SIES (Nerul) College of Arts, Science and Commerce
NAAC Re-Accredited 'A' Grade

Sri Chandrasekarendra Saraswathy Vidyapuram,
Plot 1-C, Sector V, Nerul, Navi Mumbai-400 706.

PROJECT REPORT ON

Live Voice Rooms

SUBMITTED TO

MUMBAI UNIVERSITY

BY

Neha Milind Gholap
(T.Y. B Sc. Computer Science)

(2021-2022)

SIES (Nerul) College of Arts, Science and Commerce
NAAC Re-Accredited 'A' Grade

Sri Chandrasekarendra Saraswathy Vidyapuram,
Plot 1-C, Sector V, Nerul, Navi Mumbai-400 706.

Certificate

This is to certify that the project entitled "**LIVE VOICE ROOMS**" developed using HTML, CSS, NodeJS, Express and MongoDB is successfully completed by **Ms. Neha Gholap** of Third Year Bachelor of Science (Computer Science) as per the requirement of University of Mumbai in part fulfillment for the completion of Degree of Bachelor of Science (Computer Science). It is also to certify that this is the original work of the candidate done during the academic year 2021-2022.

Seat No.: S.19.29 Date of Submission: __/__/21

Asst. Prof. Manasvi Sharma
(Project Guide)
Date: _____

Dr. Sheeja Ravi
(HOD)
Date: _____

External Examiner:
Date: _____
(College Seal)

INDEX

CHAPTER 1: INTRODUCTION

1.1 WHAT IS LIVE-VOICE-ROOMS?

Live-Voice-Rooms is an interactive, voice-based platform where users can join real-time discussions in designated rooms. You can think of it as tuning into a TED talk or podcast — but with the option to jump in and chat.

Since it's voice-only app and doesn't use your camera, this Web-App hopes that you won't worry about "eye-contact, what you're wearing or where you are".

1.2 HOW DOES LIVE-VOICE-ROOMS WORK?

This app is meant to be a place to meet, talk, and share ideas, the app is pretty basic. It lets you create and join "rooms" - where you can chat with others in a conference call. You can't share pictures, videos, or even text. All you can do is talk. Users and join and leave the call at any time, too.

When you open the app, you will see a list of "rooms" also number of people in each room.

You can join the room by tapping on it- they're all open for you to hop in or out. If you want to talk you can raise your hand and unmute your mic to speak.

1.3 ORGANISATION OF THE REPORT

1.4.1 INTRODUCTION

This section includes the overall view of the project i.e., the basic problem definition and the general overview of the problem which describes the problem in layman terms.

1.4.2 SOFTWARE REQUIREMENTS SPECIFICATION

This section includes the Software and hardware requirements for the smooth running of the application.

1.4.3 DESIGN AND PLANNING

It also contains technical diagrams like the Entity Relationship diagram, Use Case diagram and Activity diagram.

1.4.4 SYSTEM IMPLEMENTATION

This section consists of pieces of code-logic and their explanations

1.4.5 RESULTS AND DISCUSSION

This section has screenshots of all the implementation i.e., user interface and their description.

1.4.6 SUMMARY AND CONCLUSION

This section has future scope of the project, conclusion and references.

CHAPTER 2: REQUIREMENTS AND SPECIFICATIONS

The system has been designed keeping basic system requirements into consideration related to hardware and software.

SOFTWARE AND HARDWARE REQUIREMENTS:

Software Requirements:

Frontend: ReactJS, Redux for State Management, CSS

Backend: NodeJS, Express

Database: MongoDB

Real-Time Communication: WebRTC, Socket.io library

Tools: Visual Studio Code, Hyper terminal, MongoDB Compass

Platform: Windows 7 & above

Browser Used: Any of Mozilla, Opera, Chrome, etc

Hardware Requirements:

Processor: Dual core or higher

Speed: 2.0GHz

Primary Memory: 512 MB RAM

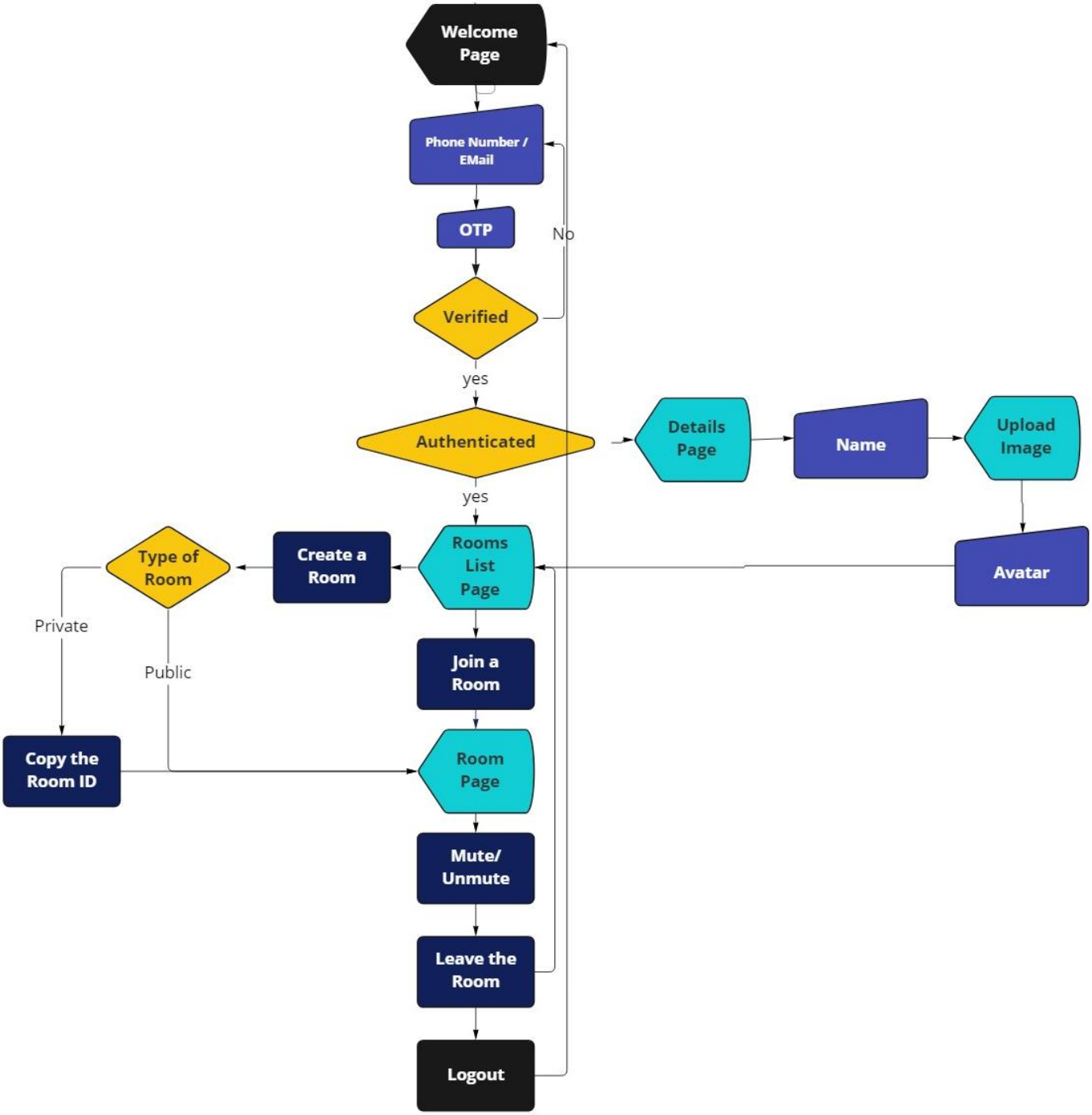
Hard Disk: 2GB

From Users View Point:

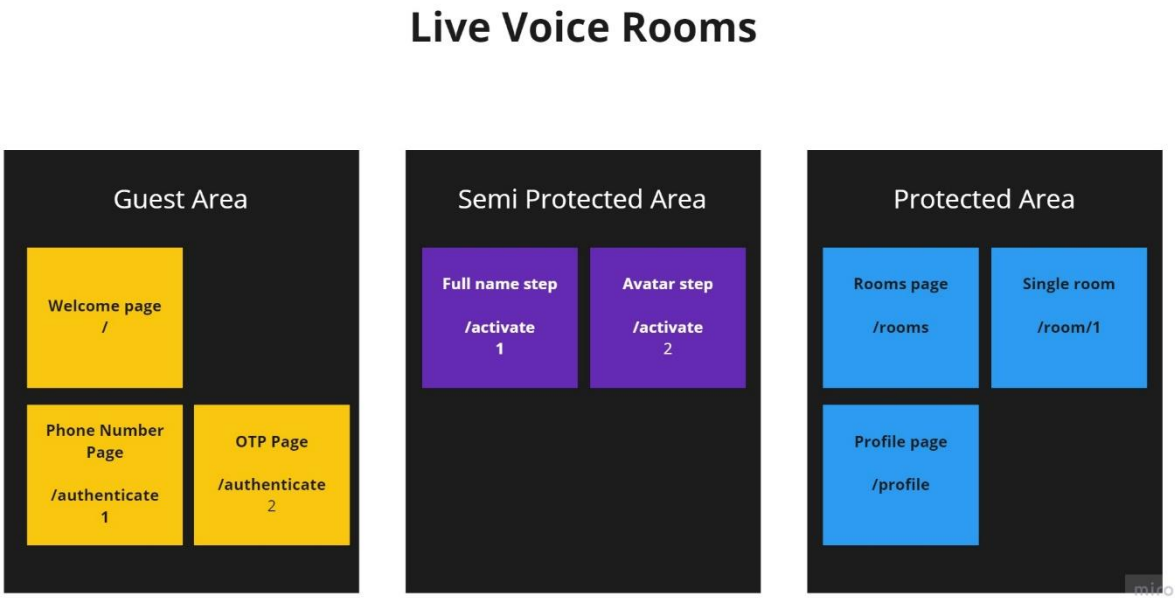
For viewing the web-application or using it, a user will need a desktop PC or a smart phone with an *active* internet connection and a browser (Google Chrome)

CHAPTER 3: DESIGNING AND PLANNING

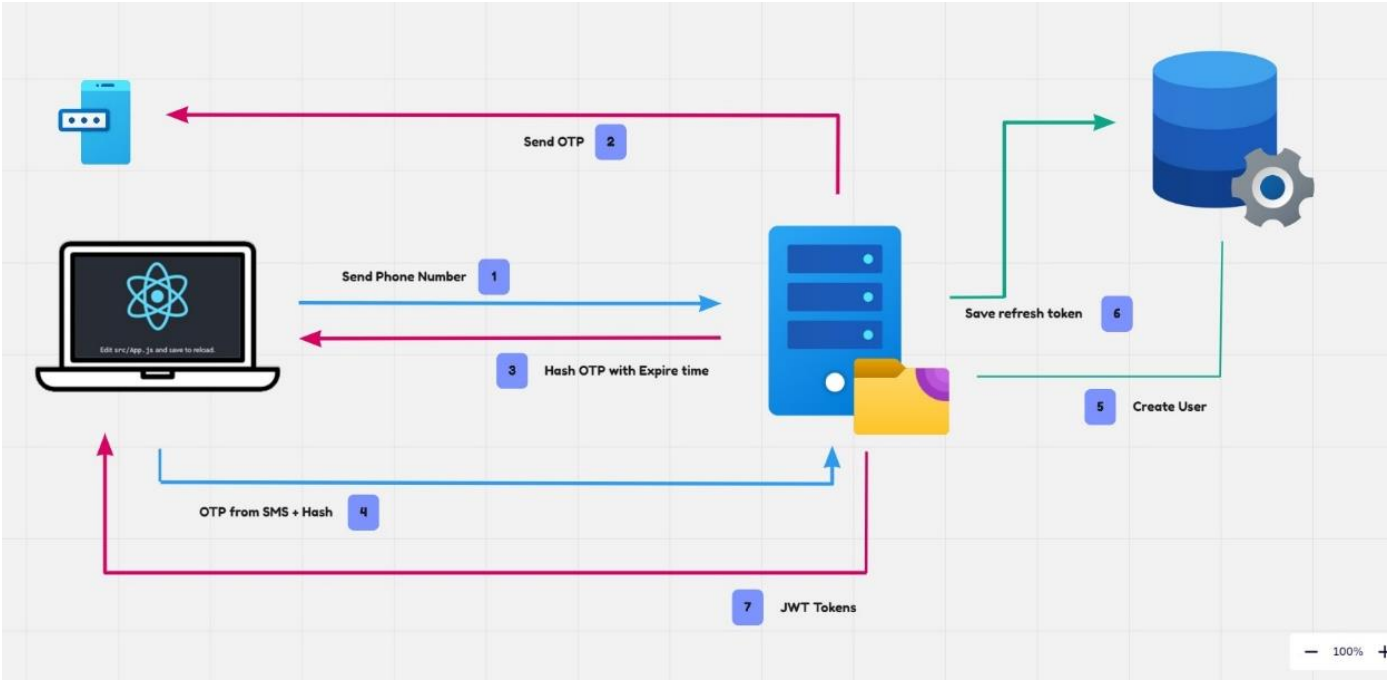
3.1 USER JOURNEY DIAGRAM



3.2 AREAS AND THEIR ROUTES



3.3 AUTHENTICATION AND OTP PROCESS FLOW



CHAPTER 4: IMPLEMENTATION OF MODULES

4.1 How did I implement the OTP Service?

I created routes in routes.js file which are /api/send-otp and /api/verify-otp

backend > routes.js

```
const router = require('express').Router();
const authController = require('./controllers/auth-controller');
const activateController = require('./controllers/activate-controller');
const authMiddleware = require('./middlewares/auth-middleware');
const roomsController = require('./controllers/rooms-controller');

router.post('/api/send-otp', authController.sendOtp);
router.post('/api/verify-otp', authController.verifyOtp);
```

The logic can be written inside the callback function of /api/send-otp but to make the code more organized I created another file named auth-controller.js. and reference this function over there.

