

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Современные платформы программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

ВИДЕОХОСТИНГ

БГУИР КР 1-40 01 01 613 ПЗ

Студент: Ковалевский М.Ю. гр. 851006

Руководитель: Низовцов Д.В.

Минск 2021

Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ПОИТ
Лапицкая Н.В.

(подпись)

2021 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Ковалевскому Михаилу Юрьевичу, 851006

1. Тема работы Видеохостинг
2. Срок сдачи студентом законченной работы 17.05.2021 г.
3. Исходные данные к работе Язык программирования JavaScript
4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)
Введение.
 1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;
 2. Анализ требований к программному средству и разработка функциональных требований;
 3. Проектирование программного средства;
 4. Тестирование, проверка работоспособности и анализ полученных результатов;
 5. Руководство по установке и использованию;
- Список используемой литературы
- Заключение

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. "Видеохостинг", А1, схема программы, чертеж.

6. Консультант по курсовой работе Низовцов Д.В.

7. Дата выдачи задания 30.01.2021

РУКОВОДИТЕЛЬ _____ Низовцов Д.В.
(подпись)

Задание принял к исполнению Ковалевский М.Ю. _____ 30.01.2021
(дата и подпись студента)

СОДЕРЖАНИЕ

Введение	5
1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству.....	6
1.1 Анализ существующих аналогов.....	6
1.2 Постановка задачи.....	9
2 Моделирование предметной области и разработка функциональных требований	10
2.1 Протокол TCP	10
2.2 Протокол HTTP	11
2.3 Библиотека Socket.IO.....	12
2.4 Формат мультимедиа MP4	13
2.5 Токен Json Web Token.....	15
3 Проектирование программного средства	16
3.1 Создание и проверка Json Web Token	16
3.2 Схемы Mongoose	17
3.3 Алгоритм регистрации.....	18
3.4 Алгоритм загрузки видео на сервер	18
3.5 Алгоритм отправки видео на клиент.....	20
4 Тестирование, проверка работоспособности и анализ полученных результатов .	21
4.1 Тестирование функционала программы	21
4.2 Вывод из прохождения тестирования	22
5 Руководство по установке и использованию	23
5.1 Руководство по установке	23
5.2 Руководство по использованию	23
Заключение	26
Список литературы	27
Приложения	28

ВВЕДЕНИЕ

Видеохостинг - веб-сервис, позволяющий загружать и просматривать видео в браузере с использованием потоковой мультимедиа.

Потоковое мультимедиа - мультимедиа, которое непрерывно получает пользователь от провайдера потокового вещания. Это понятие применимо как к информации, распространяемой через телекоммуникации, так и к информации, которая изначально распространялась посредством потокового вещания (например, радио, телевидение) или не потоковой (например, книги, видеокассеты, аудио CD).

За последние два десятилетия развитие информационных систем сделали потоковую мультимедийную информацию доступной широкому кругу простых пользователей. Различные интернет-сервисы предлагают пользователям возможность просмотра видео без скачивания.

В технологической основе системы лежит три элемента:

- Генератор видеопотока (либо из списка видео файлов, либо прямой оцифровкой с видеокарты, либо копируя существующий в сети поток) и направляет его серверу.
- Сервер - принимает видеопоток и перенаправляет его копии всем подключённым к серверу клиентам, по сути является репликатором данных.
- Клиент - принимает видеопоток от сервера и преобразует его в видеосигнал, который и видит пользователь.

Серверы могут передавать различающиеся по форматам видеоданные, например, MP4, TS, MKV.

В качестве клиента можно использовать любой видеоплеер, поддерживающий потоковое видео и способный декодировать формат, в котором вещает сервер.

Мультимедиа потоки бывают двух видов: по запросу или живыми. Потоки информации, вызываемой по запросу пользователя, хранятся на серверах продолжительный период времени. Живые потоки доступны короткий период времени, например, при передаче видео со спортивных соревнований.

Примерно в 2002 году интерес к единому унифицированному потоковому формату и широкое распространение Adobe Flash поспособствовали разработке формата потокового видео через Flash, который использовался в Flash-проигрывателях, которые размещались на многих популярных видеохостингах, таких как YouTube, сегодня потоковые мультимедиа по умолчанию проигрываются в формате HTML5 видео, которые заменили Flash-проигрыватели.

1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

1.1 Анализ существующих аналогов

1.1.1 Веб-сервис «YouTube»

YouTube - видеохостинг, предоставляющий пользователям услуги хранения, доставки и показа видео.

На сайте YouTube.com пользователи могут загружать видео в нескольких распространённых форматах, в том числе .mpeg и .avi. YouTube автоматически конвертирует их в .mp4 с использованием несвободного кодека H.264, а также кодеков для формата WebM, и делает их доступными для просмотра в онлайн.

Достоинства:

- Поддерживает все распространенные видеоформаты;
- Не взимает плату за использование;
- Позволяет авторам монетизировать видео с помощью рекламы;
- Возможность выкладывания видео с мобильных устройств;
- Широкая языковая поддержка интерфейса позволяет «безбарьерно» пользоваться программным обеспечением носителям различных языковых категорий.

Недостатки:

- Требуется подключение к интернету.

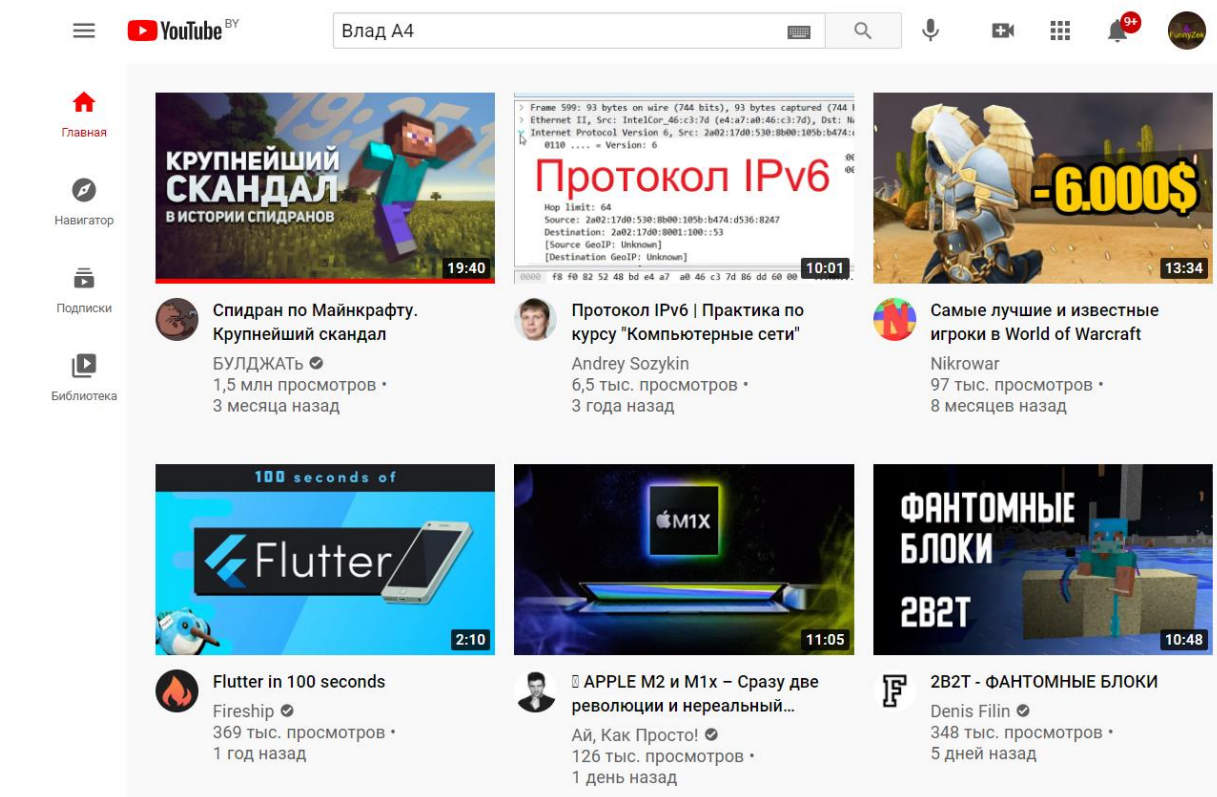


Рисунок 1.1 – Скриншот главной страницы «YouTube»

1.1.2 Веб-сервис «Hulu»

Hulu - это американская служба видео по запросу с подпиской, полностью контролируемая и контролируемая компанией Walt Disney. Hulu вместе с Disney + и ESPN + являются тремя основными потоковыми платформами Disney в США.

Служба подписки Hulu была запущена в бета-версии 29 июня 2010 года и официально запущена 17 ноября 2010 года под брендом Hulu Plus. Служба по-прежнему поддерживалась рекламой, но она предлагает расширенную библиотеку контента, включая полные сезоны, ежедневный доступ к контенту текущего сезона и другие доступные эпизоды шоу. Hulu также запустила приложения Hulu Plus на других типах устройств, включая мобильные, цифровые медиаплееры и игровые консоли. К концу 2011 года у Hulu Plus было около 1,5 миллиона подписчиков.

Количество просмотров сайта отслеживается такими компаниями, как ComScore, Nielsen Ratings и Quantcast. В партнерстве с comScore Hulu является первой цифровой компанией, которая получает многоплатформенные измерения на индивидуальном уровне, включая совместный просмотр устройств в гостиной.

Надежность этих показателей была поставлена под сомнение, отчасти из-за сильно различающихся оценок. Например, в период с мая по июнь 2010 года ComScore обновила свою методологию оценки и свои оценки для Hulu, упав с 43,5 миллионов уникальных и специальных зрителей до 24 миллионов за один месяц. В отчете comScore о тенденциях в области цифровых технологий за 2010 год. Отчет comScore Digital Year in Review показал, что Hulu смотрели вдвое больше, чем зрители, которые смотрели на веб-сайтах пяти основных телевизионных сетей вместе взятых.

В мае 2018 года Hulu объявила, что число подписчиков в США превысило 20 миллионов. Подсчет, согласно которому Netflix составляет около 36 миллионов подписок, был раскрыт на презентации для СМИ в недавно названном театре Hulu Theater в Мэдисон-Сквер-Гарден в Нью-Йорке. Hulu заявила, что увеличила общую вовлеченность более чем на 60%, при этом 78% просмотров приходится на гостиную на подключенных телевизорах.

Достоинства:

- Прост в использовании;
- Имеется веб-версия приложение;
- Приложение можно установить на многие популярные операционные системы;

Недостатки:

- Отсутствие бесплатного варианта сервиса и значительная ограниченность сравнительно «дешевых» вариантов в плане пропускной способности, количества одновременных слушателей и размера серверного хранилища.
- Отсутствует широкая языковая поддержка, что ограничивает пользование клиентов, не владеющих английским на уровне понимания интерфейса программного средства.



Рисунок 1.2 – Скриншот главной страницы «Hulu»

1.1.3 Веб-сервис «Vimeo»

Vimeo - это американская платформа видеохостинга, обмена и предоставления услуг со штаб-квартирой в Нью-Йорке. Vimeo специализируется на доставке видео высокой четкости на различные устройства. Бизнес-модель Vimeo основана на программном обеспечении как услуге (SaaS).

Они получают доход, предоставляя планы подписки для предприятий и производителей видеоконтента. Vimeo предоставляет своим подписчикам инструменты для создания, редактирования и трансляции видео, корпоративные программные решения, а также средства для общения профессионалов в области видео с клиентами и другими профессионалами.

Первоначально сайт был создан Джейком Лодвиком и Заком Кляйном в 2004 году как дополнительный доход от CollegeHumor для обмена юмористическими видео среди коллег, но был отложен в сторону, чтобы поддержать растущую популярность CollegeHumor.

