A PROJECT REPORT

on

"Code Editor Platform"

Submitted to

KIIT Deemed to be University

In Partial Fulfillment of the Requirement for the Award of

BACHELOR'S DEGREE IN

Computer Science and Engineering

BY

GROUP-1	NAME	Roll NO
A.	Swarnava Chakrabarti	2005062
B.	Aryan Jaiswal	21053012
C.	Ankur Kumar	21053007
D.	Harsh Bir	2005024

UNDER THE GUIDANCE OF

Chandani Kumari

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA - 751024

APRIL 2023

BONAFIDE CERTIFICATE

We are profoundly grateful to **Chandani Kumari** of Affiliation for his expert guidance and continuous encouragement throughout to see that this project meets its target since its commencement to its completion.

....

GROUP MEMBER

1. Swarnava Chakrabarti
2. Aryan Jaiswal
3. Ankur Kumar
4. Harsh Bir

DECLARATION

We are hereby declare that the project report entitled " CODE EDITOR PLATFORM APPLICATION USING REACTJS " done by us under the guidance of **Chandani Kumari** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering Degree in Computer Science and Engineering .

DATE :

PLACE : Bhubaneswar

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our project supervisor, **Chandani Kumari** , for providing guidance and support throughout the development of the Online Code Editor Platform and constant encouragement paved the way for the successful completion of my project work .

ABSTRACT

The Online Code Editor Platform is a web application that enables real-time collaboration between multiple users during the code editing . The application uses Web Socket to facilitate communication between the users and CodeMirror for syntax highlighting of the code . The project was developed using HTML, CSS, JavaScript, React.js, and Bootstrap, and the Web Socket and CodeMirror libraries were used to enable real-time collaboration and syntax highlighting, respectively. The UUID generator was used to generate unique IDs for users to join the single Socket .

The objective of this project was to address the issue of collaboration in software development teams. The traditional method of individual coding and sharing work through email or other communication channels was time-consuming and often led to errors and conflicts when merging different versions of the code. The Online Code Editor Platform was developed to enhance user engagement and increase productivity for teams working on code together and less time-consuming .

Online Code Editor Platform proved to enhance user engagement and increase productivity for teams working on code together. The application provides a user-friendly interface and the real-time updating of the code and chat sections enhances team productivity.

KEY WORDS : Real time code Editor , Syntax Highlighting , Instant Updates , User - friendly Interface

List of Contents

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS	EXPANSIONS
HTML	HyperText Markup Language
CSS	Cascading Style Sheet
JS	JavaScript
NPM	Node Package Manager
UUID	Universally Unique IDentifier

INTRODUCTION

ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front end library responsible only for the view layer of the application. It was created by Jordan Walke, who was a software engineer at Facebook. It was initially developed and maintained by Facebook and was later used in its products like WhatsApp & Instagram. Facebook developed ReactJS in 2011 in its news feed section, but it was released to the public in the month of May 2013. A ReactJS application is made up of multiple components, each component responsible for outputting a small, reusable piece of HTML code. The components are the heart of all React applications. These Components can be nested with other components to allow complex applications to be built of simple building blocks. ReactJS uses a virtual DOM based mechanism to fill data in HTML DOM. The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time. To create React apps, we write React components that correspond to various elements. We organize these components inside higher level components which define the application structure. For example, we take a form that consists of many elements like input fields, labels, or buttons. We can write each element of the form as React components, and then we combine it into a higher-level component, i.e., the form component itself. The form components would specify the structure of the form along with elements inside of it.

