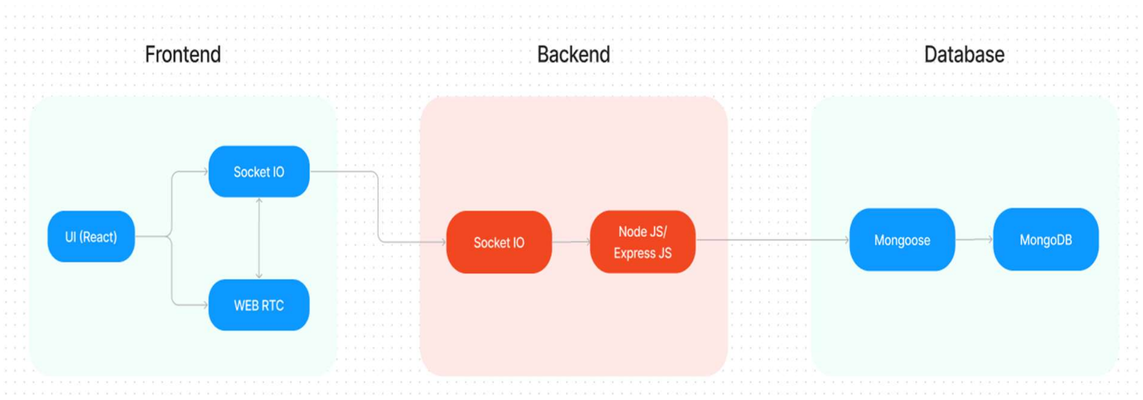
1. Introduction

- **Project Title:** VideoLink - Video Conferencing Application
- **Team Members:** Shobhit Khare (Full Development)

2. Project Overview

- **Purpose:** VideoLink is a web-based video conferencing application designed to provide users with a seamless and reliable platform for online meetings, collaboration, and communication. With its innovative features, seamless user experience, and robust security measures, it empowers individuals and businesses to communicate effectively, regardless of physical distance. Whether it's a casual catch-up with friends, a virtual team meeting, or a large-scale webinar, VideoLink is the ultimate solution for all your video conferencing needs.
- **Features:**
  - Real-time video and audio communication
  - Screen sharing
  - Text chat during meetings
  - User authentication and authorization
  - Secure meeting rooms
  - Responsive design for cross-device compatibility
  - Meeting scheduling
  - Recording meetings
  - Downloading meeting recordings

3. Architecture



- **Frontend:** The frontend utilizes the socket.io-client to establish real-time bidirectional communication with the backend server. This enables seamless and instant exchange of audio, video, and chat data between participants during the video conference. Additionally, the WebRTC API plays a crucial role in facilitating peer-to-peer communication, enabling direct audio and video streaming between users without the need for intermediate servers.

- **Backend:** On the backend side, we employ socket.io and Express.js frameworks to handle the server-side logic and communication. Socket.io allows for real-time event-based communication between the server and clients, enabling efficient synchronization of video conference data and seamless updates across all participants.

- **Database:** For data storage and retrieval, our backend relies on MongoDB. MongoDB allows for efficient and scalable storage of user data, including user profiles, meeting schedules. It ensures reliable and quick access to the necessary information during video conferences.

4. Setup Instructions

- **Prerequisites:**
  For frontend, we use:

- React.Js
- Socket.io-client
- Bootstrap
- Material UI
- Axios
- recordrtc
- downloadjs
- agora-rtc-react
- agora-rtc-sdk-ng

For backend, we use:

- bcrypt
- body-parser
- cors
- dotenv
- express
- http
- jsonwebtoken
- mongoose
- socket.io

- **Installation:**
  1. Clone the repository: `git clone https://github.com/skhare075/VideoLink--a-video-conferencing-app`
  2. Navigate to the project directory: `cd <your project directory>`
  3. Install frontend dependencies:
     - `cd client`
     - `npm install`
  4. Install backend dependencies:
     - `cd ../server`
     - `npm install`
  5. Set up environment variables:
     - Create a `.env` file in the `server` directory.
     - Add the following environment variables:

```
JWT_SECRET=<your_jwt_secret_key >
```

