Министерство образования и науки Российской Федерации

ФГБОУ ВПО «Сыктывкарский государственный университет»

Колледж экономики, права и информатики

ДОПУСТИТЬ К ЗАЩИТЕ:

Зам. директора КЭПИ

\_\_\_\_\_\_\_\_\_\_\_\_\_Т. В. Гилева

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2014 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

Передача потоковых данных на основе WebRTC

Научный руководитель:

\_\_\_\_\_\_\_\_\_\_\_ Л. М. Мартынова

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2014 г.

Исполнитель:

студент 35 группы

\_\_\_\_\_\_\_\_\_\_ К. И. Лопырев

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2014 г.

Сыктывкар 2014

**СОДЕРЖАНИЕ**

СПИСОК СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ..................................... 3

ВВЕДЕНИЕ................................................................................................ 5

1 ВНУТРЕННЯЯ АРХИТЕКТУРА И API WEBRTC............................. 7

1.1 Сравнение с аналогичными технологиями............................. 7

1.1.1 Java................................................................................ 7

1.1.2 Flash............................................................................... 8

1.1.3 WebRTC........................................................................ 8

1.1.4 Итоговое сравнение..................................................... 9

1.2 Внутренняя архитектура WebRTC.......................................... 12

1.3 Использование Web API........................................................... 15

1.3.1 Создание подключения............................................... 15

1.3.2 Получение локального потока.................................... 18

1.3.3 Отправка и получение потоков................................... 19

2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ НА ОСНОВЕ WEB API.............. 21

2.1 Web-приложение, внедряемое на сторонний ресурс............. 21

2.2 WebRTC-сервис......................................................................... 31

ЗАКЛЮЧЕНИЕ.......................................................................................... 38

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.................................. 39

**СПИСОК СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

BSD-3 - лицензионное соглашение на программное обеспечение. По сравнению с другими лицензиями на свободное ПО лицензия BSD налагает меньше ограничений, поэтому использование этой лицензии ближе к помещению программы в категорию общественного достояния. BSD-3 позволяет не только использовать программный код, но и модифицировать его для использования в собственных проектах.

GIPS (Global IP Solutions) - компания, занимавшаяся разработкой программного обеспечения для обработки и передачи видео и аудио в IP-сетях. В 2010 году компанию выкупила корпорация Google, после чего она стала подразделением Google и ныне известна как WebRTC Project.[7]

API (Application Programming Interface) - набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой) для использования во внешних программных продуктах. Часто выполняет роль слоя абстракции, упрощающего доступ к функциям приложения (библиотеки). Используется для написания всевозможных приложений, основанных на готовом программном решении.[20]

JRE (Java Runtime Environment) - среда выполнения программного кода на языке Java.

RTP (Real Time Transport Protocol) - открытый протокол потоковой передачи данных в IP-сетях, работающий на прикладном уровне и использующийся при передаче трафика реального времени. Протокол был разработан Audio-Video Transport Working Group и впервые опубликован в 1996 году.[11]

FCS MX (Flash Communication Server) - серверная среда от Adobe, прародитель Adobe Media Server.

RTMP (Real Time Messaging Protocol) - проприетарный протокол потоковой передачи данных, в основном используется для передачи потокового медиа в Интернет.

VoIP - IP-телефония - набор технологий для передачи видео и аудио сигнала по протоколу IP, имеющий функции обычной телефонной связи, такие как, например, набор номера, дозвон и т.п.[16]

STUN (Session Traversal Utilities for NAT) - утилиты прохождения сессий для NAT - сетевой протокол, который позволяет клиенту, находящемуся за сервером трансляции адресов, определить свой внешний IP-адрес, способ трансляции адреса и порт во внешней сети. Эта информация используется для установления соединения UDP между двумя хостами, находящимися за NAT-маршрутизаторами.[14]

NAT (Network Address Translation) - преобразование сетевых адресов - это механизм в IP-сетях, используемый для преобразования IP-адресов передаваемых по сети пакетов.[8]

TURN (Traversal Using Relay NAT) - протокол, который позволяет узлу за NAT или файерволом получать входящие данные через TCP или UDP соединения.[15]

Libjingle - набор C++ библиотек с открытым исходным кодом для организации подключений реального времени через Интернет.

W3C (World Wide Web Consortium) - Консорциум Всемирной паутины - организация, разрабатывающая и внедряющая технологические стандарты для Всемирной паутины. Консорциум возглавляет Тимоти Джон Бернерс-Ли, автор множества разработок в области информационных технологий.[21]

SDP (Session Description Protocol) - протокол прикладного уровня и формат сообщений для описания сессий потоковой передачи данных.

**ВВЕДЕНИЕ**

WebRTC (Real Time Communications) - коммуникации в реальном времени - стек технологий, включающий набор видео и аудио кодеков и транспортных протоколов для организации подключений между клиентскими устройствами и передачи потоковых данных по технологии “точка-точка”.

Основной идеей проекта WebRTC была идея о том, чтобы сделать технологии подключений реального времени открытыми и простыми для использования. Поэтому сейчас исходный код WebRTC опубликован под лицензией BSD-3.

WebRTC позволяет создавать полнодуплексное соединение между клиентами и не требует наличия сервера для передачи потоков. WebRTC стал первым стеком технологий, дающим возможности организации полноценной видео и аудио связи без использования дополнительных плагинов и приложений, с использованием только HTML5 и JavaScript. Эти аспекты выделили данную технологию из ряда других уже используемых и способствовали быстрому внедрению на рынок. По мнению большинства экспертов, после полноценного внедрения WebRTC в самые популярные браузеры и веб-клиенты браузер Google Chrome может составить конкуренцию Skype.[19]

Одной из серьезных задач web-технологий в последнее время - сделать подключения реального времени (RTC) такими же естественными, как, например, ввод текста на web-странице. До появления WebRTC технологии, дающие возможности потоковой передачи данных были закрытыми и требовали от разработчика наличия лицензии на использование. Это сильно затормаживало развитие этой области web-технологий. Ситуация стала меняться, когда этой проблемой вплотную занялась корпорация Google. В 2008 году компания On2 Technologies разработала видео кодек VP8 как замену предыдущим кодекам VP7 и VP6. В 2010 году компания Google приобрела фирму-создателя и 19 мая 2010 года представляет открытые исходные коды. В 2009 году корпорация GIPS создала аудио кодек iSAC, ядро аудио и видео обработки и протокол WebRTC. GIPS на тот момент являлся лидером в разработке технологий, используемых для RTC. GIPS был куплен Google вместе с патентными правами на их разработки, которые были опубликованы Google в 2011 под лицензией BSD-3 как WebRTC Project. И уже в ноябре 2012 Google объявил о внедрении поддержки WebRTC в свой браузер Google Chrome. В начале 2013 года осуществлён первый видеозвонок между Chrome и Firefox.[12]

На момент написания данной работы технология поддерживается браузерами:

* Google Chrome;
* Google Chrome Mobile;
* Mozilla Firefox;
* Mozilla Firefox Mobile;
* Opera.

Для других браузеров можно использовать расширение webrtc4all. Таким образом, по заявлениям инженеров компании Google, на данный момент существует более миллиарда конечных точек, поддерживающих WebRTC.

В рамках данной дипломной работы будет изучена технология WebRTC и возможность ее практического применения уже сейчас.

**1 ВНУТРЕННЯЯ АРХИТЕКТУРА И API WEBRTC**

**1.1 Сравнение с аналогичными технологиями**

**1.1.1 Java**

Первой технологией, которая имела возможности организации браузерных видеозвонков, была Java. JRE, как правило, присутствует на большинстве ПК либо предустановлена в виде плагина в браузере. В итоге Java-код может выполняться ПК или браузерным плагином, захватывать медиа и отправлять его на сервер по стандартному протоколу RTP.[5]

В подобных приложениях на Java была обеспечена надлежащая безопасность, так как апплет должен быть подписан, и при запуске происходит запрос разрешений у пользователя, желает ли он запустить подписанный апплет от данного производителя, который может получить доступ к тем или иным функциям.

Плюсы:

* нет лишних звеньев, возможность прямого взаимодействия с сервером по RTP;
* широкая распространенность и доступность JRE для конечного пользователя;
* протокол RTP имеет возможность восстановления утерянных в сети пакетов за счет Jitter-буфера.

Минусы:

* слабое подавление эха;
* ручная регулировка усиления;
* слабое подавление шума;
* низкое быстродействие Java по сравнению с C++ (на котором написан WebRTC);
* клиент-серверная архитектура.