IAC приобрела CollegeHumor и Vimeo в 2006 году, а после того, как Google приобрела YouTube за 1,65 миллиарда долларов США, IAC направил дополнительные усилия на Vimeo, чтобы конкурировать с YouTube, сосредоточившись на предоставлении тщательно подобранного контента и видео высокой четкости, чтобы отличаться от других сайтов обмена видео.

Лодвик и Кляйн в конце концов ушли к 2009 году, и IAC внедрил структуру, более ориентированную на корпорацию, для создания сервисов Vimeo, с нынешним генеральным директором Анджали Судом, работающим с июля 2017 года. В декабре

2020 года IAS объявил о планах выделить Vimeo в качестве отдельного подразделения. публичная компания.

Достоинства:

- Прост в использовании.
- Поддерживается большое количество форматов кодирования видеофайлов.
- Есть возможность бесплатного использования
- Есть мобильное приложения

Недостатки:

- Отсутствует широкая языковая поддержка, что ограничивает пользование клиентов, не владеющих английским на уровне понимания интерфейса программного средства.

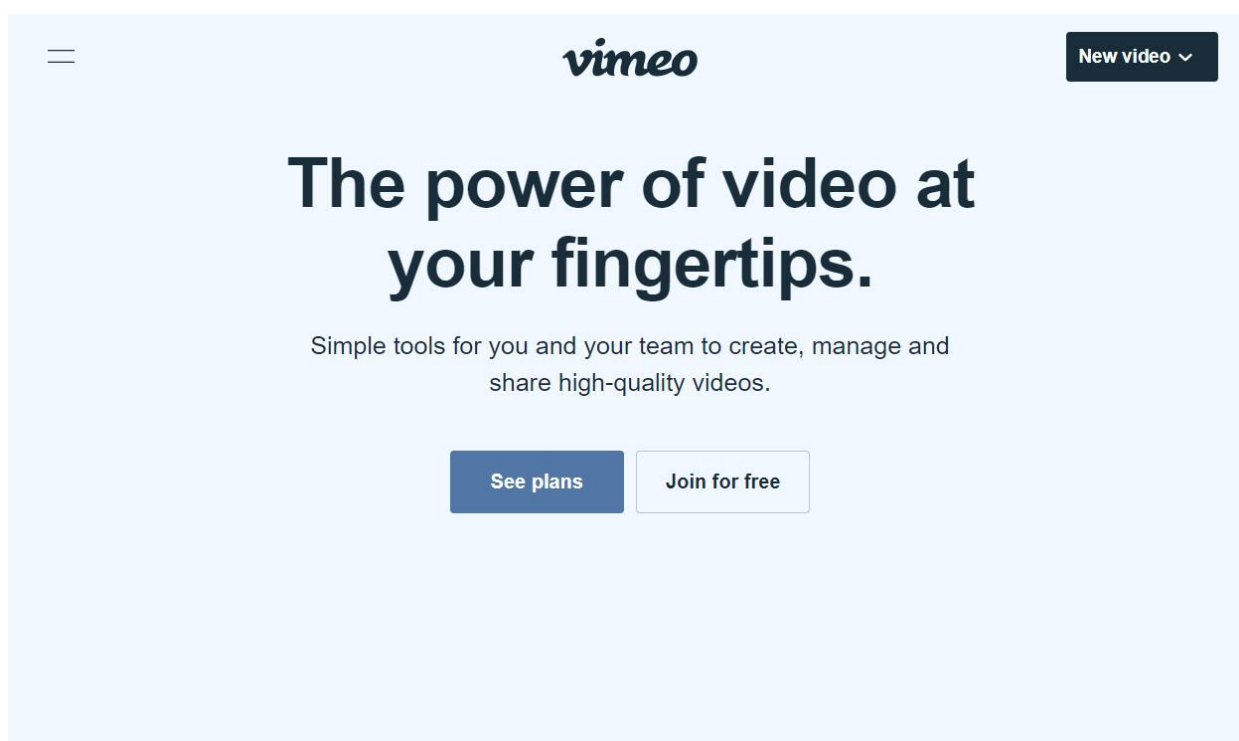


Рисунок 1.3 – Скриншот главной страницы «Vimeo»

1.2 Постановка задачи

После анализа прототипов были сформированы следующие требования к проектированному программному средству:

- Сервер должен иметь возможность передачи данных более чем 1 клиенту;
- Возможность быстрой регистрации пользователя;
- Сервер должен предоставлять список возможных к просмотру видео;
- Пользователь должен иметь возможность ставить лайки;
- Пользователь должен иметь возможность удалять видео;
- Пользователь должен иметь возможность смотреть общую информацию о своих действиях.

2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

2.1 Протокол TCP

Transmission Control Protocol (TCP) - это один из основных протоколов, предназначенный для управления передачей данных в сетях и подсетях TCP/IP. Обеспечивает гарантированную доставку как одиночных сообщений, так и потоков данных, сборку фрагментов, квитирование, повторную передачу пакета и установку приоритетов. Выполняет функции протокола транспортного уровня в стеке протоколов TCP/IP. TCP поддерживает только однонаправленную связь и отправку из 1 узла в 1 узел. Это также называется связью «точка-точка»

Бит/Смещение	0 - 3	4 - 9	10 - 15	16 - 31
0	Порт источника			Порт назначения
32	Номер последовательности (порядковый номер)			
64	Номер подтверждения			
96	Длина заголовка	Зарезервировано	Флаги	Размер окна
128	Контрольная сумма			Указатель срочных данных
160	Опции			
160/192+	Данные			

Рисунок 2.1 – Структура пакета TCP

Взаимодействие партнеров с использованием протокола TCP строится в три этапа: установление логического соединения, обмен данными, закрытие соединения.

Перед началом передачи полезных данных, TCP позволяет убедиться в том, что получатель существует, слушает нужный отправителю порт и готов принимать данные для этого устанавливается соединение при помощи механизма трёхстороннего рукопожатия, схема которого приведена ниже.

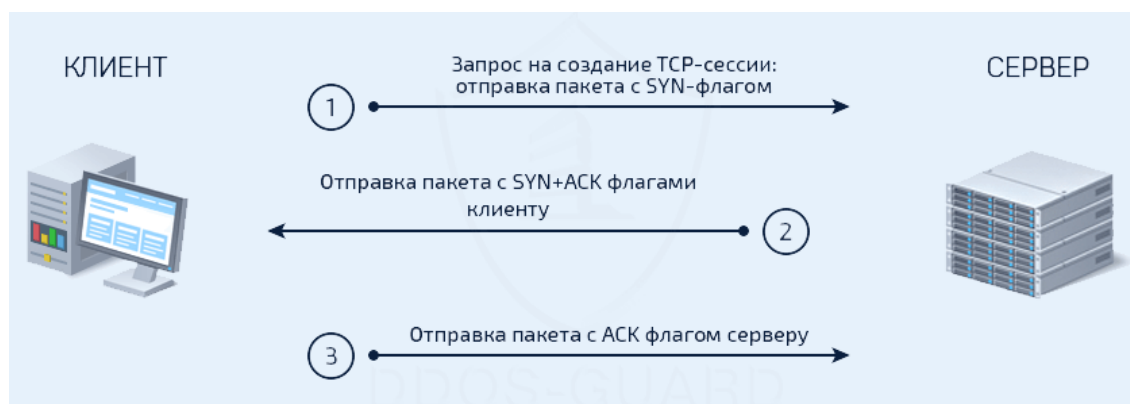


Рисунок 2.2 – Схема TCP соединения

Далее, в рамках соединения передаются пользовательские данные. В большинстве случаев каждый ТСР-сегмент пересылается в одной IP-дейтаграмме. Однако при необходимости ТСР будет расщеплять сегменты на несколько IP-дейтаграмм, вмещающихся в физические кадры данных, которые используют для передачи информации между компьютерами в сети. Поскольку IP не гарантирует, что дейтаграммы будут получены в той же самой последовательности, в которой они были посланы, ТСР осуществляет повторную сборку ТСР-сегментов на другом конце маршрута, чтобы образовать непрерывный поток данных.

При пересылке данных используется понятие окна ТСР. Окно ТСР – это общий объем данных, которые могут быть отправлены и не подтверждены. Размер окна изменяется динамически прямо во время работы и зависит от условий сети. Чем больше размер окна, тем больший объем информации будет передан до получения подтверждения. Для надёжных сетей подтверждения можно присылать редко, чтобы не добавлять трафика, поэтому размер окна в таких сетях автоматически увеличивается. Если же ТСР видит, что данные теряются, размер окна автоматически уменьшается.

После завершения передачи соединение закрывается, тем самым получатель извещается о том, что данных больше не будет, а отправитель извещается о том, что получатель извещён.

Отличие упорядоченного закрытия соединения от его разрыва заключается в том, что при упорядоченном закрытии соединения данные, находящиеся в пути, будут ожидаться пользователем и только потом соединение будет закрыто. При разрыве соединения все недошедшие данные теряются.

2.2 Протокол HTTP

Протокол передачи гипертекста (HTTP) - это протокол прикладного уровня для распределенных, совместных гипермедийных информационных систем. HTTP - это основа передачи данных во всемирной паутине, где гипертекстовые документы включают гиперссылки на другие ресурсы, к которым пользователь может легко получить доступ, например, щелчком мыши или касанием экрана в веб-браузере.

HTTP функционирует как протокол запроса-ответа в вычислительной модели клиент-сервер. Например, веб-браузер может быть клиентом, а приложение, работающее на компьютере, на котором размещен веб-сайт, может быть сервером. Клиент отправляет на сервер сообщение HTTP-запроса. Сервер, который предоставляет ресурсы, такие как файлы HTML и другое содержимое, или выполняет другие функции от имени клиента, возвращает клиенту ответное сообщение. Ответ содержит информацию о статусе запроса, а также может содержать тело сообщения.

Сообщение запроса состоит из следующего:

- Строка запроса (например, GET /images/logo.png HTTP / 1.1, который запрашивает ресурс с именем /images/logo.png с сервера);
- Поля заголовка запроса (например, Accept-Language: en);
- Пустая строка;
- Необязательное тело сообщения.

Строка запроса и другие поля заголовка должны заканчиваться на <CR> <LF> (то есть символ возврата каретки, за которым следует символ перевода строки). Пустая строка должна состоять только из <CR> <LF> и никаких других пробелов. В протоколе HTTP / 1.1 все поля заголовка, кроме Host, являются необязательными.

Строка запроса, содержащая только имя пути, принимается серверами для обеспечения совместимости с HTTP-клиентами до спецификации HTTP / 1.0 в RFC 1945.

HTTP определяет методы (иногда называемые глаголами, но нигде в спецификации не упоминается глагол, а OPTIONS или HEAD не является глаголом), чтобы указать желаемое действие, которое должно быть выполнено над идентифицированным ресурсом. Что представляет этот ресурс, будь то уже существующие данные или данные, которые генерируются динамически, зависит от реализации сервера. Часто ресурс соответствует файлу или выходным данным исполняемого файла, находящегося на сервере. Спецификация HTTP / 1.0 определила методы GET, HEAD и POST, а спецификация HTTP / 1.1 добавила пять новых методов: OPTIONS, PUT, DELETE, TRACE и CONNECT. Поскольку они указаны в этих документах, их семантика хорошо известна, и на нее можно положиться. Любой клиент может использовать любой метод, а сервер можно настроить для поддержки любой комбинации методов. Если метод неизвестен промежуточному звену, он будет рассматриваться как небезопасный и неидемпотентный метод. Нет ограничений на количество методов, которые могут быть определены, и это позволяет указывать будущие методы без нарушения существующей инфраструктуры. Например, WebDAV определил семь новых методов, а RFC 5789 определил метод PATCH.

Имена методов чувствительны к регистру. Это контрастирует с именами полей заголовка HTTP, которые нечувствительны к регистру.

