`

Table of Contents

`

`

Abstract

Goal of this thesis is to summarize theoretical and practical understanding of how to develop a solution in order to enable medical consultancy seekers to search, find, book and realize online consultations with medical professionals using cutting edge web technologies, namely WebRTC.

Developed solution will consist of

- Client side (Web application and Mobile application)
- Server side (Booking Web API, Call Signaling Server, Database)

Special effort is made to emphasize security concerns and solutions to facilitate secure and private information exchange. Because data privacy and secure transmission of data is of utmost importance when developing such applications. Firstly in order to be compliant with government and medical standards. Furthermore, to guarantee that patient and doctor confidentiality and secrecy is protected.

To achieve these goals, the author proposes to use a peer to peer media and data exchange solution (WebRTC) that is native to web browsers and naturally addresses such concerns.

WebRTC enables us to transmit audio, video, data without any direct third party involvement. Furthermore, Peer to Peer nature of this technology implies that there is little chance that communication session between two peers can be easily eavesdropped.

Finally, let’s acknowledged that this is still a cutting edge developing technology and 100% guaranteed security cannot be achieved as with any other technology. Our job as researchers and engineers is to research, find and implement solutions to minimize the risk and provide a working solution that is acceptable for most cases.

Keywords: WebRTC, telemedicine, REST API, mobile app

`

Introduction

Networking and internet technology has enabled us with tools to communicate over vast distances. From email to World Wide Web, from video streaming to online banking to social media these technologies have reformed the landscape of society and economy.

But few challenges still remain for the web is to enable internet users to communicate using audio, video and data in real time natively in their web browsing experience. Real time communication should not be a luxury. It is a necessity as a natural part of our daily web usage almost same as inputting a text into text box or clicking a button on a web page. Without such a technology we are limited in our ability to innovate and develop in our ways with communication. Communication and relaying of information is a cornerstone for innovation and development.

Luckily such technology was in development for the past decade and it is finally maturing enough to allow us internet users to experience Real Time Communication natively in our internet browsing experience without needing any kind of third party plugin. It is called WebRTC specification which consists of few native browser APIs that allow capturing and transmission of any kind of audio video and data in real time, in peer-to peer fashion.

WebRTC technology opens floodgates for innovative disruption in the fields of telecommunication, social media, insurance, medicine and many other fields.

Our topic of interest is to develop such a telemedicine practice solution using WebRTC, allowing patient doctor communication in secure and reliable manner.

`

Chapter 1 - Mobile Applications in Medical Practice

One of the most challenging aspects of Medical Healthcare Services is the ability to consult and get an opinion from a specialist without having to wait in queues and wasting precious time. Healthcare focused mobile applications can help to remedy some of the above mentioned problems.

Popularity of online doctor appointment applications have been on the rise. A report from IHS [1] predicts that virtual doctor consultations will reach 5.4 million by 2020. No doubt there is a huge demand for high-quality healthcare services that can be provided to patients online, will greatly benefit doctors as well.

Advantages for patients

- No more waiting in queues
- Easily search and access best providers in the industry
- Anonymity
- Save money and time
- Review doctors experience and rating according to past consultations

Advantages for doctors

- Work From Anywhere
- Online Presence
- Exposure to more client base
- Self-Advertisement
- 24/7 access for patients

There are many types of medical care applications available in the market, ranging from doctor search engines, appointment booking applications, medical record management, billing management, tele-appointment applications and many combinations of these.

`

1.1 Why is there a necessity for such a solution?

Video calling and data exchange over the internet is not a new thing. There exists a vast amount of applications, systems and solutions to do so. Skype , Cisco WebEx and numerous other proprietary solutions are used today by millions of users. But these existing systems suffer from many disadvantages for a number of reasons:

- Often these solutions are not integral to users' web browsing experience.
- They require you to download and install extra programs and plugins in order to enable video and audio calling abilities.
- The communication session is not streamed peer to peer directly. Often user session is relayed over providers streaming server, which can observe and record ongoing session easily.
- More advanced proprietary solutions can often be too expensive to purchase and maintain.
- These solutions maybe using proprietary technologies that are incompatible to each other
- They might require IT professionals to maintain it.
- Security solutions that are used might be proprietary and not standardized. Thus not up to date.

One of the main requirements for medical applications is to have end to end data and transport security and privacy. There are many solutions available on the market to service such need to connect patients to qualified professionals. But most of them don't guarantee secure way of communication. Most solutions today use some kind of intermediate third party that relays information from patient to doctor, in essence violating patient's right of confidentiality. No one can guarantee that this third party will not look into data in transit.

Main feature of thesis solution is to utilize one of the most cutting edge browser API, which is WebRTC to establish secure data connection between two peers and allow them to exchange data securely over a peer to peer connection. In this scenario there is no third party involved in exchanging confidential data. In this way patient and doctor secrecy is kept.

`

1.2 Examples of Existing Medical Web and Mobile Applications

1.2.1 Practo

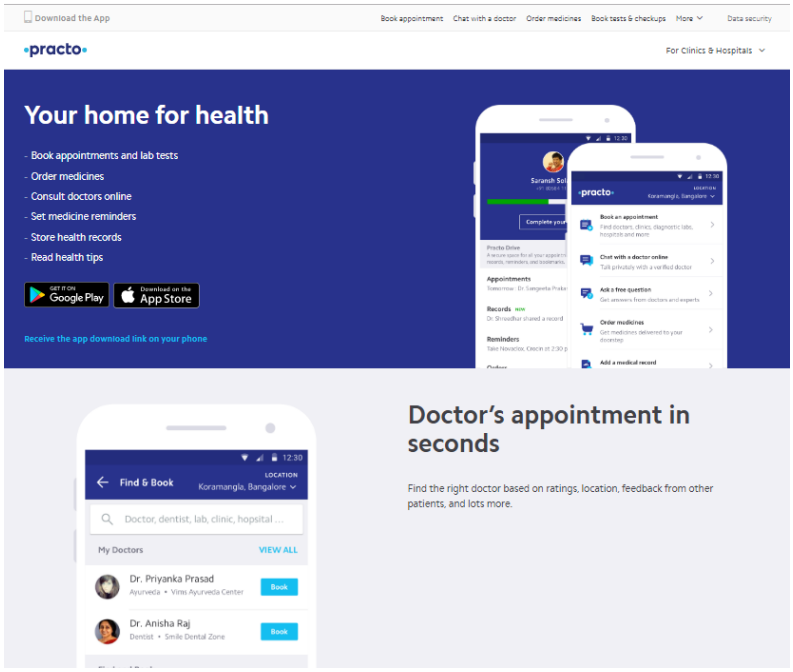


Fig1.1 Practo website and mobile app. [2]

Practo claims to be largest medical professional search platform in Asia. 100 thousand verified doctors served 25 million patients on the platform across the Asia. They provide website and a mobile application for users. Following are main features:

- Appointment of doctors online
- Delivery of medicine
- Storage of medical records
- Medicine reminders
- Health experts writing blogs

`

1.2.2 ZocDoc

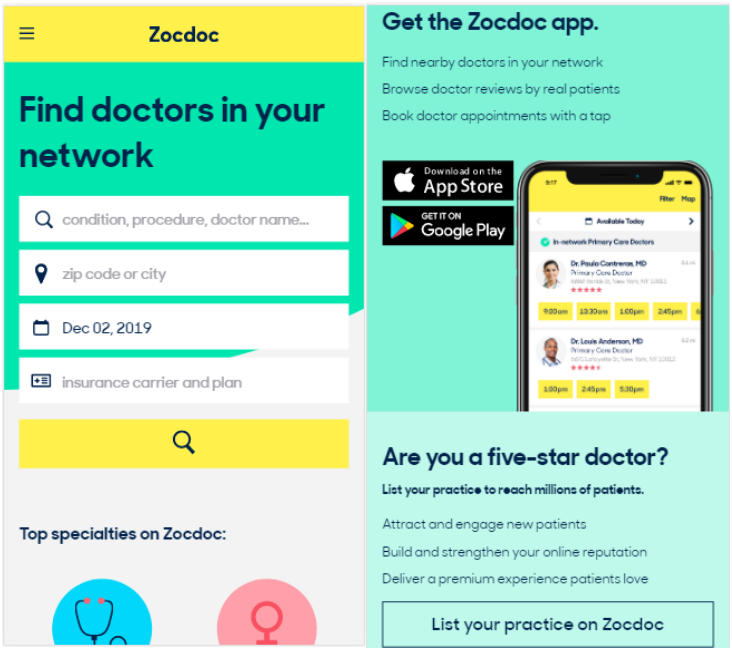


Fig1.2 ZocDoc website and mobile app. [3]

ZocDoc is a fairly popular mobile app for online appointment booking in USA. Every month more than 6 million users use this app across the USA for online appointments. Besides users, doctors also have to register with the application and pay a yearly free. These are the most popular features of the application.

- Online bookings, scheduling and appointments
- In-app notifications
- Map for finding doctors in your location
- Easy search features, such as, patients can search for one word, let's say flu, and be connected to right specialist.

`

1.2.3 KRY

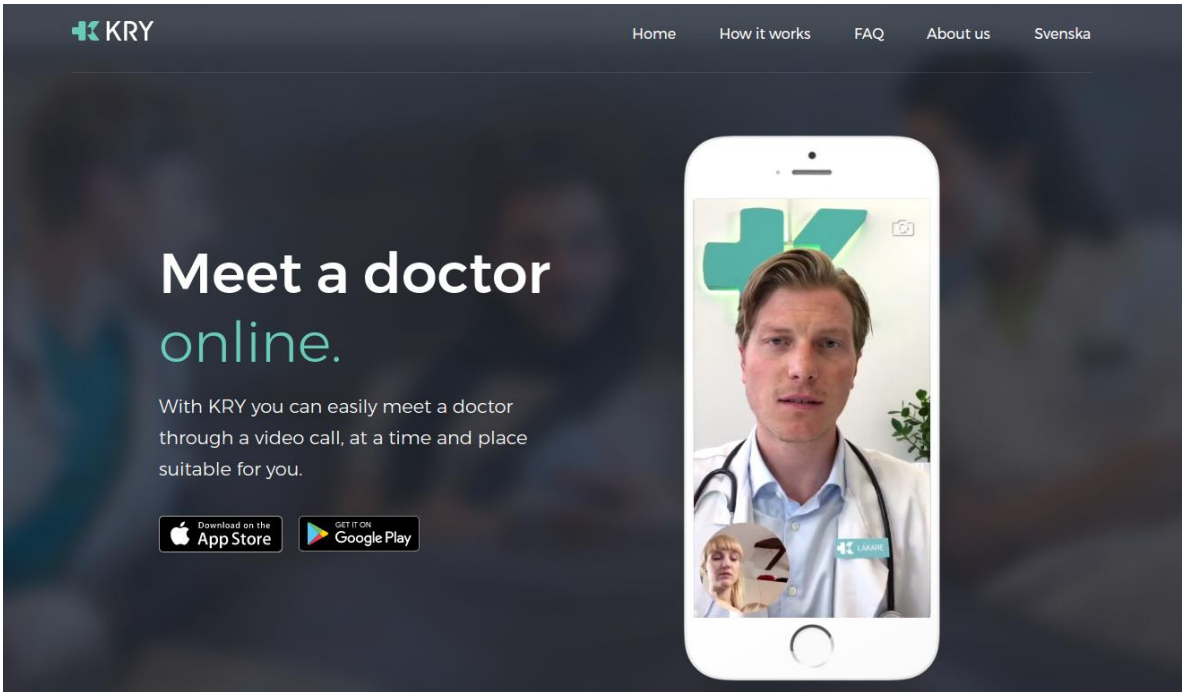


Fig1.2 KRY website and mobile app. [4]

KRY was founded in 2014 as a telehealth startup in Sweden. Their solution allows patients to meet with qualified health professionals within minutes using the mobile app. In Nordic region they served over half a million patients.

User can download the mobile app. Sign in with BankId. After answering some questions about symptoms they can book an appointment for a specific time or see a doctor quickly with “Drop-In” service.

After the appointment is made, doctor reviews patients answers for health questionnaire and calls the patient to realize the appointment. Experience is face to face, just like meeting a doctor at a clinic. Doctor may recommend prescriptions, lab tests or refer to another specialist.

`

1.3 Main Features of Healthcare Applications

There are dozen features that are characteristic to healthcare applications that are very useful. Although, trying to implement every feature is not feasible for this thesis project. Therefore some of the core features were selected for implementation.

Online Consultation – This is the main feature of this thesis project. Allows patients to book and realize online audio, video consultations 24/7. (Implemented in Thesis)

Doctor Search with filters – Search may include such filters as location, specialty, experience, rating, etc. (Implemented in Thesis)

Doctor's Profile – All necessary information such as doctor's experience, spoken languages, available time, rating, etc. should be available for patients. (Implemented in Thesis)

Appointment Calendar – A calendar with available date and time. (Implemented in Thesis)

Online Prescriptions – After successful consultation doctor may prescribe some drugs.

Inapp Payment – Depending on the duration of the consultation session, apps can withdraw payment from patients account.

Push Notifications – Inapp reminder about upcoming sessions for both doctor and patients. Can also be used for reminding timing of online prescriptions.

Online Medicine Ordering – App can order medicine from online pharmacy and get it delivered to the patient at home.

Medicine Reference Tool – Information about dosages, side effects, safety and their prescribing.

Health Tracking – Application can generate charts and statistics based on users' health data such as age, weight, glucose levels and blood pressure etc.

`

1.4 Mobile App Development

Usages of personal smart devices are increasing every day. Experts predict that in the near future mobile devices are destined to surpass desktop devices by a big margin. Major internet companies like Google have deployed strategies and policies to enforce the importance of development of mobile technologies and mobile friendly services. Such attempt is called Mobilegeddon [5] which activated very important changes to Google's search engine algorithm in order to prioritize mobile friendly services.

In the face of such developments companies and other financial or non-financial institutions have found themselves in need of developing multi-platform applications. Because just developing web applications to provide services to the public is no longer enough. Institutions have to consider many avenues of access that users can engage with content and services provided. Biggest platforms that majority of services provided are web platforms and mobile application platforms.

While mobile application platforms are distributed via proprietary market places (Google Play Store and Apple App Store), with major development of web technologies such as HTML5, CSS3, distribution of applications are being democratized.

In this chapter author is going to focus on the definition of the terminology of mobile application. Author shall discuss the most basic requirements and needs for development of mobile application. Author will also make an introduction about various approaches that are available for developing mobile applications as of now. Because new technologies are constantly emerging and old technologies are disappearing. It is really difficult to tell which platform will emerge as the best candidate as clear winner. So we shall discuss advantages and disadvantages for each development platform.

1.4.1 Definition of Mobile Application

The terminology of Mobile Application is meant to describe an executable program that developed to run on the mobile devices, such as smartphones, smart watches, pdas and tablets etc. While traditionally mobile applications are considered as isolated and small piece of software that are limited in functionality, in contrast to desktop grade software. But recently memory capacity, processing capacity and storage capacity started to rival desktop software. As a result we are seeing mobile software that are on par with desktop software. As means of distribution mobile applications are distributed via "App Store" while desktop applications are distributed via numerous ways.

`

1.4.2 User Interface Designation

Most important designation regarding user interface is considered to be ease of use. While desktop devices has physical keyboard and mouse controls , mobile devices mainly controller by finger gestures on touch screen which spans only few inches.

Because of limited screen real estate user interface design is really important requirement and demands careful considerations. There are several industry standard specifications that are specifically designed to help with such challenges. Most popular being “Material Design” [6] specification. Such design specifications require proper sizing, and adequate placement of elements in limited space that is available. Additionally various hardware vendors implement their proprietary versions of Human Interface Guidelines. Apples implementation [7] of these guidelines are good example for this. One of the most important expectations is having native functionality that are expected by average users due to established intuitive control schemes. Such controls are native keyboard, native icon collection and native menu. Most of the users are already familiar with certain style of controls that are usually identical among various platforms. Keeping such control elements familiar makes new application intuitive and less confusing.

1.4.3 Cost of Development

Cost of development can vary proportional to business and technology scope of the project. Careful consideration should be made by developers and project managers to estimate cost of development. In contrast to web and desktop applications, mobile applications tend to diverge by means of financing. Cost requirement also can include distribution and other costs in addition to development cost.

Due to user expectations and other market effects, cost of mobile application prices differ from traditional software. The figure 1.4 illustrates average price of an application in Apple App Store as of September 2018.

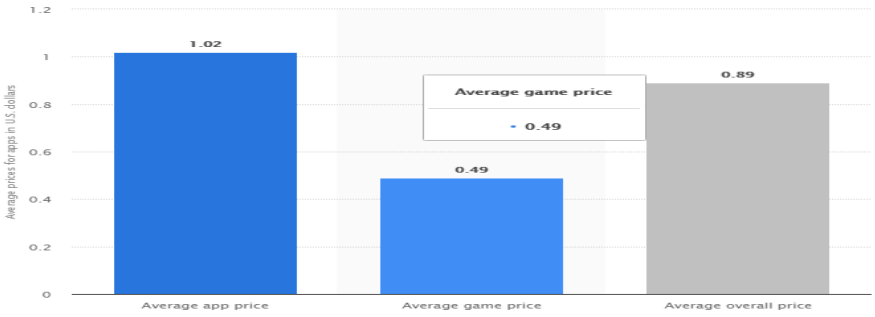


Fig1.4 Average prices for apps in the Apple App Store as of September 2018 (in U.S. dollars)

`

1.4.4 User Experience Requirements

Despite limited hardware parameters of mobile devices, users have come to expect smooth and fluid experience with mobile applications. This was not always the case. In early days of mobile devices, user interaction was slow and clunky. Apple managed to capture very big portion of user market by providing smooth user experience by innovative hardware design. It is important for the mobile application to be responsive so the user experience is comfortable.