**1.1.2 Flash**

Выпущенный в 2002 году Flash Player 6 умел взаимодействовать с FCS MX 1.0, захватывать медиапоток пользователя и обмениваться с сервером аудиопотоками по протоколу RTMP. Эхоподавления в этой технологии не было, но платформа исправно передавала медиапоток от плеера к плееру через сервер и на тот момент могла претендовать на монопольность. Протокол работал поверх TCP, следствием этого было сохранение утерянных пакетов. Большой размер буфера был причиной довольно больших задержек при передаче потока. В результате на базе RTMP развивались, в основном, live video, или вебинары с одним ведущим, где задержка не так важна. Только в последующих версиях была добавлена поддержка UDP и эхоподавления.[5]

Плюсы:

* распространенность (самый широкий охват);
* качественная VoIP передача аудио и видео.

Минусы:

* не поддерживал кодек H.264. Из-за этого во многих приложениях на этой; технологии применяется транскодинг, что в свою очередь вызывает избыточное потребление ресурсов процессора;
* требует промежуточного сервера;
* не поддерживает открытые UDP протоколы, такие как RTP/SRTP;
* медленное развитие;
* клиент-серверная архитектура;
* закрытость исходного кода.

**1.1.3 WebRTC**

WebRTC на стадии релиза спецификации версии 1.0 состоял из стека технологий: SRTP, DTLS, ICE, STUN, AEC, AGC, адаптивный Jitter-буфер, Opus, VP8. SRTP и DTLS обеспечивает защиту трафика между WebRTC узлами. ICE и STUN помогают преодолеть NAT. AEC, AGC и Jitter-буфер работают для того, чтобы сделать аудио и видео качественным. Кодеки Opus и VP8 хорошо подходят для глобального Интернета, где битрейт при передаче может падать до очень низких значений из-за низкого качества связи.[5]

Плюсы:

* полноценное VoIP только средствами браузера;
* архитектура “точка-точка”;
* открытый исходный код под лицензией BSD-3.

Минусы:

* пока недостаточно поддерживающих технологию браузеров, хотя ситуация довольно быстро исправляется;
* отсутствие совместимости с традиционным VoIP.

**1.1.4 Итоговое сравнение**

Возможность браузерных звонков в той или иной форме существовала задолго до появления WebRTC, но осуществлялась за счет внешних, не встроенных в сам браузер технологий (плагинов, апплетов). Сам браузер получил возможности для захвата медиа, принятия и отправки потоков только с появлением WebRTC. В этом заключается самое главное положительное отличие WebRTC от других технологий.

На таблицах ниже представлено итоговое сравнение трех описанных выше технологий.

Таблица 1 - Поддерживаемые протоколы и технологии

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Java** | **Flash** | **WebRTC** |
| UDP | **\*** | **-** | **-** |
| TCP | **\*** | **-** | **-** |
| RTMP | **\*** | **+** | **-** |
| RTMFP | **\*** | **+** | **-** |

Таблица 1 - Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| RTP | **\*** | **-** | **+** |
| SRTP | **\*** | **-** | **+** |
| DTLS | **\*** | **-** | **+** |
| SIP | **\*** | **-** | **-** |
| ICE | **\*** | **-** | **+** |
| STUN | **\*** | **-** | **+** |
| TURN | **\*** | **-** | **+** |

Таблица 2 - Поддерживаемые аудио кодеки

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Java** | **Flash** | **WebRTC** |
| G.711 | **\*** | **+** | **+** |
| G.729 | **\*** | **-** | **-** |
| Speex | **\*** | **+** | **-** |
| NellyMoser | **\*** | **+** | **-** |
| Opus | **\*** | **-** | **+** |
| iLBC | **\*** | **-** | **-** |

Таблица 3 - Поддерживаемые видео кодеки

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Java** | **Flash** | **WebRTC** |
| H.264 | **\*** | **+** | **-** |
| H.263 | **\*** | **-** | **-** |
| VP8 | **\*** | **-** | **+** |

Таблица 4 - Поддержка браузерами

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Java** | **Flash** | **WebRTC** |
| Firefox | **+** | **+** | **+** |
| Chrome 30 | **+** | **+** | **+** |
| IE 10 | **+** | **+** | **-** |
| Safari | **+** | **+** | **-** |
| Opera | **+** | **+** | **+** |
| Firefox Mobile | **?** | **?** | **+** |
| Chrome Mobile | **?** | **?** | **+** |

Знаком “\*” обозначены технологии, которые теоретически могут быть реализованы, вопрос только в том, сколько на это понадобится времени и ресурсов. Знаком “?” обозначены браузеры, которые имеют частичную поддержку технологии.

Из приведенных выше таблиц видно, что WebRTC имеет поддержку большего количества протоколов и технологий установки соединения. WebRTC поддерживает меньше видео и аудио кодеков, но за счет открытости исходного кода эта проблема может быть решена сторонними разработчиками. Важное положительное отличие WebRTC от других технологий - архитектура “точка-точка”, избавляющая от необходимости использования мощного медиа-сервера. И самым важным отличием WebRTC является отсутствие в необходимости использования дополнительного программного обеспечения, так как браузер, поддерживающий WebRTC, имеет весь функционал, необходимый для организации потоковой передачи данных. Исходя из этого, можно сделать вывод о том, что WebRTC наиболее полно отвечает требованиям современного сообщества web-разработчиков.

**1.2 Внутренняя архитектура WebRTC**

Общая архитектура WebRTC представлена на рисунке 1.

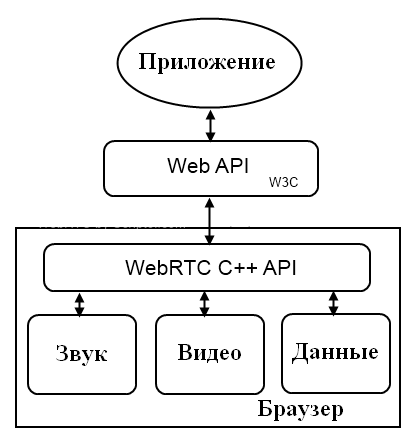


Рисунок 1 - Общая архитектура WebRTC

Основной частью стека WebRTC является C++ API - набор библиотек и интерфейсов, написанных на C++. Предполагается, что разработчики различных web-клиентов или браузеров могут встроить в свои приложения набор C++ компонентов WebRTC и обеспечить к ним доступ через высокоуровневый Web API на JavaScript. Этот набор библиотек обеспечивает весь функционал стека WebRTC:

Транспорт - компоненты для организации подключения и управления пользовательскими сессиями, представляют из себя переработанные и дополненные компоненты Libjingle. Имеют функционал для обхода NAT, прокси и файерволов путем предварительного согласования соединения через STAN и TURN сервера, передачи потока по протоколам RTP или SRTP, работающим поверх TCP и UDP, шифрования потока при передаче по SRTP, управления полосой пропускания.[18]

Движок обработки аудио - компонент, обеспечивающий передачу аудиосигнала от аудиокарты к сетевому интерфейсу и обратно. Он включает в себя аудио кодеки iSAC и Opus, системы эхоподавления и подавления шума. Кодек iSAC является широкополосным аудио кодеком для VoIP и потокового аудио, использует частоту дискретизации 16/32 кГц с адаптивной передачи данных от 12 до 52 Кбит/с. Opus поддерживает постоянный и переменный битрейт от 6 до 510 Кбит/с и частоты дискретизации от 8 до 48 кГц.

Движок обработки видео - компонент для управления видеосигналом, включающий кодек VP8, систему адаптивного видеобуффера Jitter Buffer, автоматически подстраивающегося под текущее состояние соединения и систему коррекции изображения. Кодек VP8 имеет повышенную устойчивость к потере пакетов, механизм фильтрации артефактов, который может применяться по-разному к разным частям кадра в зависимости от их изменения, возможность масштабировать декодирование, имеет профили, оптимизированные для проведения видео-конференций в реальном времени.

C++ API имеет два основных интерфейса: Stream API и PeerConnection, которые, в свою очередь, подразделяются на несколько дочерних интерфейсов. Stream API предназначен для получения медиапотока от аппаратного обеспечения клиента (веб-камера/микрофон) или преобразования в поток данных, которые предоставит пользователь, например, для вещания заготовленного пользователем файла и обработки видео и аудио потоков. PeerConnection API предназначен для согласования и создания соединения, обхода NAT, файервола или прокси, и передачи потока по сети.[18] Общая схема работы представлена на рисунке 2.

