Med Connect

3rd Inc by Group 17

Suramya Chukka 11

Sarath Chand Lingamgunta 43

Praveen Kumar Surapaneni 58

I. Introduction:

For the past few years we have been seeing a lot of companies adopting video conferencing as their primary medium of communication with their customers. The technological advancements done in the field of Video conferencing are more than enough to make the customers think that VC is better medium of communication, as it provides lot of benefits like zero time for travel, reduced cost for infrastructure. There are lot of Video Conferencing Applications like Skype, Hangouts which provide best possible way to video chat with others, but these doesn't offer scheduling of the meetings, record the meeting, searching through the meetings and chatting with the customer. We want to create an Android/IOS app for medical field which can streamline the interactions between the doctor and patient by letting them video chat with each other.

II. Project Goal and Objectives:

Overall Goal

The purpose of this App is to let the doctor video chat with the patient, for doing so we have to implement a WebRTC which transfers the real-time video data across the systems. Two new REST services has to be built which will take care of uploading the data about the meeting in to user's personal calendar using Date Picker plug-in. Another service has to be created to upload all the meeting videos in to Cloud database (Mongo DB) which must be saved for further references. A new message Queue must be built to handle the chat between the patient and doctor. Implement push notifications to send remainders to the users about their appointments.

Specific objectives

Visiting a doctor can be huge task for a patient if he is unable to move, if the patient wants to know what doctor said during his previous visit, if the patient want to receive notifications about the Appointment, if the patient want a log of everything that happened between him and his personal doctor. To provide all these needs to the user the App must have the below functionalities.

- Provide the time slots of all the doctors that are open to the patients
- Send a notification to the doctor if he is booked for a particular time slot
- Create a meeting in the doctor and patient's calendar
- Suggest the best doctors for a particular health condition
- Provide rating system for the patients to rate the doctors
- Record the meeting and save in the patient's history to watch it later
- Provide a search engine to search for doctors and their videos
- Maintain the chat history between the doctor and his patient
- Provide users an easy access to their previous and upcoming appointments

We want to bundle all of these tasks in a single app which will help our app stand out and deliver best features to the users.

Features Implemented

As we are following an agile methodology, we implemented the primary objectives of our application for this increment. This increment mainly focuses on the functionalities that our app provides to the Patient. They are as follows:

- Register new users (Patients) to our app with required information.
- Log in functionality for the registered users which is the gateway to our app.
- Pull out the existing Doctors of our app and displaying the list of Doctors to the Patient.
- Search functionality which helps Patients to search for a particular Doctor or a particular specialization.
- View the complete details of a selected Doctor.
- Schedule appointment with the desired Doctor, which includes selecting date and time.
- View scheduled appointments.
- Start a Video session with the Doctor on the scheduled date.
- Video call between Doctor and Patient.
- Separate flow for doctors and patients.
- Doctors and patients can chat with each other
- Doctors can view their daily schedule

We believe the above mentioned features are the core functionalities of our application and hence they are implemented for this increment. All the additional features would be fulfilled as we move on to the next increment.

III. Existing Services /API

The following are the various plug-ins / existing services we included in our app for this increment.

Date Picker:

In order for the Patient to pick a date and time for his schedule, we included the Date Picker plug - in of Cordova which would show the calendar for the user to select a date. The selected date would then get dumped in the database and gets stored. This could be viewed in the appointments section for future reference.

White List:

As far as the video conferencing is concerned, we deployed it over IBM Bluemix and it is a well-known fact that Ionic framework does not exclusively allow reference to the external URLs. So we are using ng Cordova white list plug-in which will white list our Bluemix url so that ionic can open the specified web page inside the App.

Facebook OAuth:

The user of our application can even log in using his/her Facebook login. To fulfill this feature, we included Facebook OAuth which allows user to provide access permissions to the app leading to further login.

Mongo DB Data API:

Mongo DB is the database we have chosen to store all the application data. In order to access and perform CRUD operations of our application, we used Mongo Lab Data API in our services.

Socket.IO:

We are using simple http server built in java script to handle the chat between the clients. For sending and receiving messages we are using socket.io

IV. Detail Design of services:

The following is the document where all the design diagrams (Class, Sequence, Architecture diagrams), Wireframes and Mockups of our application are made.

https://www.dropbox.com/s/5gwwolu5d8xi4kh/WireFrame.docx?dl=0


Design .docx

User Stories/ Use Cases:

For the second iteration:

1) Create login screen having user name and password fields along with sign in button.
2) Provide feature to login with their Facebook and google accounts.
3) Also provide signup button for new users.
4) In the signup screen user can enter his basic information.
5) Provide tabbed view in home screen.
6) In the home screen first tab, provide all the list of doctors along with their professions.
7) Provide search feature to search a doctor from the list.
8) Once a doctor is selected then he should view the doctor information along with the schedule an appointment option.
9) By selecting date and time user should schedule an appointment with the doctor.
10) Second tab should list all the scheduled appointments.
11) Provide a join button for every scheduled appointment.
12) By clicking that join button user should proceed to video conference screen.
13) In the third tab provide account setting options.

For the third iteration:

1) Create a login screen for doctor.
2) Create a sign up screen for doctor and collect all the info related to his medical practice
3) Doctors will be able to go through their day's schedule
4) Doctors will be able to search their appointments
5) Users will be able to update their profile
6) Doctors can update their availability times and can create new availability times
7) Fixed the bugs related to searching the patients
8) Patients will be able to chat with the doctor.

User Services Description:

A brief summary of the services our app provides is given below:

Registration Service:

The starting point of our application is registration service for the new users. The front end User Interface designed using Ionic framework would prompt the user to provide the necessary details. When the user clicks on the "Register" button, API call is made using POST method to the Mongo DB collection of Users.

All the details are then pushed into the Collection forming a new document. A part of the source code for the above service looks as follows

```
    var promise = deferred.promise;
    $http({
  method: 'POST',
  url: 'https://api.mongolab.com/api/1/databases/medcon/collections/Users?apiKey=BSHQMdPlD-USKTsizAPOBio_XS-05S-b',
  data: JSON.stringify({
 firstname: fname,
  lastname: lname,
  address: address,
  age: age,
  email: email,
  username: username,
  password: password,
```

Login Service:

Once a user is registered, he can access our app by providing his credentials. The user interface prompts for the username and password. These details provided by the user would be sent back to the MongoDB for verification using an API call with GET method.

```
        $http({
    method: 'GET',
    url: 'https://api.mongolab.com/api/1/databases/medcon/collections/Users?q={username:\''+name+'\'}&apiKey=BSHQMdPlD-
sizAPOBio_XS-05S-b',
    contentType:"application/json"

}).success(function(data){
  if (name == data[0].username && pw == data[0].password) {
     localStorage.setItem("patientid",data[0]._id.$oid );
          deferred.resolve('Welcome ' + data[0].username + '!');
       } else {
          deferred.reject('Wrong credentials.');
       }
```

Only when this service returns a success message, the user is allowed access to the application. Or else, an error message would be displayed.

