.**	**Назначение**
addUser	public	void	user: String	Метод для добавления пользователя в список задействованных
addUsers	public	void	newUsers: Set<String>	Метод для добавления нескольких пользователей в список задействованных
deleteUser	public	void	user: String	Метод для удаления пользователя из списка задействованных
deleteAllUsers	public	void	user: String	Метод для удаления всех пользователей из списка задействованных

Таблица 11 — Описание полей, методов и свойств класса Room

Room			
Поля			
Имя	**М. дост.**	**Тип**	**Назначение**
name	public	String	Имя комнаты
creationDate	public	LocalDateTime	Дата создания комнаты
id	public	Long	ID комнаты
users	public	Set<User>	Список пользователей комнаты

Таблица 12 — Описание полей, методов и свойств класса User

User			
Поля			
Имя	**М. дост.**	**Тип**	**Назначение**
username	public	String	Имя пользователя
room	public	Room	Комната, к которой подключён пользователь
id	public	Long	ID пользователя

Таблица 13 — Описание полей, методов и свойств класса RoomActionReadyRequest

RoomActionReadyRequest			
Поля			
Имя	**М. дост.**	**Тип**	**Назначение**
actionId	public	Long?	ID действия для подтверждения готовности

Таблица 14 — Описание полей, методов и свойств класса RoomActionRequest

RoomActionRequest			
Поля			
Имя	**М. дост.**	**Тип**	**Назначение**
seekTime	public	Double?	Время видео для перемотки
type	public	String?	Тип события

Таблица 15 — Описание полей, методов и свойств класса RoomChatMessageRequest

RoomChatMessageRequest			
Поля			
Имя	**М. дост.**	**Тип**	**Назначение**
text	public	String?	Текст сообщения

Таблица 16 — Описание полей, методов и свойств класса RoomReactionRequest

RoomReactionRequest			
Поля			
Имя	**М. дост.**	**Тип**	**Назначение**
reaction	public	String?	Тип реакции

Таблица 17 — Описание полей, методов и свойств класса Response

Response			
Поля			
Имя	**М. дост.**	**Тип**	**Назначение**
notificationType	public	NotificationType	Тип ответа

Таблица 18 — Описание полей, методов и свойств класса RoomActionResponse

RoomActionResponse			
Поля			
Имя	**М. дост.**	**Тип**	**Назначение**
actionId	public	Long	ID действия
author	public	String	Автор действия
seekTime	public	Double	Время видео для перемотки
type	public	ActionType	Тип действия
step	public	ActionType	Этап действия
actionTime	public	LocalDateTime	Дата и время действия

Таблица 19 — Описание полей, методов и свойств класса RoomChatMessageResponse

RoomChatMessageResponse			
Поля			
Имя	**М. дост.**	**Тип**	**Назначение**
userId	public	Long	ID пользователя-автора
text	public	String	Текст сообщения
dateTime	public	LocalDateTime	Дата и время сообщения в чате

Таблица 20 — Описание полей, методов и свойств класса RoomJoinResponse

RoomJoinResponse			
Поля			
Имя	**М. дост.**	**Тип**	**Назначение**
newUserId	public	String	Новый подключённый пользователь
allUserIds	public	List<String>	Все пользователи после подключения
dateTime	public	LocalDateTime	Дата и время входа

Таблица 21 — Описание полей, методов и свойств класса RoomLeftResponse

RoomLeftResponse			
Поля			
Имя	**М. дост.**	**Тип**	**Назначение**
leftUserId	public	String	Вышедший пользователь
remainingUserIds	public	List<String>	Все пользователи после отключения
dateTime	public	LocalDateTime	Дата и время выхода

Таблица 22 — Описание полей, методов и свойств класса RoomReactionResponse

RoomReactionResponse			
Поля			
Имя	**М. дост.**	**Тип**	**Назначение**
author	public	String	Автор реакции
type	public	Reaction	Тип реакции
dateTime	public	LocalDateTime	Дата и время реакции