1.5 Types of Mobile Application Development

Typically any modern data driven software application will consist of two main components. These components are :

- **Front-End (Client)**: Provides user interface elements such as forms, buttons, lists etc. These elements represent properties and actions that are operable in the server side of the application. Server side provides an API (Application Programing Interface) which application developer is responsible to implement in frontend app.

- **Back-End(Server)** : There are many kinds of architectural patterns available for implementing server side functionality. But as of today most popular server side architecture is considered to be RESTful services. Backend server is responsible to implement CRUD(Create,Read,Update,Delete) operations in tandem with business requirements of applications. Through this REST API frontend mobile app is able to perform data persistence operations.

We have to note that not all types of mobile applications require backend service. Applications which don't require persistence of data in remote server usually store all data locally to mobile device. Simple applications and some type of games are example to this.

Our main focus is on the front-end of the applications , especially on how to develop a mobile application that can be deployed on multiple platforms without requiring development for each platform separately.

Currently we have mainly three popular methods for developing Client side mobile applications. It is crucial to analyze and choose wisely a method that fits our various (Cost,Code Sharing, UI-UX etc.) requirements. Our choices are Native, Hybrid and Web application development. Each one of these approaches has pros and cons that will be discussed in detail in this document.

`

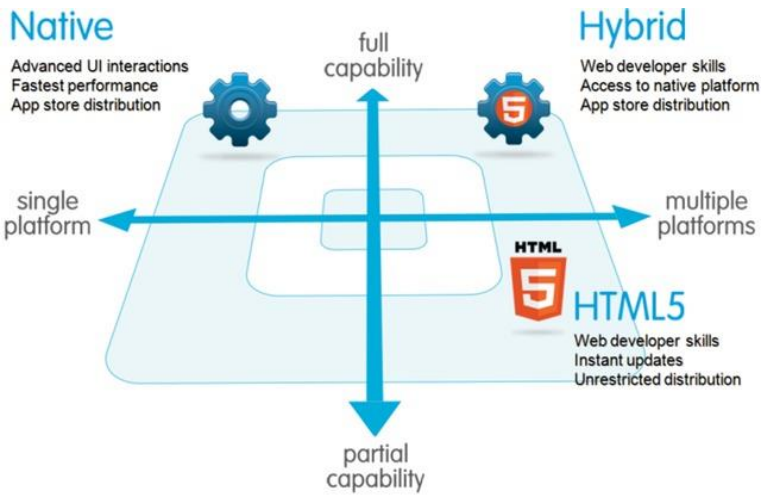


Fig1.5 Mobile app development platforms at a glance. [8]

1.5.1 Native Development

If we choose to develop an application natively we have to use tools provided by each platform in order to create native user experience that is expected by users. For Apple devices we have to use language like Objective-C or Swing and Xcode IDE. For Android devices we have to use languages like Java or Kotlin and Android Studio IDE. While we can make sure that resulting application will be performing at its best and will have the look and feel of the other native applications, we have to be mindful of costs of learning and mastering these languages and ides. Many companies are struggling to weigh the pros and cons of choosing this route. Many giants like Facebook[9] , Linkedin[10] and Airbnb[11] started their mobile apps as pure web apps or hybrid apps. But eventually as their user base and apps grew they were forced to come back to native apps. As native apps provided unparalleled performance compared to other options.

Pros

- Optimized User Interface - Specific design guidelines for each platform allows optimal implementation of user interfaces
- High Responsiveness and Smoothness - In comparison to web and hybrid apps native apps are substantially more fluid and responsive resulting in optimal user experience.
- Internet Connectivity Not Essential - Native applications are built in mind that they don’t always have full connectivity to the internet. So native applications can handle no internet at all or occasional breaking of connection. And they still provide acceptable user experience by caching most recent user data.

`

Cons

- Isolated Development - We have to develop for each platform using specific languages and tools that are provided by that platform
- High Cost of Development - By having to support multiple platforms , it is guaranteed that development costs will shoot up as we have to hire more developers and assets.
- Maintenance Complications - Native applications can only be distributed via specific market places such as Apple Appstore and Google Play. If users decide not to upgrade we will end up with fragmented user base.

1.5.2 Web Development

Many websites and web applications are developed today are built mobile devices in mind. Responsive Design [11] guidelines demand that web applications should use "Mobile First" approach when starting a new project. CSS has capability named "Media Queries" which allow serving different styles for different screen sizes. In recent years Google announced their "Progressive Web Apps" [12] specification which enables many native app like features on the browser. These capabilities enable us to create web applications that can more or less compete with traditional mobile development methods.

To develop a web application HTML, CSS and JavaScript is used. And it is served on web browsers. Web browsers can be a source of many problems that hold web apps back from conquering this whole space. But as time passes we have seen that web platform matured a lot.

Pros

- Single Application For All Platforms - We don't have to maintain different versions of our application for every supported platform. We have a single codebase that is served equally on all platforms
- Development Costs Are Lower - Learning curve for web development skills are much lower. And numbers of web developers are in abundance. Web developers can translate their skills to mobile development very easily.
- Deployment Of Application Is Simplified - We have to maintain only one codebase that is distributed to all platforms. It is not necessary to wait for app updates as all updates are loaded on the next request for web page.

`

Cons

- Unoptimized User Interface - Web applications user interfaces are not necessarily optimized to a specific device or platform. We don't have the ability to reuse ui components from native platforms. We are restricted to HTML and CSS ui elements.
- Mediocre Fluidity and Responsiveness - Web applications can't provide same interaction as native devices. Because native devices execute their code close to hardware. Web apps can only execute at the web browser level. Although this ramifications are going to disappear with the invention and popularity of Web Assembly [13].
- Constant Internet Connection - Web applications usually served by online web servers. We can't launch web applications without access to the web server. Although new technologies are developing to remedy this disadvantage, namely Progressive Web Apps.

1.5.3 Hybrid Development

As Web technologies continue to mature , application developers decided to take the best from both worlds and create hybrid applications that have ease of development of web applications and can provide functionality like native applications. By combining these two methods, we can start thinking about how to avoid the drawbacks associated with the previously mentioned methods.

Hybrid applications lifecycle is nearly identical to native apps as far as users point of view concerned. User can find it from app store, easily install it and update it automatically as usual.

As far as developers are concerned, application development is done using web technologies (HTML,CSS,JavaScript). Resulting set of files are served inside a native wrapper which uses webview browser to display the application. When applications are started native wrapper starts a WebView program, which is a browser without an address bar and maximized to full screen. Having a browser as an application container allows us to deploy the same app on multiple platforms without a concern of compatibility thus reducing the development costs.

Hybrid development doesn't alleviate us from maintaining a code base for different platforms. We still have to keep a separate copy of application for every supported platform. But we can reuse web code (HTML,CSS,Javascript) among this versions. Tools

`

like phonegap and cordova [15] can help us to automate this maintenance task to some degree.

1.5.4 Comparison of different methods

There are several main characteristics shared by all methods of application development. The table 1.1 provides a brief comparison of how well the methods perform at each specific point.

Features	Web	Hybrid	Native
User Interface	Average	Good	Very Good
User Experience	Average	Average	Very Good
Development Time	Low	Average	High
Costs	Low	Average	High
Deployment	Simple	Complicated	Complicated
Maintenance	Variable	Variable	Variable
Offline Mode	No	Yes	Yes

Table 1.1 Comparison of different development methodologies

1.5.5 Choosing a development framework for client side app

Ionic framework is chosen for realizing this application because of the following reasons

- Authors familiarity with web development (html,css,javascript) and Ionic framework itself.
- Develop once run everywhere philosophy
- Best of both worlds (Ease of web development and ability to access device hardware with native APIs)
- Tooling available for Ionic development (Webpack compiler,Typescript, Hot Reload etc.)

What is Ionic Framework?

Ionic Framework [16] is an open source UI library that allows building multiplatform applications with web technology (HTML, CSS, and JavaScript).

`

Chapter 2 - Security

Client server infrastructure is defacto architecture for developing any kind industry scale software. Applications that are developed to utilize data connectivity and transport are composed of two main components. Client and Server side. For healthcare and medical applications it is imperative to assure data security on both ends. We have to consider security considerations of mobile application development separately. Because there are separate challenges that are presented on both sides. Prior research shows that there are many considerations needs to be assessed for both client and server side security [17, 18].

2.1 Client Side Security

Healthcare and medical applications pose as client side component in our architecture. Mobile applications for healthcare and medical are growing in importance for both patients and healthcare professionals. There are diverse categories of applications available on the market, ranging from displaying important information (ie drug interactions) to helping keeping track of health condition. These applications store and transport sensitive information. Weaknesses in transport security in these applications can lead to safety issues regarding data confidentiality and integrity.

With the popularity of smartphones, abundant Internet access and the app ecosystems around, health information technology also found its way to mobile devices. Mobile health (mHealth) [19] describes using mobile devices to organize medical or health-related purposes. mHealth apps also give patients the ability to keep track of their medical data .

Because smartphones are used for a diverse set of medical apps, they have a more careful need for protection [20]. Mobile device vendors are required to ship regular updates in order to provide security for the latest threats. This represents an issue especially for low-cost Android-based devices [21]. Besides device security, data transport security is also a very important consideration.
In European Union privacy regulations have been proposed for handling security concerns regarding data handling by application providers [22]. European Commission have provided The Privacy Code of Conduct for mobile and healthcare applications which highlights very important points for heightened security requirements [23].

`

2.2 Transport Security

Communication to a server is required in order to provide information or to facilitate transmission of medical data to a service provider. When data are sent using public network, data can potentially be observed, modified, or redirected.

The transport layer security (TLS) is the protocol that plays the biggest role for securing internet connection. It was engineered to give remedy against the above mentioned problems. TLS offers authentication, data integrity, and confidentiality through asymmetric and symmetric cryptography. But TLS is not without its deficiencies. A lot of weaknesses have been discovered in this protocol suit.

- Padding Oracle On Downgraded Legacy Encryption (POODLE) [24]
- Browser Exploit Against SSL and TLS
- Factoring RSA Export Keys
- Heartbleed [25, 26] is an implementation weakness

Privacy and security of users can be compromised if older protocol versions or deprecated implementations are used during development. We should bear in mind Open Web Application Security Project Mobile Top 10 [27] which points out the most important considerations when developing a secure mobile application.

User and server data can be easily revealed from mobile devices. Web browsers have a way to identify if a website is insecure and warning users with notifications about certificate validations. Mobile applications don't have this kind of validation mechanism. And applications don't automatically issue warning the user about invalid certification or missing of encryption. We have to be very mindful of such shortcomings while considering transport security for mobile applications.

HTTP (Hypertext Transfer Protocol) is a primary way to conduct client server communication in mobile applications. It is an application layer protocol and can be used on top of a secure TLS connection. TLS and its predecessor SSL are engineered to ensure confidentiality (encryption), integrity, and authenticity between the client and server sides that involved in the transmission.
During initial handshake and key exchange asymmetric cryptography and PKI(Public Key Infrastructure) is used. After initial connection establishment symmetric cryptography techniques are used to fast and securely transmit data. Version of the transport security protocol is very important as there are various security issues in older TLS and SSL versions. This makes critical to check for protocol version used in application.

TLS implementation by default in a mobile operating system trusts a pre-installed root public certificates from certificate authorities. When an application makes a secure connection to a server, this server is responsible to authenticate itself with a valid certificate. The TLS client on the mobile device is responsible to assure that this certificate

`

was obtained from one of its trusted certificates. Application developers don't have control over which trusted certificates are pre-installed on the device. This opens possibility that device might be compromised with malicious compromised certificates.This enables an attacker to act as a middle man (man-in-the-middle) between the client and the server. No need to say that from this point onward users data confidentiality and integrity is breached.

2.3 Server Side Security

The functionality of many data driven applications depends on communication with a remote server over the internet. In the context of mobile applications HTTP is the standard for client server communication. By default HTTP offers very limited security features and data is sent in clear text. The communication with public internet can potentially be modified, observed, or redirected. .This endangers the integrity of data displayed by an app and confidentiality of the data sent to and received from a server and also could enable a malicious party to impersonate a server. Furthermore, a publicly reachable server must also guarantee availability.
The medium between client and server is mostly considered to be untrusted infrastructure. Because any third party device is in a privileged position between both communication peers can read and modify all data transmitted. These devices are routers, switches and other kinds of gateways. A common attack technique is called address resolution protocol (ARP) spoofing. With these method attackers is able to receive all requests that are destined to a router on a local network. Man in the middle attack can be performed once attacker can intercept packets.

Confidentiality and integrity of data sent through an untrusted medium can be protected by using the Transport Layer Security. TLS protects authenticity, integrity, and confidentiality of the communication channel. TLS offers secure authentication, data integrity protection, and confidentiality using asymmetric and symmetric cryptography. TLS is used to provide security for HTTP. Once HTTP is encapsulated in a TLS packet it is called HTTPS connection.
Furthermore TLS can offer forward secrecy [28].Such as, if secret keys are compromised in the future , past data cannot be decrypted with broken credentials, thus past communication can stay secure.

TLS authenticates a server to the client by using a digital certificate. These certificates are issued and signed by a third party trusted entity; also called Certificate Authorities (CA).Certificates should have certain properties to be considered valid

- Issued for a valid domain
- Issued at a valid from date , before current data
- Issued for a valid until date in the future
- Certificate has not been revoked

`

CA's maintain a certificate revocation list, which can be checked to ensure that certificate has not been revoked. Crucial part of server security maintenance is to manage and secure server certificates.

2.4 Database Security

Securing data in the database is achieved via encryption. Nowadays database encryption should be a standard practice regardless of how sensitive is the data that is stored in the database, especially for medical applications. Loss of customer data can be very embarrassing and can affect customer confidence.

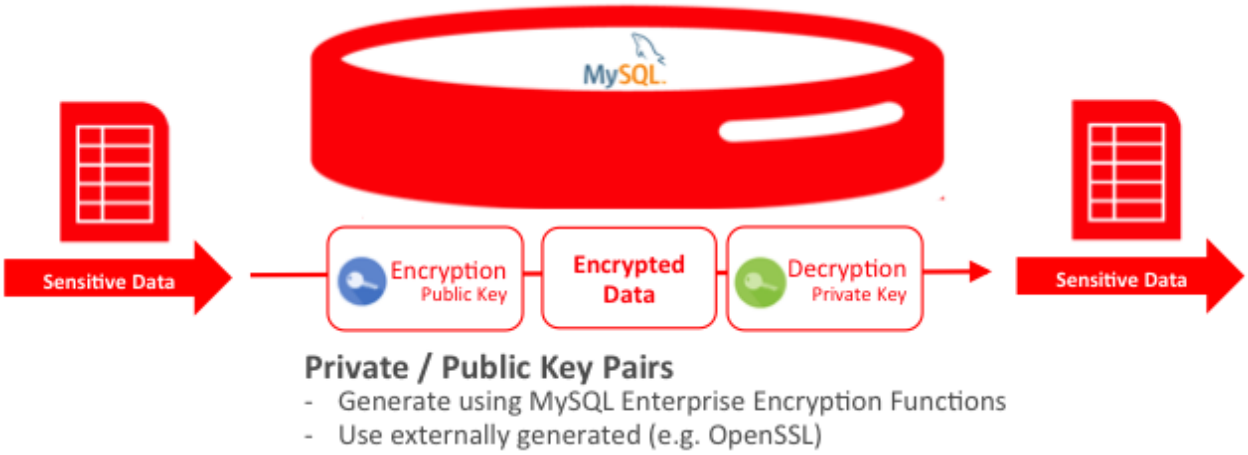


Fig2.1 MySQL functionality for asymmetric encryption [29].

To implement encryption one can use Mysql built in AES_ENCRYPT() and AES_DECRYPT() functions [30]. These functions use official AES (Advanced Encryption Standard) algorithm. AES uses 128 bit keys, which is considered fast and secure enough for most cases. Of course we could increase the key length to 256, 512, 1024 etc. but one have to consider tradeoff between speed and security.

`

Chapter 3 – WebRTC and Supporting Protocols

To achieve the goal of this thesis author has to rely on few fundamental underlying technologies that enable us to establish connections between peers, to serve necessary files, handle call session establishment signals etc. Fortunately, a number of standardized web technologies have matured in the last decade to allow us to build such a service in a fairly straightforward manner. In this chapter author will overview the brief summary of these technologies.

3.1 WebRTC

WebRTC (Web Real Time Communication) is a standard specification that allows real time peer to peer communication between web capable devices (mainly browsers). Peer to peer connection using TLS makes WebRTC naturally secure. That is why it was chosen for this project.Second advantage is that, we don't need a third party plugin to achieve this capability. WebRTC was created to enable highly effective RTC solutions to be developed for browser, mobile devices and IOT devices. Thus allowing them to communicate via common set of protocols [31].

WebRTC was initially developed by Google and it was open sourced in 2011. Upon its well acceptance, IETF and W3C started an initiative to standardize it in order to enable development of browsers that can handle common set of protocols to work across different devices. This process is still ongoing. But we can say that over the years interest for WebRTC has been growing and browser and device vendors slowly but surely catching up with these common standards. Google Chrome and Mozilla Firefox are main flag bearers in this effort. But over the time other popular vendors such as Microsoft Edge and Apple Safari caught up with them. Majority of features that were envisioned by W3C proposals are supported across many vendors and devices [32].
It is expected that by the year 2025 WebRTC market will worth US$81 billion, thus showing the increasing demand and popularity for this technology [33]. WebRTC presents us an opportunity for web innovations that will rely upon web browsers for variety of functionalities which were not possible previously without specialized software and plugins. Some of the features are real time audio and video chat, file sharing without intermediate server and multiplayer games exchanging data in peer to peer fashion. Further more interesting use cases will be discussed in upcoming chapters.

