

# **International Institute of Information Technology, Bangalore**

**Software Project Engineering Mini Project**

## **Telemedicine**

Under the Guidance of  
Prof. B. Thangaraju and  
Teaching Assistant: Atibhi Agrawal



**Group 10**

Jaymeen Kachrola      Parth Patel  
(MT2020035)      (MT2020057)

# Tables of Content

Title	Page No.
<b>1. Introduction</b>	<b>3</b>
1.1 Overview	3
1.2 Features	3
<b>2. System Configuration</b>	<b>4</b>
2.1 Operating System	4
2.2 CPU and RAM	4
2.3 Language	4
2.4 Kernel Version	4
2.5 Database	4
2.6 Building Tools	4
2.7 DevOps Tools	4
<b>3. Software Development Life Cycle</b>	<b>5</b>
3.1 Source Code	5
3.1.1 Frontend	6
3.1.2 Backend	10
3.1.3 WebRTC Server	11
3.2 Source Code Management (SCM)	12
3.3 Building	13
3.4 Testing	14
3.5 Logging	16
3.6 Docker, Docker Compose and Dockerhub	17
3.7 Ansible	20
3.8 Jenkins and Pipeline	21
3.9 Monitoring using ELK	23
<b>4. Result and Discussion</b>	<b>25</b>
4.1 Login Page	25
4.2 Signup Page	26
4.3 Doctor's Home Page	27
4.4 Patient's Home Page	27
4.5 Profile Update Page	28
4.6 Doctor Consulting Patient	28
4.7 Doctor Writing Prescription for Patient	30
<b>5. Scope for future work</b>	<b>31</b>
<b>6. Conclusion</b>	<b>32</b>
<b>7. References</b>	<b>33</b>

# 1. Introduction

## 1.1 OverView

Our project idea was to design a system where doctors can meet/consult patients remotely over a video/audio call. As we know there are many already available solutions where doctors and patient can consult over a video call (Hangouts, Skype, Zoom, Google Meet etc.), what our design provides which is unique/different from these already available solutions ?

## 1.2 Features

### Peer to Peer Connection

- For WebRTC connection, what we will need is a signaling server. The job of this signaling server would be to help the peers connect before establishing a peer to peer connection.
- Once the peer to peer connection is established, the signaling server will become obsolete and it won't be required. Even if you disconnect the signaling server while the video call is still going on, you won't face any kind of disconnections.
- Once the connection is established, the entire thing becomes peer to peer so we can have a secure video chat without worrying about the security issues.

### Multi-Stream Video Conferences

- In this app, any user can add multiple streams over the same connection. Why would we require such a functionality ?
- With this functionality in the patient side there can be devices connected to his/her PC and doctor can see output of it while having conversation with the patient simultaneously.

### No requirement of sharing ID's

- One another feature of this app is there no requirement to share ID's with which the doctors and patient gets connected.
- When the patient calls any doctor, the notification reaches to the doctor in real time (if he is logged in, in case the doctor is not logged in, the call will be stored in backend as pending and the notification will reach doctor when he logs in) and he/she has to just click a button to accept/reject calls.

## **2. System Configuration**

### **2.1 Operating system**

- Ubuntu 20.04 Focal Fossa

### **2.2 CPU and RAM**

- 2 core processor and RAM 8 GB

### **2.3 Language**

- JavaScript, TypeScript, Angular web framework, Mocha Testing Library and Winston Logging Library

### **2.4 Kernel Version**

- Linux Machine 5.4.0-28-generic

### **2.5 Database**

- MySQL 8.0.20 (MySQL Community Server - GPL)

### **2.6 Building Tools**

- Node for build frontend

### **2.7 DevOps Tools**

- Source Control Management - GitHub
- Continuous Integration - Jenkins
- Containerization – Docker Compose
- Continuous deployment - Ansible
- Monitoring - ELK Stack (Elastic Search, Logstash, Kibana)

### 3. Software Development life cycle

#### 3.1 Source Code

We have following project structure. There are 4 folder namely ansible\_deployment, Backend, Ng-frontend, WebRTC\_Signaling\_Server in entire project. Backend, Ng-frontend, and WebRTC\_Signaling\_Server contain main application code and ansible\_deployment contain deployment related code. Let's try to understand each folder step by step.

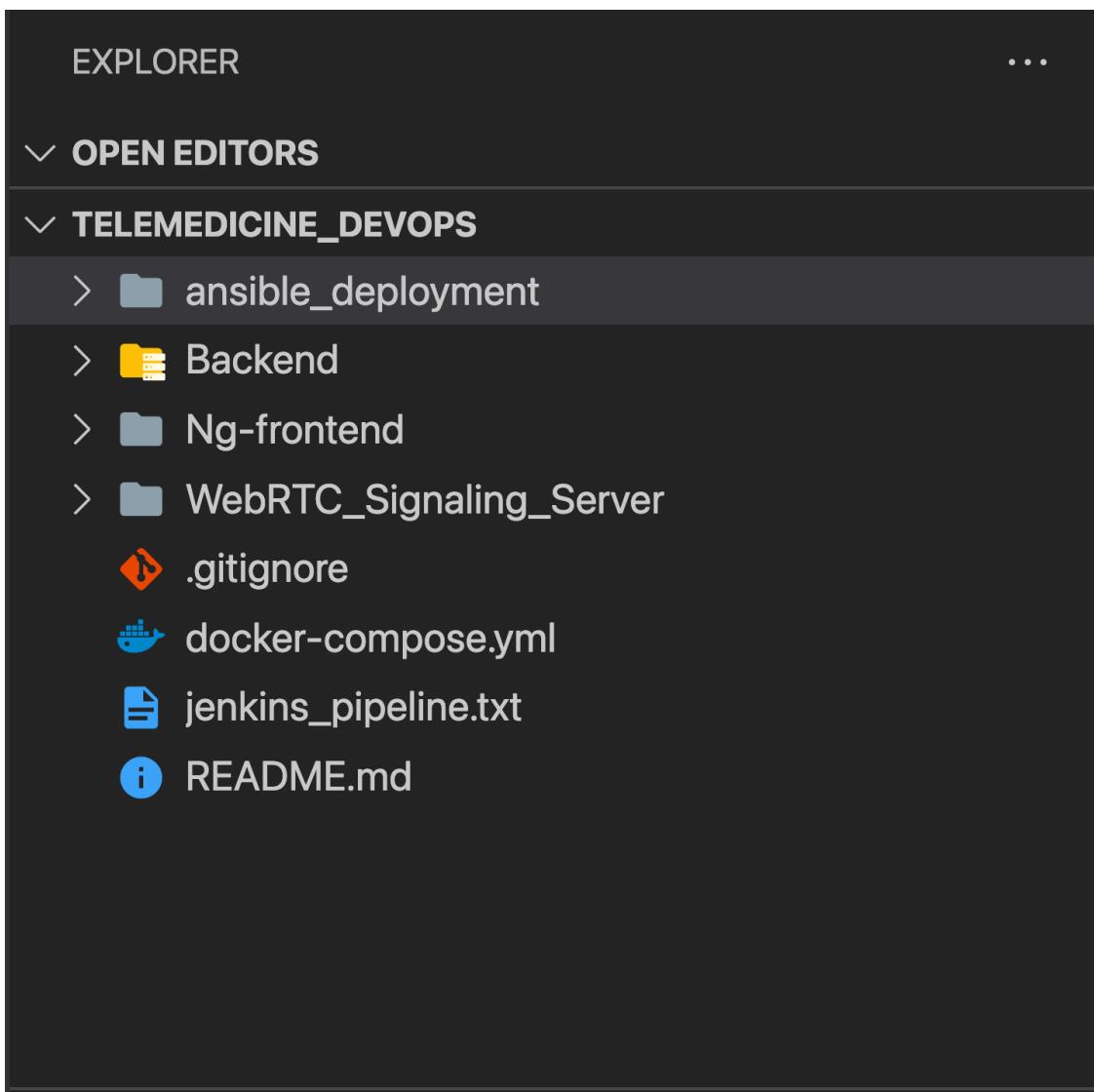


Figure 1 Project Structure

### 3.1.1 Frontend

Ng-frontend contains the code of frontend. Angular framework has been used for creating the frontend.

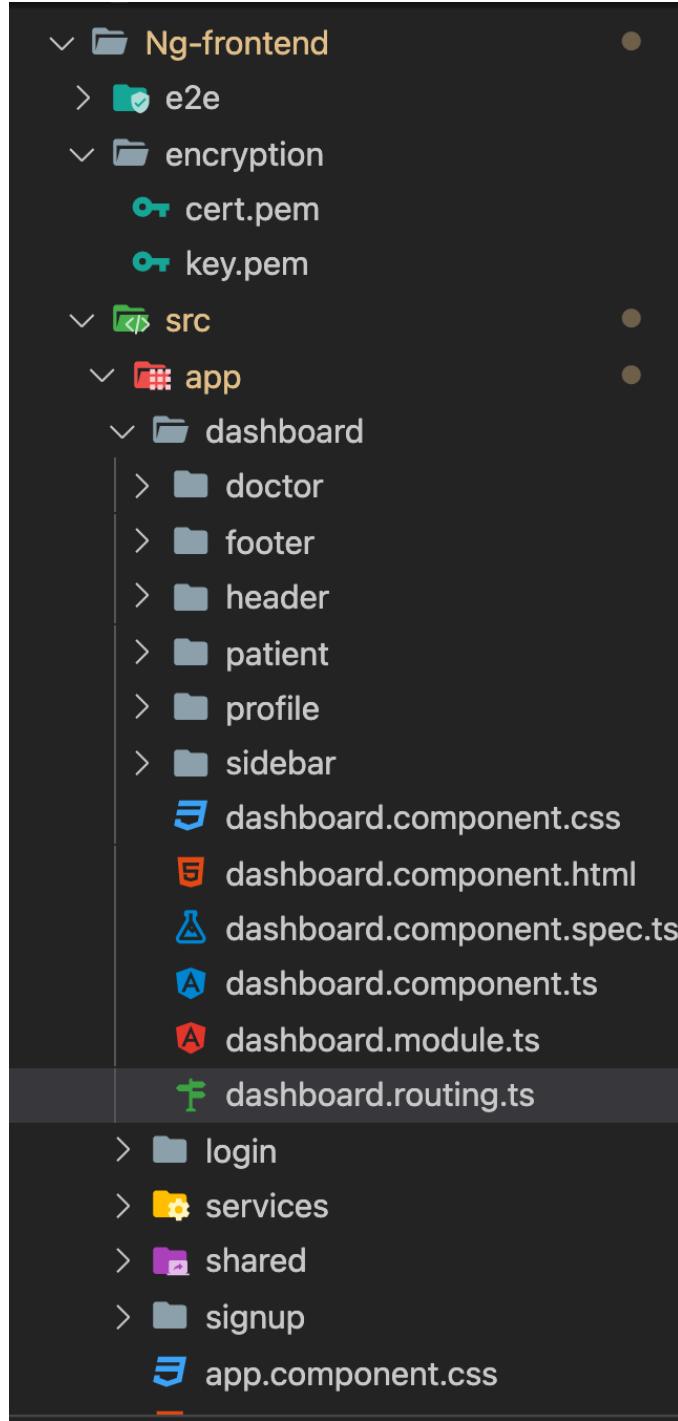
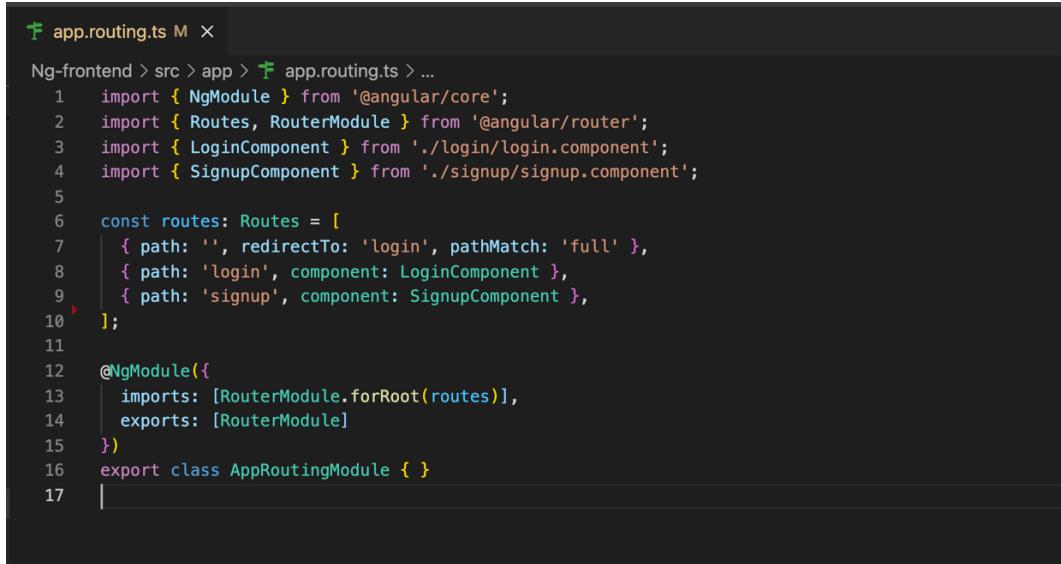


Figure 2 Frontend Components and Module

Above Image shows the list of component and modules that are there in the frontend. Also we have self singed ssl certificate which is essential for our application to work as it is accessing the Microphone and Web Camera of the computer. Following images show the different routes and the services of the frontend.