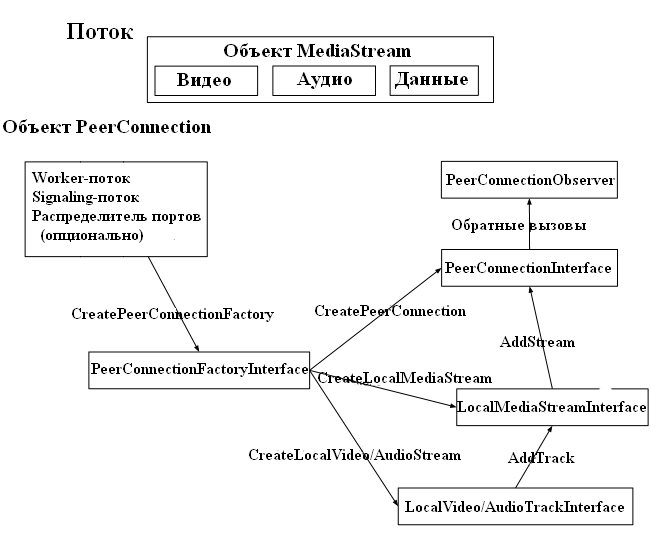


Рисунок 2 - Схема взаимодействия с C++ API

Разработчики приложений могут внедрить в свои приложения C++ API и обеспечить к нему доступ через высокоуровневое Web API на JavaScript, которое описывается стандартом W3C. Стандарт Web API от W3C на данный момент имеет стабильную версию 1.0. Разработчики конечных клиентских приложений могут использовать Web API для доступа к функциям WebRTC, не вдаваясь в подробности внутренней архитектуры WebRTC и концентрируя внимание на архитектуре своего web-приложения. Далее в данной работе будут описываться методы Web API, практическая часть будет основываться на нем же. Web API, также как и C++ API, имеет два основных интерфейса: MediaStream и RTCPeerConnection.

Одной из интересных особенностей MediaStream, является то, как он работает с аудио и видео, принимая данные от каждого устройства и передавая его в виде потока. При получении аудио и видео MediaStream работает отдельно с каждым потоком. Таким образом, есть возможность отправлять или получать от других клиентов сразу несколько независимых потоков. Это особенно актуально для портативных устройств, таких как планшеты и смартфоны, которые могут иметь более одной web-камеры (задняя и фронтальная) или микрофона. MediaStream не обязательно должен быть отправлены через Интернет. Поток может быть использован локально, преобразован, сконвертирован или же записан в файл.

RTCPeerConnection занимается установкой прямого подключения и управляет потоками между клиентами. Он может быть ассоциирован с двумя наборами потоков: входящих и исходящих. Исходящий поток отправляется в объект RTCPeerConnection от MediaStream и вещается другим клиентам, входящий поток передается обратно в MediaStream для обработки и отображения в пользовательском интерфейсе, если это необходимо.

**1.3 Использование Web API**

**1.3.1 Создание подключения**

Для того, чтобы создать объект WebRTC-подключения, необходимо вызвать конструктор **RTCPeerConnection**, который принимает два аргумента в качестве параметров: объект со списком ICE серверов и объект с параметрами подключения. Конструктор вернет объект подключения, на котором будут вызываться все остальные описанные ниже методы.[10]

Несмотря на то, что WebRTC позволяет реализовать прямую передачу данных между браузерами, необходимо использовать промежуточные сервера для координации соединения, обхода NAT и файерволов. Signaling, процесс координации подключения, необходим для того, чтобы браузеры перед началом передачи данных могли обменяться служебной информацией. Эта служебная информация передается от клиента к клиенту в формате SDP и может содержать:

* сообщениями управления сессиями, которые необходимы для открытия или закрытия сеанса связи;
* сообщениями об ошибках;
* метаданными, такими как информация о кодеках, ширине полосы пропускания и типе передаваемых данных;
* ключевыми данными для установки защищенного соединения;
* сетевой информацией, такой как IP-адрес и порт.[13]

Пример SDP при координации WebRTC-соединения для передачи аудио и видео показан на рисунке 3.



Рисунок 3 - Пример SDP-сообщения

SDP генерируется методами объекта RTCPeerConnection **createOffer** и **createAnswer**. Первый создает SDP для запроса подключения, второй - SDP для ответа. Когда клиент получает от другого клиента запрос на подключение, он может отправить ответ, сгенерированный createAnswer. После того как оба клиента сохранят полученные друг от друга SDP методом **setRemoteDescription** объекта RTCPeerConnection, а свои собственные SDP сохранят методом **setLocalDescription**, соединение между этими клиентами будет создано.Процесс отправки и принятия SDP-сообщений может быть реализован при помощи серверных сокетов или баз данных реального времени.

Большинство пользователей в Интернет защищены несколькими слоями NAT, могут иметь антивирусное программное обеспечение, блокирующее определенные порты и протоколы или выходят через прокси и файерволы и поэтому не могут обмениваться информацией. Файервол и NAT может быть реализован одним и тем же устройством, таким как обычный Wi-Fi роутер. WebRTC приложения могут использовать ICE, чтобы обойти ограничения, наложенные NAT и файерволами.[13] Для этого необходимо задать объекту RTCPeerConnection список STUN и TURN серверов, как это показано на рисунке 4.



Рисунок 4 - Создание RTCPeerConnection со списком ICE-серверов

ICE будет пытаться найти наилучший путь для подключения. Он попробует все возможности и выбирает наиболее эффективный вариант. ICE сначала попытается установить соединение с использованием адреса, полученного от операционной системы и сетевой карты. Если это не удается (клиент находится за одним или несколькими NAT), ICE получает внешний адрес, используя сервер STUN, и если это также не удается, трафик направляется через TURN сервера. Другими словами:

* STUN сервер используется для получения внешнего сетевого адреса клиента, находящегося за NAT;
* TURN сервер используются для передачи трафика, если прямое соединение не удается.

**1.3.2 Получение локального потока**

Получение доступа и получение медиа потоков от пользователя реализуется средствами, которые предоставляет MediaStream API. Основнной метод MediaStream - метод **getUserMedia**, вызываемый на глобальном объекте **navigator**. getUserMedia принимает в качестве параметров три аргумента. Первый указывает на то, какие данные необходимо получить - аудио, видео или оба. Второй аргумент - функция, которая будет вызвана при удачном получении потока, аргументом этой функции будет объект потока. Третий аргумент - функция, которая будет вызвана при возникновении ошибки либо в том случае, если пользователь запретит взятие медиа потоков. После вызова getUserMedia браузер запросит у пользователя разрешение на взятие потока. Имя метода getUserMedia отличается в разных браузерах и вызывается с префиксами, потому что WebRTC еще не до конца стандартизирован. Так, в Mozilla Firefox, getUserMedia вызывается с префиксом **moz**, в Chrome - с префиксом **webkit**.[12] Простой пример вызова getUserMedia для получения аудио и видео на рисунке 5.

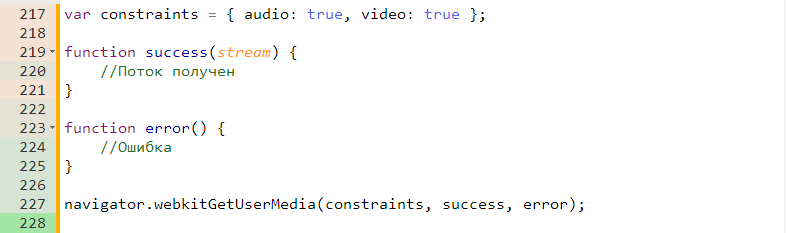


Рисунок 5 - Вызов getUserMedia

После того, как локальный поток получен, он может быть показан пользователю. Это реализуется средствами HTML5. Видео может быть отображено на web-странице путем добавления HTML-элемента **video**, в который добавлен поток через свойство src как показано на рисунке 6.



Рисунок 6 - Прикрепление потока

После того, как поток пользователя получен и отображен, он может быть отправлен другому пользователю.

**1.3.3 Отправка и получение потоков**

Для отправки локального потока, полученного из getUserMedia, другим клиентам, необходимо вызвать метод **addStream** объекта RTCPeerConnection с потоком в качестве параметра.