5. Folder Structure

- **Client (React Frontend):**

```
client/
├── src/
│   ├── components/
│   │   ├── Chat.jsx
│   │   ├── MeetData.jsx
│   │   ├── Participant.jsx
│   │   ├── ProfileCard.jsx
│   │   ├── VideoPlayer.jsx
│   │   └── Controls.jsx
│   ├── context/
│   │   ├── AuthContext.jsx
│   │   └── SocketContext.jsx
│   ├── pages/
│   │   ├── Home.jsx
│   │   ├── Login.jsx
│   │   ├── MeetRoom.jsx
│   │   ├── Profile.jsx
│   │   ├── Register.jsx
│   ├── protectedRoute/
│   │   ├── RouteProtector.jsx
│   │   |-----LoginProtector.jsx
│   ├── styles/
│   │   ├── Home.css
│   │   ├── LoginRegister.css
│   │   ├── MeetData.css
│   │   ├── MeetPage.css
│   │   ├── Profile.css
│   │   ├── ProfileCrd.css
│   |---- AgoraRtMSetup.js
│   |----- AgoraSetup.js
│   ├── App.js
│   ├── App.test.js
│   ├── index.css
│   ├── index.js
│   ├── logo.svg
│   ├── reportWebVitals.js
│   ├── setupTests.js
│   └── socket.js
├── public/
│   ├── ...
├── package.json
|---- node_modules
```

- **Server (Node.js Backend):**

```
server/
├── controllers/
│   ├── auth.js
├── middleware/
│   ├── auth.js
├── models/
│   ├── Rooms.js
│   ├── User.js
├── routes/
│   ├── auth.js
├── socket/
│   ├── roomHandler.js
```

```
├── index.js
|---- .env
├── package-lock.json
└── package.json
|----- node_modules
```

6. Running the Application

- **Frontend:**
    1. Navigate to the `client` directory: `cd client`
    2. Start the development server: `npm start`
- **Backend:**
    1. Navigate to the `server` directory: `cd server`
    2. Start the server: `node index.js`

7. API Documentation

1. User Authentication:

POST /api/auth/register:

- **Description**: Register a new user.
- **Request Parameters**:

  - username (string, required): The username of the new user.
  - password (string, required): The password of the new user.
  - email (string, required): The email of the new user.

Example Request:

```
{
 "username": "Sandeep_kumar",
 "password": "password123",
 "email": "sandeep@example.com"
   }
```

Example Response:

```
{
 "message": "User registered successfully",
 "userId": "12345"
   }
```

POST /api/auth/login:

- **Description:** Authenticate a user and return a token.
- **Request Parameters:**

  - username(string, required): The username of the user.
  - password(string, required): The password of the user.

Example Request:

```
{
 "username": "Sandeep_kumar",
 "password": "password123"
   }
```

Example Response:

```
{
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
```

2. Video Conferencing:

POST /api/conference/create:

- **Description :** Create a new video conference.
- **Request Parameters:**

  - `title` (string, required): The title of the conference.
  - `hostId` (string, required): The ID of the host user.

Example Request:

```
{
  "title": "Team Meeting",
  "hostId": "12345"
    }
```

Example Response:

```
{
 "conferenceId": "67890",
  "title": "Team Meeting",
  "hostId": "12345",
  "createdAt": "2025-03-11T10:00:00Z"
   }
```

GET /api/conference/{conferenceId}:

- **Description**: Get details of a specific conference.
- **Request Parameters**:

  - `conferenceId` (string, required): The ID of the conference.

Example Response:

```
 {
"conferenceId": "67890",
 "title": "Team Meeting",
 "hostId": "12345",
  "participants": [
{
  "userId": "12345",
   "username": "Sandeep kumar"
```

```
      },
    {
      "userId": "67890",
      "username": "Sandeep_kumar"
      }
        ],
      "createdAt": "2025-03-11T10:00:00Z"
        }
```