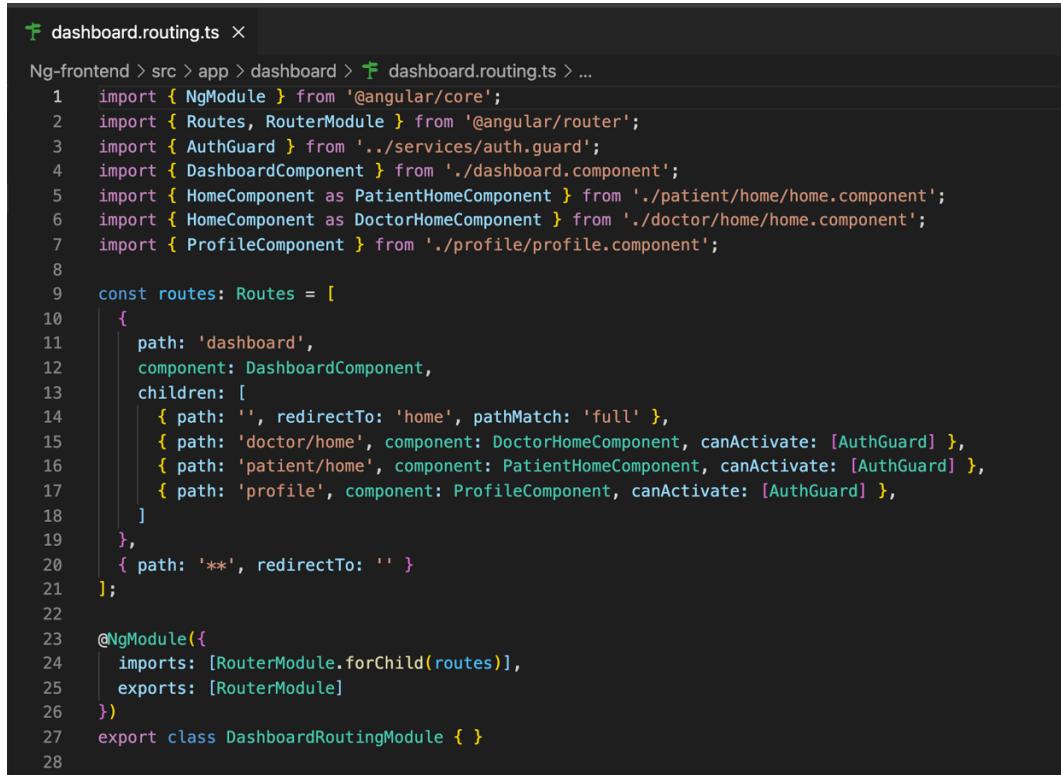
```

app.routing.ts M ×

Ng-frontend > src > app > app.routing.ts > ...
1 import { NgModule } from '@angular/core';
2 import { Routes, RouterModule } from '@angular/router';
3 import { LoginComponent } from './login/login.component';
4 import { SignupComponent } from './signup/signup.component';
5
6 const routes: Routes = [
7   { path: '', redirectTo: 'login', pathMatch: 'full' },
8   { path: 'login', component: LoginComponent },
9   { path: 'signup', component: SignupComponent },
10 ];
11
12 @NgModule({
13   imports: [RouterModule.forRoot(routes)],
14   exports: [RouterModule]
15 })
16 export class AppRoutingModule { }
17

```

Figure 3 App Routing



```

dashboard.routing.ts X

Ng-frontend > src > app > dashboard > dashboard.routing.ts > ...
1 import { NgModule } from '@angular/core';
2 import { Routes, RouterModule } from '@angular/router';
3 import { AuthGuard } from '../services/auth.guard';
4 import { DashboardComponent } from './dashboard.component';
5 import { HomeComponent as PatientHomeComponent } from './patient/home/home.component';
6 import { HomeComponent as DoctorHomeComponent } from './doctor/home/home.component';
7 import { ProfileComponent } from './profile/profile.component';
8
9 const routes: Routes = [
10   {
11     path: 'dashboard',
12     component: DashboardComponent,
13     children: [
14       { path: '', redirectTo: 'home', pathMatch: 'full' },
15       { path: 'doctor/home', component: DoctorHomeComponent, canActivate: [AuthGuard] },
16       { path: 'patient/home', component: PatientHomeComponent, canActivate: [AuthGuard] },
17       { path: 'profile', component: ProfileComponent, canActivate: [AuthGuard] },
18     ]
19   },
20   { path: '**', redirectTo: '' }
21 ];
22
23 @NgModule({
24   imports: [RouterModule.forChild(routes)],
25   exports: [RouterModule]
26 })
27 export class DashboardRoutingModule { }
28

```

Figure 4 Dashboard Routing

```

auth-service.service.ts ×
Ng-frontend > src > app > services > auth-service.service.ts ...
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { BehaviorSubject, Observable } from 'rxjs';
4 import { baseUrl, webRtcServerUrl } from 'src/environments/environment';
5 import { map } from 'rxjs/operators';
6 import { WebRtcService } from './webrtc.service';
7
8
9 @Injectable({
10   providedIn: 'root'
11 })
12 export class AuthServiceService {
13
14   private currentUserSubject: BehaviorSubject<any>;
15   public currentUser: Observable<any>;
16
17   constructor(private http: HttpClient) {
18     this.currentUserSubject = new BehaviorSubject(JSON.parse(localStorage.getItem('currentUser')));
19     this.currentUser = this.currentUserSubject.asObservable();
20   }
21
22   public get getCurrentUser() {
23     this.currentUserSubject = new BehaviorSubject(JSON.parse(localStorage.getItem('currentUser')));
24     this.currentUser = this.currentUserSubject.asObservable();
25     return this.currentUserSubject.value;
26   }
27
28   validate(): Observable<any> {
29     return this.http.get(`${baseUrl}users/validate`);
30   }
31
32   login(data): Observable<any> {
33     return this.http.post(`${baseUrl}users/login`, data)
34       .pipe(map(results => {
35         if (results['success']) {

```

Figure 5 Auth Service

```

users.service.ts ×
Ng-frontend > src > app > services > users.service.ts ...
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4 import { map } from 'rxjs/operators';
5 import { baseUrl } from 'src/environments/environment';
6
7 @Injectable({
8   providedIn: 'root'
9 })
10 export class UsersService {
11
12   constructor(private http: HttpClient) {}
13
14   update(data): Observable<any> {
15     return this.http.patch(`${baseUrl}users/`, data)
16       .pipe(map(data => {
17         return data;
18       }));
19   }
20
21   getDoctors(specialization) {
22     return this.http.post(`${baseUrl}users/getdoctors/`, { 'specialization': specialization });
23   }
24
25   getWaitingPatients(doctor_id){
26     return this.http.post(`${baseUrl}users/getwaitingpatients/`, { 'doctor_id': doctor_id });
27   }
28
29   removePatientFromWaitlist(patient_room_id){
30     return this.http.post(`${baseUrl}users/removefromwaitlist/`, { 'room_id' : patient_room_id });
31   }
32
33   addPatientToWaitList(doctor_id, roomId){
34     const currentUser = JSON.parse(localStorage.getItem('currentUser'));
35     const patient_id = currentUser['id'];

```

Figure 6 User Service

```
webrtc.service.ts ×
Ng-frontend > src > app > services > webrtc.service.ts > ...
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { webrtcServerUrl } from 'src/environments/environment';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class WebrtcService {
9
10   constructor(private http: HttpClient) { }
11
12   createRoom() {
13     return this.http.get(` ${webrtcServerUrl}/createRoom`);
14   }
15
16   joinRoom(roomId) {
17     return this.http.get(` ${webrtcServerUrl}/joinRoom?roomId=${roomId}`);
18   }
19
20   createNamespace(namespace_id) {
21     return this.http.get(` ${webrtcServerUrl}/createNamespace?namespace_id=${namespace_id}`);
22   }
23
24   removeNamespace(namespace_id) {
25     return this.http.get(` ${webrtcServerUrl}/removeNamespace?namespace_id=${namespace_id}`);
26   }
27
28   checkAvailabilityOfDoctor(namespace_id){
29     return this.http.get(` ${webrtcServerUrl}/checkAvailability?namespace_id=${namespace_id}`);
30   }
31
32 }
33
```

Figure 7 webrtc service

### 3.1.2 Backend

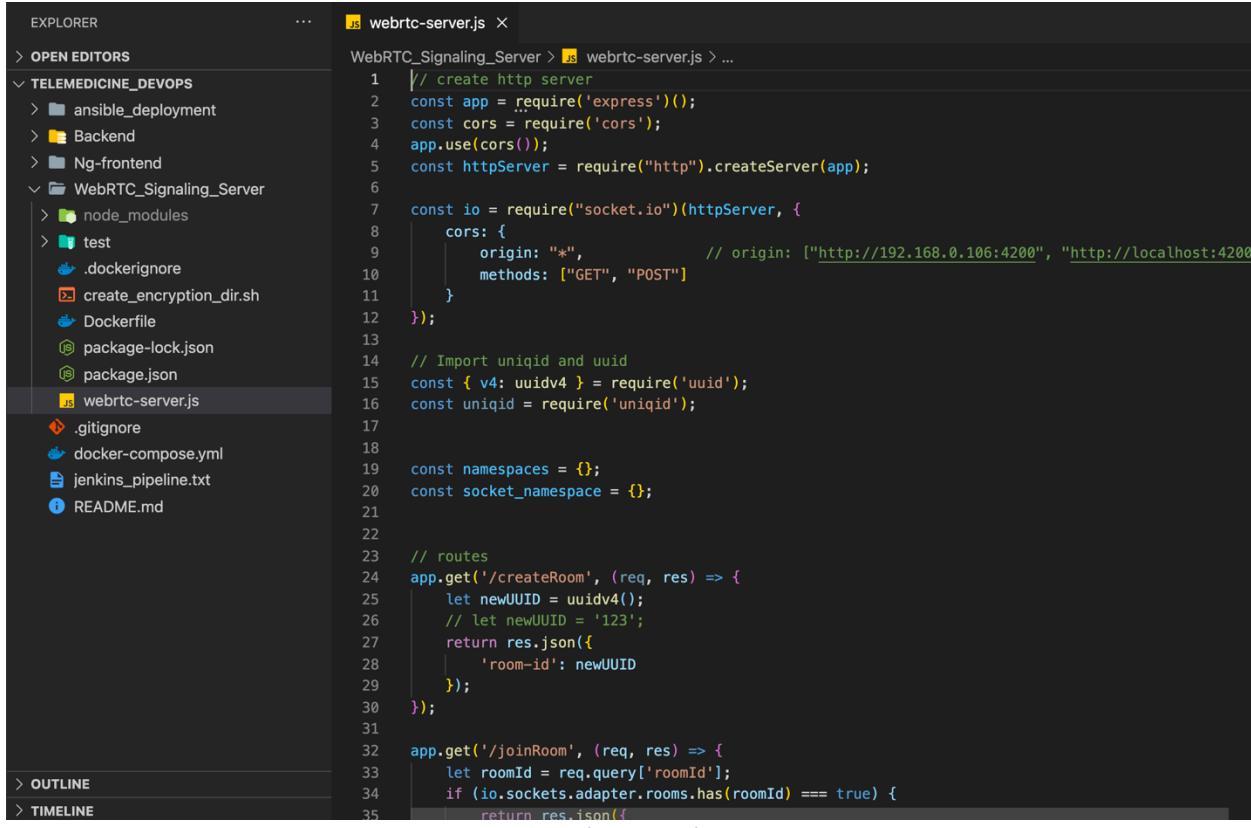
For the backend we have used express and node. Following figure shows the list of files and middleware (left panel) we have used in the backend and main app.js file. It also contains dockerfile which is used for docker containerization. Backend is used for Authentication and CRUD operation associated with the application.

```
Backend > app.js > ...
1 const logger = require('./logger')
2 require("dotenv").config();
3 const express = require("express");
4 const cors = require('cors');
5 const userRouter = require("./users/user.router");
6
7
8 const app = express();
9
10 app.use(cors());
11 app.use(express.json());
12 app.use("/api/users", userRouter);
13
14
15 const mysql = require('mysql');
16 const con = mysql.createConnection({
17   host: process.env.DB_HOST,
18   user: process.env.DB_USER,
19   password: process.env.DB_PWD
20 });
21
22 // Create database and table if not exists.
23 con.connect(function (err) {
24   if (err) throw err;
25
26   con.query(`CREATE DATABASE IF NOT EXISTS \`${process.env.MYSQL_DB}\``, function (err, result) {
27     if (err) throw err;
28   });
29
30
31   stmt = `CREATE TABLE IF NOT EXISTS \`${process.env.MYSQL_DB}\`.\`user_doctor\` (`;
32   ```id` INT NOT NULL AUTO_INCREMENT," +
33   ```firstname` VARCHAR(45) NOT NULL," +
34   ```lastname` VARCHAR(45) NULL," +
35   ```email` VARCHAR(45) NOT NULL," +
```

