Содержание

Введение

На данный момент существует множество программ, реализующих системы телеприсутствия, но лишь относительно немногие компании разрабатывают такие системы для использования на мобильных роботах. Хотя в учебной индустрии, индустриях медицины и военных технологий, горном деле, ядерных технологий, исследовании и разработке подводных ресурсов ощущается необходимость в подобных системах.

Система телеприсутствия для мобильного робота будет полезна во многих сферах, где физическое присутствие человека по каким-либо причинам невозможно, из-за опасности нахождения вблизи объекта исследования, в связи с болезнью или другими причинами.

Одним из примеров использования можно описать телеприсутствие ученика на уроке. Часто бывает, что ученики по каким-либо причинам не могут посетить учебное заведение, в связи с чем происходит недопонимание материала. Использование мобильного робота с системой телеприсутствия поможет учебным заведениям повысить успеваемость и сократить отставание учеников по программе.

В связи с вышеописанным появилась идея реализации системы телеприсутствия для мобильного робота.

Данная работа посвящена разработке системы телеприсутствия, которая может быть использована на мобильном роботе.

1. **Техническое задание**

1.1 Назначение разработки и область применения

Разрабатываемая система предназначена для телеприсутствия в месте, где нет возможности находиться физически в данный момент. Данная система может применяться в учебной, медицинской и военной областях, горном деле и других областях, где требуется удаленное присутствие человека и выполнение каких-либо действий.

1.2 Технические требования

Требования, предъявляемые разрабатываемой системе к ЭВМ:

1. Операционная система с графическим интерфейсом;
2. Требования к аппаратному обеспечению определяются операционной системой;
3. Мышь, дисплей, веб камера, микрофон.

Требования к функционалу разрабатываемой системы телеприсутствия для мобильного робота:

1. Система должна предоставить возможность пользователю выбирать комнату для установки соединения
2. Система должна предоставить возможность пользователю отправлять запрос на подключение к существующей комнате
3. Система должна предоставить возможность пользователю подтвердить или отклонить запрос на подключение собеседника к комнате
4. Система должна осуществлять установку соединения между пользователями
5. Система должна предоставить возможность пользователю отправлять управляющие команды на робота
6. Система должна предоставить возможность пользователю включать и выключать передачу видео и звука
7. Система должна предоставить возможность пользователю возможность открывать получаемое видео на весь экран

2. **Анализ технического задания**

2.1 Выбор операционной системы

В данный момент самыми распространенными операционными системами являются Windows, MacOS и Linux. Рассмотрим каждую из них.

Windows – это семейство коммерческих операционных систем, разработанное корпорацией Microsoft. Windows имеет дружественный и понятный графический интерфейс, в связи с чем имеет большую популярность среди пользователей. Данная линейка операционных систем имеет большую область применений. Она используется как на персональных компьютерах, так и на смартфонах, в настоящий момент разработка мобильной платформы прекращена из-за конкуренции. Так же имеются серверные операционные системы на базе Windows.

Linux – это семейство Unix-подобных операционных систем, разрабатываемое программистами по всему миру. Это свободное программное обеспечение и распространяется на бесплатной основе, что является плюсом данной системы. Так же Linux является хорошо расширяемой системой и готовой к установке обновлений и сопровождения. Но Linux является менее безопасной системой, а также обладает худшей поддержкой периферийного оборудования, чем Windows, такого как сканеры, принтеры и т.д.

MacOS – операционная система, разработанная фирмой Apple, известная раньше, как Macintosh Operating System. Она является Unix-подобной операционной системой. По популярности занимает второе место, после ОС Windows. Также имеются серверные операционные системы на базе MacOS.

Создание программного обеспечения для каждой из перечисленных операционных систем слишком затратно и не удобно, поэтому разрабатываемое программное обеспечение не должно быть привязано к конкретной операционной системе и иметь возможность запуска на любой платформе.

2.2 Выбор языка программирования

Для разрабатываемой системы телеприсутствия для мобильного робота могут быть применены следующие языки программирования: JavaScript, Python, C++. Рассмотрим подробнее каждый из них.

Python – высокоуровневый язык программирования, который хорошо интегрируется с существующими программами и обладает высокой читаемостью кода. Но в то же время для его использования необходим интерпретатор, что не всегда удобно.

C++ - язык программирования, обладающий высокой скоростью выполнения программного кода, а так же поддерживающий работу с графикой. Язык программирования, который имеет в своем составе большую стандартную библиотеку с множеством функций.

JavaScript – динамический язык программирования, который обладает универсальностью. С помощью него можно создавать как серверную часть приложения, так и интерактивную клиентскую часть. Программные интерфейсы (API), встроенные в браузеры, обеспечивают такие функциональные возможности как динамическое создание HTML и установку CSS, захват и манипуляция видеопотоком и работа с веб-камерой пользователя.

Проанализировав плюсы и минусы рассматриваемых языков программирования, можно сделать вывод, что наилучшим вариантом будет использование языка JavaScript.

2.3 Выбор среды разработки

В данном проекте будет использоваться интегрированная среда разработки (IDE). Данная среда представляет собой совокупность программных средств, обеспечивающих поддержку всех этапов разработки проекта, начиная от написания кода и заканчивая отладкой.

Сейчас существует множество интегрированных сред разработки, как платных, так и бесплатных. Большинство из них предназначены для написания программ на каком-то одном языке программирования. Однако, существуют и такие среды разработки, которые поддерживают сразу несколько языков программирования. Именно такие среды разработки и будут рассматриваться для написания данной работы.

Visual Studio Code – редактор исходного кода, разработанный Microsoft для Windows, Linux и MacOS. Поддерживает ряд языков программирования, подсветку синтаксиса, технологию автодополнения, рефакторинг, отладку, навигацию по коду, поддержку Git и другие возможности.

Eclipse – свободная среда разработки, изначально создаваемая для Java-приложений, сейчас же поддерживает и такие языки программирования как C, C++, JavaScript и другие. В этой IDE имеется возможность создания графических интерфейсов, построения диаграмм, составления отчетов, тестирования и многое другое. Но все-таки Eclipse больше нацелен на разработку Java-приложений и при создании программного обеспечения на языке, выбранном для разрабатываемой системы, а именно на языке JavaScript, возникает множество сложностей при установке и дальнейшей работе с этой средой разработки.

Netbeans – свободная интегрированная среда разработки приложений на языках программирования Java, Python, JavaScript и ряда других. В Netbeans декларируется поддержка UML, SOA, а также средств для создания приложений на J2ME для мобильных телефонов. Для тестирования доступны отчеты. Но все же Netbeans больше нацелен на разработку Java-приложений, к тому же без предустановленного Sun JDK или J2EE SDK установка среды невозможна.

Учитывая все достоинства и недостатки рассмотренных IDE, в качестве среды разработки был выбран Visual Studio Code за свою легкость, интуитивно понятный интерфейс и наличие всех необходимых зависимостей для разработки веб приложений.

2.4 Обзор существующих систем телеприсутствия

Рассмотрим несколько наиболее популярных систем телеприсутствия.

Система телеприсутствия MAX PRESENCE с эффектом присутствия от Huawei. Используются IP-средства многоканальной связи позволяющие совмещать видеовызовы, голосовые сервисы и передачу данных в режиме реального времени. Видеоизображение воспроизводится на десяти 70-дюймовых HD-экранах, объединенных в один ультраширокий экран.

Рисунок 1. Обзор системы телеприсутствия MAX PRESENCE

Существенными недостатками данной системы являются:

- Необходимость в 10 70-дюймовых экранах

- Необходимость использования запатентованных камер Huawei

- Минимальный размер помещения (В х Ш х Г) 3000 мм х 7800 мм х 15000 мм

Система телеприсутствия Polycom EagleEye Director II. Имеет автоматическое обнаружение активных динамиков и точную обрезку изображения на дисплее за счет распознавания лиц с одновременной триангуляцией по голосу. Увеличивает изображение выступающего, за счет чего лучше понимаются невербальные сигналы. Предоставляется аналитическая информация, такая как отслеживание использования помещений, что позволяет проверить посещаемость автоматических конференций и убедиться, что они не проводятся в пустых комнатах.

Рисунок 2. Обзор системы Polycom EagleEye Director II

Существенными недостатками данной системы являются:

- Необходимость в использовании совместимых камер той же фирмы изготовителя
- Необходимость использования совместимых видеосистем той же фирмы изготовителя

3. **Разработка системы на структурном уровне**

3.1 Разработка общей структуры системы

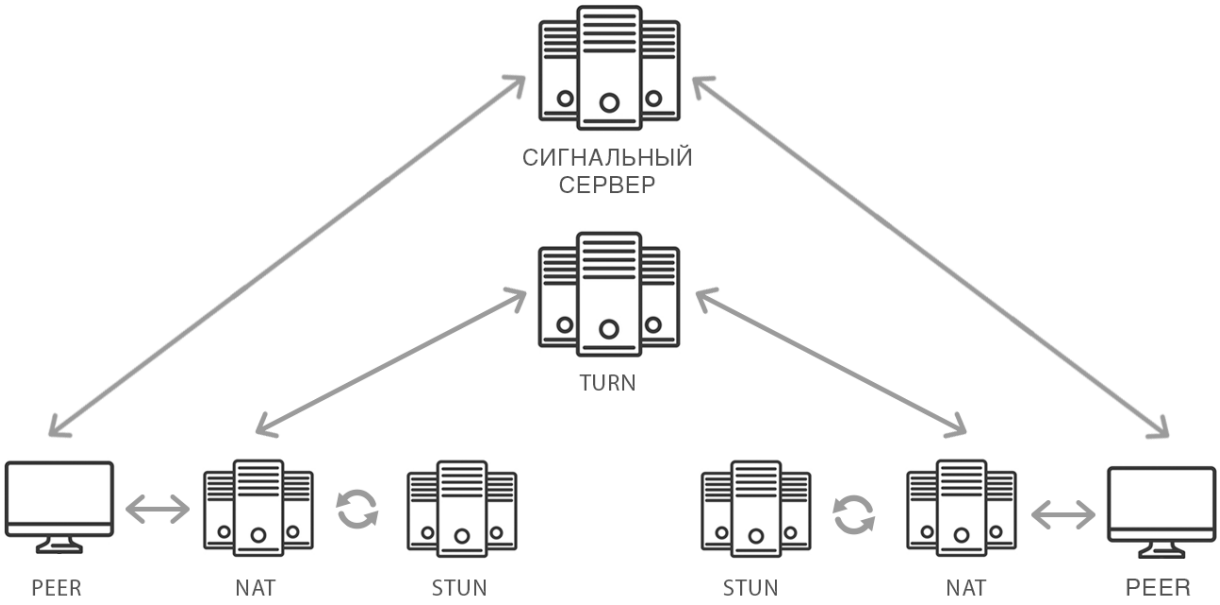

Рисунок 3. Общая структурная схема

Структурная схема системы телеприсутствия включает в себя ПК пользователя, с которого происходит подключение, сигнальный сервер, который отвечает за коммуникацию между клиентами, инициализацию и закрытие соединения, механизм NAT, который транслирует внутренние локальные IP-адреса, расположенные за файрволом, во внешние глобальные IP-адреса, STUN-сервер, сервер в интернете, который возвращает обратный адрес, то есть адрес узла отправителя и TURN-сервер, улучшенная версия STUN-сервера, который может работать как и STUN-сервер, но если peer-to-peer коммуникация невозможна, то TURN-сервер переходит в режим ретранслятора.