Таблица 23 — Описание полей, методов и свойств класса ActionsStore

ActionsStore			
Поля			
Имя	**М. дост.**	**Тип**	**Назначение**
logger	public	Logger	Объект для логгирования
index	public	HashMap<Long, Action>	Тип реакции
waitingActions	public	LocalDateTime	Дата и время реакции

ПРИЛОЖЕНИЕ 6

Описание и функциональное назначение полей, методов и свойств модуля VideoService

Таблица 24 — Описание полей, методов и свойств класса VideoController

VideoController				
Поля				
Имя	**М. дост.**	**Тип**	**Назначение**	
logger	private	Logger	Объект для логгирования	
MEDIA_TYPE	private	String	Тип для загружаемого файла	
videoService	private	VideoService	Сервис для работы с видео	
Методы				
Имя	**М. дост.**	**Тип**	**Арг.**	**Назначение**
getBaseFile	public	ResponseEntity <FileSystem Resource>	response: HttpServletResponse, videoId: String, fileName: String	Метод для получения базового .m3u8 файла с информацией о доступных разрешениях видео
getVideoFile	public	ResponseEntity <FileSystem Resource>	response: HttpServletResponse, videoId: String, resolution: String, fileName: String	Метод для получения видеофайлов-сегментов
uploadFile	public	VideoUploadResponse	request: HttpServletRequest, file: MultipartFile	Метод для загрузки нового видеофайла на сервер

Таблица 25 — Описание полей, методов и свойств класса FfmpegManager

FfmpegManager				
Поля				
Имя	**М. дост.**	**Тип**	**Назначение**	
AVAILABLE _RESOLUTIONS	private	Logger	Доступные разрешения	
COMMAND _HLS_BASE	private	String	Базовая часть команды генерации HLS	
COMMAND _HLS_ONE _RESOLUTION	private	VideoService	Команда для определённого разрешения генерации HLS	
COMMAND _RESOLUTION	private	VideoService	Команда определённого разрешения	
path	private	VideoService	Путь сохранения файла	
fileName	private	VideoService	Имя файла	
listener	private	VideoService	Слушатель обновлений прогресса	
Методы				
Имя	**М. дост.**	**Тип**	**Арг.**	**Назначение**
getVideoResolution	public	VideoResolution	-	Метод для получения разрешения видео
getHlsCommand	private	String	-	Метод для получения комагды генерации HLS
run	public	void	-	Метод обработки видео в отдельном потоке

Таблица 26 — Описание полей, методов и свойств класса VideoService

VideoService				
Поля				
Имя	**М. дост.**	**Тип**	**Назначение**	
logger	private	Logger	Объект для логгирования	
DIRECTORY _PATH	private	String	Директория для загрузки видео	
mainClient	private	VideoService	Клиент для взаимодействия с MainService	
Методы				
Имя	**М. дост.**	**Тип**	**Арг.**	**Назначение**
createDirectory IfNotExists	private	void	realPath: String	Метод для получения базового .m3u8 файла с информацией о доступных разрешениях видео
createBaseFile	private	void	path: String, videoId: String, videoResolution: VideoResolution	Метод для получения видеофайлов-сегментов
createWork Directories	private	void	realPath: String, videoId: String	Метод для загрузки нового видеофайла на сервер
saveVideo	public	VideoUploadResponse	file: MultipartFile, realPath: String	Метод для загрузки нового видеофайла на сервер
onUpdatePercent	public	void	videoUuid: String, percent: Double	Метод для загрузки нового видеофайла на сервер

Таблица 27 — Описание полей, методов и свойств класса MainClient

MainClient				
Методы				
Имя	**М. дост.**	**Тип**	**Арг.**	**Назначение**
createVideo	public	String?	videoUuid: String?	Метод для сохранения комнаты в базу
setVideoProgress	public	String?	videoUuid: String?, videoProgress: Int	Метод для поиска комнаты в базе по имени

Лист регистрации изменений

Изм.	Номера листов				Всего листов в документе	№ документа	Входящий № сопровод. докум. и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					