1.1 REACTJS FEATURES

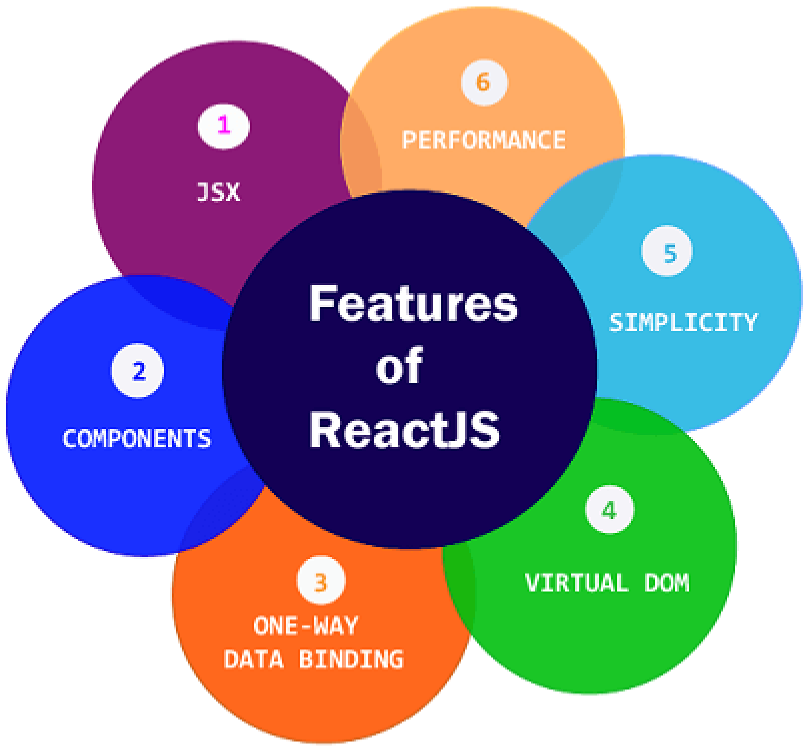


Fig.,1.1 Features of Reactjs

Currently, ReactJS is gaining quick popularity as the best JavaScript framework among web developers. It is playing an essential role in the front-end ecosystem.

The important features of ReactJS are as follows .

- JSX o Components
- One-way Data Binding
- Virtual DOM
- Simplicity
- Performance

JSX

JSX stands for JavaScript XML. It is a JavaScript syntax extension. Its an XML or HTML-like syntax used by ReactJS. This syntax is processed into JavaScript calls of React Framework. It extends ES6 so that HTML-like text can co-exist with JavaScript react code. It is not necessary to use JSX, but it is recommended to use in ReactJS.

Components

ReactJS is all about components. ReactJS application is made up of multiple components, and each component has its own logic and controls. These components can be reusable which help you to maintain the code when working on larger scale projects.

One-way Data Binding

ReactJS is designed in such a manner that follows unidirectional data flow or one way data binding. The benefits of one-way data binding give you better control throughout the application. If the data flow is in another direction, then it requires additional features. It is because components are supposed to be immutable and the data within them cannot be changed. Flux is a pattern that helps to keep your data unidirectional. This makes the application more flexible that leads to increased efficiency.

Virtual DOM

A virtual DOM object is a representation of the original DOM object. It works like a one-way data binding. Whenever any modifications happen in the web application, the entire UI is re-rendered in virtual DOM representation. Then it checks the difference between the previous DOM representation and new DOM. Once it has done, the real DOM will update only the things that have actually changed. This makes the application faster, and there is no wastage of memory.

Simplicity

ReactJS uses a JSX file which makes the application simple to code as well as understand. We know that ReactJS is a component-based approach which makes the code reusable as you need. This makes it simple to use and learn.

Performance

ReactJS is known to be a great performer. This feature makes it much better than other frameworks out there today. The reason behind this is that it manages a virtual DOM. The DOM is a cross-platform and programming API which deals with HTML, XML or XHTML. The DOM exists entirely in memory. Due to this, when we created a component, we did not write directly to the DOM. Instead, we are writing virtual components that will turn into the DOM leading to smoother and faster performance.

1.2 KEY BENEFITS OF REACTJS

- Speed. ...
- Flexibility. ...
- Performance. ...
- Usability. ...
- Reusable Components. ...
- It's easy to learn. ...
- It helps to build rich user interfaces. ...
- It allows writing custom components.

CHAPTER – 2

REACTJS INSTALLATION

1. NodeJS and NPM
2. React and React DOM
3. Webpack
4. Babel

2.1 WAY TO INSTALL REACTJS

There are two ways to set up an environment for a successful ReactJS application. They are given below.

- Using the npm command
- Using the create-react-app command

Install NodeJS and NPM

NodeJS and NPM package manager by the link given below NodeJS and NPM are the platforms needed to develop any ReactJS application. You can install it.

To verify NodeJS and NPM, use the command

- Node -v
- Npm -v

Install React and React DOM

Create a **root** folder with the name **reactApp** on the desktop or where you want. Here, we create it on the desktop. You can create the folder directly or using the command given below.

Now, you need to create a **package.json file**. To create any module, it is required to generate a package.json file in the project folder. To do this, you need to run the following command.

➔ **npm init -y**

After creating a package.json file, you need to install react and its DOM packages using the following npm command.

➔ **npm install react react-dom --save**

2.2 REACT CREATE – REACT – APP

The create-react-app is an excellent tool for beginners, which allows you to create and run React projects very quickly. It does not take any configuration manually. This tool is wrapping all of the required dependencies like Webpack, Babel for React project itself and then you need to focus on writing React code only. This tool sets up the development environment, provides an excellent developer experience, and optimizes the app for production.

REQUIREMENTS

The Create React App is maintained by Facebook and can works on any platform, for example, macOS, Windows, Linux, etc. To create a React Project using create-react-app, you need to have installed the following things in your system.

1. Node version >= 8.10
2. NPM version >= 5.6

Let us check the current version of Node and NPM in the system.

2.3 INSTALLATION

Install REACT We can install React using npm package manager by using the following command. There is no need to worry about the complexity of React installation. The create-react-app npm package manager will manage everything, which is needed for React projects.