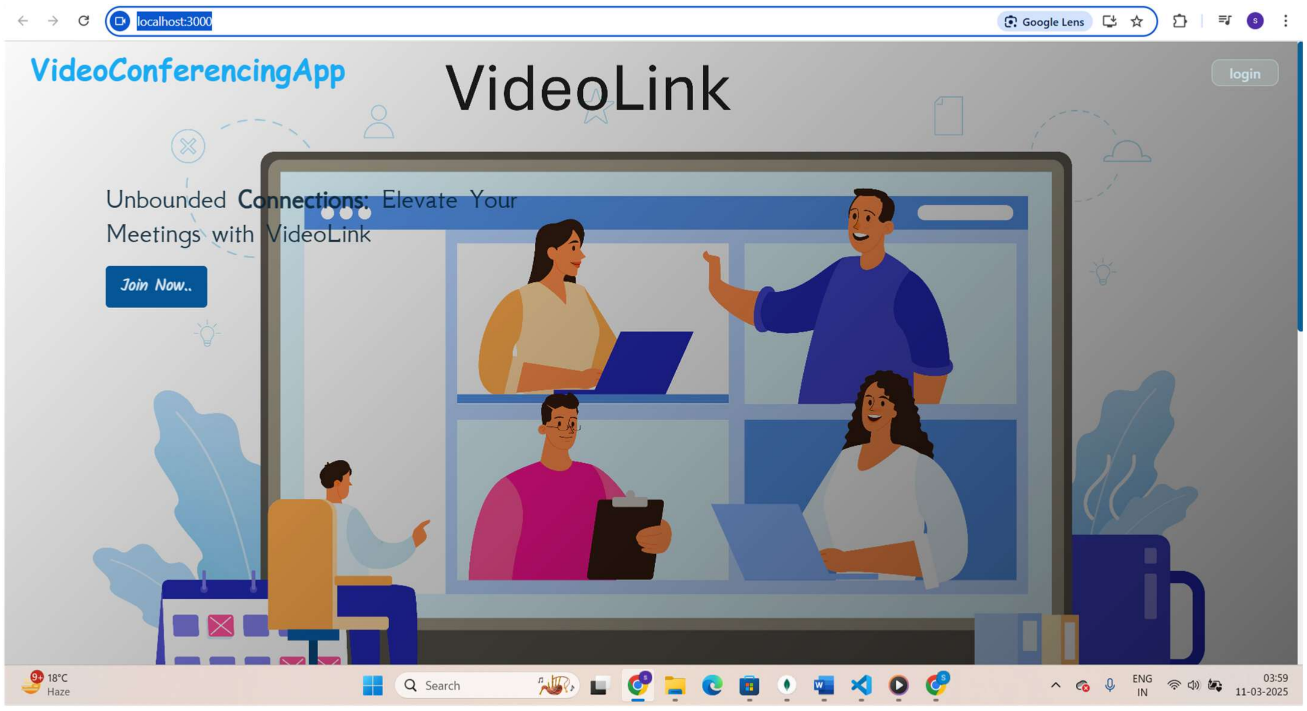
8. Authentication

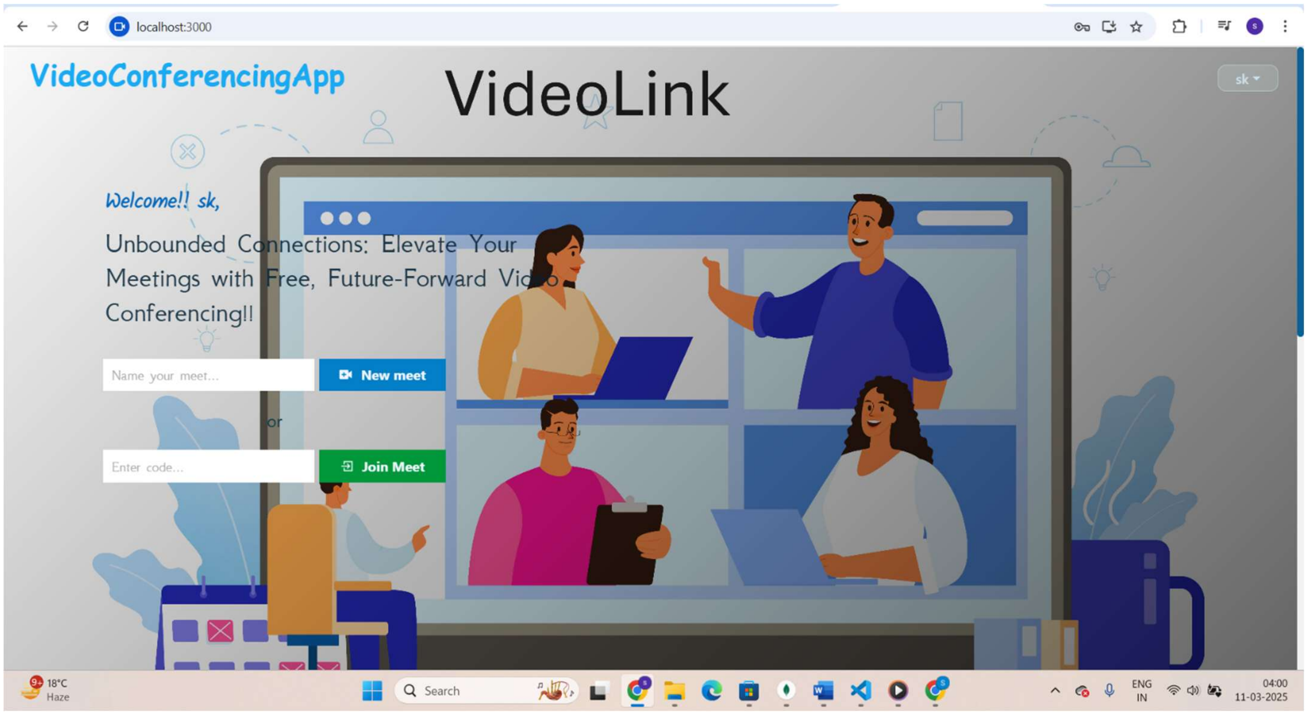
Authentication and authorization are handled using JSON Web Tokens (JWT). When a user registers or logs in, the backend generates a JWT containing the user's ID and other relevant information. This token is then sent back to the client, which stores it (e.g., in local storage or cookies).

