МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки / специальности)

«Бизнес-информатика»
(направленность (профиль) / специализация)

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка децентрализованного чата с функцией видеосвязи»

Обучающийся Л.А. Антошин _____
(Инициалы Фамилия) (личная подпись)

Руководитель канд.пед.наук, доцент, Т.А. Агошкова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант канд.филол.наук, Н.В. Андрюхина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2022

## Аннотация

С. 67, рис. 20, табл. 2, лит. 27 источников

Разработан проект децентрализованного чата с функцией видеосвязи на основе технологии передачи данных WebRTC.

Дано описание концепций развития Всемирной паутины, выявлены концепции развития. Описаны протоколы и методы запросов установки соединения между двумя участниками сети. Разработаны клиентская часть и часть сервера сигнализации. Произведен выбор и обоснование технологий построения видеосвязи, а также проектных решений разработки видеочата. Сформулированы цель и задачи проектирования и требования к децентрализованному чату с функцией видеосвязи.

Построены блок-схемы алгоритмов элементов управления, описано программное, компьютерное, сетевое и технологическое обеспечение децентрализованного чата с функцией видеосвязи.

Представлены диаграмма вариантов использования, диаграмма классов и диаграмма потоков данных.

Оценены преимущества децентрализации при организации связи.

Проведено тестирование и сравнение с приложениями, использующими аналоги способов организации связи.

Работа находится на стадии дополнительной разработки.

## Abstract

The title of the graduation work is «Development of a decentralized chat with a video function». This graduation work is devoted to the development of decentralized chat project with a video communication function based on WebRTC data transfer technology. The graduation work consists of an explanatory note on 67 pages, introduction, including 20 figures, 2 tables, the list of 27 references including 5 foreign sources and 6 appendices, and the graphic part on 12 A4 sheets.

The protocols and methods for requesting a connection between two network participants are described. The client part and the signaling server part have been developed. The choice and justification of technologies for building video communications, as well as design solutions for the development of video chat, were made. The purpose and objectives of the design and requirements for a decentralized chat with a video communication function are formulated. The author made a comparative analysis of JavaScript frameworks suitable for the goals set. The graduation work presents the description of the process of an application development using the React library.

Block diagrams of algorithms for control elements are constructed, software, computer, network and technological support for a decentralized chat with a video communication function are described. A use case diagram, a class diagram, and a data flow diagram are provided. The advantages of decentralization in the organization of communication are estimated. Testing and comparison with applications using analogues of communication methods were carried out.

As a result of the graduated work the advantages of decentralized «client-client» communication were confirmed, which was displayed on the application testing graphs. Comparison of these results with the results of testing of an application using analogues of data transfer confirms that the developed application has an advantage in data transfer speed. Nevertheless, it is necessary to obtain additional experimental data in the further development of the project, and the addition of multi-functionality in particular.

## Оглавление

# Введение

Актуальность темы исследования децентрализации в стремительно развивающихся технологиях обуславливается переходом множества информационных отраслей на новый этап развития: на основе децентрализованных сетей уже второе десятилетие развивается криптоэкономика, а во Всемирной паутине возникает концепция «Web 3.0», появляются сервисы децентрализованного хранения данных. Распределение возможностей и ответственности между участниками одной сети являются перспективной альтернативой централизованным технологиям. Данная работа выполнялась по инициативе автора.

Общение между клиентами без использования серверов уже используется программами многих компаний, из которых самые известные это Google – «Google Meet» и «Zoom».

Проблема быстрой и доступной связи сегодня актуальна, так как сотовая связь еще не может обеспечить покрытие во всех уголках планеты. Для организации как можно лучшего качества необходимо уменьшать количество посредников, на которых задерживается связь.

Целью данной работы является описание и анализ способов разработки децентрализованного чата с функцией видеосвязи.

Объектом изучения является организация связи в одноранговой сети.

Предмет изучения – способы передачи информации в децентрализованой сети, а также разработка для этих целей программного обеспечения.

Выполненная работа решает проблему выбора наиболее подходящих сред и средств разработки приложения, организующего связь.

Путем сравнительного анализа протоколов передачи данных определяется технология разработки проекта.

Теоретическая значимость состоит в систематизации теоретических знаний вопроса исследования и его актуализация.

Практическим значением разработки децентрализованного чата с функцией видеосвязи является подтверждение практичности данного вида связи, рассмотрение разницы преимуществ протоколов и определение сложности реализации проекта.

ВКР состоит из введения, трех глав, заключения, списка литературы и приложений.

В первой главе проведено изучение развития Всемирной паутины, определены понятия децентрализации, рассмотрены этапы развития вэба, изложены протоколы и технологии передачи информации в одноранговых сетях, описаны методы передаваемых запросов.

Во второй главе проведено сравнение популярных библиотек среды разработки Node.js, таких как React, Vue и Angular. Описаны шаги разработки сервера сигнализации и проверки его работы.

Третья глава посвящена разработке клиентской части видеочата, описаны библиотеки и порядок создания компонентов приложения. Проведена проверка работы проекта между двумя удаленными устройствами, проведено тестирование приложения и его сравнение с аналогами, использующими другие технологии.

# Глава 1 Изучение актуальности понятия децентрализации

## 1.1 Исследование предметной области

Целью исследования, в этой работе, является исследование децентрализованных технологий, как нового этапа развития Всемирной паутины и как результат исследования разработка проекта децентрализованного чата с функцией видеосвязи. Для полноты картины необходимо изучить предметную область и изложить историю возникновения Интернета, развитие Всемирной паутины и концепции, по которым происходит дальнейшее развитие и разработка программного обеспечения в этом направлении.

В последнее время с развитием технологий все направления жизни человека стремительно меняются. За последние десятилетия мир сменился до неузнаваемости: после появления интернета общество получило доступ к бесчисленному количеству знаний, упростило множество процессов, занимающих до этого значительное количество времени, люди получили возможность общаться, находясь на любом расстоянии. Интернетом пользуется половина населения планеты (около 4,66 млрд чел, что составляет 59,5% населения) [3], количество пользователей постоянно растет, что свидетельствует о распространении и развитии технологии.

Таким, как мы его знаем, Интернет появился в 1991 году, когда на основе Интернета начала функционировать распределенная система доступа к документам среди связанных между собой сетью компьютеров – Всемирная паутина (также называемая «веб» – англ. «web»), имеющая аббревиатуру WWW [8].

Приблизительно с 2001 года подход к содержанию страниц в вебе изменился. Если до этого содержимое страниц редактировалось только владельцем, то теперь начали появляться блоги (интернет-журналы), что добавило страницам и их содержимому интерактивности. 30 сентября 2005

года в американском издательстве O'Reilly Media выходит статья «Tim O'Reilly — What Is Web 2.0», в которой американский активист за свободное программное обеспечение Тим О'Рейлли объединил появление большого количества сайтов, обладающими общими концепциями с развитием интернет-общества [23]. Условно эту дату считают датой появления понятия «Web 2.0», а интернет до этой даты условно стали считать «Web 1.0». Web 2.0 является комплексным подходом к реализации, поддержке и организации веб ресурсов, а не технологией. В дальнейшем Web 2.0 вытесняет Web 1.0.

Web 2.0 продолжает использовать все предыдущие технологии и является лишь концепцией, имеющей свои отличительные признаки [25].

Самым большим недостатком является сбор личных данных пользователей (личные данные, статистика, данные о личной жизни, карьере и семье). Недостаточная защита данных приводит к распространению их и может быть использована в качестве шантажа или манипулирования пользователями ресурсов. В подавляющем большинстве информация и личные данные хранятся на сервере, то есть являются централизованными.

Руководитель американской IT корпорации Netscape Communications Corporation Джейсон Калаканис в своем личном блоге 10 марта 2007 года опубликовал определение концепции Web 3.0, где было отмечено, что большинство ресурсов, созданных на мощных серверах, теряют свою уникальность и в большинстве своем единообразны [27]. Идеей концепции, по его мнению, является пересмотр принципов Web 2.0 для создания уникального и качественного контента. Основные понятия до конца еще не выработаны, так как развитие еще находится на ранних стадиях. Основной сутью концепции Web 3.0 является децентрализация, способствующая отказу от серверов [7]. Данные распределяются среди пользователей сети, и компьютер пользователя одновременно является и клиентом, и сервером. Вычислительные процессы будут происходить на устройствах пользователей вместо дата-центров и серверов [5].

Таким образом, исследовав этапы развития Интернета, можно наблюдать тенденцию стремления к децентрализации информационного пространства. В похожем направлении двигается мир крипто экономики, который полностью построен на принципе децентрализации.

В вопросе децентрализации огромное значение имеет архитектура децентрализованной (пиринговой) сети. Одноранговая сеть - это сеть, которая представляет собой совокупность узлов, связанных между собой без центрального элемента. Таким образом каждый участник сети представляет собой и клиент, и сервер одновременно. Плюсы такой архитектуры заключаются в сохранении приватности информации, не выходящей за пределы связи двух участников. Также преимуществом такого подхода является увеличение скорости передачи информации, так как она попадает непосредственно к участнику связи. Отрицательным моментом отсутствия сервера является распределение вычислительных мощностей на самих участников сети, но постоянное развитие компьютеров и гаджетов способствуют повышению производительности [13].

## 1.2 Выбор технологии разработки чата с функцией видеосвязи

### 1.2.1 Протоколы и технологии передачи информации

Установкой соединения и передачи информации в сети занимается протокол управления передачей (Transmission Control Protocol – “TCP”), последняя спецификация которого датируется сентябрем 1981 года (RFC 793). TCP обычно реализуется операционной системой пользователя. Пакеты данных, реализуемых в протоколе называют сегментами [20]. Используется другими протоколами прикладного уровня, такими как FTP, Telnet, SMTP и HTTP. Протокол, хоть и требует для реализации достаточно много времени, используется для установки надежной связи. Если необходима быстрая передача данных без обратной отправки информации о ее целостности используется User Datagram Protocol (UDP) [10]. С постепенным уходом

сервиса Flash браузеры начали подключать новые средства для обмена мультимедийными данными (аудио и видеопотоки). Например, для сервисов которым необходима постоянная неразрывная и двухсторонняя передача данных хорошо подходит протокол WebSocket. Этот протокол через постоянное соединение обеспечивает постоянную передачу данных. С помощью специальных заголовков протокол посылает запрос серверу, и, если тот поддерживает WebSocket они начинают работу [13] (Рисунок 1).

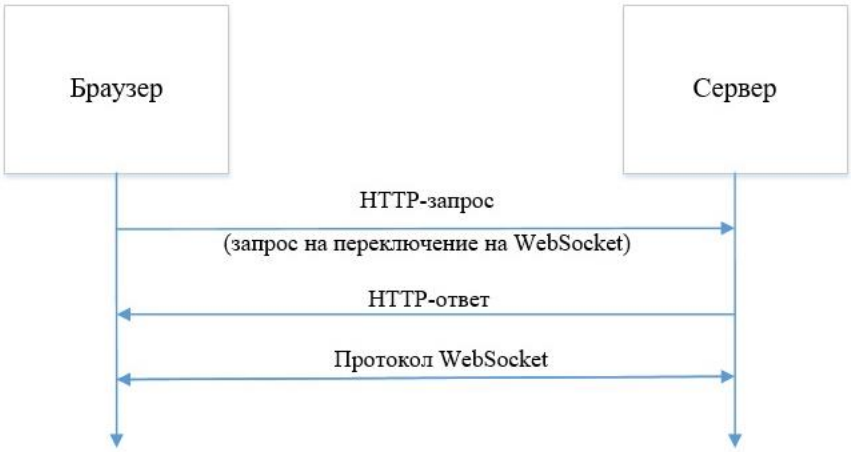


Рисунок 1 – Запрос протокола WebSocket к серверу

На таком протоколе работают такие сервисы связи, как Socket.io и Autobahn. Кроме WebSocket непрерывную связь осуществляют такие протоколы, как LP (Long Polling) и SSE (Server-Sent Events). Ниже приведена разница между ними (Таблица 1) [26].

WebSocket использует HTTP в качестве начального транспортного механизма, но поддерживает TCP-соединение после получения ответа HTTP для последующего использования при отправке сообщений между сервером и клиентом [13].

Таблица 1 - Сравнение протоколов поддержки данных

| | SSE | WebSocket | LP |
|---|---|---|---|
| Протокол работы | http/https | ws или wss (wss - защищенный вид протокола) | http/https |
| Направленность данных | однонаправленная | двунаправленная | двунаправленная |
| Скорость | низкая | высокая | низкая |
| Количество параллельных соединений из браузера | 6 на сервер | 1024 | 6 на сервер |
| Поддержка всеми браузерами | Нет (84% - полностью не поддерживается браузерами IE и Edge) | Да (90%) | Да (100%) |
| Прочее | При разрыве соединения происходит автоматическое переподключение | Возможна передача бинарных данных | |

WebRTC (англ. real-time communications) – стандарт, разработанный компанией Global IP Solution, стандартизируется на протокольном уровне сообществом IETF, а на уровне API консорциумом всемирной паутины (W3C). С 2010 года принадлежит корпорации Google. Эта технология описывает передачу потоковых аудио- видеоданных и файлов между браузерами или любыми поддерживающими его программами без предварительной установки дополнительных программ или плагинов. Для начала общения необходимо просто открыть страницу веб-конференции, таким образом браузер сам становится конечным терминалом конференцсвязи. Преимуществом WebRTC над Flash стала возможность peer-to-peer соединения. Технология WebRTC полностью заменила Flash в HTML5. На данной технологии работают такие сервисы конференцсвязи, как Google Meet, Jitsi Meet, BigBlueButton. В своей работе WebRTC пользуется протоколами WebSocket [4].

