

# JAYPEE INSTITUTE OF INFORMATION AND TECHNOLOGY



## Computer Networks and Internet of Things [B15CS311]

**TOPIC: Peak-A-View**

**BATCH:** B9

**SUBMITTED TO:**

**Dr. Vikash**

**Dr. Meenal Jain**

**Ms. Amarjeet Kaur**

**SUBMITTED BY:**

**Kritarth Bansal      20103256**

**Ayush Raj              20103261**

**Saksham Gupta      20103268**

## TABLE OF CONTENT

TOPIC	PAGE NO.
1. Abstract of the project	3
2. Scope of the project	4
2.1. Functional Requirements	4
2.2. Non-functional Requirements	4
2.3. Description of the modules of the project	4
3. Tools and Technologies used	5
4. Design of the Project	6
5. Implementation Details	8
6. Testing Details	9
7. References	13

## **Abstract of the project**

The purpose of this project is to create a screen sharing application that allows users to share their screens with others in real-time. The application is built using HTML, CSS, and JavaScript on the front-end, and React.js for user interface design. The back-end of the application is developed using Node.js and Electron.js, which enable cross-platform compatibility. The Web Socket Protocol is utilized for real-time data transmission between the client and server.

The application utilizes various Node modules such as Electron, Screenshot-desktop, Socket.io, CORS, Express, and UUID. Electron is used to create a native desktop application for cross-platform compatibility, while Screenshot-desktop is used to capture screen images for sharing. Socket.io is used for real-time communication between the client and server, while CORS is used for cross-origin resource sharing. Express is used as the server framework, and UUID is used for generating unique IDs for each session.

The screen sharing application allows users to share their screens with others by creating a unique session ID. Other users can join the session by entering the session ID on their devices. The application supports multiple users simultaneously and provides options for controlling the sharing permissions. Users can choose to share their entire screen or select specific windows or applications to share.

The application has been designed to provide a simple and intuitive user experience. The UI is designed to be user-friendly and responsive, and the application is designed to be easy to use even for users with limited technical knowledge.

Overall, this project is aimed at creating a user-friendly and efficient screen sharing application that utilizes the latest technologies to provide a seamless experience. The application has the potential to be used in various settings, including remote work, online education, and collaborative projects.

## **Scope of the Project**

### **Functional requirements**

1. Screen Sharing: The application must allow users to share their screens with other users in real-time.
2. Session Management: The application must allow users to create and manage unique session IDs for screen sharing.
3. Permission Control: The application must allow users to control the sharing permissions, including selecting specific windows or applications to share.
4. Multi-User Support: The application must support multiple users simultaneously.
5. Cross-Platform Compatibility: The application must work on various operating systems, including Windows, macOS, and Linux.

### **Non-functional requirements**

1. Security: The application must ensure secure data transmission between the client and server.
2. Reliability: The application must be reliable and stable, with minimal downtime.
3. Performance: The application must provide efficient screen sharing performance, with minimal lag or delay.
4. Usability: The application must have an intuitive and user-friendly interface, with clear instructions for users with limited technical knowledge.