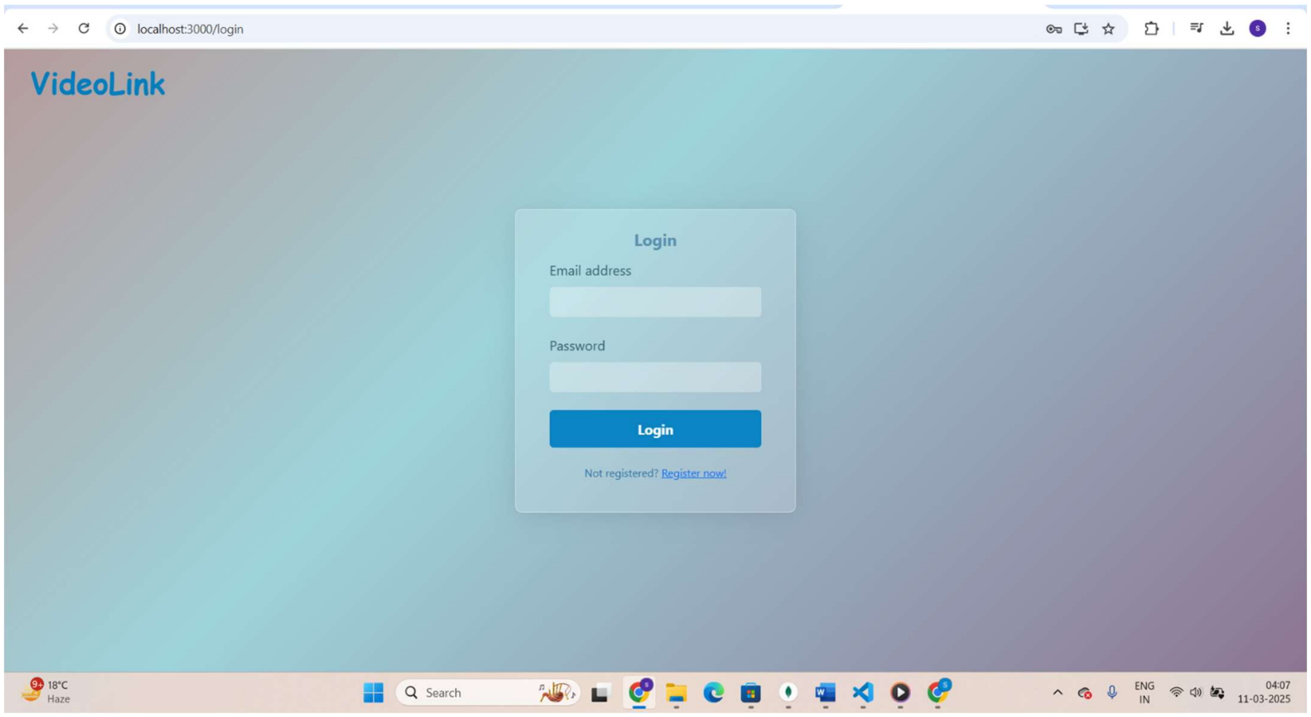
Whenever the client needs to access protected resources on the backend, it includes the JWT in the `Authorization` header of the HTTP request. The backend then verifies the token to ensure that the user is authenticated and authorized to access the requested resource.