### 1.2.2 Принцип работы WebRTC

Связь по WebRTC настраивается с помощью языка JavaScript, в котором реализуются и подготавливаются все необходимые объекты. На

первом браузере происходит вызов метода createOffer(), который является методом WebRTC. Результатом работы метода является SDP пакет, содержащий информацию, о необходимых коммуникациях (аудио, видеоданные, файлы), наличие кодеков и параметров браузера.

Следующим шагом, в зависимости от реализации технологии пакет должен быть передан второму браузеру. Чаще всего для этого используется или сигнальный сервер, или передача проходит по протоколу WebSocket. Получив пакет данных второй браузер через метод WebRTC setRemoteDescription() вызывает метод createAnswer(), который формирует такой же SDP пакет, но с учетом информации, пришедшей от первого браузера. Пакет отправляется первому отправителю. Теперь оба осведомлены о друг друге [22].

Параллельно с обменом информации обоими браузерами происходит исследование сети. На этапе инициации объектов в JavaScript необходимо предусмотреть передачу STUN-сервера, который в ответ на UDP пакет с запросом IP-адреса его предоставляет. STUN-сервер (Session Traversal Utilities for NAT) необходим для получения и сравнения «внутреннего» и «внешнего» IP адресов, таким образом можно узнать используется ли NAT (Network Address Translation – средство предоставления устройству публичного IP адреса) и если используется, то какие обратные порты будут использоваться для передачи UDP пакетов.

В случае использования двойного NAT необходимо использование TURN-сервера (Traversal Using Relays around NAT), которые, по сути, превращают соединение «клиент-клиент» в «клиент-сервер-клиент» (Рисунок 2).

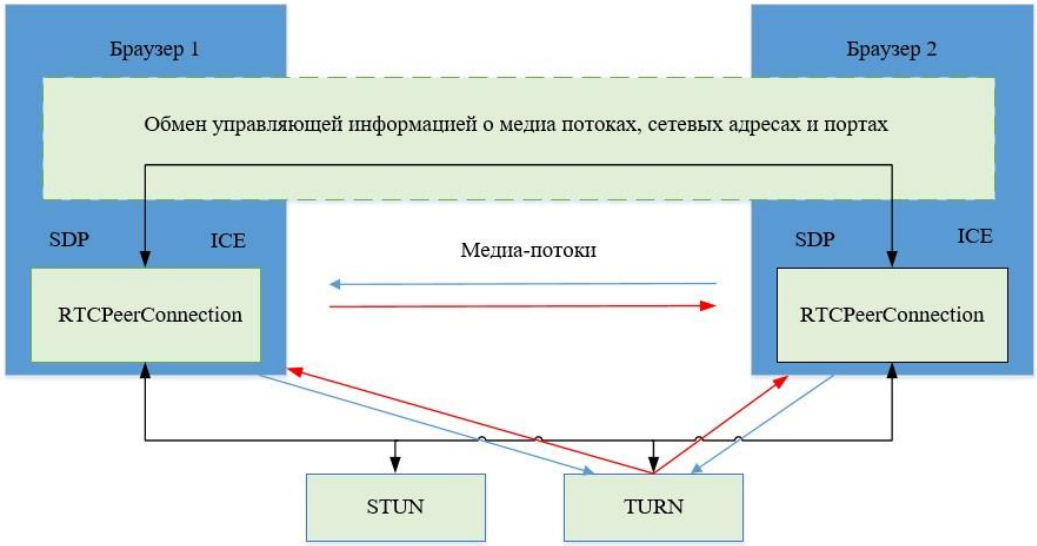


Рисунок 2 - Схематическое изображение работы WebRTC

Отобразим направления потоков данных с помощью DF диаграммы (Рисунок 3).

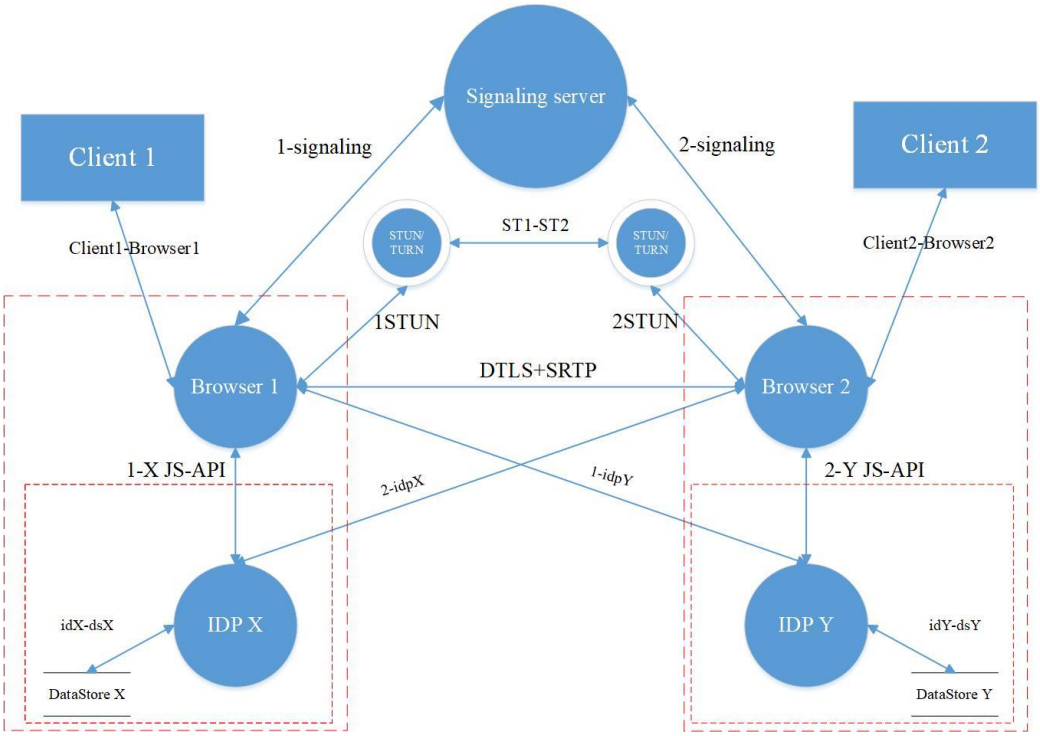


Рисунок 3 – Диаграмма потоков данных, нотация Йордана

В случае успешного прохождения всех этапов устанавливается соединение. Периодически WebRTC на обоих браузерах будет вызывать колбек событие onicecandidate, передавая второму браузеру SIP (протокол установления сеанса) пакет с информацией об IP, NAT, подключением между клиентами. Пакет передается WebRTC с помощью addIceCandidate() (Рисунок 4).

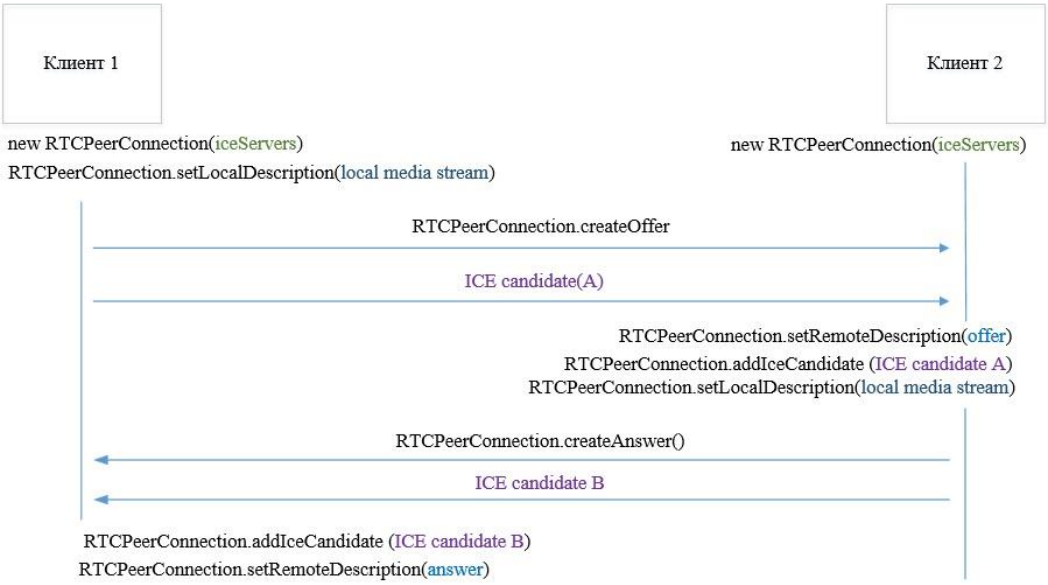


Рисунок 4 - Методы, вызываемые при общении клиентов

WebRTC использует фреймворк ICE (Interactive Connectivity Establishment) для соединения двух клиентов вне зависимости от топологии сети. Даже если оба пира используют транслятор сетевых адресов протокол ICE помогает найти и установить соединение. Для соединения двух одноранговых узлов алгоритм ищет путь с наименьшей задержкой, перебирая варианты в следующем порядке:

- прямое соединение UDP (только в этом случае STUN-сервер используется для поиска сетевого адреса узла);
- прямое соединение TCP (сначала HTTP, потом HTTPS порты);
- косвенное соединение через TURN в случае сбоя прямого.

Представим работу технологии WebRTC с помощью диаграммы классов (Рисунок 5).

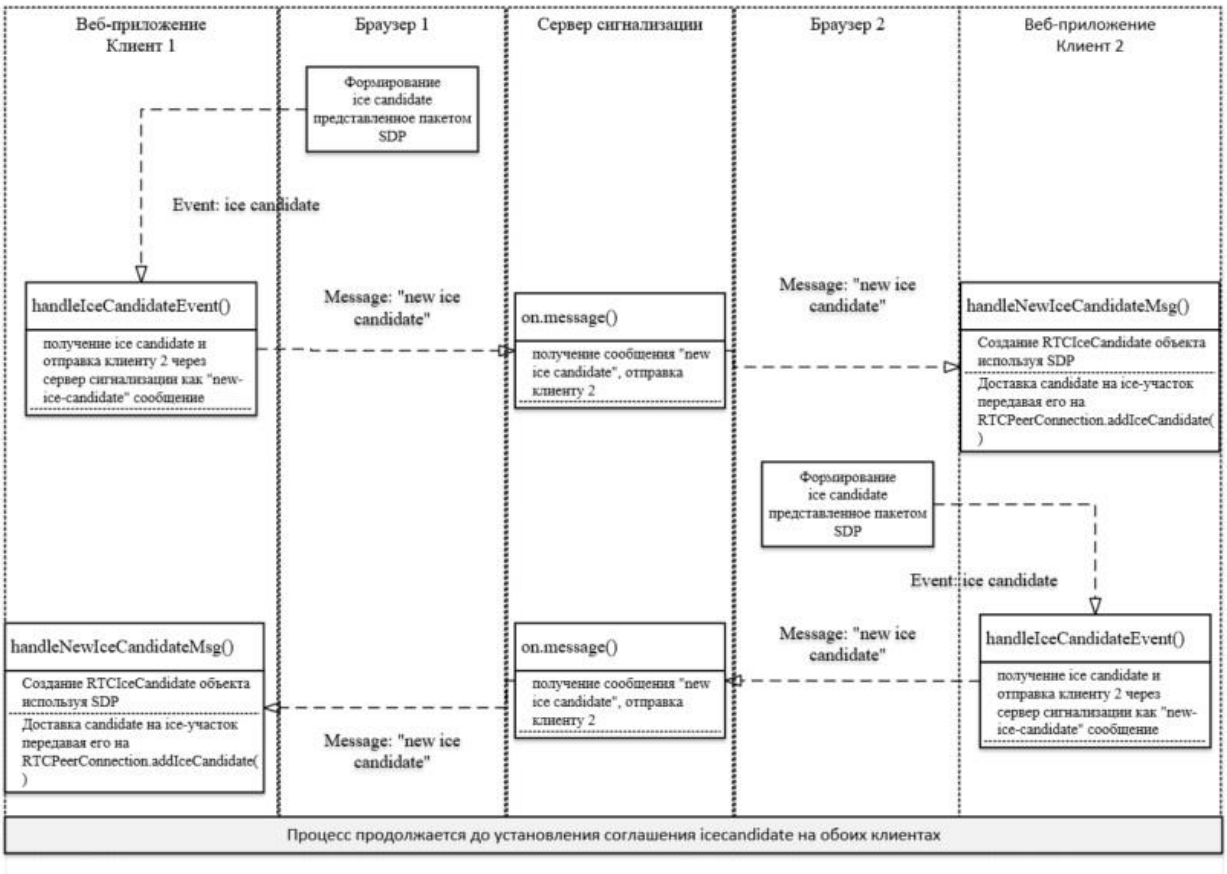


Рисунок 5 – Диаграмма классов WebRTC.