Login with Facebook:

In order for the user to login using Facebook credentials, the Facebook API plug-in is installed and is accessed using the OAuth Key provided by the Facebook Developers service.

When the user clicks the "login with Facebook" button, a pop up of Facebook login page opens. After the user has successfully entered his details, Facebook asks the user to allow access to the MedConnect app.

A part of the source code for the above service is as follows:

```
    loginProcessed = true;
    if (url.indexOf("access_token=") > 0) {
        queryString = url.substr(url.indexOf('#') + 1);
        obj = parseQueryString(queryString);
        tokenStore.fbAccessToken = obj['access_token'];
        if (loginCallback) loginCallback({status: 'connected', authResponse: {accessToken: obj['access_token']}});
    } else if (url.indexOf("error=") > 0) {
        queryString = url.substring(url.indexOf('?') + 1, url.indexOf('#'));
        obj = parseQueryString(queryString);
        if (loginCallback) loginCallback({status: 'not_authorized', error: obj.error});
    } else {
        if (loginCallback) loginCallback({status: 'not_authorized'});
    }
}
```

View Doctors:

Once the user has successfully logged in to the application, the first tab of the next screen displays a list of existing doctors with their name and specialization field.

All the displayed data is actually pulled from the existing database collection of Doctors through an API call using GET method.

```
    $http({
        method: 'GET',
        url: 'https://api.mongolab.com/api/1/databases/medcon/collections/Doctors?&apiKey=BSHQMdPlD-USKTsizAPOBio_XS-05S-b',
        contentType:"application/json"

    }).success(function(data){
        Doctors = data;
        deferred.resolve(data);
    })
    return deferred.promise;
},
```

Search Doctors:

To provide easier access, our app provides search service which allows users to search for a specific doctor or a particular specialization. The front end shows a Search button where the user enters a term of his search and clicks it.

Once the Search button is clicked, an API call is made to the Database and search is performed using the search key provided by the user.

```
var results = employees.filter(function(element) {
    var fullName = element.firstname + " " + element.lastname;
    return fullName.toLowerCase().indexOf(searchKey.toLowerCase()) > -1;
});
```

The returned Doctor would be displayed to the user meeting his search criteria.

View Doctor Information / Schedule Appointment:

When the user clicks on the desired Doctor, a new page opens displaying the full details of the Doctor. The details include his name, specialization, location, reports and contact information.

If the patient wishes to schedule an appointment with this doctor, he can choose the date and time from the buttons provided. Each click would make a call to the corresponding function where the DatePicker plug -in is called.

Once the user sets both the date and time and clicks on the "Schedule" button, a new appointment is made. An API call is made to the appointments collection and the corresponding data is pushed into the new document created.

```
$http({
  method: 'POST',
  url: 'https://api.mongolab.com/api/1/databases/medcon/collections/appointments?apiKey=BSHQMdPlD-USKTsizAPOBio_XS-05S-b',
  data: JSON.stringify({
      patientid: patientid,
       doctormongoid:doctormongoid,
      doctorid:doctorid,
      date:date,
      h:h,
      m:m,
      a:m,
      doctorname:doctorname
  }),
```

The Patient will then be directed to the "View Appointments" page which is the second tab.
The newly created appointment would be shown there.

View Appointments:

If the user wants to view all his appointments, he clicks on the second tab. There the front end shows a list of his appointments made so far.

An API call to the appointments collection would pull all the data and sends back to the User Interface.

```
    $http({
        method: 'GET',
        url: 'https://api.mongolab.com/api/1/databases/medcon/collections/appointments?&q=
:\''+patientid+'\'}apiKey=BSHQMdPlD-USKTsizAPOBio_XS-05S-b',
        contentType:"application/json"

    }).success(function(data){
        appointments = data;
        deferred.resolve(data);
    })
    return deferred.promise;
```

Chatting:

We are introducing a new feature for doctors and patients which helps them in contacting each other using text chat. Before this update the only mode of contact is video chat, but video chat may not be required for all time and it may not be appropriate for small queries. So text chat can be a very handy feature for all the users of our App**.**

```
var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);

app.get('/', function(req, res){
  res.sendFile(__dirname + '/index.html');
});

io.on('connection', function(socket){
  socket.on('chat message', function(msg){
    io.emit('chat message', msg);
  });
});

http.listen(3000, function(){
  console.log('listening on *:3000');
});
```

We are using sockets.io to establish a secure connection between the two users and by using sockets the burden on the server will also be reduced by almost 80 as it doesn't have to listen for any new message, instead the socket will notify the server that a new has been received for a particular user**.**

Design Patterns Implemented:

Module:

Created a config module that unified all the classes related to the socket.io video chat interface.

Command:

In our code for displaying the video during the video chat we are using the getMediaElement command which encapsulates all the code related to creating a video element and displaying it to the user. Here command is getMediaElement, invoker is the client who invokes the video chat.

Null Object:

There are many instances where we implemented null object pattern, one such beautiful instance is when the other party is not available for video chat, the invoker will just see his own

video rather than a blank screen indicating that the system is waiting for the other party to join the call.

Singleton:
Whenever user log in to the App all his personal information will be kept inside a singleton abject and this info will be used by all the views.

Observer:
When a new participant joins the call an observer will invoke the services in the client machine and let them know that a new participant has joined the call.

Factory:
For updating a particular field in the document stored in mongo DB, we created a factory which will decide which field has to be updated based on the changes done on the user.