*Figure 8 Backend and app.js*

### 3.1.3 WebRTC Signaling Server

For the WebRTC Signaling Server, we have used express and node. It has a single file `webrtc-server.js`, which is useful for establishing the connection between peers (Doctor and Patient). It also contains dockerfile which is used for docker containerization.



```
// create http server
const app = require('express')();
const cors = require('cors');
app.use(cors());
const httpServer = require("http").createServer(app);

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

// Import uniqid and uuid
const { v4: uuidv4 } = require('uuid');
const uniqid = require('uniqid');

const namespaces = {};
const socket_namespace = {};

// routes
app.get('/createRoom', (req, res) => {
  let newUUID = uuidv4();
  // let newUUID = '123';
  return res.json({
    'room-id': newUUID
  });
});

app.get('/joinRoom', (req, res) => {
  let roomId = req.query['roomId'];
  if (io.sockets.adapter.rooms.has(roomId) === true) {
    return res.json({
      'status': 'success'
    });
  } else {
    return res.json({
      'status': 'failure'
    });
  }
});
```

Figure 9 WebRTC signaling server

## 3.2 Source Code Management (SCM)

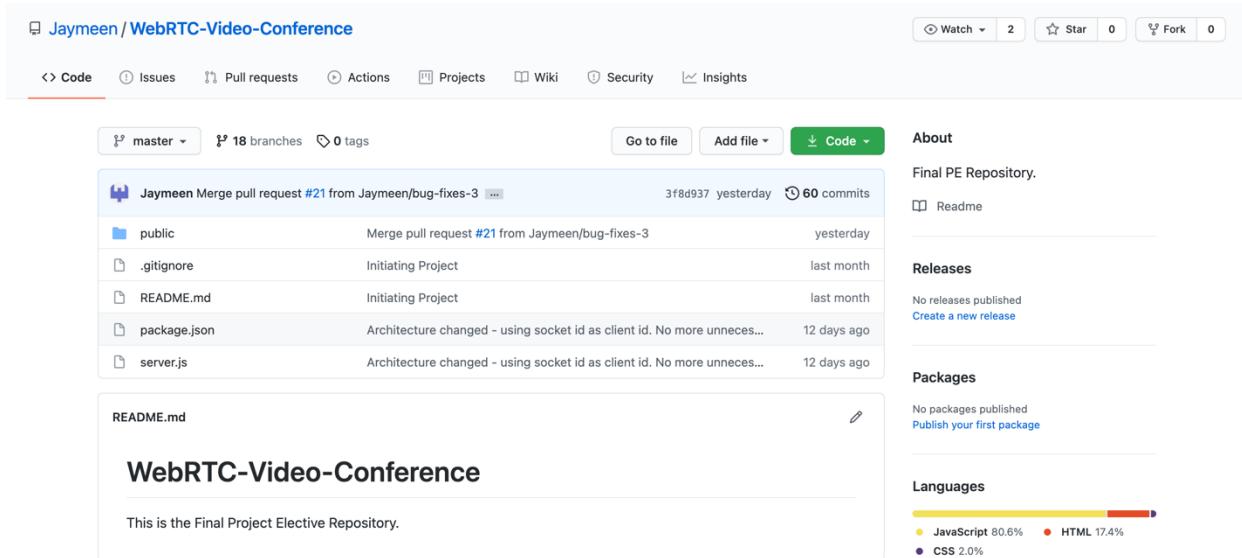


Figure 10 WebRTC-Video-Conference

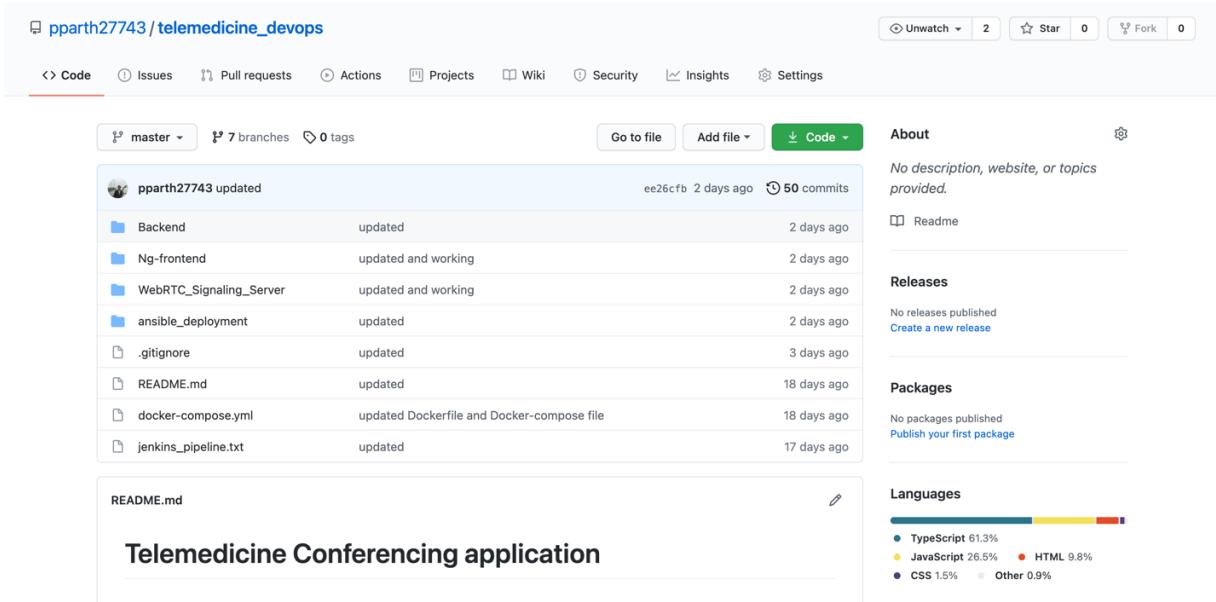


Figure 11 Full-Fledged Angular WebRTC Application

Following is the GitHub repository for of the Project, WebRTC-Video-Conference: <https://github.com/Jaymeen/WebRTC-Video-Conference>

Full-Fledged Angular WebRTC Application:  
[https://github.com/pparth27743/telemedicine\\_devops](https://github.com/pparth27743/telemedicine_devops)

### 3.3 Building

We have used node for building our frontend application. Following figure shows the build files of frontend.

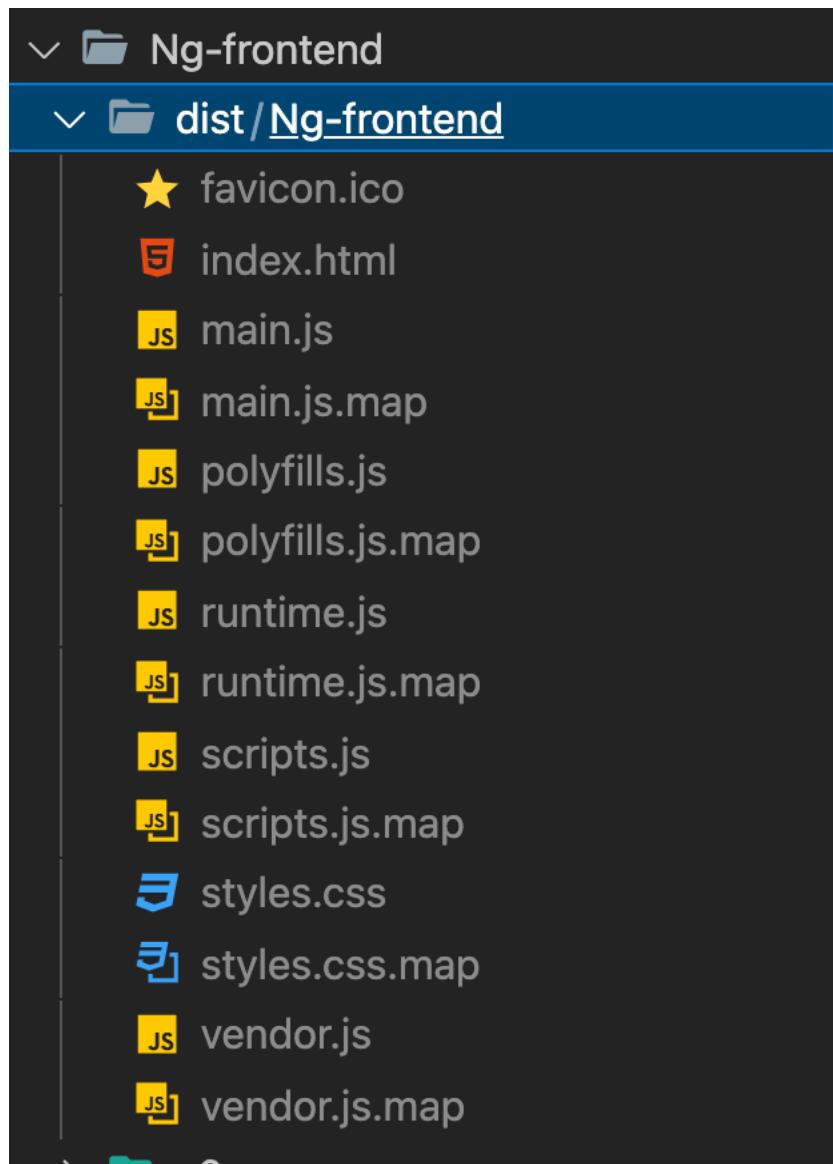
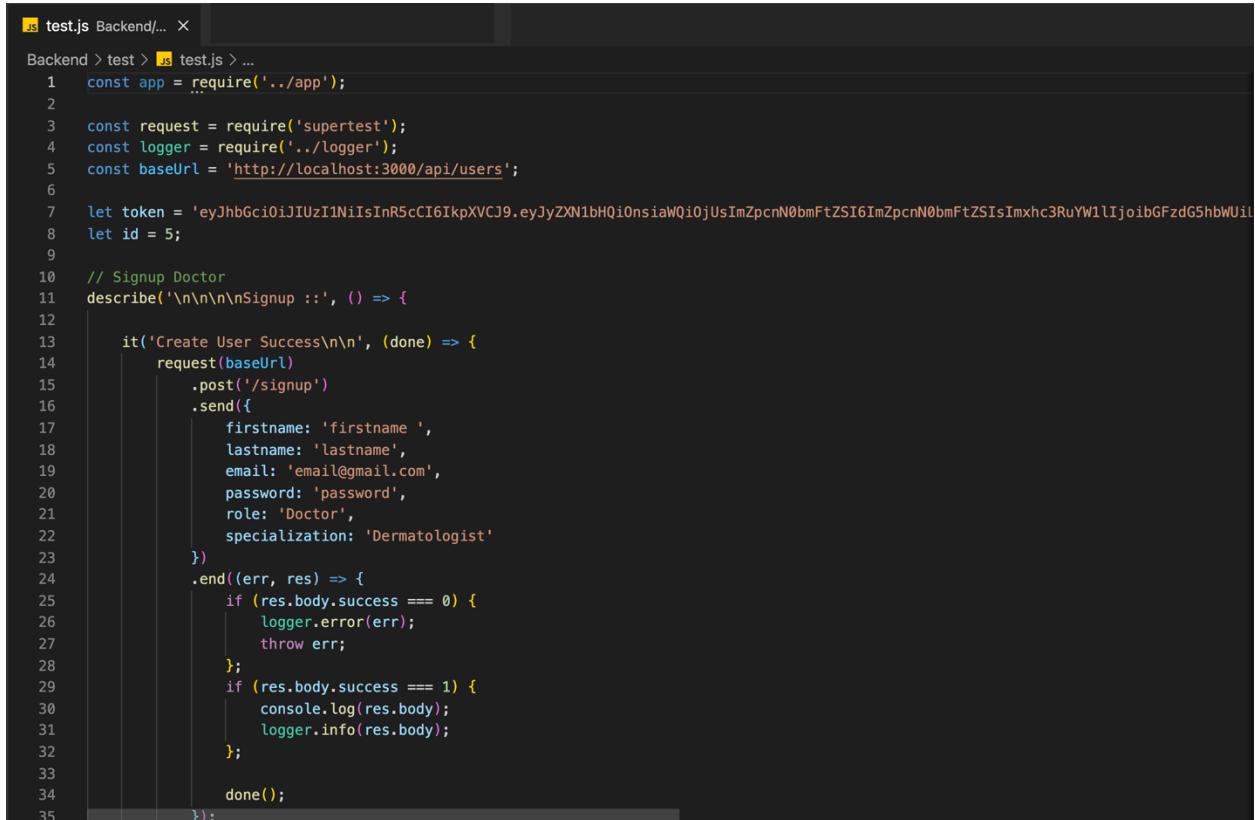