One of the main drawbacks that held back development of WebRTC is browser interoperability. Although over time API is almost fully standardized most real world implementations rely on compatibility plugin to address the fact that every web browser implemented these standards in slightly different manner. Google release adapter.js library to remedy this problem and abstract away api changes across different browsers [34].

`

Another issue that WebRTC faces is, for a long time there hasn't been a consensus on which set of audio and video codes will be used by WebRTC. Agreeing upon common set of audio and video codecs is very important otherwise it would be very difficult to maintain compatibility among different browser vendors. Two main proposals were around Google's VP8 and AVC H.264 which eventually were marked as required. For audio codecs OPUS and G.711 were decided to be mandatory [35].

3.2 WebRTC Use Cases

The fundamental use case for WebRTC, by its nature is in the field of audio/video communication and teleconferencing. One of the first adopters of WebRTC, Mozilla integrated WebRTC in the Firefox browser as an embedded application and it was called Firefox Hello. It showcased features of WebRTC without needing additional plugins. Firefox Hello was discontinued later on [36].
Zingaya is a click-to-call service for use on websites that allows visitors to video chat with a representative/support person thus eliminating the need for a telephone call [37].
WebRTC has also been utilized in software development and collaborative language learning as shown in [38] and [39].

3.3 WebRTC APIs

In order to realize high quality RTC applications complex set of functionality is required. Luckily for us, modern browsers managed to abstract away most of this complexity behind simple APIs. WebRTC consists of three main API components:

- MediaStream : gathering of audio/video streams
- RTCPeerConnection: receive and transmission of audio/video data
- RTCDataChannel: receive and transmission of arbitrary application data

Behind this APIs we can discover a lot of layers of protocols that supports this functionality, which consists of signaling, peer discovery connection negotiation, security etc. Author will try to summarize these underlying protocols in the following sections.

3.3.1 WebRTC MediaStream API

MediaStream API is one of the main components of WebRTC which is responsible for providing streams of data from various sources such as:

- Video stream from user webcam
- Audio from microphone
- Screen capture stream from user screen
- Data stream from canvas element etc.

According to W3C specification WebRTC defines a set of JavaScript APIs that allow a web application to request audio and video streams from the device. Additional APIs are

`

provided to process and manipulate acquired media streams. **MediaStream** object is the interface that allows this functionality.

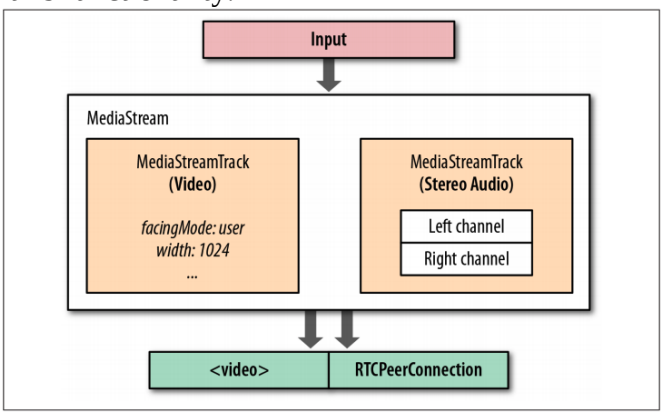


Fig 3.1 Synchronized tracks are carried in MediaStream object [40]

MediaStream object can contain one or more media tracks (audio and/or video). Tracks are synchronized. The inputs are usually sourced from device webcam and microphone. But optionally can be a file from hard disk or network storage as well. The output of MediaStream can be attached to a html video element or it can be attached to RTCPeerConnection to be sent to a remote peer.

Because various devices can support a wide range of video resolutions and audio codecs, MediaStream object requires a **constraint** configuration. This configuration specifies mandatory and/or optional settings such as video resolution.

```
const constraints = {
        audio: true,
        video: {
           mandatory: {
              width: { min: 320 },
              height: { min: 180 }
           },
           optional: [
              { width: { max: 1280 }},
              { frameRate: 30 },
              { facingMode: "user" }
           ]
        }
     }
```

In this example configuration we are requesting at least 320x180 resolution for video stream and audio stream is enabled with a default constraint. Optionally 1280px wide video

`

stream can be provided if available. We are also requesting user facing camera as primary video source.

Once the streams are acquired, we can further manipulate and process them.

- We can process the audio with Web Audio API
- We can capture and post-process individual frames with Canvas API
- We can apply various effects and filters with CSS3 and WebGL

MediaStream API describes [41]:

- Streams of audio video data
- Methods to work with audio video data streams
- Constraints that are applied to audio video data streams
- Success and error callbacks that are associated with such situations
- Various events that are triggered during these processes

We can use MediaDevice.getUserMedia() method which prompts browser user to give permission to use audio video sources such as webcam or any other usable source. Screen capture is also subject to user permission.

```
const constraints = {audio: true, video: true}
navigator.mediaDevices.getUserMedia(constraints);
```

`

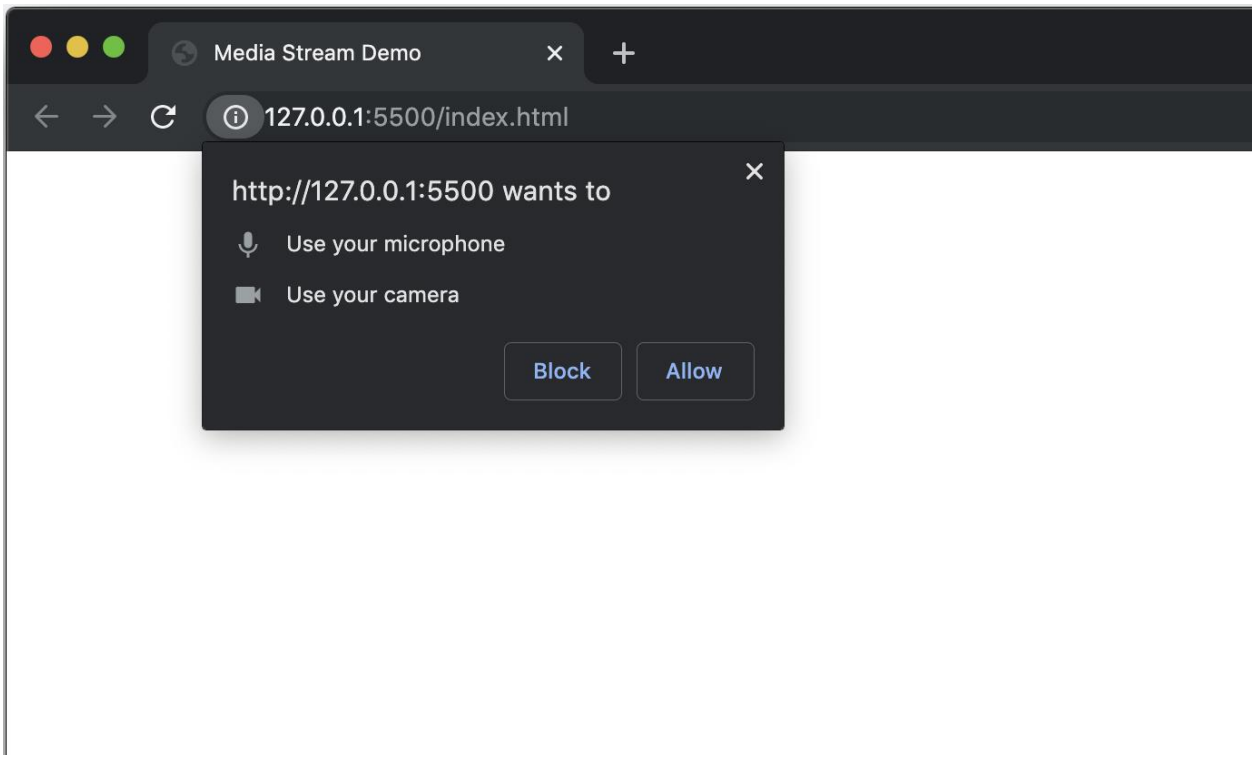


Fig 3.2 Chrome browser prompts user for permission to stream audio and video

Similarly ,MediaStream API has MediaDevice.getDisplayMedia() method to prompt for screen capture data. Appeared user interface allows us to choose the whole screen, individual application windows or specific tabs of the browser, which allows granular control of what to show to remote peers and what not to expose.

```
navigator.mediaDevices.getDisplayMedia();
```

`

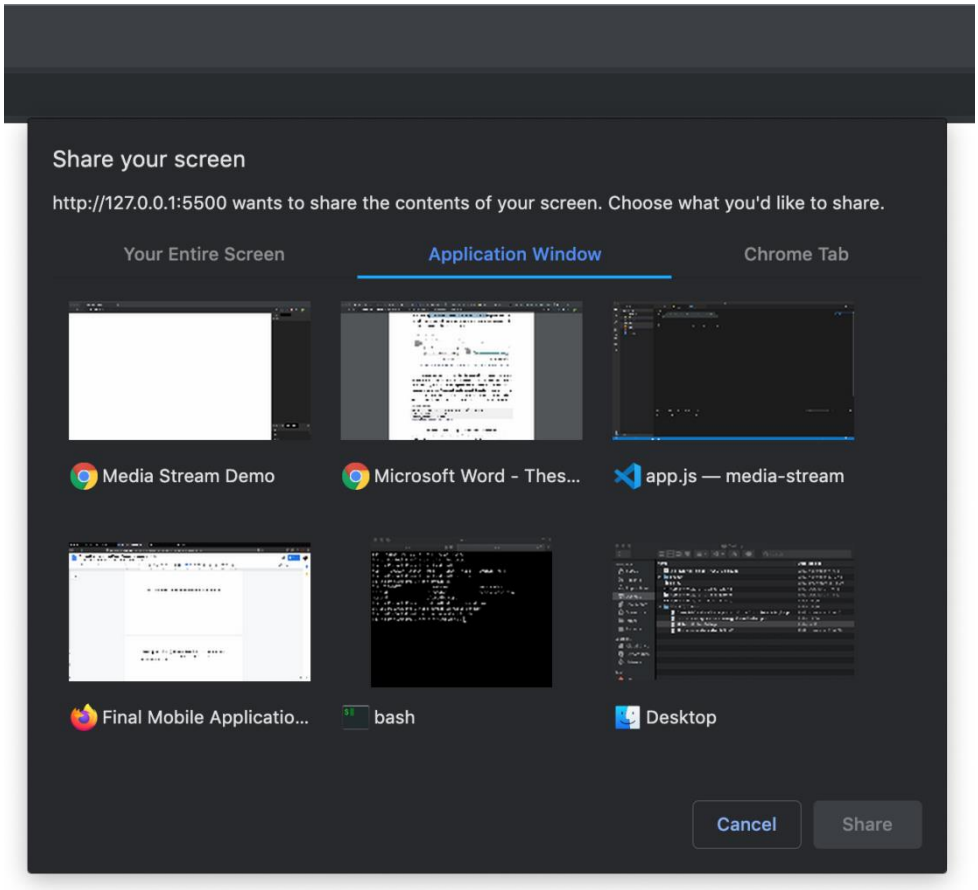


Fig 3.3 Chrome browser prompts user for permission to choose screen sharing option

These methods return Javascript Promise object.If permission is granted by the user, Promise object is resolved to a MediaStream object. Otherwise if permission is denied or media is not available , then the promise object is rejected with PermissionDeniedError or NotFoundError respectively.

3.3.2 WebRTC Audio and Video Engine

Using WebRTC APIs we are able to tap into the resources of local hardware and capture audio and video streams. But simply transmitting raw captured streams is not acceptable. Therefore, each stream must be processed in order to optimize it for changing conditions of network availability. For example, if bandwidth between two peers is reduced, WebRTC is able to negotiate new settings and reduce the bitrate of transmission. On the remote side of connection, remote peer must be able to decode the stream, and adjust to jitter and delay that may be introduced on the traversing network path. This is no easy task to implement from scratch, but WebRTC comes with fully featured video and audio engine that is capable of signal processing , bitrate adjustment and so on. Processing is done right in the

`

browser. But more critically the browser is capable to dynamically adjust its processing routines in order to react to continuously changing parameters of video and audio streams and network conditions. As a result of optimized processing web application receives the best media stream that is available. Which can be rendered in HTML video element , forwarded to remote peer or can be further processed with CSS etc.

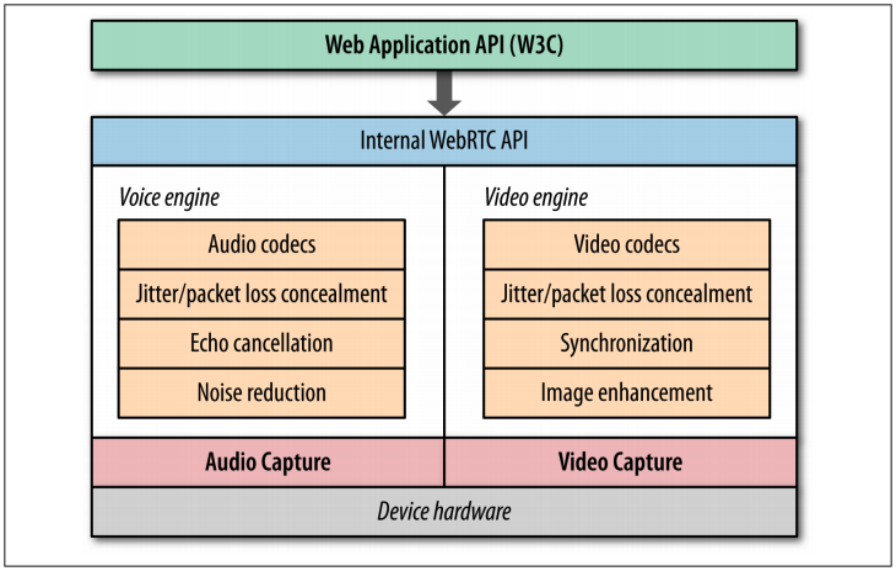


Fig 3.4 WebRTC Audio Video Engine Structure [40]

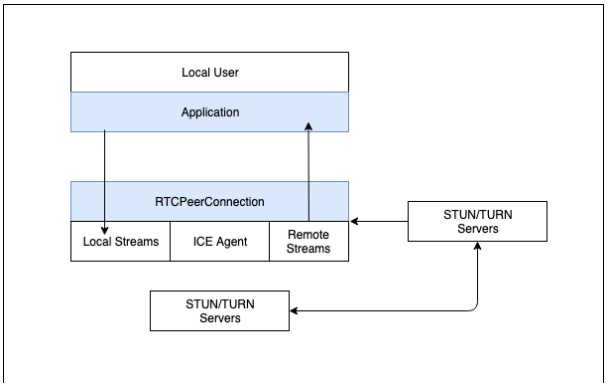


Fig 3.5 RTCPeerConnection Components[40]

3.3.3 WebRTC Network Transport Structure

Because of real time nature of WebRTC, it uses UDP at the transport layer. Main advantage and disadvantage of UDP is that there is no reliability and ordering of data. UDP doesn't

`

have all necessary features to facilitate WebRTC transmissions. That is why we need layers of supporting protocols on top of UDP in order to realize acceptable communication.

ICE, STUN and TURN protocols are used to establish and maintain peer to peer connection over UDP. WebRTC specification emphasizes that encryption is required. DTLS is used on top of UDP to achieve this. DTLS is used to transmit control data securely between peers. All other data is transmitted with help of SRTP and SCTP. SCTP and SRTP add such functionalities as

- Stream multiplexing
- Reliable delivery
- Congestion and Flow control

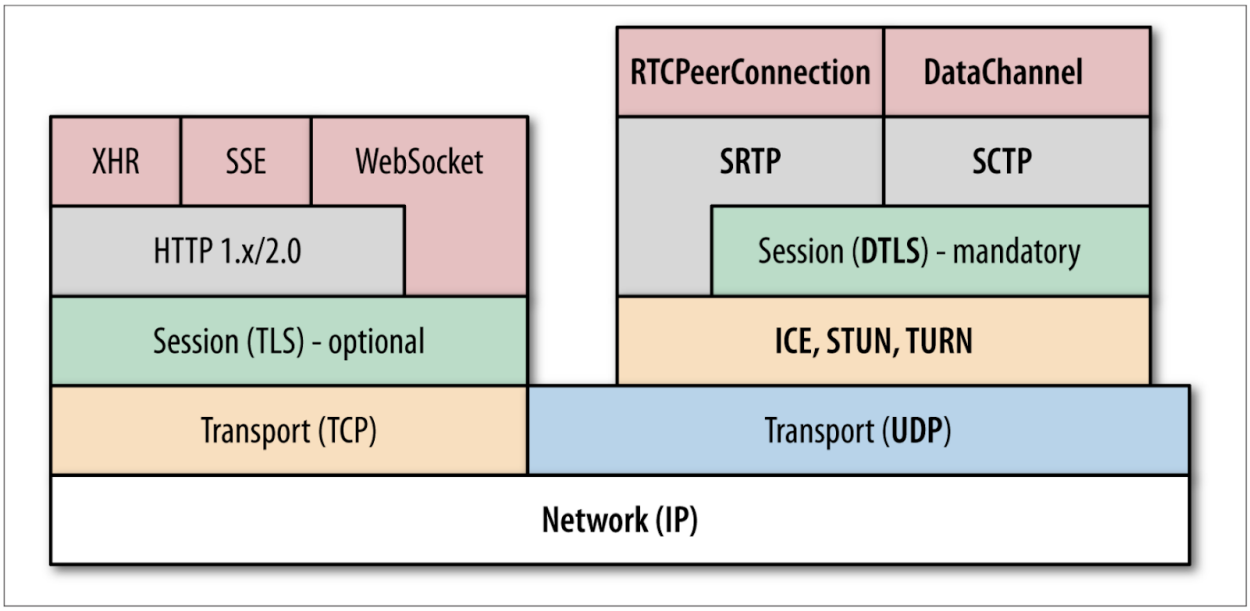


Fig 3.6 WebRTC Network Protocol Stack [40]

`

3.3.4 WebRTC Peer Connection API

RTCPeerConnection is the interface that JavaScript API exposes. Despite the complexity of network layer, it is relatively simple to work with. RTCPeerConnection object is responsible with maintaining lifecycle of peer to peer WebRTC connection.