V. Testing:

Unit testing:

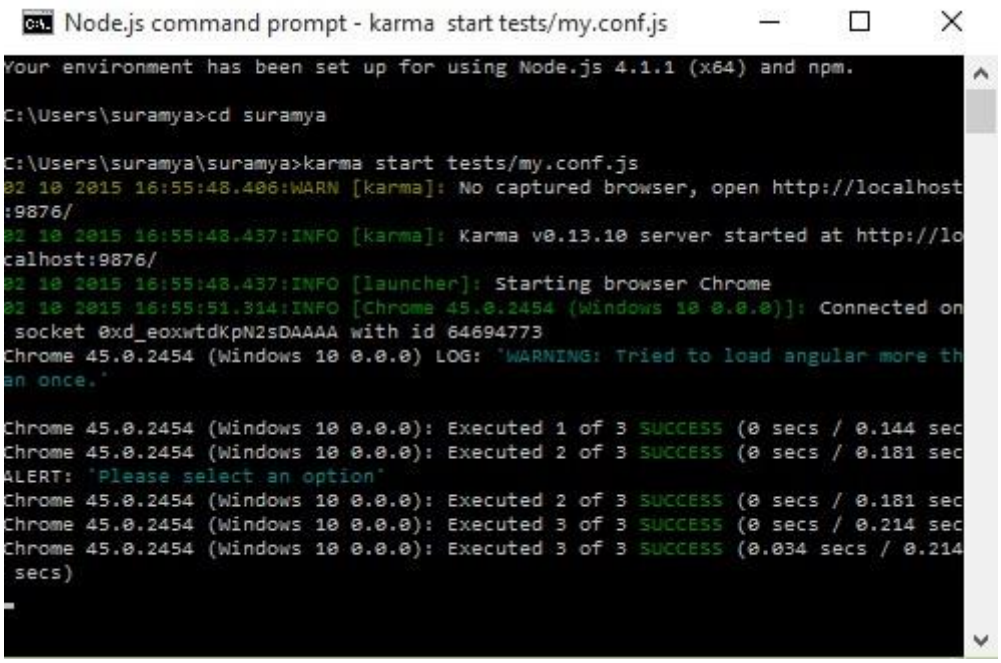
Karma tool is used to perform the unit testing for our app. For this increment, we have written three test cases.

- The first test case verifies the functionality of login page. No other user other than the authorized user should be allowed access to the application.
- The second test case checks for the functionality of search.
- The third test case tests the functionality of scheduling the appointments.

All the three test cases are written in separate files. my.conf.js file is configured so that all the required test files are included .

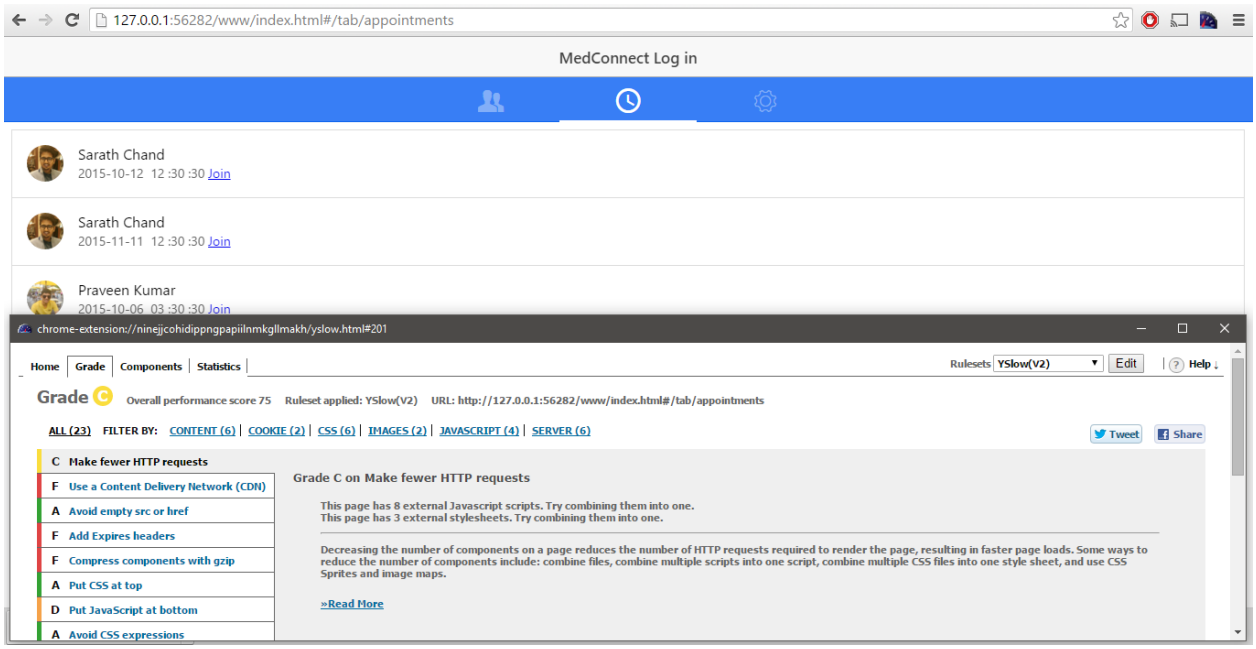
karma is initialized and the test cases are executed.

As we have implemented necessary verifications, all our three test cases passed.



Performance testing:

To conduct Performance testing, we used YSlow. Our app is served on the browser and YSlow is initialized.



As stated by YSlow, we moved the entire JavaScript to the bottom of the App, css to the top of the file, removed inline CSS expressions, removed the empty href. But we were unable to load JavaScript from CDN.

VI. Implementation:

Video Calling Server Side

Video conferencing part of our mobile app has been developed as separate web application and we deployed it IBM Bluemix. We are referencing this web app inside our Mobile client using web view. Cordova doesn't allow Ionic to refer external urls other than those that start with file://. So we are using ng Cordova white list plugin which will white list our Bluemix url so that ionic can open the specified web page inside the App. As the video streaming and recording is being handled by our web app there is no need for install additional plugins for ng Cordova camera and video recording. This helped us in substantially reducing the size of our App precisely from 13MB to 4MB. One more advantage of using web view for video conferencing is

we effectively detached it from mobile development so we can do changes in our web app without the need to update the Mobile App.

The two important steps involved in developing a video chat App is establishing a secure connection between the two clients and examining the video packets between the two clients. For establishing a secure connection between the two clients we need a signaling server.

Signaling is the communication mechanism between the two clients. For a WebRTC app to establish a connection between the clients its clients' needs to exchange the following info.

- Session control data for opening or closing the connection.
- Error messages.
- Meta data of the media like supported codecs, video types, average bandwidth
- Secure connection keys, for data exchange.
- Network data, IP addresses and mac address as seen by the world.

Signaling Server Development

WebRTC API, by default doesn't provide any signaling server, so we have to create our own way of signaling between the two clients. We are using Firebase for signaling the data between the two users and socket.io js for creating the sockets. We will be treating every new connection as a channel and for every new channel we will be creating a new Firebase reference indicating the data write point of this channel.