Figure 12 Build of frontend

## 3.4 Testing

We have used mocha which is a testing JavaScript library. We have done testing of various functionality related to CRUD operation and Authentication in the Backend and also testing of WebRTC server functionality such as create room, checking room availability, adding and removing namespace of socket.io etc.,



```
js test.js Backend... ×
Backend > test > js test.js > ...
1  const app = require('../app');
2
3  const request = require('supertest');
4  const logger = require('../logger');
5  const baseUrl = 'http://localhost:3000/api/users';
6
7  let token = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJxN1bHQiOjUsImZpcnN0bmFtZSI6ImZpcnN0bmFtZSIiImxhc3RuYW1lIjoibGFzdG5hbWUi';
8  let id = 5;
9
10 // Signup Doctor
11 describe('Signup Doctor', () => {
12
13   it('Create User Success\n\n', (done) => {
14     request(baseUrl)
15       .post('/signup')
16       .send({
17         firstname: 'firstname',
18         lastname: 'lastname',
19         email: 'email@gmail.com',
20         password: 'password',
21         role: 'Doctor',
22         specialization: 'Dermatologist'
23       })
24       .end((err, res) => {
25         if (res.body.success === 0) {
26           logger.error(err);
27           throw err;
28         }
29         if (res.body.success === 1) {
30           console.log(res.body);
31           logger.info(res.body);
32         }
33         done();
34       });
35   });
36 });
37 
```

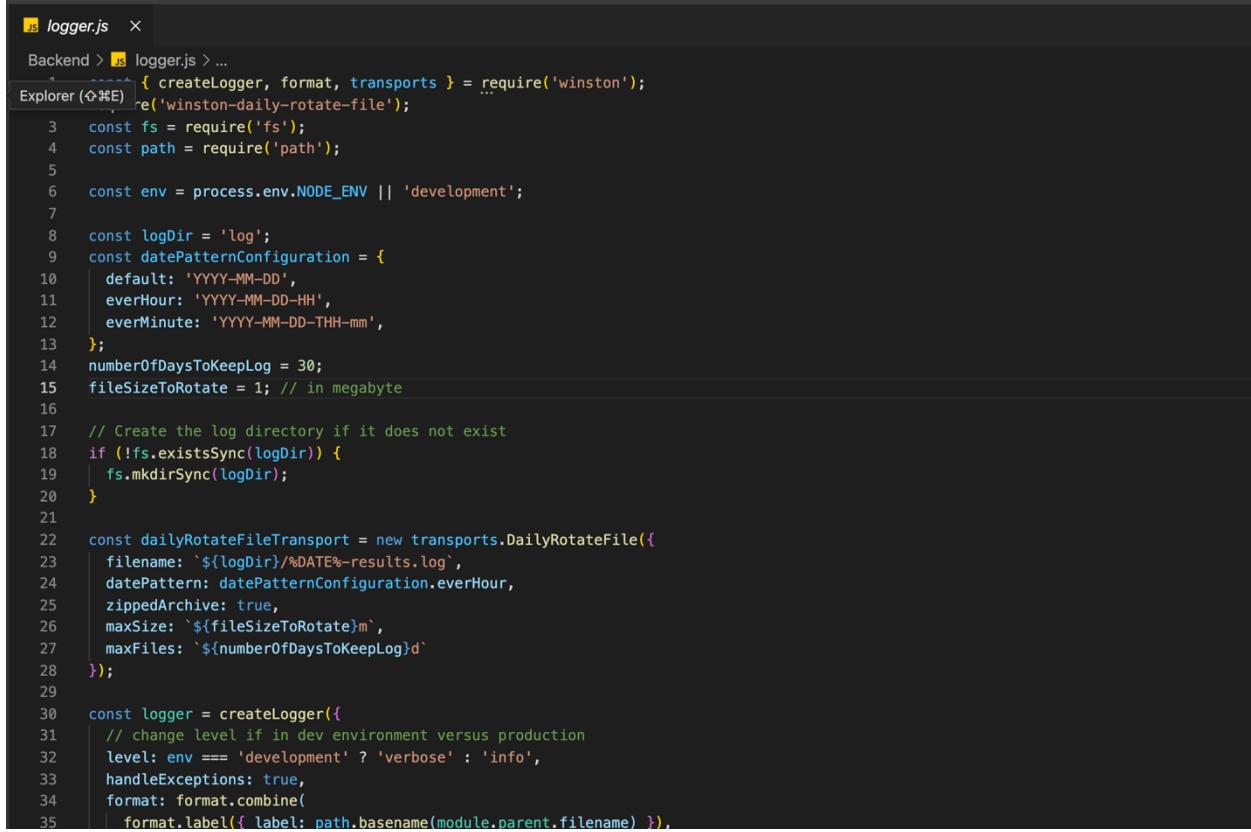
Figure 13 Backend test.js

```
  js test.js WebRTC_Signaling_Server/... ×
WebRTC_Signaling_Server > test > js test.js > ...
1  const app = require('../webrtc-server');
2
3  const request = require('supertest');
4  const baseUrl = 'http://localhost:4440';
5
6  let roomId = '123';
7
8
9  // Get Room Id
10 describe(`\n\nGet Room Id:: `, () => {
11   it('Get Room Id \n', (done) => {
12
13     request(baseUrl)
14       .get('/createRoom')
15       .end((err, res) => {
16         if (!res.body['room-id']) throw err;
17         else {
18           console.log(res.body);
19           roomId = res.body['room-id'];
20         }
21       })
22     });
23   });
24 });
25
26 // Join Room
27 describe(`\n\nJoin Room :: `, () => {
28
29   it('Join Room with given Id \n', (done) => {
30     request(baseUrl)
31       .get(`/joinRoom?roomId=${roomId}`)
32       .end((err, res) => {
33         console.log(res.body);
34         done();
35       });
36   });
37 });
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
```

Figure 14 WebRTC Server test.js

## 3.5 Logging

We have used Winston, which is logging JavaScript library. We have logged each and every event of the Backend that is happening there. These log files created later used in ELK for monitoring of the application.



The screenshot shows a code editor window with the file 'logger.js' open. The code is written in JavaScript and uses the Winston library for logging. It includes configuration for log rotation, creation of log directories, and setting up a logger object.

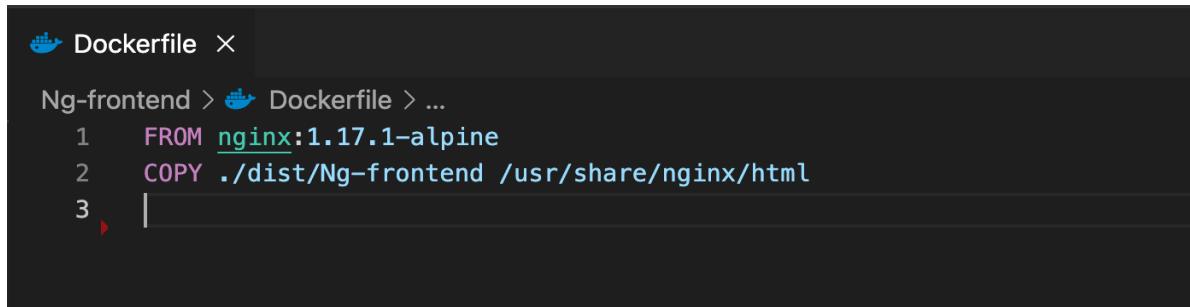
```
js logger.js ×
Backend > logger.js > ...
1  import { createLogger, format, transports } = require('winston');
2  import { winstonDailyRotateFile } from 'winston-daily-rotate-file';
3  const fs = require('fs');
4  const path = require('path');
5
6  const env = process.env.NODE_ENV || 'development';
7
8  const logDir = 'log';
9  const datePatternConfiguration = {
10    default: 'YYYY-MM-DD',
11    everHour: 'YYYY-MM-DD-HH',
12    everMinute: 'YYYY-MM-DD-THH-mm',
13  };
14  numberOfWorksToKeepLog = 30;
15  fileSizeToRotate = 1; // in megabyte
16
17  // Create the log directory if it does not exist
18  if (!fs.existsSync(logDir)) {
19    fs.mkdirSync(logDir);
20  }
21
22  const dailyRotateFileTransport = new transports.DailyRotateFile({
23    filename: `${logDir}/%DATE%-results.log`,
24    datePattern: datePatternConfiguration.everHour,
25    zippedArchive: true,
26    maxSize: `${fileSizeToRotate}m`,
27    maxFiles: `${numberOfWorksToKeepLog}d`
28  });
29
30  const logger = createLogger({
31    // change level if in dev environment versus production
32    level: env === 'development' ? 'verbose' : 'info',
33    handleExceptions: true,
34    format: format.combine(
35      format.label({ label: path.basename(module.parent.filename) })
    
```

Figure 15 logger.js

### 3.6 Docker, Docker Compose and DokcerHub

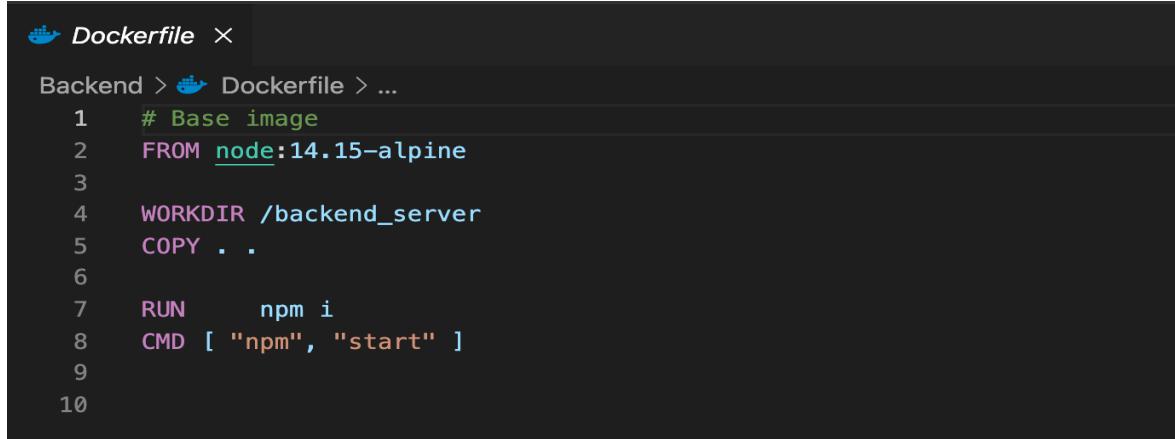
Docker container is popularly known as a lightweight virtual machine that only contains the application and its dependencies. On a machine, we can run thousands of containers whereas fewer Virtual Machines can be created on the machine. To create a docker container of our application we need to create a Docker Image and for that, we first need to create Dockerfile which has all the instructions and dependencies that are downloaded when the image is built.