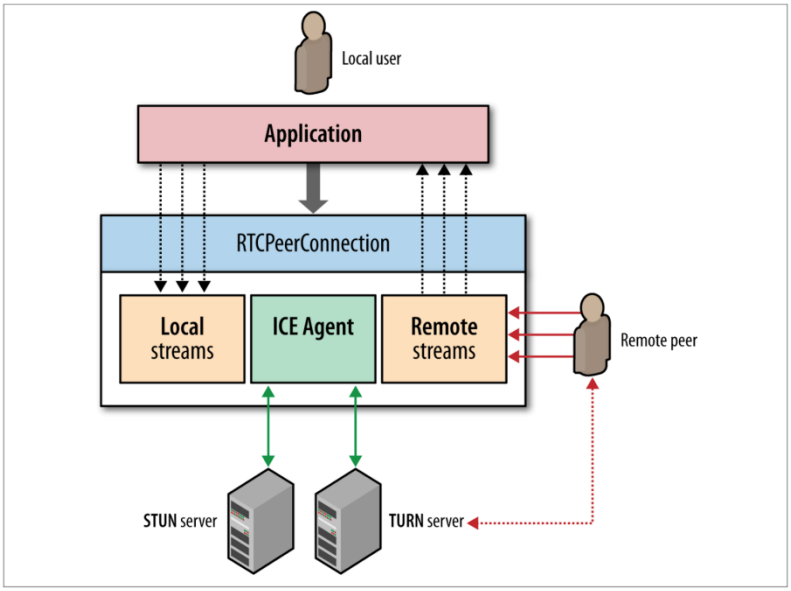


Fig 3.7 RTCPeerConnection API [40]

WebRTC RTCPeerConnection API performs following functions

- Performs ICE negotiation for NAT traversal
- Maintains STUN network keep lives between peers
- Keeps track of local and remote streams
- Automatically renegotiates and reestablishes connection between peers
- Provides methods to generate offer and answer messages and monitor information current state of the connection

WebRTC specification defines RTCPeerConnection interface to represent a connection between two remote peers. It has algorithms and methods defined to handle efficient and effective way of streaming real time data between them.
To be functional RTCPeerConnection object requires to be configured with ICE agent, signaling state and ICE connection state object.

3.3.5 WebRTC Peer Connection Establishment

Due to how the internet evolved over the past decades, IPv4 addresses are scarce and not every device on the internet can have unique IP address. Most of internet enabled devices

`

remain behind one more NAT devices and share public IP address of their default gateway. As a result of this, establishing peer to peer connection between two remote devices is not easy. In contrast Client Server based connections are fairly straightforward. Because Servers' IP address is publicly available and client can send direct connection request to the Server. In a peer to peer scenario this is not always true. The remote peer might be offline or unreachable. We don't have sufficient information to establish direct connection. To solve this problem we have to consider the following points:
1. We have to somehow signal our intent to other side that it should start listening for our packets.
2. We have to identify numerous network routing paths between peers and relay this information.
3. We have to negotiate and exchange parameters of RTC connection, such as supported codecs , encodings etc.

WebRTC is able resolve routing paths between peers using ICE. WebRTC specification defines ICE protocol in order to collect all possible IP address and port combinations. And then ICE tests this information to find a working network path between two peers.

WebRTC isn't concerned with the problem of signaling. There are dozens of Signaling protocols available in the market. Some of the most popular on the internet are SIP, XMPP Jingle etc. Signaling solutions is left up to the application developer to choose. Any kind of signaling can be used with WebRTC. Some of the popular methods of signaling used with WebRTC are HTTP Ajax, Web sockets, SIP messages or even email. Signaling is just a text message that needs to be relayed back and forth between peers. In this thesis implementation author chooses to use Socket.IO software to solve this question.

3.3.6 Session Description Protocol (SDP)

First step of establishing WebRTC connection is to generate SDP offer message.
Following code snippet demonstrates how to initialize WebRTC session

```
// SocketIO used as signaling channel
const socketIO = new SocketIO();
// Initialize WebRTC Peer Connection
const peer = new RTCPeerConnection({});
// Request Audio Video Stream From Local Device
const stream=navigator.getUserMedia({audio:true,video:true});
// Generate SDP offer message and assign it as local    description
const offer = peer.createOffer();
peer.setLocalDescription(offer);
// Send the generated offer to remote peer for negotiation
socketIO.sendOffer(offer)
```

`

This is a generated SDP message snippet taken from console log message.

```
(... SDP snipppet ...)
m=audio 1 RTP/SAVPF 111 ...
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=candidate:1862263974 1 udp 2113937151 192.168.1.73 60834 typ host ...
a=mid:audio
a=rtpmap:123 opus/53000/2
a=fmtp:123 minptime=20
(... SDP snippet  ...)
```

SDP is defined in RFC 4568 as a simple text based protocol to describe properties of candidate session. Above snippet describes audio section of SDP. As a web application developers we don't have to be concerned much about processing SPD messages. RTCPeerConnection object defines methods to deal with inner workings of SDP using JSEP(Javascript Session Establishment Protocol). After we derive SPD description from RTCPeerConnection object, we can forward this information to remote peer as an SPD offer using a signaling channel of our choice, in our case using Socket.IO connection. To complete RTCPeerConnection lifecycle both peers have to exchange SPD descriptions that describes their audio video and other data streams. After remote peer receives our offer, it has to generate a similar message as an SPD answer and send it back. Both peers has to call setRemoteDescription() and setLocalDescription() methods with offer and answer messages as arguments. Once SPD information is exchanged between peers, second step is to establish network routing path. Next section describes how it is done.

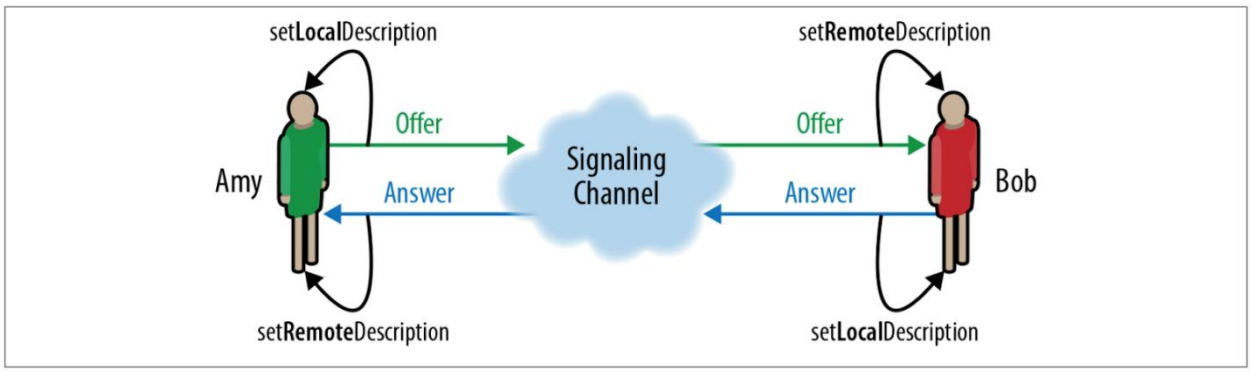


Fig 3.8 SDP Offer and Answer Exchange Lifecycle [40]

`

3.3.7 ICE (Interactive Connectivity Establishment)

As discussed earlier NAT poses several difficulties for establishing peer to peer connection between devices in a private networks that placed behind NAT devices. Luckily RTCPeerConnection object has ICE protocol agent that automates discovering most optimal network routing information between peers. ICE agent is responsible to gather local IP address port number combinations as candidates. Then, ICE agent performs connectivity checks between candidates. Once optimal peer is found ICE agent maintains the connection with keep alive messages. Once we call setLocalDescription() function is called on RTCPeerConnection object, local ICE agent starts discovering possible IP and Port combinations for local peer. ICE agent discovers local IP addresses by querying the OS. ICE agent queries configured STUN server to discover remote IP and Port configuration for remote peer. If STUN server fails TURN server is used as a last resort relay server to facilitate RTC connection. STUN protocol allows browser to discover its own public IP address and Port number information, similar to how we can use sites like whatismyip.com.

Whenever a new candidate IP address and Port number combination is discovered ICE agent triggers ‘icecandidate’ event on RTCPeerConnection. Same event is triggered when ICE gathering is completed.

```
peer.addEventListener('icecandidate',handleIceCandidate);
function handleIceCandidate(event) {
   if(event.target.iceGatheringState === 'complete') {
      const offer = await peerConnection.createOffer();
      socketIO.send(offer)
   }
}
```

Once SPD offer and answer messages exchanged between peers, second phase of connection can start. Now both sides of connection have the list of candidate IP address and Port numbers. ICE agent checks this list to find a connection that is reachable. Once working path found , ICE agent sends STUN binding message which remote has to respond with successful STUN response message. If this succeeds we have a working network routing path between peers. If it is failed agent has to fail back to TURN server as a relay server to establish a connection. Ice agent continuously sends keep alive STUN messages periodically. This process is mostly invisible to application developer and fully handled by RTCPeerConnection object. But it is useful to know about it in order to be able to troubleshoot possible connectivity issues.

`

3.3.8 Trickle ICE

There is one best practice used in modern WebRTC projects is to use trickling method for ICE information exchange. Traditional ICE gathering phase is time consuming because we have to wait for STUN queries to make a round trip around the internet and perform connectivity checks between remote peers. WebRTC extended ICE protocol with Trickle ICE that allows gathering of ICE information incrementally. SDP message is exchanged without ICE candidates. ICE candidates are sent with signaling channel as soon as they are discovered. ICE connectivity checks are run as new information is received. This process speeds up the connectivity process.

```
socketIO.on('icecandidatereceived', handleNewIceCandidate)
function handleNewIceCandidate(candidate) {
   peerConnection.addIceCandidate(candidate);
}
```

Trickle ICE method generates more traffic over the signaling channel. But for performance reasons it is a recommended strategy for all WebRTC applications. Idea is to send SDP offer message as soon as possible and trickle the ICE candidates as they are found out.
There is a possibility that ICE phase cannot be established. In this case we have to handle events that are generated by ICE agent. These events signal the status of ICE process.

iceGatheringState Object can be in three different states.
new: Object is in initial state
gathering: ICE agent is gathering local candidates
complete: ICE agent completed gathering local candidates

`

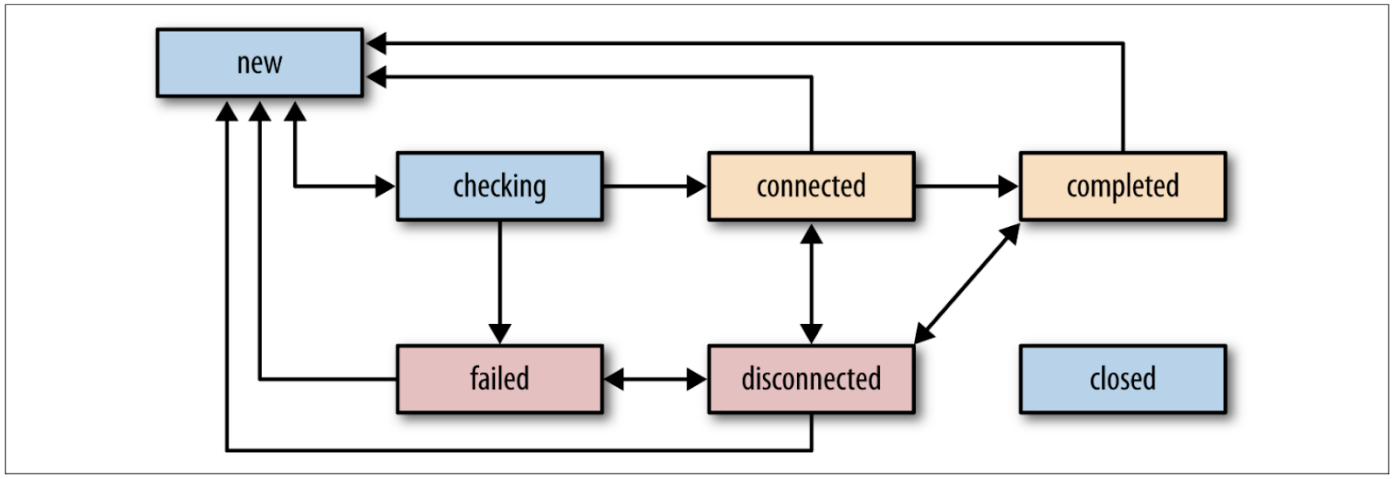


Fig 3.9 ICE Agent state machine diagram [40]

3.4 WebRTC Transport Layer Security

Once RTCPeerConnection opens initial connection, we have a raw UDP channel. Simple UDP is not sufficient for good performing RTC apps. Without congestion control, flow control and error checking the connection will perform very bad. In order to optimize our connection WebRTC adds few supporting protocols on top of UDP.

- **DTLS** (Datagram Transport Layer Security) main functionality is to negotiate secret keys for encrypting data and secure transport channel.
- **SRTP** (Secure Real-Time Transport) is the protocol that carries audio and video streams
- **SCTP** (Stream Control Transport Protocol) is used to transport application data, for example DataChannel data.

WebRTC can only work over encrypted channel. That is why HTTPS is required. Transferred audio video and arbitrary data must be encrypted. Over a TCP connection this is accomplished by Transport Layer Security (TLS). But in WebRTC case we only have UDP which doesn't support TLS. Instead DTLS is used which provides similar functionality as TLS. DTLS implements some features of TCP, such as handshake sequence and fragment offset etc. DTLS deals with packet loss by using timers. If a reply is not received within certain interval packets are retransmitted.
DTLS uses record sequence number, offset and retransmission timers to perform the handshake over UDP. At the end of the sequence both peers generate self-signed certificate and emulate normal TLS handshake.
A self-signed certificate doesn't require a certificate chain to verify. This means that DTLS can only guarantee encryption and integrity. Authentication should be provided at the application level.

`

3.4.1 WebRTC Transport Layer Protocols

WebRTC can acquire audio video stream at a high quality from the local device. But there is no guarantee that the quality of the stream can be preserved over the network. Because WebRTC is designed to work over the public internet. On the internet bandwidth is provided on a best effort basis. So WebRTC has no control over how the stream will end up on the other side. To combat this every connection is started at a low bitrate (<500Kpbs) and then quality is adjusted to match the available bandwidth.
Traditional media gateways and VoIP devices transmit audio and video streams using RTP (Real Time Transport Protocol) and RTCP (Real-time Transport Control Protocol). RTCP carries control information and statistics for RTP session. RTP is used to transmit the media stream itself. Because of security requirements WebRTC uses **secure** versions of this protocol(SRTP and SRTCP). Neither of these protocols provides mechanism to exchange encryption keys. In order to remedy this DTLS handshake sequence must be performed, and shared secret key must be exchanged between peers.

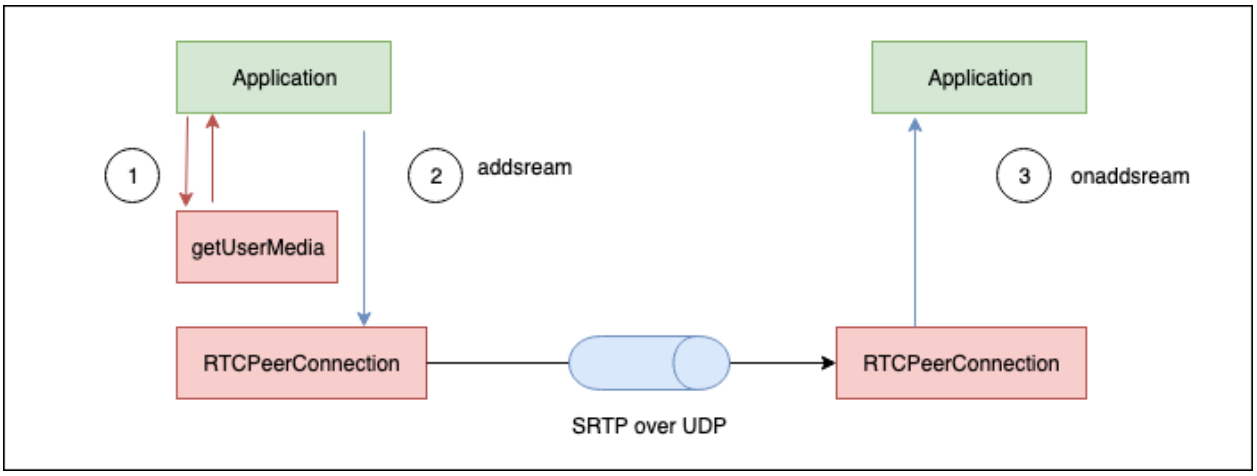


Fig3.10 Media Stream Transport via SRTP over UDP [40]

SRTP and SRTCP reuses these keys to encrypt the media stream traffic. To conclude, it requires more than just sending raw UDP packets to achieve good quality. Luckily we don't have to deal with complexities of low level protocols. WebRTC implementation deals with underlying complexity for us.