Inside auth-controller.js I created a class named AuthController and a method inside it sendOtp()

backend > controllers > auth-controller.js

```
const otpService = require('../services/otp-service');
const hashService = require('../services/hash-service');
const userService = require('../services/user-service');
const tokenService = require('../services/token-service');
const UserDto = require('../dtos/user-dto');

class AuthController {
    async sendOtp(req, res) {
        const { phone } = req.body;
        if (!phone) {
            res.status(400).json({ message: 'Phone field is required!' });
        }

        const otp = await otpService.generateOtp();
        // const otp = 7777;

        const ttl = 1000 * 60 * 2; // 2 min
        const expires = Date.now() + ttl;
        const data = `${phone}.${otp}.${expires}`;
        const hash = hashService.hashOtp(data);

```

In otp-service.js I have created OtpService() class in which there are three methods generateOtp(), sendBySms(), verifyOtp().
A random integer between 1000-9999 is created for otp generation. I have also imported twilio library for sending sms on phone.

backend > services > JS otp-service.js

```js
const crypto = require('crypto');
const hashService = require('./hash-service');

const smsSid = process.env.SMS_SID;
const smsAuthToken = process.env.SMS_AUTH_TOKEN;
const twilio = require('twilio')(smsSid, smsAuthToken, {
    lazyLoading: true,
});

class OtpService {
    async generateOtp() {
        const otp = crypto.randomInt(1000, 9999);
        return otp;
    }

    async sendBySms(phone, otp) {
        return await twilio.messages.create({
            to: phone,
            from: process.env.SMS_FROM_NUMBER,
            body: `Your codershouse OTP is ${otp}`,
        });
    }

    verifyOtp(hashedOtp, data) {
        let computedHash = hashService.hashOtp(data);
        return computedHash === hashedOtp;
    }
}

module.exports = new OtpService();
```

The server will also save a copy of hash otp .
When the user sends the otp it also returns a hash.
The server matches the copy of hash it already had with the hash received by the user.
I generated hash using this function

backend > services > JS hash-service.js

```js
const crypto = require('crypto');

class HashService {
    hashOtp(data) {
        return crypto
            .createHmac('sha256', process.env.HASH_SECRET)
            .update(data)
            .digest('hex');
    }
}

module.exports = new HashService();
```

```
    verifyOtp(hashedOtp, data) {
        let computedHash = hashService.hashOtp(data);
        return computedHash === hashedOtp;
    }
}

module.exports = new OtpService();
```

```
    // send OTP
    try {
        // await otpService.sendBySms(phone, otp);
        res.json({
            hash: `${hash}.${expires}`,
            phone,
            otp
        });
    } catch (err) {
        console.log(err);
        res.status(500).json({ message: 'message sending failed' });
    }
}
```

To match if the hash matches, we use if-else loop and create a user in database.

```
async verifyOtp(req, res) {
    const { otp, hash, phone } = req.body;
    if (!otp || !hash || !phone) {
        res.status(400).json({ message: 'All fields are required!' });
    }

    const [hashedOtp, expires] = hash.split('.');
    if (Date.now() > +expires) {
        res.status(400).json({ message: 'OTP expired!' });
    }

    const data = `${phone}.${otp}.${expires}`;
    const isValid = otpService.verifyOtp(hashedOtp, data);
    if (!isValid) {
        res.status(400).json({ message: 'Invalid OTP' });
    }

    let user;
    try {
        user = await userService.findUser({ phone });
        if (!user) {
            user = await userService.createUser({ phone });
        }
    } catch (err) {
        console.log(err);
        res.status(500).json({ message: 'Db error' });
    }

    const { accessToken, refreshToken } = tokenService.generateTokens({
        _id: user._id,
        activated: false,
    });
```

Web-RTC actions

backend > JS actions.js

```
const ACTIONS = {
    JOIN: 'join',
    LEAVE: 'leave',
    ADD_PEER: 'add-peer',
    REMOVE_PEER: 'remove-peer',
    RELAY_ICE: 'relay-ice',
    RELAY_SDP: 'relay-sdp',
    SESSION_DESCRIPTION: 'session-description',
    ICE_CANDIDATE: 'ice-candidate',
    MUTE: 'mute',
    UNMUTE: 'unmute',
};

module.exports = ACTIONS;
```

All the routes

backend > JS routes.js

```
const router = require('express').Router();
const authController = require('./controllers/auth-controller');
const activateController = require('./controllers/activate-controller');
const authMiddleware = require('./middlewares/auth-middleware');
const roomsController = require('./controllers/rooms-controller');

router.post('/api/send-otp', authController.sendOtp);
router.post('/api/verify-otp', authController.verifyOtp);
router.post('/api/activate', authMiddleware, activateController.activate);
router.get('/api/refresh', authController.refresh);
router.post('/api/logout', authMiddleware, authController.logout);
router.post('/api/rooms', authMiddleware, roomsController.create);
router.get('/api/rooms', authMiddleware, roomsController.index);
router.get('/api/rooms/:roomId', authMiddleware, roomsController.show);

module.exports = router;
```