### **Description of the Modules of the Project**

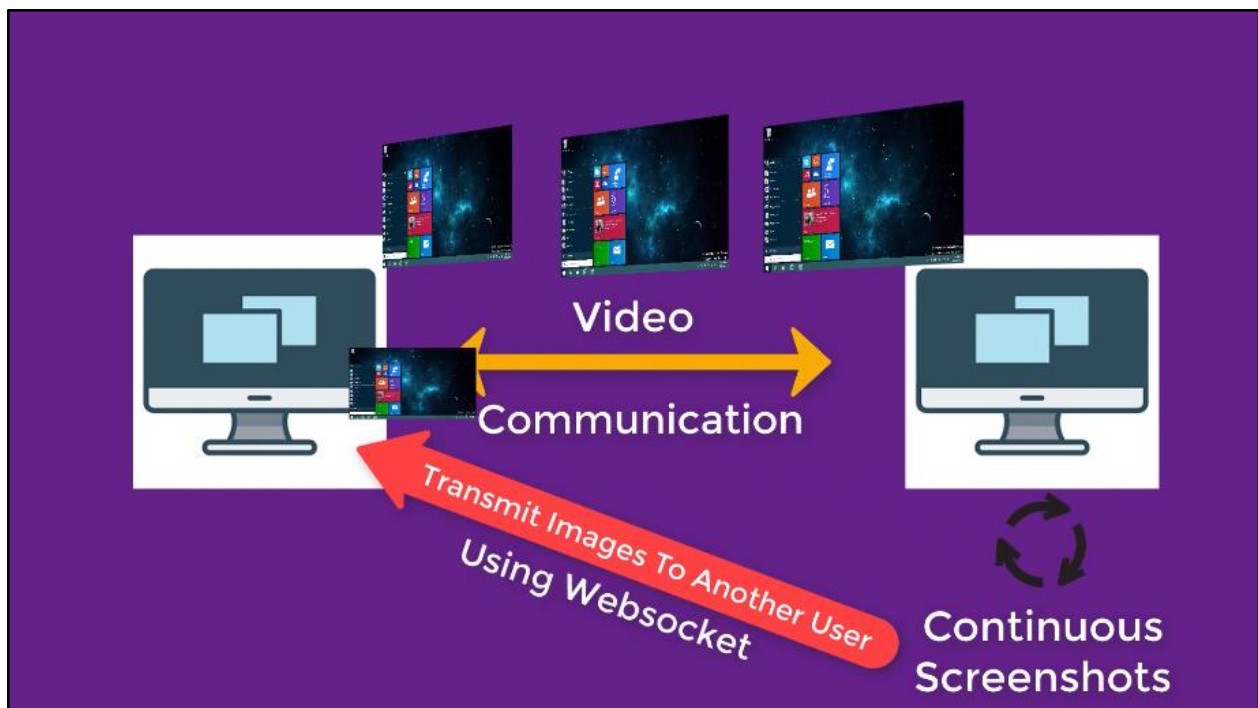
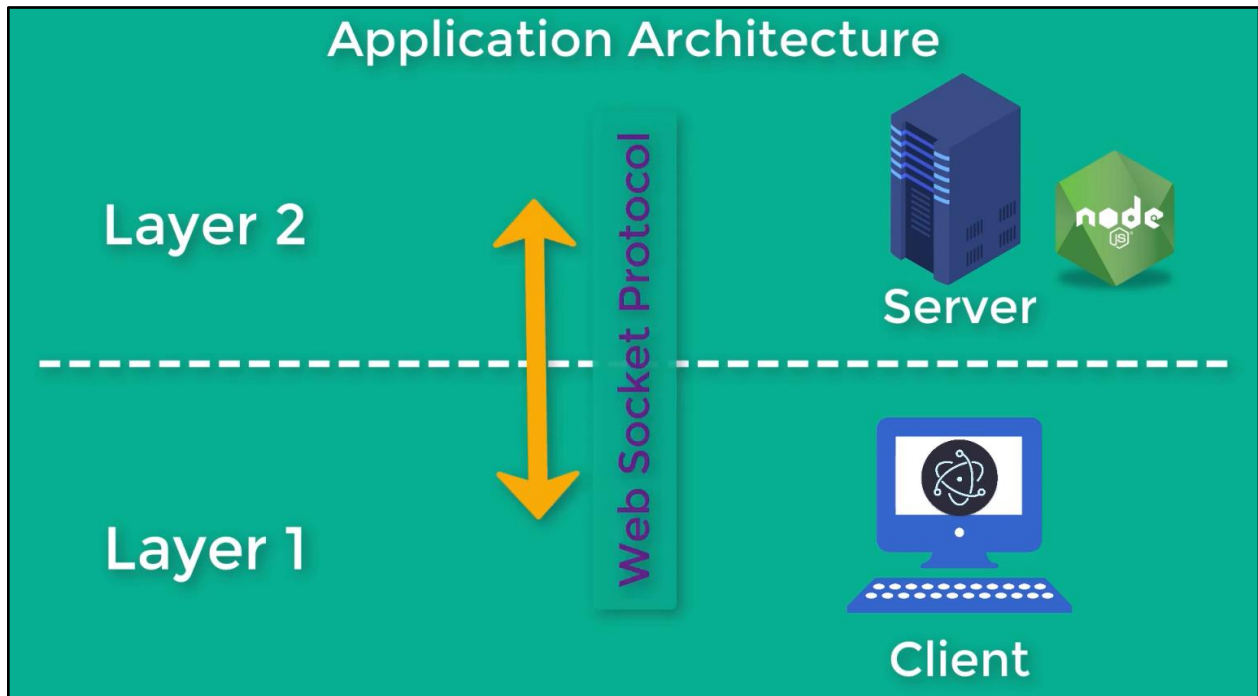
1. Front-end Module: The front-end module is responsible for the user interface design, built using HTML, CSS, and JavaScript. The module uses React.js for designing the user interface, which includes the session creation form, screen sharing options, and sharing permission control.
2. Back-end Module: The back-end module is developed using Node.js and Electron.js. The module is responsible for handling the server-side functionalities, including the creation and management of unique session IDs, screen capture, and real-time data transmission between the client and server using the Socket.io library.
3. Screenshot-desktop Module: The Screenshot-desktop module is used to capture screen images for sharing.
4. Socket.io Module: The Socket.io module is used for real-time communication between the client and server, allowing for efficient and low-latency data transmission.
5. Express Module: The Express module is used as the server framework, providing an easy-to-use and scalable solution for managing server-side functionalities.
6. CORS Module: The CORS module is used for cross-origin resource sharing, enabling communication between the client and server from different origins.