Using WebRTC DataChannel we can transmit any arbitrary data. SRTP protocol is not good enough for this purpose. SCTP (Stream Control Transmission Protocol) is used instead. SCTP runs on top of DTLS tunnel as well.
DataChannel API requires that underlying transport must support multiplexing multiple data channels. Each channel must support in and out of order delivery. Each channel must support reliable and unreliable delivery. Message oriented API is requires as, each

`

application packet might be fragmented and reconstructed by the channel. Congestion control and flow control must be implemented. Confidentiality and integrity must be provided.
SCTP protocol satisfies all these points. SCTP is a popular protocol for service carriers, similar to TCP and UDP. In case of WebRTC usage, SCTP runs over DTLS tunnel which also runs on top over UDP.SCTP combines the best features of TCP and UDP. It is a message oriented , configurable protocol which has built in flow and congestion control functionality.

	TCP	UDP	SCTP
Reliability	reliable	unreliable	configurable
Delivery	ordered	unordered	configurable
Transmission	byte-oriented	message-oriented	message-oriented
Flow Control	yes	no	yes
Congestion Control	yes	no	yes

Table 3.1 TCP vs UDP vs SCTP Comparison

3.5 WebRTC Signaling Channel with SocketIO

Although WebRTC main functionality is to enable peer to peer connection between two users without a relay server, initial exchange of signaling information is required. Signaling information includes the following:

- IP address and port number
- Media stream metadata such as codecs, bandwidth, video resolution constraints, etc.
- Chat room creation, joining

`

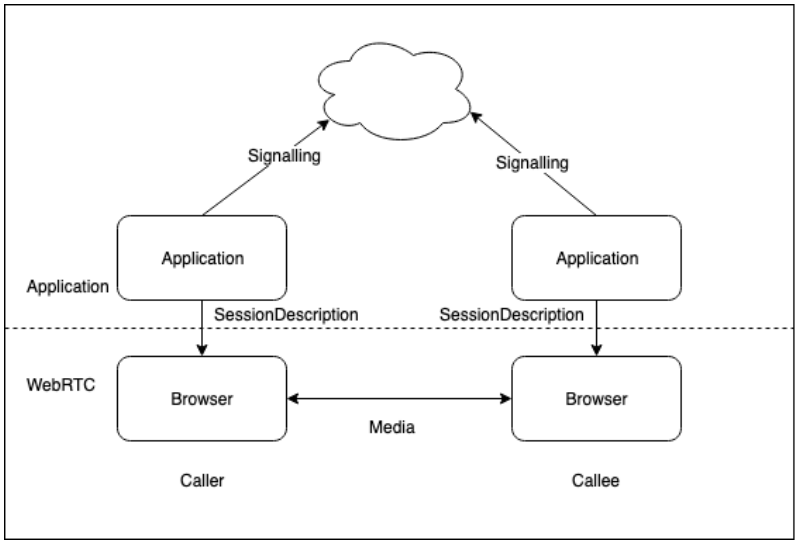


Fig 3.11 Signaling Channel Architecture

WebRTC signaling is based on new specification called JSEP(JavaScript Session Establishment Protocol). JSEP provides functionality to exchange offer and answer messages using SPD.

SocketIO is a popular choice for implementing signaling channels for WebRTC. It is a JavaScript library for developing real time applications that enables two directional communication based on events model. It consists of two components, the client side library that is deployed to a browser and server side library which runs on Node.js server and listens for incoming connections from browser.

Node.js is the server side implementation of JavaScript that allows javascript code to run on the servers. It consists of Google's V8 engine wrapped with C++ code. Nodejs uses event driven single threaded model that enables fast and efficient concurrent requests. Nodejs includes a HTTP server. Its single threaded event loop management model facilitates non-blocking I/O operations.

SocketIO utilizes web socket technology as a first choice. But it is capable to fall back to older technologies. For example, if HTTP session is not able to be upgraded to Web socket, it can revert to HTTP long polling as a fallback mechanism.

`

Chapter 4 – Project Implementation

Project consists of three separate components.

- Client side App (Mobile and Web App)
- Web REST API
- Signaling Server

4.1 Client Side Application

Traditional web applications use server side generated HTML pages to provide the content to the client. This means every time that something needs to be updated on the client side, server needs to re render html page with updated data and send it back to the client, so that client browser can display updated information. As one can imagine this approach is not efficient in many ways. It generates too many redundant network requests. Every time browser needs to refresh the page in order to get the latest information.

In contrast, modern web applications download main functionality of the web application in the form of HTML, JavaScript, CSS bundle once. Every time some information is needed, web application can send AJAX requests to the web server. This eliminates the need to refresh the web page. Also data loading speed is improved significantly as we don't need to reload the same page over and over again. Information is exchanged with micro HTTP web requests such as GET, POST, PUT, etc.

Because of this, we are generating web application and mobile application using Ionic framework. We can take advantage of this. Ionic framework is based on html.

Client side app provides following functions

- User interface to register and login to application
- User interface for searching doctors based on various parameters
- Pages for listing doctors and viewing doctor profiles
- User interface for showing calendar and available booking timeslots
- User interface for booking time with doctors
- User interface to realize online audio video consultation with doctor

`

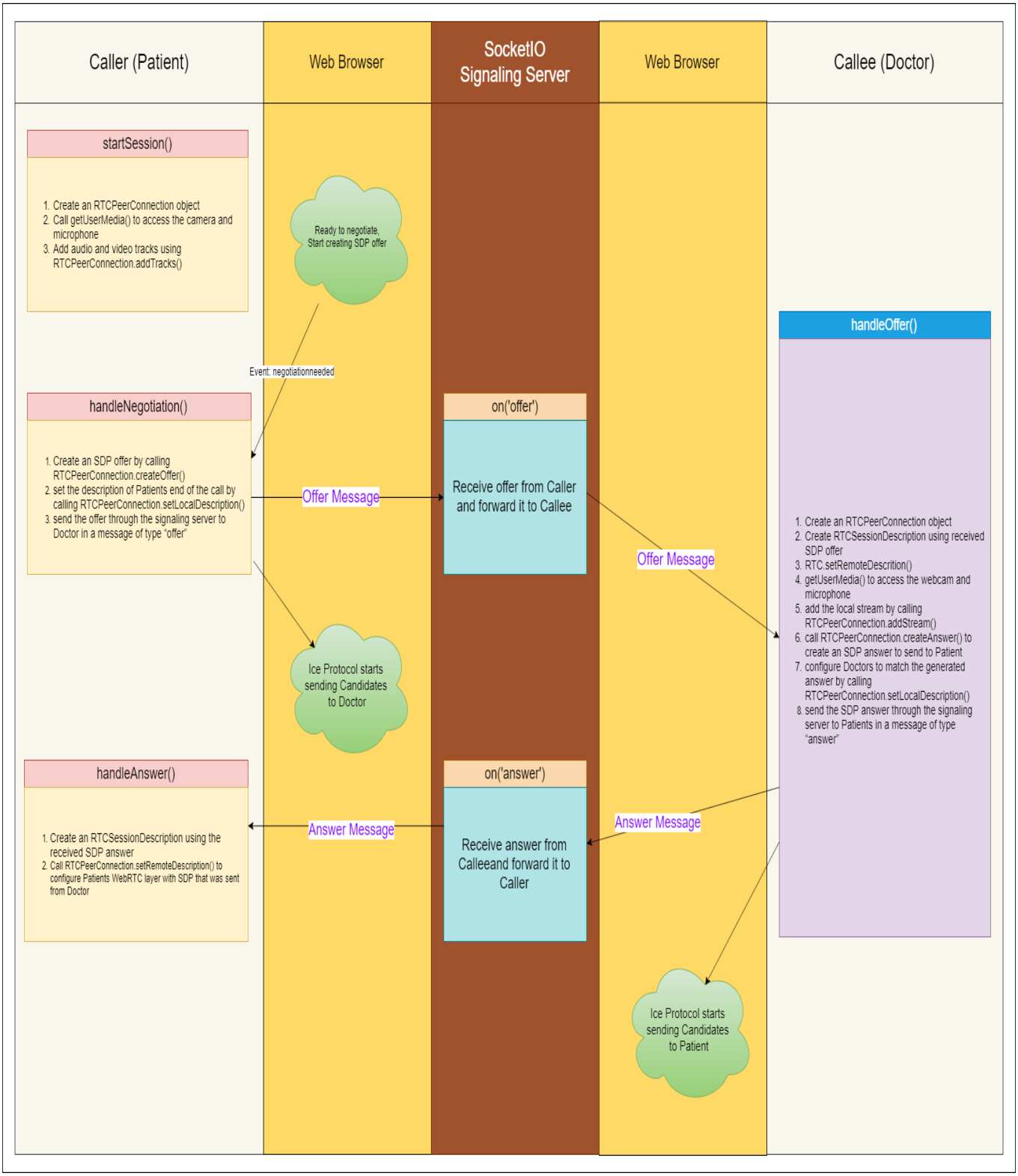


Fig 4.0 WebRTC Session Sequence Diagram

`

4.1.1 Client Side User Interface

For this thesis application author's aim is to produce a web application that can be used in normal browsers and also produce a mobile application that can be deployed to mobile devices, such as mobile phones and tablets. To achieve this goal **Ionic Framework** was chosen for the reasons stated in an earlier chapter. But the main reason is Ionic allows us to develop a single codebase in order to derive an application for various platforms. Using Ionic we can generate a standard html web app. We can also compile source code to generate Android app or iPhone app. We can even produce a desktop app if needed. Main idea is, we can wrap html code with any platforms web view container and be able to run it as native application.

Name of the application is **Telemedic**.

In order to use the application user has two options.

- User can open https://telemedic.herokuapp.com in a browser
- User can install apk file for mobile application (Currently only Android app available)

Next step, user has to register or login with the system

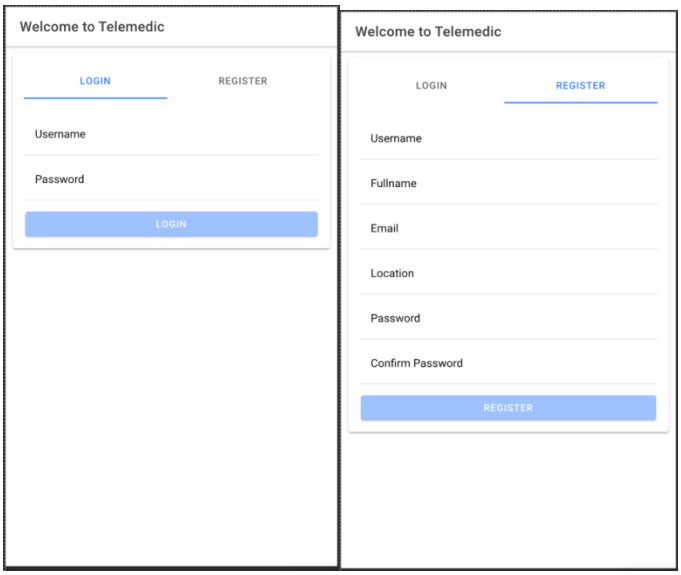


Fig 4.1 Login and Registration Screen

Once user is registered he/she is automatically logged in to the application. If the user is already has an account, then he/she can login using the login tab. Application authenticates the user by sending username and password to the authentication API of the web server. If the user is authenticated, the web server responds with an authentication token and user is redirected to the main tab, otherwise "401 Unauthorized" error is generated.

`

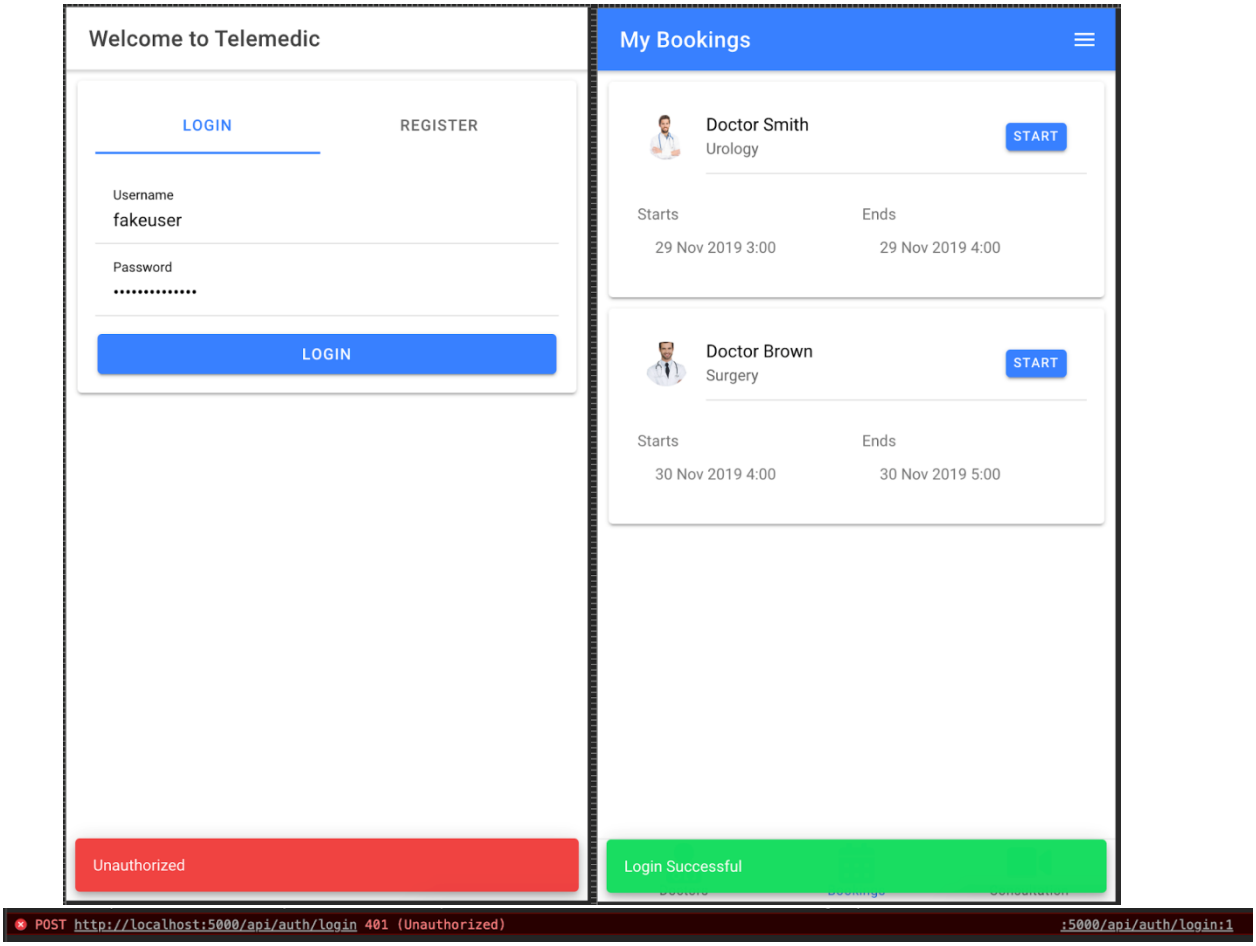


Fig 4.2 Login failed state and login successful state

Using this authentication token, user is able to make further requests to the web server, such as requesting list of doctors or booking an appointment with a doctor.
Beside authentication information token also contains some information about user itself. This information is used to populate user interface, such as user id and fullname.

- exp : Expire date of the token
- Given_name: Full name of the logged in user

`

- Nameid: Id of the user in the database
- Role : Is the user doctor or not
- Unique_name: username of the user

If user is logged in as a patient, at the bottom of the page user can see three tabs.

- **Doctors Tab** - Patient can search and filter doctors based on various parameters such as specialization and location.

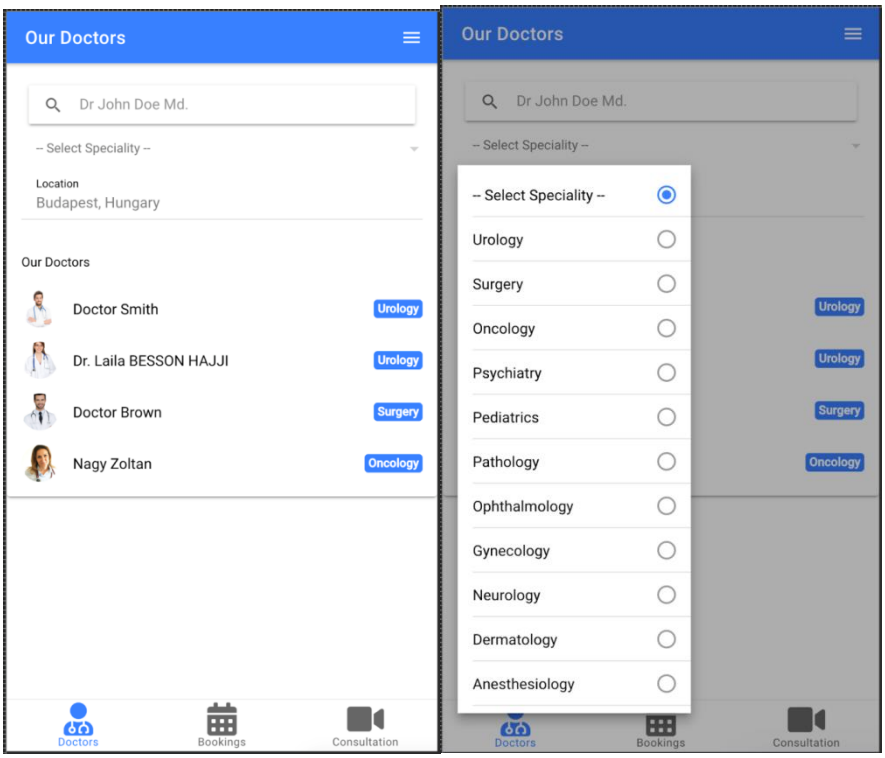


Fig 4.3 Doctors Tab

Once patient finds doctor it is interested in, it can click on the doctor to open the detailed view about doctor profile. Profile view includes more information about selected doctor. Also doctor's current schedule is displayed. User can swipe left or right to navigate in the calendar to future or past dates. User can click on the time slots to open a modal page, in order to book a future consultation. Of Course it is only possible to book future dates. Already booked time slots are filled and user is not allowed to book the same time slot.