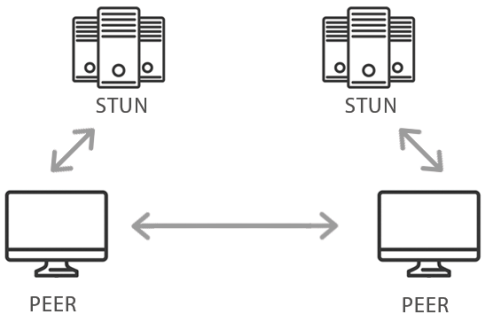

Рисунок 4. Схема подключения с использованием STUN-серверов

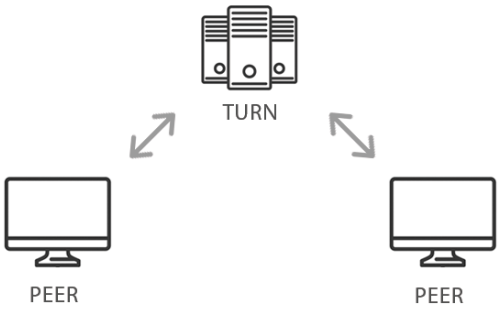

Рисунок 5. Схема подключения с использованием TURN-сервера

3.2 Логика работы системы

Основной частью любой системы является логика ее работы, обеспечиваемая программной частью. Для реализации подключения пользователя к определенной или случайной комнате был разработан алгоритм.

Программа сначала проверяет найдена ли комната с указанным номером, после чего либо выполняется создание комнаты и последующее подключение в нее, либо проверка на количество пользователей в комнате в текущий момент. Если пользователей в комнате 2, то выводится информационное сообщение, что комната переполнена, в другом случае происходит подключение к выбранной комнате.

Рисунок 6. Блок-схема подключения к комнате

Жизненный цикл программы предусматривает подключение к комнате, где производится отправка запроса на подключение, либо ожидание запроса на подключение, далее установку соединения и создание конференции между участниками комнаты и последующее завершение конференции с разрывом соединения и переходом к ожиданию нового запроса на создание конференции.

Рисунок 7. Жизненный цикл программы

4. Разработка системы на программном уровне

4.1 Выбор подхода к решению задачи передачи потоковых аудиоданных и видеоданных от браузера к браузеру в режиме реального времени

Для решения задачи передачи потоковых аудиоданных и видеоданных от браузера к браузеру была выбрана технология WebRTC.

Технология WebRTC – с помощью нее работают Google Hangouts, Facebook Messenger. Это новая развивающаяся технология, она появилась в 2011 году и продолжает развиваться. Она позволяет получить доступ к камере и микрофону. Установить peer-to-peer – соединение между двумя компьютерами, двумя браузерами. Позволяет передавать медиапотоки в режиме реального времени. Кроме того, с её помощью можно передавать информацию, то есть любую бинарную дату. Но WebRTC не предоставляет браузерам способ найти друг друга.

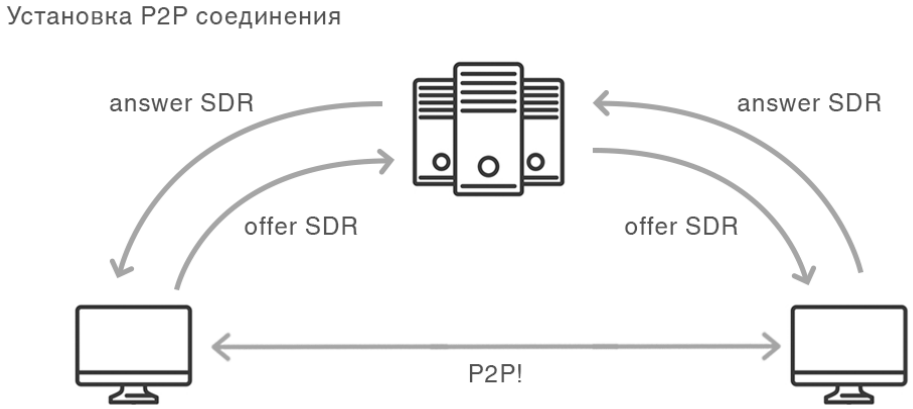

Рисунок 8. Установка peer-to-peer соединения

Рассмотрим пример установки соединения с использованием сигнального сервера. Есть сигнальный сервер, который держит постоянное соединение с клиентами, например, по WebSocket или с помощью HTTP. Первый клиент формирует метаинформацию, и с помощью WebSocket или HTTP пересылает ее на сигнальный сервер. Пересылает также какую-то часть информации, с кем именно он хочет соединиться, например, никнейм или еще какой-то идентификатор.

Сигнальный сервер по этому идентификатору устанавливает какому именно клиенту нужно переадресовать метаинформацию, и пересылает ее. Второй клиент берет эту метаинформацию, использует, устанавливает себе, формирует ответ, и с помощью сигнального механизма пересылает ее на сигнальный сервер, тот в свою очередь

ретранслирует ее первому клиенту. Таким образом оба клиента в данный момент обладают всей необходимой датой и метаинформацией, чтобы установить peer-to-peer -соединение.

Клиенты обмениваются SDP-датаграммой, Session Description Protocol.

SDP-датаграмма

```
v=0
o=- 1815849 0 IN IP4 192.168.0.15
s=Cisco SDP 0
c=IN IP4 194.67.15.181
t=0 0
m=audio 20062 RTP/AVP 99 18 101 100
a=rtpmap:99 G.729b/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=rtpmap:100 X-NSE/8000
a=fmtp:100 200-202
```

Рисунок 9. SDP-датаграмма

Это текстовый файл, который содержит всю необходимую информацию, чтобы установить соединение. Там есть информация об IP-адресе, о портах, которые используются, о том, какая именно информация гоняется между клиентами, что это такое – аудио, видео, какие кодеки используются.

Во второй строчке SDP-датаграммы указан IP-адрес компьютера, который находится в локальной сети. Если два компьютера, каждый из которых находится в своей локальной сети, каждый из которых знает свой IP-адрес в рамках этой сети, хотят установить соединение. С такой датаграммой это сделать не получится, поскольку они не знают своих внешних IP-адресов.

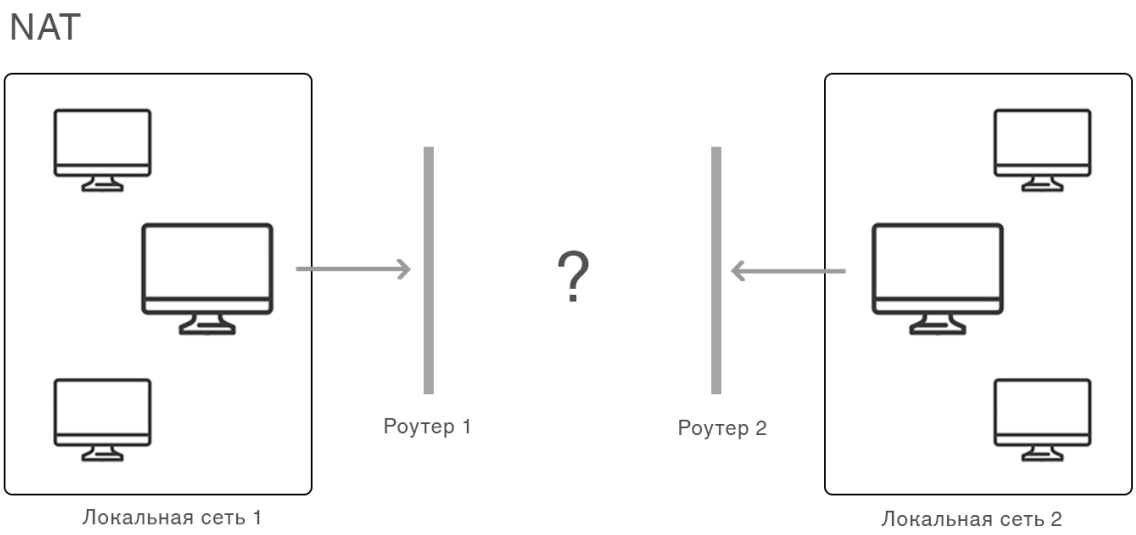

Рисунок 10. NAT

Внешние IP-адреса скрыты за NAT. Рассмотрим работу NAT. Есть локальные сети, внутри которых компьютеры знают свои адреса, есть роутер, который обладает внешним IP-адресом, и наружу все эти компьютеры выходят с IP-адресом этого роутера. Когда пакет от компьютера в локальной сети идет на роутер, роутер смотрит, куда его нужно переадресовать. Если на другой компьютер этой локальной сети, то он просто его ретранслирует, а если нужно его отправить вовне, в интернет, то составляется таблица маршрутизации.

Таблица маршрутизации

внутренний IP	внутренний порт	внешний IP	внешний порт
192.168.0.15	35825	10.6.201.7	820

Рисунок 11. Таблица маршрутизации

Заполняется внутренний IP-адрес компьютера, который желает переслать пакет, его порт, ставится внешний IP-адрес, IP-адрес роутера, и делается также подмена порта. Это делается для предотвращения путаницы внутри локальной сети. Например, два компьютера обращаются к одному и тому же ресурсу, и нужно правильно смаршрутизировать ответные

пакеты. Их будут идентифицировать по порту, порт будет по каждому из компьютеров уникальным, в то время как внешний IP-адрес будет совпадать.

Фреймворк ICE – Internet Connectivity Establishment. Он описывает способы обхода NAT, способы установки соединения, если есть NAT.

Этот фреймворк использует приписывание так называемого STUN-сервера.

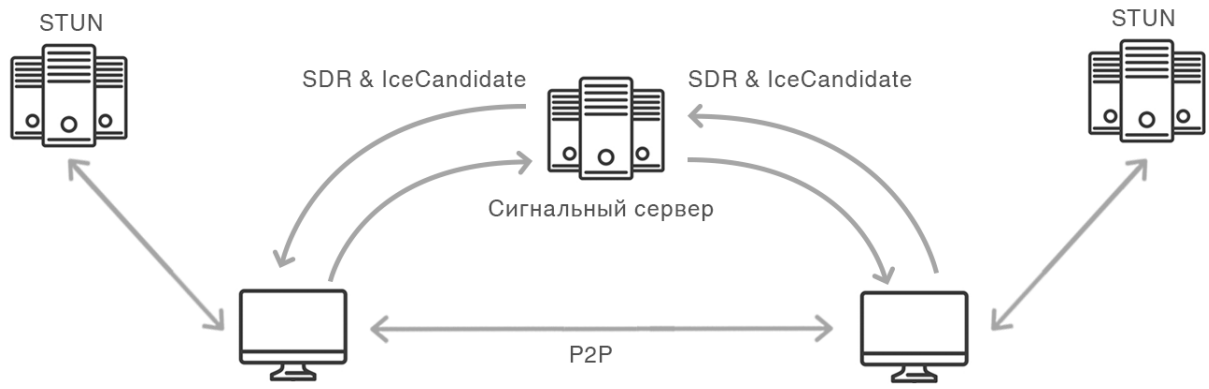

Рисунок 12. Peer-to-peer - соединение с использованием STUN-сервера

Это специальный сервер, обращаясь к которому, можно узнать внешний IP-адрес компьютера, с которого происходит обращение. Таким образом, в процессе установки peer-to-peer - соединения, каждый из клиентов должен сделать по запросу к этому STUN-серверу, чтобы узнать свой IP-адрес, и сформировать дополнительную информацию, IceCandidate, и с помощью сигнального механизма также этим IceCandidate обменяться. Тогда клиенты будут знать друг о друге с правильными IP-адресами, и смогут установить peer-to-peer - соединение.

Если компьютер скрыт за двойным NAT использование STUN-сервера не будет иметь результата. В этом случае фреймворк ICE предписывает использование TURN-сервера.