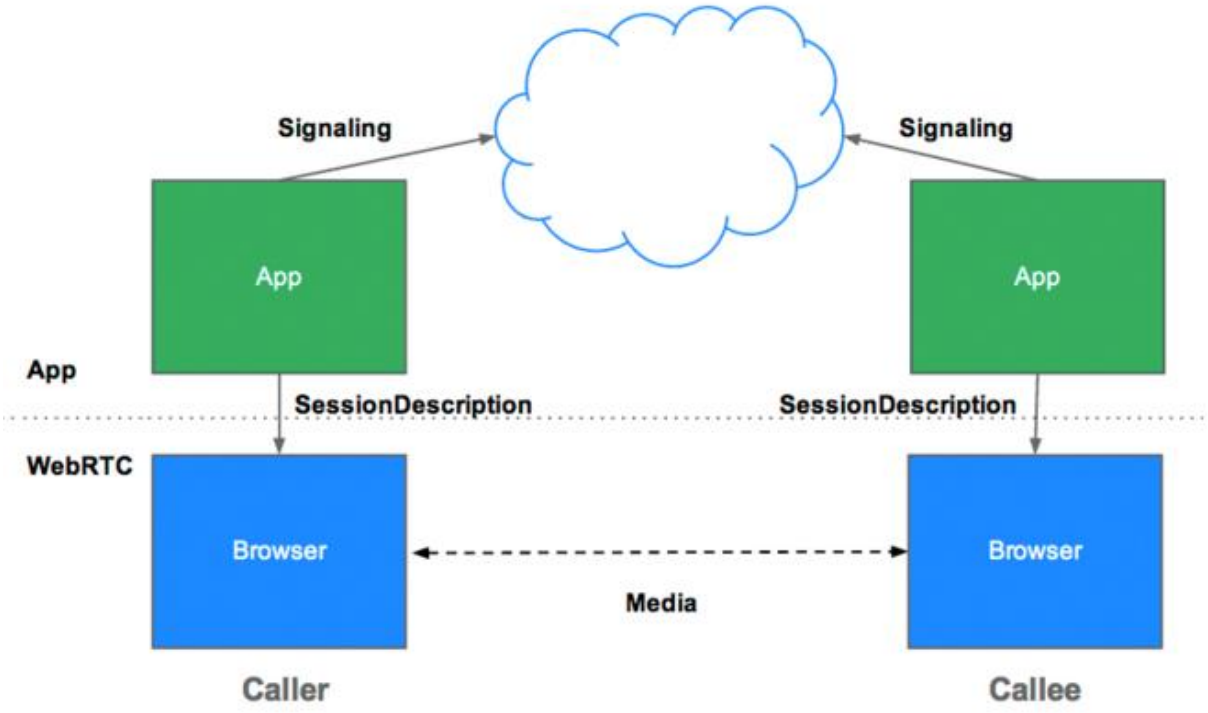


Fig: Architecture of Signaling Server

```
var channel = config.channel || location.href.replace( /\/|:|#|%|\.|\[|\]/g , '');
var socket = new Firebase('https://webrtc.firebaseIO.com/' + channel);
socket.channel = channel;
socket.on("child_added", function(data) {
   config.onmessage && config.onmessage(data.val());
});
socket.send = function(data) {
   this.push(data);
};
```

Fig: Code for creating firebase reference

What the above code snipped does is it will create a new reference in firebase for data write point and whenever there are any child's added to this channel we will get a call back and we will send our data. Channel id is unique id given to each new connection. Based on the channel id we will identify which two clients' wants to chat with other. If A is the channel for system1 and B is channel id for system2, if A requests for B and B requests for A then we assume that these two clients want to chat with other and by using Firebase we will exchange the signaling data between these two and establish a connection.

RTCPeerConnection Implementation

WebRTC uses RTCPeerConnection API to transfer video and audio data between the clients. In WebRTC every bit of data will be transferred using peer to peer protocol, so there won't be a server overhead for us where we have to transfer data from client to server and server to client. WebRTC is an advanced and only modern browsers like chrome, Firefox and safari supports it. To explain the entire flow process better, please consider the scenario where Taylor is trying to call Megan

- Taylor creates an RTCPeerConnection object.
- Taylor creates an offer (an SDP session description) with the RTCPeerConnection createOffer () method.
- Taylor calls setLocalDescription() with his offer.
- Taylor stringifies the offer and uses a signaling mechanism to send it to Eve.
- Megan calls setRemoteDescription() with Taylor's offer, so that her RTCPeerConnection knows about Taylor's setup.
- Megan calls createAnswer(), and the success callback for this is passed a local session description: Megan's answer.
- Megan sets her answer as the local description by calling setLocalDescription().
- Megan then uses the signaling mechanism to send her stringified answer back to Alice.
- Taylor sets Megan's answer as the remote session description using setRemoteDescription().

Taylor and Megan also need to exchange network information.

- Taylor creates an RTCPeerConnection object with an onicecandidate handler.
- The handler is called when network candidates become available.
- In the handler, Taylor sends stringified candidate data to Megan, via their signaling channel.
- When Megan gets a candidate message from Taylor, she calls addIceCandidate(), to add the candidate to the remote peer description.

Data Base

We are using Mongo DB as our data base. We hosted our DB in mongo labs so it can be remotely accessible at any point of time. For now we are using three different collections one to hold the list of doctors and their info, other to hold to the personal chats and video chat history of each patient and last one to hold all the users info that was collected during sign up.

appointments	15	false	11.50 KB
Doctors	3	false	9.44 KB
Users	2	false	8.45 KB

All the data from these collections will be accessed by a rest service created by us and we use this rest service in our app to access the data base.

Rest Services

We have developed four different REST services and one Web App which are all deployed in IBM Bluemix. Huge number of network call has to happen to begin and establish a hand shake between the two clients for video conferencing and if we implement this in mobile environment it will impact the performance of our App. So we decoupled the video chat part of our App and developed a separate Web app which will handle all the signaling and connection establishment and only video will be sent out to the user. Most of the processing will be done by the server.

Below is the URL for the Web App we developed for Video Conferencing.

medconnect.mybluemix.net

There won't be join buttons or call buttons in this web app as all the UI will be part of the mobile app and web app will get the user id and channel id from the Mobile client and web App will establish the video session and send the video data to the client.