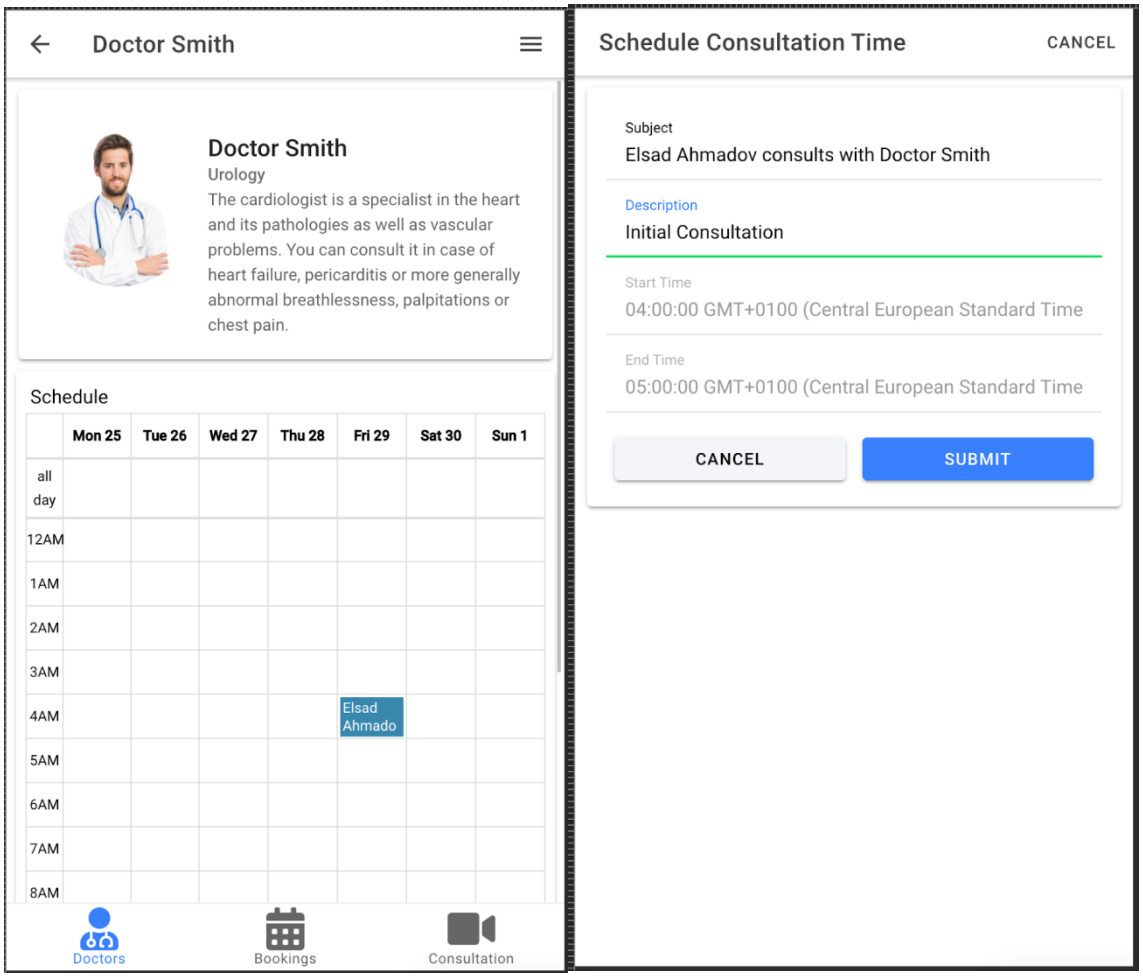


Fig4.4 Doctors Profile and Schedule Page

- **Bookings Tab** - Patient can see upcoming consultation sessions that has been booked.
    - User can see Doctor name and specialty
    - Start and end date time for consultation
    - If the current time is within the time range, Patient can click the START button and join the consultation session with the given doctor. Otherwise, popup notification warns user that the time is out of the range.

`

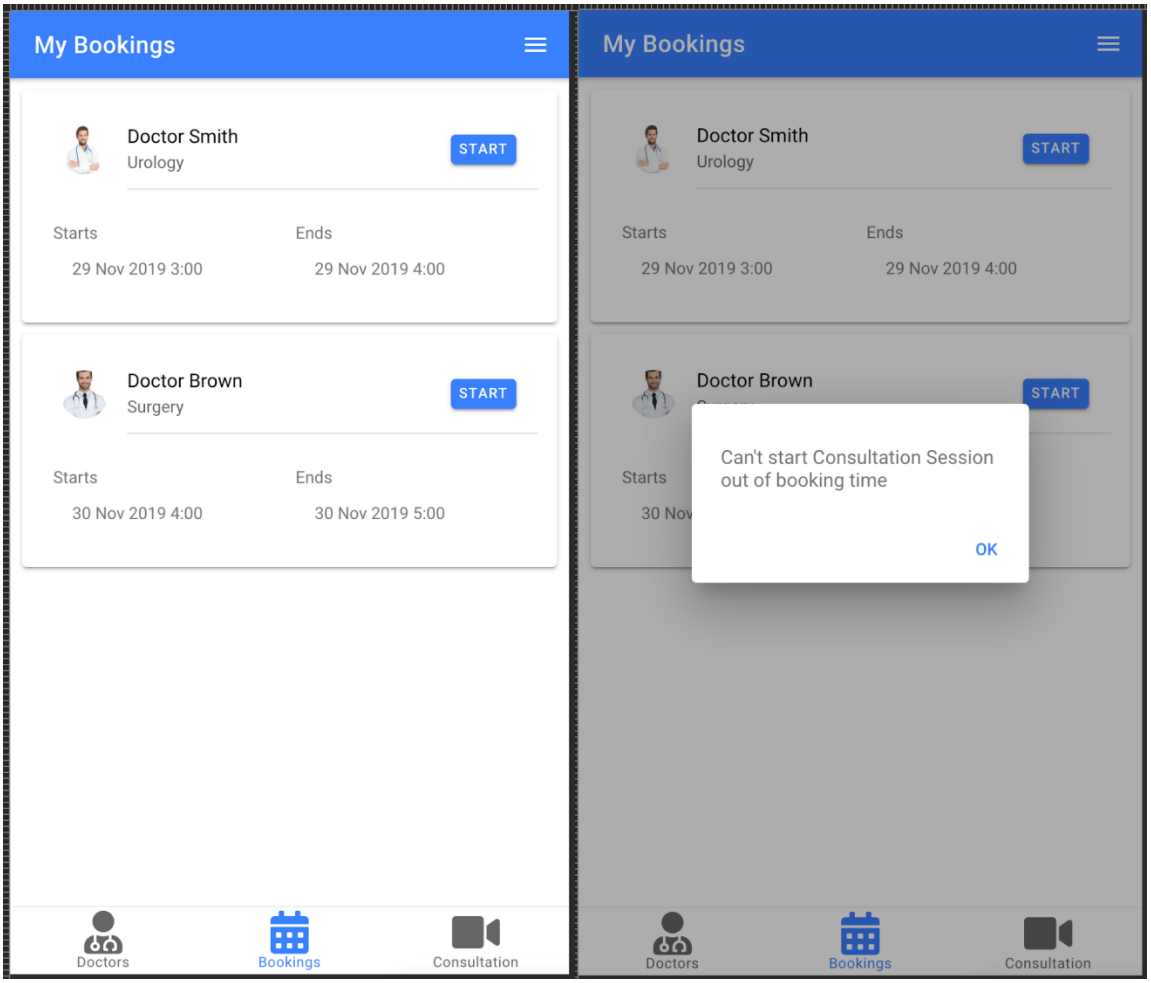


Fig4.5 Bookings Tab

- **Consultation Tab** - This is the user interface which realizes WebRTC session. Once both users have joined the current session, user can start or stop the session. On the top of the page user info such as fullname is displayed. Below that video stream from remote peer is displayed as a big section. Local video stream is displayed as a thumbnail. This can be switched on or off with “Show local video switcher”

`

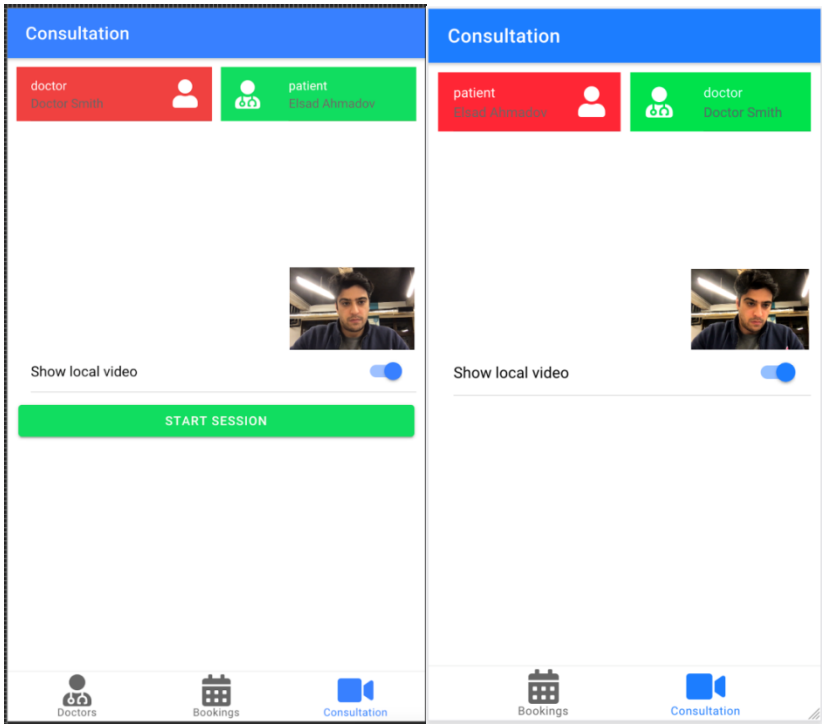


Fig.4.6 Before Session Started/ Patients view on the left , Doctors view on the right

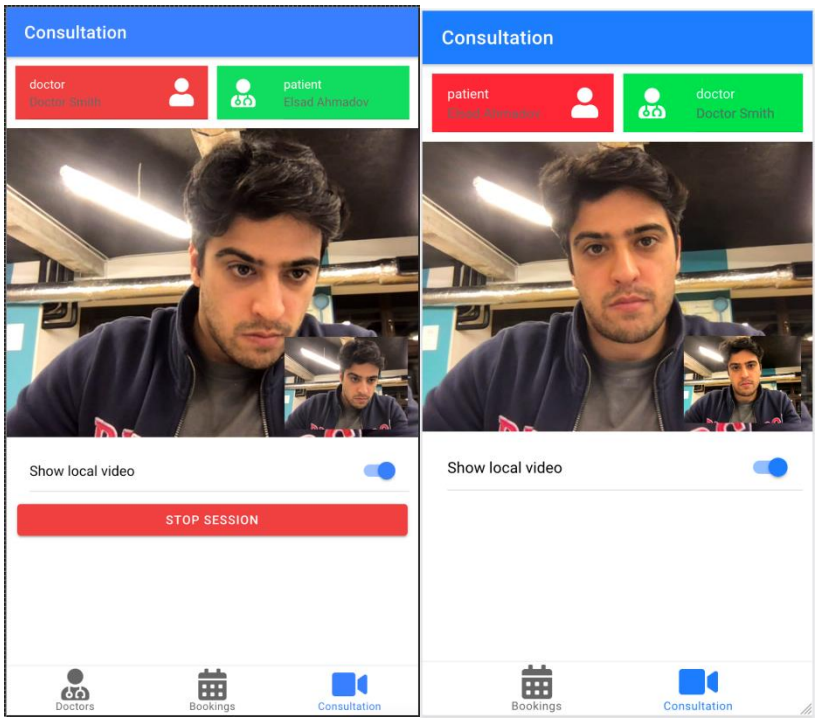


Fig 4.7 After Session Started/ Patients view on the left, Doctors view on the right

`

4.2 Server Side Web API

In this thesis project server side solely used as REST API application. This API serves following functionality

- Database Repository
- Authentication (Registering and Logging in Users)
- Doctor search functionality
- Booking and managing the appointments of patients with doctors
- Scheduling online consultation

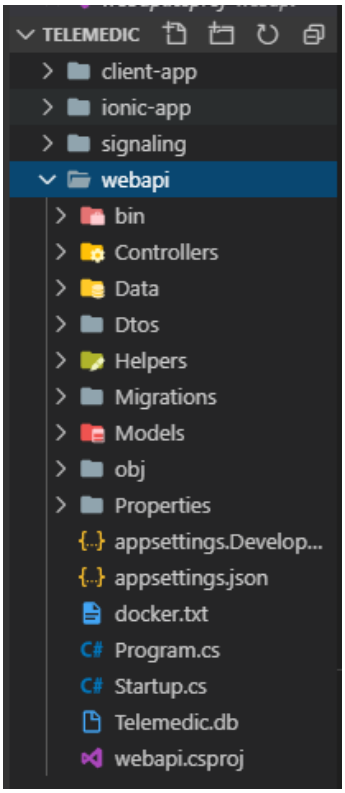


Fig 4.8 Server side project structure

In order to generate this project we have to download **.Net Core 2.2** and install. Then we have to run following command in cmd.

```
dotnet new webapi --name webapi --output webapi
```

`

Next we have to create our entity classes and create a **DataContext** class. **DataContext** is derived from Entity Framework **DbContext** parent class and is responsible for managing the database operations.

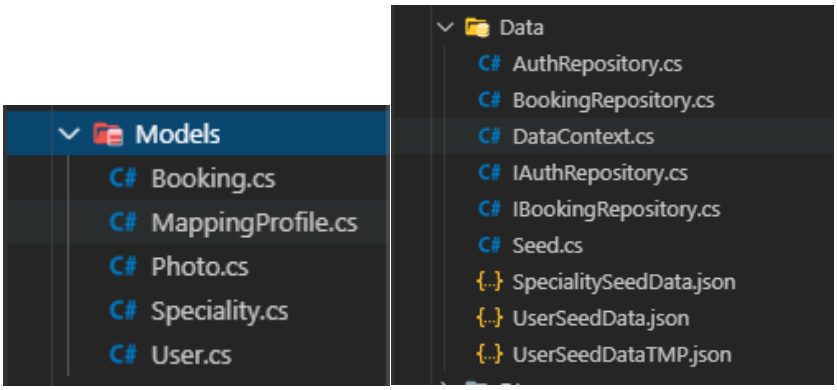


Fig 4.9,4.10 Entity classes and repository classes/interfaces

4.2.1 Database Repository

In order to manage the database in this project "Microsoft Entity Framework" is used. Entity Framework (EF) is an ORM that implements "Repository Pattern" [42] that used to abstract away functionality of database actions such as querying, inserting, updating and deleting data from database. In DataContext class we can also define **Entity Relationships** such as one to many relationships**.**

```
public class DataContext : DbContext
{
    public DataContext(DbContextOptions options) : base(options)
    {
    }

    override protected void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Booking>().HasOne(b => b.Bookee).WithMany(u => u.Bookers).HasForeignKey(b => b.BookeeId);
        modelBuilder.Entity<Booking>().HasOne(b => b.Booker).WithMany(u => u.Bookees).HasForeignKey(b => b.BookerId);

        modelBuilder.Entity<Photo>().ToTable("Photos");
    }

    public DbSet<User> Users { get; set; }
    public DbSet<Booking> Bookings { get; set; }
    public DbSet<Photo> Photos { get; set; }
    public DbSet<Speciality> Specialities { get; set; }
}
```

Fig 4.11 DataContext Class

`

In order to create database tables “Code First Migrations” are used. This approach allows us to create our object entity classes first. Then we have to register them with EF as show in above code snippet.

```
dotnet ef migrations add InitialMigration
dotnet ef database update
```

After running these commands, EF generates queries in order to create database tables in the db. Next, EF updates the database. Now we are ready to interact with db using EF. We don’t have to run raw SQL queries.

```
namespace webapi.Data
{
    public interface IAuthRepository
    {
        Task<User> Register(User user,string password);
        Task<User> Login(string username,string password);
        Task<bool> UserExists(string username);
    }
}
```

Fig 4.12 Repository interface for authentication.

```
namespace webapi.Data
{
    public interface IBookingRepository
    {
        void Add<T>(T Entity) where T: class;
        void Delete<T>(T Entity) where T: class;
        Task<bool> SaveAll();
        Task<Booking> GetBookingByDateRange(int bookeeId,DateTime start,DateTime end);
        Task<IEnumerable<User>> GetUsers(bool isBookee);
        Task<IEnumerable<Speciality>> GetSpecialities();
        Task<User> GetUser(int id);
        Task<Photo> GetPhoto(int id);
        Task<IEnumerable<Booking>> GetBookingsByBookerId(int bookerId);
        Task<IEnumerable<Booking>> GetBookingsByBookeeId(int bookeeId);
    }
}
```

Fig 4.13 Repository interface for Booking.