7. **UUID Module:** The UUID module is used for generating unique IDs for each session, ensuring that users can join the correct session and preventing unauthorized access to shared screens.

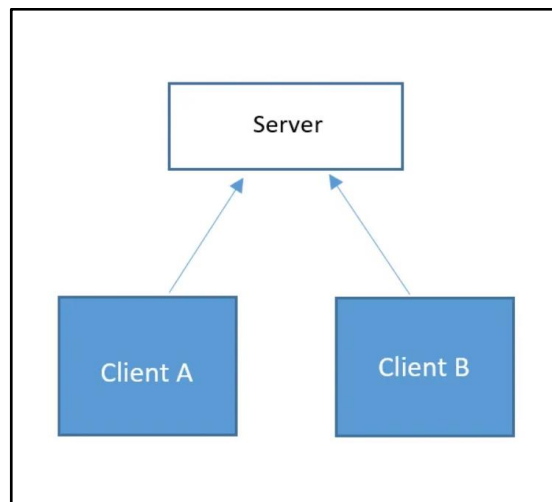
## **Tools and Technologies used**

1. **HTML, CSS, and JavaScript** - These are the core web technologies used for creating the user interface and client-side logic.
2. **React.js** - React is a popular JavaScript library for building user interfaces. It is used in this project for designing the user interface and managing the state of the application.
3. **Node.js** - Node.js is a JavaScript runtime that allows developers to build server-side applications using JavaScript.
4. **Electron.js** - Electron.js is a framework for building desktop applications using web technologies. It is used in this project for creating a native desktop application for cross-platform compatibility.
5. **Web Socket Protocol** - The Web Socket Protocol is used for real-time data transmission between the client and server.
6. **Socket.io** - Socket.io is a JavaScript library for real-time communication between the client and server.
7. **Screenshot-desktop** - Screenshot-desktop is a Node.js module used to capture screen images for sharing.
8. **Express** - Express is a popular Node.js web framework used for building server-side applications.
9. **CORS** - CORS is a Node.js module used for cross-origin resource sharing, enabling communication between the client and server from different origins.
10. **UUID** - UUID is a Node.js module used for generating unique IDs for each session.