There are 3 different Dockerfile we have created for frontend, backend and webrtc-signaling-server as follows.



```
Ng-frontend > Dockerfile > ...
1  FROM nginx:1.17.1-alpine
2  COPY ./dist/Ng-frontend /usr/share/nginx/html
3  |
```

Figure 16 Dockerfile for frontend



```
Backend > Dockerfile > ...
1  # Base image
2  FROM node:14.15-alpine
3
4  WORKDIR /backend_server
5  COPY . .
6
7  RUN npm i
8  CMD [ "npm", "start" ]
9
10
```

Figure 17 Dockerfile for Backend

```
# Base image
FROM node:14.15-alpine
WORKDIR /webrtc_server
COPY . .
RUN npm i
CMD [ "npm", "start" ]
```

Figure 18 Dockerfile for WebRTC Server

```
Specify the set of services that your app is composed of.

services:
  db:
    container_name: db
    image: mysql:5.7
    command: --default-authentication-plugin=mysql_native_password
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: 'rootroot'
      MYSQL_ROOT_HOST: '%'
    volumes:
      - db-data:/var/lib/mysql
  frontend:
    container_name: frontend
    restart: always
    build: ./Ng-frontend/
    ports:
      - "4200:80"
    depends_on:
      - backend
      - webrtc_server
  backend:
    container_name: backend
    restart: always
    build: ./Backend/
    ports:
      - "3000:3000"
    depends_on:
      - db
  webrtc_server:
    container_name: webrtc_server
    restart: always
    build: ./WebRTC_Signaling_Server/
    ports:
      - "4440:4440"
volumes:
  db-data:
```

Figure 19 Docker Compose File

Docker compose is used to do docker containerization. Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration[6].

Docker Compose file is creating 4 services namely db, frontend, backend and webrtc\_server. Here we used docker volume for persistent storage which is attached to db (database) service.

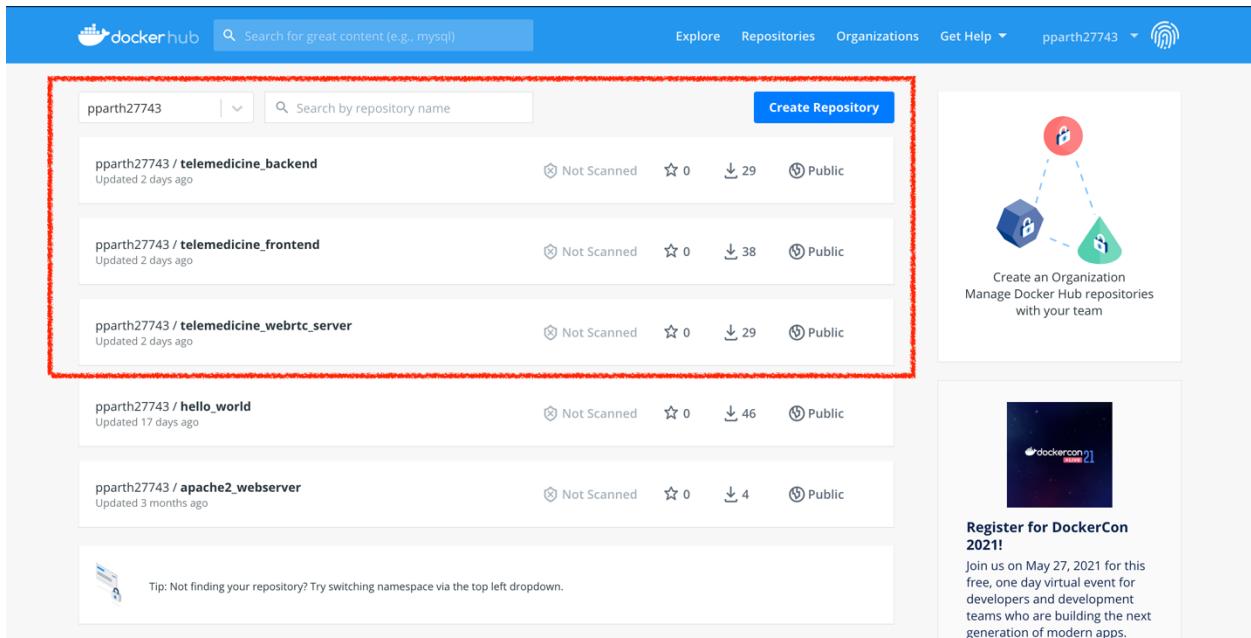


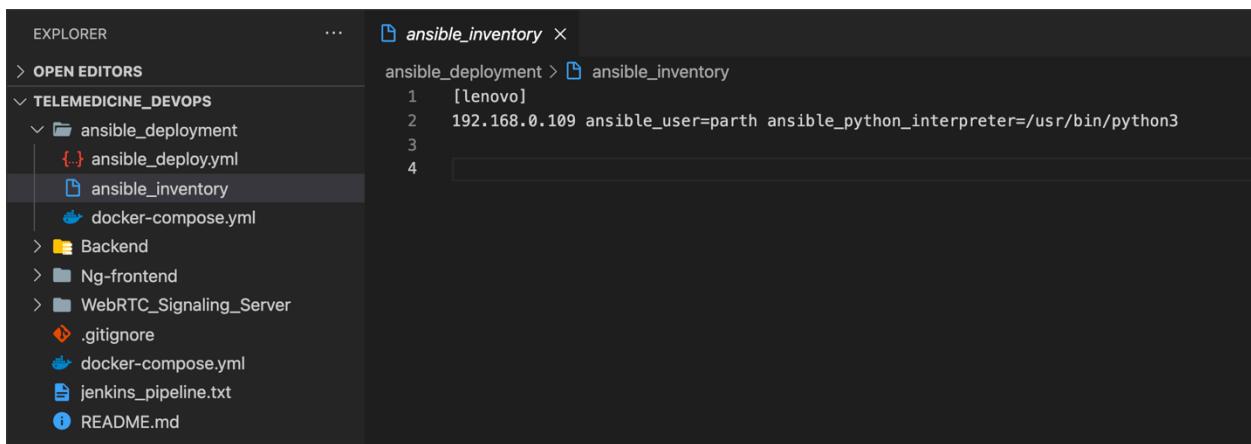
Figure 20 Docker Images uploaded on DockerHub

### 3.7 Ansible

Ansible is a simple open-source IT engine that automates application deployment, intra service orchestration, cloud provisioning, and many other IT tools.

Ansible is easy to deploy as it is agentless tool. It only needs to be installed on the controlled node. The managed node need not install ansible in it. Ansible uses a playbook to define tasks that need to be performed on the managed host and these tasks are known as play in an ansible world. Ansible uses ssh for logging into the managed host. And this host information is mentioned inside the inventory file.

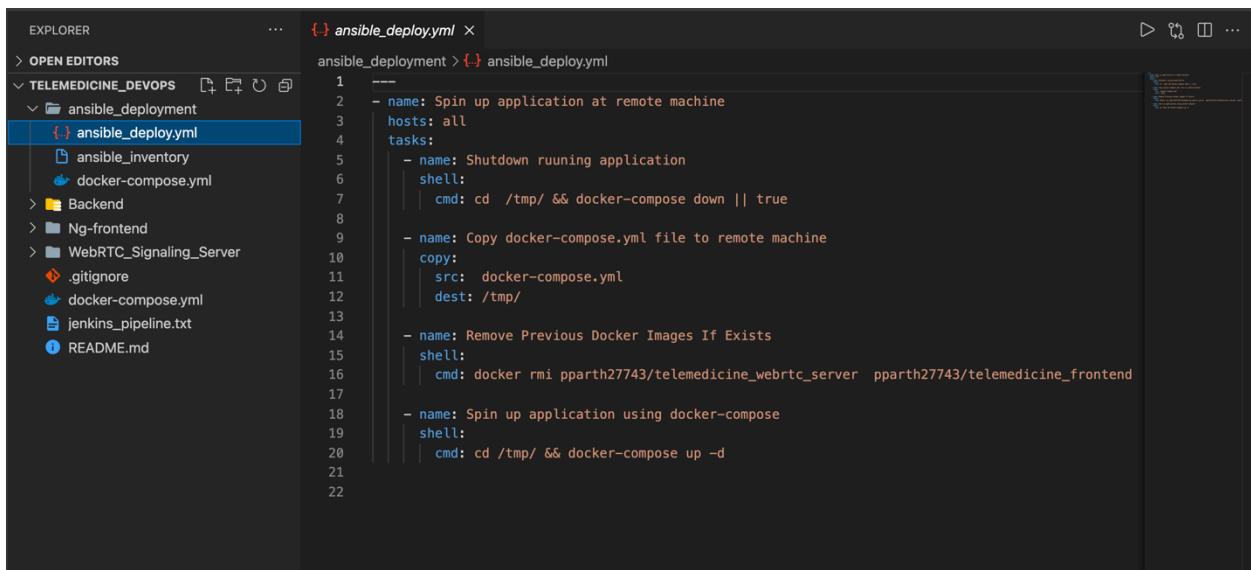
Ansible playbooks are in a very simple language that is YAML.



The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor pane on the right. The Explorer sidebar shows a project structure under 'TELEMEDICINE\_DEVOPS'. In the 'ansible\_deployment' folder, the 'ansible\_inventory' file is selected and highlighted in blue. The Editor pane displays the contents of the 'ansible\_inventory' file:

```
[lenovo]
192.168.0.109 ansible_user=parth ansible_python_interpreter=/usr/bin/python3
```

Figure 21 Ansible Inventory file



The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor pane on the right. The Explorer sidebar shows the same project structure as Figure 21. In the 'ansible\_deployment' folder, the 'ansible\_deploy.yml' file is selected and highlighted in blue. The Editor pane displays the contents of the 'ansible\_deploy.yml' file:

```
---
1 - name: Spin up application at remote machine
2   hosts: all
3   tasks:
4     - name: Shutdown running application
5       shell:
6         cmd: cd /tmp/ && docker-compose down || true
7
8     - name: Copy docker-compose.yml file to remote machine
9       copy:
10        src: docker-compose.yml
11        dest: /tmp/
12
13     - name: Remove Previous Docker Images If Exists
14       shell:
15         cmd: docker rmi pparth27743/telemedicine_webrtc_server pparth27743/telemedicine_frontend
16
17     - name: Spin up application using docker-compose
18       shell:
19         cmd: cd /tmp/ && docker-compose up -d
20
21
22
```

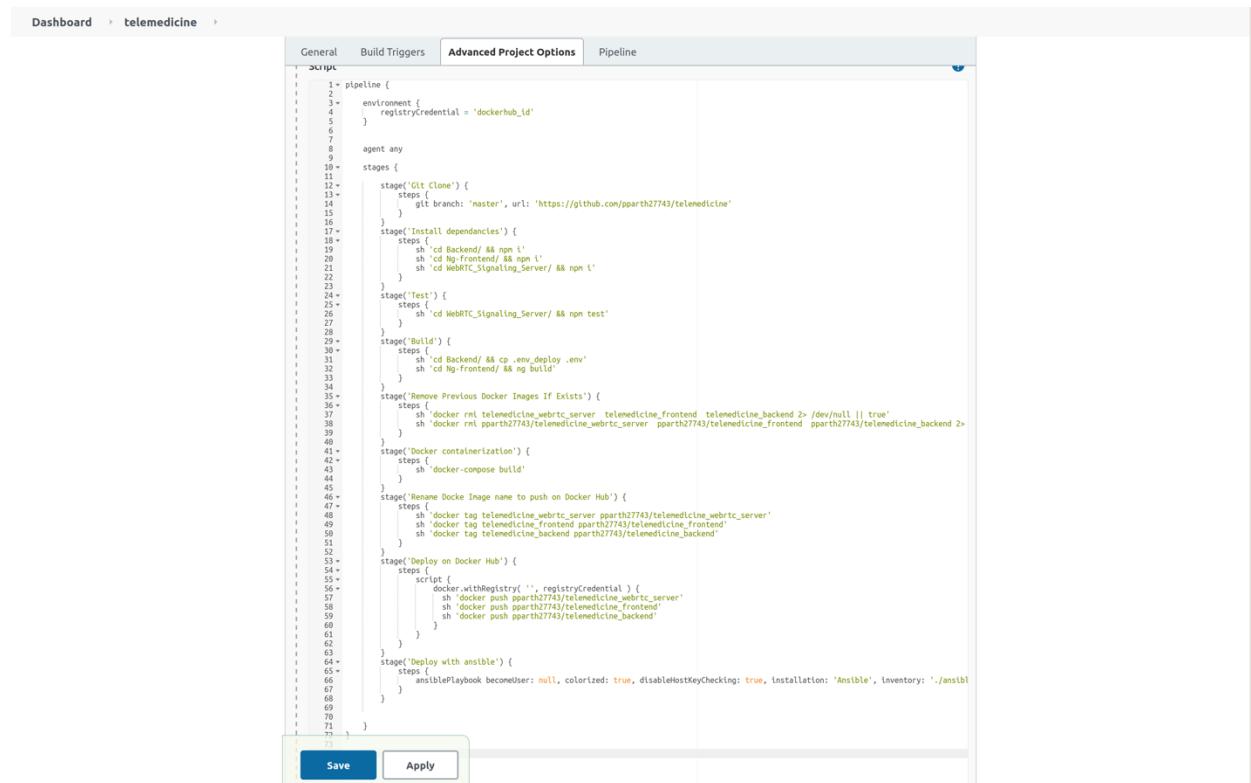
Figure 22 Ansible playbook

Ansible Playbook does series of operation on managed host starting from Shutting down already running application to deployment of updated application using docker compose file.