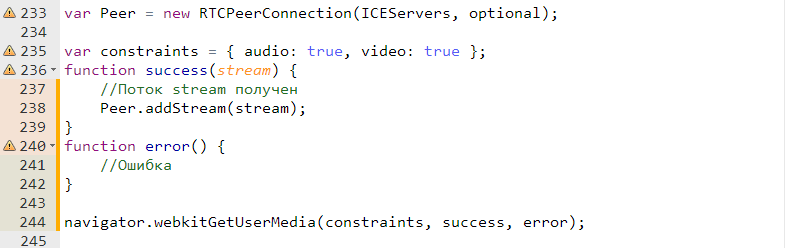


Рисунок 7 - Отправка потока в RTCPeerConnection

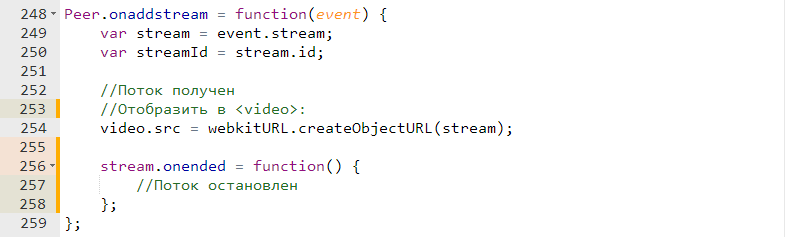
После координации подключения и создания объекта RTCPeerConnection другой клиент может получить входящий поток, когда сработает событие **onaddstream** объекта RTCPeerConnection. Это событие вернет объект, содержащий поток клиента и информацию о соединении. 

Рисунок 8 - Отлов входящего потока

Таким образом можно организовать простейший сеанс потоковой передачи данных, используя основные методы Web API WebRTC.

**2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ НА ОСНОВЕ WEB API**

**2.1 Web-приложение, внедряемое на сторонний ресурс**

WebRTC позволяет организовывать видеосвязь средствами JavaScript и HTML5, иными словами, конечное клиентское приложение может быть размещено на сайте или web-странице. Первой практической реализацией в рамках данной дипломной работы была попытка создать переносимое web-приложение, которое могло бы быть встроено на сторонний сайт для определенных клиентов. Это было бы полезно, если есть необходимость в потоковой передаче данных между клиентами определенного ресурса, в то время как нет доступа к исходному коду этого ресурса и нет возможности его модифицировать. Например, если функционал реализуется сторонним разработчиком или самими клиентами ресурса. Для реализации этой задачи была выбрана социальная сеть Вконтакте. Чтобы создать некое подобие интеграции приложения на сайт, было выбрано встроить его в браузерное расширение, которое будет запускать основной код при заходе на сайт, а основной исполняемый код WebRTC-приложения разместить на стороннем хранилище, откуда он будет подгружаться в iframe. Весь исходный код находится в приложении Б данной дипломной работы.

Общая структура приложения имеет вид, изображенный на рисунке 9.

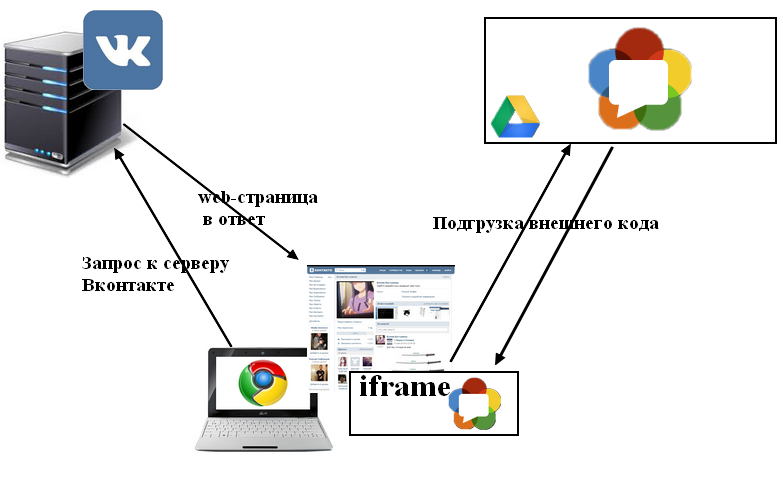


Рисунок 9 - Схема работы приложения

Для обмена SDP-сообщениями было решено использовать базу данных Firebase, она будет выполнять роль сигнального сокета. В качестве облачного хранилища для исполняемого JavaScript был выбран Google Drive. Это обосновано возможностью получения доступа к файлу по защищенному https протоколу.

Реализацию подобного приложения можно разбить на основные этапы:

1. написание WebRTC-приложения и размещение на облачном хранилище;
2. создание браузерного расширения, взаимодействующего с сайтом;
3. разбор HTML-верстки необходимых страниц сайта и реализация функций парсинга;
4. интегрирование в расширение кода, взаимодействующего со страницами сайта;
5. подключение внешнего WebRTC-приложения к расширению;
6. внедрение WebRTC-приложения в страницу сайта.

Для реализации WebRTC-приложения была написана JavaScript-библиотека RTCPeerBroadcasting, имеющая функции для создания подключений между клиентами в режиме конференции и передачи потоковых данных. Исходный код библиотеки RTCPeerBroadcasting находится в приложении А.

Для создания объекта подключения в библиотеке реализована функция RTCPeerConnection, принимающая в качестве параметра объект с настройками подключения.

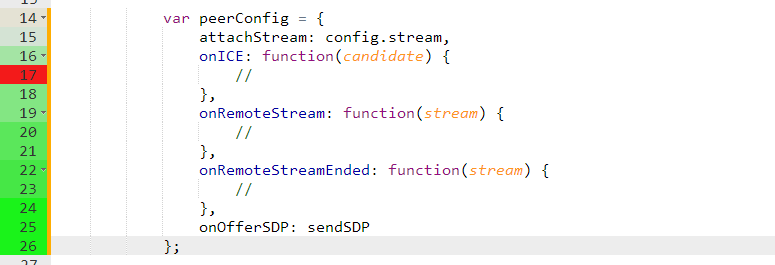


Рисунок 10 - Объект параметров функции RTCPeerConnection

Пример такого объекта показан на рисунке 10, где attachStream - объект видео потока, который будет отправлен другому клиенту, onICE - функция-обработчик события подключения нового клиента, отправляющая ему ответ, onRemoteStream - функция-обработчик события получения потока от клиента, onRemoteStreamEnded - функция-обработчик события остановки потока, onOfferSDP и onAnswerSDP - функции отправки запросов и ответов на запрос подключения. Функция RTCPeerConnection при вызове создает объект WebRTC-подключения, отправляет запрос на подключение к другому клиенту и поток, если он задан. После того, как клиент подтвердит подключение и отправит ответ, для установки соединения можно вызвать функцию addAnswerSDP, которая сохраняет SDP клиента.

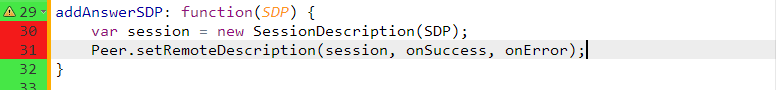


Рисунок 11 - Сохранение SDP

Функции onOfferSDP и onAnswerSDP изначально не заданы в коде функции PeerConnection, они задаются при ее вызове. Это сделано для того, чтобы данная библиотека была как можно более переносимой. В WebRTC нет встроенных функций для обмена SDP-сообщениями, поэтому программист может сам выбрать, что использовать для этого: web-сокеты, базу данных или собственный сигнальный сервер.

Вторая важная функция класса RTCPeerBroadcasting - функция Broadcasting. Эта функция является оберткой для RTCPeerConnection, выполняет роль слоя абстракции и создана для упрощения создания конференц-связи, когда подключенных клиентов более одного. Принимает на вход объект, пример которого показан на рисунке 12.