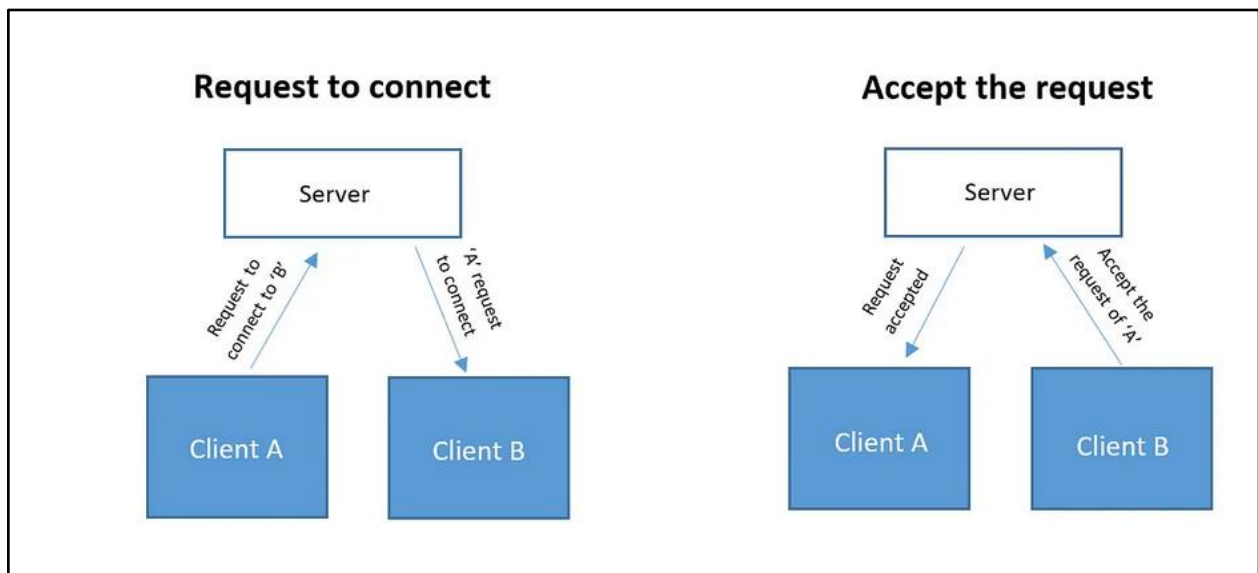
## Design of the Project



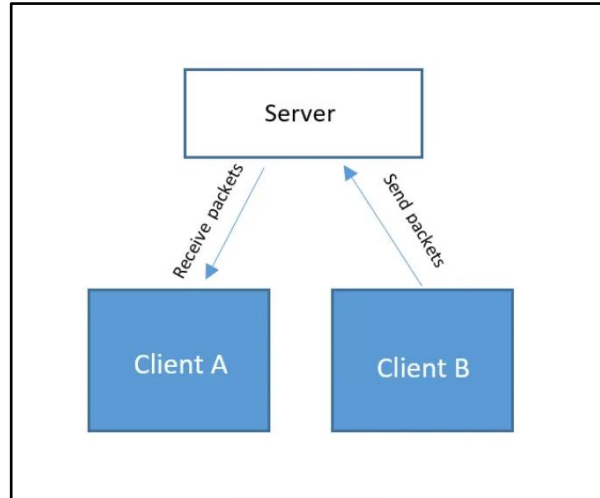
- Clients will connect to the socket server through which they'll communicate with each other.



- Any client could request to connect to any other client via server. Consider client A wants to connect to client B, so client A will request the server to connect to client B. Server will forward that request to client B.



- If the client accepts the request, it will be notified to the source client via server. Client B will accept the request of client A and the server will notify to client A. Then client B will start sending the data packets to client A.



## **Implementation Details**

### 1. Setting up the Development Environment:

The development environment for the application can be set up by installing Node.js, Electron.js, and the required Node modules (Socket.io, Express, Screenshot-desktop, UUID, and CORS). The front-end of the application can be developed using a code editor and a web browser, while the back-end can be developed using Node.js and Electron.js.

### 2. Front-end Development:

The front-end of the application is developed using HTML, CSS, and JavaScript. React.js is used for designing the user interface, which includes the session creation form, screen sharing options, and sharing permission control. The front-end communicates with the back-end using Socket.io, enabling real-time communication between the client and server.

### 3. Back-end Development:

The back-end of the application is developed using Node.js and web socket protocol. The module is responsible for handling the server-side functionalities, including the creation and management of unique session IDs, screen capture, and real-time data transmission between the client and server using the Socket.io library. Express is used as the server



framework, providing an easy-to-use and scalable solution for managing server-side functionalities.

4. Screen Sharing Functionality:

The screen sharing functionality is implemented using the Screenshot-desktop module, which captures screen images for sharing. The sharing permissions are controlled using Socket.io, which enables real-time communication between the client and server.