### 3.8 Jenkins and Pipeline

All the above steps can be done in an automated fashion by using Jenkins. Jenkins is a free and open-source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. Jenkins by default runs on 8080 port.

We need to install the necessary plugins for Git, Ansible and need to store credentials to push docker image on to docker hub.



```

Dashboard > telemedicine >
General Build Triggers Advanced Project Options Pipeline
Script
1+ pipeline {
2+
3+     environment {
4+         registryCredential = 'dockerhub_id'
5+
6+
7+     agent any
8+
9+     stages {
10+         stage('Git Clone') {
11+             steps {
12+                 git branch: 'master', url: 'https://github.com/pparth27743/telemedicine'
13+             }
14+         }
15+         stage('Install dependancies') {
16+             steps {
17+                 sh 'cd Backend/ && npm i'
18+                 sh 'cd Np-frontend/ && npm i'
19+                 sh 'cd WebRTC_Signaling_Server/ && npm i'
20+             }
21+         }
22+         stage('Test') {
23+             steps {
24+                 sh 'cd WebRTC_Signaling_Server/ && npm test'
25+             }
26+         }
27+         stage('Build') {
28+             steps {
29+                 sh 'cd Backend/ && cp .env deploy.env'
30+                 sh 'cd Np-frontend/ && ng build'
31+             }
32+         }
33+         stage('Remove Previous Docker Images If Exists') {
34+             steps {
35+                 sh 'docker rmi telemedicine_webrtc_server telemedicine_frontend telemedicine_backend 2>/dev/null || true'
36+                 sh 'docker rmi pparth27743/telemedicine_webrtc_server pparth27743/telemedicine_frontend pparth27743/telemedicine_backend 2>/dev/null || true'
37+             }
38+         }
39+         stage('Docker containerization') {
40+             steps {
41+                 sh 'docker-compose build'
42+             }
43+         }
44+         stage('Rename Docker Image name to push on Docker Hub') {
45+             steps {
46+                 sh 'docker tag telemedicine_webrtc_server pparth27743/telemedicine_webrtc_server'
47+                 sh 'docker tag telemedicine_frontend pparth27743/telemedicine_frontend'
48+                 sh 'docker tag telemedicine_backend pparth27743/telemedicine_backend'
49+             }
50+         }
51+         stage('Deploy on Docker Hub') {
52+             steps {
53+                 script {
54+                     docker.withRegistry('', registryCredential) {
55+                         sh 'docker push pparth27743/telemedicine_webrtc_server'
56+                         sh 'docker push pparth27743/telemedicine_frontend'
57+                         sh 'docker push pparth27743/telemedicine_backend'
58+                     }
59+                 }
60+             }
61+         }
62+         stage('Deploy with ansible') {
63+             steps {
64+                 ansiblePlaybook becomeUser: null, colorized: true, disableHostKeyChecking: true, installation: 'Ansible', inventory: './ansible'
65+             }
66+         }
67+     }
68+ }
69+
70+
71+ }
72+ }

```

Figure 23 Pipeline Script

Activities Firefox Web Browser ▾ May 19 19:11

telemedicine [Jenkins] X AllMS chief explains ▾ + localhost:8080/job/telemedicine/ ... search ? 1 admin log out

**Jenkins**

Dashboard > telemedicine >

**Pipeline telemedicine**

**Stage View**

Git Clone	Install dependancies	Test	Build	Remove Previous Docker Images if Exists	Docker containerization	Rename Docker Image name to push on Docker Hub	Deploy on Docker Hub	Deploy with ansible
539ms	38s	1s	48s	1s	48s	839ms	1min 24s	1min 36s
May 16 18:20 No Changes	539ms	38s	1s	48s	1s	48s	839ms	1min 24s

Average stage times: (Average full run time: ~5min 21s)

**Recent Changes**

**Build History**

#	Date	Time
#51	May 16, 2021	6:20 PM
#50	May 16, 2021	5:51 PM
#49	May 16, 2021	4:57 PM
#48	May 16, 2021	4:27 PM
#47	May 16, 2021	4:23 PM
#46	May 16, 2021	4:13 PM
#45	May 16, 2021	4:08 PM
#44	May 16, 2021	4:03 PM
#43	May 16, 2021	3:58 PM
#42	May 16, 2021	3:49 PM
#41	May 16, 2021	3:46 PM

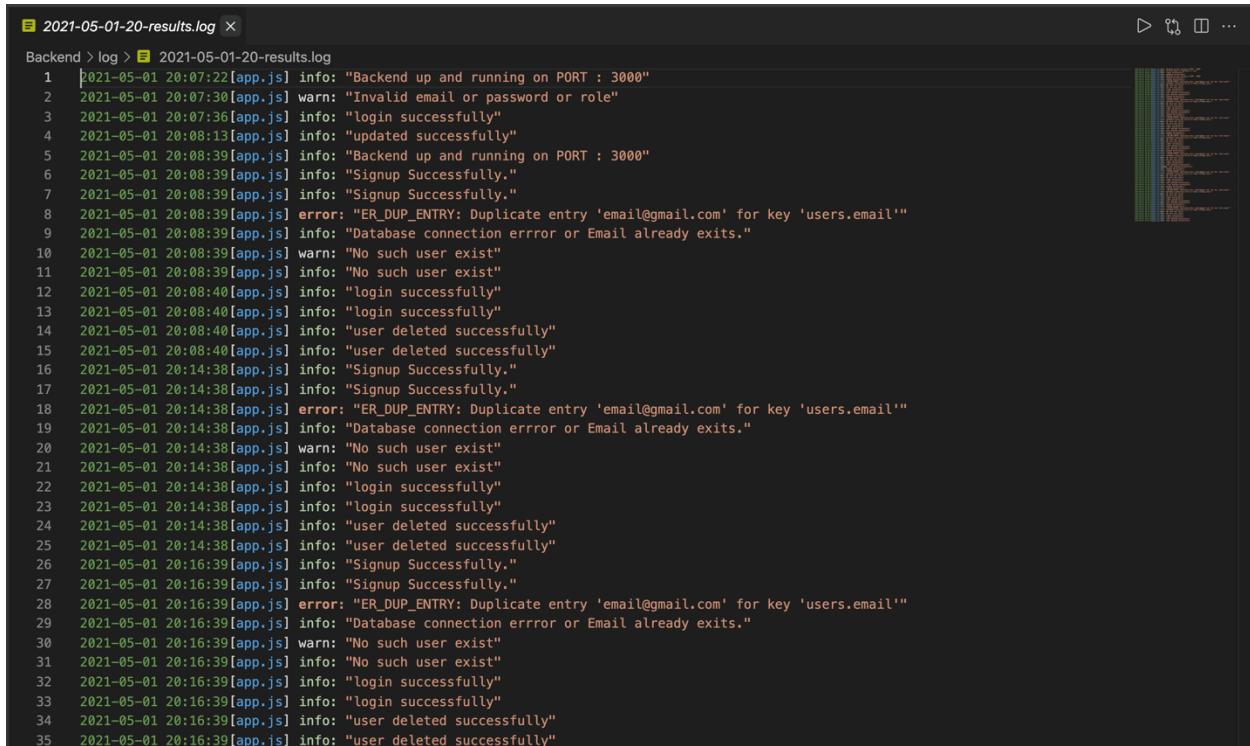
**Permalinks**

- Last build (#51), 3 days 0 hr ago
- Last stable build (#51), 3 days 0 hr ago
- Last successful build (#51), 3 days 0 hr ago
- Last failed build (#48), 3 days 2 hr ago
- Last unsuccessful build (#48), 3 days 2 hr ago
- Last completed build (#51), 3 days 0 hr ago

Figure 24 Build of Pipeline

### 3.9 Monitoring using ELK:

"ELK" is the acronym for three open source projects: Elasticsearch, Logstash, and Kibana. Elasticsearch is a search and analytics engine. Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch. Kibana lets users visualize data with charts and graphs in Elasticsearch. The Elastic Stack is the next evolution of the ELK Stack.



```
Backend > log > 2021-05-01-20-results.log
1  2021-05-01 20:07:22[app.js] info: "Backend up and running on PORT : 3000"
2  2021-05-01 20:07:30[app.js] warn: "Invalid email or password or role"
3  2021-05-01 20:07:36[app.js] info: "login successfully"
4  2021-05-01 20:08:13[app.js] info: "updated successfully"
5  2021-05-01 20:08:39[app.js] info: "Backend up and running on PORT : 3000"
6  2021-05-01 20:08:39[app.js] info: "Signup Successfully."
7  2021-05-01 20:08:39[app.js] info: "Signup Successfully."
8  2021-05-01 20:08:39[app.js] error: "ER_DUP_ENTRY: Duplicate entry 'email@gmail.com' for key 'users.email'"
9  2021-05-01 20:08:39[app.js] info: "Database connection error or Email already exists."
10 2021-05-01 20:08:39[app.js] warn: "No such user exist"
11 2021-05-01 20:08:39[app.js] info: "No such user exist"
12 2021-05-01 20:08:40[app.js] info: "login successfully"
13 2021-05-01 20:08:40[app.js] info: "login successfully"
14 2021-05-01 20:08:40[app.js] info: "user deleted successfully"
15 2021-05-01 20:08:40[app.js] info: "user deleted successfully"
16 2021-05-01 20:14:38[app.js] info: "Signup Successfully."
17 2021-05-01 20:14:38[app.js] info: "Signup Successfully."
18 2021-05-01 20:14:38[app.js] error: "ER_DUP_ENTRY: Duplicate entry 'email@gmail.com' for key 'users.email'"
19 2021-05-01 20:14:38[app.js] info: "Database connection error or Email already exists."
20 2021-05-01 20:14:38[app.js] warn: "No such user exist"
21 2021-05-01 20:14:38[app.js] info: "No such user exist"
22 2021-05-01 20:14:38[app.js] info: "login successfully"
23 2021-05-01 20:14:38[app.js] info: "login successfully"
24 2021-05-01 20:14:38[app.js] info: "user deleted successfully"
25 2021-05-01 20:14:38[app.js] info: "user deleted successfully"
26 2021-05-01 20:16:39[app.js] info: "Signup Successfully."
27 2021-05-01 20:16:39[app.js] info: "Signup Successfully."
28 2021-05-01 20:16:39[app.js] error: "ER_DUP_ENTRY: Duplicate entry 'email@gmail.com' for key 'users.email'"
29 2021-05-01 20:16:39[app.js] info: "Database connection error or Email already exists."
30 2021-05-01 20:16:39[app.js] warn: "No such user exist"
31 2021-05-01 20:16:39[app.js] info: "No such user exist"
32 2021-05-01 20:16:39[app.js] info: "login successfully"
33 2021-05-01 20:16:39[app.js] info: "login successfully"
34 2021-05-01 20:16:39[app.js] info: "user deleted successfully"
35 2021-05-01 20:16:39[app.js] info: "user deleted successfully"
```

Figure 25 Log file

Now, this log file needs to be uploaded to <https://cloud.elastic.co>. It is an online service that provides ELK.