Подытоживая вышесказанное, можно выделить преимущества технологии WebRTC:

- открытый исходный код, предоставляющий удобство в реализации через JavaScript и HTML;

- доступна интеграция с другими сервисами, существует возможность встраивания;

- поддержка современных кодеков (видео – VP8, H.264, аудио - Opus), обеспечивающих высокое качество медиа информации;

- шифрование информации по протоколам TLS и SRTP;

- эхо- и шумоподавление;

- реализация с бэкенд системами через WebSocket;

- не требуется установка дополнительного ПО для работы и реализации;

Из отрицательных характеристик можно выделить отсутствие совместимости приложений разных разработчиков и необходимость сервера видеоконференцсвязи (микширующего видео и звук) для проведения аудио- и видеоконференций в группе, так как браузер не может синхронизировать два и более входящих потока [4].

### 1.2.3 Постановка задачи на разработку проекта создания децентрализованного чата с функцией видеосвязи

Целью проекта является разработка децентрализованного чата с функцией видеосвязи, которое будет представлено в виде веб-приложения. Должны быть полностью и четко определены характер, цели и содержание проекта, а также все конечные продукты проекта с их характеристиками [14]. Разработанное приложение должно отвечать функциям по созданию и поддержке связи, быть простым и удобным в использовании [11]. Видеочат должен позволять двум пользователям при получении как минимум одним пользователем уникального идентификатора (ID) другого пользователя устанавливать соединение «клиент-клиент» и воспроизводить двухстороннюю аудио- видеосвязь.

Для достижения цели требуется рассмотреть такие задачи как:

- рассмотреть способы реализации приложения;

- сравнить работающие с сокетами фреймворки и библиотеки;

- построить диаграмму вариантов использования;

- разработать серверную часть (сервер сигнализации);

- разработать клиентскую часть с помощью выбранного фреймворка;

- протестировать разработанное приложение и представить результаты тестирования;

- описать полученные результаты, сформулировать выводы и заключения;

Таким образом, веб-приложение должно соответствовать понятиям децентрализации, работать без использования медиа- и датасерверов, предоставлять равные права и обязанности пользователям, не хранить данные на сервере.

Этапы разработки должны быть последовательны и фиксироваться после каждого этапа выполнения [9].

В общем случае видеочат должен предоставлять следующие возможности:

- предоставление возможности видеосвязи между клиентами;
- высокая скорость связи благодаря малой загруженности сети;
- завершение видеовызова по инициативе одной из сторон;
- безопасность и конфиденциальность;

Требования к разработке приложения:

- среда разработки Node.js;
- язык программирования JavaScript;
- UI-библиотека React;
- построение зависимостей JSON;
- HTML5, AJAX, WebSocket;
- выбор и расположение площадки развертывания частей приложения («деплой площадки»);
- система контроля версий «GitHub»;

Так же разработанное веб-приложение должно быть протестировано на отсутствие ошибок передачи пакетов данных.

Выводы по главе 1

Изучив предметную область и историю развития Всемирной паутины и интернета, также рассмотрев современные тренды и концепции развития понятия децентрализации удалось прийти к понятию актуальности этого вопроса. Также удалось доказать преимущество принципов децентрализации, изучить технологии и протоколы передачи данных в одноранговых сетях.

Сравнив технологии передачи пакетов данных для разработки децентрализованного чата с функцией видеосвязи был выбран протокол WebSocket, и использующая его технология передачи информации в реальном времени WebRTC.

Графически были представлены модели показывающие принципы работы пакетов передачи данных WebSocket между браузером и сервером, представлены методы передачи запросов WebRTC между клиентами, представлена диаграмма потока данных между двумя клиентами при установлении связи по технологии WebRTC.

Реализация задачи по разработке децентрализованного чата с функцией видеосвязи возможна с помощью языка JavaScript, но с целью улучшения качества разработки и работы самого приложения стоит рассмотреть возможность разработать веб-приложение с помощью фреймворков и библиотек языка JavaScript.

Следующим шагом является рассмотрение технологий реализации передачи пакетов данных, сравнение библиотек, работающих с вебсокетами, и способными отобразить визуальную часть разрабатываемого видеочата в браузере. Следующая глава посвящена разработке сервера сигнализации децентрализованного чата с функцией видеосвязи, проверке его работы на локальном сервере и описанию последовательности предпринимаемых действий.

# Глава 2 Разработка серверной части децентрализованного чата с функцией видеосвязи

## 2.1 Выбор среды разработки и фреймворков (библиотек)

Как уже было сказано ранее, для работы видеосвязи достаточно возможностей, которые предоставляет браузер, который в случае общения через вебсокеты выполняет роль конечного терминала. Приложение будет состоять из двух частей: серверной и клиентской.

Видеочат будет реализован в виде веб-приложения, так как такая форма в отличии от обычной веб-страницы имеет свои преимущества [2].

Например, в случае реализации приложения с помощью фреймворков «React», «Angular», «Vue» языка программирования JavaScript происходит формирование виртуального дерева DOM (Document Object Model) – структурного описания содержимого веб-страниц с помощью объектов. В отличии от обычного DOM могут загружаться только те компоненты, которые или должны показываться, или чье состояние изменилось, что существенно снижает нагрузку на браузер. Еще одним преимуществом веб приложений является меньшее число запросов, так как сервер и клиент находятся на одном устройстве [16], [24].

React — это библиотека пользовательского интерфейса, Angular — это полноценный интерфейсный фреймворк, а Vue.js — прогрессивный фреймворк. Проведем их сравнительный анализ.

React – один из самых популярных JavaScript проектов, который был разработан компанией «Facebook» (на сегодняшний день «Meta»). Библиотека работает на построении наименьших блоков – элементов, которые представляют собой синтаксическую «смесь» HTML верстки и JavaScript. Несколько элементов можно поместить в компонент – многократно используемый блок, который принимает входящие данные и отображают элементы в зависимости от приобретенного состояния [6].

Vue – самый популярный из трех рассматриваемых фреймворков с самым большим количеством пользователей, которым пользуется более одного миллиона сайтов. Vue называется прогрессивным фреймворком, но при использовании сторонних библиотек Vuex и Router выступает как полноценный фреймворк. Для работы использует компоненты, содержащие HTML, JS и CSS в одном файле.

Angular – библиотека разработанная Google, из трех рассматриваемых библиотек самая менее популярная. Проект на этой библиотеке состоит из как минимум одного корневого компонента и корневого модуля. Структурируется по модулям, компонентам и службам. Фреймворк использует TypeScript (сходный с языком JavaScript язык программирования, имеющий более строгую типизацию).

В таблице 2 приведены сравнения этих фреймворков.

Таблица 2 – Сравнение JavaScript фреймворков

| Показатель | Angular | React | Vue |
|---|---|---|---|
| Основное понятие | Высокопроизводительный фреймворк | Библиотека для интерфейсов | Библиотека собирающая положительные качества React и Angular |
| Кому подходит | TypeScript разработчикам | «Все в JavaScript» | Когда используется и HTML и JavaScript |
| Создатели | Google | Facebook | Бывший сотрудник Google |
| Релиз | Сентябрь 2016 | Март 2013 | Февраль 2014 |
| Функциональность | Хорошо подходит для нативных, гибридных и веб-приложений | Отлично подходит для создания одностраничного или мобильного приложения | Разработка улучшенных одностраничных и нативных приложений |
| Отличный для | Крупномасштабных приложений | Современная веб-разработка и веб и ios приложений | Легковесной веб-разработки |
| Кривая обучения | Глубокое обучение | Проще Angular | Неглубокое обучение |
| Базирование | MVC/DOM | Virtual DOM | Virtual DOM |
| Написан на | TypeScript | JavaScript | JavaScript |

Продолжение Таблицы 2

| Показатель | Angular | React | Vue |
|---|---|---|---|
| Поддержка сообщества | Большое сообщество разработчиков и последователей | Поддержка сообщества Facebook | Проект с исходным кодом развивающийся на краудфайдинге |
| Популярность | Широко популярен среди разработчиков | 27000 звезд GitHub в год | 40000 звезд GitHub в год |
| Использующие компании | Google, Forbes, Wix, weater.com | Meta, Uber, Netflix, Twitter, Reddit, Paypal | Alibaba, Baidu, GitLab |

Из трех вышеперечисленных библиотек пакет для работы с сокетами socket.io поддерживает React и Angular, но так как React имеет более отзывчивый для разработки веб-компонентов синтаксис остановим свой выбор на нем [18].

В работе браузера для работы веб-страниц и веб приложений используются технологии, которые отвечают каждый за свою область. HTML (HyperText Markup Language) – язык гипертекстовой разметки используется для структурной разметки узлов и блоков с содержимым. CSS (Cascading Style Sheets) – используют для стилизации и создания визуальных эффектов содержимого. Язык программирования JavaScript и его библиотеки отвечают за действия и события. Также на языке JavaScript с помощью платформы Node.js, как и с помощью языка программирования PHP, возможна разработка серверной части, работа с запросами API, REST и SOAP.

Так как видеочат работает в браузере, использует WebRTC и требует создания вебсокетов разработка будет производиться на языке JavaScript. Также для создания веб-приложения и работы серверной части необходима установка пакета библиотек «npm» (node package manager). Удобством «npm» является наличие огромного количества модулей, устанавливаемых непосредственно в рабочую область и кроссплатформенность технологии. «npm» входит в состав Node.js [21].

Предварительно, для работы серверной части необходима установка среды Node.js, установщик которой находится в свободном доступе в интернете на официальном сайте разработчиков. После установки нам становиться доступна установка пакетов «npm», но требуется убедиться в корректной установке платформы. Для этого необходимо открыть консоль и ввести команду: node –version. В случае успешной установки ответом на запрос в консоли отобразиться установленная версия. Вместе с Node.js автоматически устанавливается и «npm».

Для написания и редакции кода необходимо воспользоваться редактором кода. Самыми удобными и популярными для работы с JavaScript являются редакторы «Visual Studio Code», «Webstorm», «Atom». В них не наблюдается особая критичная разница и разработка видеочата будет проходить в редакторе «Visual Studio Code».

## 2.2 Создание файлов и подключение пакетов

Для создания серверной части необходимо создать и настроить зависимости, установить необходимые модули из «npm» и настроить работу веб-сокетов [1]. Обычно, в веб-приложениях зависимости хранятся в формате JSON, и для создания файла с зависимостями сначала создадим директорию, в которой будут находиться файлы видеочата. Созданную директорию необходимо открыть в редакторе кода, после чего зайти во вкладку терминал и написать команду npm init –y. Таким образом будет создан файл package.json с базовыми зависимостями. Также в этот файл в дальнейшем будут вписаны установленные пакеты и их версии.

Следующим шагом является установка необходимых пакетов, а именно:

– Cors (Cross-Origin Resource Sharing) – технология использующая дополнительные заголовки http запросов, используемые в промежутке между браузерами.

– Express – необходим для запуска локального сервера.

– Nodemon – сервис, позволяющий обновлять запускать сервер автоматически, при сохранении изменений.

– Socket.io – событийно-ориентированная JavaScript библиотека состоящая из серверной и клиентской части. В данном случае является самой важной, так как отвечает за передачу веб сокетов, а равно и данных. Преимуществом данного фреймворка является передача сокетов даже в тех случаях, когда работа WebSocket’ов не доступна (альтернативно использует AJAX Long Polling, AJAX Multipart stream Flash Socket). Библиотека удобна в использовании и обладает функцией автоматического переподключения при непредвиденном разрыве соединения [18].

Для установки перечисленных модулей необходимо во вкладке терминал ввести команду «npm install cors express nodemon socket.io» и дождаться установки файлов. Далее создадим файл, который в нашем проекте будет выступать в роли локального сервера. Для этого в той же директории создадим файл index.js с переменными (Листинг 1).

Листинг 1 – Создание переменных и привязка к ним библиотек

```
const app = require("express")();
const server = require("http").createServer(app);
const cors = require("cors");
```

Переменная app обращается к модулю express и представляет его экземпляр, server работает с http запросами, а cors присоединяет дополнительные запросы. Дальше происходит объявление переменной io, с помощью которой будет проходить инициализация клиента, а также добавка запросов cors на использование любых источников cors-запросов, дополнительно указываем необходимые методы запросов (Листинг 2).

Листинг 2 – Код указания дополнительных запросов

```
const io = require("socket.io")(server, {
   cors: {
      origin: "*",
      methods: ["GET", "POST"]
   }
});
```

Для запуска сервера вызываем его с параметром cors (Листинг 3).