The user and room dtos

backend > dtos > JS user-dto.js

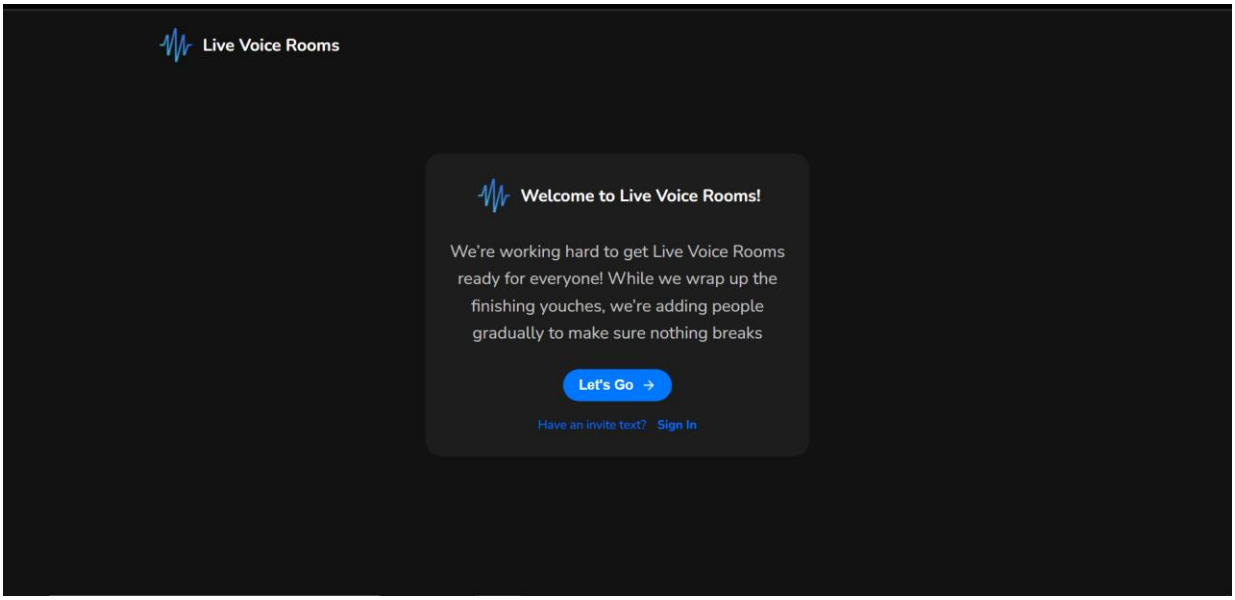
```
class UserDto {
    id;
    phone;
    name;
    avatar;
    activated;
    createdAt;

    constructor(user) {
        this.id = user._id;
        this.phone = user.phone;
        this.name = user.name;
        this.avatar = user.avatar;
        this.activated = user.activated;
        this.createdAt = user.createdAt;
    }
}

module.exports = UserDto;

```

backend > dtos > JS room.dto.js

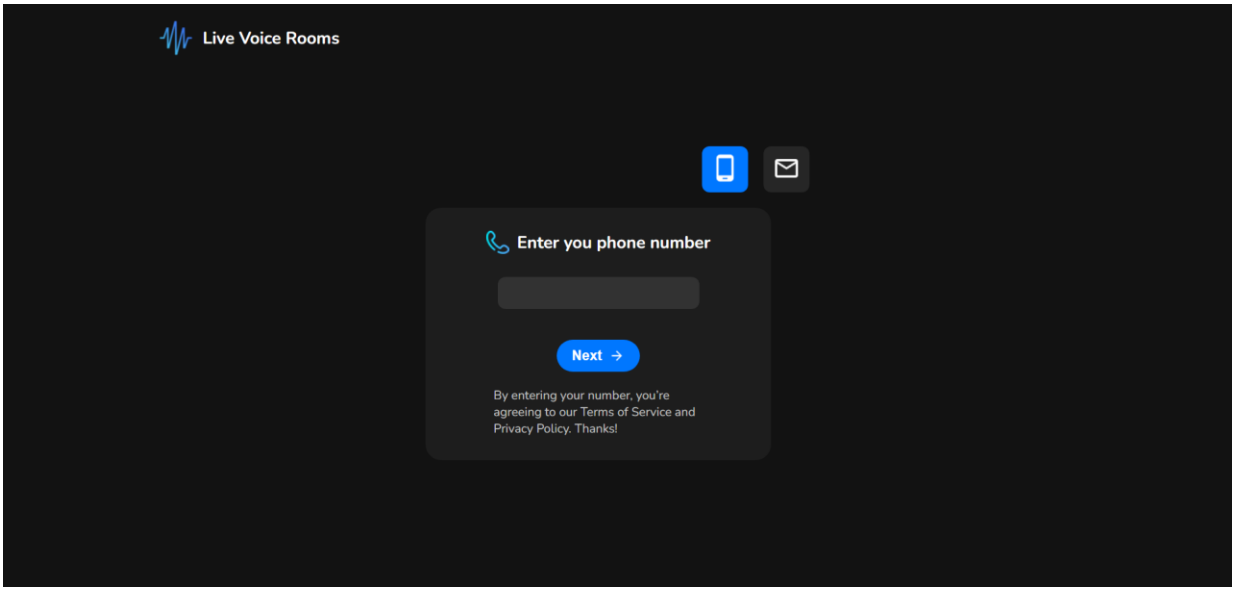
```
class RoomDto {
    id;
    topic;
    roomType;
    speakers;
    ownerId;
    createdAt;

    constructor(room) {
        this.id = room._id;
        this.topic = room.topic;
        this.roomType = room.roomType;
        this.ownerId = room.ownerId;
        this.speakers = room.speakers;
        this.createdAt = room.createdAt;
    }
}
module.exports = RoomDto;