The four REST services we created for now will be handling all the database calls to Mongo DB. One for create, other for read, and the other two for delete and update. We have followed this two architecture for interacting with the DB as we are will be receiving huge amounts of data server and we don't to process them in the mobile. So our REST will process the data in the server and will send out the output to the mobile client.

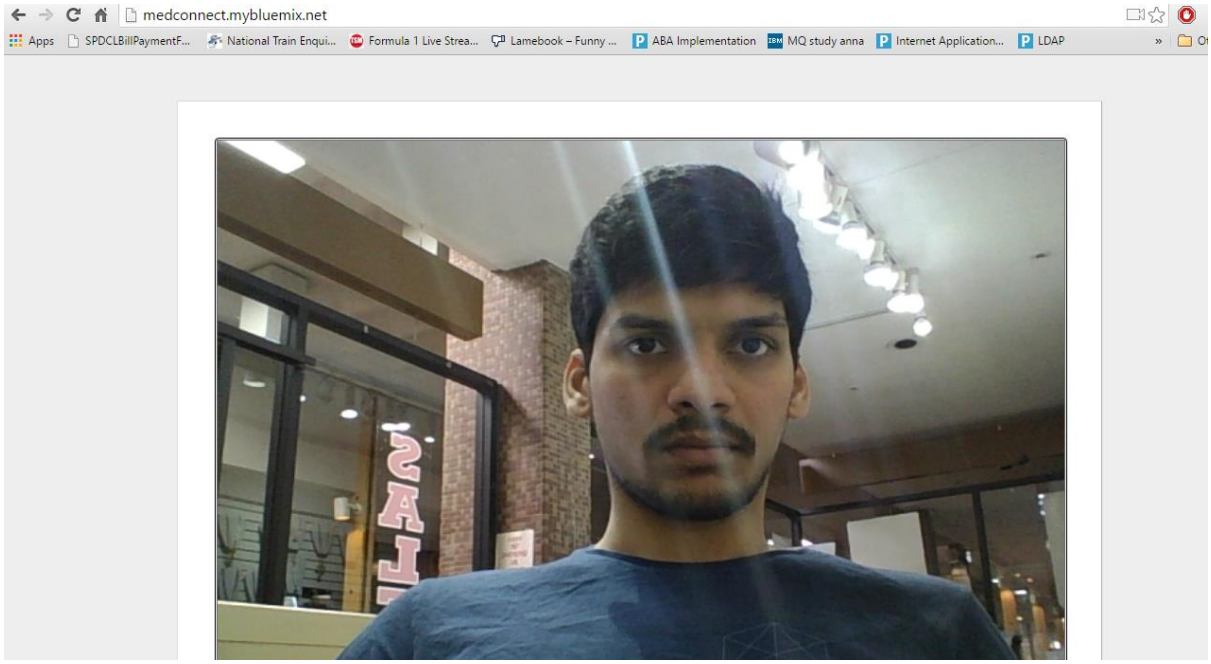


Fig: Video Conferencing from our web app in Laptop

Mobile Client Implementation

We are using Ionic framework for developing our Mobile App. Our Mobile is primarily targeted for Android run time as IOS doesn't support WebRTC yet and building an IOS requires a Mac book which our team doesn't possess. Everything we have developed till now is hybrid and if we have any performance issues regarding the capabilities of a mobile phone we moved to web services where processing will be done on the server and mobile client will be communicating with the server using the REST services.

Plugins used

Cordova Whitelist

Our App will experience 404 errors if we try to make network requests to external urls. In our App we will be using a web view to display the content of our web app deployed in Bluemix. So we have to white list out site by using Cordova whitelist plugin so that ionic will be able display our website.

Code used for whitelisting is

<meta http-equiv="Content-Security-Policy" content="default-src *; style-src 'self' 'unsafe-inline'; script-src 'self' 'unsafe-inline' 'unsafe-eval' http://medconnect.mybluemix.net/;">

Cordova DatePicker

When scheduling an appointment with the doctor user has to pick a date. For allowing the user to pick the date from a modal window we are using Cordova DatePicker plugin.

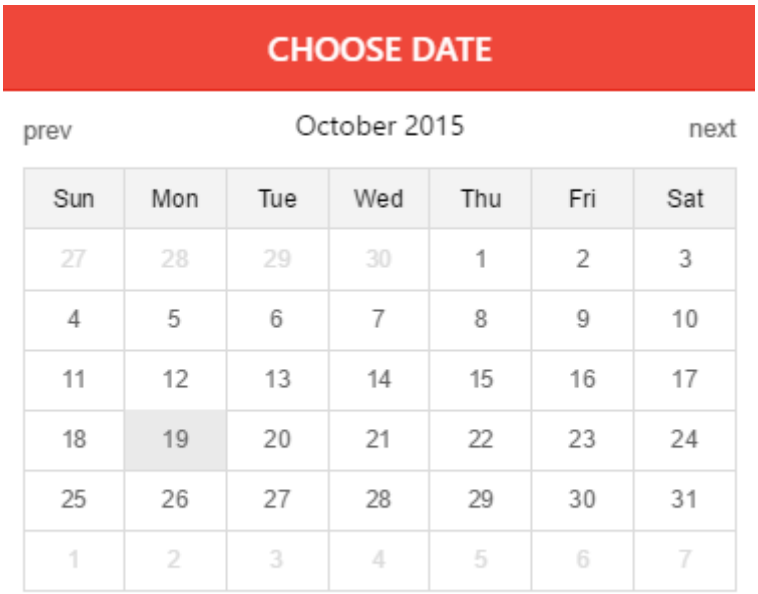


Fig: DatePicker

Java Script Frameworks used

Socket.io

Socket.io enables real time, bi-directional communication between web clients and servers. In our App metadata about the media, client key info has to be shared between the server and client's real time without refreshing. We chose socket.io to provide the best possible output with very less burden on the server and mobile client. For each new client we will be creating a socket and storing them in the firebase and whenever client joins the call, we will indicate the other client has joined and we will start transmitting the video data between the two clients.

openfb.js

By using this framework we can provide the login with Facebook feature for the user, this feature is still in development in our App, but we made some ground on this feature so we are continuing with this and we will complete this by next increment.