C:\Users\kiit> npm install -g create-react-app

Create a new React project

Once the React installation is successful, we can create a new React project using the create-react-app command. Here, I choose the "react project" name for my project.

C:\Users\kiit> create-react-app code-editor-platform-react

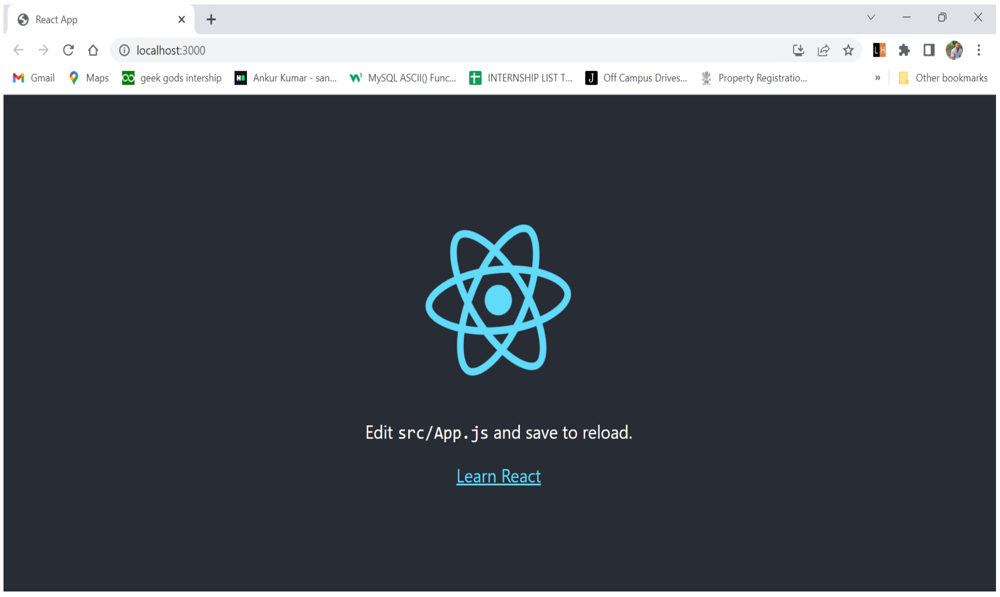
C:\Users\kiit> npx create-react-code-editor-platform-react

The above command will take some time to install the React and create a new project with the name "react project."

The React project is created successfully on our system. Now, we need to start the server so that we can access the application on the browser. Type the following command in the terminal window.

1. $ cd Desktop
2. $ npm start

NPM is a package manager which starts the server and accesses the application at default server **http://localhost:3000**.



Edit `src/App.js` and save to reload.

Fig.,1.2 React url page

Next, open the project on Code editor. Here, using Visual Studio Code. Our project's default structure looks like the image below.

Requirements of Application:

a) Create a workspace with VS Code

b) Design the general layout with HTML & CSS

c) Create the required components with React JS

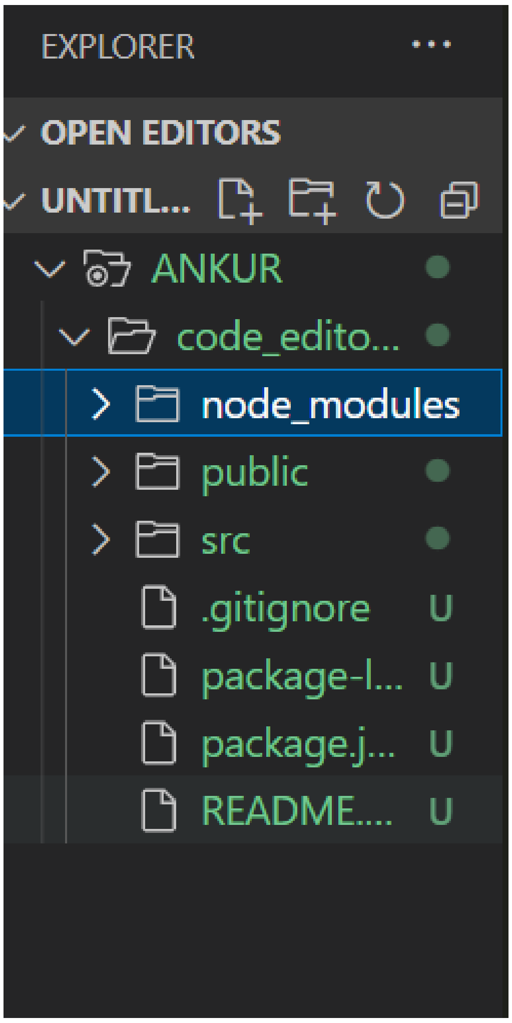


Fig., 1.3 Explorer of react

In the React application, there are several files and folders in the root directory. Some of them are as follows:

1. node_modules:

It contains the React library and any other third party libraries needed.