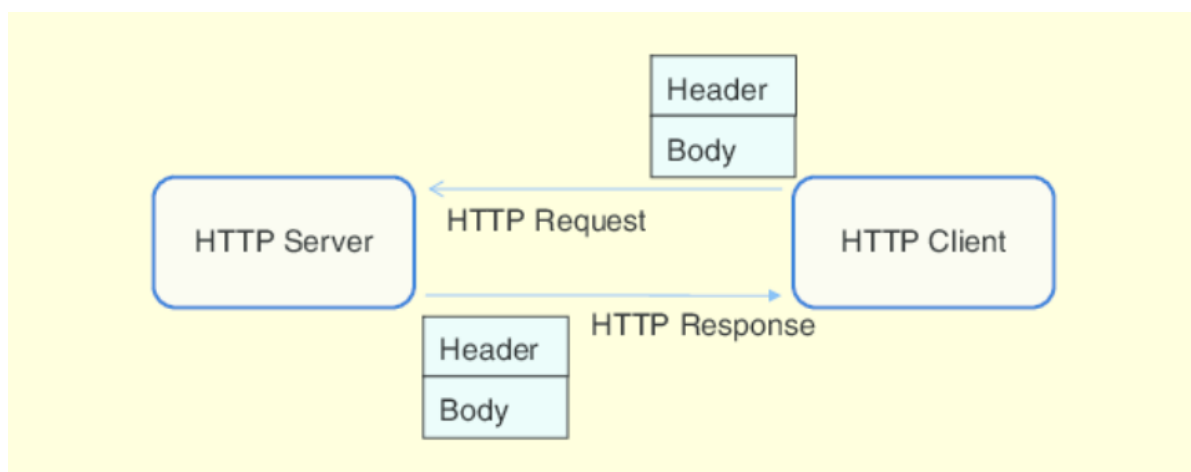


Рисунок 2.3 – Схема работы протокола HTTP

2.3 Библиотека Socket.IO

Socket.IO - это библиотека JavaScript для веб-приложений в реальном времени. Он обеспечивает двустороннюю связь в реальном времени между веб-клиентами и серверами. Он состоит из двух частей: клиентской библиотеки, работающей

в браузере, и серверной библиотеки для Node.js. Оба компонента имеют почти идентичный API. Как и Node.js, он управляется событиями.

Socket.IO в основном использует протокол WebSocket с опросом в качестве запасного варианта, обеспечивая при этом тот же интерфейс.

Хотя его можно использовать просто как оболочку для WebSocket, он предоставляет гораздо больше возможностей, включая широковещательную рассылку на несколько сокетов, хранение данных, связанных с каждым клиентом, и асинхронный ввод-вывод.

Socket.IO предоставляет возможность реализовать аналитику в реальном времени, двоичную потоковую передачу, обмен мгновенными сообщениями и совместную работу с документами.

Socket.IO прозрачно обрабатывает соединение. Если возможно, он автоматически обновится до WebSocket. Для этого программист должен обладать только знаниями Socket.IO.

Socket.IO не является библиотекой WebSocket с возможностью перехода к другим протоколам реального времени. Это настраиваемая реализация транспортного протокола реального времени поверх других протоколов реального времени. Сервер, реализующий Socket.IO, не может подключиться к клиенту WebSocket, отличному от Socket.IO. Клиент, реализующий Socket.IO, не может взаимодействовать с сервером WebSocket, отличным от Socket.IO, или сервером Long Polling Comet. Socket.IO требует использования библиотек Socket.IO как на стороне клиента, так и на стороне сервера.

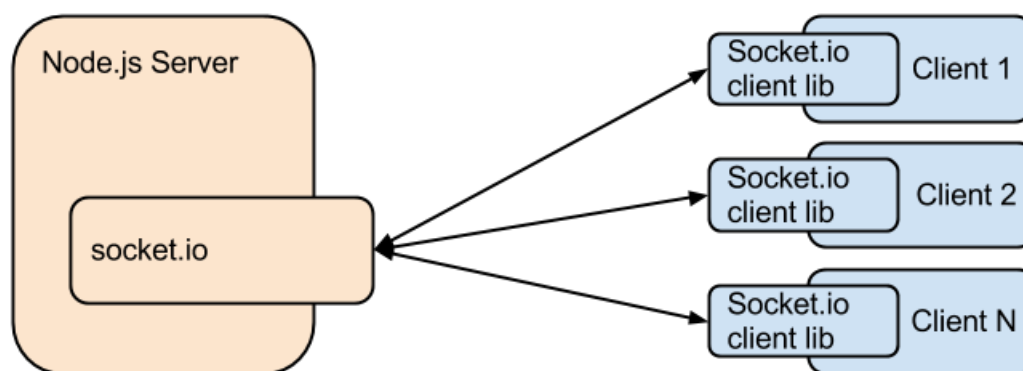


Рисунок 2.4 – Схема работы Socket.IO

2.4 Формат мультимедиа MP4

MP4 - это формат цифрового мультимедийного контейнера, наиболее часто используемый для хранения видео и аудио, но его также можно использовать для хранения других данных, таких как субтитры и неподвижные изображения. Как и большинство современных форматов контейнеров, он позволяет осуществлять потоковую передачу через Интернет. Единственное расширение имени файла для файлов MPEG-4 Part 14, как определено в спецификации, - .mp4. MPEG-4 Part 14 (формально ISO / IEC 14496-14: 2003) - это стандарт, определенный как часть MPEG-4.

Большинство типов данных могут быть встроены в файлы MPEG-4 Part 14 через частные потоки. Для включения потоковой информации в файл используется отдельная подсказка. Зарегистрированные кодеки для файлов на основе MPEG-4 Part 12 публикуются на веб-сайте органа регистрации MP4 (mp4ra.org), но большинство из них не широко поддерживаются проигрывателями MP4.

Широко поддерживаемые кодеки и дополнительные потоки данных:

Видео: MPEG-H, часть 2 (H.265 / HEVC), MPEG-4, часть 10 (H.264 / AVC) и MPEG-4, часть 2. Реже используются другие форматы сжатия: MPEG-2 и MPEG-1.

Аудио: расширенное кодирование звука. Также аудиообъекты MPEG-4 Part 3, такие как Audio Lossless Coding (ALS), Scalable Lossless Coding (SLS), MP3, MPEG-1 Audio Layer II (MP2), MPEG-1 Audio Layer I (MP1), CELP, HVXC (речь), TwinVQ, интерфейс преобразования текста в речь (TTSI) и язык структурированного аудио оркестра (SAOL)

Другие форматы сжатия используются реже: Apple Lossless, Free Lossless Audio Codec (добавлен в конце 2018 года) и Opus (добавлен в конце 2018 года).

Субтитры: синхронизированный текст MPEG-4 (также известный как синхронизированный текст 3GPP).

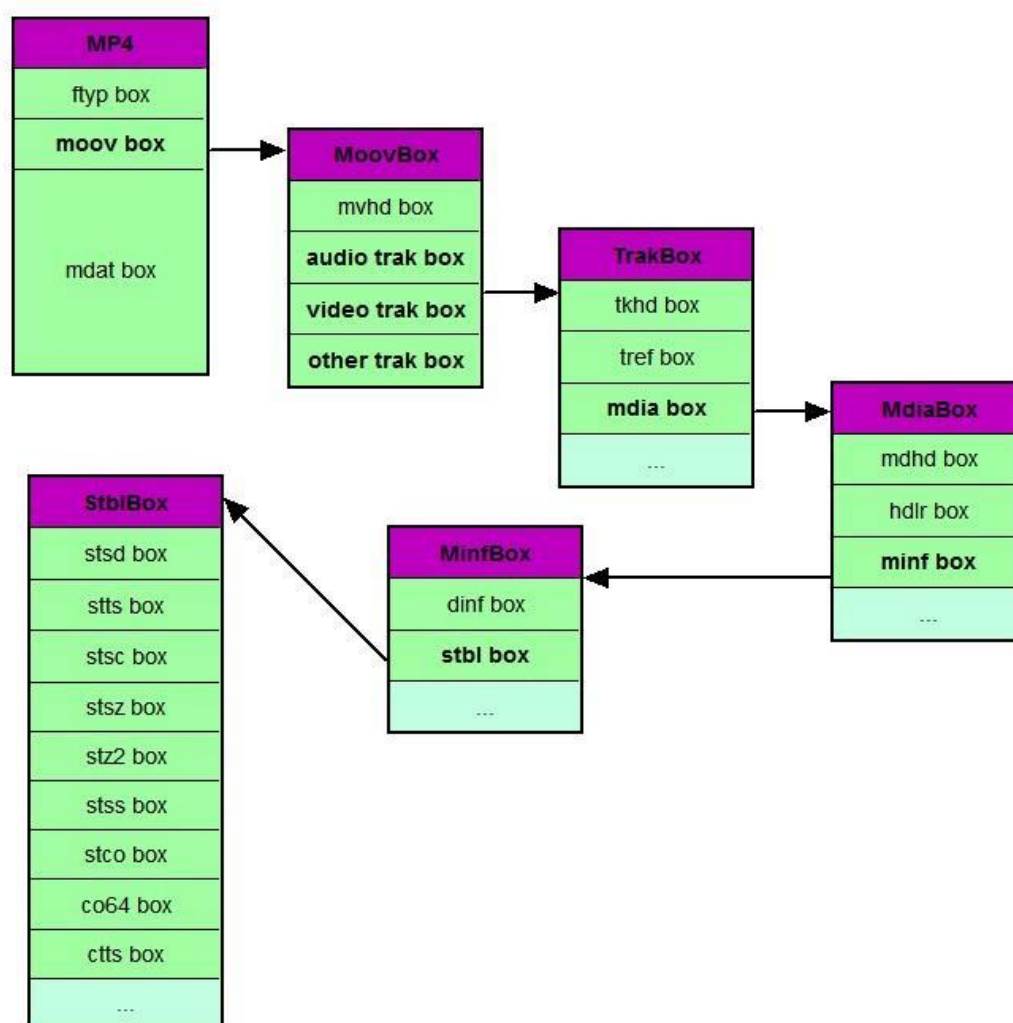


Рисунок 2.5 – Схема структуры файла MP4

2.5 Токен Json Web Token

JSON Web Token - это предлагаемый Интернет-стандарт для создания данных с дополнительной подписью и / или дополнительным шифрованием, полезная нагрузка которого содержит JSON, который утверждает некоторое количество утверждений.

Токены подписываются с использованием личного секрета или открытого / закрытого ключа. Например, сервер может сгенерировать токен с утверждением «зарегистрировано как администратор» и предоставить его клиенту. Затем клиент может использовать этот токен, чтобы доказать, что он вошел в систему как администратор.

Токены могут быть подписаны закрытым ключом одной стороны (обычно сервером), чтобы эта сторона могла впоследствии проверить, является ли токен законным. Если другая сторона каким-либо подходящим и заслуживающим доверия способом владеет соответствующим открытым ключом, она также может проверить легитимность токена. Маркеры разработаны так, чтобы быть компактными, безопасными для URL-адресов и использоваться, особенно в контексте единого входа в веб-браузере (SSO).

Утверждения JWT обычно могут использоваться для передачи удостоверений аутентифицированных пользователей между поставщиком удостоверений и поставщиком услуг или любых других типов утверждений в соответствии с требованиями бизнес-процессов.



Рисунок 2.6 – Схема структуры Json Web Token

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Приложение состоит из 2 частей: серверной и клиентской. Серверная часть создана с использованием Node.js и Socket.IO. Клиентская часть создана с использованием React и Socket.IO.

3.1 Создание и проверка Json Web Token

Json Web Token (JWT) используются для авторизации пользователей. Для работы с JWT используется библиотека `jsonwebtoken`. При любом взаимодействии с сервером клиент отправляет токен. Если токен не валиден, то сервер не выполнит запрошенную операцию. Токен на клиенте хранится в Cookies.