Листинг 3 – Код запуска сервера с параметрами Cors

```
app.use(cors());
app.get("/", (req, res) => {
   res.send('Сервер запущен.');
});
```

Определим, какой порт должен слушать сервер и, если сервер не найдется установим ему значение localhost://5000.

const PORT = process.env.PORT || 5000;

Для проверки правильной работы порта в случае успешного подсоединения выведем результат в консоль.

Код вывода сообщения работы сервера в консоль:

server.listen(PORT, () => console.log(`Server listening on port ${PORT}`));

Теперь, при запуске сервера через терминал с помощью команды nodemon index.js мы можем видеть сообщение о том, что сервер успешно подключен (Рисунок 6) [8].

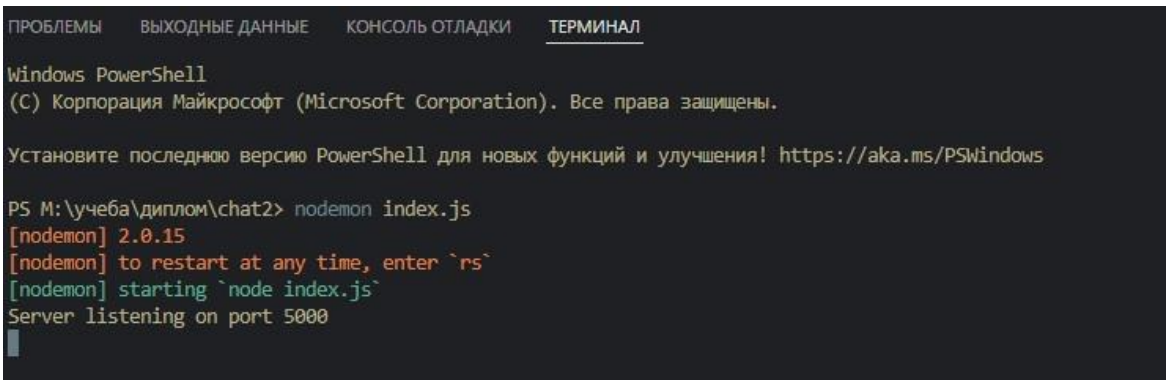


Рисунок 6 – Сообщение консоли в ответ на запуск сервера

Следующим шагом необходимо настроить работу сокетов.

## 2.3 Настройка работы веб-сокетов

Для настройки работы начнем с того, что уже к имеющейся переменной io подключим обработчик событий, который будет срабатывать при подключении к серверу, где в качестве аргумента коллбек функции (функции, вызываемой после совершения какого-либо действия) находится сокет, который является передаваемыми данными, такими как аудио, видео и т.д.: «io.on('connection', (socket) => {});». Внутри колбек функции назначим эмиттер, который в дальнейшем будет использовать обработчик событий, являющийся экземпляром класса EventEmitter, определяющий наш специфический идентификатор: «socket.emit('me', socket.id);». Далее определим обработчик событий, который будет срабатывать при событии разъединения связи (Листинг 4).

Листинг 4 – Код обработчика события разъединения связи

```
socket.on('disconnect', () =>{
      socket.broadcast.emit("callended");
   });
```

Таким образом всем участникам будет транслироваться эмиттер «canllended».

Добавим еще два обработчика событий, один из которых будет принимать данные с клиентской стороны, такие как id звонящего, данные сигнала, откуда поступает вызов и имя. В колбек функции к звонящему будет применен эмиттер, передающий информацию о данных сигнала, откуда поступает сигнал и имя (Листинг 5).

Листинг 5 – Код обработчиков события

```
socket.on("calluser", ({userToCall, signalData, from, name}) => {
    io.to(userToCall).emit("calluser", { signal: signalData, from, name});
});
socket.on("answercall", (data) => {
    io.to(data.to).emit("callaccepted",data.signal);
});
```

Второй обработчик события «answercall» будет принимать с клиентской стороны данные и создавать из них событие «callaccepted», которое мы в дальнейшем будем использовать при разработке клиентской стороны приложения, как и все эмиттеры, объявленные прежде (приложение А). После размещения серверной части на площадке развертывания серверов возможно протестировать работу веб-сокетов через браузер.

Выводы по главе 2

Таким образом для создания децентрализованного чата с функцией видеосвязи были выбраны средства и инструменты разработки, созданы необходимые зависимости, установлены среда разработки Node.js, редактор кода и все необходимые для разработки библиотеки. Был проведен сравнительный анализ популярных фреймворков способных организовать

связь посредством сокетов и организовать медиа обмен данными по технологии WebRTC, в результате чего выбор остановился на интерфейсной библиотеке React, способной реализовать как сервер сигнализации, так и клиентскую часть приложения.

Был создан локальный серверный файл, позволяющий работать с веб-сокетами, который в дальнейшем будет принимать информацию, производить и обрабатывать события с передачей информации на клиентскую сторону, что отображается браузером во вкладке «Сеть/WS» (Рисунок 7) [12].

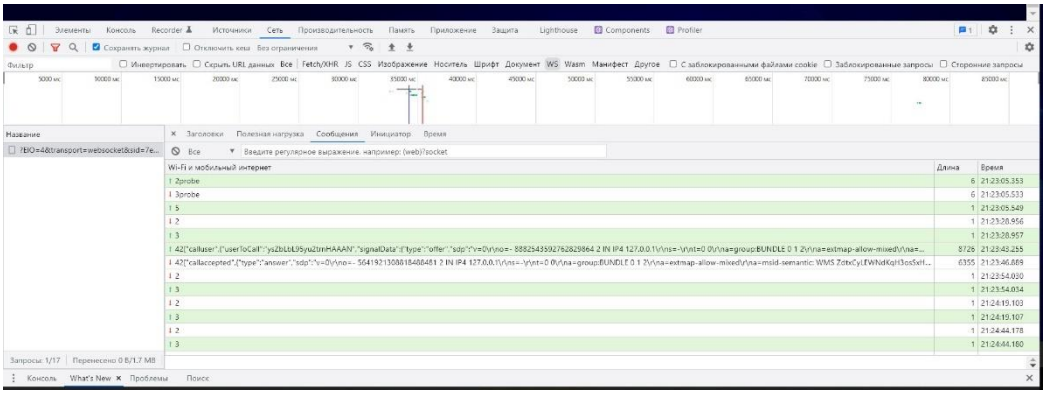

Рисунок 7 – Отображение работы веб-сокета браузером

Описанный метод показывает, как можно без медиасерверов организовать серверную часть работы видеочата, что способствует уменьшению времени отправки и ожидания запросов. Созданная серверная часть позволяет приступить к разработке клиентской части согласно выбранного фреймворка JavaScript, создать веб-приложение, подключить необходимые модули и разработать пользовательский интерфейс децентрализованного чата.

После разработки приложения необходимо провести функциональное тестирование, которое необходимо реализовать согласно вариантов использования, убедиться в организации устойчивой видеосвязи, передачи пакетов данных и отображению соответствующих элементов при получении пакетов с данными приложением.

# Глава 3 Разработка клиентской части

## 3.1 Требования к интерфейсу приложения

Структурируем наш видеочат таким образом, чтоб он состоял из одной страницы, которая визуально будет поделена на три горизонтальные части: сверху будет находиться надпись: «Видеочат», под ней будет находится поле с отображением видеоконтента (при входе на страницу до подключения к собеседнику будет находиться одно окно, в котором пользователь будет видеть изображение, передаваемое собственной камерой, при подключении к собеседнику в том же ряду будет появляться второе окно, отображающее изображение с камерой подключенного собеседника).

Отобразим варианты использования приложения с помощью диаграммы «use case diagram» (Рисунок 8).

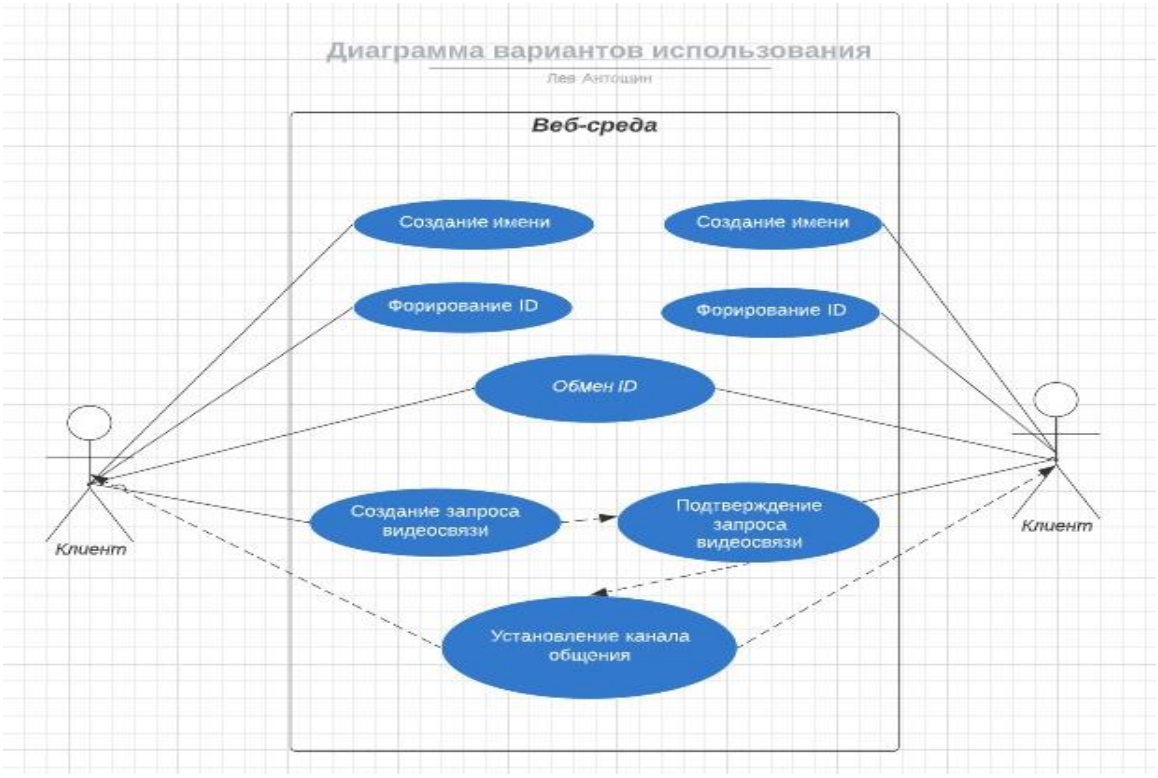


Рисунок 8 – Диаграмма вариантов использования

Под полем с видеоконтентом будет находиться блок навигации, состоящий из двух столбцов. Первый столбец будет называться «Информация», ниже заголовка «Информация» разместим поле для ввода имени, которое будет отображаться в верху собственного окна с видеоизображением.

Так как соединение будет происходить через уникальный идентификатор (id) под полем ввода имени создадим кнопку, копирующую в буфер обмена id. Второй столбец будет иметь заголовок «Позвонить», ниже будет находиться поле, в которое будет необходимо помещать скопированный идентификатор, а под ним поместим кнопку позвонить, при нажатии на которую будет происходить отправка запроса на установление связи.

При получении запроса у второго участника сети под блоком навигации будет появляться поле, сообщающее о том, что ему поступает входящий вызов и кнопкой, позволяющей установить связь. При установке связи кнопка «Вызов» будет меняться на кнопку «Отклонить». При входящем вызове будет отображаться имя, которое ввел вызывающий собеседник и кнопка «Ответить».

## 3.2 Создание веб-приложения с помощью библиотеки «React»

Децентрализованный видеочат будет разрабатываться с помощью фреймворка «React», который является библиотекой языка программирования JavaScript. React использует для рендеринга веб компонентов расширенный языковой формат JavaScript - JSX, который является сочетанием верстки HTML и синтаксиса Javascript, что позволяет структурировать содержимое страниц непосредственно в файлах React. Файлы, содержащие JSX, переводятся в обычный JavaScript формат транскомпилятором Babel, и затем компилируются браузером [15].

Для создания приложения необходимо открыть вторую вкладку терминала, так как на первой вкладке у нас запущен сервер nodemon.

Инициация создания приложения на React происходит путем обращения к «npx» - утилите запуска пакетов «npm». Для этого мы прописываем в консоли находясь в корневой папке проекта: npx create-react-app ./client

Таким образом приложение будет создано в папке «client» куда и загрузит все необходимые модули. Именно в этой директории и будут храниться все необходимые файлы для работы клиентской стороны приложения. После создания приложения React необходимо установить ему необходимые нам для разработки модули:

– socket.io-client – клиентская часть библиотеки socket.io, подробнее о которой мы упоминали в предыдущей главе. Необходима для работы с сокетами.

– simple-peer - библиотека, позволяющая установить p2p соединение по протоколам WebRTC.

– react-copy-to-clickboard – компонент, позволяющий сохранить выбранную информацию в кэш, будет использоваться для более удобного копирования уникального идентификатора.