```

CHAPTER 5: RESULTS

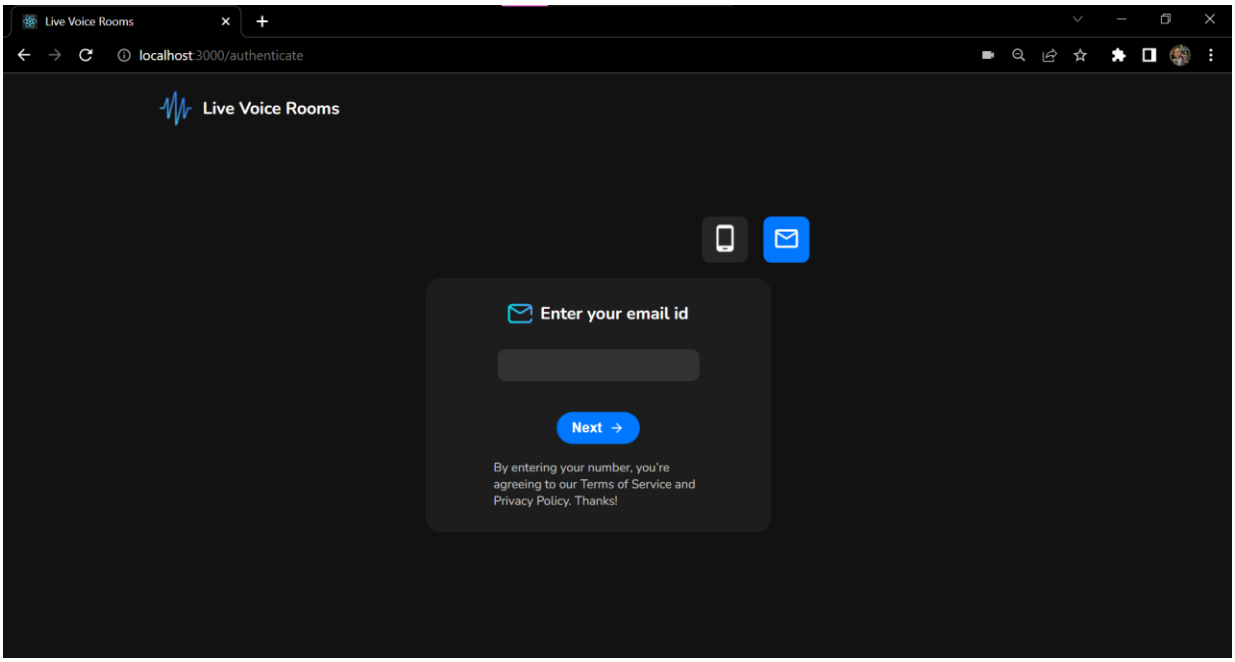
5.1 Welcome Screen



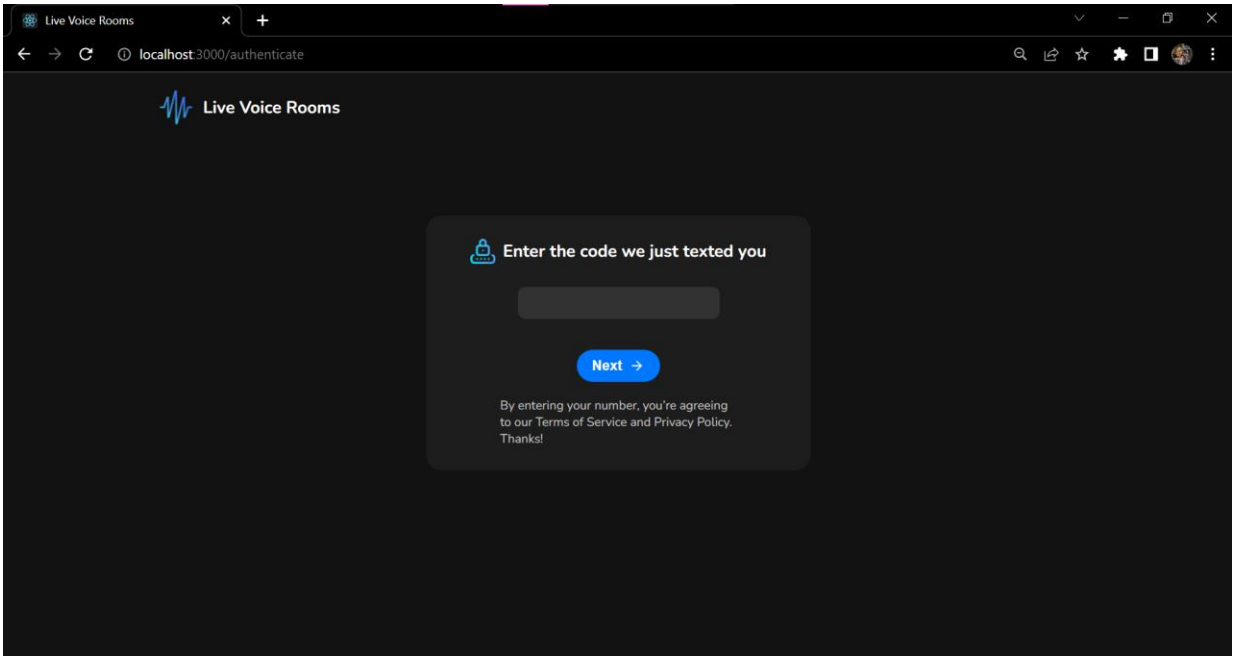
5.2 Phone Number Screen



5.3 Email Screen

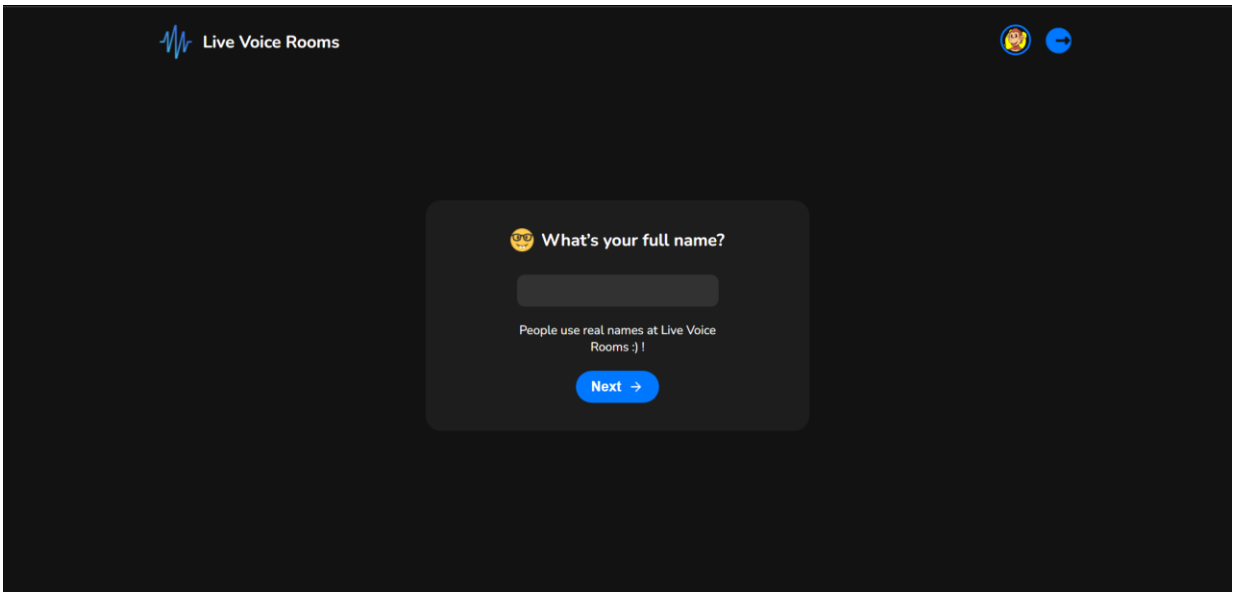


5.4 OTP Screen

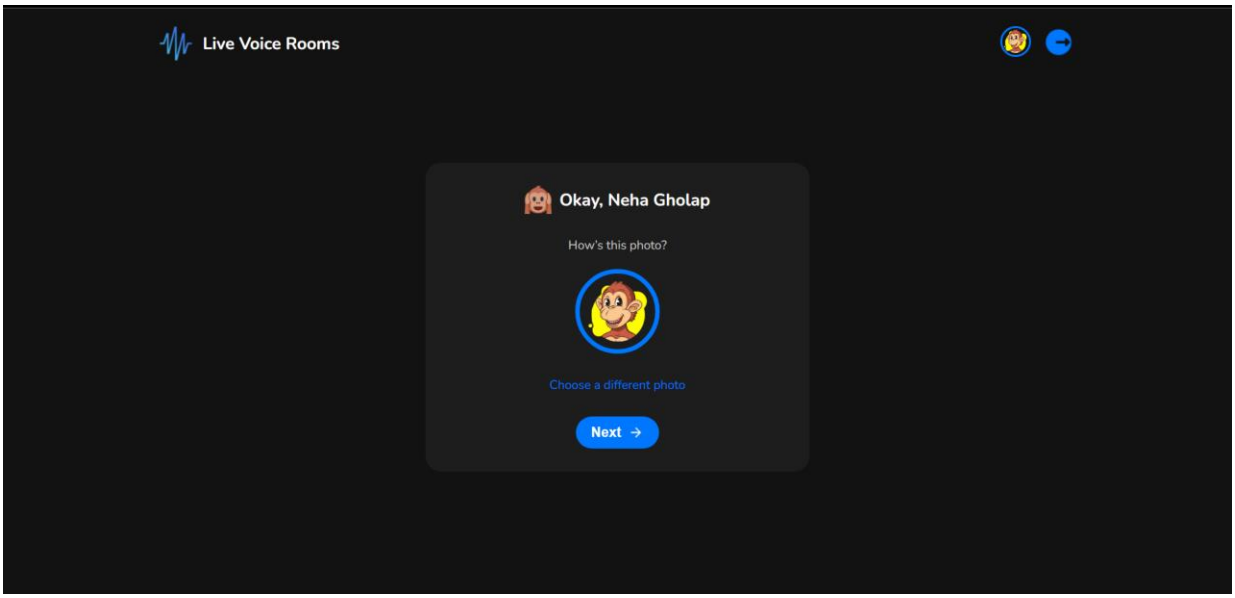


\

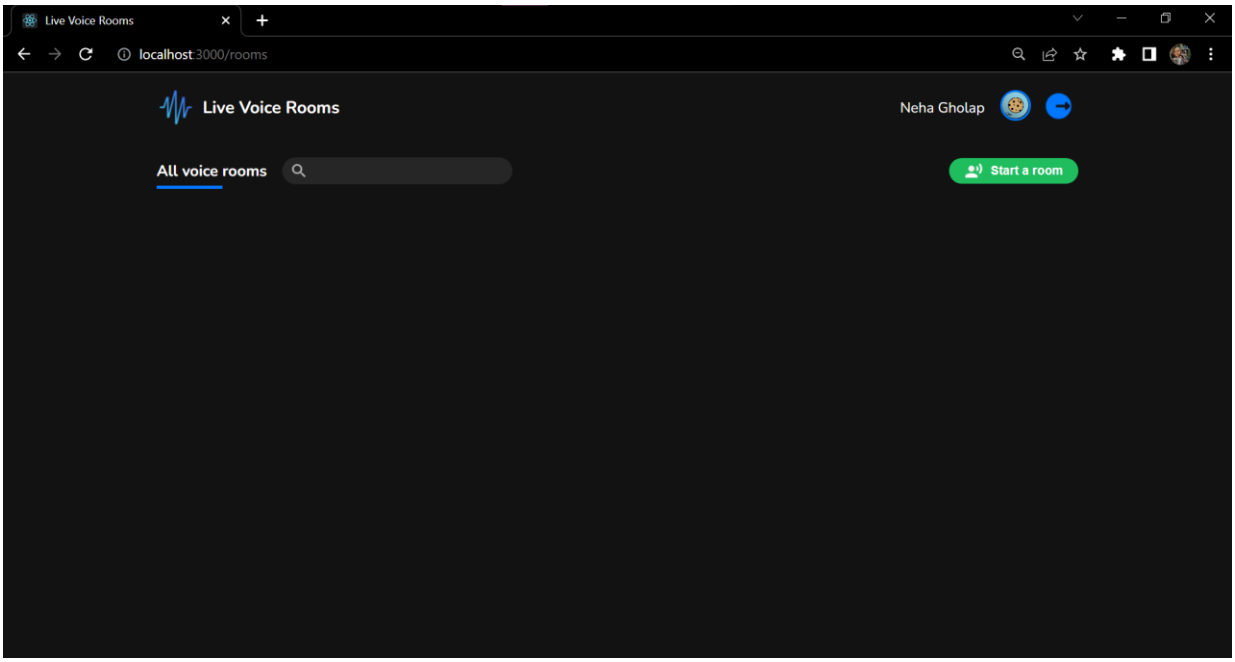
5.5 Name Input Screen



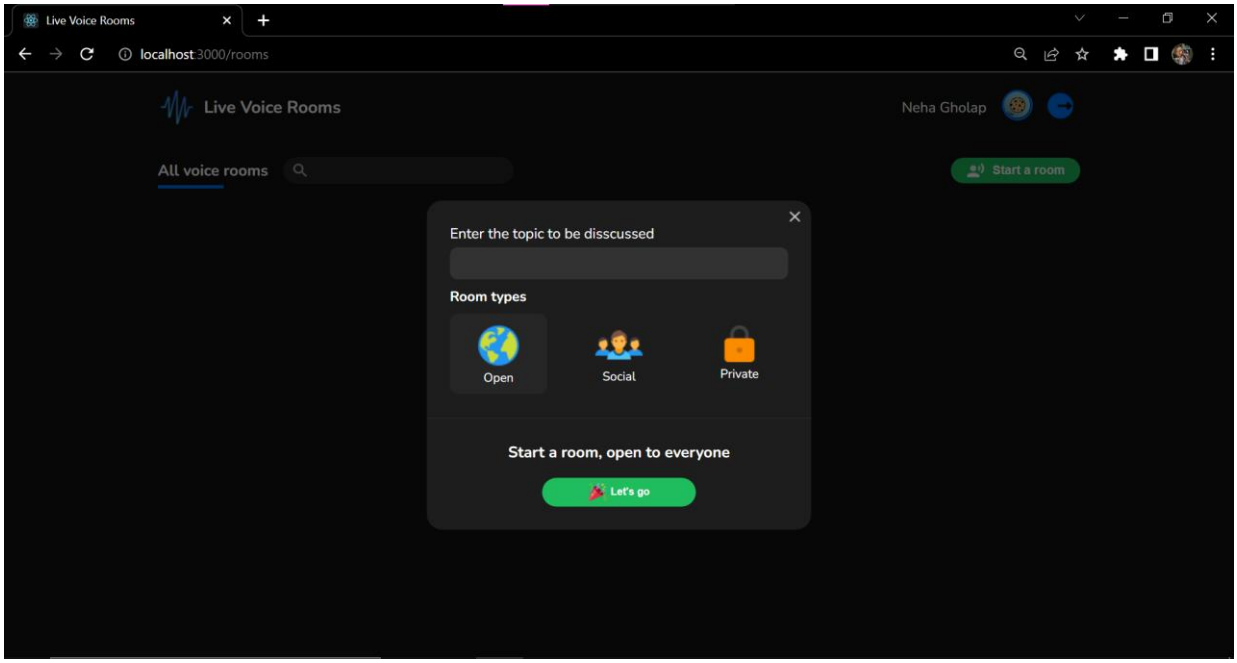
5.6 Photo uploading Screen