Design Patterns Implementation

Module:

Created a config module that unified all the classes related to the socket.io video chat interface.

```
var config = {
    openSocket: function(config) {
        var channel = config.channel || location.href.replace( /\/|:|#|%|\.|\[|\]/g , '');
        var socket = new Firebase('https://webrtc.firebaseIO.com/' + channel);
        socket.channel = channel;
        socket.on("child_added", function(data) {
            config.onmessage && config.onmessage(data.val());
        });
        socket.send = function(data) {
            this.push(data);
        };
        config.onopen && setTimeout(config.onopen, 1);
        socket.onDisconnect().remove();
        return socket;
    },
    onRemoteStream: function(media) {
        var mediaElement = getMediaElement(media.video, {
            width: videosContainer.clientWidth,
            buttons: ['mute-audio']
        });
```

The above code has all the functions related to opening a connection and handling incoming video chat requests.

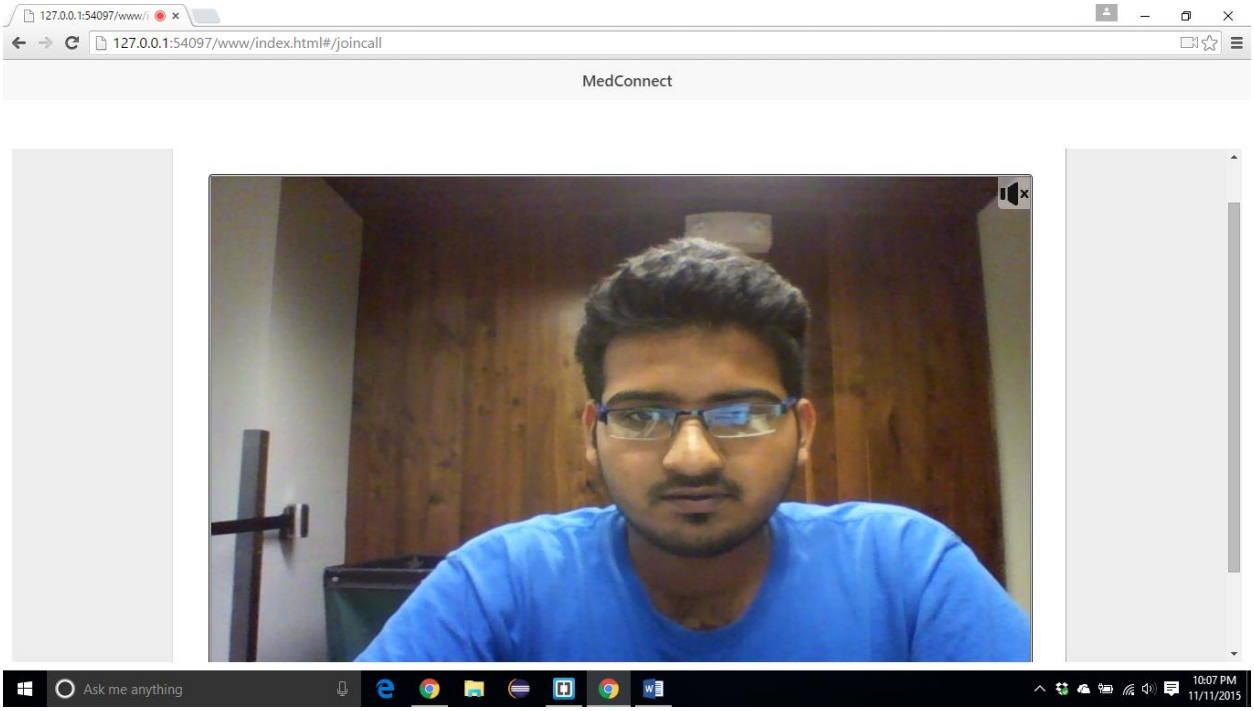
Command:

In our code for displaying the video during the video chat we are using the getMediaElement command which encapsulates all the code related to creating a video element and displaying it to the user. Here command is getMediaElement, invoker is the client who invokes the video chat.

```
var mediaElement = getMediaElement(video, {
    width: (videosContainer.clientWidth),
    buttons: ['mute-audio']
});
```

Null Object:

There are many instances where we implemented null object pattern, one such beautiful instance is when the other party is not available for video chat, the invoker will just see his own video rather than a blank screen indicating that the system is waiting for the other party to join the call.



When there are no clients for his session the presenter will see his own video as default

Singleton:

Whenever user log in to the App all his personal information will be kept inside a singleton abject and this info will be used by all the views.

Observer:

When a new participant joins the call an observer will invoke the services in the client machine and let them know that a new participant has joined the call.

```
onRemoteStreamEnded: function(stream, video) {
    if (video.parentNode && video.parentNode.parentNode && video.parentNode.parentNode.parentNode) {
        video.parentNode.parentNode.parentNode.removeChild(video.parentNode.parentNode);
    }
},
onRoomFound: function(room) {
    if(rooms.length>0) return;
 /*   var tr = document.createElement('tr');
    tr.innerHTML ='<td><button class="join">Join</button></td>';
    roomsList.insertBefore(tr, roomsList.firstChild);
    var joinRoomButton = tr.querySelector('.join');
    joinRoomButton.setAttribute('data-broadcaster', room.broadcaster);
    joinRoomButton.setAttribute('data-roomToken', room.roomToken);*/
    rooms.push( room.broadcaster);
    roomfound = true;
    joinCall(room.roomToken, room.broadcaster);
},
onRoomClosed: function(room) {
```

Factory:
For updating a particular field in the document stored in mongo DB, we created a factory which will decide which field has to be updated based on the changes done on the user.

VII. Deployment:

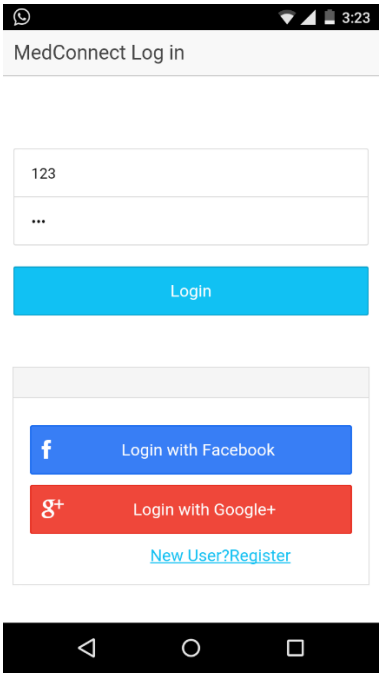
The below is our GitHub link for the project source code.

https://github.com/surapanenipraveen52/MedConnect_ASE/tree/master/MedConnect

The below is our link for the Android APK.