– @material-ui/core, @material-ui/icons – набор веб компонентов React для простой и быстрой разработки интерфейса компонентов.

Для установки всех перечисленных библиотек и компонентов необходимо ввести в терминале в папке client «npm install socket.io-client simple-peer react-copy-to-clickboard @material-ui/core @material-ui/icons».

После того, как все необходимые библиотеки были установлены можно увидеть, что в папке client появились новые папки с файлами. Изначально новое приложение React создается по шаблону, который затем необходимо редактировать и настраивать под свои нужды разработки [17].

Все необходимые файлы для работы с клиентской частью приложения находятся в директории src, но так как шаблоном мы пользоваться не станем все файлы из этой папки должны быть удалены, и в пустой папке src мы будем создавать собственные файлы.

## 3.3 Создание необходимых файлов и компонентов

### 3.3.1 Создание файлов отображения и подключение стилизации

Сначала необходимо создать файл, в котором React будет собирать все компоненты и передавать их к привязанному элементу с идентификатором блока  на html странице. Создаем файл index.js в папке src. После создания в пустом файле необходимо провести импортирование компонентов из библиотек «react» и «react-dom», импортировать компонент «App», который будет собирательным компонентом других компонентов. Для применения стилей пользовательского интерфейса также создадим и импортируем файл styles.css, который должен располагаться в той же папке, что и index.js (Листинг 6).

Листинг 6 – Код импортирования файлов и объектов

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import './styles.css';
```

Компонент App будет находиться в соответствующем файле App.js в папке с предыдущими импортируемыми файлами. Создав файл для описания стилей страницы, определим в файле styles.css селектору body задний фон на случай, если не будет срабатывать ссылка на изображение заднего плана [15] (Листинг 7).

Листинг 7 – Код установки заднего плана.

```
body {
   background:
```

```
    linear-gradient(
      rgba(0, 0, 0, 0.3),
      rgba(0, 0, 0, 0.3)
    ),
    url(https://unsplash.com/photos/knTKij60p3g/download?ixid=MnwxMjA
3fDB8MXxzZWFyY2h8OHx8YWJzdHJhY3QlMjBiYWNrZ3JvdW5kfGVufDB8
fHx8MTY1MDM2NDU4Nw&force=true&w=1920);
    background-repeat: no-repeat;
    background-attachment: fixed;
    background-size: cover;
}
```

Установим для всей видимой области линейный градиент, а также ссылку для изображения заднего плана, дополнительно определив, что изображение не должно повторяться, должно занимать всю видимую поверхность и оставаться неподвижным.

Для всех остальных компонентов необходимо убрать отступы и сделать выравнивание блоков по границам [19] (Листинг 8).

Листинг 8 – Код файла CSS

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
```

В файле App.js предварительно импортировав компонент React из библиотеки создадим компонент в виде функции стрелки (Листинг 9).

Листинг 9 – Код компонента App

```
const App () => {
  return (
    <div></div>
  );
}
export default App
```

Для стилизации компонентов импортируем @material-ui/core, @material-ui/icons. После импорта их в файл у нас есть набор заранее предопределенных готовых компонентов. Например, компонент <Typography> имеет заранее готовый набор размеров шрифтов, используемых по умолчанию, а компонент <AppBar> является панелью приложения, в которую можно помещать информацию и действия, относящиеся к текущему экрану.

С помощью компонентов структурируем расстановку необходимых нам компонентов, в пустой блок  поместим три необходимые нам секции: заголовок, зону отображения видео и блок с опциями, в который поместим поле с возникающими при вызове уведомлениями (прил. Б) (Листинг 10).

Листинг 10 – Код структуры компонентов

```
return (
    <div>
        <AppBar>
            <Typography>Видеочат</Typography>
        </AppBar>
        <VideoPlayer/>
```

```
        <Options>
          <Notifications/>
        </Options>
      </div>
    )
```

Компоненты <Videoplayer>, <Options> и <Notifications> не предусмотрены установленными библиотеками и будут создаваться нами в соответствующих файлах.

Для стилизации верхнего заголовка импортируем стили компонентов из '@material-ui/core/styles'.

```
import { makeStyles } from '@material-ui/core/styles';
```

Для применения стилей создадим функцию useStyles и поместим в нее компонент makeStyles с описанием стилей в формате объектов (appBar, image, wrapper). Чтобы применить стили к заголовку поместим функцию useStyles() в переменную classes, и назначим классы соответствующим компонентам. Компонентам <AppBar> и <Typography> добавим линейных стилей.

```
<AppBar className={classes.appBar} position="static" color="inherit">
<Typography variant="h2" align="center">Видеочат</Typography>
```

### 3.3.2 Создание компонентов

Для отображения и работы блока с отображением видео, блока с опциями и поля с уведомлениями создадим соответствующие компоненты в соответствующих файлах. Создадим папку components внутри которой поместим три файла: VideoPlayer.jsx, Options.jsx, Notifications.jsx. В каждый из них импортируем необходимые библиотеки и в каждой создадим стрелочную функцию. В аргументы функции Options необходимо поместить свойство {children} После импорта этих трех компонентов в файл App.js начнется компиляция страницы с отображением заголовка (Рисунок 9).

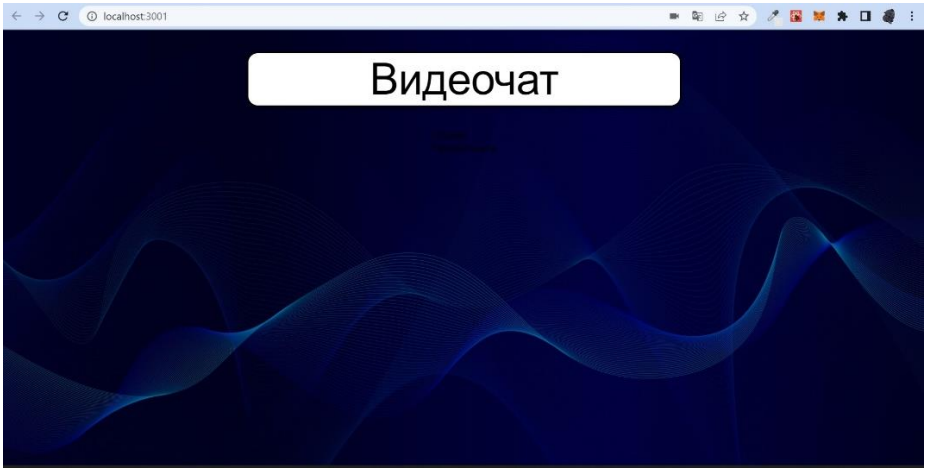


Рисунок 9 - Отображение заголовка видеочата

Разработка контекста сокетов.

Для настройки работы сокетов создадим в папке src файл SocketContext.js. Импортируем ему необходимые библиотеки (Листинг 11).

Листинг 11 – Код импортирования файлов и объектов

```
import React, {createContext, useState, useRef, useEffect} from "react";
import { io } from "socket.io-client";
import Peer from 'simple-peer';
```

В переменную SocketContext передадим вызов функции createContext из библиотеки react. Контекст необходим для передачи свойств по DOM дереву без прохождения через промежуточные компоненты. Для клиентской части библиотеки socket.io создадим переменную socket и поместим в нее компонент io с адресом сервера (в нашем случае сервер настроен на localhost:5000).

Создадим функцию ContextProvider, принимающую аргумент ({children}), который будет содержать в себе функцию ответа на поступающий вызов, функцию создания и завершения вызова. Сначала создадим хук UseEffect, который при загрузке страницы будет получать видео и аудио. Хук

(англ. hook) в React – это функция, позволяющая «подцепиться» к состоянию жизненного цикла без подключения классов (Листинг 12).

Листинг 12 – Код получения собственного медиаконтента

```
useEffect(()=>{
    navigator.mediaDevices.getUserMedia({video:true, audio: true})
      .then((currentStream) => {
        setStream(currentStream);
        myVideo.current.srcObject = currentStream;
    });
    socket.on('me', (id) => setMe(id));
    socket.on('calluser', ({from, name: callerName, signal}) => {
      console.log(from, callerName, signal);
      setCall({isReceivingCall: true, from, name: callerName, signal})
    });
  },[]);
```

Функция ContextProvider будет возвращать компонент <SocketContext.Provider> со значениями вызова, приема вызова, собственным видео, видео собеседника, потоком, именем, установкой имени, завершенном вызове, собственным идентификатором, вызовом собеседника, завершении вызова и ответом на вызов (прил. В) (Листинг 13).

Листинг 13 – Код функции ContextProvider

```
  useEffect(()=>{
    navigator.mediaDevices.getUserMedia({video:true, audio: true})
      .then((currentStream) => {
        setStream(currentStream);
```

```
                myVideo.current.srcObject = currentStream;
            });
            socket.on('me', (id) => setMe(id));
            socket.on('calluser', ({from, name: callerName, signal}) => {
                console.log(from, callerName, signal);
                setCall({isReceivingCall: true, from, name: callerName, signal})
            });
        },[]);
        const answerCall = () => {
            setCallAccepted(true)
            const peer = new Peer({initiator: false, trickle: false, stream});
            peer.on('signal',(data) =>{
                socket.emit('answercall', {signal: data, to: call.from})
            });
            peer.on('stream',(currentStream)=>{
                userVideo.current.srcObject = currentStream;
            });
            peer.signal(call.signal);
            connectionRef.current = peer;
        }
        const callUser = (id) => {
            console.log(me,socket);
            try{
                const peer = new Peer({initiator: true, trickle: false, stream});
            peer.on('signal',(data) =>{
                socket.emit('calluser', {userToCall:id, signalData:data, from: me,
name});
            });
            peer.on('stream',(currentStream)=>{
                userVideo.current.srcObject = currentStream;
```

```
        });
        socket.on('callaccepted',(signal) => {
            setCallAccepted(true);
            peer.signal(signal);
        });
        connectionRef.current = peer;
        }catch(e){
            console.log(e)
        }
    }
    const leaveCall = () => {
        setCallEnded(true);
        connectionRef.current.destroy();
        window.location.reload();
    }
    return (
        <SocketContext.Provider value={{ call, callAccepted, myVideo,
userVideo, stream, name, setName, callEnded, me, callUser, leaveCall,
answerCall}}>
            {children}
        </SocketContext.Provider>)}
```

Создание компонента отображения видеоконтента.

В файле VideoPlayer.jsx импортируем все необходимые библиотеки и компоненты (Листинг 14).

Листинг 14 – Код импортирования объектов и компонентов

```
import React, { useContext } from 'react';
import { Grid, Typography, Paper} from '@material-ui/core';
```

```
import { makeStyles } from '@material-ui/core/styles';
import {SocketContext} from '../SocketContext';
```

Для применения стилей к блоку отображения видео создадим функцию useStyles, пропишем в ней объекты, которые будут применяться у нас в качестве классов и поместим эту функцию в переменную classes (Листинг 15).

Листинг 15 – Код применения стилей.

```
const useStyles = makeStyles((theme) => ({
   video: {
     width: '550px',
     [theme.breakpoints.down('xs')]: {
       width: '300px',
     },
   },
   gridContainer: {
     justifyContent: 'center',
     [theme.breakpoints.down('xs')]: {
       flexDirection: 'column',
     },
   },
   paper: {
     padding: '10px',
     border: '2px solid black',
     margin: '10px',
   },
 }));
```

В самом компоненте VideoPlayer с помощью деструктуризации объекта выделим из компонента SocketContext необходимые переменные:

const {name, callAccepted, myVideo, userVideo, callEnded, stream, call} = useContext(SocketContext);

Компонент VideoPlayer будет возвращать при наличии потока данных окно с медиа данными от пользователя. Чтобы пользователь не слышал собственный голос в тэге <video> необходимо указать значение «muted». Если состояния «callAccepted» && !«callEnded» будут находиться в булиновом значении «true», то на странице будет отображаться второе окно, отображающее медиаданные от собеседника (Рисунок 10) (Листинг 16).