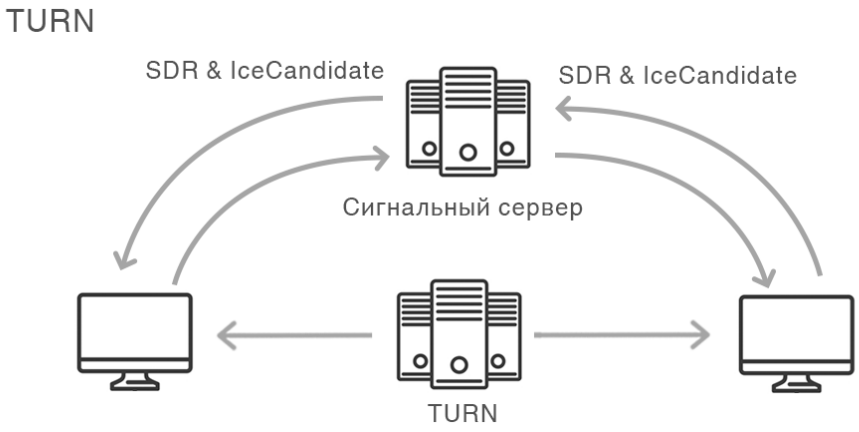

Рисунок 13. Peer-to-peer - соединение с использованием TURN-сервера

Это специальный сервер, который превращает соединение клиент-клиент, peer-to-peer, в соединение клиент-сервер-клиент, то есть выступает в роли ретранслятора.

Итого, чтобы установить соединение, нужно выбрать и реализовать некий механизм сигнализации, посредника, который будет помогать пересылать метаинформацию. WebRTC дает всю необходимую метаинформацию для этого, из-за чего и был выбран в качестве способа реализации задачи.

Рисунок 14. Медиа-потоки

Медиа-потоки – это каналы, которые внутри себя содержат треки. Треки внутри медиа-потока синхронизированы. Аудио и видео не будет расходиться, они будут идти с единым временем. Можно внутри медиапотока сделать любое количество треков, треками можно управлять по отдельности, например, можно приглушить аудио, оставив только картинку. Также можно передавать любое количество медиа-потоков, что позволяет, например, реализовать конференцию.

```
navigator.mediaDevices.getUserMedia(constraints, successCallback, errorCallback);


navigator.mediaDevices.enumerateDevices();
```

Рисунок 15. Получение доступа к медиа

Метод getUserMedia принимает на вход набор ограничений (констрейнтов). Это специальный объект, где указывается, к каким именно устройствам нужно получить доступ, к какой именно камере, к какому микрофону. Указываются характеристики, которые необходимо иметь, какое именно разрешение, и есть также два аргумента – successCallback и errorCallback, которые вызываются в случае успеха или неудачи.

Есть также удобный метод enumerateDevices, который возвращает список всех подключенных к компьютеру медиа-устройств, что дает возможность показать их

пользователю, нарисовать какой-то селектор, чтобы пользователь мог выбрать, какую конкретно камеру можно использовать.

```
const peerConnection = new RTCPeerConnection({
    iceServers: [
     {
       urls: "turn:" + turnHost,
       username: "webrtc",
       credential: "turnserver"
     },
     …
    ]
});


peerConnection.onicecandidate = function (evt) { };
```

Рисунок 16. Создание peer-to-peer - соединения

Центральным объектом в API служит RTCPeerConnection. Когда выполняется соединение, то берется класс RTCPeerConnection, который возвращает объект peerConnection. В качестве конфигурации указывается набор ICE-серверов, то есть STUN- и TURN-серверов, к которым будет происходить обращение в процессе установки. Ивент onicecandidate, срабатывает каждый раз, когда нужна помощь сигнального механизма. То есть технология WebRTC сделала запрос, например, к STUN-серверу, компьютеры узнали внешние IP-адреса, появился новый сформированный ICECandidate, и нужно переслать его с помощью стороннего механизма, в этот момент ивент срабатывает.

```
peerConnection.createOffer();

peerConnection.setLocalDescription();

peerConnection.setRemoteDescription();

peerConnection.createAnswer();
```

Рисунок 17. Создание и установка SDP

Когда устанавливается соединение и нужно проинициализировать вызов, используется метод createOffer(), чтобы сформировать начальную SDP, offer SDP, мета-информацию, которую нужно переслать партнеру.

Чтобы установить ее в PeerConnection, используется метод setLocalDescription(). Собеседник получает эту информацию по сигнальному механизму, устанавливает себе с

помощью метода setRemoteDescription() и формирует ответ с помощью метода createAnswer(), который с помощью сигнального механизма пересылается первому клиенту.

```
peerConnection.addStream(stream);

peerConnection.onaddstream = function (evt) { };
```

Рисунок 18. Работа с потоками

Когда есть доступ к медиа, имеется медиапоток, он передается в peer-to-peer - соединение с помощью метода addStream, а собеседник узнает об этом, у него сработает ивент onaddstream. Он получит медиапоток и сможет его отобразить.

WebRTC работает только по протоколу HTTPS, сайт должен быть подписан сертификатом. Медиапотоки тоже шифруются, используется протокол DTLS. Технология не требует установки ничего дополнительного, что облегчает ее использование убирая лишние зависимости.

4.2 Выбор подхода к решению задачи передачи управляющих сигналов на робота

Для решения задачи передачи управляющих сигналов был выбран протокол SSH. Он позволяет производить туннелирование TCP-соединений и используется для удаленного управления.

Разрабатываемый облачный сервис подключается к роботу через обратный SSH-туннель. Использование SSH-туннелирования обусловлено шифрованием данных, передаваемых в этом туннеле, а именно отправкой команд на выполнение скрипта на стороне робота.

Управление движением робота реализовано следующим способом. При нажатии управляющей кнопки в приложении происходит выполнение скрипта, который отправляет роботу по обратному SSH-туннелю команду. На роботе, в результате получения команды, начинает исполняться скрипт на отправку управляющей команды в последовательный порт.

Функции серверной части для вызова скриптов на роботе:

```
function forward() {

      const forward = spawn('ssh', ['-p 11000', 'pi@localhost', 'expect -f
      forward']);
}

function backward() {
        const backward = spawn('ssh', ['-p 11000', 'pi@localhost', 'expect -f
      backward']);
}

function left() {
        const left = spawn('ssh', ['-p 11000', 'pi@localhost', 'expect -f
      left']);
}

function right() {
        const right = spawn('ssh', ['-p 11000', 'pi@localhost', 'expect -f
      right']);
}

function stop() {
        const stop = spawn('ssh', ['-p 11000', 'pi@localhost', 'expect -f
      stop']);
}
```

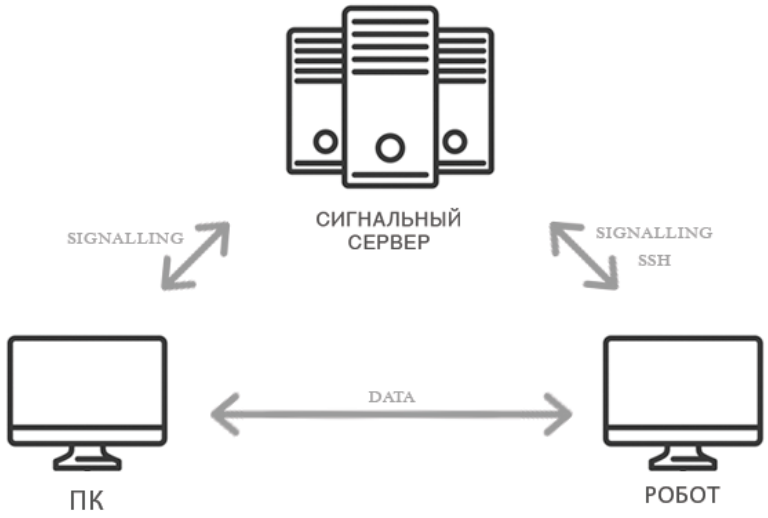

Рисунок 19. Структурная схема подключения сигнального сервера к роботу

5. Презентация разрабатываемой системы

Рассмотрим подробнее интерфейсную часть разрабатываемой программной системы. На главной странице присутствуют две кнопки, “Join”, отвечающая за подключение к определенной комнате, номер комнаты вводится в соответствующее поле, и кнопка “Random”, при нажатии на которую выбирается случайная комната. Также есть возможность подключения в комнаты, которые были использованы ранее, переход в них осуществляется при нажатии на гиперссылку с номером комнаты.

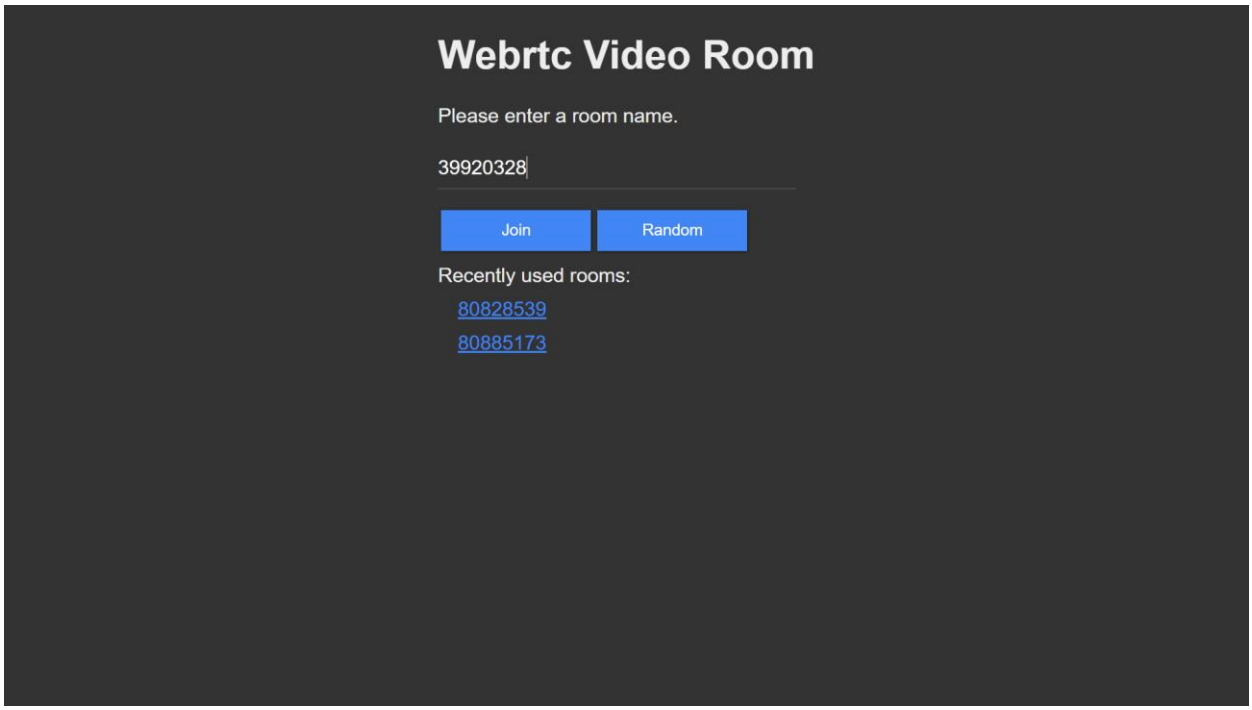

Рисунок 20. Главная страница

Обработчики событий подключения к определенной комнате по кнопке “Join” и к случайной комнате по кнопке “Random” описаны в файлах исходного кода Home.js и HomePage.js, представленных в приложении.

При первом открытии страницы комнаты запрашивается разрешение на использование микрофона и камеры, в зависимости от настроек браузера выбор запоминается или запрашивается каждый раз вновь.

На странице комнаты есть четыре возможных состояния. При первом состоянии ожидается подключение собеседника, при втором отправляется запрос на установку соединения с собеседником, при третьем состоянии происходит получение запроса на установку соединения, при четвертом состоянии соединение установлено и происходит конференция.