5. Session Management:

The session management functionality is implemented using the UUID module, which generates unique IDs for each session. The back-end server stores the session ID and the sharing permissions for each session, allowing users to join the correct session and preventing unauthorized access to shared screens.

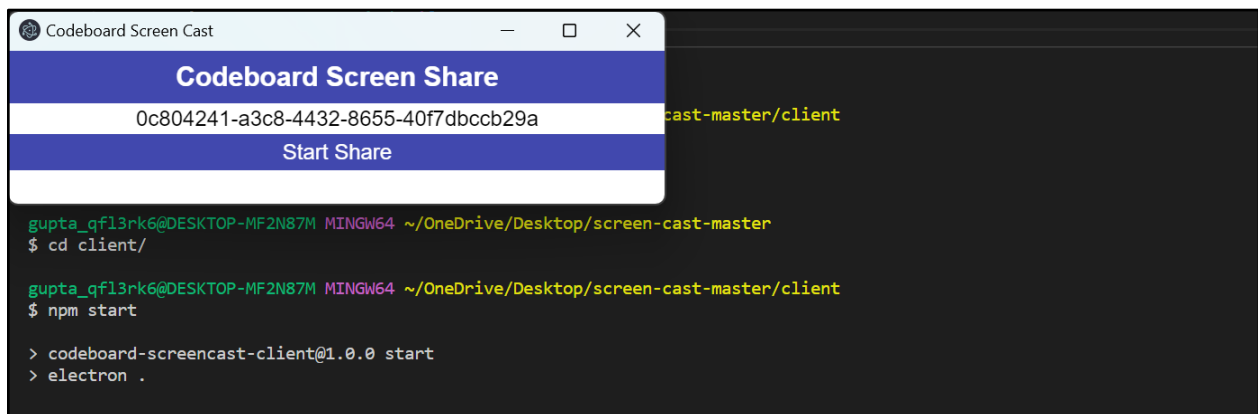
6. Cross-Platform Compatibility:

The application is made cross-platform compatible using Electron.js, which enables the creation of native desktop applications for Windows, macOS, and Linux. The application can be packaged as a desktop application for distribution, providing users with an easy-to-use solution for screen sharing across different operating systems.

Overall, the application is built using a combination of front-end and back-end technologies, providing users with a seamless and efficient solution for screen sharing. The application can be used in various settings, including remote work, online education, and collaborative projects.

## Testing Details

- To run a client side program, write `cd client` then `npm start`.



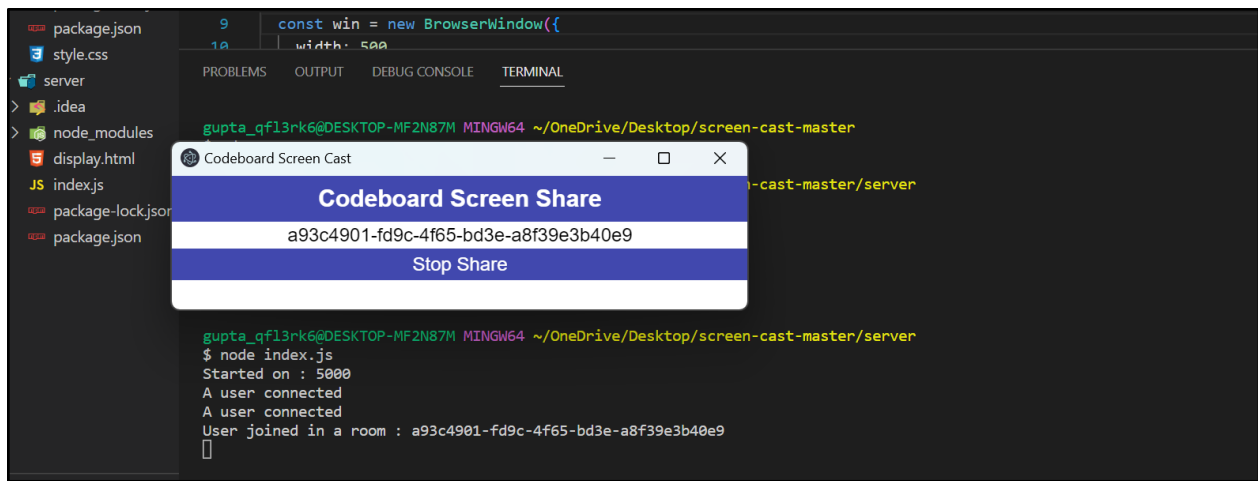
The screenshot shows a 'Codeboard Screen Cast' window in the foreground. The window has a blue header with the text 'Codeboard Screen Share'. Below the header, there is a white bar containing a long alphanumeric string: '0c804241-a3c8-4432-8655-40f7dbccb29a'. Below this, there is a blue bar with the text 'Start Share'. In the background, a terminal window is visible, showing the following commands and output:

```
gupta_qf13rk6@DESKTOP-MF2N87M MINGW64 ~/OneDrive/Desktop/screen-cast-master
$ cd client/

gupta_qf13rk6@DESKTOP-MF2N87M MINGW64 ~/OneDrive/Desktop/screen-cast-master/client
$ npm start

> codeboard-screencast-client@1.0.0 start
> electron .
```