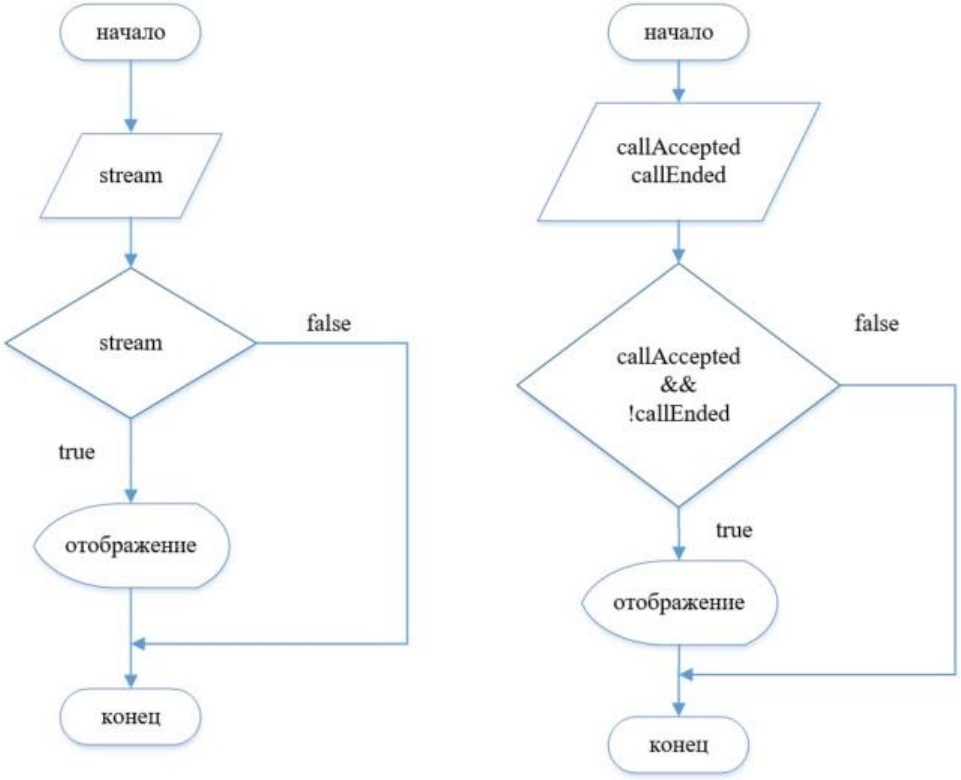


Рисунок 10 - Блок схема алгоритма отображения видеоблоков

Листинг 16 – Код отображения компонентов

```
return (
   <Grid container className={classes.gridContainer}>
```

```
            { stream && (
                <Paper className={classes.paper}>
                <Grid item xs={12} md={6}>
                  <Typography variant="h5" gutterBottom>{name || 'Имя
пользователя'}</Typography>
                  <video playsInline muted ref={myVideo} autoPlay
className={classes.video}/>
                </Grid>
              </Paper>
            )}
            { callAccepted && !callEnded && (
              <Paper className={classes.paper}>
              <Grid item xs={12} md={6}>
                <Typography variant="h5" gutterBottom>{call.name || 'Имя
пользователя'}</Typography>
                <video playsInline ref={userVideo} autoPlay
className={classes.video}/>
              </Grid>
            </Paper>
            )}
          </Grid>);
```

Таким образом, так как мы пока ни к кому не подключены, на странице будет отображаться окно с видеоизображением с камеры пользователя (Рисунок 11).

Полный код компонента находится в приложении Г.

Создание компонента опций и уведомления.

Оставшиеся компоненты опций и уведомлений находятся у нас в соответствующих файлах, в которые необходимо импортировать

необходимые компоненты, темы стилей, изображения иконок и библиотеку копирования текста в буфер.


Рисунок 11 - Отображение заголовка и видеоблока

Для стилизации компонента Options, как и в предыдущих случаях необходимо прописать классы с описанием настроек стилей в виде объектов, которые необходимо поместить в функцию useStyles, а затем поместить функцию в переменную classes в самом компоненте. Также, как и в предыдущем компоненте, с помощью деструктуризации получим все необходимые переменные из контекста.

const {me, callAccepted, name, setName, callEnded, leaveCall, callUser} = useContext(SocketContext);

Структурируем необходимые компоненты и добавим к ним классы стилей (Листинг 17).

Листинг 17 – Код компонентов опций и уведомления

```
<Container className={classes.container}>
    <Paper elevation={10} className={classes.paper}>
      <form className={classes.root} noValidate autoComplete="off">
```

```
<Grid container className={classes.gridContainer}>
<Grid item xs={12} md={6} className={classes.padding}>
<Typography gutterBottom
variant="h6">Информация</Typography>
<TextField label="Введите ваше имя" value={name}
onChange={(e) => setName(e.target.value)} fullWidth/>
{console.log(me)}
<CopyToClipboard text={me}
className={classes.margin}>
<Button variant="contained" color="primary" fullWidth
startIcon={<Assignment fontSize="large"/>}>
Копировать ваш ID
</Button>
</CopyToClipboard>
</Grid>
<Grid item xs={12} md={6} className={classes.padding}>
<Typography gutterBottom
variant="h6">Позвонить</Typography>
<TextField label="ID кому звонить" value={idToCall}
onChange={(e) => setIdToCall(e.target.value)} fullWidth />
```

С помощью тернарного оператора определим, какая кнопка будет отображаться. Если состояние «callAccepted» && !«callEnded» будет соответствовать «true», то будет отображаться кнопка «Отклонить», при нажатии на которую будет срабатывать событие «leaveCall». Если состояние будет соответствовать «false», будет отображаться кнопка «Вызов» при нажатии на которую при наличии в поле «ID кому позвонить» идентификатора будет выполняться стрелочная функция () => callUser(idToCall) (Рисунок 12) (приложение Д).

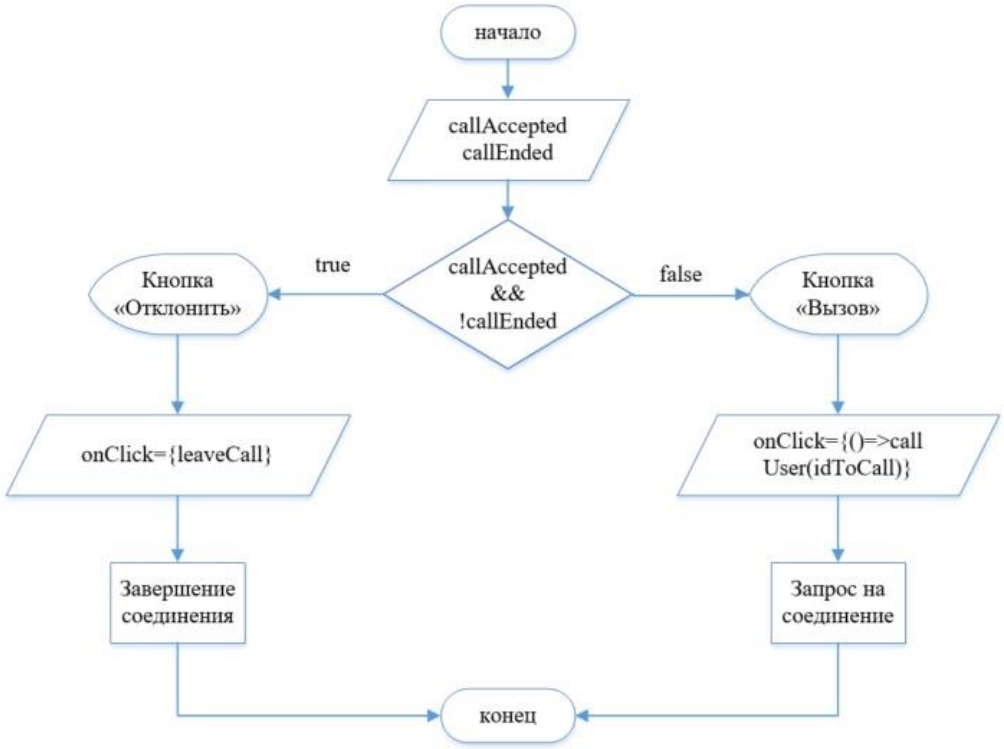


Рисунок 12 - Блок-схема алгоритма отображения кнопки

Теперь под блоком видеоизображения будет отображаться панель с опциями, реагирующая на состояние компонентов (Рисунок 13).


Рисунок 13 - Отображение панели опций на странице

Компонент уведомлений требуется перенести в компонент опций, так как отображаться он будет только при определенных условиях. Импортируем

библиотеку React, компоненты графического интерфейса и контекст сокета. В самом компоненте Notifications деструктурируем «answerCall, call, callAccepted». Компонент Notifications будет возвращать нам поле, на котором будет видно имя вызывающего собеседника и будет появляться кнопка «Ответить». Для появления этого поля необходимо, чтоб состояния «call.isReceiving» && «!callAccepted» соответствовали «true» (Рисунок 14).

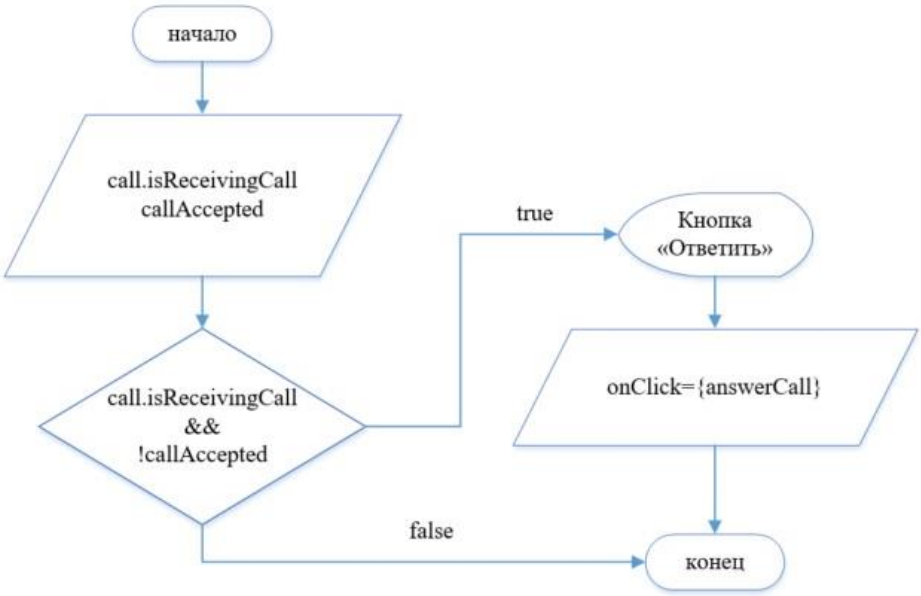


Рисунок 14 - Блок-схема алгоритма вывода кнопки ответа.

При нажатии будет срабатывать функция answerCall(), которую мы определили в контексте сокета (Приложение Е).

### 3.4 Функциональное тестирование чата

После компиляции всех компонентов необходимо протестировать работу децентрализованного чата. В поле для ввода собственного имени введем «Лев», скопируем свой ID нажав на кнопку и вставим его в поле идентификатора собеседника, которому будет отправлен вызов. При нажатии на кнопку позвонить срабатывает передача SDP-пакета через сервер (Рисунок

15), и мы получаем запрос на ответ на собственный вызов, при котором появляется уведомление (Рисунок 16) [12], [8].

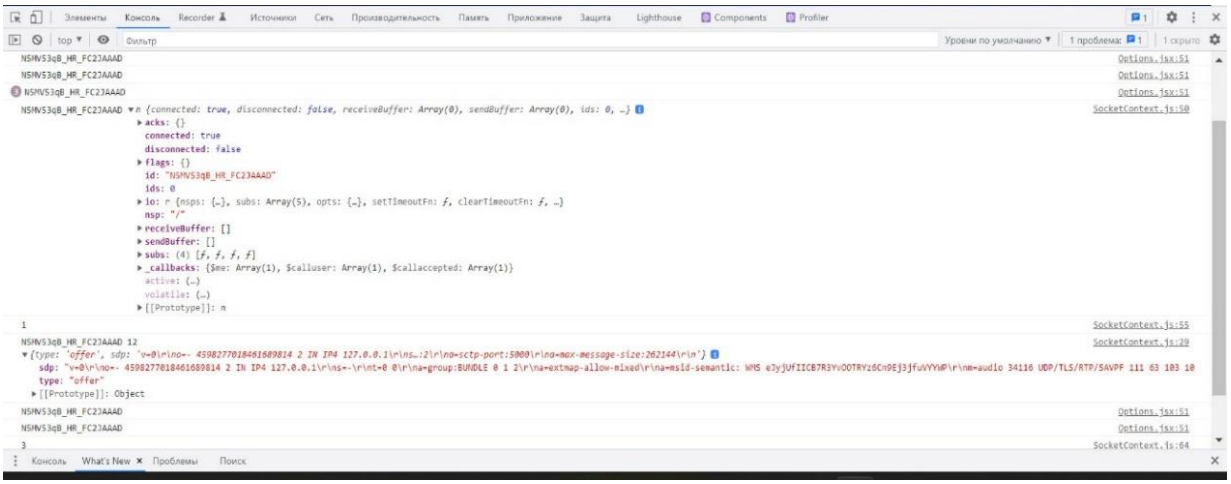

Рисунок 15 – Отображение SDP пакета в браузере

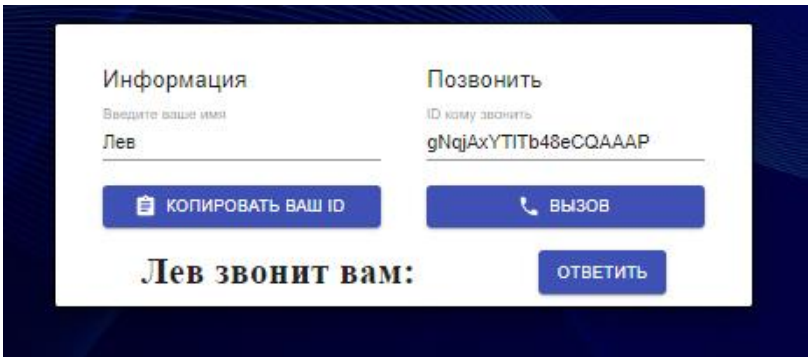


Рисунок 16 - Отображение уведомления о звонке

При нажатии на кнопку ответить устанавливается соединение и появляется второе окно, отображающее, в данном случае, изображение с одной камеры и микрофона (Рисунок 17).