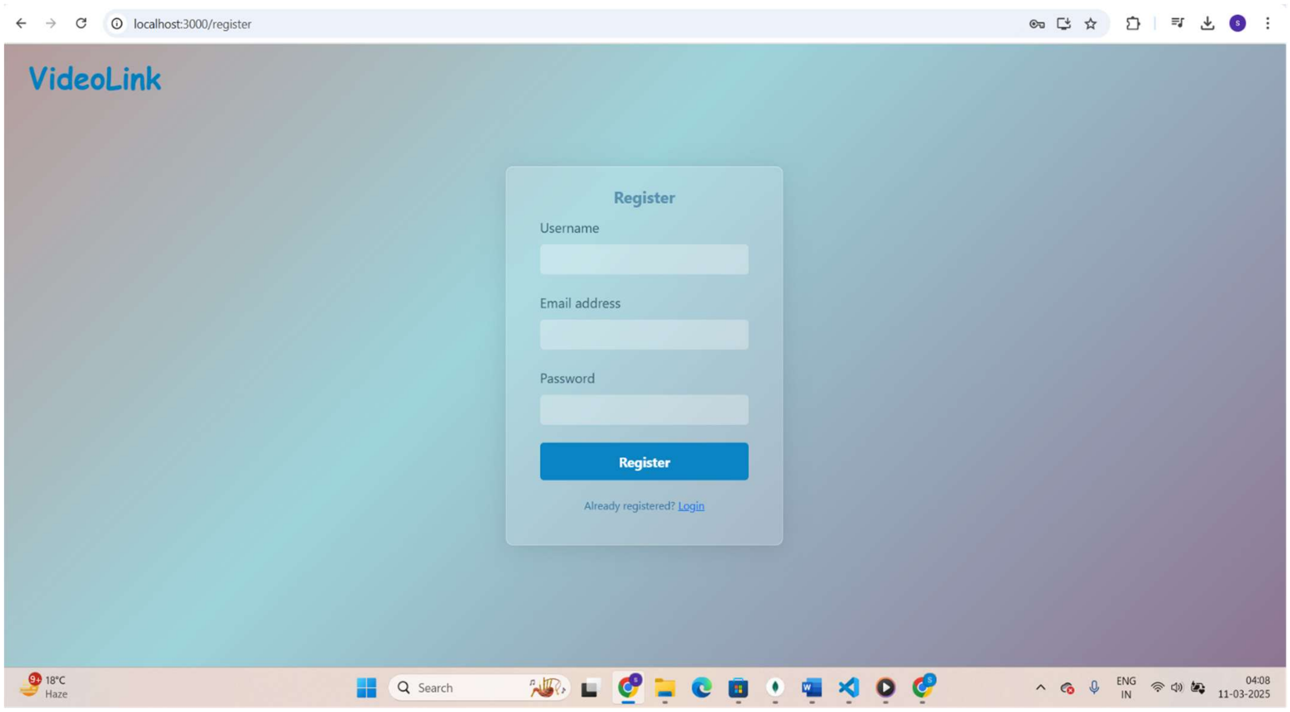
- **Middleware:** The `auth.js` middleware in the `server/middleware` directory handles the verification of JWT tokens.