- To run the server side program, write node index.js



The screenshot shows a code editor with a file explorer on the left containing files like package.json, style.css, server, .idea, node\_modules, display.html, index.js, package-lock.json, and package.json. The main editor area shows a JavaScript snippet: 

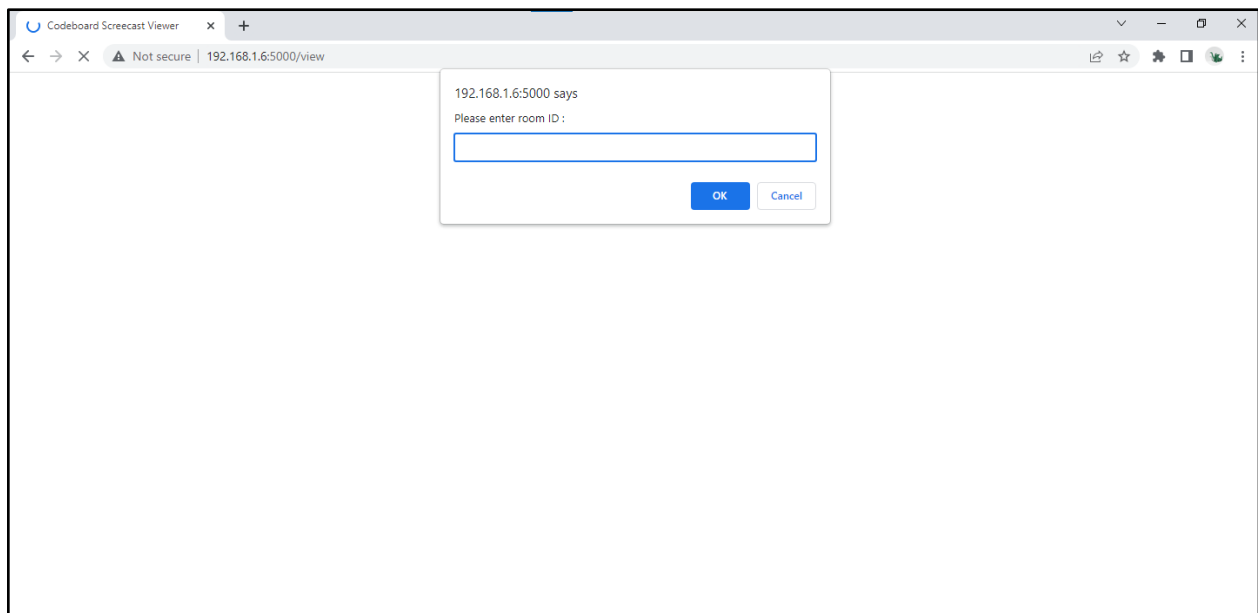
```
const win = new BrowserWindow({
  width: 500
```