В токене можно зашифровать любую текстовую информацию (например, `id` пользователя). Для создания токена требуется указать ключ и его время жизни.

```
function createToken(user) {  
  return jwt.sign(  
    {userId: user.id},  
    config.get('jwtSecret'),  
    {expiresIn: '1h'}  
  )  
}
```

Рисунок 3.1 – Создание Json Web Token

Для проверки токена требуется ключ, которым он был подписан при создании. Если был предоставлен верный ключ, то происходит проверка его времени жизни.

```
function verifyJwt(token) {  
  try {  
    if (!token) {  
      return null  
    }  
    return jwt.verify(token, config.get('jwtSecret'))  
  } catch {  
    return null  
  }  
}
```

Рисунок 3.2 – Проверка Json Web Token

3.2 Схемы Mongoose

Для хранения данных используется MongoDB. Были созданы схемы для пользователя и видео.

Схема пользователя содержит email, хэш пароля (bcrypt) и массив видео, которым пользователь поставил лайк.

```
const schema = new Schema({
  email: {
    type: String,
    required: true,
    unique: true
  },
  hashedPassword: {
    type: String,
    required: true
  },
  likedVideoIds: [{
    type: Types.ObjectId,
    ref: 'Video'
  }]
})
```

Рисунок 3.3 – Схема пользователя

Схема видео содержит название видео, название файла, id автора, флаг, указывающий на то, обрабатывается ли видео в данный момент времени и количество лайков.

```
const schema = new Schema({
  title: {
    type: String,
    required: true
  },
  fileName: {
    type: String,
    required: true
  },
  authorId: {
    type: Types.ObjectId,
    ref: 'User',
    required: true
  },
  isProcessing: {
    type: Boolean,
    default: true
  },
  likes: {
    type: Number,
    default: 0
  }
})
```

Рисунок 3.4 – Схема видео

3.3 Алгоритм регистрации

Для регистрации нового пользователя необходимо отправить сообщение `sign_up` с полями `email` и `password` нового пользователя.

Сервер проверяет наличие пользователя с переданным `email` в базе данных, если он существует, то на клиент возвращается сообщение об ошибке. Если пользователя не существует, то вычисляется хэш пароля и создается новый пользователь с переданным `email` и вычисленным хэшем пароля. После сохранения в БД создается JWT токен, соответствующий новому пользователю и отправляется на клиент.

```
socket.on('sign_up', async (data) => {
  const {email, password} = data

  if (await User.findOne({email})) {
    socket.emit('auth_result', { error: 'User with that email already exists' })
    return
  }

  const hashedPassword = await bcrypt.hash(password, 12)
  const user = new User({
    email,
    hashedPassword
  })

  await user.save()

  const jwt = createToken(user)

  socket.emit('auth_result', { jwt })
})
```

Рисунок 3.5 – Алгоритм регистрации пользователя

3.4 Алгоритм загрузки видео на сервер

Для загрузки видео клиент должен отправить название выкладываемого видео и файл на сервер с использованием `multipart/form-data`. На сервере отправленный файл сохраняется со следующей конфигурацией хранилища:

```
const storageConfig = multer.diskStorage({
  destination: (req, file, cb) =>{
    cb(null, config.get('videosRoot'));
  },
  filename: (req, file, cb) =>{
    cb(null, 'source' + '-' + Date.now() + '-' + Math.random() + path.extname(file.originalname));
  }
});
```

Рисунок 3.6 – Конфигурация хранилища Multer

При успешной загрузке видео будет произведена конвертация формата файла

в MP4, а также будет изменено разрешение файла для экономии места на сервере с использованием ffmpeg.

```
const process = new ffmpeg(fileToProcessPath);
process.then(async function (ffvideo) {
  ffvideo
    .setVideoFormat('mp4')
    .setVideoSize('?x480', true, true, '#ffffff')
    .save(processedFilePath, async function (error, file) {
      if (!error) {

        video.fileName = processedFileName
        video.isProcessing = false

        await Video.findByIdAndUpdate(video._id, {
          isProcessing: false,
          fileName: processedFileName
        }, {
          upsert: true,
          useFindAndModify: false
        })

        if (config.get('shouldDeleteUnneededVideoFilesImmediately')) {
          fs.unlink(fileToProcessPath, function(err){
            if(err) return console.log(err);
            console.log(fileToProcessPath + ' deleted successfully');
          });
        }

        res.json({
          message: "Successfully uploaded file"
        })
      } else {
        console.log(error);
        res.status(500).json({
          message: "Something went wrong..."
        })
      }
    })
});
}, function (err) {
  console.log(err);
  res.status(500).json({
    message: "Something went wrong..."
  })
});
});
```

Рисунок 3.7 – Применение ffmpeg к выложенному видео

3.5 Алгоритм отправки видео на клиент

Стандартный HTML плеер принимает MP4 файлы, отправленные чанками. Был реализован алгоритм, вырезающий нужный чанк из запрошенного файла и отправляющий его на клиент.

```
const video = await Video.findById(req.params.id)
const filePath = path.join(config.get('videosRoot'), video.fileName)
if (fs.existsSync(filePath)) {
  const stat = fs.statSync(filePath)
  const fileSize = stat.size
  const range = req.headers.range
  if (range) {
    const parts = range.replace(/bytes=/, "").split("-")
    const start = parseInt(parts[0], 10)
    const end = parts[1]
      ? parseInt(parts[1], 10)
      : fileSize - 1
    const chunksize = (end - start) + 1
    const file = fs.createReadStream(filePath, {start, end})
    const head = {
      'Content-Range': `bytes ${start}-${end}/${fileSize}`,
      'Accept-Ranges': 'bytes',
      'Content-Length': chunksize,
      'Content-Type': 'video/mp4',
    }
    res.writeHead(206, head)
    file.pipe(res)
  } else {
    const head = {
      'Content-Length': fileSize,
      'Content-Type': 'video/mp4',
    }
    res.writeHead(200, head)
    fs.createReadStream(filePath).pipe(res)
  }
} else {
  res.sendStatus(404);
}
```

Рисунок 3.8 – Алгоритм отправки видео на клиент

4 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

В результате разработки программного средства было проведено тестирование его функционала. Тестирование онлайн-сервиса «Видеохостинг» производилось на персональном компьютере с установленной операционной системой MacOS с использованием ручного тестирования. Была составлена таблица 4.1, отображающая ожидаемые и реальные результаты тестируемых функциональностей.

4.1 Тестирование функционала программы

Таблица 4.1 – Тестирование функционала программы

Номер теста	Тестируемая функциональность	Ожидаемый результат	Результат теста
1	Регистрация нового пользователя	После нажатия кнопки Sign Up был произведен переход на домашнюю страницу со списком видео	Тест пройден
2	Регистрация уже зарегистрированного пользователя	Появилось сообщение “User with that email already exists”	Тест пройден
3	Авторизация с неправильными учетными данными	Появилось сообщение “Invalid email or password”	Тест пройден
4	Авторизация существующего пользователя	После нажатия кнопки Sign In был произведен переход на домашнюю страницу со списком видео	Тест пройден
5	Добавление видео	После нажатия кнопки Upload появился Loader. После пропадания Loader и перехода на главную страницу видео появилось в списке	Тест пройден
6	Просмотр видео	После нажатия на видео произошел переход на новую страницу и появился плеер	Тест пройден
7	Лайк видео	После нажатия на кнопку лайка счетчик лайков увеличился	Тест пройден

Продолжение таблицы 4.1

Номер теста	Тестируемая функциональность	Ожидаемый результат	Результат теста
8	Убрать лайк с видео	После нажатия на кнопку лайка счетчик лайков уменьшился	Тест пройден
9	Удалить видео	После нажатия на кнопку удаления был произведен переход на главную страницу и видео пропало из списка	Тест пройден
10	Выход из учетной записи	После нажатия на кнопку Sign Out был произведен переход на страницу авторизации	Тест пройден
11	Просмотр списка выложенных видео	После нажатия кнопки Profile произошел переход на новую страницу, где был загружен список и отображен таблицей	Тест пройден
12	Просмотр списка лайкнутых видео	После нажатия кнопки Profile произошел переход на новую страницу, где был загружен список и отображен таблицей	Тест пройден

4.2 Вывод из прохождения тестирования

Разработка приложения велась с использованием системы контроля версий GitHub, позволившая сохранять состояние программы на каждом отдельном этапе по ходу добавления нового функционала или изменения уже существующего. Появление новых точек возврата происходит посредством группировки изменённых файлов, а затем они объединяются под общим именем «коммита», в котором кратко изложена суть изменений.

Приложение успешно прошло все тесты, что показывает корректность работы разработанного программного средства и соответствие функциональным требованиям.

5 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

5.1 Руководство по установке

Для запуска сервера потребуется установить на компьютер Node.js. Для запуска клиента требуется любой браузер последней версии.

5.2 Руководство по использованию

При первом использовании видеохостинга будет показана страница авторизации, на которой требуется ввести почту и пароль, после чего нажать кнопку Sign Up.

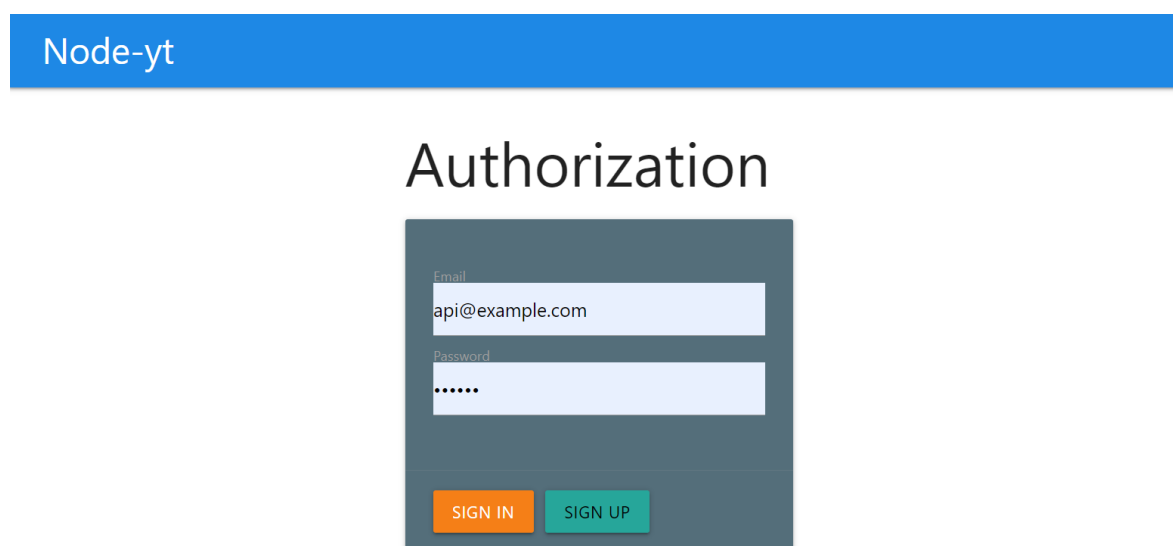


Рисунок 5.1 – Страница авторизации

После успешного входа будет показана главная страница со списком всех доступных видео