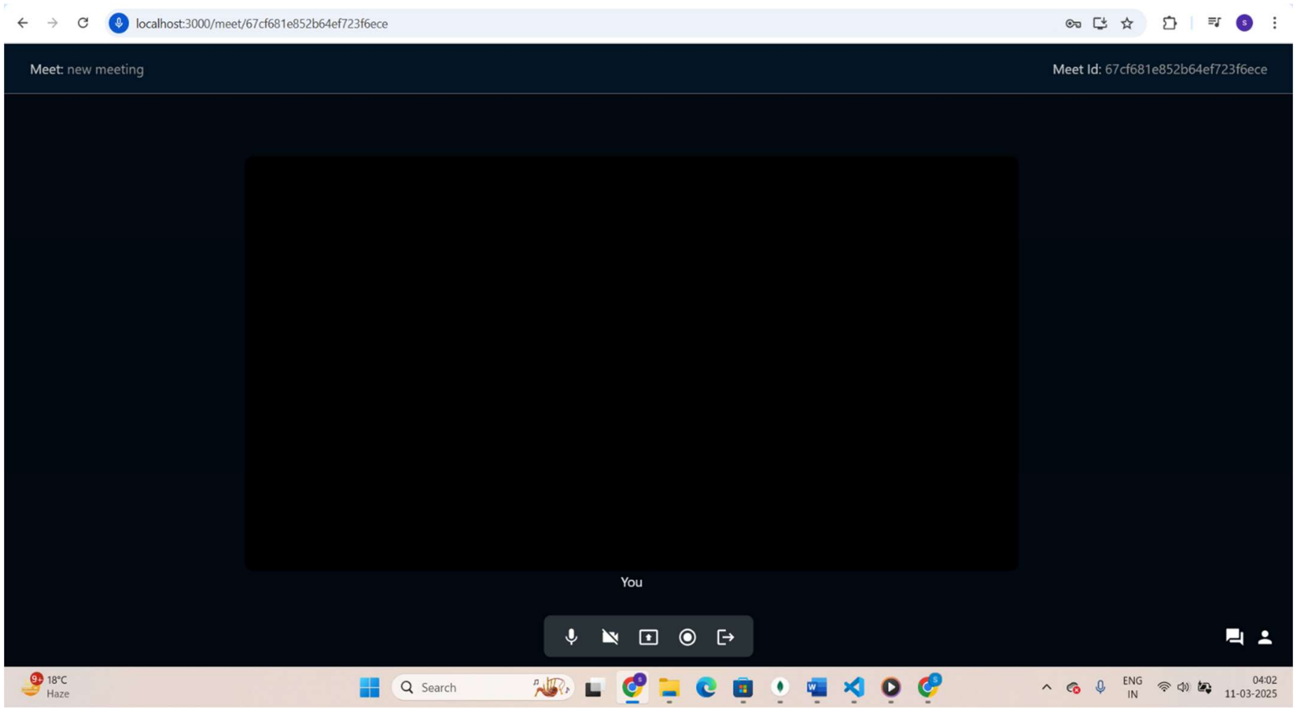
9. User Interface

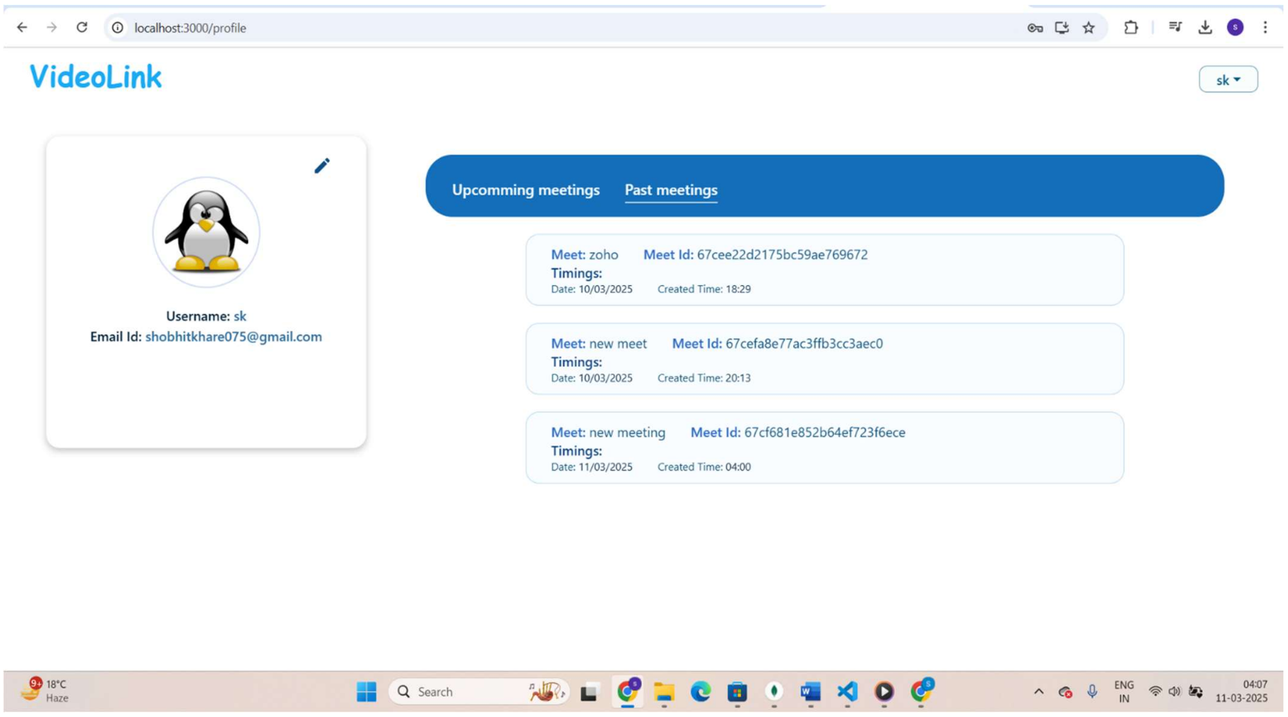



localhost:3000
VideoConferencingApp
VideoLink
sk
Welcome!! sk,
Unbounded Connections: Elevate Your Meetings with Free, Future-Forward Video Conferencing!!
Name your meet...
New meet
or
Enter code...
Join Meet
18°C Haze
Search
ENG IN
04:00 11-03-2025


localhost:3000/login
VideoLink
Login
Email address
Password
Login
Not registered? Register now!
18°C Haze
Search
ENG IN
04:07 11-03-2025


localhost:3000/register
VideoLink
Register
Username
Email address
Password
Register
Already registered? Login
18°C
Haze
Search
ENG
IN
04:08
11-03-2025


localhost:3000/meet/67cf681e852b64ef723f6ece
Meet: new meeting
Meet Id: 67cf681e852b64ef723f6ece
You
18°C
Haze
Search
ENG
IN
04:02
11-03-2025



Username: sk
Email Id: shobhitkhare075@gmail.com

Meet: zoho Meet Id: 67cee22d2175bc59ae769672
Timings:
Date: 10/03/2025 Created Time: 18:29

Meet: new meet Meet Id: 67cefa8e77ac3ffb3cc3aec0
Timings:
Date: 10/03/2025 Created Time: 20:13

Meet: new meeting Meet Id: 67cf681e852b64ef723f6ece
Timings:
Date: 11/03/2025 Created Time: 04:00

10. Testing

1. Unit Testing:
   - Purpose: To test individual components or functions to ensure they work as expected.
   - Tools Used: Jest.
   - Example: Testing the user authentication functions to ensure they correctly validate user credentials.
2. Integration Testing:
   - Purpose: To test the interaction between different components of the application.
   - Tools Used: Postman.
   - Example: Testing the API endpoints to ensure they correctly handle requests and responses.
3. End-to-End (E2E) Testing:
   - Purpose: To test the entire application flow from start to finish.
   - Tools Used: Cypress, Selenium.
   - Example: Testing the user journey from logging in to joining a video conference and sending a chat message.

11. Screenshots or Demo

https://drive.google.com/file/d/1wLDY7R3Il7dDwqQGyPcFs6Y8868BXNae/view?usp=sharing

12. Known Issues

- If a user is already registered, it doesn't give any message or error.

13. Future Enhancements

- Implement end-to-end encryption for enhanced security.
- Add support for more advanced meeting scheduling options (e.g., recurring meetings).
- Improve the UI/UX based on user feedback.
- Add more advanced moderation controls for meeting hosts.
- Implement AI noise cancellation techniques.
- Address bandwidth and latency issues.