`

4.2.2 API Controllers

In order to process incoming requests we have to use controller classes. These classes are inherited from .NET Core BaseController parent class, which provides common functionality such as generating standard HTTP reponses such as:

- 200 Ok()
- 404 NotFound()
- 400 BadRerequest() etc.

```
[Route("api/[controller]")]
[ApiController]
public class AuthController : ControllerBase
{
    private readonly IAuthRepository _repo;
    private readonly IConfiguration _config;

    public AuthController(IAuthRepository repo, IConfiguration config)
    {
        this._config = config;
        this._repo = repo;
    }

    [HttpPost("register")]
    public async Task<IActionResult> Register(UserForRegisterDto userForRegisterDto)
    {
        userForRegisterDto.Username = userForRegisterDto.Username.ToLower();
        if (await _repo.UserExists(userForRegisterDto.Username))
        {
            return BadRequest("User already exists");
```

Fig 4.14 Sample code from AuthRepository

In order to interact with database repository we can use **Dependency Injection** to inject repository instances in our controller's constructor. Then, we are able to relay request data into the instance repository and execute our business logic.

`

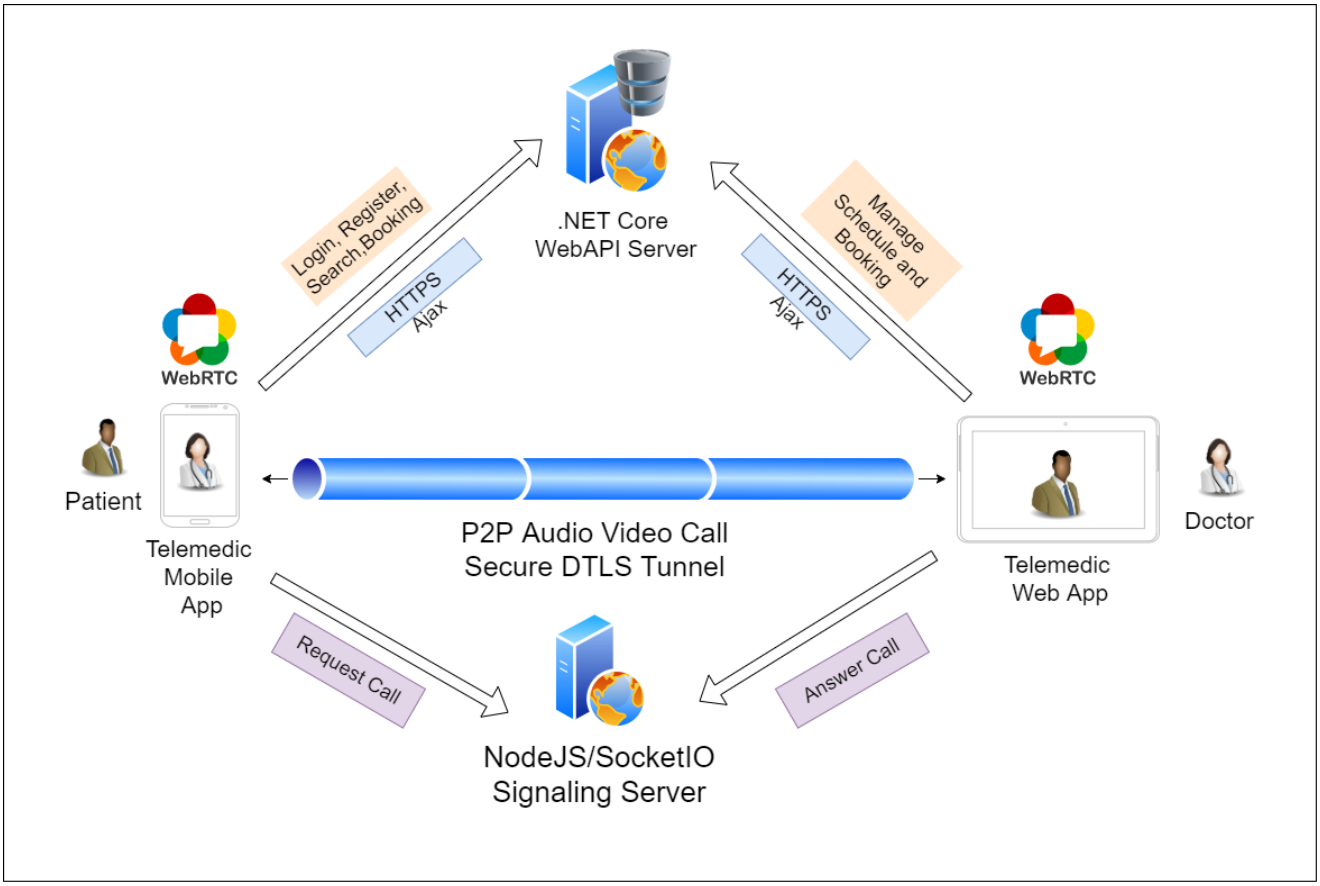


Fig 4.15 Telemedic Communication Architecture

`

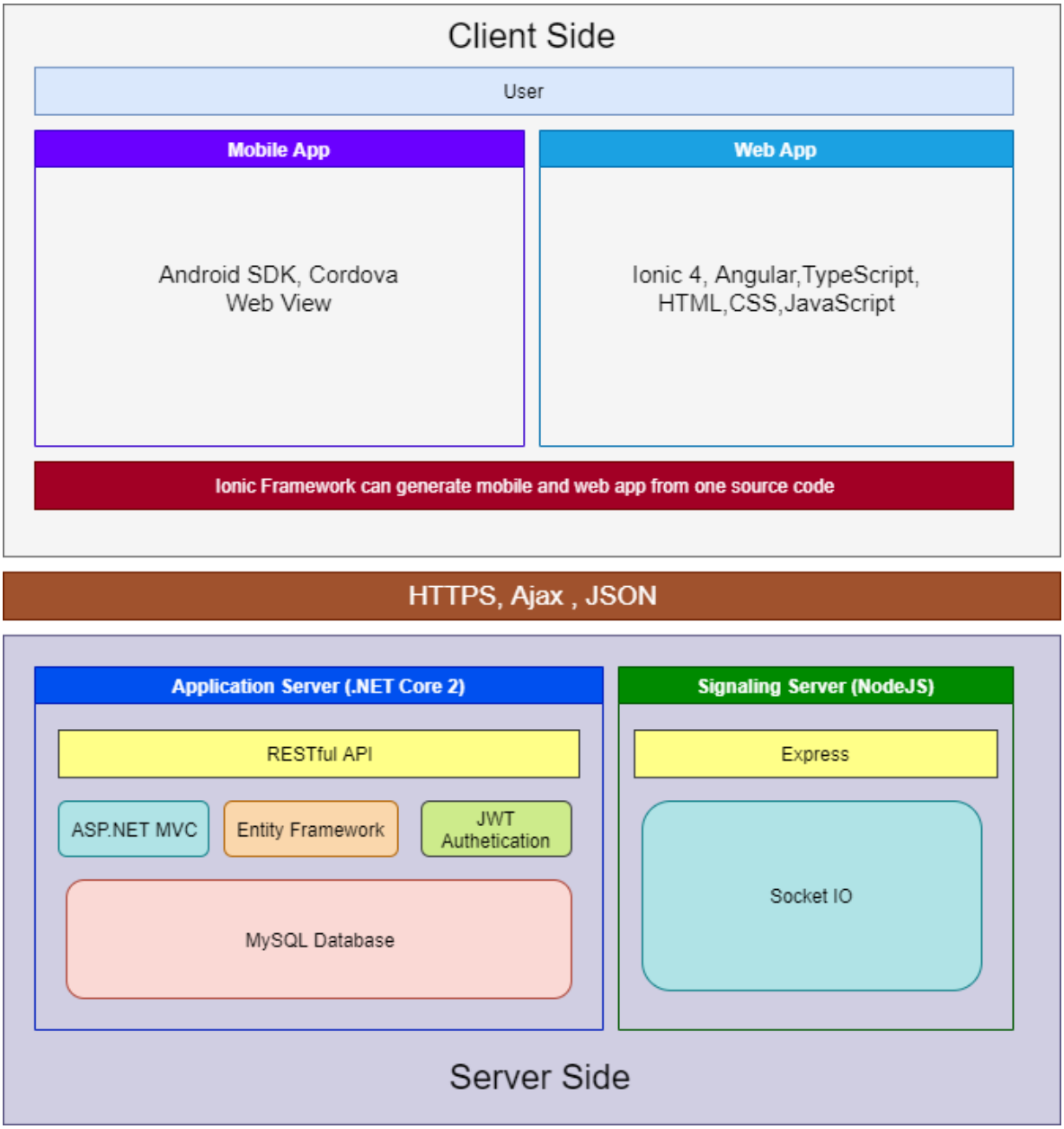


Fig 4.16 High Level Solution Architecture

`

4.3 Testing and Benchmarking

In order to test the performance of the application we can use Chrome browser built in diagnostic tools. To do so, start a new call session, and in a new tab address bar we have to type **chrome://webrtc-internals/**

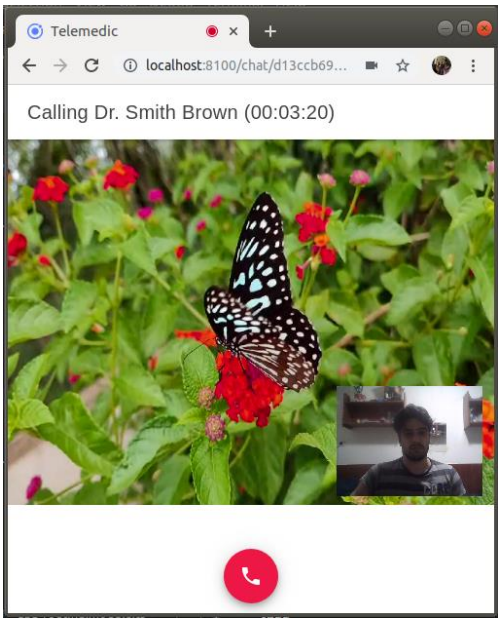
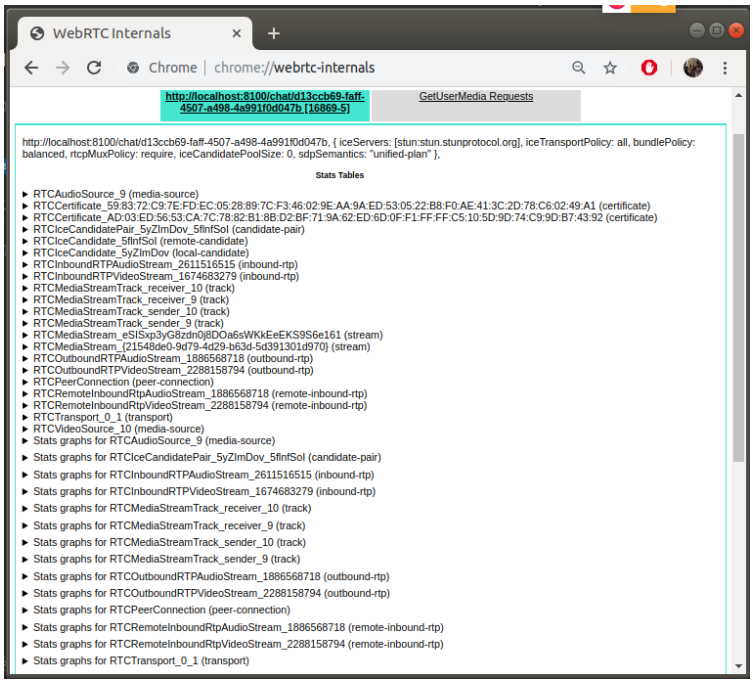


Fig 4.3.1 Example Consultation Session



`

Fig 4.3.2 **chrome://webrtc-internals/**

This page collects extensive data about current webrtc session. Using this data I can produce charts to visualize performance of the application with various parameters.

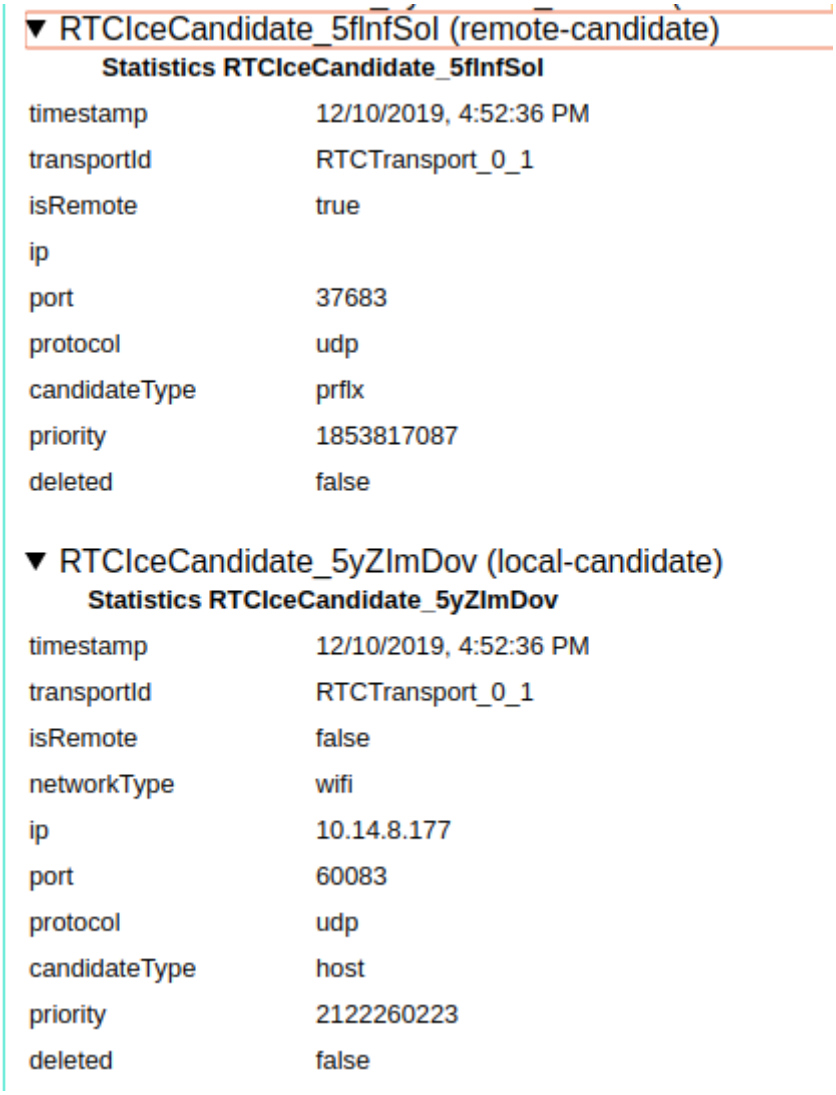


Fig 4.3.3 RTCIceCandidates show that transport protocol is UDP, user is connected over WIFI. Ip address and port combination can be seen.

`

▼ RTCInboundRTPAudioStream_2611516515 (inbound-rtp)

Statistics RTCInboundRTPAudioStream_2611516515

timestamp	12/10/2019, 4:56:13 PM
ssrc	2611516515
isRemote	false
mediaType	audio
kind	audio
trackId	RTCMediaStreamTrack_receiver_9
transportId	RTCTransport_0_1
codecId	RTCCodec_0_Inbound_111
[codec]	opus (payloadType: 111)
packetsReceived	50169
[packetsReceived/s]	50.11240422701066
bytesReceived	2759291
[bytesReceived/s]	2756.182232485586
packetsLost	0
lastPacketReceivedTimestamp	12003.3
jitter	0.004

Fig 4.3.4 Audio uses OPUS codec. Observe that packet transmission rate and jitter is healthy.

▼ RTCMediaStream_eSISxp3yG8zdn0j8DOa6sWKkEeEKS9S6e161 (stream)

Statistics RTCMediaStream_eSISxp3yG8zdn0j8DOa6sWKkEeEKS9S6e161

timestamp	12/10/2019, 5:02:24 PM
streamIdentifier	eSISxp3yG8zdn0j8DOa6sWKkEeEKS9S6e161
trackIds	["RTCMediaStreamTrack_sender_9","RTCMediaStreamTrack_sender_10"]

▼ RTCMediaStream_{21548de0-9d79-4d29-b63d-5d391301d970} (stream)

Statistics RTCMediaStream_{21548de0-9d79-4d29-b63d-5d391301d970}

timestamp	12/10/2019, 5:02:24 PM
streamIdentifier	{21548de0-9d79-4d29-b63d-5d391301d970}
trackIds	["RTCMediaStreamTrack_receiver_9","RTCMediaStreamTrack_receiver_10"]

Fig 4.3.5 Represents audio video data as they were acquired by MediaStream

▼ RTCTransport_0_1 (transport)

Statistics RTCTransport_0_1

timestamp	12/10/2019, 5:04:34 PM
bytesSent	178694978
[bytesSent/s]	142659.12601127382
bytesReceived	368337047
[bytesReceived/s]	275043.4644254228
dtlsState	connected
selectedCandidatePairId	RTCIceCandidatePair_5yZImDov_5flnfSol
localCertificateId	RTCCertificate_AD:03:ED:56:53:CA:7C:78:82:B1:8B:D2:BF:71:9A:62:ED:6D:0F:F1:FF:FF:C5:10:5D:9D:74:C9:9D:B7:43:92
remoteCertificateId	RTCCertificate_59:83:72:C9:7E:FD:EC:05:28:89:7C:F3:46:02:9E:AA:9A:ED:53:05:22:B8:F0:AE:41:3C:2D:78:C6:02:49:A1
selectedCandidatePairChanges	1

Fig 4.3.6 Current state of Transport layer. DTLS is connected and healthy. Connection is secured with certificates.