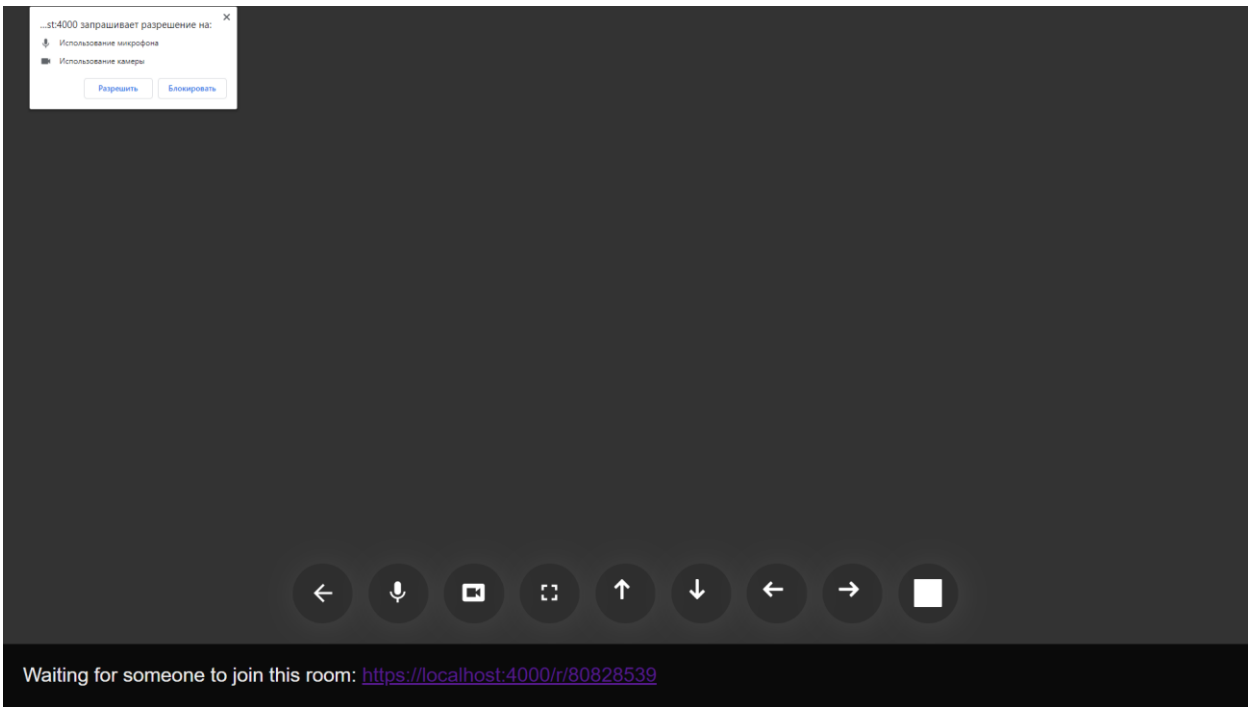

Рисунок 21. Страница комнаты. Состояние ожидания подключения собеседника и запрос на разрешение использования медиаданных

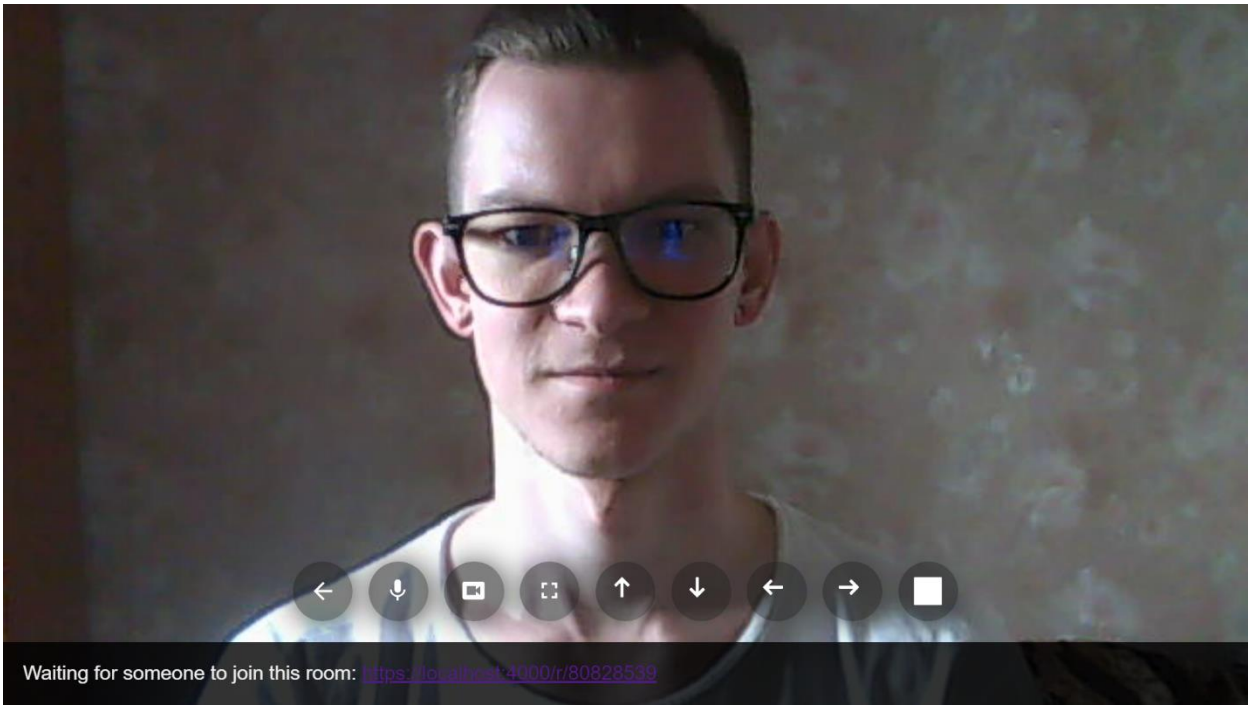

Рисунок 22. Страница комнаты. Состояние ожидания подключения собеседника с разрешенным доступом к медиаданным

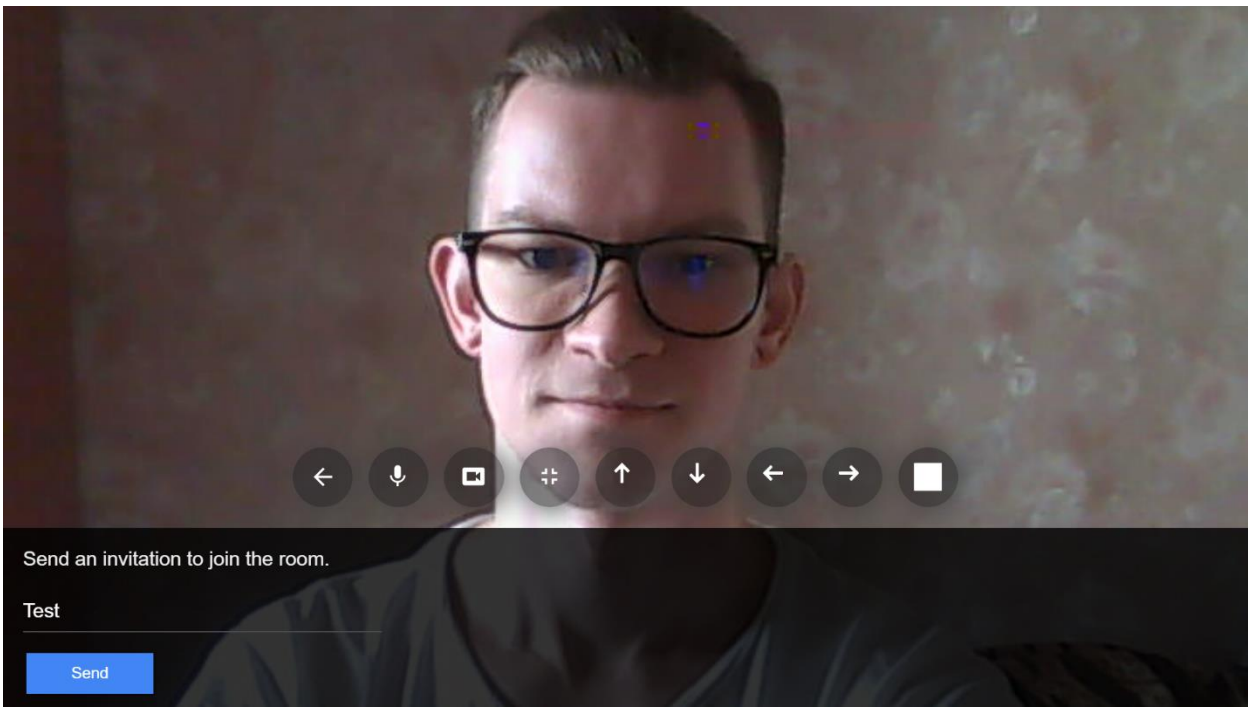

Рисунок 23. Страница комнаты. Состояние отправки запроса на установку соединения

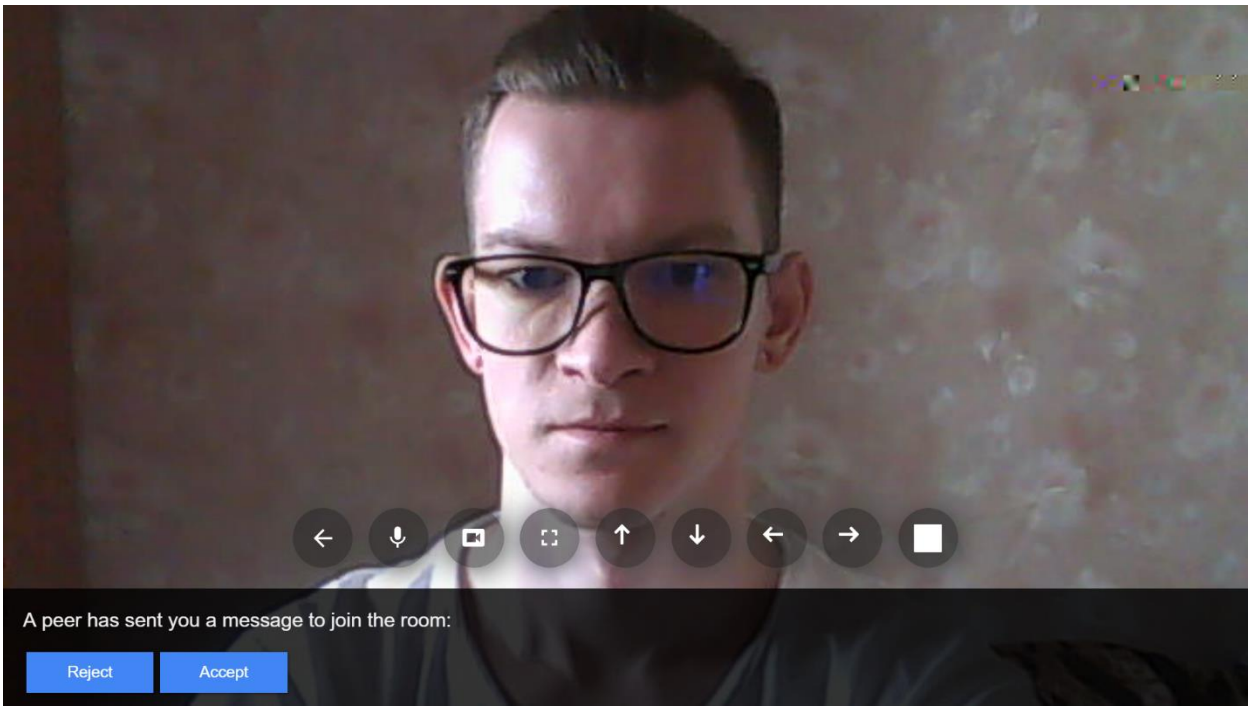

Рисунок 24. Страница комнаты. Состояние получение запроса на установку соединения