Для работы необходимо разместить сервер и развернуть клиентскую часть на специальных площадках. Для примера мы поместим серверную часть на деплой-площадку «Heraku», а клиентскую на «Netifly» воспользовавшись соответствующими инструкциями, находящимся на страницах площадок.

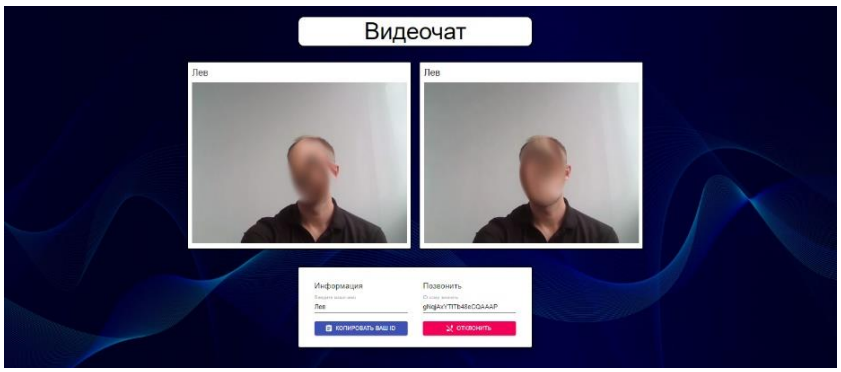


Рисунок 17 - Работа видеочата на локальном сервере

Таким образом при пройдя по ссылке https://p2p-webrtc-chat.netlify.app/ мы можем наблюдать результаты разработки проекта (Рисунок 18).

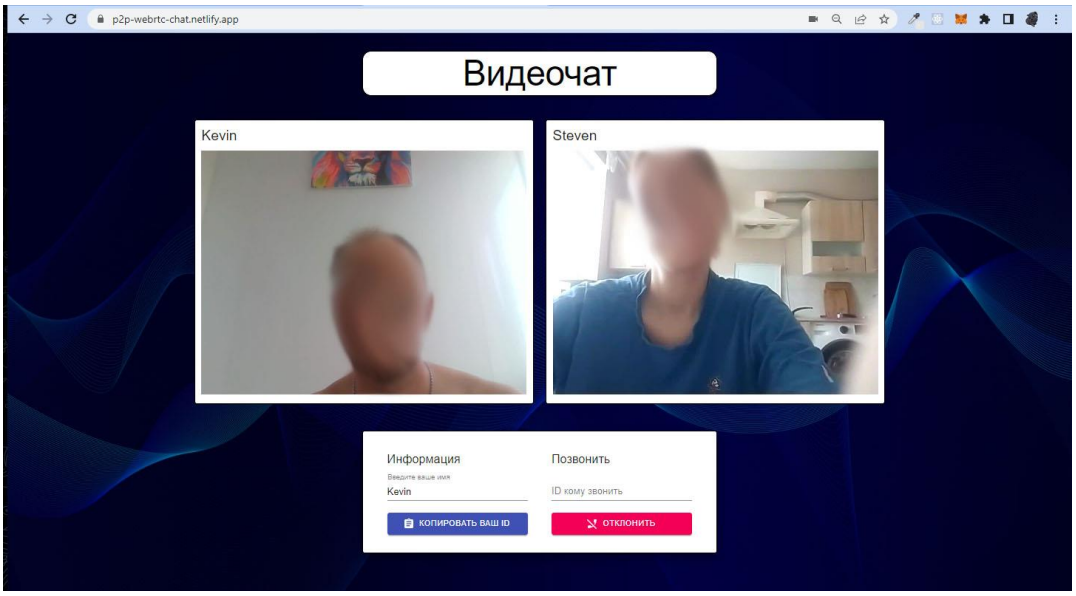


Рисунок 18 – Работа видеочата на удаленных устройствах

С помощью тестовой площадки «testRTC» удалось получить пропускную способность и узнать качество связи в разработанном приложении (Рисунок 19).

Таким образом можно увидеть, что средняя пропускная способность по видеоданным составляет 1650 кбит/с и 22 кбит/с по аудиоданным.


Рисунок 19 – Результаты тестирования видеочата

Для сравнения представим результат тестирования сервиса видеосвязи «antmedia.io», использующий «HTTP Live Streaming» вместо WebRTC (Рисунок 20).


Рисунок 20 – Результат тестирования видеосвязи по «HLS»

Результат тестирования показывает, что средняя пропускная способность видеоданных 302 кбит/с и 46 кбит/с по аудиоданным.

Таким образом доказано преимущество технологии WebRTC над аналогами.

Выводы по главе 3

Были определены требования к структуре и внешнему виду проекта.

С помощью библиотеки React удалось разработать клиентскую часть проекта децентрализованного видеочата, настроить работу сокетов и с помощью алгоритмов определить какое содержимое будет находиться на странице в зависимости от состояния компонентов и входящих запросов. Также использовав для разработки библиотеку React удалось изучить набор пакетов и компонентов, наглядно показать преимущества этого фреймворка и убедиться в преимуществах современных веб-приложений.

Разработанный проект был протестирован и показал успешную работу как компонентов, так и установки, и поддержки соединения.

Результатом разработки проекта стало веб-приложение, для использования которого необходимо воспользоваться ссылкой на площадку, где данное приложение находится. Связь между двумя клиентами осуществляется по предварительной договоренности, если хотя бы один из клиентов знает идентификатор другого.

Для организации связи обоим участникам видеообщения необходимо наличие сети интернет, воспользоваться площадкой размещения приложения одновременно и одному из клиентов воспользовавшись полученным от второго клиента уникального идентификатором осуществить запрос вызова. При подтверждении вызова с принимающей стороны осуществляется двусторонняя видео- и аудиосвязь.

Приложение было протестировано на пропускную способность и показало преимущество по сравнению с аналогами, использующими другие технологии связи.

# Заключение

Изучив концепции развития, тренды и актуальность изучаемого вопроса была доказана практичность децентрализованного обмена данными, как возможность без применения лишних промежуточных узлов взаимодействия пиров. Взаимодействия такого плана позволяют избавиться от администраторов и серверов, предоставляя больше возможностей участникам сети, распределяя вычислительную нагрузку и роли равномерно между включенными в сеть участниками. Исследовав текущее состояние концепций разработки ПО для Всемирной паутины можно сказать, что в перспективе дальнейшее стремление к децентрализации может иметь одну из главных ролей.

Ознакомившись с историей развития интернета и Всемирной паутины удалось изучить и изложить их концепции и характеристики, направления развития и отношение к вопросам децентрализации.

В результате проведенного исследования удалось ознакомиться с разными аспектами и возможностями реализации принципов децентрализации в процессе разработки видеочата. Как итог – веб-приложение обладает необходимыми возможностями для создания соединения «клиент - клиент».

Анализ способов передачи данных позволил выбрать оптимальные решения в вопросе выбора протоколов передачи данных и подбора подходящих средств разработки.

Были представлены диаграмма классов технологии WebRTC и ее диаграмма потока данных («DF»). Для самого приложения создана диаграмма вариантов использования («use case diagram»).

Проведено функциональное тестирование приложения, и результаты отображены передачей пакетов данных в консоли браузера.

При разработке проекта мы ознакомились с синтаксисом написания компонентов, использованием их состояния и последующего отображения на

странице. Также была изучена и отображена работа «хуков» библиотеки «React».

В процессе исследования библиотек и пакетов удалось подтвердить удобство пользованием пакетами управления веб-сокетами, которые одновременно реализуют и функционал сервера сигнализации, и функционал клиентской части приложения. Библиотека работы с пользовательскими интерфейсами «React» соответствует требованиям актуальности для разработки подобных приложений.

Так как приложение используется небольшим количеством участников оно приобретает еще некоторые преимущества: малая загруженность (благодаря чему скорость связи не будет снижаться), конфиденциальное средство общения (связь организовывается между двумя участниками по индивидуальному идентификатору, соединение защищено протоколом шифрования данных HTTPS), безопасность личных данных (данные не хранятся на сервере и удаляются сразу при разрыве соединения). Также к преимуществам разработанного веб-приложения можно отнести гибкость планировки. При необходимости его можно снабдить необходимыми компонентами или поменять внешний вид, что в последующем отобразиться у всех пользователей без необходимости переустановки программного обеспечения.

В перспективе дальнейшей разработки полученного приложения, добавления ему дополнительного функционала возможно получить удобное и многопользовательское приложение, которое не занимает памяти на устройствах пользователя. Также в дальнейшем развитии рационально добавить возможность групповых чатов с возможностью подключения множества пользователей.

## Список используемой литературы


1. Абстракция — ключ к простому коду. [Электронный ресурс]. URL: https://habr.com/ru/company/skillfactory/blog/508716/
2. Актуальность применения АИС в работе интернет провайдера // Гуманитарные научные исследования. 2015. № 12 [Электронный ресурс]. URL: https://human.snauka.ru/2015/12/13222
3. Актуальная статистика и аудитория социальных сетей в мире и Беларуси. [Электронный ресурс]. - URL: https://belretail.by/article/digital-aktualnaya-statistika-i-auditoriya-sotsialnyih-setey-v-mire-i-belarusi
4. Александр Златков. Как работает JS: WebRTC и механизмы P2P-коммуникаций. [Электронный ресурс]. - URL: https://habr.com/ru/company/ruvds/blog/416821/
5. Алексей Погорелый «Web 3.0 — все о новом направлении с применением блокчейн стека» [Электронный ресурс]. - URL: https://inlnk.ru/DBkxYk
6. Бэнкс Алекс, Порселло Ева. React и Redux: функциональная веб-разработка. — СПб.: «Питер», 2018. — С. 336. — ISBN 978-5-4461-0668-4.
7. Даниил Шатухин Что такое Web 3.0, и почему он всем стал нужен [Электронный ресурс]. - URL: https://habr.com/ru/post/653533/
8. Джеймс Уиттакер Как тестируют в Google [Электронный ресурс]:[учебн.курс]/ Джеймс Уиттакер, Джейсон Арбон, Джефф Каролло. — СПб.: «Питер», 2014. — С. 320. — ISBN 978-5-496-00893-8.
9. Долженко А. И. Управление информационными системами [Электронный ресурс] : [учеб. курс] / А. И. Долженко. - 2-е изд., испр. - Москва : ИНТУИТ, 2016. - 181 с. : ил.
10. Дуглас Камер. Сети TCP/IP, том 1. Принципы, протоколы и структура = Internetworking with TCP/IP, Vol. 1: Principles, Protocols and Architecture. — М.: «Вильямс», 2003. — С. 880. — ISBN 0-13-018380-6.

11. Золотов С. Ю. Проектирование информационных систем [Электронный ресурс]: учеб. пособие / С. Ю. Золотов; Томский гос. ун-т систем управления и радиоэлектроники. - Томск: Эль Контент, 2013. – 86 с.

12. Котляров В. П. Основы тестирования программного обеспечения [Электронный ресурс] : [учеб. пособие] / В. П. Котляров. - 2-е изд., испр. - Москва : ИНТУИТ, 2016. - 335 с. : ил.

13. Коржов Валерий. Многоуровневые системы клиент-сервер. Издательство «Открытые системы», 1997;

14. Лукманова И. Г. Управление проектами [Электронный ресурс]: учеб. пособие / И. Г. Лукманова, А. Г. Королев, Е. В. Нежникова. – М.: МГСУ, 2013. – 171 с.

15. Мардан Азат. React быстро. Веб-приложения на React, JSX, Redux и GraphQL. — СПб.: «Питер», 2019. — С. 560. — ISBN 978-5-4461-0952-4.

16. Мержевич Влад. Вёрстка веб-страниц. [Электронный ресурс]. - URL: http://htmlbook.ru/blog/verstka-veb-stranits

17. Обзор библиотеки React. [Электронный ресурс]. - URL: https://ru.reactjs.org/

18. Обзор технологии socket.io [Электронный ресурс]. - URL: https://www.npmjs.com/package/socket.io7. Andrew Lombardi. WebSocket: Lightweight Client-Server Communications/ Andrew Lombardi; SHROFF 2015г;

19. Стоян Стефанов. React. Быстрый старт/ Стоян Стефанов; - - Санкт-Петербург: «Питер», 2016

20. Ю. Н. Гуркин, Ю. А. Семенов Файлообменные сети P2P: основные принципы, протоколы, безопасность [Электронный ресурс]. - URL: http://ccc.ru/magazine/depot/06_11/read.html?0302.htm

21. Янг А., Мек Б., Кантелон М. Node.js в действии. — 2-е изд.. — СПб.: «Питер», 2018. — С. 432. — ISBN 978-5-496-03212-4.