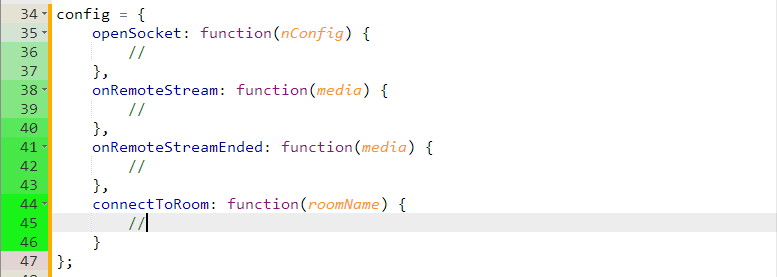


Рисунок 12 - Объект параметров Broadcasting

В данном объекте openSocket - функция открытия сокета, в которой необходимо задать функции обмена SDP-сообщениями, onRemoteStream и onRemoteStreamEnded - функции-обработчики начала и завершения потоков, connectToRoom - функция, которая будет вызвана при подключении к уже существующей конференции. При вызове Broadcasting отправляет в сокет SDP и ожидает ответ. Если ответ получен, создается новый объект RTCPeerConnection для нового подключенного клиента и устанавливается соединение. Broadcasting возвращает объект, содержащий методы createRoom, joinRoom, leaveRoom. Метод createRoom создает новую конференцию с указанным именем, после вызова отправляет в сокет сообщение с информацией о новой конференции. Метод joinRoom отправляет запрос на подключение к уже существующей конференции, создает новый объект RTCPeerConnection. Метод leaveRoom запускает цикл по списку всех объектов RTCPeerConnection и на каждом вызывает метод **close**, который закрывает прямое подключение. В итоге библиотека имеет простой интерфейс для создания конференц-подключений. Эта библиотека является основой для WebRTC-приложения, которое будет подгружаться расширением на страницу Вконтакте.

Для данного приложения еще необходимо было создать пользовательский интерфейс и функционал для создания некоторого подобия вызовов и ответов на видеозвонок. Так как структура данного приложения предполагает, что приложение будет подгружаться в iframe на странице с другим доменом, то необходимо реализовать механизм обмена данными между родительским окном и содержимым iframe. Для этого был выбран механизм POST-сообщений. Они отправляются в виде строки. Чтобы отправить массив с данными, можно закодировать его в JSON-строку, а при получении декодировать.

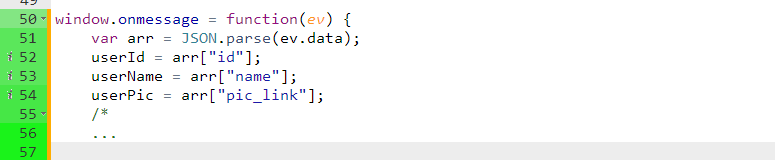


Рисунок 13 - Отлов POST-сообщения

При получении POST-сообщения с необходимыми данными (id звонящего и принимающего, ссылка на фотографию, имя) скрипт запускает новую конференцию, вызывая метод createRoom библиотеки RTCPeerBroadcasting, в качестве имени конференции будет указан id принимающего пользователя. После того, как конференция создана, в базу данных будет записана информация о вызове. Если принимающий вызов пользователь будет в режиме онлайн, то он получит сообщение о вызове из базы данных. Если пользователь отклонит вызов, то в базу данных будет отправлено соответствующее сообщение, которое получит вызывающий. Если пользователь примет вызов, то браузерное расширение откроет iframe и загрузит страницу с WebRTC-приложением. В него также будет отправлено POST-сообщение о принятии вызова, после чего будет вызван метод joinRoom библиотеки RTCPeerBroadcasting. Пользователь будет подключен к конференции и между ним и вызывающим будет установлено прямое подключение.

После завершения работы над WebRTC-приложением было написано браузерное расширение. В манифесте расширения было указано, с какими доменами ему будет разрешено работать:

*"permissions": ["http://vk.com/\*", "https://vk.com/\*", "notifications", "tabs", "storage"]*

К Вконтакте можно получить доступ по защищенному протоколу https, и расширению будет разрешено работать по http и https, но, чтобы исключить возможность перехвата данных третьми лицами, в коде, который взаимодействует непосредственно со страницей сайта (файл content.js) был добавлен редирект на https, если пользователь зашел на сайт через http.



Рисунок 14 - Редирект

Когда пользователь заходит на страницу сайта, в базу данных сохраняется информация о том, что пользователь находится в режиме онлайн. При закрытии вкладки браузера эта информация будет удалена. Для работы приложения была необходима информация о пользователе: имя, id, фотография. Для получения этой информации отправляется асинхронный запрос на адрес *vk.com/id0*, с которого автоматически производится редирект на страницу пользователя. При помощи парсинга по селекторам классов из содержимого страницы вытаскивается все необходимое.

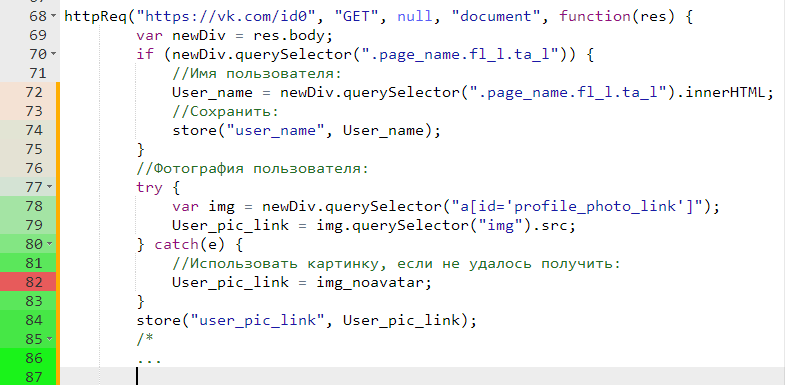


Рисунок 15 - Парсинг полученной страницы

После получения данных на директорию пользователя в базе данных вешается обработчик получения новых сообщений о входящем видеовызове. При вызове создается всплывающее окно с кнопками для принятия и отклонения вызова. Одновременно с этим проверяется адрес текущей страницы. Это необходимо для того, чтобы проверить, находится ли пользователь на личной странице другого пользователя. Если это так, то из содержимого данной страницы будет получена информация об этом пользователе, чтобы проверить, имеет ли этот пользователь установленное расширение. На странице пользователя, который тоже имеет расширение, будет отображена кнопка для совершения видеовызова. При нажатии на нее в базу данных отправляется сообщение о вызове, а на странице загружается iframe с WebRTC-приложением, в который отправляется POST-сообщение.

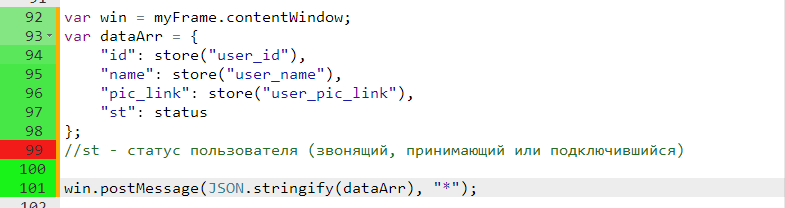


Рисунок 16 - Отправка POST-сообщения

Вторым параметром функции postMessage должен быть указан домен, на который разрешено отправлять POST-сообщение, в примере выше же указан символ \* для того, чтобы разрешить отправку на любой домен, так как iframe в данном приложении будет подгружаться со стороннего сервера.

Когда пользователь будет подключен к конференции, это отобразится на его странице Вконтакте. Другим пользователям приложения в этом случае вместо кнопки вызова будет отображена кнопка подключения к конференции. При подключении, как и при принятии, вызова будет загружен iframe и произойдет попытка подключения к сокету методом joinRoom библиотеки RTCPeerBroadcasting.

Помимо уведомления о вызове в виде всплывающего окна необходимо было также добавить звуковое уведомление, которое пользователь смог бы настраивать сам. Это было реализовано средствами Audio API. Оно позволяет воспроизводить звуковые файлы как полученные в виде ссылки на файл в Интернете, так и в виде файла формата MP3. Для реализации возможности установки аудиозаписи в качестве мелодии вызова был задействован парсинг для того, чтобы обнаружить на странице Вконтакте элементы аудиозаписей. Поиск производится по селектору *audio*. При нахождении элементов аудио в них добавляются кнопки установки мелодии вызова.

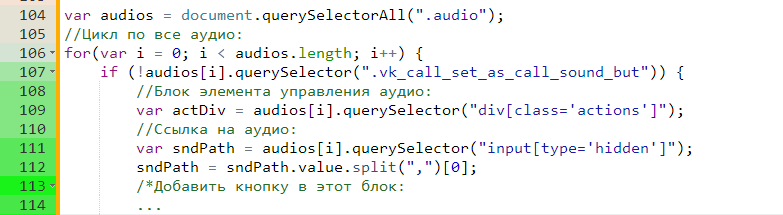


Рисунок 17 - Поиск аудио-элементов

При нажатии на кнопку ссылка на аудио сохраняется как мелодия вызова. В расширение была добавлена страница настроек, на которой пользователь может выбрать в качестве мелодии вызова аудио из списка.