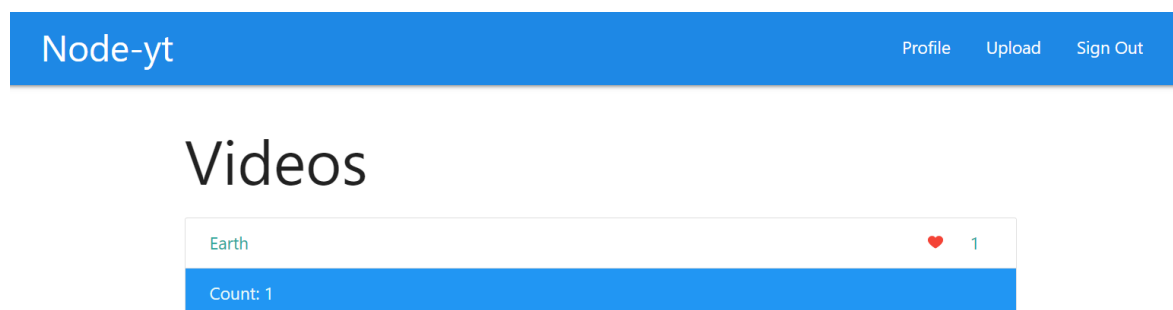


Рисунок 5.2 – Главная страница

Для просмотра видео требуется нажать на него в списке

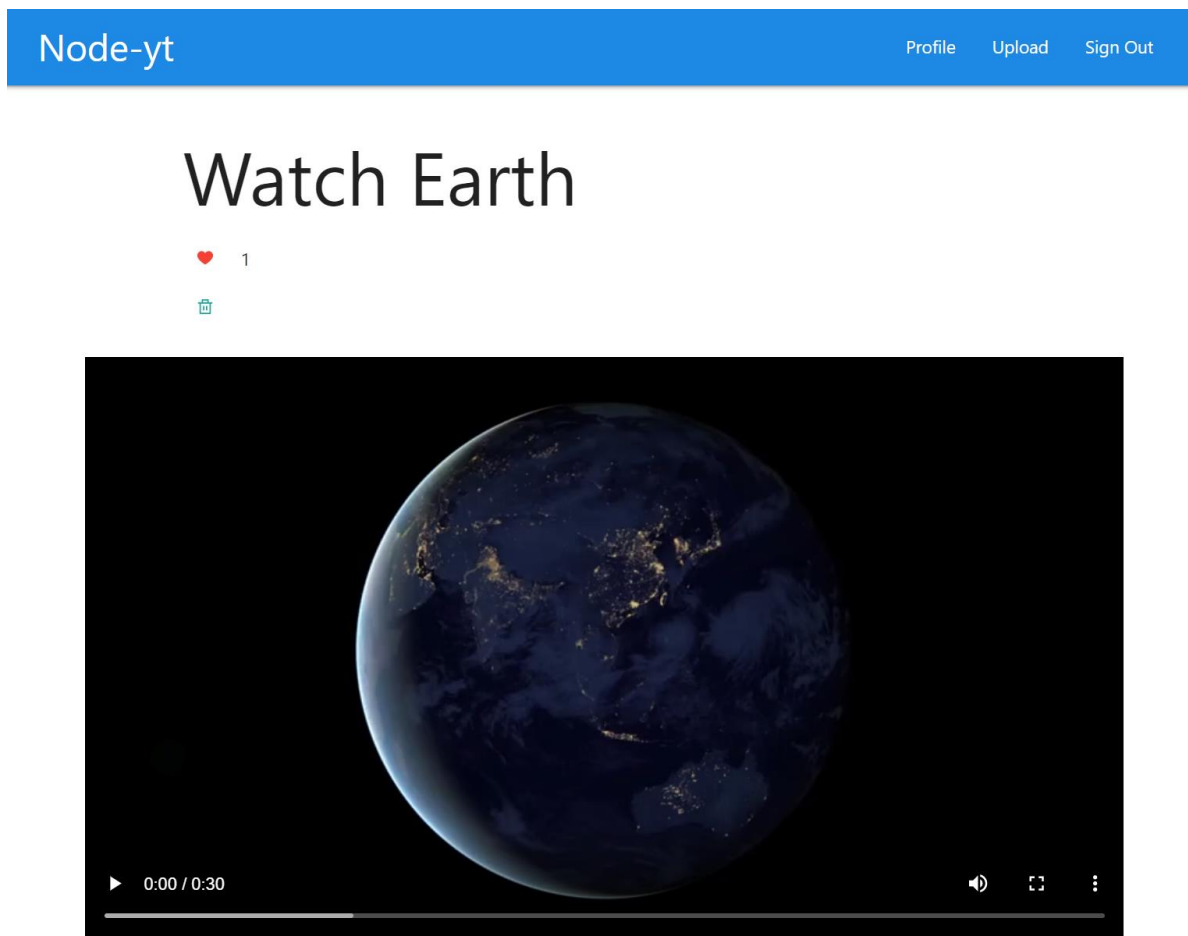


Рисунок 5.3 – Страница просмотра видео

Чтобы выложить видео, требуется нажать на кнопку Upload и заполнить все необходимые поля, после чего нажать Upload и дождаться завершения загрузки.

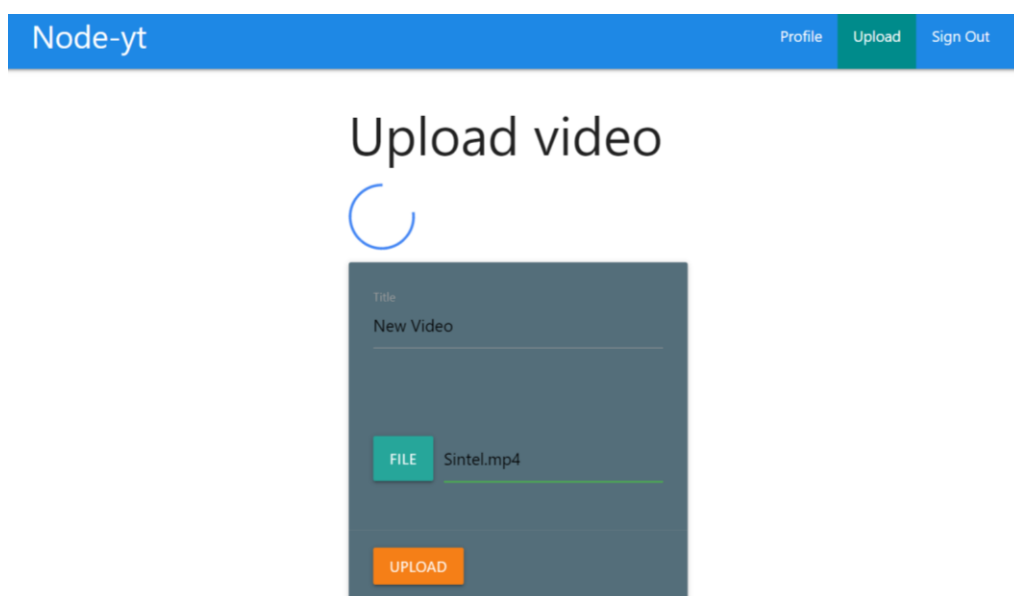


Рисунок 5.4 – Страница загрузки видео

После завершения загрузки можно перейти на главную страницу и увидеть только что добавленное видео

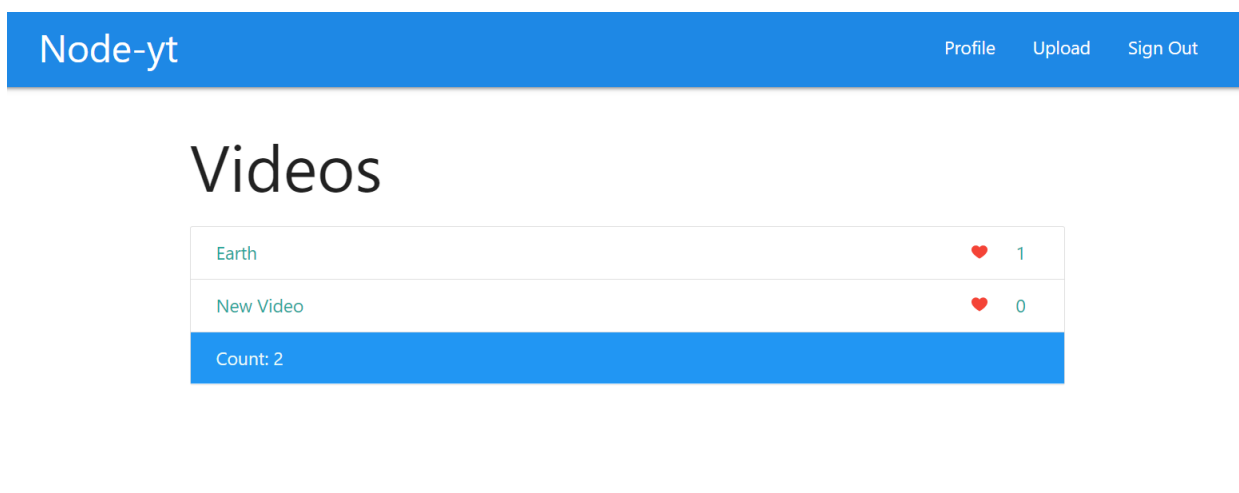


Рисунок 5.5 – Главная страница

Для удаления видео требуется нажать на кнопку удаления на странице просмотра видео, а для лайка – кнопку лайка.



Рисунок 5.6 – Дополнительные действия на странице просмотра

Для выхода из пользователя требуется нажать кнопку Sign Out в правом верхнем углу главной страницы

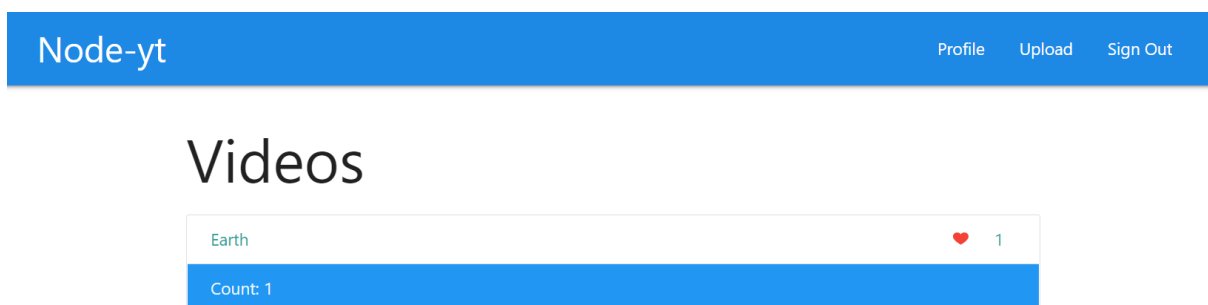


Рисунок 5.7 – Главная страница

ЗАКЛЮЧЕНИЕ

Таким образом, в результате проделанной работы был создан сервер и клиент, позволяющие пользователю совершить регистрацию, авторизацию, просмотр видео онлайн, поставить лайк, удалить видео, загрузить видео любого формата и посмотреть статистику о себе.

При использовании правильного API присутствует возможность использования сервера не только с написанным клиентом.

В ходе разработки был более глубоко изучен язык программирования JavaScript, протоколы стека TCP/IP, в частности были усовершенствованы знания по протоколам TCP, HTTP, HTTPS, также была изучена библиотека Socket.IO.

Было произведено тестирование серверной и клиентской частей приложения, которое показало полное соответствие разработанного приложения спецификации требований. Разработано руководство пользователя.