22. Andrii Sergiienko.Webrtc Blueprints/ Andrii Sergiienko; Packt Publishing 2014;

23. Jonathan Strickland. Is there a Web 1.0? [Электронный ресурс]. - URL: https://computer.howstuffworks.com/web-10.htm#pt1

24. Tim Ambler JavaScript Frameworks for Modern Web Dev [Электронный ресурс] : [учеб. пособие]/Tim Ambler, Nicholas Cloud. – 1е изд., - Apress, 2015. – С. 520 – ISBN 978-1484206638

25. Web 2.0 в образовании. [Электронный ресурс]. - URL: http://mozlicey.by/web2_0.php

26. WEBRTC VS WEBSOCKETS. [Электронный ресурс]. - URL: http://surl.li/bwgwb

27. Web 3.0, the “official” definition. [Электронный ресурс]. - URL: https://calacanis.com/2007/10/03/web-3-0-the-official-definition/

Приложение А

## Код файла index.js (серверная часть)

```
const app = require("express")();
const server = require("http").createServer(app);
const cors = require("cors");
const io = require("socket.io")(server, {
    cors: {
        origin: "*",
        methods: ["GET", "POST"]
    }
});
app.use(cors());
const PORT = process.env.PORT || 5000;
app.get("/", (req, res) => {
    res.send('Server is running.');
});
io.on('connection', (socket) => {
    socket.emit('me', socket.id);
    socket.on('disconnect', () =>{
        socket.broadcast.emit("callended");
    });
    socket.on("calluser", ({userToCall, signalData, from, name}) => {
        io.to(userToCall).emit("calluser", { signal: signalData, from, name});
    });
    socket.on("answercall", (data) => {
        io.to(data.to).emit("callaccepted",data.signal);
    });
});
server.listen(PORT, () => console.log(`Server listening on port ${PORT}`));
```

Приложение Б

## Код файла App.js

```
import React from 'react';
import { Typography, AppBar } from '@material-ui/core';
import { makeStyles } from '@material-ui/core/styles';
import VideoPlayer from './components/VideoPlayer';
import Options from './components/Options';
import Notifications from './components/Notifications';

const useStyles = makeStyles((theme) => ({
    appBar: {
      borderRadius: 15,
      margin: '30px 100px',
      display: 'flex',
      flexDirection: 'row',
      justifyContent: 'center',
      alignItems: 'center',
      width: '600px',
      border: '2px solid black',
      [theme.breakpoints.down('xs')]: {
        width: '90%',
      },
    },
    image: {
      marginLeft: '15px',
    },
    wrapper: {
      display: 'flex',
      flexDirection: 'column',
      alignItems: 'center',
      width: '100%',
    },
  }));
const App = () => {
```

```
const classes = useStyles();
  return (
    <div className={classes.wrapper}>
        <AppBar className={classes.appBar} position="static" color="inherit">
            <Typography variant="h2" align="center">Видеочат</Typography>
        </AppBar>
        <VideoPlayer/>
        <Options>
            <Notifications/>
        </Options>
    </div>
  )
}
export default App;
```

Приложение В

## Код файла SocketContext.js

```
import React, {createContext, useState, useRef, useEffect} from "react";
import { io } from "socket.io-client";
import Peer from 'simple-peer';

const SocketContext = createContext();
const socket = io('https://p2p-webrtc-chat.herokuapp.com');

const ContextProvider = ({ children }) => {
   const[stream, setStream] = useState(null);
   const [me, setMe] = useState('');
   const [call,setCall] = useState({});
   const [callAccepted, setCallAccepted] = useState(false);
   const [callEnded, setCallEnded] = useState(false);
   const [name, setName] = useState('');

   const connectionRef = useRef();
   const myVideo = useRef();
   const userVideo = useRef();

   useEffect(()=>{
      navigator.mediaDevices.getUserMedia({video:true, audio: true})
         .then((currentStream) => {
            setStream(currentStream);
            myVideo.current.srcObject = currentStream;
      });
      socket.on('me', (id) => setMe(id));
      socket.on('calluser', ({from, name: callerName, signal}) => {
         console.log(from, callerName, signal);

         setCall({isReceivingCall: true, from, name: callerName, signal})
      });

   },[]);

   const answerCall = () => {
      setCallAccepted(true)
      const peer = new Peer({initiator: false, trickle: false, stream});
      peer.on('signal',(data) =>{
         socket.emit('answercall', {signal: data, to: call.from})
      });
      peer.on('stream',(currentStream)=>{
         userVideo.current.srcObject = currentStream;
      });
      peer.signal(call.signal);
      connectionRef.current = peer;
   }
```

```
const callUser = (id) => {
        console.log(me,socket);
        try{
            const peer = new Peer({initiator: true, trickle: false, stream});
        peer.on('signal',(data) =>{
            socket.emit('calluser', {userToCall:id, signalData:data, from: me, name});
            console.log('1');
        });
        peer.on('stream',(currentStream)=>{
            userVideo.current.srcObject = currentStream;
            console.log('2');
        });
        socket.on('callaccepted',(signal) => {
            setCallAccepted(true);
            peer.signal(signal);
            console.log('3');
        });
        connectionRef.current = peer;
        }catch(e){
            console.log(e)
        }

    }

    const leaveCall = () => {
        setCallEnded(true);
        connectionRef.current.destroy();
        window.location.reload();
    }
    return (
        <SocketContext.Provider value={{ call, callAccepted, myVideo, userVideo, stream, name,
setName, callEnded, me, callUser, leaveCall, answerCall}}>
            {children}
        </SocketContext.Provider>
    )
}

export {ContextProvider, SocketContext};
```

Приложение Г

## Код компонента Videoplayer

```
import React, { useContext } from 'react';
import { Grid, Typography, Paper} from '@material-ui/core';
import { makeStyles } from '@material-ui/core/styles';
import {SocketContext} from '../SocketContext';
const useStyles = makeStyles((theme) => ({
    video: {
      width: '550px',
      [theme.breakpoints.down('xs')]: {
        width: '300px',
      },
    },
    gridContainer: {
      justifyContent: 'center',
      [theme.breakpoints.down('xs')]: {
        flexDirection: 'column',
      },
    },
    paper: {
      padding: '10px',
      border: '2px solid black',
      margin: '10px',
    },
  }));
const VideoPlayer = () => {
    const classes = useStyles();
    const {name, callAccepted, myVideo, userVideo, callEnded, stream, call} =
useContext(SocketContext);
  return (
    <Grid container className={classes.gridContainer}>
        { stream && (
            <Paper className={classes.paper}>
            <Grid item xs={12} md={6}>
```

```
<Typography variant="h5" gutterBottom>{name || 'Имя пользователя'}</Typography>
              <video playsInline muted ref={myVideo} autoPlay className={classes.video}/>
            </Grid>
          </Paper>
        )}
        { callAccepted && !callEnded && (
          <Paper className={classes.paper}>
          <Grid item xs={12} md={6}>
            <Typography variant="h5" gutterBottom>{call.name || 'Имя
пользователя'}</Typography>
            <video playsInline ref={userVideo} autoPlay className={classes.video}/>
          </Grid>
        </Paper>
        )}
    </Grid>
  );
}
export default VideoPlayer
```

Приложение Д

## Код компонента Options

```
import React, { useState, useContext } from 'react';
import {Button, TextField, Grid, Typography, Container, Paper} from '@material-ui/core';
import { makeStyles } from '@material-ui/core/styles';
import { CopyToClipboard } from 'react-copy-to-clipboard';
import { Assignment, Phone, PhoneDisabled } from '@material-ui/icons';
import { SocketContext } from '../SocketContext';
const useStyles = makeStyles((theme) => ({
  root: {
    display: 'flex',
    flexDirection: 'column',
  },
  gridContainer: {
    width: '100%',
    [theme.breakpoints.down('xs')]: {
      flexDirection: 'column',
    },
  },
  container: {
    width: '600px',
    margin: '35px 0',
    padding: 0,
    [theme.breakpoints.down('xs')]: {
      width: '80%',
    },
  },
  margin: {
    marginTop: 20,
  },
  padding: {
    padding: 20,
  },
  paper: {
```

```
      padding: '10px 20px',
     border: '2px solid black',
   },
 }));

const Options = ( {children}) => {
   const {me, callAccepted, name, setName, callEnded, leaveCall, callUser} =
useContext(SocketContext);
   const [idToCall, setIdToCall] = useState('');
   const classes = useStyles();
 return (
   <Container className={classes.container}>
      <Paper elevation={10} className={classes.paper}>
         <form className={classes.root} noValidate autoComplete="off">
            <Grid container className={classes.gridContainer}>
               <Grid item xs={12} md={6} className={classes.padding}>
                  <Typography gutterBottom variant="h6">Информация</Typography>
                  <TextField label="Введите ваше имя" value={name} onChange={(e) =>
setName(e.target.value)} fullWidth/>
                  {console.log(me)}
                     <CopyToClipboard text={me} className={classes.margin}>
                        <Button variant="contained" color="primary" fullWidth
startIcon={<Assignment fontSize="large"/>}>
                           Копировать ваш ID
                        </Button>
                     </CopyToClipboard>
               </Grid>
               <Grid item xs={12} md={6} className={classes.padding}>
            <Typography gutterBottom variant="h6">Позвонить</Typography>
            <TextField label="ID кому звонить" value={idToCall} onChange={(e) =>
setIdToCall(e.target.value)} fullWidth />
            {callAccepted && !callEnded ? (
```

```
          <Button variant="contained" color="secondary" startIcon={<PhoneDisabled
fontSize="large" />} fullWidth onClick={leaveCall} className={classes.margin}>
                Отклонить
              </Button>
            ) : (
              <Button variant="contained" color="primary" startIcon={<Phone fontSize="large" />}
fullWidth onClick={() => callUser(idToCall)} className={classes.margin}>
                Вызов
              </Button>
            )}
          </Grid>
            </Grid>
          </form>
          {children}
        </Paper>
    </Container>
  )
}
export default Options
```

Приложение Е

## Код компонента Notifications

```
import React, { useState, useContext } from 'react';
import {Button, TextField, Grid, Typography, Container, Paper} from '@material-ui/core';
import { makeStyles } from '@material-ui/core/styles';
import { CopyToClipboard } from 'react-copy-to-clipboard';
import { Assignment, Phone, PhoneDisabled } from '@material-ui/icons';
import { SocketContext } from '../SocketContext';
const useStyles = makeStyles((theme) => ({
  root: {
    display: 'flex',
    flexDirection: 'column',
  },
  gridContainer: {
    width: '100%',
    [theme.breakpoints.down('xs')]: {
      flexDirection: 'column',
    },
  },
  container: {
    width: '600px',
    margin: '35px 0',
    padding: 0,
    [theme.breakpoints.down('xs')]: {
      width: '80%',
    },
  },
  margin: {
    marginTop: 20,
  },
  padding: {
    padding: 20,
  },
  paper: {
```

```
    padding: '10px 20px',
    border: '2px solid black',
  },
 }));

const Options = ( {children}) => {
  const {me, callAccepted, name, setName, callEnded, leaveCall, callUser} =
useContext(SocketContext);
  const [idToCall, setIdToCall] = useState('');
  const classes = useStyles();
 return (
  <Container className={classes.container}>
    <Paper elevation={10} className={classes.paper}>
      <form className={classes.root} noValidate autoComplete="off">
        <Grid container className={classes.gridContainer}>
          <Grid item xs={12} md={6} className={classes.padding}>
            <Typography gutterBottom variant="h6">Информация</Typography>
            <TextField label="Введите ваше имя" value={name} onChange={(e) =>
setName(e.target.value)} fullWidth/>
            {console.log(me)}
              <CopyToClipboard text={me} className={classes.margin}>
                <Button variant="contained" color="primary" fullWidth
startIcon={<Assignment fontSize="large"/>}>
                  Копировать ваш ID
                </Button>
              </CopyToClipboard>
          </Grid>
          <Grid item xs={12} md={6} className={classes.padding}>
        <Typography gutterBottom variant="h6">Позвонить</Typography>
        <TextField label="ID кому звонить" value={idToCall} onChange={(e) =>
setIdToCall(e.target.value)} fullWidth />
        {callAccepted && !callEnded ? (
```

```
            <Button variant="contained" color="secondary" startIcon={<PhoneDisabled
fontSize="large" />} fullWidth onClick={leaveCall} className={classes.margin}>
              Отклонить
            </Button>
          ) : (
            <Button variant="contained" color="primary" startIcon={<Phone fontSize="large" />}
fullWidth onClick={() => callUser(idToCall)} className={classes.margin}>
              Вызов
            </Button>
          )}
        </Grid>
          </Grid>
        </form>
        {children}
      </Paper>
    </Container>
  )
}

export default Options
```