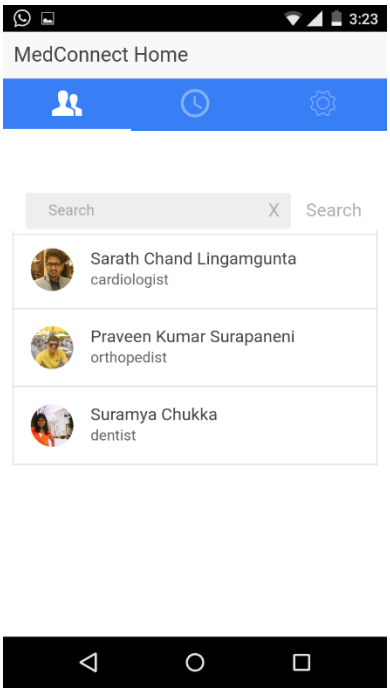
https://onedrive.live.com/redir?resid=4F669F0E90DE6885!21220&authkey=!AAe4DnYun4Ux4sQ&ithint=file%2capk

Demo:

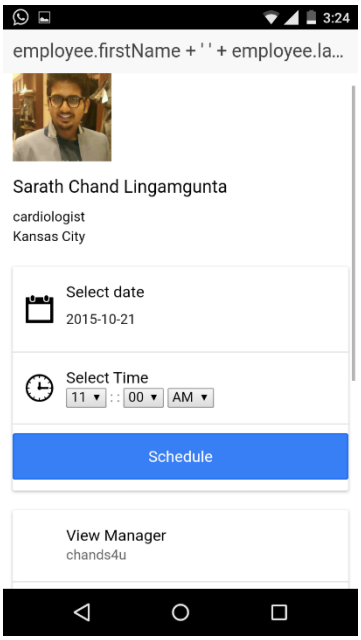


This is the gateway screen to enter into our med connect world. If you are new user then click the bottom link in order to get the entry pass. Then sign in with your

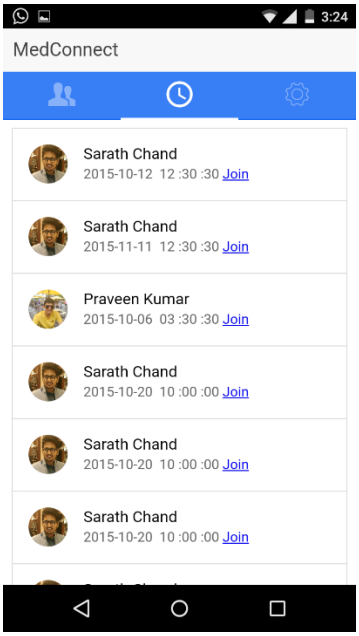
credentials. You can also sign in with your Facebook or google+ credentials.



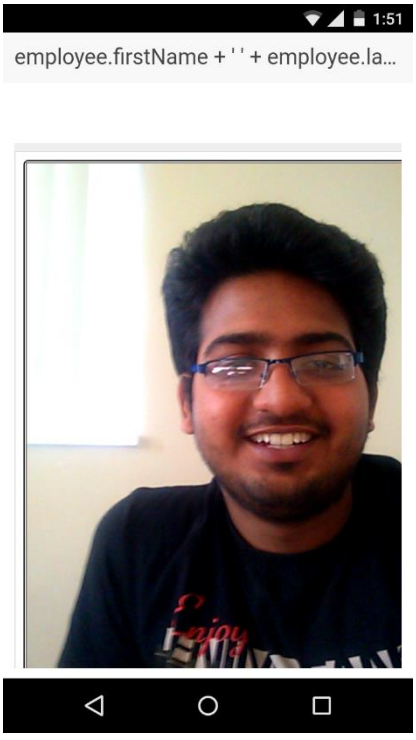
Once a patient logged into his account, he can view list of all doctors who are in the med connect system. He can search the doctor with his name and select a doctor for his appointment.



Once a doctor is selected then we can view his profile as well as we can schedule an appointment.



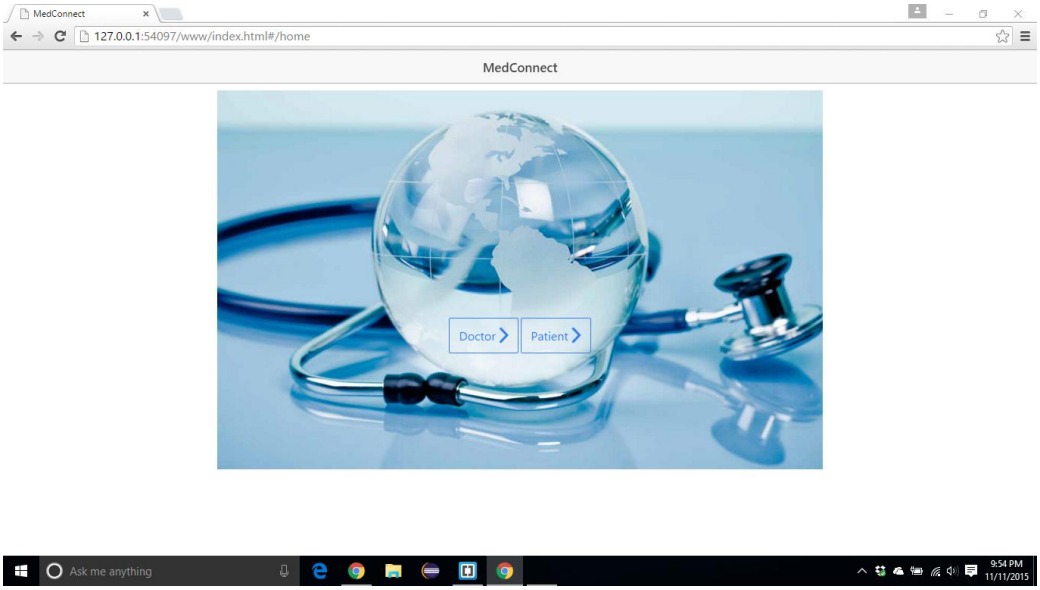
In the second tab we will provide all the scheduled timings. You can join in video conference by clicking join button.