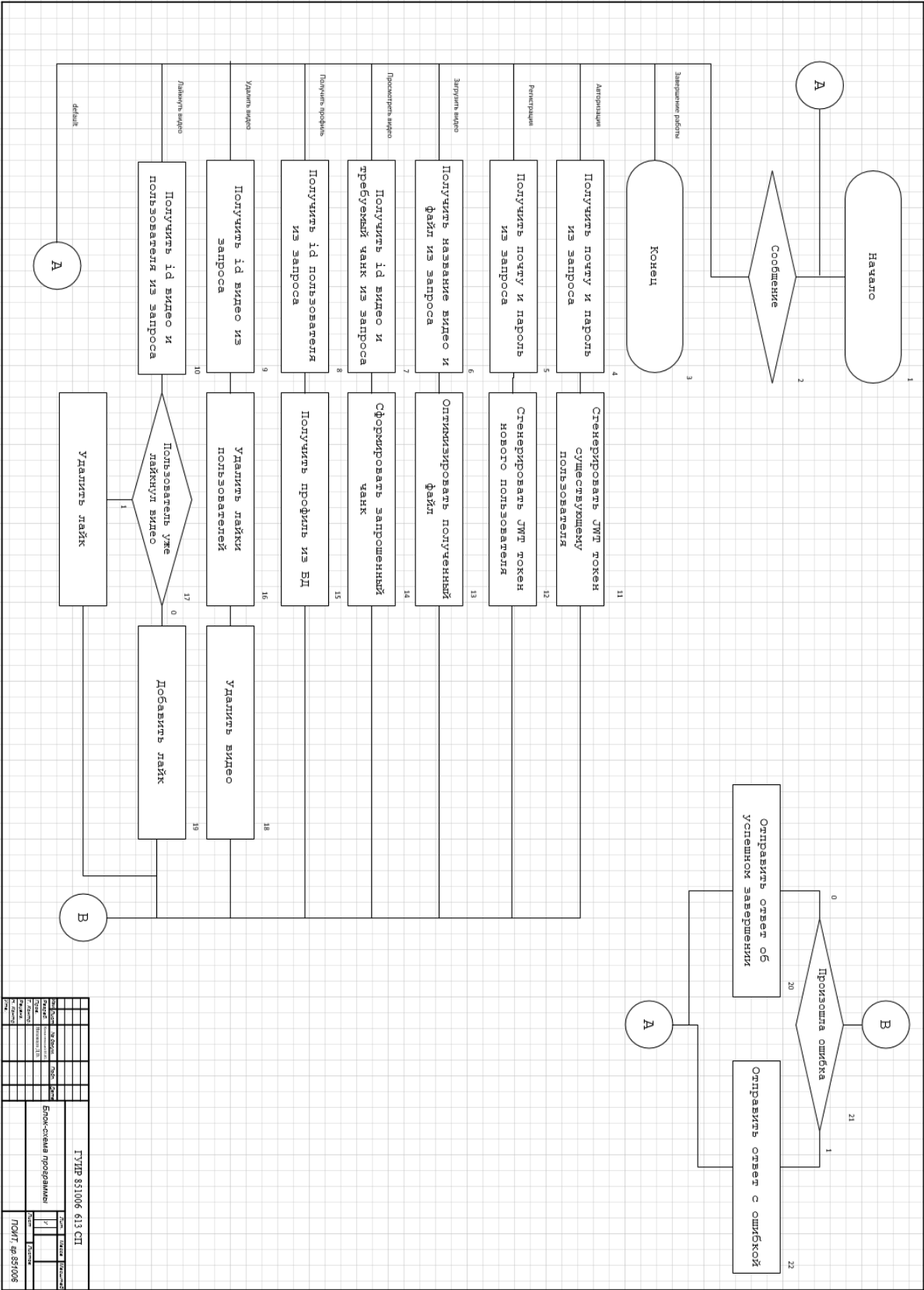
Суммируя вышесказанное, можно сказать, что разработанное программное средство обладает всеми заявленными требованиями и представляет полноценный функциональный продукт, доступный для широкого пользователя.

СПИСОК ЛИТЕРАТУРЫ

- [1] O'Reilly Media «Web Development with Node and Express: Leveraging the JavaScript Stack 2nd Edition» (2019) – 346 с.
- [2] CreateSpace Independent Publishing Platform «JSON Web Token: Third Edition» (2018) – 134 с.
- [3] Stoyan Stefanov «Object-Oriented JavaScript» (2013) – 358 с.
- [4] Stoyan Stefanov «JavaScript Patterns: Build Better Applications with Coding and Design Patterns» (2010) – 236 с.
- [5] developer.mozilla.org [Электронный портал]. – Электронные данные. – Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- [6] docs.mongodb.com [Электронный портал]. – Электронные данные. – Режим доступа: <https://docs.mongodb.com/manual/>
- [7] reactjs.org [Электронный портал]. – Электронные данные. – Режим доступа: <https://reactjs.org/docs/getting-started.html>
- [8] nodejs.org [Электронный портал]. – Электронные данные. – Режим доступа: <https://nodejs.org/en/docs/>
- [9] github.com [Электронный портал]. – Электронные данные. – Режим доступа: <https://github.com/azat-io/you-dont-know-js-ru>

ПРИЛОЖЕНИЯ

Приложение 1 (схема алгоритма программы на А1)



Приложение 2 (Код программы)

app.js

```
const express = require('express')
const socket = require('socket.io')
const config = require('config')
const mongoose = require('mongoose')
const cookieParser = require('cookie-parser');
const logger = require('morgan');
const User = require('./models/User')
const Video = require('./models/Video')
const bcrypt = require('bcryptjs')

const app = express()
const cors = require('cors')
const jwt = require('jsonwebtoken');
const fs = require("fs");
const path = require("path");

app.use(cors())
app.use(cookieParser());
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.use('/api/videos', require('./routes/videos.routes'))
app.use('/api/upload', require('./routes/upload.routes'))

const PORT = config.get('port') || 5000

function createToken(user) {
  return jwt.sign(
    {userId: user.id},
    config.get('jwtSecret'),
    {expiresIn: '1h'}
  )
}

function verifyJwt(token) {
  try {
    if (!token) {
      return null
    }
    return jwt.verify(token, config.get('jwtSecret'))
  } catch {
    return null
  }
}

async function start() {
  try {
    let mongoUri = config.get('mongoUri')
    console.log(`Connecting to MongoDB with uri ${mongoUri}`)

    await mongoose.connect(mongoUri, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
      useCreateIndex: true
    })
  }
}
```

```

const server = app.listen(PORT, () => {
  console.log(`App was started on port ${PORT}`)
})

const io = socket(server)
io.on('connection', (socket) => {
  console.log('User connected with id = ' + socket.id)

  socket.on('restore_auth', (data) => {
    if (data) {
      let {jwt} = data
      if (verifyJwt(jwt)) {
        socket.emit('auth_result', { jwt })
      } else {
        socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
      }
    } else {
      socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
    }
  })

  socket.on('sign_up', async (data) => {
    const {email, password} = data

    if (await User.findOne({email})) {
      socket.emit('auth_result', { error: 'User with that email already exists' })
      return
    }

    const hashedPassword = await bcrypt.hash(password, 12)
    const user = new User({
      email,
      hashedPassword
    })

    await user.save()

    const jwt = createToken(user)

    socket.emit('auth_result', { jwt })
  })

  socket.on('sign_in', async (data) => {
    const {email, password} = data

    const user = await User.findOne({email})
    if (!user) {
      socket.emit('auth_result', { error: "User does not exist" })
      return
    }

    const isMatch = await bcrypt.compare(password, user.hashedPassword)
    if (!isMatch) {
      socket.emit('auth_result', { error: "Invalid password" })
      return
    }

    const jwt = createToken(user)

    socket.emit('auth_result', { jwt })
  })

  socket.on('get_videos_list', async (data) => {

```

```

    if (data) {
      let {jwt} = data
      if (verifyJwt(jwt)) {
        let videos = await Video.find({})
        socket.emit('get_videos_list_result', { videos })
      } else {
        socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
      }
    } else {
      socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
    }
  })

socket.on('get_video', async (data) => {
  if (data) {
    let {videoId, jwt} = data
    if (verifyJwt(jwt)) {
      let videos = await Video.find({_id: videoId})
      socket.emit('get_video_result', { videos })
    } else {
      socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
    }
  } else {
    socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
  }
})

socket.on('delete_video', async (data) => {
  if (data) {
    let {id, jwt} = data
    if (verifyJwt(jwt)) {
      const video = await Video.findById(id)
      const videoFilePath = path.join(config.get('videosRoot'), video.fileName)

      const users = await User.find({likedVideoIds: id})

      for (var i = 0; i < users.length; i++) {
        const user = users[i]

        var likedVideosByUser = user.likedVideoIds
        likedVideosByUser.splice(likedVideosByUser.indexOf(id), 1)

        await User.findByIdAndUpdate(user._id, {
          likedVideoIds: likedVideosByUser
        }, {
          upsert: true,
          useFindAndModify: false
        })
      }

      await Video.findOneAndDelete({_id: id})

      if (config.get('shouldDeleteUnneededVideoFilesImmediately')) {
        fs.unlink(videoFilePath, function(err){
          if(err) return console.log(err);
          console.log(videoFilePath + ' deleted successfully');
        });
      }

      socket.emit('delete_video_result', { })
    } else {
      socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
    }
  }
})

```

```

    } else {
      socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
    }
  })

socket.on('video_like', async (data) => {
  if (data) {
    let { id, jwt } = data
    let decoded = verifyJwt(jwt)
    if (decoded) {
      try {
        const user = await User.findById(decoded.userId)
        const video = await Video.findById(id)

        var likedVideosByUser = user.likedVideoIds
        var videoLikesCount = video.likes

        const index = likedVideosByUser.indexOf(id);
        if (index > -1) {
          likedVideosByUser.splice(index, 1);
          videoLikesCount--
        } else {
          likedVideosByUser.push(id)
          videoLikesCount++
        }

        await User.findByIdAndUpdate(decoded.userId, {
          likedVideoIds: likedVideosByUser
        }, {
          upsert: true,
          useFindAndModify: false
        })

        await Video.findByIdAndUpdate(id, {
          likes: videoLikesCount
        }, {
          upsert: true,
          useFindAndModify: false
        })

        socket.emit('video_like_result', { likes: videoLikesCount })
      } catch (e) {
        socket.emit('video_like_result', { likes: -1 })
      }
    } else {
      socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
    }
  } else {
    socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
  }
})

socket.on('get_user', async (data) => {
  if (data) {
    let { jwt } = data
    let decoded = verifyJwt(jwt)
    if (decoded) {
      try {
        const user = await User.findById(decoded.userId)
        socket.emit('get_user_result', { user })
      } catch (e) {
        socket.emit('auth_result', { error: 'Invalid user id in jwt' })
      }
    }
  }
})

```



```

    }
  } else {
    socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
  }
} else {
  socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
}
})

socket.on('get_uploaded_videos', async (data) => {
  if (data) {
    let {jwt} = data
    let decoded = verifyJwt(jwt)
    if (decoded) {
      try {
        const videos = await Video.find({ authorId: decoded.userId })
        socket.emit('get_uploaded_videos_result', { videos })
      } catch (e) {
        socket.emit('auth_result', { error: 'Invalid user id in jwt' })
      }
    } else {
      socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
    }
  } else {
    socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
  }
})

socket.on('get_liked_videos', async (data) => {
  if (data) {
    let {jwt} = data
    let decoded = verifyJwt(jwt)
    if (decoded) {
      try {
        const user = await User.findById(decoded.userId)
        const ids = user.likedVideoIds

        const videos = await Video.find({
          '_id': { $in: ids }
        })

        socket.emit('get_liked_videos_result', { videos })
      } catch (e) {
        socket.emit('auth_result', { error: 'Invalid user id in jwt' })
      }
    } else {
      socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
    }
  } else {
    socket.emit('auth_result', { error: 'Unable to verify provided jwt' })
  }
})

socket.on('disconnect', async () => {
  console.log('User disconnected with id = ' + socket.id)
})
})
} catch (e) {
  console.log(`Server error: ${e.message}`)
  process.exit(1)
}
}