После завершения написания всего программного кода и тестирования был сверстан пользовательский интерфейс всех элементов приложения на HTML. Общий вид элементов завершенного приложения показан на скриншотах (рисунки 18, 19, 20 и 21).

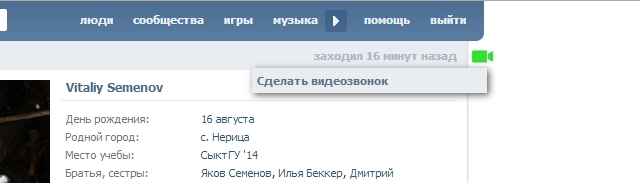


Рисунок 18 - Кнопка видеовызова

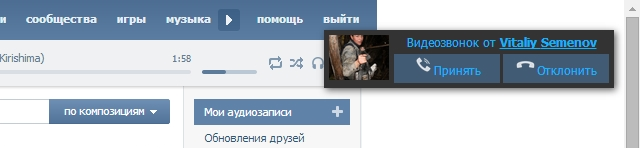


Рисунок 19 - Входящий вызов

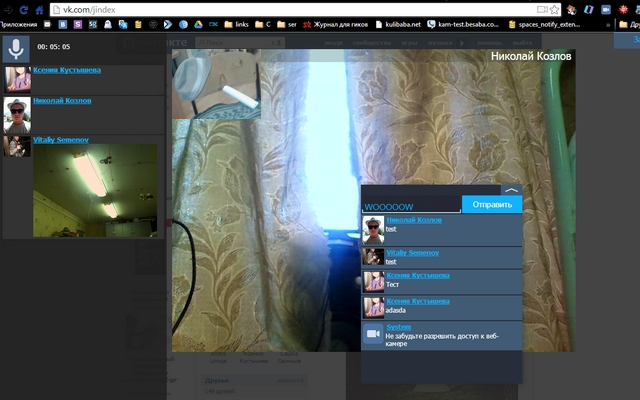


Рисунок 20 - Iframe с приложением. Конференция с тремя участниками



Рисунок 21 - Установка аудио в качестве сигнала вызова

Данная практическая часть этого дипломного проекта была попыткой проверить, возможно ли уже сейчас, даже несмотря на недостаточную стандартизированность WebRTC, реализовать полноценное переносимое WebRTC-приложение, которое могло бы быть встроено на Интернет-ресурс со стороны клиента при помощи браузерного расширения. Поставленная задача была успешно выполнена. Так как приложение предполагалось запускать при помощи расширения для браузера Google Chrome, не было необходимости в реализации кроссбраузерной поддержки, что существенно упростило разработку.

**2.2 WebRTC-сервис**

Второй практической реализацией был web-сервис, использование которого возможно в разных браузерах, поддерживающих WebRTC. Исходный код находится в приложении Г данной дипломной работы. Сервис, так же как и первая практическая реализация данной дипломной работы, основывается на библиотеке RTCPeerBroadcasting. Для того, чтобы реализовать возможность использования в различных браузерах, необходимо было переписать функции, взаимодействующие с функциями Web API WebRTC и добавить в них необходимые префиксы. Это касается объекта **window.URL**, функций **getUserMedia**, **PeerConnection**, **SessionDescription**. Чтобы сделать выбор нужного префикса автоматическим, используется конструкция с условным ИЛИ, перебирающая методы с разными префиксами.[2]

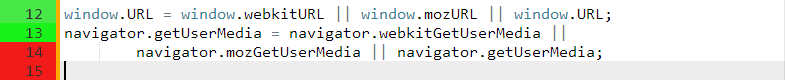


Рисунок 22 - Перебор префиксов

То же самое сделано для функций PeerConnection и SessionDescription.

Процесс прикрепления медиа потока к HTML-элементу video также различается в браузерах на движке Webkit и Gecko. Так, в Mozilla поток прикрепляется через свойство **mozSrcObject**, а в Chome - через свойство **src**.[9] Чтобы сделать процесс прикрепления потока универсальным, в начало библиотеки добавлена проверка браузера, от которой зависит процесс добавления потока.



Рисунок 23 - Проверка User-Agent

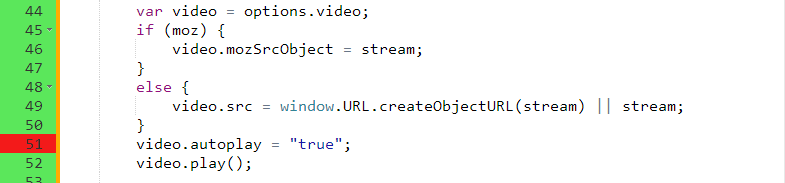


Рисунок 24 - Прикрепление потока к видео-элементу

Некоторые сложности при реализации кроссбраузерности вызвало недекларированное различие в реализации метода **onaddstream** объекта RTCPeerConnection в браузерах на движках Webkit и Gecko. В браузере Google Chrome (Webkit) из объекта stream можно получить набор полей, в том числе **id** и **label**, по которым можно однозначно установить связь между клиентом и потоком, который исходит от него. В браузере Mozilla Firefox (Gecko) объект stream не содержит таких полей. Об этом не говорится в документации, поэтому это вызвало определенные проблемы на этапе тестирования библиотеки. Данная проблема была решена путем переопределения объекта stream и добавления в него уникального ключа.

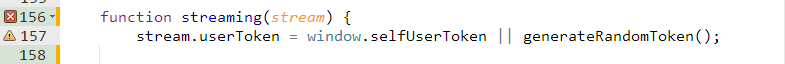


Рисунок 25 - Добавление ключа к потоку

Вышеописанные изменения добавили поддержку основными браузерами, такими как Mozilla, Chrome, Opera.

Второй задачей данной реализации было упростить процесс создания видеоконференций и реализовать дополнительный функционал, такой как, например, блокирование пользователей, выключение камеры и микрофона, сохранение дополнительной информации о пользователе. Для этого был написан класс Conference, выполняющий роль слоя абстракции над библиотекой RTCPeerBroadcasting и упрощающий работу с ней. Исходный код класса Conference находится в приложении В.

Конструктор Conference принимает в качестве параметров два аргумента: адрес базы данных Firebase и имя конференции. На новом объекте Conference вызывается метод startConnection для подключения или создания новой конференции. Этот метод принимает в параметрах имя пользователя и функции обратных вызовов. Имя пользователя проверяется на наличие и корректность методом checkLoginAlreadyInUse.

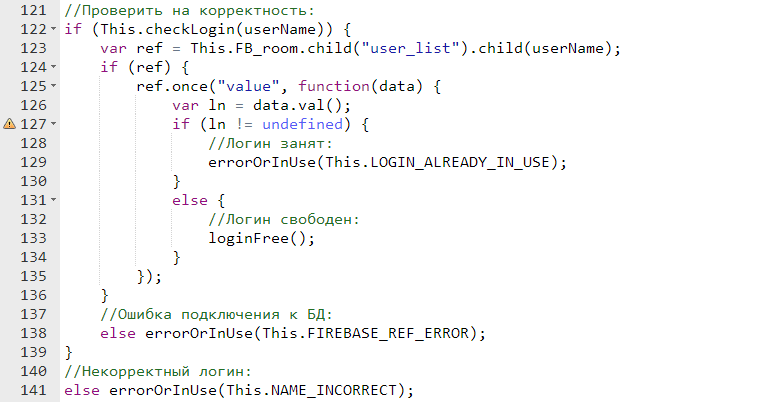


Рисунок 26 - Метод checkLoginAlreadyInUse

Если имя может быть использовано, то производится проверка количества пользователей в данной конференции. В зависимости от количества пользователей будет произведено либо подключение к конференции, либо создание новой (если пользователей нет). Для проверки количества проверяется значение в соответствующей директории базы данных.

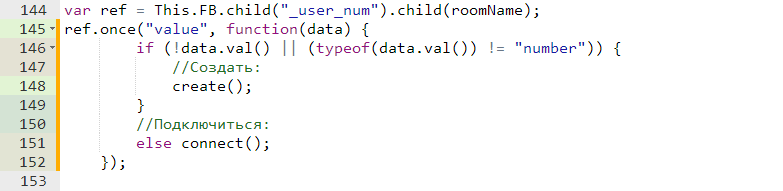


Рисунок 27 - Проверка количества пользователей

После всех проверок создается новый объект RTCPeerBroadcasting, на котором вызывается метод joinRoom или createRoom, а в базу данных записывается информация о пользователе. Таким образом, все проверки автоматизируются и процесс создания или подключения к конференции заметно упрощается по сравнению с использованием библиотеки RTCPeerBroadcasting, требуя лишь вызова одного метода startConnection.