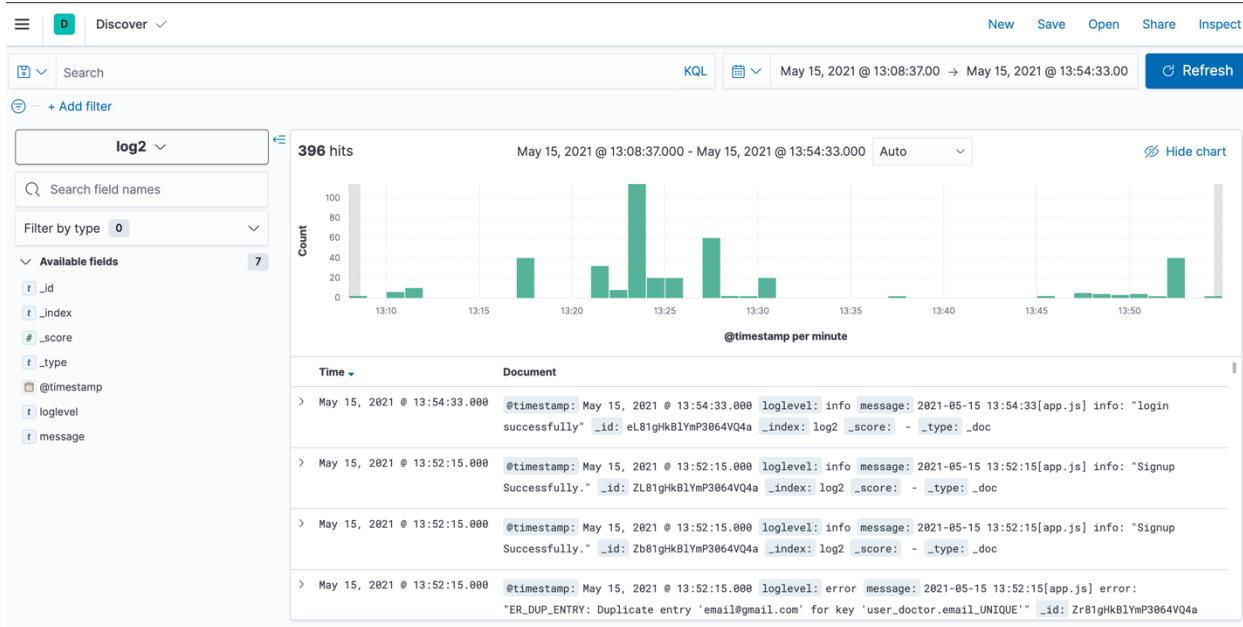


Figure 26 ELK logs

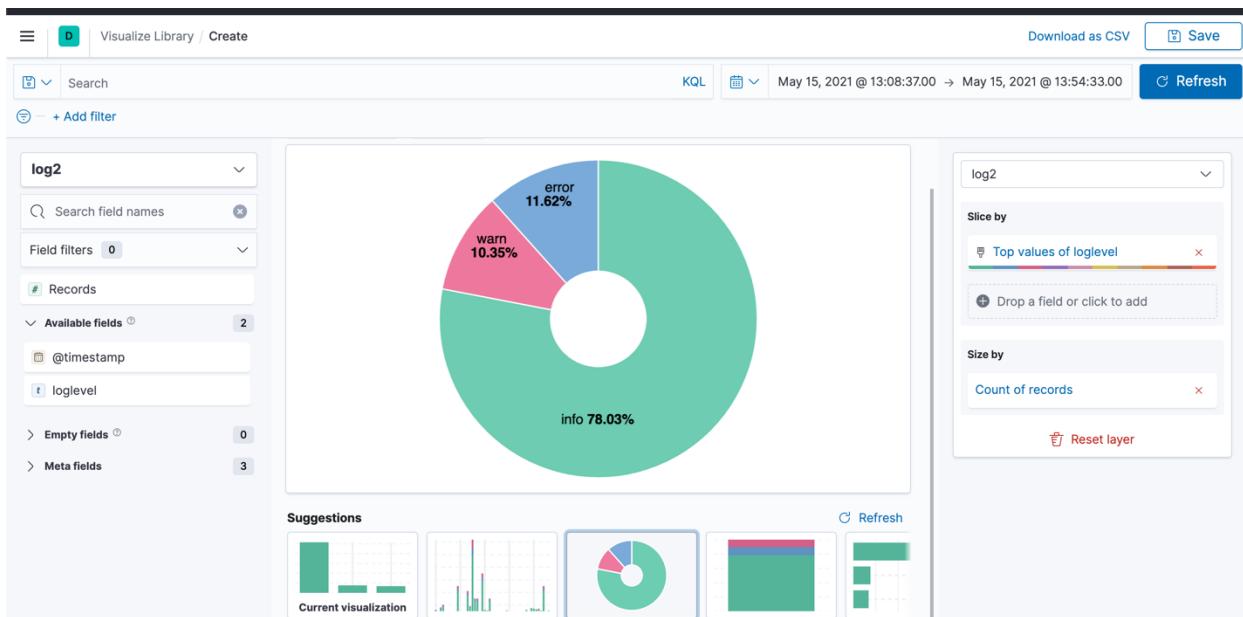


Figure 27 Graph Based on Log Level

## 4. Results and Discussion

### 4.1 Login Page

The screenshot shows a login form titled "Login Page". It includes fields for "Email" and "Password", a dropdown menu for "Role", and a "Login" button. Below the form is a link for users who don't have an account.

**Login Page**

Email

Password

Role ▾

Login

Din't Have Account? [Signup](#)

Figure 28 Login Page

## 4.2 Signup Page

### Signup Page

firstname

lastname

Email

Password

Role

[Go back to Login](#)

Figure 29 Signup Page

## 4.3 Doctor's Home Page

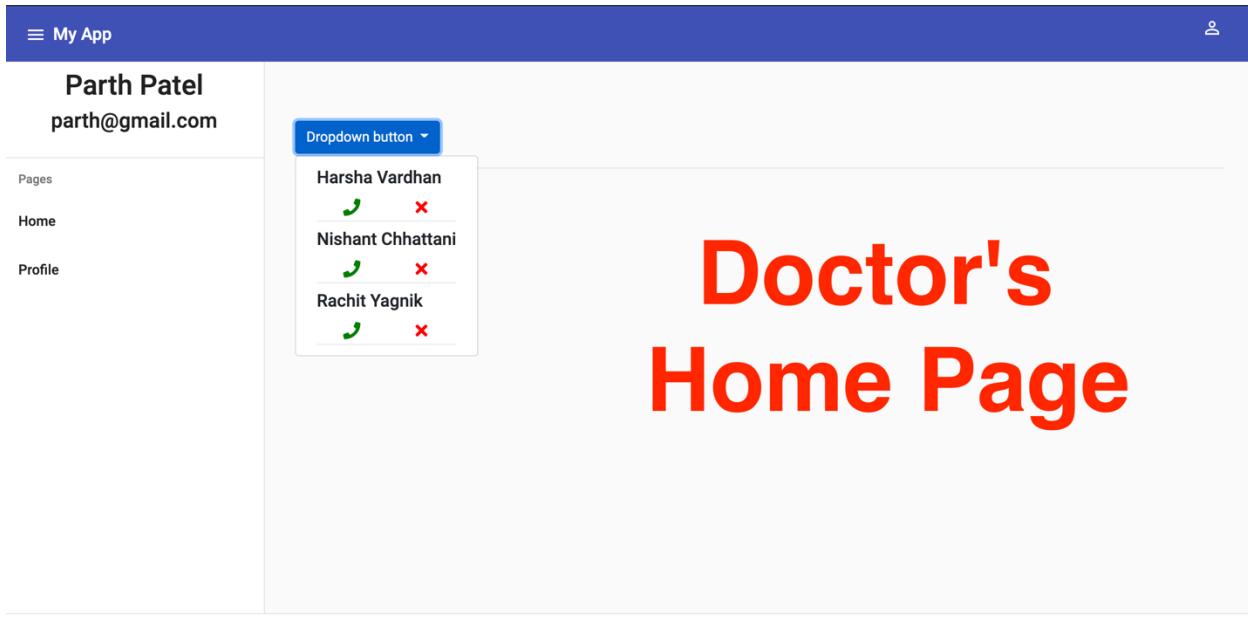


Figure 30 Doctor's Home Page

## 4.4 Patient's Home Page

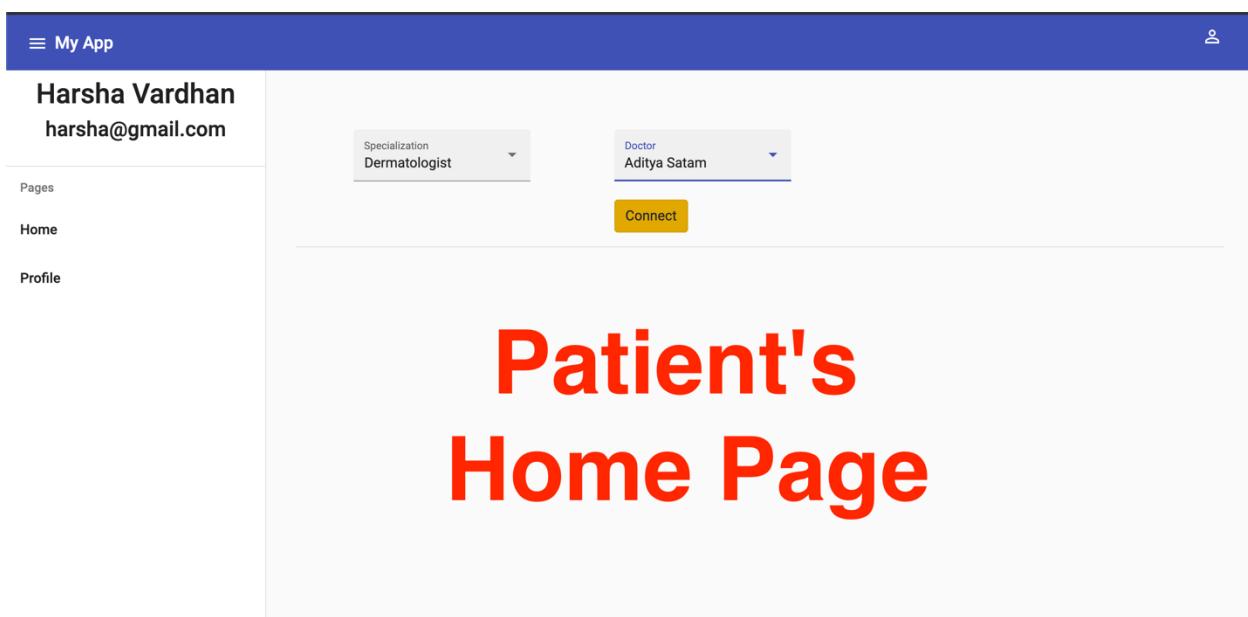


Figure 31 Patient's Home Page

## 4.5 Profile Update Page

The screenshot shows a mobile application interface for updating a profile. At the top, there is a blue header bar with the text "My App" and a user icon. Below the header, the user's name "Parth Patel" and email "parth@gmail.com" are displayed. A sidebar on the left lists "Pages", "Home", and "Profile", with "Profile" being the active tab. The main content area has a large red title "Update Profile". Below it is a "User Details" form. The form fields include:

- firstname: Parth
- lastname: Patel
- Email: parth@gmail.com
- Specialization: Neurologist (dropdown menu)

There is also a checkbox labeled "Want to change Password ?" and a blue "Update Details" button at the bottom of the form.

© All rights reserved May 19, 2021

Figure 32 Profile Update Page

## 4.6 Doctor Consulting Patient

Here You can see Patient has added multiple steam in the call. Those stream are of Audio and Video through which Patient can send real-time heart rate or stethoscope audio so doctor can treat him/her precisely.