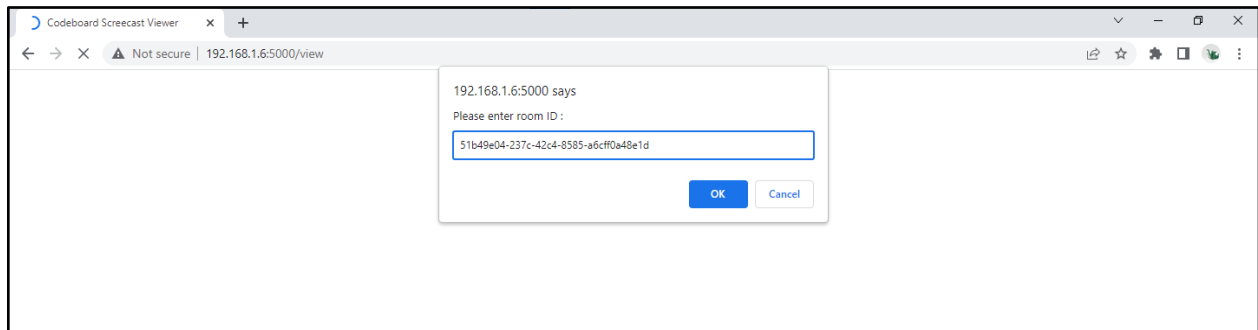
. Below the editor is a terminal window with the following output: 

```
gupta_qf13rk6@DESKTOP-MF2N87M MINGW64 ~/OneDrive/Desktop/screen-cast-master
$ node index.js
Started on : 5000
A user connected
A user connected
User joined in a room : a93c4901-fd9c-4f65-bd3e-a8f39e3b40e9
```

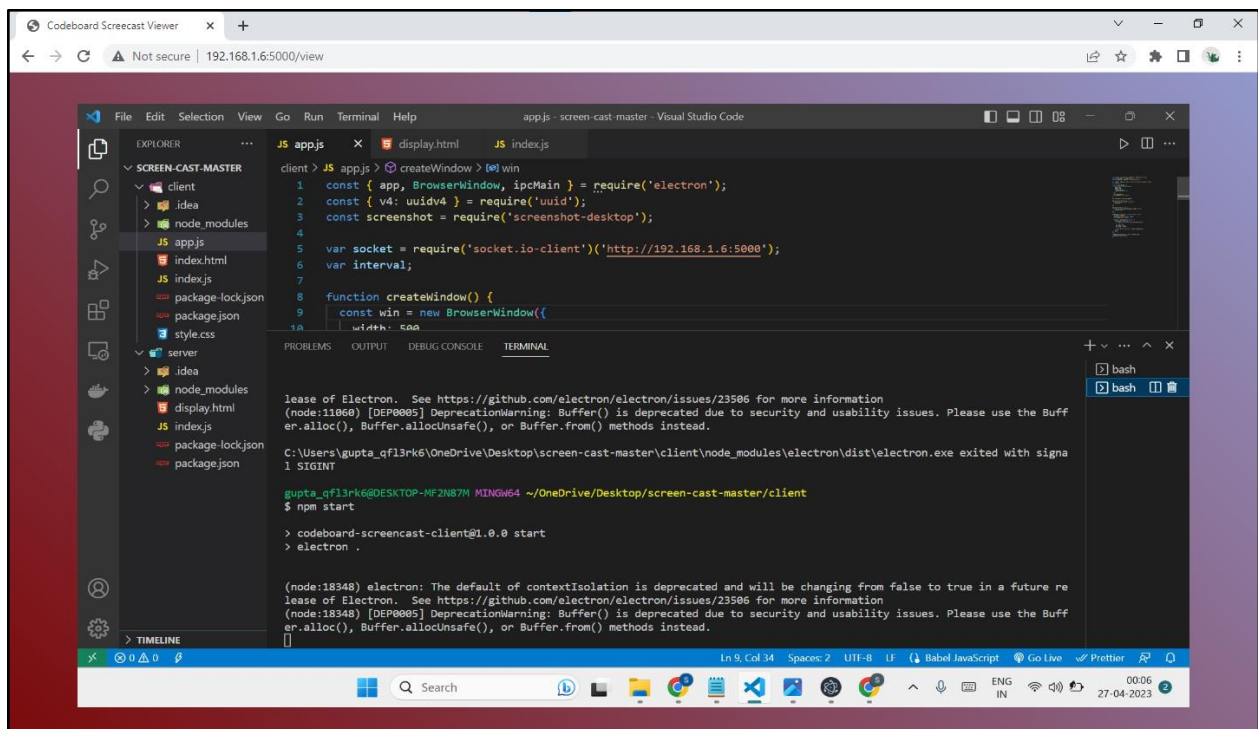
. A 'Codeboard Screen Cast' dialog box is overlaid on the terminal, displaying the room ID 'a93c4901-fd9c-4f65-bd3e-a8f39e3b40e9' and a 'Stop Share' button.