Для отключения от конференции реализован метод disconnect, который вызывает метод leaveRoom библиотеки RTCPeerBroadcasting и производит отключение от базы данных с удалением директории пользователя. Этот метод может быть использован как для самостоятельного отключения, так и для автоматического, когда пользователь будет выгнан из конференции.

Для реализации функции отключения других пользователей добавлен метод kickUser, принимающий в качестве параметра имя пользователя. При вызове этого метода в директорию указанного пользователя записывается значение “kicked”. При подключении к конференции на директорию пользователя вешается событие появления значения “kicked”, при срабатывании которого производится вызов метода disconnect.

Для реализации функций отключения камеры и микрофона добавлены методы switchAudio и switchVideo, принимающие параметром логическое значение (true или false). Чтобы переключать вещание видео или аудио потоков задействованы методы Stream API, а именно: необходимым исходящим потокам задается свойство **enabled**.

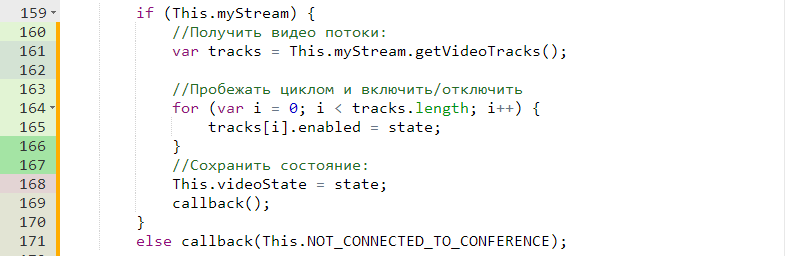


Рисунок 28 - Реализация метода switchVideo

Процесс получения потоков от других пользователей реализован в виде события. После создания объекта Conference необходимо задать обработчик события onUserConnected, при срабатывании которого будет получено имя пользователя, идентификатор потока и HTML-видео. Обработчик события вызывается из функции onRemoteStream объекта RTCPeerBroadcasting как это показано на рисунке 29.

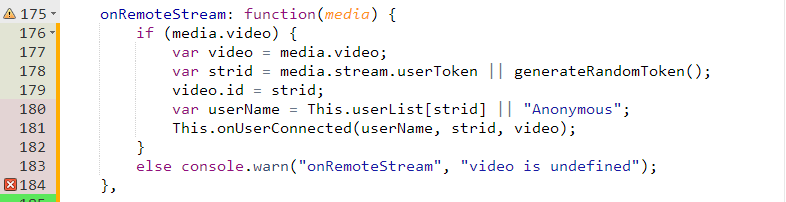


Рисунок 29 - Событие onRemoteStream

Помимо этого объект Conference имеет еще три события: onUserDisconnected, onKicked и onUserCount.

Класс Conference, помимо реализации базового функционала, необходимого для создания видеоконференций, имеет функцию слоя абстракции над RTCPeerBroadcasting, упрощающего работу с ним, автоматизируя процессы работы с сигнальным сокетом, чтения и записи служебных данных, проверки корректности данных, проверки наличия сокетов. Так, для того, чтобы создать подключение, даже если пользователи находятся на разных web-страницах или приложениях, достаточно импортировать три библиотеки (Firebase, RTCPeerBroadcasting и Conference) и вызвать три метода класса Conference (рисунок 30).

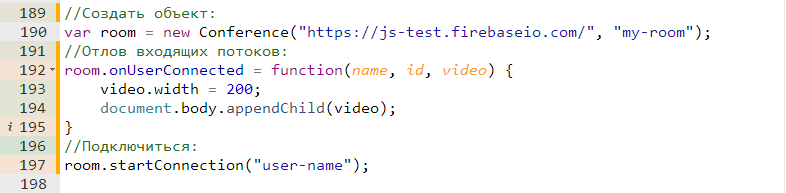


Рисунок 30 - Вызов трех основных методов для создания подключения

Иными словами, можно значительно упростить работу с WebRTC, если написать собственные интерфейсы доступа к функциям Web API.

После того как была закончена разработка класса Conference, был сверстан интерфейс для WebRTC-сервиса, который был связан с методами класса Conference. Помимо страницы конференции, была добавлена стартовая страница, чат, возможность добавления тегов для присвоения конференции статуса публичной и страница со списком публичных конференций. В конечном итоге, сервис имел вид, показанный на рисунках 31 и 32.

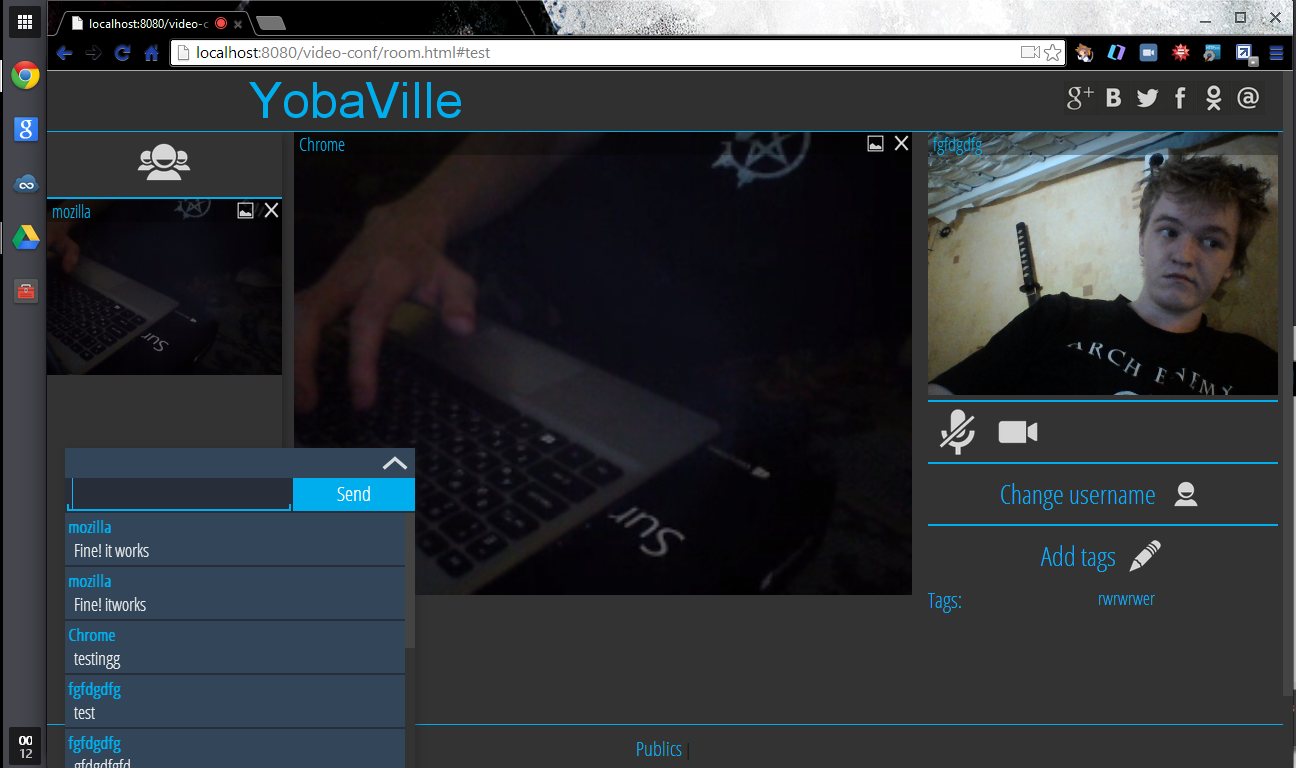


Рисунок 31 - Интерфейс приложения с запущенной конференцией с тремя пользователями