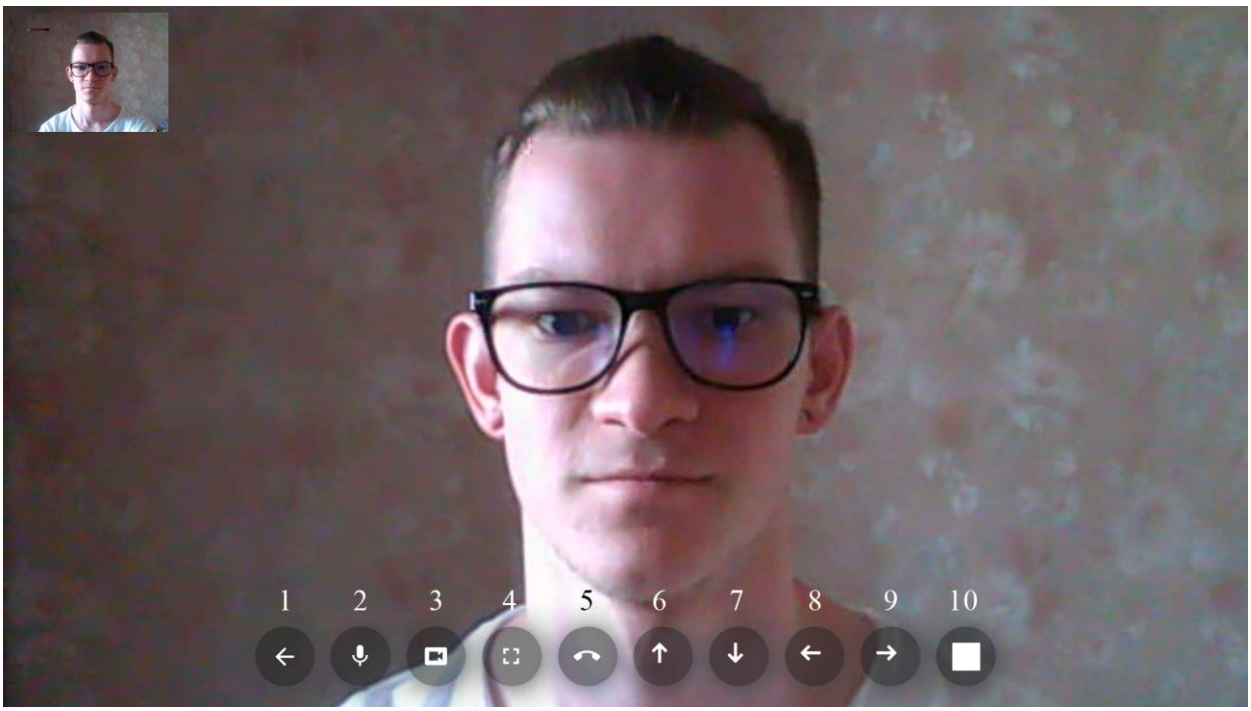

Рисунок 25. Страница комнаты. Состояние конференции

Обработчики событий этой страницы описываются в файлах исходного кода RoomPage.js, Communication.js, CommunicationContainer.js, ToggleFullScreen.js и MediaContainer.js. Эти файлы также представлены в приложении.

В файле Communication.js реализуются следующие функции:

1. Кнопка “call-exit-button” реализует разрыв соединения и переход в главное меню
2. Кнопка “audio-button” реализует отключение микрофона и обратное включение при повторном нажатии
3. Кнопка “video-button” реализует отключение захвата веб-камеры и обратное включение при повторном нажатии
4. Кнопка “fullscreen-button” реализует открытие страницы на весь экран, при повторном нажатии возвращает исходный размер
5. Кнопка “hangup-button” реализует разрыв соединения с последующим ожиданием собеседника в текущей комнате
6. Кнопка “forward” выполняет отправку команды на робота для движения вперед
7. Кнопка “backward” выполняет отправку команды на робота для движения назад
8. Кнопка “left” выполняет отправку команды на робота для движения налево
9. Кнопка “right” выполняет отправку команды на робота для движения направо
10. Кнопка “stop” выполняет отправку команды на робота для остановки движения.

6. **Тестирование системы**

Тестирование программной системы телеприсутствия для мобильного робота – процесс исследования, испытания данной системы, имеющий следующие цели: демонстрация работоспособности системы, соответствие предъявленным требованиям; выявить ситуации, в которых поведение системы является неправильным, нежелательным или несоответствующим спецификации. Один из основных видов проводимого тестирования – функциональное тестирование.

Функциональное тестирование рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом. К видам функционального тестирования относятся функциональное тестирование, тестирование пользовательского интерфейса, тестирование безопасности и тестирование взаимодействия.

В качестве тестирования данной системы телеприсутствия для мобильного робота можно выделить следующие функции, которые необходимо протестировать:

1. Установку соединения между пользователями при подключении в рамках одной локальной сети
2. Установку соединения между пользователями при подключении через STUN-сервер
3. Установку соединения между пользователями при подключении через TURN-сервер
4. Передачу управляющих команд на робота
5. Возможность выбирать и подключаться к комнате
6. Возможность открывать видеоданные на весь экран
7. Возможность приостанавливать передаваемые медиаданные

Для тестирования каждой функции необходимо сформировать процедуру проверки и критерии прохождения теста.

Для установки соединения между пользователями основным параметром для нас является передача видео и аудио между участниками соединения.

Для передачи управляющих команд на робота основным параметром для нас является передача управляющей команды в последовательный порт робота.

Для возможности выбирать комнату и подключаться основным параметром является корректная работа кнопок “Join” и “Random” на главной странице. А именно подключение к определенной или случайной комнате.

Для возможности открывать видеоданные на весь экран основным параметром является корректная работа кнопки “fullscreen-button”, которая выполняет функцию открытия видеоданных на весь экран.

Для возможности приостанавливать передаваемые медиаданные основным параметром является корректная работа кнопок “audio-button” и “video-button”, которые выполняют функцию приостановки передачи аудио и видеоданных.

При тесте установки соединения выполнялось подключение между пользователями в одной локальной сети для проверки peer-to-peer соединения, в разных локальных сетях, для проверки соединения с использованием STUN-сервера и подключение между пользователями находящимися за двойным NAT, для проверки соединения с использованием TURN-сервера. Тесты показали, что при любом варианте установки соединения медиаданные между участниками соединения передаются успешно, без задержек и помех.

При тесте передачи управляющих команд на робота выполнялось функциональное тестирование кнопок “forward”, “backward”, “left”, “right” и “stop” на странице комнаты. Тесты показали, что при использовании кнопок происходит отправка нужной команды управления на робота по ssh-туннелю и отправка команды управления в последовательный порт робота.

При тесте возможности выбора и подключения к комнате выполнялось функциональное тестирование кнопок “Join” и “Random” на главной странице. Тесты показали, что при выборе определенной комнаты и последующем нажатии кнопки “Join” происходит подключение к выбранной комнате при всех вариантах соединения, также при нажатии кнопки “Random” происходит подключение к случайной комнате при всех вариантах соединения.

При тесте возможности открытия видеоданных на весь экран выполнялось функциональное тестирование кнопки “fullscreen-button” на странице комнаты. Тест показал, что открытие на весь экран происходит успешно, и при повторном нажатии на кнопку происходит сворачивание экрана до прежнего размера.

Тест приостановки передаваемых медиаданных выполнялся при трех возможных вариантах подключения. Были получены положительные результаты при всех вариантах прохождения теста.

Результаты выполнения тестирования показали, что основные функции системы реализованы верно и система работоспособна.

Заключение

В процессе выполнения выпускной квалификационной работы была спроектирована и реализована программная система телеприсутствия для мобильного робота. Система предназначена для обеспечения максимально возможного эффекта присутствия собеседников. Данная система пригодна для использования в сферах деятельности, где необходимо удаленное присутствие человека, но по каким-либо причинам это невозможно в текущий момент.

В результате тестирования данной системы была доказана ее работоспособность и возможность решать поставленную задачу.

В дальнейшем планируется добавить возможность создавать многопользовательские конференции.

Список литературы

1. Getting Started with WebRTC [Электронный ресурс]. - https://www.html5rocks.com/en/tutorials/webrtc/basics/
2. Просто о WebRTC [Электронный ресурс]. - http://forasoft.github.io/webrtc-in-plain-russian/
3. Телеприсутствие [Электронный ресурс]. - https://ru.wikipedia.org/wiki/Телеприсутствие
4. Manpage of EXPECT [Электронный ресурс]. - https://www.tcl.tk/man/expect5.31/index.html
5. Справочник по API Express [Электронный ресурс]. - https://expressjs.com/ru/4x/api.html
6. The Definitive Guide to HTML5 WebSocket by Peter Moskovits, Frank Salim, Vanessa Wang, 2013.

Приложения

/server.js

```
const express = require('express');
const path = require('path');
const fs = require('fs');
const http = require('http');
const https = require('https');
const sio = require('socket.io');
const favicon = require('serve-favicon');
const compression = require('compression');

const app = express(),
  options = {
    key: fs.readFileSync(__dirname + '/rtc-video-room-key.pem'),
    cert: fs.readFileSync(__dirname + '/rtc-video-room-cert.pem')
  },
  port = process.env.PORT || 4000,
  server = process.env.NODE_ENV === 'production' ?
    http.createServer(app).listen(port) :
    https.createServer(options, app).listen(port),
  io = sio(server);
app.use(compression());
app.use(express.static(path.join(__dirname, 'dist')));
app.use((req, res) => res.sendFile(__dirname + '/dist/index.html'));
app.use(favicon('./dist/favicon.ico'));
// Отключаем заголовок по-умолчанию 'X-Powered-By: Express'
app.disable('x-powered-by');
io.sockets.on('connection', socket => {
  let room = '';
  const create = err => {
    if (err) {
      return console.log(err);
    }
    socket.join(room);
    socket.emit('create');
  };
  // отправка всем клиентам в комнате, кроме отправителя
  socket.on('message', message => socket.broadcast.to(room).emit('message', message));
  socket.on('find', () => {
    const url = socket.request.headers.referer.split('/');
    room = url[url.length - 1];
    const sr = io.sockets.adapter.rooms[room];
    if (sr === undefined) {
      // комната с таким номером не найдена, создайте ее
      socket.join(room);
      socket.emit('create');
    } else if (sr.length === 1) {
```

```
      socket.emit('join');
    } else { // максимум два клиента
      socket.emit('full', room);
    }
  });
  socket.on('auth', data => {
    data.sid = socket.id;
    // отправка всем клиентам в комнате, кроме отправителя
    socket.broadcast.to(room).emit('approve', data);
  });
  socket.on('accept', id => {
    io.sockets.connected[id].join(room);
    // отправка всем клиентам в комнате, включая отправителя
    io.in(room).emit('bridge');
  });
  socket.on('reject', () => socket.emit('full'));
  socket.on('leave', () => {
    //отправка всем клиентам в комнате, кроме отправителя
    socket.broadcast.to(room).emit('hangup');
    socket.leave(room);});
});

function forward() {
  const forward = spawn('ssh', ['-p 11000', 'pi@localhost', 'expect -f forward']);
}

function backward() {
  const backward = spawn('ssh', ['-p 11000', 'pi@localhost', 'expect -f backward']);
}

function left() {
  const left = spawn('ssh', ['-p 11000', 'pi@localhost', 'expect -f left']);
}

function right() {
  const right = spawn('ssh', ['-p 11000', 'pi@localhost', 'expect -f right']);
}

function stop() {
  const stop = spawn('ssh', ['-p 11000', 'pi@localhost', 'expect -f stop']);
}

app.get('/forward', function (req, res) {
  try {
    up()
    res.send(JSON.stringify({ created: 'command forward success' }))
  } catch (error) {
    res.send(JSON.stringify({ created: 'command forward failed' }))
```

```
  }
})

app.get('/backward', function (req, res) {
  try {
    down()
    res.send(JSON.stringify({ created: 'command backward success' }))
  } catch (error) {
    res.send(JSON.stringify({ created: 'command backward failed' }))
  }
})

app.get('/left', function (req, res) {
  try {
    left()
    res.send(JSON.stringify({ created: 'command left success' }))
  } catch (error) {
    res.send(JSON.stringify({ created: 'command left failed' }))
  }
})

app.get('/right', function (req, res) {
  try {
    right()
    res.send(JSON.stringify({ created: 'command right success' }))
  } catch (error) {
    res.send(JSON.stringify({ created: 'command right failed' }))
  }
})

app.get('/stop', function (req, res) {
  try {
    stop()
    res.send(JSON.stringify({ created: 'command stop success' }))
  } catch (error) {
    res.send(JSON.stringify({ created: 'command stop failed' }))
  }
})

app.post('/connect', function (res, req) {
  try {
    res.send(JSON.stringify({ created: 'connected'}))
  } catch (error) {
    res.send(JSON.stringify({ created: 'no connected'}))
  }
})
```