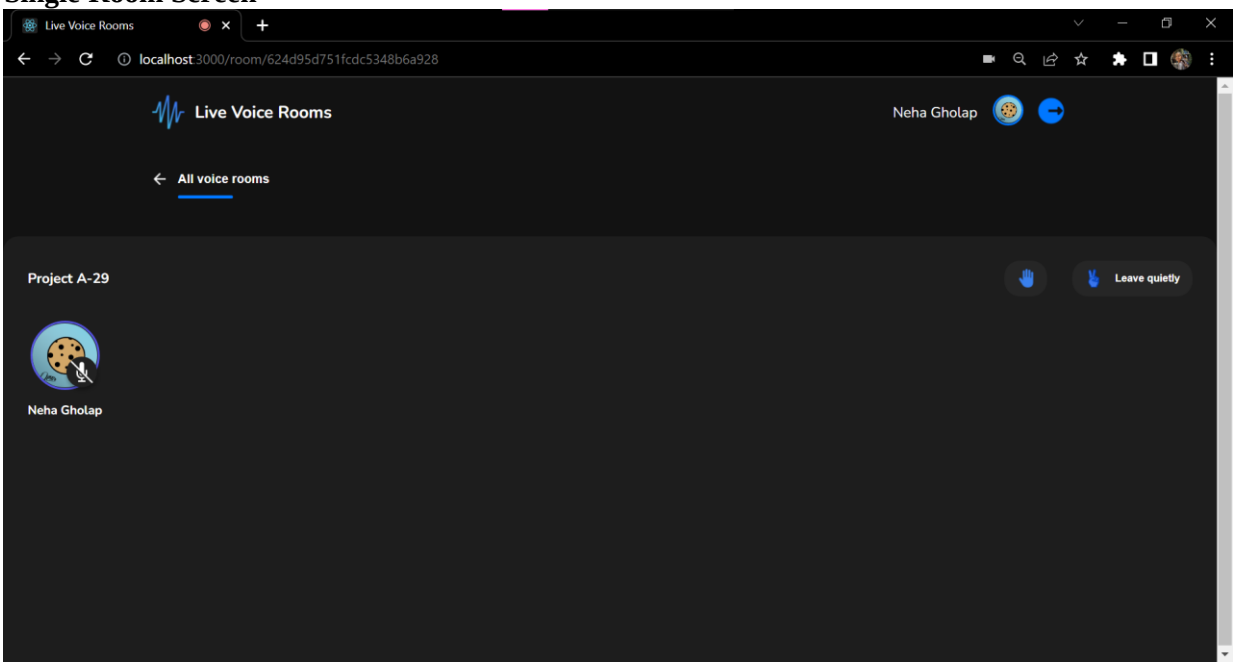
5.7 Rooms List Screen



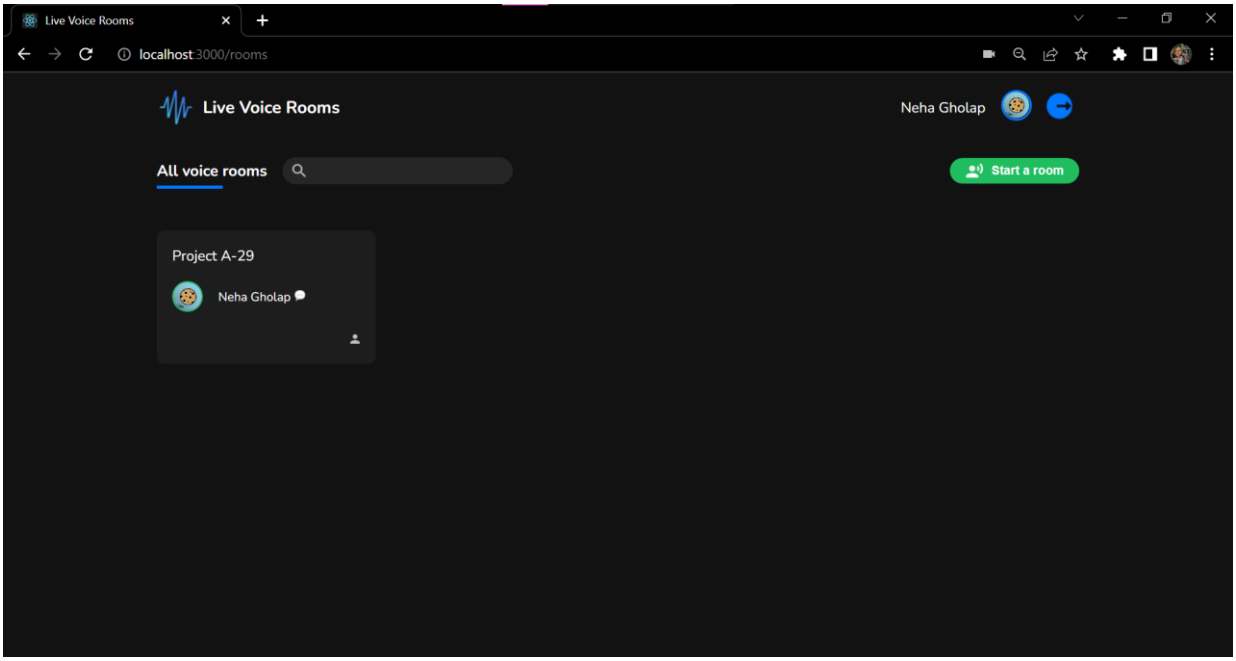
5.8 Room Type Screen



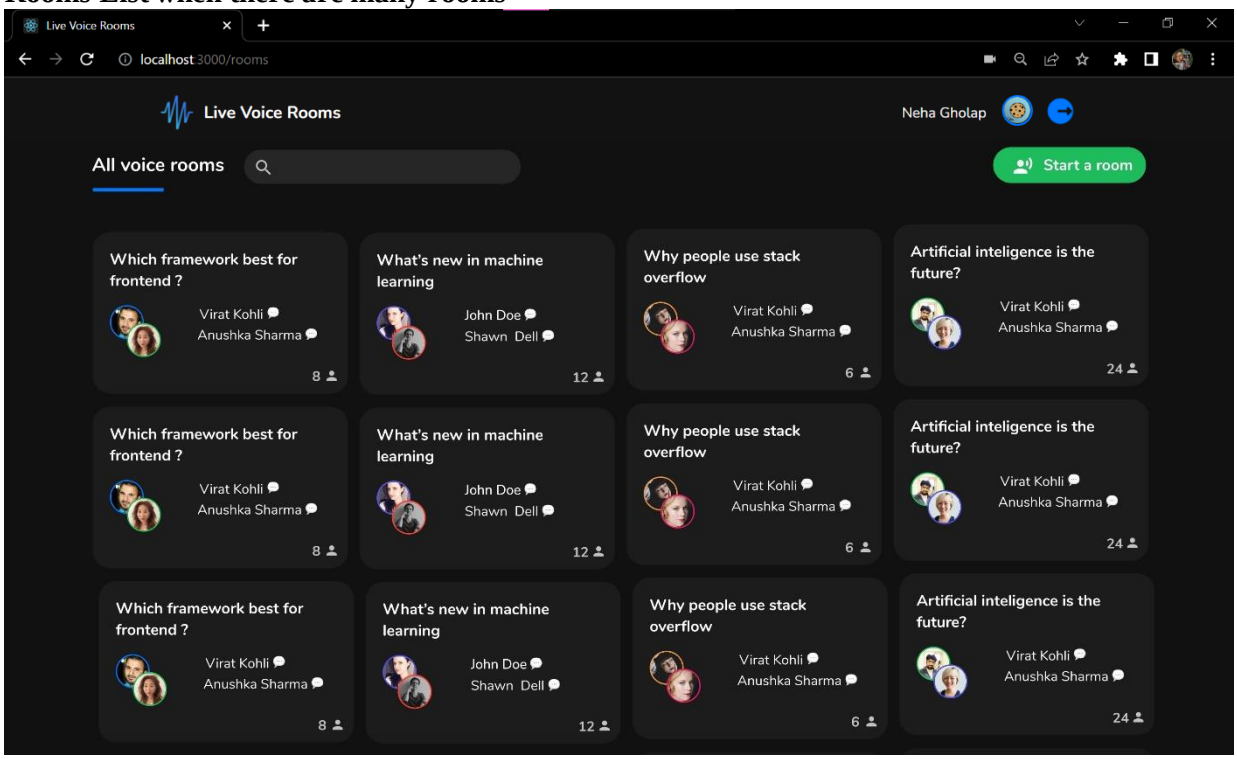
5.9 Single Room Screen



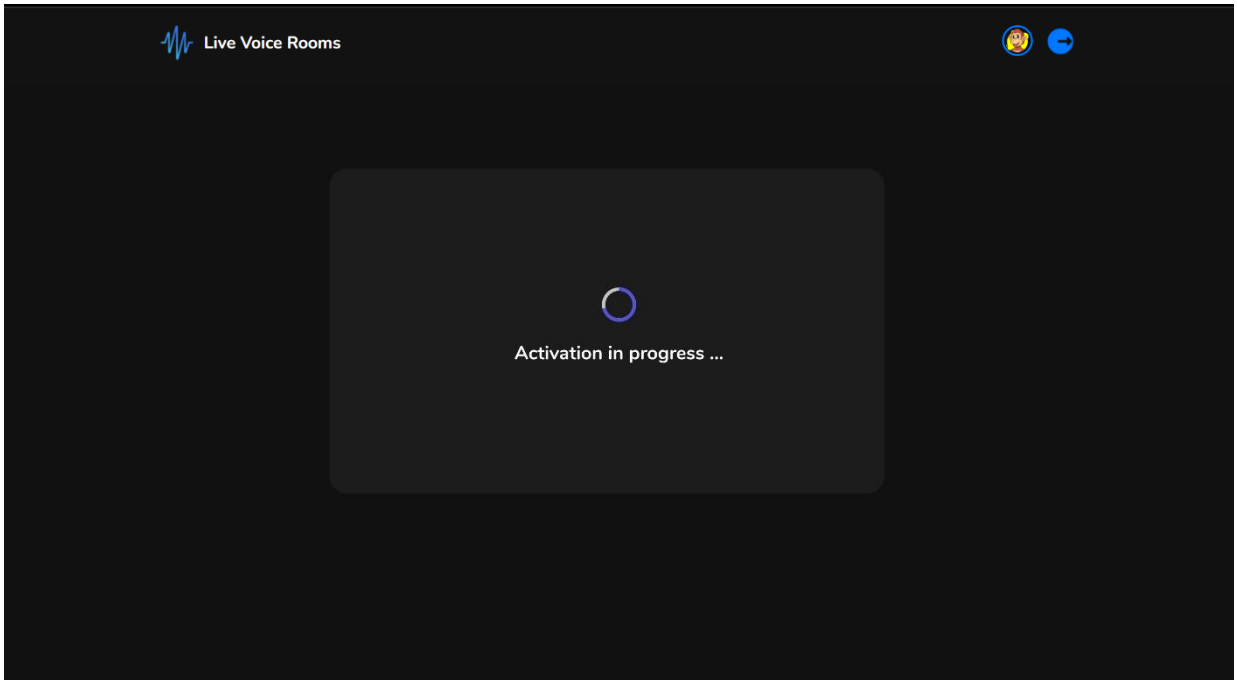
5.10 Updated Rooms List



5.11 Rooms List when there are many rooms



5.12 Loading/Activation Screen



CHAPTER 6: CONCLUSION

6.1 CONCLUSION

The idea of a social network with voice communication can turn out to be more attractive to people than the ordinary podcasts or television talk shows because now you can ask speakers questions and even engage in casual, unpolished conversations with famous and influential people.

People can use this app in a variety of ways. The app is for aspiring creators, business owners, and entrepreneurs, but many people can join the different rooms and have fun. The app is versatile, and it has many rooms for the users like talent shows, ‘meet new people’ rooms, and rooms discussing trending topics.

CHAPTER 8: REFERENCES

References :

1. https://www.tutorialspoint.com/
2. https://www.stackoverflow.com/
3. https://geeksforgeeks.com/
4. https://socket.io/docs/v4/
5. https://www.npmjs.com/package/bcrypt
6. https://reactjs.org/docs/getting-started.html
7. https://webrtc.org/
8. https://www.twilio.com/docs/sms/app-verification