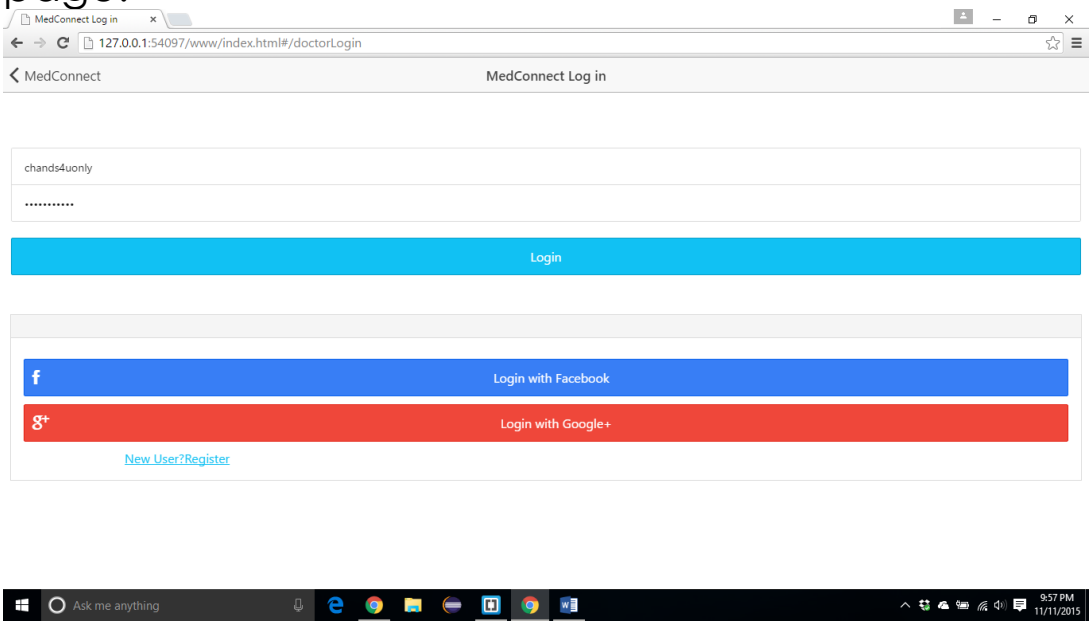
Once the doctor accepts your video request you can view his video until them you will view your video like all other video conference applications.

There is a different flow for doctor:

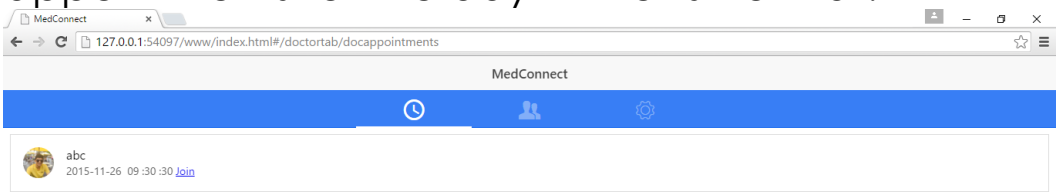
User has to select whether he is a patient or doctor



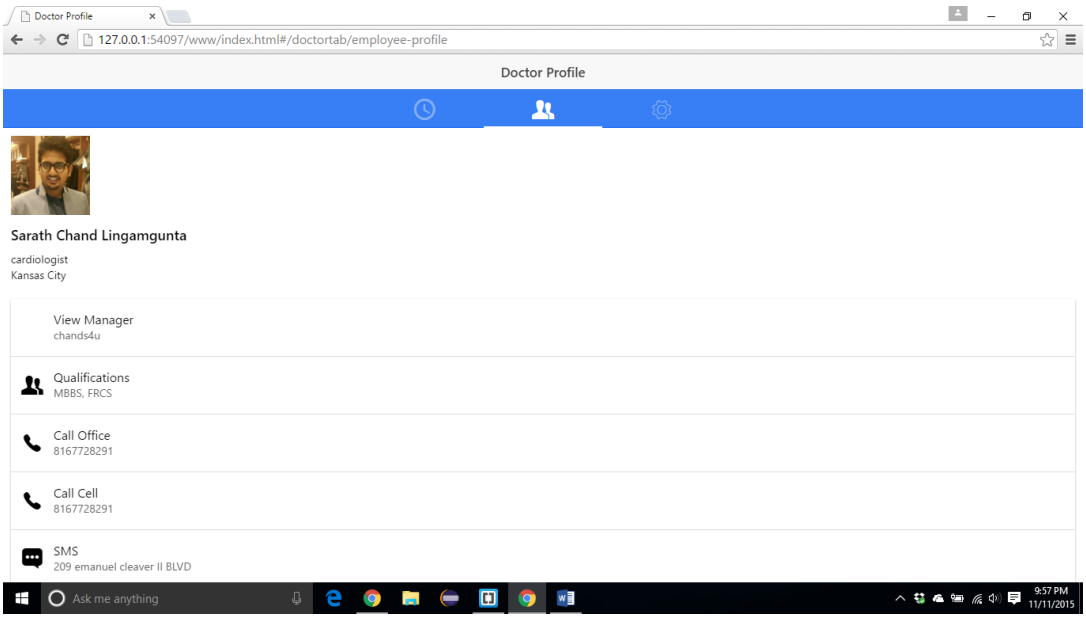
If he selects doctor then he will be routed to doctor login page.



Once the doctor logs in, he can view all the scheduled appointments for the day in the list format.



He can edit his contact information also.



VIII. Project Management:

Here we followed agile methodology to implement our project. In the first iteration we have gathered all the requirements and prepared a project plan.

In second increment, here we started the actual implementation part of our app. As we are following the MVC pattern while coding, we divided our accordingly. All the view part (User Interface) including date picker plug-in is done by Suramya. The model part like connecting to mongo lab and concerns about all the data is done by Sarath Chand. Finally all the controller logic is done by Praveen like implementing video conference and connecting all our individual code into one single meaningful project. While combining the code we exchanged our work and discussed all the features implemented by each individual. So that all the project members would know about every piece of the code that was done.

For third increment we have made some drastic changes to the UI and improved the performance of our server side implementation. Doctor's flow was done by Sarath, Chat was done by Suramya and all the bug fixes, testing and performance fixes are done by praveen

The below is our Kanban link for the project management.

https://frnds.kanbantool.com/b/176260-med-connect#?

Concerns:

As there limited documentation for the ionic, we took more time to debug the errors. Later we overcome these problems by using **AngularJs Batarang**.

IX. Bibliography:

http://www.webrtc.org/native-code/android

https://en.wikipedia.org/wiki/WebRTC

https://frnds.kanbantool.com/dashboard

http://docs.mongolab.com/data-api/

https://www.gliffy.com/uses/uml-software/

http://developer.android.com/guide/topics/providers/calendar-provider.html

http://developer.android.com/index.html

http://www.frengly.com/

http://w3c.github.io/mediacapture-main/archives/20150925/getusermedia.html