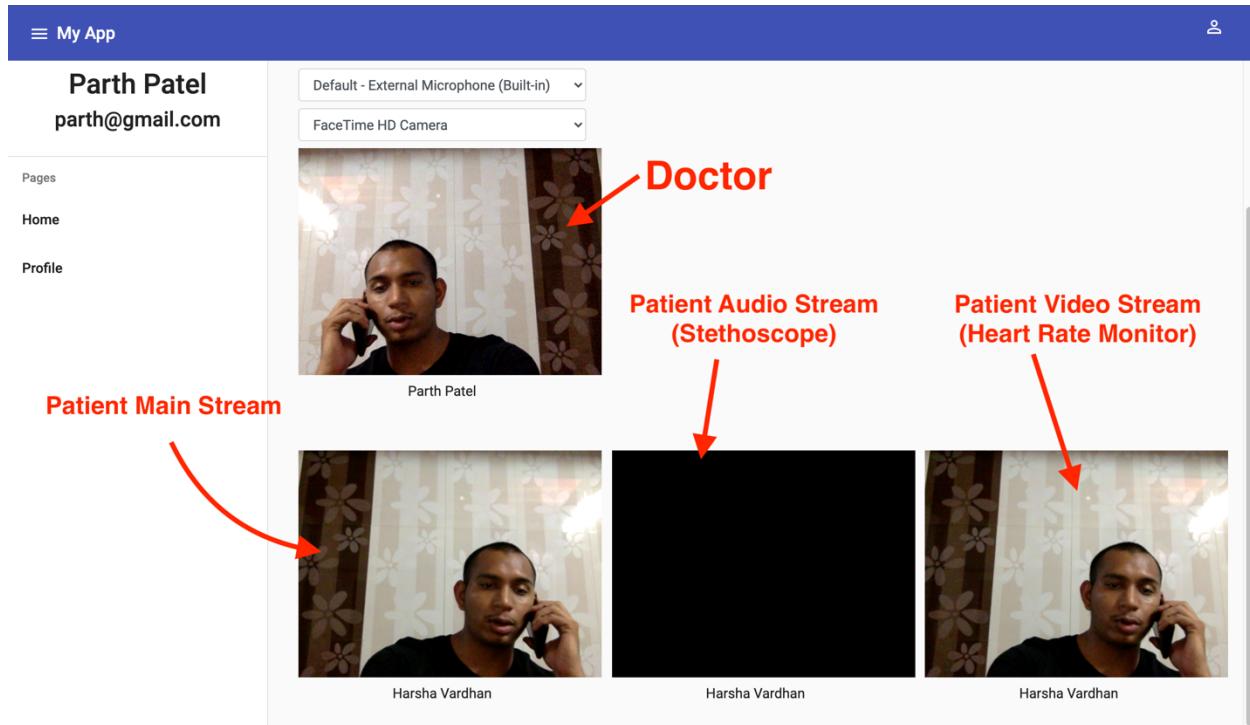


Figure 33 Doctor and Patient Having Call

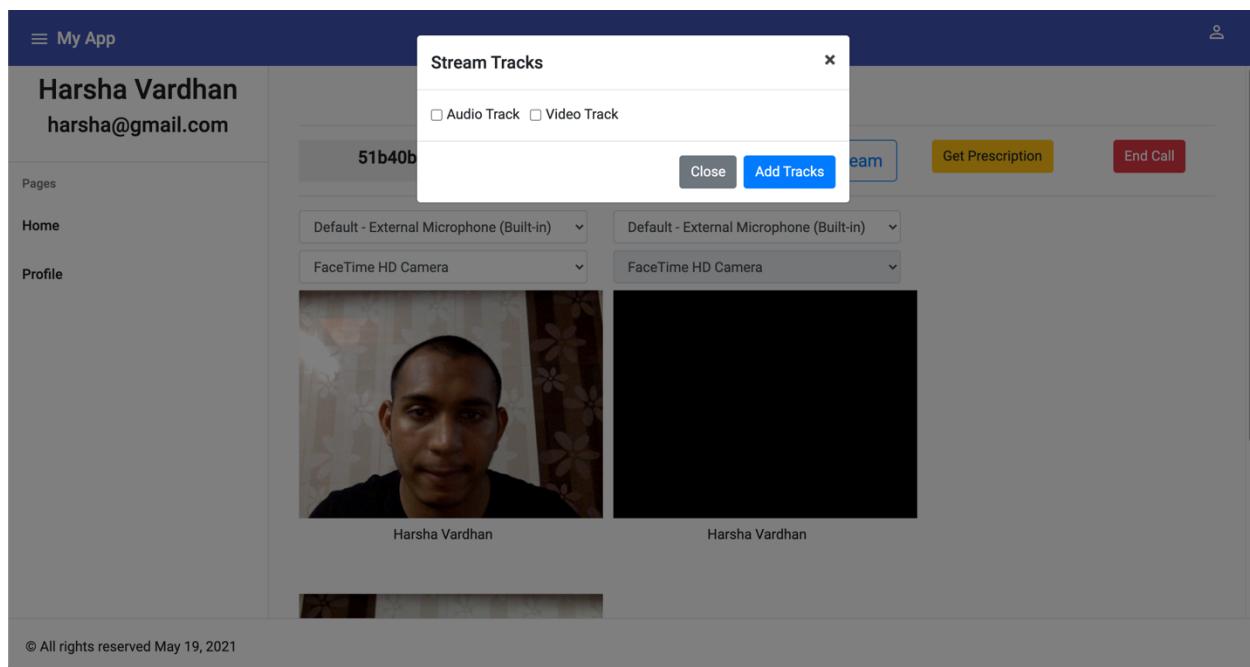
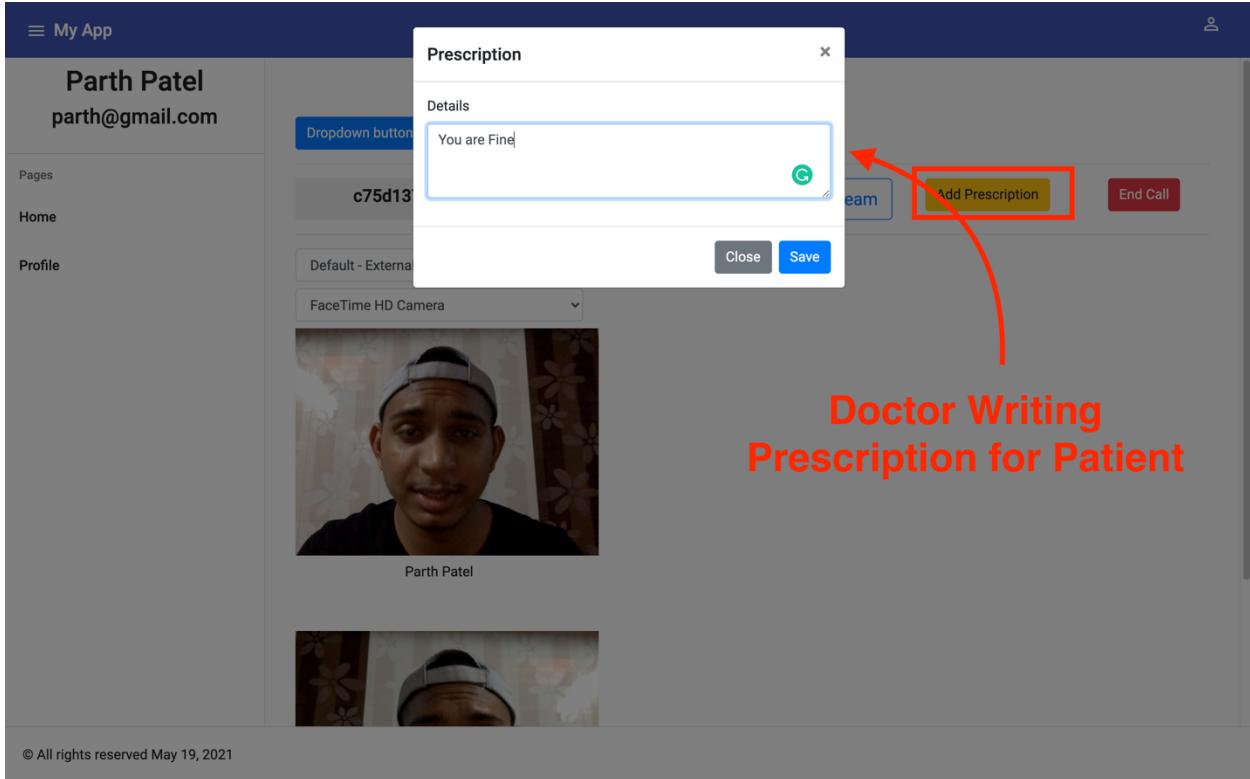


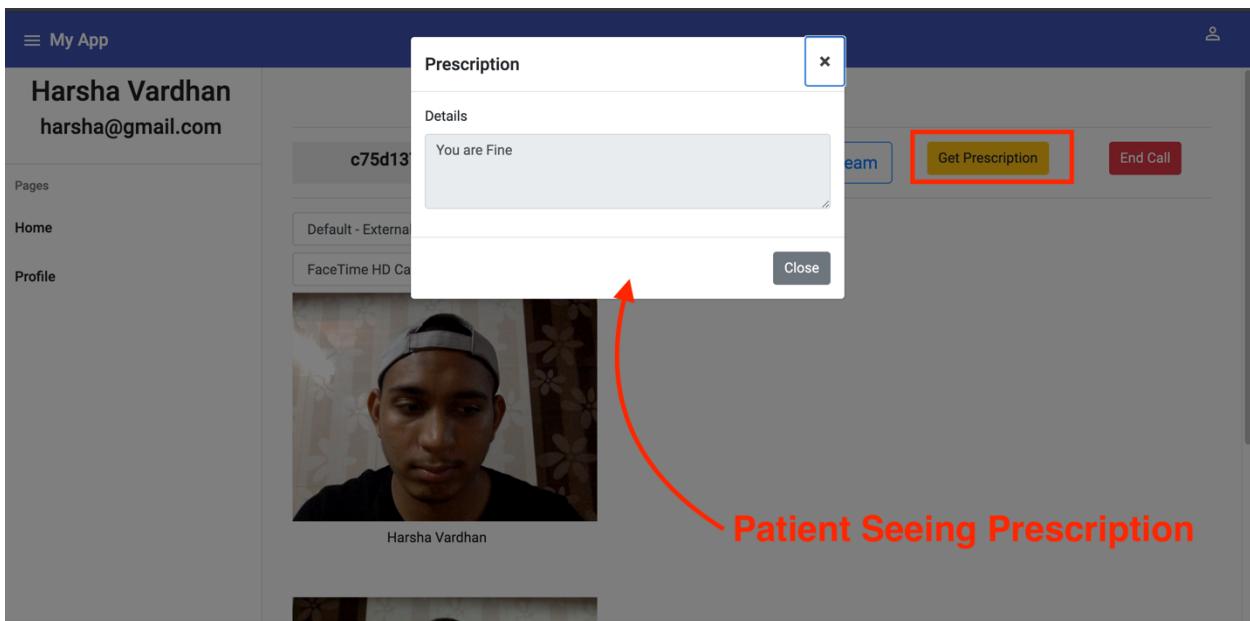
Figure 34 Add Stream

## 4.7 Doctor Writing Prescription for Patient



Doctor Writing  
Prescription for Patient

Figure 35 Doctor Writes Prescription



Patient Seeing Prescription

Figure 36 Patient Reads Prescription

## **5. Scope for future work**

As This project is not complete in itself. There are many functionality can be added to this to make a production ready application. We just implemented Video Conferencing and Doctor writing prescription for the patient. But There are many such as text chat, file sharing, screen sharing, reminders etc. things can be added in this application. This was about functional features. But non functional things can be security enhancement, scalability, availability, efficiency can be added to the project as well.

## **6. Conclusion**

We have successfully build a Telemedicine application through which Patient can take appointment online and consult Doctor remotely. In doing so, we have various DevOps tools like used GitHub, Jenkins, Docker, Docker Compose, Ansible. These tools are integrated using Jenkins to make entire SDLC cycle automated.

DevOps methodology and tools prove that it is better than Agile methodology for an application which is rich in features and required features to be deployed as soon as they are developed. And that's what modern enterprise required to stay in competition.

## 7. References:

- [1] <https://www.jenkins.io/doc/>
- [2] [https://docs.ansible.com/ansible/latest/scenario\\_guides/guide\\_docker.html](https://docs.ansible.com/ansible/latest/scenario_guides/guide_docker.html)
- [3] <https://docs.github.com/en>
- [4] <https://www.google.co.in/>
- [5] <https://stackoverflow.com/>
- [6] <https://docs.docker.com/compose/>