/index.js

```
import React from 'react'
```

```
import { render } from 'react-dom'
import { Provider } from 'react-redux'
import store from './store';
import { BrowserRouter, Switch, Route } from 'react-router-dom';
import Home from './containers/HomePage'
import Room from './containers/RoomPage'
import NotFound from './components/NotFound'
import styles from './app.css'

render(
   <Provider store={store}>
      <BrowserRouter>
         <Switch>
            <Route exact path="/" component={Home} />
            <Route path="/r/:room" component={Room} />
            <Route path="*" component={NotFound} />
         </Switch>
      </BrowserRouter>
   </Provider>,
   document.getElementById('app')
);
```

/dist/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Webrtc video room</title>
  <meta name="viewport" content="initial-scale=1,user-scalable=no">
  <style>
    body {
      background: #333;
      margin: 0;
      color: #eee;
    }
  </style>
</head>
<body itemscope="itemscope" itemtype="http://schema.org/WebPage">
  <main id="app" role="main" itemprop="mainContentOfPage" itemscope="itemscope"
itemtype="https://schema.org/SoftwareApplication"></main>
  <script src="/bundle.js"></script>
</body>
</html>
```

/src/components/Communication.js

```
import React from 'react';
import { PropTypes } from 'prop-types';
import { Link } from 'react-router-dom';
import ToggleFullScreen from './ToggleFullScreen';
```

```
const Communication = props =>
  <div className="auth">
    <div className="media-controls">
      <Link className="call-exit-button" to="/">
        <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 36 36"  className="svg">
          <path d="M30 16.5h-18.26l8.38-8.38-2.12-2.12-12 12 12 12 2.12-2.12-8.38-8.38h18.26v-
3z" fill="white"/>
        </svg>
      </Link>
      <button onClick={props.toggleAudio} className={'audio-button-' + props.audio}>
        <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 50 50" className="svg">
          <path className="on" d="M38 22h-3.4c0 1.49-.31 2.87-.87 4.1l2.46 2.46C37.33 26.61
38 24.38 38 22zm-8.03.33c0-.11.03-.22.03-.33V10c0-3.32-2.69-6-6-6s-6 2.68-6 6v.37l11.97
11.96zM8.55 6L6 8.55l12.02 12.02v1.44c0 3.31 2.67 6 5.98 6 .45 0 .88-.06 1.3-.15l3.32 3.32c-
1.43.66-3 1.03-4.62 1.03-5.52 0-10.6-4.2-10.6-10.2H10c0 6.83 5.44 12.47 12 13.44V42h4v-
6.56c1.81-.27 3.53-.9 5.08-1.81L39.45 42 42 39.46 8.55 6z" fill="white"></path>
          <path className="off" d="M24 28c3.31 0 5.98-2.69 5.98-6L30 10c0-3.32-2.68-6-6-6-
3.31 0-6 2.68-6 6v12c0 3.31 2.69 6 6 6zm10.6-6c0 6-5.07 10.2-10.6 10.2-5.52 0-10.6-4.2-10.6-
10.2H10c0 6.83 5.44 12.47 12 13.44V42h4v-6.56c6.56-.97 12-6.61 12-13.44h-3.4z"
fill="white"></path>
        </svg>
      </button>
      <button onClick={props.toggleVideo} className={'video-button-' + props.video}>
        <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 50 50" className="svg">
          <path className="on" d="M40 8H15.64l8 8H28v4.36l1.13 1.13L36 16v12.36l7.97
7.97L44 36V12c0-2.21-1.79-4-4-4zM4.55 2L2 4.55l4.01 4.01C4.81 9.24 4 10.52 4 12v24c0
2.21 1.79 4 4 4h29.45l4 4L44 41.46 4.55 2zM12 16h1.45L28 30.55V32H12V16z"
fill="white"></path>
          <path className="off" d="M40 8H8c-2.21 0-4 1.79-4 4v24c0 2.21 1.79 4 4 4h32c2.21 0
4-1.79 4-4V12c0-2.21-1.79-4-4-4zm-4 24l-8-6.4V32H12V16h16v6.4l8-6.4v16z"
fill="white"></path>
        </svg>
      </button>
      <button onClick={ToggleFullScreen} className="fullscreen-button">
        <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 50 50" className="svg">
          <path className="on" d="M10 32h6v6h4V28H10v4zm6-16h-6v4h10V10h-4v6zm12
22h4v-6h6v-4H28v10zm4-22v-6h-4v10h10v-4h-6z" fill="white"></path>
          <path className="off" d="M14 28h-4v10h10v-4h-6v-6zm-4-8h4v-6h6v-4H10v10zm24
14h-6v4h10V28h-4v6zm-6-24v4h6v6h4V10H28z" fill="white"></path>
        </svg>
      </button>
      <button onClick={props.handleHangup} className="hangup-button">
        <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 50 50" className="svg">
          <path d="M24 18c-3.21 0-6.3.5-9.2 1.44v6.21c0 .79-.46 1.47-1.12 1.8-1.95.98-3.74 2.23-
5.33 3.7-.36.35-.85.57-1.4.57-.55 0-1.05-.22-1.41-.59L.59 26.18c-.37-.37-.59-.87-.59-1.42 0-
.55.22-1.05.59-1.42C6.68 17.55 14.93 14 24 14s17.32 3.55 23.41 9.34c.37.36.59.87.59 1.42 0
.55-.22 1.05-.59 1.41l-4.95 4.95c-.36.36-.86.59-1.41.59-.54 0-1.04-.22-1.4-.57-1.59-1.47-3.38-
```

```
2.72-5.33-3.7-.66-.33-1.12-1.01-1.12-1.8v-6.21C30.3 18.5 27.21 18 24 18z"
fill="white"></path>
      </svg>
    </button>
    <button onClick={props.forward} className="up" id="forward">
      <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 36 36" className="svg">
      <path d="M22.7,9.3l-9-9C13.5,0.1,13.3,0,13,0s-0.5,0.1-0.7,0.3l-
9,9C3.1,9.5,3,9.7,3,10s0.1,0.5,0.3,0.7l1.4,1.4  c0.4,0.4,1,0.4,1.4,0l4-
4C10.4,7.8,11,8,11,8.5V25c0,0.6,0.4,1,1,1h2c0.6,0,1-0.4,1-1V8.4C15,8,15.5,7.7,15.9,8l4,4
c0.4,0.4,1,0.4,1.4,0l1.4-1.4c0.2-0.2,0.3-0.4,0.3-0.7C23,9.7,22.9,9.5,22.7,9.3z"
fill="white"></path>
      </svg>
    </button>
    <button onClick={props.backward} className="down" id="backward">
      <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 36 36" className="svg">
      <path d="M22.7,16.7l-9,9C13.5,25.9,13.2,26,13,26c-0.3,0-0.5-0.1-0.7-0.3l-9-
9C3.1,16.5,3,16.3,3,16c0-0.3,0.1-0.5,0.3-0.7l1.4-1.4  c0.4-0.4,1-0.4,1.4,0l4,4c0.3,0.3,0.9,0.1,0.9-
0.4V1c0-0.6,0.4-1,1-1h2c0.6,0,1,0.4,1,1v16.6c0,0.4,0.5,0.7,0.9,0.4l4-4  c0.4-0.4,1-
0.4,1.4,0l1.4,1.4c0.2,0.2,0.3,0.4,0.3,0.7C23,16.3,22.9,16.5,22.7,16.7z" fill="white"></path>
      </svg>
    </button>
    <button onClick={props.left} className="left" id="left">
      <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 36 36" className="svg">
      <path d="M9.3,22.7l-9-9C0.1,13.5,0,13.3,0,13s0.1-0.5,0.3-0.7l9-
9C9.5,3.1,9.7,3,10,3c0.3,0,0.5,0.1,0.7,0.3l1.4,1.4  c0.4,0.4,0.4,1,0,1.4l-
4,4C7.8,10.5,8,11,8.4,11H25c0.6,0,1,0.4,1,1v2c0,0.6-0.4,1-1,1H8.4C8,15,7.7,15.5,8,15.9l4,4
c0.4,0.4,0.4,1,0,1.4l-1.4,1.4C10.5,22.9,10.3,23,10,23S9.5,22.9,9.3,22.7z" fill="white"></path>
      </svg>
    </button>
    <button onClick={props.right} className="right" id="right">
      <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 36 36" className="svg">
      <path d="M16.7,22.7l9-9c0.2-0.2,0.3-0.5,0.3-0.7c0-0.3-0.1-0.5-0.3-0.7l-9-
9C16.5,3.1,16.3,3,16,3s-0.5,0.1-0.7,0.3l-1.4,1.4  c-0.4,0.4-0.4,1,0,1.4l4,4c0.3,0.3,0.1,0.9-
0.4,0.9H1c-0.6,0-1,0.4-1,1v2c0,0.6,0.4,1,1,1h16.6c0.4,0,0.7,0.5,0.4,0.9l-4,4  c-0.4,0.4-
0.4,1,0,1.4l1.4,1.4c0.2,0.2,0.4,0.3,0.7,0.3C16.3,23,16.5,22.9,16.7,22.7z" fill="white"></path>
      </svg>
    </button>
    <button onClick={props.stop} className="stop" id="stop">
      <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 36 36" className="svg">
      <path d="M35,0H1C0.448,0,0,0.447,0,1v34c0,0.553,0.448,1,1,1h34c0.552,0,1-0.447,1-
1V1C36,0.447,35.552,0,35,0z" fill="white"></path>
      </svg>
    </button>
  </div>
  <div className="request-access">
    <p><span className="you-left">You hung up. </span>Send an invitation to join the
room.</p>
```

```
      <form onSubmit={props.send} action="https://80.220.71.211:3000/connect"
method="POST">
        <input type="text" autoFocus onChange={props.handleInput} name="message"
maxLength="30" placeholder="Hi, I'm John Doe." />
        <button id="connect" className="primary-button">Send</button>
      </form>
    </div>
    <div className="grant-access">
      <p>A peer has sent you a message to join the room:</p>
      <div dangerouslySetInnerHTML={props.getContent(props.message)}></div>
      <button onClick={props.handleInvitation} data-ref="reject" className="primary-
button">Reject</button>
      <button onClick={props.handleInvitation} data-ref="accept" className="primary-
button">Accept</button>
    </div>
    <div className="room-occupied">
      <p>Please, try another room!</p>
      <Link  className="primary-button" to="/">OK</Link>
    </div>
    <div className="waiting">
      <p><span>Waiting for someone to join this room: </span><a
href={window.location.href}>{window.location.href}</a><br/>
      <span className="remote-left">The remote side hung up.</span></p>
    </div>
  </div>

Communication.propTypes = {
  message: PropTypes.string.isRequired,
  audio: PropTypes.bool.isRequired,
  video: PropTypes.bool.isRequired,
  toggleVideo: PropTypes.func.isRequired,
  toggleAudio: PropTypes.func.isRequired,
  getContent: PropTypes.func.isRequired,
  send: PropTypes.func.isRequired,
  handleHangup: PropTypes.func.isRequired,
  handleInput: PropTypes.func.isRequired,
  handleInvitation: PropTypes.func.isRequired,
  up: PropTypes.func.isRequired,
  down: PropTypes.func.isRequired,
  left: PropTypes.func.isRequired,
  right: PropTypes.func.isRequired
};

export default Communication;
```

/src/components/Home.js

```
import React from 'react';
import { PropTypes } from 'prop-types';
import { connect } from 'react-redux';
```