```

```
start()
```

upload.routes.js

```
const { Router } = require('express')
const Video = require('../models/Video')
const config = require('config')
const verifyToken = require('../middlewares/verifyToken.middleware')
const multer = require('multer')
const router = Router()
const path = require("path");
const ffmpeg = require("ffmpeg");
const fs = require("fs");

const storageConfig = multer.diskStorage({
  destination: (req, file, cb) =>{
    cb(null, config.get('videosRoot'));
  },
  filename: (req, file, cb) =>{
    cb(null, 'source' + '-' + Date.now() + '-' + Math.random() + path.extname(file.originalname));
  }
});

router.post(
  '/',
  verifyToken,
  async function(req, res) {
    try {
      let upload = multer({ storage: storageConfig }).single('video')
      upload(req, res, async function (err) {
        if (err instanceof multer.MulterError) {
          // A Multer error occurred when uploading.
          console.log(err)
          res.status(500).json({
            message: "Something went wrong..."
          })
          return
        } else if (err) {
          // An unknown error occurred when uploading.
          console.log(err)
          res.status(500).json({
            message: "Something went wrong..."
          })
          return
        }
      })

      console.log('Successfully received file ' + req.file.filename);

      try {
        const fileName = req.file.filename
        const title = req.body.title
        const authorId = req.user.userId
        const video = new Video({
          title,
          fileName,
          authorId
        })

        await video.save()

        const processedFileName = 'video-' + video._id + '.mp4'
```

```

const fileToProcessPath = path.join(config.get('videosRoot'), video.fileName)
const processedFilePath = path.join(config.get('videosRoot'), processedFileName)

console.log('Initiating ffmpeg process for ' + fileToProcessPath);
const process = new ffmpeg(fileToProcessPath);
process.then(async function (ffvideo) {
  ffvideo
    .setVideoFormat('mp4')
    .setVideoSize('?x480', true, true, '#ffffff')
    .save(processedFilePath, async function (error, file) {
      if (!error) {

        video.fileName = processedFileName
        video.isProcessing = false

        await Video.findByIdAndUpdate(video._id, {
          isProcessing: false,
          fileName: processedFileName
        }, {
          upsert: true,
          useFindAndModify: false
        })

        if (config.get('shouldDeleteUnneededVideoFilesImmediately')) {
          fs.unlink(fileToProcessPath, function(err){
            if(err) return console.log(err);
            console.log(fileToProcessPath + ' deleted successfully');
          });
        }

        res.json({
          message: "Successfully uploaded file"
        })
      } else {
        console.log(error);
        res.status(500).json({
          message: "Something went wrong..."
        })
      }
    });
}, function (err) {
  console.log(err);
  res.status(500).json({
    message: "Something went wrong..."
  })
});
} catch (e) {
  console.log(e)
  res.status(500).json({
    message: "Something went wrong..."
  })
}
}
)

```

```
module.exports = router
```

videos.routes.js

```
const { Router } = require('express')
const Video = require('../models/Video')
const config = require('config')
const verifyToken = require('../middlewares/verifyToken.middleware')
const router = Router()
const path = require("path");
const fs = require("fs");

router.get(
  '/file/:id',
  verifyToken,
  async (req, res) => {
    try {
      const video = await Video.findById(req.params.id)
      const filePath = path.join(config.get('videosRoot'), video.fileName)
      if (fs.existsSync(filePath)) {
        const stat = fs.statSync(filePath)
        const fileSize = stat.size
        const range = req.headers.range
        if (range) {
          const parts = range.replace(/bytes=/, "").split("-")
          const start = parseInt(parts[0], 10)
          const end = parts[1]
            ? parseInt(parts[1], 10)
            : fileSize - 1
          const chunksize = (end - start) + 1
          const file = fs.createReadStream(filePath, { start, end })
          const head = {
            'Content-Range': `bytes ${start}-${end}/${fileSize}`,
            'Accept-Ranges': 'bytes',
            'Content-Length': chunksize,
            'Content-Type': 'video/mp4',
          }
          res.writeHead(206, head);
          file.pipe(res);
        } else {
          const head = {
            'Content-Length': fileSize,
            'Content-Type': 'video/mp4',
          }
          res.writeHead(200, head)
          fs.createReadStream(filePath).pipe(res)
        }
      } else {
        res.sendStatus(404);
      }
    } catch (e) {
      console.log(e)
      res.status(500).json({
        message: "Something went wrong..."
      })
    }
  }
)

module.exports = router
```

User.js

```
const { Schema, model, Types } = require('mongoose')

const schema = new Schema({
  email: {
    type: String,
    required: true,
    unique: true
  },
  hashedPassword: {
    type: String,
    required: true
  },
  likedVideoIds: [{
    type: Types.ObjectId,
    ref: 'Video'
  }]
})

module.exports = model('User', schema)
```

Video.js

```
const { Schema, model, Types } = require('mongoose')

const schema = new Schema({
  title: {
    type: String,
    required: true
  },
  fileName: {
    type: String,
    required: true
  },
  authorId: {
    type: Types.ObjectId,
    ref: 'User',
    required: true
  },
  isProcessing: {
    type: Boolean,
    default: true
  },
  likes: {
    type: Number,
    default: 0
  }
})

module.exports = model('Video', schema)
```

verifyToken.middleware.js

```
const jwt = require('jsonwebtoken')
const config = require('config')

const verifyToken = async (req, res, next) => {
```

```

try {

  const token = req.cookies['jwt']
  if (!token) {
    res.status(401).send({ message: 'Not authorized' })
    return
  }

  const decoded = jwt.verify(token, config.get('jwtSecret'))

  req.user = {
    userId: decoded.userId
  }

  next()
} catch {
  res.status(401).send({ message: 'Not authorized' })
}
}

module.exports = verifyToken

```

App.js

```

import React from 'react'
import {BrowserRouter as Router} from "react-router-dom";
import {useRoutes} from "./routes";
import 'materialize-css'
import {AppContext} from "./context/AppContext";
import {NavBar} from "./components/NavBar";
import {LoaderScreenCentered} from "./components/LoaderScreenCentered";
import {useState, useEffect} from 'react'
import {io} from "socket.io-client";
import {useMessage} from "./hooks/message.hook";
import { getCookie, eraseCookie, setCookie } from './utils/CookieAssistant';

const socket_p = 'socket';

function App() {
  const [socket, setSocket] = useState(false)
  const [isAuthenticated, setIsAuthenticated] = useState(false)
  const [isReady, setIsReady] = useState(false)

  const message = useMessage()

  const signUp = (email, password) => {
    let data = {email, password}
    socket.emit('sign_up', data)
  }

  const signIn = (email, password) => {
    let data = {email, password}
    socket.emit('sign_in', data)
  }

  const signOut = () => {
    eraseCookie('jwt')
    setIsAuthenticated(false)
  }

  useEffect(() => {
    if (!isReady) {

```

```

const socket = io("")
setSocket(socket)

// restore authorization
let jwt = getCookie('jwt')
if (jwt !== null) {
  let data = {jwt}
  socket.emit('restore_auth', data)
} else {
  setIsReady(true)
}

socket.on('auth_result', (data) => {
  if (JSON.stringify(data)) {
    let error = data.error
    if (error) {
      message(error)
    } else {
      let jwt = data.jwt
      if (jwt) {
        setCookie('jwt', jwt, 1.0/24.0)
        setIsAuthenticated(true)
      } else {
        message('No jwt token received')
      }
    }
  } else {
    message('Invalid auth_result data')
  }
  setIsReady(true)
})
}, [socket_p])

const routes = useRoutes(isAuthenticated)

return (
  <AppContext.Provider value={{
    isAuthenticated,
    signUp,
    signIn,
    signOut,
    socket
  }}>
    <Router>
      <NavBar />
      { isReady
        ?
          <div className="container">
            {routes}
          </div>
        :
          <LoaderScreenCentered />
      }
    </Router>
  </AppContext.Provider>
)
}

export default App

```

routes.js

```

import React from 'react'
import {Switch, Route, Redirect} from 'react-router-dom'
import {VideosPage} from "../pages/VideosPage";
import {VideoUploadPage} from "../pages/VideoUploadPage";
import {VideoWatchPage} from "../pages/VideoWatchPage";
import {AuthPage} from "../pages/AuthPage";
import {ProfilePage} from "../pages/ProfilePage";

```

```

export const useRoutes = isAuthenticated => {
  if (isAuthenticated) {
    return (
      <Switch>
        <Route path="/" exact>
          <VideosPage/>
        </Route>
        <Route path="/profile" exact>
          <ProfilePage/>
        </Route>
        <Route path="/upload" exact>
          <VideoUploadPage/>
        </Route>
        <Route path="/watch" exact>
          <VideoWatchPage/>
        </Route>
        <Redirect to="/" />
      </Switch>
    )
  } else {
    return (
      <Switch>
        <Route path="/auth" exact>
          <AuthPage/>
        </Route>
        <Redirect to="/auth" />
      </Switch>
    )
  }
}

```

CookieAssistant.js

```

export function setCookie(name,value,days) {
  let expires = "";
  if (days) {
    let date = new Date();
    date.setTime(date.getTime() + (days*24*60*60*1000));
    expires = "; expires=" + date.toUTCString();
  }
  document.cookie = name + "=" + (value || "") + expires + "; path=/";
}

export function getCookie(name) {
  let nameEQ = name + "=";
  let ca = document.cookie.split(';');
  for(let i = 0;i < ca.length; i++) {
    let c = ca[i];
    while (c.charAt(0) === ' ') c = c.substring(1, c.length);
    if (c.indexOf(nameEQ) === 0) return c.substring(nameEQ.length, c.length);
  }
  return null;
}

```



```

}

export function eraseCookie(name) {
  document.cookie = name +'='; Path=/; Expires=Thu, 01 Jan 1970 00:00:01 GMT;';
}

```

message.hook.js

```

import {useCallback} from 'react'

export const useMessage = () => {
  return useCallback(
    (text) => {
      if (window.M && text) {
        window.M.toast({html: text})
      }
    },
    []
  )
}

```

AppContext.js

```

import {createContext} from 'react'
import {Socket} from "socket.io-client";

function nothing() {}

export const AppContext = createContext({
  isAuthenticated: false,
  signUp: nothing,
  signIn: nothing,
  signOut: nothing,
  socket: Socket
})

```

Loader.js

```

import React from 'react'

export const Loader = () => {
  return (
    <div className="preloader-wrapper big active">
      <div className="spinner-layer spinner-blue-only">
        <div className="circle-clipper left">
          <div className="circle"/>
        </div>
        <div className="gap-patch">
          <div className="circle"/>
        </div>
        <div className="circle-clipper right">
          <div className="circle"/>
        </div>
      </div>
    </div>
  )
}

```

LoaderScreenCentered.js

```
import React from 'react'
import {Loader} from "../Loader";

export const LoaderScreenCentered = () => {
  return (
    <div className='centered'>
      <Loader />
    </div>
  )
}
```

NavBar.js

```
import React, {useContext} from 'react'
import {NavLink, useLocation} from 'react-router-dom'
import {AppContext} from "../context/AppContext";

export const NavBar = () => {
  const authContext = useContext(AppContext)

  const logoutHandler = async (event) => {
    event.preventDefault()
    authContext.signOut()
  }

  let title = 'Node-yt'
  const location = useLocation().pathname
  return (
    <nav>
      <div className="nav-wrapper blue darken-1" style={{ padding: '0 2rem' }}>
        <span className="brand-logo"><NavLink to="/">{title}</NavLink></span>
        { (location !== '/auth')
          ?
            <ul id="nav-mobile" className="right hide-on-med-and-down">
              <li style={ location === '/profile' ? {background: 'darkcyan'} : {}}><NavLink to="/profile">Pro-
file</NavLink></li>
              <li style={ location === '/upload' ? {background: 'darkcyan'} : {}}><NavLink to="/upload">Up-
load</NavLink></li>
              <li><a href="/" onClick={logoutHandler}>Sign Out</a></li>
            </ul>
          :
            <div />
          }
        </div>
      </nav>
    )
  }
```

AuthPage.js

```
import React, {useState, useContext} from 'react'
import {AppContext} from "../context/AppContext";

export const AuthPage = () => {
  const authContext = useContext(AppContext)
  const [form, setForm] = useState({
    email: "",
    password: ""
  })
}
```

```

}))

const changeHandler = event => {
  setForm({...form, [event.target.name]: event.target.value})
}

const signupHandler = async () => {
  authContext.signUp(form.email, form.password)
}

const signinHandler = async () => {
  authContext.signIn(form.email, form.password)
}

return (
  <div className="row">
    <div className="col s6 offset-s3">
      <h2>Authorization</h2>
      <div className="card blue-grey darken-1">
        <div className="card-content white-text">
          <div style={{ marginTop: 30 }}>
            <div className="input-field">
              <input
                id="email"
                type="email"
                name="email"
                value={ form.email }
                onChange={ changeHandler }
              />
              <label htmlFor="email">Email</label>
            </div>
            <div className="input-field">
              <input
                id="password"
                type="password"
                name="password"
                value={ form.password }
                onChange={ changeHandler }
              />
              <label htmlFor="password">Password</label>
            </div>
          </div>
        </div>
      </div>
      <div className="card-action">
        <button
          className="btn yellow darken-4"
          style={{ marginRight: 10 }}
          onClick={ signinHandler }
          disabled={ authContext.isLoading }
        >
          Sign In
        </button>
        <button
          className="btn gray lighten-1 black-text"
          onClick={ signupHandler }
          disabled={ authContext.isLoading }
        >
          Sign Up
        </button>
      </div>
    </div>
  </div>
)