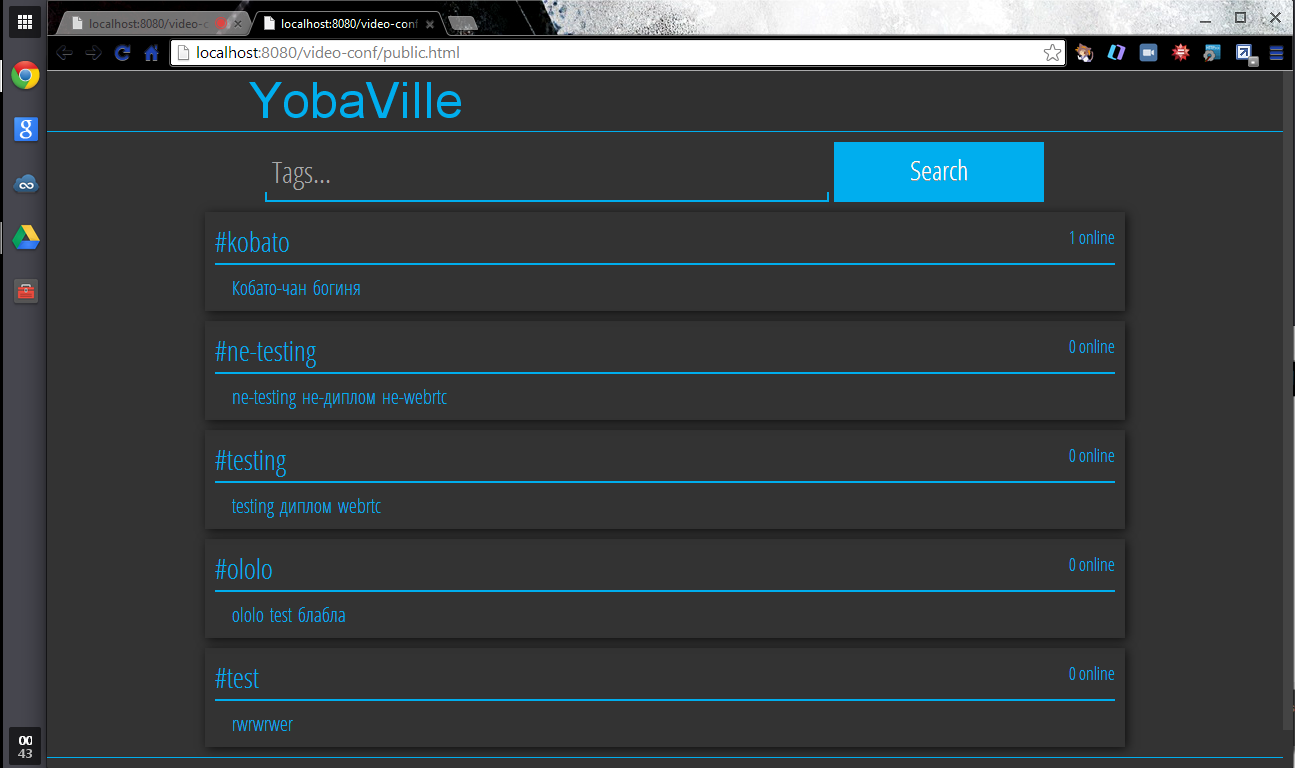


Рисунок 32 - Список публичных конференций

**ЗАКЛЮЧЕНИЕ**

В рамках данной дипломной работы была изучена технология WebRTC и возможности ее применения.

Ключевым фактором успеха развития Интернета в целом было то, что технологии, используемые в нем были изначально открытыми и свободными для реализации. WebRTC стал первым набором технологий, имеющим такую же открытость и дающим при этом возможности высококачественной полнодуплексной потоковой передачи данных. Основная идея проекта WebRTC была в том, чтобы сделать технологии подключений реального времени открытыми и простыми для использования. До появления WebRTC технологии, дающие аналогичные возможности, были закрытыми и требовали от разработчика наличия лицензии.

В целях изучения возможностей применения WebRTC было изучено Web API, архитектура которого была описана в разделе 1. На основе полученных знаний об API было создано два web-приложения. Ход проделанной работы и основные нюансы разработки были описаны в разделе 2 данной дипломной работы.

По результатам проделанной работы можно сделать выводы о том, что, несмотря на неполную стандартизированность, применение WebRTC в реальных проектах уже сейчас вполне оправдано, так как открытых аналогов технологии WebRTC на данный момент не существует. Практические реализации в данной дипломной работе показали то, что технология, как и заявляют ее разработчики, проста в использовании, а благодаря открытости исходного кода может быть еще более упрощена или оснащена дополнительным функционалом.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Basics of WebRTC peer connection. [Электронный ресурс]. Режим доступа: https://medium.com/programming-ideas-tutorial-and-experience/basics-of-webrtc-peer-connections-c1ed743de2f6 [Дата обращения: 20.04.2014 г.].

2. Daniel Davis. Cross-browser camera capture with getUserMedia. [Электронный ресурс]. Режим доступа: https://hacks.mozilla.org/2013/02/cross-browser-camera-capture-with-getusermediawebrtc/ [Дата обращения: 10.02.2014 г.].

3. Eric Rescorla. WebRTC security architecture. [Электронный ресурс]. Режим доступа: http://www.ietf.org/proceedings/82/slides/rtcweb-13.pdf [Дата обращения: 03.04.2014 г.].

4. Firebase API reference. [Электронный ресурс]. Режим доступа: https://www.firebase.com/docs/javascript/firebase/index.html [Дата обращения: 24.02.2014 г.].

5. Flash vs WebRTC в борьбе за браузерные VoIP звонки - флог. [Электронный ресурс]. Режим доступа: http://flash-ripper.com/blog/flash-vs-webrtc-v-borbe-za-brauzernye-voip-zvonki?utm\_source=feedburner&utm\_medium=feed&utm\_campaign=Feed%3A+flash-ripper+%28Flash+Ripper+%7C+Ru%29 [Дата обращения: 11.03.2014 г.].

6. GetUserMedia: Accessing the camera and privacy UI. [Электронный ресурс]. Режим доступа: http://dev.opera.com/articles/getusermedia-access-camera-privacy-ui/ [Дата обращения: 11.02.2014 г.].

7. WebRTC FAQ. [Электоронный ресурс]. Режим доступа: https://sites.google.com/site/webrtc/faq#TOC-Are-the-WebRTC-components-from-Goog [Дата обращения: 07.02.2014 г.].

8. Network Address Translation (NAT). [Электронный ресурс]. Режим доступа: http://www.ietf.org/rfc/rfc4787.txt [Дата обращения: 21.05.2014 г.].

9. MediaStream API. [Электронный ресурс]. Режим доступа: https://developer.mozilla.org/en-US/docs/Web/API/MediaStream\_API [Дата обращения: 10.02.2014 г.].

10. Muaz Khan. WebRTC for beginners. [Электронный ресурс]. Режим доступа: https://www.webrtc-experiment.com/docs/webrtc-for-beginners.html [Дата обращения: 24.03.2014 г.].

11. Some Frequently Asked Questions about RTP. [Электронный ресурс]. Режим доступа: http://www.cs.columbia.edu/~hgs/rtp/faq.html [Дата обращения: 21.05.2014 г.].

12. Sam Dutton. Getting started with WebRTC. [Электронный ресурс]. Режим доступа: http://www.html5rocks.com/en/tutorials/webrtc/basics/ [Дата обращения: 10.02.2014 г.].

13. Sam Dutton. WebRTC in the real world: STUN, TURN and signaling. [Электронный ресурс]. Режим доступа: http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/ [Дата обращения: 15.02.2014 г.].

14. STUN and TCP too. [Электронный ресурс]. Режим доступа: http://nutss.gforge.cis.cornell.edu/stunt.phpДата обращения: 21.05.2014 г.].

15. Traversal Using Relay around NAT. [Электронный ресурс]. Режим доступа: http://tools.ietf.org/html/rfc5766 [Дата обращения: 21.05.2014 г.].

16. VoIP – vulnerability over internet protocol. [Электронный ресурс]. Режим доступа: http://www.continuitycentral.com/feature074.htm [Дата обращения: 21.05.2014 г.].

17. WebRTC 1.0 API. [Электронный ресурс]. Режим доступа: http://www.w3.org/TR/webrtc/ [Дата обращения: 10.03.2014 г.].

18. WebRTC native APIs. [Электронный ресурс]. Режим доступа: http://www.webrtc.org/reference/native-apis [Дата обращения: 13.03.2014 г.].

19. WebRTC project. [Электронный ресурс]. Режим доступа: http://www.webrtc.org/ [Дата обращения: 21.05.2014 г.].

20. Интерфейс программирования приложений – What is an API?. [Электронный ресурс]. Режим доступа: http://money.howstuffworks.com/business-communications/how-to-leverage-an-api-for-conferencing1.htm [Дата обращения: 21.05.2014 г.].

21. Консорциум Всемирной паутины. [Электронный ресурс]. Режим доступа: http://w3c.org.ru/ [Дата обращения: 21.05.2014 г.].