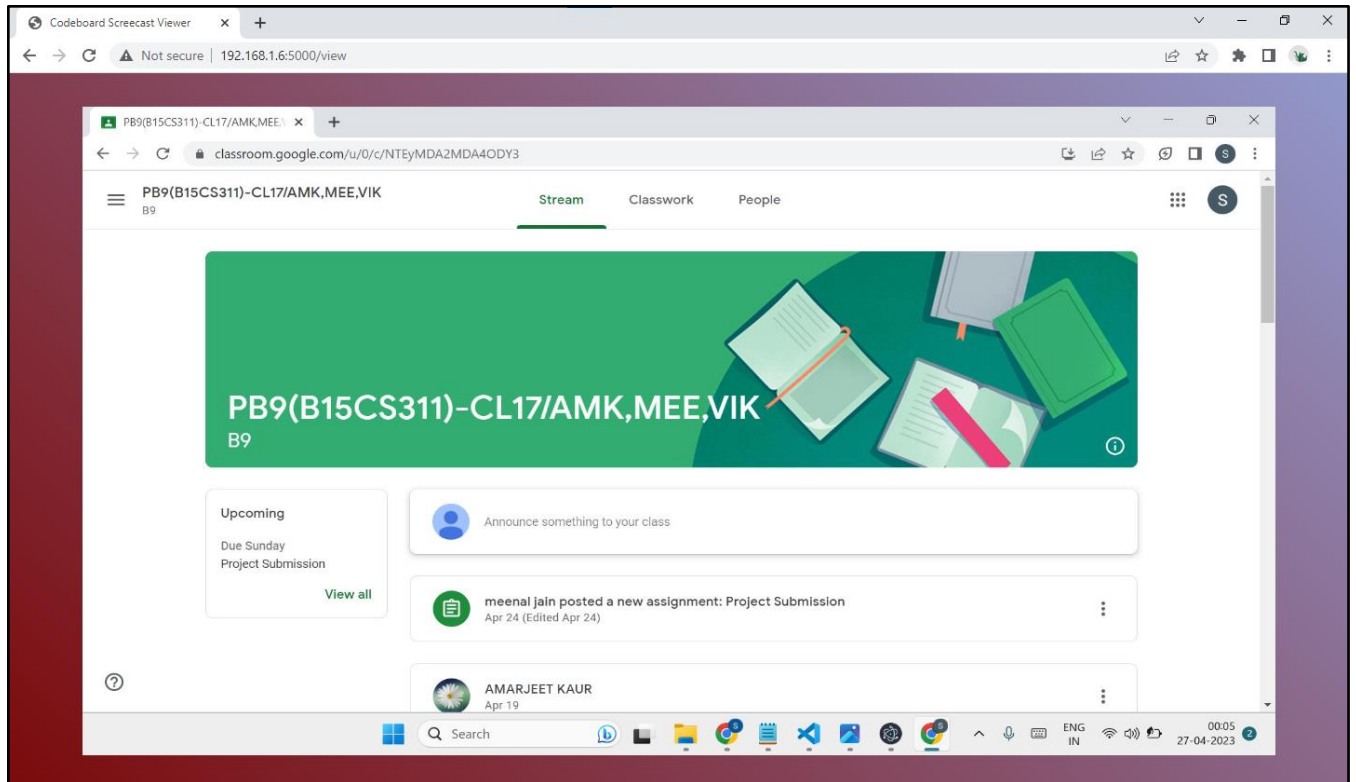
- Now open the url of the screen server application, we have to enter the room id to join the room.





- After entering the room id we can successfully view the screen of client





## References

- <https://betterprogramming.pub/desktop-sharing-application-with-nodejs-electron-and-socket-io-af389cb21d93>
- <https://www.electronjs.org/docs/latest/>
- <https://nodejs.org/en/docs>
- <https://legacy.reactjs.org/docs/getting-started.html>
- <https://expressjs.com/en/starter/installing.html>
- <https://javascript.info/websocket>