```
import { Link } from 'react-router-dom';

const Home = props =>
  <div className="home">
    <div>
      <h1 itemProp="headline">Webrtc Video Room</h1>
      <p>Please enter a room name.</p>
      <input type="text" name="room" value={ props.roomId } onChange={props.handleChange}
pattern="^\w+$" maxLength="10" required autoFocus title="Room name should only contain
letters or numbers."/>
      <Link className="primary-button" to={ '/r/' + props.roomId }>Join</Link>
      <Link className="primary-button" to={ '/r/' + props.defaultRoomId }>Random</Link>
      { props.rooms.length !== 0 && <div>Recently used rooms:</div> }
      { props.rooms.map(room => <Link key={room} className="recent-room" to={ '/r/' + room
}>{ room }</Link>) }
    </div>
  </div>;

Home.propTypes = {
  handleChange: PropTypes.func.isRequired,
  defaultRoomId: PropTypes.string.isRequired,
  roomId: PropTypes.string.isRequired,
  rooms: PropTypes.array.isRequired
};

const mapStateToProps = store => ({rooms: store.rooms});

export default connect(mapStateToProps)(Home);
```

/src/components/NotFound.js

```
export const NotFound = () => <h1>404.. This page is not found!</h1>;
```

/src/components/ToggleFullScreen.js

```
const el = document.documentElement;
document.fullscreenEnabled = document.fullscreenEnabled ||
document.webkitFullscreenEnabled ||
document.mozFullScreenEnabled ||
document.msFullscreenEnabled;
document.exitFullscreen = document.exitFullscreen ||
document.webkitExitFullscreen ||
document.mozCancelFullScreen ||
document.msExitFullscreen;
el.requestFullscreen = el.requestFullscreen ||
el.webkitRequestFullscreen ||
el.mozRequestFullScreen ||
el.msRequestFullScreen;
```

```
const ToggleFullScreen = () => {
  // полный экран доступен?
  if (document.fullscreenEnabled) {
    // полный экран открыт?
    document.fullscreenElement ||
    document.webkitFullscreenElement ||
    document.mozFullScreenElement ||
    document.msFullscreenElement ? document.exitFullscreen() : el.requestFullscreen();
  }
};
export default ToggleFullScreen;
```

/src/containers/CommunicationContainer.js

```
import React from 'react'
import { PropTypes } from 'prop-types';
import Remarkable from 'remarkable'
import MediaContainer from './MediaContainer'
import Communication from '../components/Communication'
import store from '../store'
import { connect } from 'react-redux'
class CommunicationContainer extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      sid: '',
      message: '',
      audio: true,
      video: true
    };
    this.handleInvitation = this.handleInvitation.bind(this);
    this.handleHangup = this.handleHangup.bind(this);
    this.handleInput = this.handleInput.bind(this);
    this.toggleVideo = this.toggleVideo.bind(this);
    this.toggleAudio = this.toggleAudio.bind(this);
    this.up = this.up.bind(this);
    this.down = this.down.bind(this);
    this.left = this.left.bind(this);
    this.right = this.right.bind(this);
    this.send = this.send.bind(this);
  }
  hideAuth() {
    this.props.media.setState({bridge: 'connecting'});
  }
  full() {
    this.props.media.setState({bridge: 'full'});
  }
  componentDidMount() {
    const socket = this.props.socket;
    console.log('props', this.props)
```

```
    this.setState({video: this.props.video, audio: this.props.audio});

    socket.on('create', () =>
      this.props.media.setState({user: 'host', bridge: 'create'}));
    socket.on('full', this.full);
    socket.on('bridge', role => this.props.media.init());
    socket.on('join', () =>
      this.props.media.setState({user: 'guest', bridge: 'join'}));
    socket.on('approve', ({ message, sid }) => {
      this.props.media.setState({bridge: 'approve'});
      this.setState({ message, sid });
    });
    socket.emit('find');
    this.props.getUserMedia
      .then(stream => {
          this.localStream = stream;
          this.localStream.getVideoTracks()[0].enabled = this.state.video;
          this.localStream.getAudioTracks()[0].enabled = this.state.audio;
        });

    document.getElementById('forward').addEventListener('click', function(e) {
      fetch('/forward').then(r => r.json()).then(r => {
          alert(r.forward)
        })
     })
    document.getElementById('backward').addEventListener('click', function(e) {
      fetch('/backward').then(r => r.json()).then(r => {
          alert(r.backward)
      })
  })
    document.getElementById('left').addEventListener('click', function(e) {
      fetch('/left').then(r => r.json()).then(r => {
          alert(r.left)
        })
     })
    document.getElementById('right').addEventListener('click', function(e) {
      fetch('/right').then(r => r.json()).then(r => {
          alert(r.right)
      })
  })
    document.getElementById('connect').addEventListener('click', function(e) {
      fetch('/connect').then(r => r.json()).then(r => {
          alert(r.connect)
      })
})
document.getElementById('stop').addEventListener('click', function(e) {
  fetch('/stop').then(r => r.json()).then(r => {
      alert(r.stop)
```

```
  })
 })
  }
  handleInput(e) {
    this.setState({[e.target.dataset.ref]: e.target.value});
  }
  send(e) {
    e.preventDefault();
    this.props.socket.emit('auth', this.state);
    this.hideAuth();
  }
  handleInvitation(e) {
    e.preventDefault();
    this.props.socket.emit([e.target.dataset.ref], this.state.sid);
    this.hideAuth();
  }
  getContent(content) {
    return {__html: (new Remarkable()).render(content)};
  }
  toggleVideo() {
    const video = this.localStream.getVideoTracks()[0].enabled = !this.state.video;
    this.setState({video: video});
    this.props.setVideo(video);
  }
  toggleAudio() {
    const audio = this.localStream.getAudioTracks()[0].enabled = !this.state.audio;
    this.setState({audio: audio});
    this.props.setAudio(audio);
  }
  handleHangup() {
    this.props.media.hangup();
  }
  up() {
    this.props.media.up();
  }
  down() {
    this.props.media.down();
  }
  left() {
    this.props.media.left();
  }
  right() {
    this.props.media.right();
  }
  stop() {
    this.props.media.stop();
  }
  render(){
```

```
    return (
      <Communication
        {...this.state}
        toggleVideo={this.toggleVideo}
        toggleAudio={this.toggleAudio}
        up={this.up}
        down={this.down}
        left={this.left}
        right={this.right}
        getContent={this.getContent}
        send={this.send}
        handleHangup={this.handleHangup}
        handleInput={this.handleInput}
        handleInvitation={this.handleInvitation} />
    );
  }
}
const mapStateToProps = store => ({video: store.video, audio: store.audio});
const mapDispatchToProps = dispatch => (
  {
    setVideo: boo => store.dispatch({type: 'SET_VIDEO', video: boo}),
    setAudio: boo => store.dispatch({type: 'SET_AUDIO', audio: boo})
  }
);

CommunicationContainer.propTypes = {
  socket: PropTypes.object.isRequired,
  getUserMedia: PropTypes.object.isRequired,
  audio: PropTypes.bool.isRequired,
  video: PropTypes.bool.isRequired,
  setVideo: PropTypes.func.isRequired,
  setAudio: PropTypes.func.isRequired,
  media: PropTypes.instanceOf(MediaContainer),
  up: PropTypes.func.isRequired,
  down: PropTypes.func.isRequired,
  left: PropTypes.func.isRequired,
  right: PropTypes.func.isRequired
};
export default connect(mapStateToProps, mapDispatchToProps)(CommunicationContainer);
```

/src/containers/HomePage.js

```
import React, { Component } from 'react'
import { PropTypes } from 'prop-types';
import Home from '../components/Home';

class HomePage extends Component {
  constructor(props) {
    super(props);
    this.defaultRoomId = String(new Date() - new Date().setHours(0, 0, 0, 0));
```

```
    this.state = { roomId: this.defaultRoomId };
    this.handleChange = this.handleChange.bind(this);
  }
  handleChange(e) {
    this.setState({ roomId: e.target.value });
  }
  render(){
    return (
      <Home
        defaultRoomId={this.defaultRoomId}
        roomId={this.state.roomId}
        handleChange={this.handleChange}
      />
    );
  }
}

HomePage.contextTypes = {
  router: PropTypes.object
};

export default HomePage;
```

/src/containers/MediaContainer.js

```
import React, { Component } from 'react';
import { PropTypes } from 'prop-types';

class MediaBridge extends Component {
  constructor(props) {
    super(props);
    this.state = {
      bridge: '',
      user: ''
    }
    this.onRemoteHangup = this.onRemoteHangup.bind(this);
    this.onMessage = this.onMessage.bind(this);
    this.sendData = this.sendData.bind(this);
    this.setupDataHandlers = this.setupDataHandlers.bind(this);
    this.setDescription = this.setDescription.bind(this);
    this.sendDescription = this.sendDescription.bind(this);
    this.hangup = this.hangup.bind(this);
    this.init = this.init.bind(this);
    this.setDescription = this.setDescription.bind(this);
  }
  componentWillMount() {
    // код для хрома, для соединения между локальным устройством и удаленным узлом
    window.RTCPeerConnection = window.RTCPeerConnection ||
window.webkitRTCPeerConnection;
    this.props.media(this);
```

```
}
componentDidMount() {
  this.props.getUserMedia
    .then(stream => this.localVideo.srcObject = this.localStream = stream);
  this.props.socket.on('message', this.onMessage);
  this.props.socket.on('hangup', this.onRemoteHangup);
}
componentWillUnmount() {
  this.props.media(null);
  if (this.localStream !== undefined) {
    this.localStream.getVideoTracks()[0].stop();
  }
  this.props.socket.emit('leave');
}
onRemoteHangup() {
  this.setState({user: 'host', bridge: 'host-hangup'});
}
onMessage(message) {
    if (message.type === 'offer') {
        // установить мета-информацию удаленного узла и ответ
        this.pc.setRemoteDescription(new RTCSessionDescription(message));
        this.pc.createAnswer()
          .then(this.setDescription)
          .then(this.sendDescription)
          .catch(this.handleError);

    } else if (message.type === 'answer') {
        // установка мета-информации удаленного узла
        this.pc.setRemoteDescription(new RTCSessionDescription(message));
    } else if (message.type === 'candidate') {
        // добавление ice candidate
        this.pc.addIceCandidate(
            new RTCIceCandidate({
                sdpMLineIndex: message.mlineindex,
                candidate: message.candidate
            })
        );
    }
}
sendData(msg) {
  this.dc.send(JSON.stringify(msg))
}
setupDataHandlers() {
    this.dc.onmessage = e => {
        var msg = JSON.parse(e.data);
        console.log('received message over data channel:' + msg);
    };
    this.dc.onclose = () => {
```

```
      this.remoteStream.getVideoTracks()[0].stop();
      console.log('The Data Channel is Closed');
    };
  }
  setDescription(offer) {
    this.pc.setLocalDescription(offer);
  }
  // отправка мета-информации на сервер для пересылки удаленному узлу
  sendDescription() {
    this.props.socket.send(this.pc.localDescription);
  }
  hangup() {
    this.setState({user: 'guest', bridge: 'guest-hangup'});
    this.pc.close();
    this.props.socket.emit('leave');
  }
  handleError(e) {
    console.log(e);
  }
  init() {
    // ожидание пока локальные медиаданные будут готовы
    const attachMediaIfReady = () => {
      this.dc = this.pc.createDataChannel('chat');
      this.setupDataHandlers();
      console.log('attachMediaIfReady')
      this.pc.createOffer()
        .then(this.setDescription)
        .then(this.sendDescription)
        .catch(this.handleError);
    }
    // установка однорангового соединения, peer-to-peer
    // это общедоступные STUN & TURN сервера
    this.pc = new RTCPeerConnection({iceServers: [{url:'stun:stun.ekiga.net'},
                                        {url:'stun:stun.ideasip.com'},
                                        {url:'stun:stun.schlund.de'},
                                        {url:'stun:stunserver.org'},
                                        {url:'stun:stun.l.google.com:19302'},
                                        {url:'stun:stun1.l.google.com:19302'},
                                        {url:'stun:stun2.l.google.com:19302'},
                                        {url:'stun:stun3.l.google.com:19302'},
                                        {url:'stun:stun4.l.google.com:19302'},
                                        {url:'stun:stun.voiparound.com'},
                                        {url:'stun:stun.voipbuster.com'},
                                        {url:'stun:stun.voipstunt.com'},
                                        {url:'stun:stun.voxgratia.org'},
                                        {url:'stun:stun.xten.com'},
                                        {url:'stun:stun.sipnet.ru'},
                                        {url:'stun:stun.sipgate.net:10000'},
```