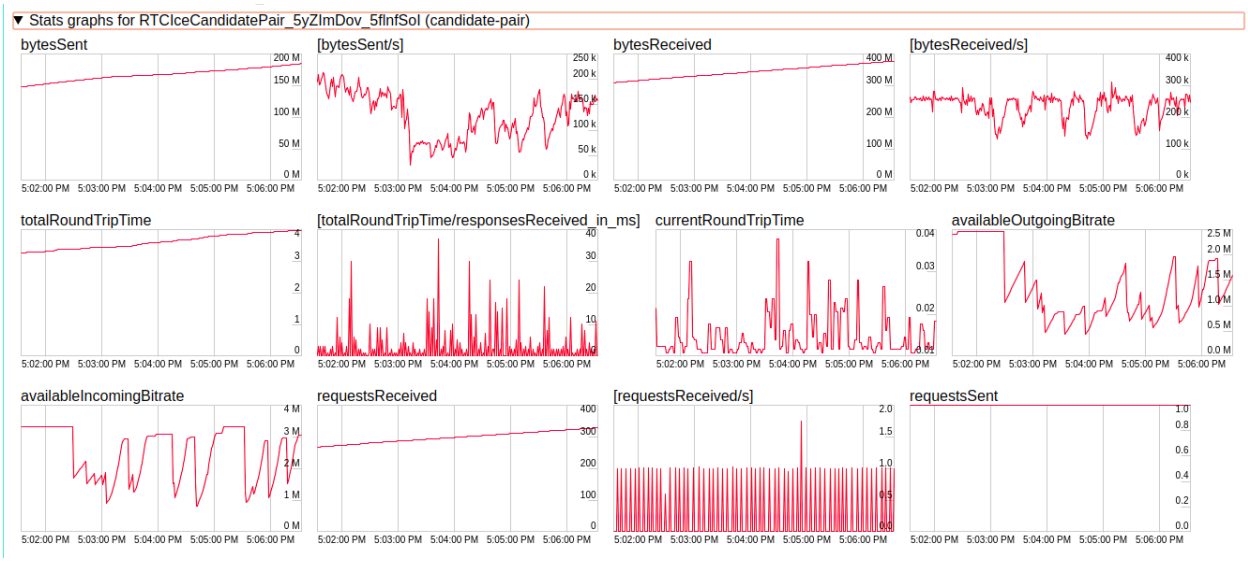


Fig 4.3.7 IceCandidate exchange visualization

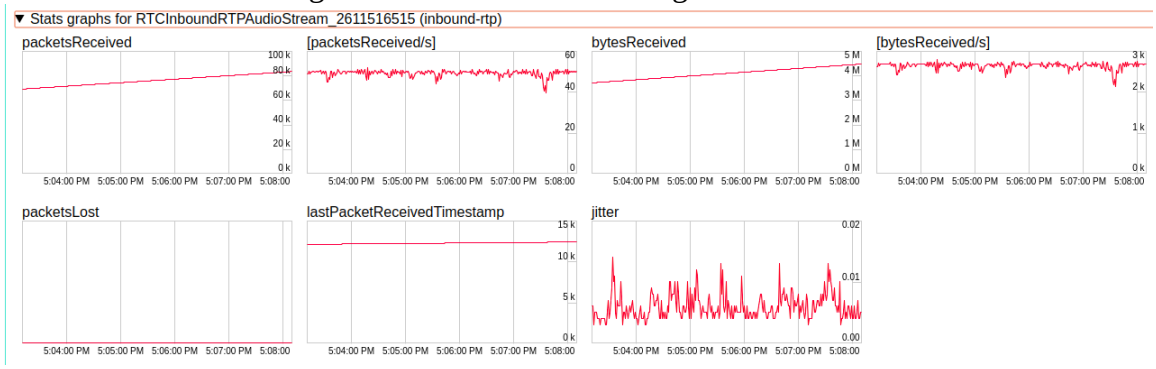


Fig. 4.3.8 Audio stream data visualisation(Jitter, Packets Sent/Received)

`

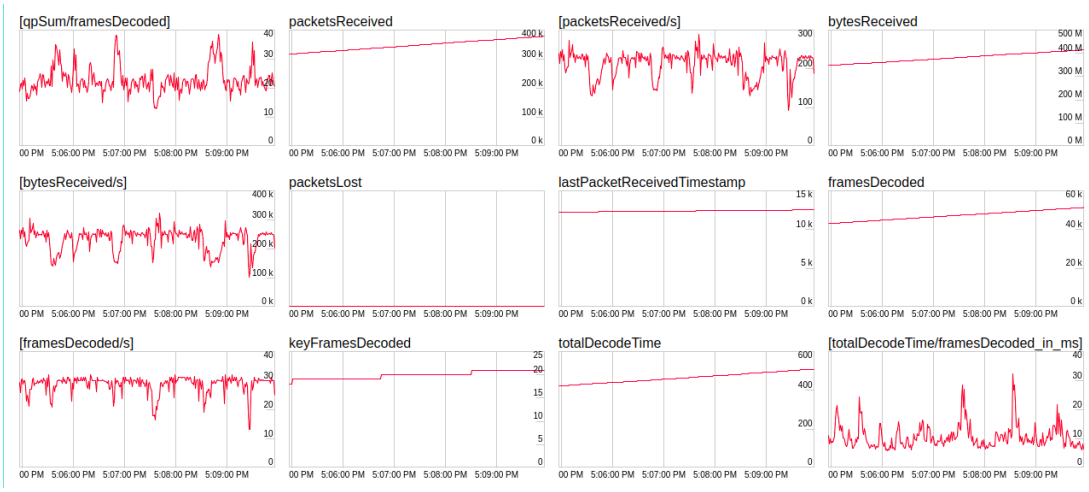


Fig.4.3.9 Video Stream charts. (Bytes Sent/Received, FPS etc.)

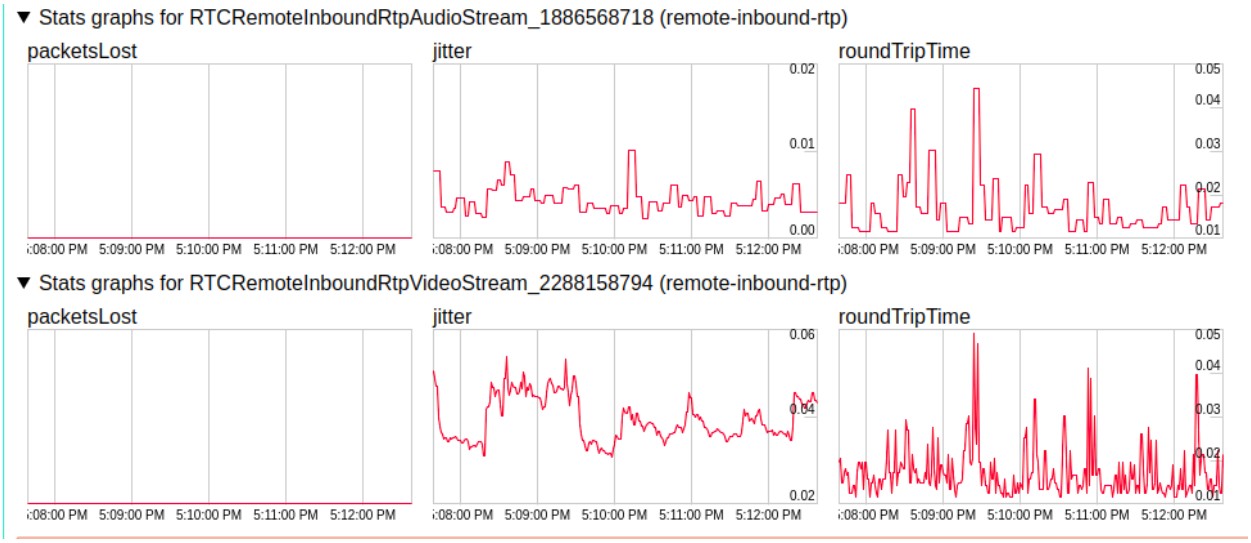


Fig.4.3.10 Audio and Video stream Jitter and RTT charts. (Very important for stream quality)

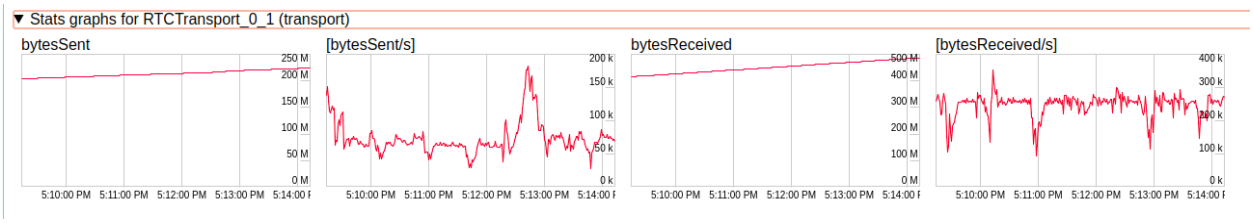


Fig 4.3.11 Transport protocol visualisation (Bytes Sent/Received) 5 minute sample.

`

Chapter 5 – Conclusion

In this thesis author summarized information necessary to build online audio video consultation application using WebRTC and other supporting technologies. Author tried to emphasize security concerns associated with building Web applications that uses HTTP as transport protocol.

To begin with, the author provided compelling reason why is there a need for such an application. It was pointed out that WebRTC has unique characteristics (peer-2-peer, https requirement, etc.) that allows building naturally secure applications from the get go. Using WebRTC it was emphasized that, peers do not need to relay their sensitive data to a third party. And they don't need to install additional software or plugins. Furthermore, author took a brief look at some of the popular existing medical consultation services that provide similar functionality.

In order to set the right expectations for feature development for this thesis project, it was discussed what are the core features required for a functional medical consultation application. After identifying main features chosen for development, author went on to review our options to select the right tools for this purpose. After comparing and contrasting few methods for mobile application development, we agreed upon using a **Hybrid Development** approach because of ease of development and multi-platform support.

Next we discussed what are the challenges and pitfalls are there regarding security of applications that use **TLS**. We reviewed requirements for server side and client side security, and addressed some security issues that present in this space.

Moreover , we presented a brief introduction to the WebRTC technologies and took a brief look at supporting protocols that help WebRTC. Additionally we provided basic information about security architecture of WebRTC. Because WebRTC doesn't specify signaling protocol we had to make a choice to select SocketIO as our signaling channel.

To conclude, we explored components of client side application. We presented screenshots from application pages and tabs to give the reader and idea how does the application work. Furthermore, we explained technologies used to realize server side of the project. We explained that server side of the application is a **.Net Core Web API** application that is responsible to process RESTful requests. Lastly we explained that we have chosen to use "Code First" approach with migrations and Entity Framework to realize tasks related to database.

In conclusion, we have to point out that, this project is by no means production ready. Because only a handful of features were implemented to demonstrate the capabilities of WebRTC. A real world application would require much more features and would require much more security measures to be considered industry ready. Author would like to recommend developing following features as a continuation of this work.

- In app reminders and notifications
- Finding a way to make signaling channel more secure
- Relaying patient data (Vital data, Sensor data)
- Emergency doctor consultation

`

Abbreviations

API	Application Programming Interface
REST	Representational State Transfer
JSEP	JavaScript Session Establishment Protocol
SDP	Session Description Protocol
ICE	Interactive Connectivity Establishment
SSL	Secure Sockets Layer
TLS	Transport Layer Security
DTLS	Datagram Transport Layer Security
MITM	Man in the Middle

`

References

1. https://www.forbes.com/sites/brucejapsen/2015/08/09/as-telehealth-booms-doctor-video-consults-to-double-by-2020/ [Online] [Accessed 2019-04-10]
2. https://www.practo.com/health-app [Online] [Accessed 2019-12-10]
3. https://www.zocdoc.com/ [Online] [Accessed 2019-12-10]
4. https://www.kry.se/en/ [Online] [Accessed 2019-12-10]
5. https://blog.cws.net/2015/04/breaking-down-mobilegeddon-and-responsive-website-design.html [Online] [Accessed 2019-04-10]
6. https://material.io/design/introduction/#principles [Online] [Accessed 2019-04-12]
7. https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/ [Online] [Accessed 2019-04-12]
8. https://myshadesofgray.files.wordpress.com/2014/04/native_html5_hybrid.jpg [Online] [Accessed 2019-04-15]
9. https://venturebeat.com/2012/09/11/facebooks-zuckerberg-the-biggest-mistake-weve-made-as-a-company-is-betting-on-html5-over-native/ [Online] [Accessed 2019-05-01]
10. https://venturebeat.com/2013/04/17/linkedin-mobile-web-breakup [Online] [Accessed 2019-05-01]
11. https://medium.com/airbnb-engineering/sunsetting-react-native-1868ba28e30a [Online] [Accessed 2019-05-01]
12. https://developers.google.com/web/fundamentals/design-and-ux/responsive/ [Online] [Accessed 2019-05-05]
13. https://developers.google.com/web/progressive-web-apps/ [Online] [Accessed 2019-05-05]
14. https://developer.mozilla.org/en-US/docs/WebAssembly [Online] [Accessed 2019-05-05]
15. https://cordova.apache.org/docs/en/latest/guide/overview/index.html [Online] [Accessed 2019-05-05]
16. https://ionicframework.com/docs [Online] [Accessed 2019-12-10]
17. Client-Focused Security Assessment of mHealth Apps and Recommended Practices to Prevent or Mitigate Transport Security Issues JMIR Mhealth Uhealth 2017
18. Server-Focused Security Assessment of Mobile Health Apps for Popular Mobile Platforms J Med Internet Res 2019
19. https://www.who.int/goe/publications/goe_mhealth_web.pdf [Online] [Accessed 2019-05-10]
20. Adibi S, editor. Mobile Health: A Technology Road Map. Cham, Switzerland: Springer International Publishing; 2015.
21. Sunyaev A. Consumer facing health care systems. e Serv J 2014 Jan;9(2):1-23.
22. Eur-lex.europa. 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC [Online] [Accessed 2019-05-10]

`

23. European Commission. Ec.europa. 2016. Privacy code of conduct on mobile health apps [Online] [Accessed 2019-05-10]
24. Möller B, Duong T, Kotowicz K. This POODLE bites: exploiting the SSL 3.0 fallback. Secur Advis 2014:1-4 [Online] [Accessed 2019-05-10]
25. Carvalho M, DeMott J, Ford R, Wheeler DA. Heartbleed 101. IEEE Secur Priv 2014 Jul;12(4):63-67.
26. Mehta N. Heartbleed. 2014. The Heartbleed Bug CVE-2014-0160
27. Mobile Top 10 2016-Top 10 [Online] [Accessed 2019-05-15]
28. L. Huang, S. Adhikarla, D. Boneh and C. Jackson, "An Experimental Study of TLS Forward Secrecy Deployments," in IEEE Internet Computing, vol. 18, no. 6, pp. 43-51, Nov.-Dec. 2014.
29. https://thinkdiff.net/mysql/encrypt-mysql-data-using-aes-techniques/ [Online] [Accessed 2019-10-10]
30. https://www.mysql.com/products/enterprise/encryption.html [Online] [Accessed 2019-10-10]
31. http://www.webrtc.org/home [Online] [Accessed 2019-10-10]
32. "Is WebRTC ready yet?" http://iswebrtcreadyyet.com/ [Online] [Accessed 2019-10-10]
33. https://www.pcworld.com/article/3102890/goodbye-hello-firefox-casts-off-its-built-in-video-chat-client-this-fall.html [Online] [Accessed 2019-08-01]
34. https://zingaya.com/product/ [Online] [Accessed 2019-08-01]
35. K. Jain, A. Himmatramka, A. Bhandary, A. D'silva and D. Barge, "Synchronized Development Using WebRTC Real-Time Collaboration in WebRTC," International Journal of Engineering Science, vol. 6, no. 4, 2016.
36. I. V. Osipov, A. A. Volinsky and A. Y. Prasikova, "E-Learning Collaborative System for Practicing Foreign Languages with Native Speakers," International Journal of Advanced Computer Science and Applications, vol. 7, no. 3, 2016.
37. https://www.prnewswire.com/news-releases/webrtc-market-worth-us8152-billion-by-2025-transparency-market-research-618487683.html [Online] [Accessed 2019-07-3]
38. Google Chrome team, "Interop Notes," Google Inc http:// www.webrtc.org/web-apis/interop [Online] [Accessed 2019-08-01]
39. https://developer.mozilla.org/enUS/docs/Web/Media/Formats/WebRTC_codecs [Online] [Accessed 2019-08-01]
40. I. Grikorik, "Chapter 18. WebRTC," in High Performance Browser Networking, O' Reilly Media, Inc., 2013
41. https://developer.mozilla.org/en-US/docs/Web/API/Media_Streams_API [Online][Accessed 2019-10-10]
42. https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design [Online] [Accessed 2019-12-04]