```

```
)  
}
```

ProfilePage.js

```
import React, {useState, useContext, useEffect} from 'react'  
import {AppContext} from "../context/AppContext";  
import {LoaderScreenCentered} from "../components/LoaderScreenCentered";  
import {getCookie} from "../utils/CookieAssistant";  
import {NavLink} from "react-router-dom";  
import {FcLike} from "react-icons/all";  
import {Loader} from "../components/Loader";  
  
export const ProfilePage = () => {  
  const authContext = useContext(AppContext)  
  
  const [isLoadingUploaded, setIsLoadingUploaded] = useState(true)  
  const [isLoadingLiked, setIsLoadingLiked] = useState(true)  
  
  const [isLoading, setIsLoading] = useState(true)  
  const [user, setUser] = useState(null)  
  const [uploadedVideos, setUploadedVideos] = useState([])  
  const [likedVideos, setLikedVideos] = useState([])  
  
  const getUploadedVideos = () => {  
    setIsLoadingUploaded(true)  
  
    let jwt = getCookie('jwt')  
    authContext.socket.emit('get_uploaded_videos', {jwt})  
  
    authContext.socket.on('get_uploaded_videos_result', (data) => {  
      setUploadedVideos(data.videos)  
      setIsLoadingUploaded(false)  
    })  
  }  
  
  const getLikedVideos = () => {  
    setIsLoadingLiked(true)  
  
    let jwt = getCookie('jwt')  
    authContext.socket.emit('get_liked_videos', {jwt})  
  
    authContext.socket.on('get_liked_videos_result', (data) => {  
      setLikedVideos(data.videos)  
      setIsLoadingLiked(false)  
    })  
  }  
  
  useEffect(() => {  
    async function fetchUser() {  
      let jwt = getCookie('jwt')  
      authContext.socket.emit('get_user', {jwt})  
  
      authContext.socket.on('get_user_result', (data) => {  
        setUser(data.user)  
        setIsLoading(false)  
  
        getUploadedVideos()  
        getLikedVideos()  
      })  
    }  
  })  
}
```

```

    if (isLoading) {
      fetchUser()
    }
  }, [authContext])

  if (isLoading) {
    return (
      <LoaderScreenCentered />
    )
  }

  return (
    <div>
      <h2>{user.email}</h2>
      <h2>My videos</h2>
      {
        isLoadingUploaded
        ? <div style={{display: "flex", flexDirection: "row", justifyContent: "center"}}><Loader/></div>
        : <div className="collection">
          {
            uploadedVideos.map(video => {
              return (
                <NavLink
                  key={video._id}
                  className="collection-item"
                  to={` /watch?id=${video._id}`}
                  onClick={(event) => video.isProcessing ? event.preventDefault() : null}
                >
                  {video.title}
                  {video.isProcessing ? ' (processing...)' : null}
                  <div className="secondary-content row">
                    <div className="col"><FcLike/></div>
                    <div className="col">{video.likes.toString()}</div>
                  </div>
                </NavLink>
              )
            })
          }
          <NavLink
            key='length'
            className="collection-item blue"
            to={` #` }
            onClick={(event) => event.preventDefault()}
          >
            Count: {uploadedVideos.length}
          </NavLink>
        </div>
      }
    <h2>Liked</h2>
    {
      isLoadingLiked
      ? <div style={{display: "flex", flexDirection: "row", justifyContent: "center"}}><Loader/></div>
      : <div className="collection">
        {
          likedVideos.map(video => {
            return (
              <NavLink
                key={video._id}
                className="collection-item"
                to={` /watch?id=${video._id}`}
                onClick={(event) => video.isProcessing ? event.preventDefault() : null}
              >

```

```

    }
  >
    {video.title}
    {video.isProcessing ? ' (processing...)' : null}
    <div className="secondary-content row">
      <div className="col"><FcLike/></div>
      <div className="col">{video.likes.toString()}</div>
    </div>
  </NavLink>
)
})
}
<NavLink
  key='length'
  className="collection-item blue"
  to={`#`}
  onClick={(event) => event.preventDefault()}
>
  Count: {likedVideos.length}
</NavLink>
</div>
}
</div>
)
}

```

VideosPage.js

```

import React, {useState, useContext, useEffect} from 'react'
import {AppContext} from "../context/AppContext";
import {NavLink} from "react-router-dom";
import {LoaderScreenCentered} from "../components/LoaderScreenCentered";
import {getCookie} from "../utils/CookieAssistant";
import {FcDislike, FcLike} from "react-icons/all";

export const VideosPage = () => {
  const authContext = useContext(AppContext)

  const [isLoading, setIsLoading] = useState(true)
  const [videos, setVideos] = useState([])

  useEffect(() => {
    async function fetchVideos() {
      let jwt = getCookie('jwt')
      authContext.socket.emit('get_videos_list', {jwt})

      authContext.socket.on('get_videos_list_result', (data) => {
        setVideos(data.videos)
        setIsLoading(false)
      })
    }

    if (isLoading) {
      fetchVideos()
    }
  }, [isLoading])

  if (isLoading) {
    return (
      <LoaderScreenCentered />
    )
  }
}

```

```

    }

    return (
      <div>
        <h2>Videos</h2>
        <div className="collection">
          {
            videos.map(video => {
              return(
                <NavLink
                  key={video._id}
                  className="collection-item"
                  to={` /watch?id=${video._id}`}
                  onClick={ (event) => video.isProcessing ? event.preventDefault() : null
                }
              >
                {video.title}
                {video.isProcessing ? ' (processing...)' : null}
                <div className="secondary-content row">
                  <div className="col"><FcLike /></div>
                  <div className="col">{video.likes.toString()}</div>
                </div>
              </NavLink>
            )
          }
        </div>
        <NavLink
          key='length'
          className="collection-item blue"
          to={` #` }
          onClick={ (event) => event.preventDefault() }
        >
          Count: {videos.length}
        </NavLink>
      </div>
    </div>
  )
}

```

VideoUploadPage.js

```

import React, {useContext, useState, useEffect} from 'react'
import {AppContext} from "../context/AppContext";
import {Loader} from "../components/Loader";

export const VideoUploadPage = () => {
  const authContext = useContext(AppContext)

  const [loading, setLoading] = useState(false);

  const [fileTitle, setFileTitle] = useState("");
  const [selectedFile, setSelectedFile] = useState(null);
  const [selectedFileName, setSelectedFileName] = useState("");
  const fileInputRef = React.useRef();

  const changeTitleHandler = (event) => {
    setFileTitle(event.target.value);
  }

```

```

};

const changeFileHandler = (event) => {
  setSelectedFile(event.target.files[0]);
  setSelectedFileName(event.target.files[0].name);
};

const changeFileNameHandler = (event) => {
  setSelectedFileName(event.target.value);
};

const handleSubmit = event => {
  event.preventDefault()

  const data = new FormData();
  data.append("title", fileTitle);
  data.append("video", selectedFile);

  setLoading(true)

  fetch('/api/upload', {
    method: 'POST',
    body: data
  }).then(() => {
    setFileTitle("")
    setSelectedFile(null)
    setSelectedFileName("");
    fileInputRef.current.value = ""
    setLoading(false)
  }).catch(() => {
    authContext.signOut()
    setLoading(false)
  })
};

return(
  <div className="row">
    <div className="col s6 offset-s3">
      <h2>Upload video</h2>
      { loading ? <Loader /> : <div/>}
      <div className="card blue-grey darken-1">
        <form action="#" onSubmit={handleSubmit}>
          <div className="card-content white-text">
            <div className="input-field">
              <input
                id="title"
                type="text"
                name="title"
                autoComplete="off"
                value={fileTitle}
                onChange={changeTitleHandler}
                required
              />
              <label htmlFor="title">Title</label>
            </div>
          </div>
          <div className="card-content file-field input-field">
            <div className="btn">
              <span>File</span>
              <input
                type="file"
                accept="video/*"
                ref={fileInputRef}

```



```

        onChange={changeFileHandler}
        required
      />
    </div>
    <div className="file-path-wrapper">
      <input
        className="file-path validate"
        type="text"
        value={selectedFileName}
        onChange={changeFileNameHandler}
        placeholder="Select file"
      />
    </div>
  </div>
  <div className="card-action">
    <button
      className="btn yellow darken-4"
      style={{ marginRight: 10 }}
      type="submit"
    >
      Upload
    </button>
  </div>
</form>
</div>
</div>
</div>
)
}

```

VideoWatchPage.js

```

import React, {useContext, useState, useEffect} from 'react'
import {useHistory, useLocation} from "react-router-dom";
import {AppContext} from "../context/AppContext";
import {LoaderScreenCentered} from "../components/LoaderScreenCentered";
import {getCookie} from "../utils/CookieAssistant";
import {FcLike, RiDeleteBinLine} from "react-icons/all";

```

```

export const VideoWatchPage = () => {

  const history = useHistory();

  const query = new URLSearchParams(useLocation().search)

  const authContext = useContext(AppContext)

  const [likes, setLikes] = useState(-1)

  const [isLoading, setIsLoading] = useState(true)
  const [isLiking, setIsLiking] = useState(false)
  const [isDeleting, setIsDeleting] = useState(false)
  const [video, setVideo] = useState(null)

  const deleteVideo = () => {
    async function deleteAsync() {
      let jwt = getCookie('jwt')
      let id = video._id

      authContext.socket.emit('delete_video', {jwt, id})

      authContext.socket.on('delete_video_result', (data) => {

```

```

        history.push('/')
      })
    }

    if (!isDeleting) {
      setIsDeleting(true);
      deleteAsync()
    }
  }

const likeVideo = () => {
  async function likeAsync() {
    let jwt = getCookie('jwt')
    let id = video._id

    authContext.socket.emit('video_like', {jwt, id})

    authContext.socket.on('video_like_result', (data) => {
      setLikes(data.likes)
      setIsLiking(false)
    })
  }

  if (!isLiking) {
    setIsLiking(true);
    likeAsync()
  }
}

useEffect(() => {
  async function fetchVideo() {
    const videoId = query.get('id')

    let jwt = getCookie('jwt')

    authContext.socket.emit('get_video', {jwt, videoId})

    authContext.socket.on('get_video_result', (data) => {
      if (data.videos) {
        setLikes(data.videos[0].likes)
        setVideo(data.videos[0])
        setIsLoading(false)
      }
    })
  }

  if (isLoading) {
    fetchVideo()
  }
}, [query])

if (isLoading || isDeleting || !video) {
  return (
    <LoaderScreenCentered />
  )
}

return (
  <div>
    <h1>Watch {video.title}</h1>
    <div className="row">
      <div className="col" onClick={e => {
        likeVideo()
      }}>

```

```

    }}<FcLike /></div>
    <div className="col">{likes.toString()}</div>
  </div>
  <div className="row">
    <div className="secondary-content col" onClick={ (e) => {
      deleteVideo()
    }}><RiDeleteBinLine /></div>
  </div>
  <div className='watch-video-div' style={{( { marginBottom: '10rem', marginTop: '2rem' } )}}>
    <video controls={true} autoPlay={false} >
      <source src={'/api/videos/file/' + video._id} type='video/mp4' />
    </video>
  </div>
</div>
)
}

```

Обозначение					Наименование					Дополнительные сведения				
					<u>Текстовые документы</u>									
БГУИР КР 1–40 01 01 613 ПЗ					Пояснительная записка					27 с.				
					<u>Графические документы</u>									
ГУИР 85100076 613 СП					"Видеохостинг", А1, схема программы, чертеж					Формат А1				
										</				