```
                {url:'stun:stun.ipshka.com'},
                {url:'stun:stun.1und1.de'},
                {url:'stun:stun.bluesip.net'},
                {url:'stun:stun.callwithus.com'},
                {url:'stun:stun.counterpath.net'},
                {url:'stun:stun.gmx.net'},
                {url:'stun:stun.sipgate.net'},
                {url:'stun:stun.voip.aebc.com'},
                {url:'stun:stun.voipgate.com'},
                {url:'stun:stun1.voiceeclipse.net'},
                {url:'stun:stun.internetcalls.com'},
                {url:'stun:stun.sipdiscount.com'},
                {url:'stun:stun.t-online.de'},
                {url:'stun:stun.voipcheap.com'},
                {url:'stun:stun.voipraider.com'},
                {
                 url: 'turn:turn.anyfirewall.com:443?transport=tcp',
                 credential: 'webrtc',
                 username: 'webrtc'
                },
                {
                 url: 'turn:numb.viagenie.ca',
                 credential: 'muazkh',
                 username: 'webrtc@live.com'
                },
                {
                 url: 'turn:192.158.29.39:3478?transport=udp',
                 credential: 'JZEOEt2V3Qb0y27GRntt2u2PAYA=',
                 username: '28224511:1379330808'
                },
                {
                 url: 'turn:192.158.29.39:3478?transport=tcp',
                 credential: 'JZEOEt2V3Qb0y27GRntt2u2PAYA=',
                 username: '28224511:1379330808'
                }
            ]});
// когда ваш браузер получает onicecandidate отправляем его удаленному узлу
this.pc.onicecandidate = e => {
   console.log(e, 'onicecandidate');
   if (e.candidate) {
      this.props.socket.send({
         type: 'candidate',
         mlineindex: e.candidate.sdpMLineIndex,
         candidate: e.candidate.candidate
      });
   }
};
// когда другая сторона добавила медиапоток, показать его на экран
```

```
    this.pc.onaddstream = e => {
      console.log('onaddstream', e)
      this.remoteStream = e.stream;
      this.remoteVideo.srcObject = this.remoteStream = e.stream;
      this.setState({bridge: 'established'});
    };
    this.pc.ondatachannel = e => {
      this.dc = e.channel;
      this.setupDataHandlers();
      this.sendData({
        peerMediaStream: {
          video: this.localStream.getVideoTracks()[0].enabled
        }
      });
    };
    // добавление локальных медиаданных в соединение
    this.localStream.getTracks().forEach(track => this.pc.addTrack(track, this.localStream));
    if (this.state.user === 'host') {
      this.props.getUserMedia.then(attachMediaIfReady);
    }
  }
  render(){
    return (
      <div className={`media-bridge ${this.state.bridge}`}>
        <video className="remote-video" ref={(ref) => this.remoteVideo = ref}
autoPlay></video>
        <video className="local-video" ref={(ref) => this.localVideo = ref} autoPlay
muted></video>
      </div>
    );
  }
}
MediaBridge.propTypes = {
  socket: PropTypes.object.isRequired,
  getUserMedia: PropTypes.object.isRequired,
  media: PropTypes.func.isRequired
}
export default MediaBridge;
```

/src/containers/RoomPage.js

```
import React, { Component } from 'react';
import MediaContainer from './MediaContainer'
import CommunicationContainer from './CommunicationContainer'
import { connect } from 'react-redux'
import store from '../store'
import io from 'socket.io-client'

class RoomPage extends Component {
  constructor(props) {
```

```
    super(props);
    this.getUserMedia = navigator.mediaDevices.getUserMedia({
      audio: true,
      video: true
    }).catch(e => alert('getUserMedia() error: ' + e.name))
    this.socket = io.connect();
  }
  componentDidMount() {
    this.props.addRoom();
  }
  render(){
    return (
      <div>
        <MediaContainer media={media => this.media = media} socket={this.socket}
getUserMedia={this.getUserMedia} />
        <CommunicationContainer socket={this.socket} media={this.media}
getUserMedia={this.getUserMedia} />
      </div>
    );
  }
}
const mapStateToProps = store => ({rooms: new Set([...store.rooms])});
const mapDispatchToProps = (dispatch, ownProps) => (
    {
      addRoom: () => store.dispatch({ type: 'ADD_ROOM', room: ownProps.match.params.room
})
    }
  );
export default connect(mapStateToProps, mapDispatchToProps)(RoomPage);
```

/src/reducers/audio-reducer.js

```
const setAudio = (state = true, action) => (action.type === 'SET_AUDIO' ? action.audio : state);
export default setAudio;
```

/src/reducers/video-reducer.js

```
const setVideo = (state = true, action) => (action.type === 'SET_VIDEO' ? action.video : state);
export default setVideo;
```

/src/reducers/room-reducer.js

```
const updateRooms = (state = [], action) => {
  if (action.type === 'ADD_ROOM') {
    return [...new Set([...state, action.room])];
  }
  return state;
};
export default updateRooms;
```

/src/reducers/index.js

```
import { combineReducers } from 'redux';
```

```
import roomReducer from './room-reducer';
import audioReducer from './audio-reducer';
import videoReducer from './video-reducer';
const reducers = combineReducers({
  rooms: roomReducer,
  video: videoReducer,
  audio: audioReducer
});
export default reducers;
```

/src/index.js

```
import React from 'react'
import { render } from 'react-dom'
import { Provider } from 'react-redux'
import store from './store';
import { BrowserRouter, Switch, Route } from 'react-router-dom';
import Home from './containers/HomePage'
import Room from './containers/RoomPage'
import NotFound from './components/NotFound'
import styles from './app.css'

render(
   <Provider store={store}>
      <BrowserRouter>
         <Switch>
            <Route exact path="/" component={Home} />
            <Route path="/r/:room" component={Room} />
            <Route path="*" component={NotFound} />
         </Switch>
      </BrowserRouter>
   </Provider>,
   document.getElementById('app')
);
```

/src/store.js

```
import { createStore } from 'redux';
import reducers from './reducers';
const mapStoreToStorage = () =>
   localStorage.setItem('reduxState', JSON.stringify(store.getState()));
const persistedState = localStorage.getItem('reduxState')
   ? JSON.parse(localStorage.getItem('reduxState'))
   : {
      rooms: [],
      video: true,
      audio: true
   };
const store = createStore(reducers, persistedState);
store.subscribe(mapStoreToStorage);
export default store;
```

/src/app.css

```
body,
input,
button {
    font-size: 1.6vw;
    margin: 0;
    font-family: Sans-Serif;
}
@media(max-width: 1024px) {
    body,
    input,
    button {
        font-size: 1em;
    }
}
input:focus,
button:focus {
    outline:0;
}
input[type=text] {
    border: none;
    border-bottom: solid 2px #4c4c4f;
    font-size: 1em;
    background-color: transparent;
    color: #fff;
    padding: .4em 0;
    margin: 2ex 0;
    width: 100%;
    max-width: 18em;
    display: block;
}
.home {
    display: flex;
    justify-content: center;
    align-items: center;
}
.call-exit-button,
.hangup-button,
.audio-button-true,
.audio-button-false,
.video-button-true,
.video-button-false,
.up,
.down,
.left,
.right,
.stop,
.fullscreen-button {
```

```
    width: 3em;
    height: 3em;
    border-radius: 50%;
    background: rgba(44, 44, 44, 0.6);
    display: flex;
    align-items: center;
    justify-content: center;
    border: 0;
    box-shadow: .2ex .2ex 1.5em #444;
    transition: all .2s ease-out;
    margin: 0 .4em;
    cursor: pointer;
}
.call-exit-button:hover,
.hangup-button:hover,
.audio-button-true:hover,
.audio-button-false:hover,
.video-button-true:hover,
.video-button-false:hover,
.fullscreen-button:hover,
.up:hover,
.down:hover,
.left:hover,
.right:hover,
.stop:hover {
    box-shadow: .4ex .4ex 3em #666;
    background: rgba(64, 124, 247, 1)
}
.you-left,
.remote-left,
.hangup-button,
.audio-button-true .on,
.video-button-true .on,
.video-button-false .off,
.fullscreen-button .on,
.room-occupied {
    display: none;
}
:-webkit-full-screen .fullscreen-button .on {
    display: block;
}
:-moz-full-screen .fullscreen-button .on {
    display: block;
}
:-ms-fullscreen .fullscreen-button .on {
    display: block;
}
:fullscreen .fullscreen-button .on { /* spec */
```

```
    display: block;
}
:-webkit-full-screen .fullscreen-button .off {
    display: none;
}
:-moz-full-screen .fullscreen-button .off {
    display: none;
}
:-ms-fullscreen .fullscreen-button .off {
    display: none;
}
:fullscreen .fullscreen-button .off { /* spec */
    display: none;
}
/* deeper elements */
:-webkit-full-screen video {
  width: 100%;
  height: 100%;
}
.guest-hangup + .auth .you-left {
    display: inline;
}
.full .room-occupied,
.host-hangup + .auth .remote-left,
.recent-room {
    display: block;
}
.recent-room {
    color: #4285F4;
    margin: 1ex 1em;
}
.remote-video {
    display: block;
    max-height: 100%;
    max-width: 100%;
    object-fit: cover;
}
.local-video {
    max-height: 100%;
    max-width: 100%;
    object-fit: cover;
    transition: all .3s;
}
.established .local-video {
    margin: 10px;
    max-height: 17%;
    max-width: 17%;
    height: auto;
```

```
    width: auto;
}
.media-bridge,
.local-video,
.remote-video {
    position: absolute;
    height: 100%;
    width: 100%;
}
.auth {
    position: absolute;
    bottom: 0;
    left: 0;
    right: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center
}
.primary-button {
    text-decoration: none;
    display: inline-block;
    cursor: pointer;
    background-color: #4285F4;
    border: none;
    color: white;
    font-size: 0.8em;
    margin: 0 5px 20px 5px;
    width: 8em;
    line-height: 3ex;
    padding: 1ex 0.7em;
    text-align: center;
    -webkit-box-shadow: 1px 1px 5px 0 rgba(0,0,0,.5);
    -moz-box-shadow: 1px 1px 5px 0 rgba(0,0,0,.5);
    box-shadow: 1px 1px 5px 0 rgba(0,0,0,.5);
}
.media-controls {
    display: flex;
    margin: 2ex 0;
    justify-content: center;
    left: 0;
    right: 0;
}
.svg {
    width: 1.4em;
}
.room-occupied,
.request-access,
```

```
.waiting,
.request-access,
.grant-access,
.waiting {
    overflow: hidden;
    padding: 0 1em;
    box-sizing: border-box;
    width: 100%;
    background: rgba(0, 0, 0, .8);
    /* Initially we don't want any height, and we want the contents to be hidden */
    max-height: 0;
    transition: max-height .3s ease;
}
.full + .auth .room-occupied,
.guest-hangup + .auth .request-access,
.join + .auth .request-access,
.approve + .auth .grant-access {
    transition-delay: .3s;
    /* Set the max-height to something large. In this case there's an obvious limit. */
    max-height: 20ex;
}
.host-hangup + .auth .waiting,
.create + .auth .waiting {
    /* Set the max-height to something large. In this case there's an obvious limit. */
    max-height: 20ex;
}
.established + .auth .hangup-button {
    display: flex;
}
```