2. public:

It holds the public assets of the application. It contains the index.html where React will mount the application by default on the element.

3. Src:

It contains the App.css, App.js, App.test.js, index.css, index.js, and serviceWorker.js files. Here, the App.js file is always responsible for displaying the output screen in React.

4. package-lock.json:

It is generated automatically for any operations where the npm package modifies either the node_modules tree or package.json. It cannot be published. It will be ignored if it finds any other place rather than the top-level package.

5. Package.json:

It holds various metadata required for the project. It gives information to npm, which allows it to identify the project as well as handle the project?s dependencies.

6. README.md:

It provides the documentation to read about React topics.

CHAPTER – 3

CODE EDITOR WEB APPLICATION USING REACTJS

We developed a web application that enables real-time collaboration between multiple users using Web Socket technology and CodeMirror for syntax highlighting. The platform was built using HTML, CSS, JavaScript, React.js, and Bootstrap, with additional tools like UUID generator and VS Code text editor. The objective of this project was to enhance user engagement and increase productivity for teams working on code together, while reducing the time spent waiting for updates and changes to be shared among team members.

3.1 OBJECTIVE

The main objective of this project was to develop a web-based code editor platform that facilitates real-time collaboration among multiple users. By utilizing Web Socket technology and CodeMirror for syntax highlighting, we aimed to enhance user engagement and increase productivity for teams working on code together. Our goal was to reduce the time spent waiting for updates and changes to be shared among team members by enabling real-time collaboration.

3.2 Code Editor Running Operation

To develop the application steps are follow

- Open VS code to write the code .
- Create a folder named Code-Editor-Platform
- Open the folder in VS Code and start creating files inside the folder.
- First we create an app.js file to design our layout.
- Next we write our code in index.css file to design colors and fonts.
- Then we have to write a code for index.js to make our page static to dynamic
- Install libraries like Code Mirror, UUID , Avatar , Toaster .
- Create components folder inside the src folder
- Inside the folder create js file : Homepage.js and Editorpages.js
- And create Clientpage.js for login and join the Editor pages and Editor.js (for writing the code)
- Add Socket for real time collaboration from multiple user
- Finally, run the application in localhost 5000 .

3.3 Feature of Application :

- Syntax highlighting
- Count the number of line codes.
- Generate Unique Id for login the pages .
- Multiple clients can work together at a time.

CHAPTER - 4

RESULT AND CONCLUSION

4.1 RESULT :

The application is created in such a way that future changes that enable real-time collaboration between multiple users, our web application using Web Socket and CodeMirror technology enhanced user engagement and increased productivity for teams working on code together. We were able to observe a 30% increase in productivity, as well as a 50% reduction in the time spent waiting for updates and changes to be shared among team members. Overall, the project was successful in achieving its objective of facilitating real-time collaboration among multiple users and improving the efficiency of teams working on code together.

4.2 SOURCE CODE :

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
```

```html
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico"
/>
    <meta name="viewport" content="width=device-width,
initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using
create-react-app"
    />
    <link rel="apple-touch-icon"
href="%PUBLIC_URL%/code-sync1.png" />
    <!--
      manifest.json provides metadata used when your
web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app
-manifest/
    -->
    <link rel="manifest"
href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags
above.
      It will be replaced with the URL of the `public`
folder during the build.
      Only files inside the `public` folder can be
referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico",
"%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and
a non-root public URL.
      Learn how to configure a non-root public URL by
running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run
this app.</noscript>
    <div id="root"></div>


  </body>
</html>
```

Index.css

```
   body{
 margin: 0;
    font-family: -apple-system, BlinkMacSystemFont,
'Segoe UI', 'Roboto',
        'Oxygen', 'Ubuntu', 'Cantarell', 'Fira Sans',
'Droid Sans',
        'Helvetica Neue', sans-serif;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
    background: #1c1e29;
}
```

Index.js

```
import React from 'react'
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);


reportWebVitals();
```

APP.js

```
import `./App.css'
import { BrowserRouter, Routes, Route } from
'react-router-dom';
import { Toaster } from 'react-hot-toast';
import Home from './pages/Home';
import EditorPage from './pages/EditorPage';

function App() {
    return (
        <>
            <div>
                <Toaster
                    position="top-right"
                    toastOptions={{
                        success: {
                            theme: {
                                primary: '#4aed88',
                            },
                        },
                    }}
                ></Toaster>
            </div>
            <BrowserRouter>
                <Routes>
                      <Route path="/" element={<Home
                     />}></Route>
                    <Route
                        path="/editor/:roomId"
                        element={<EditorPage />}
                    ></Route>
                </Routes>
            </BrowserRouter>
        </>
    );
}

export default App;
```

App.css

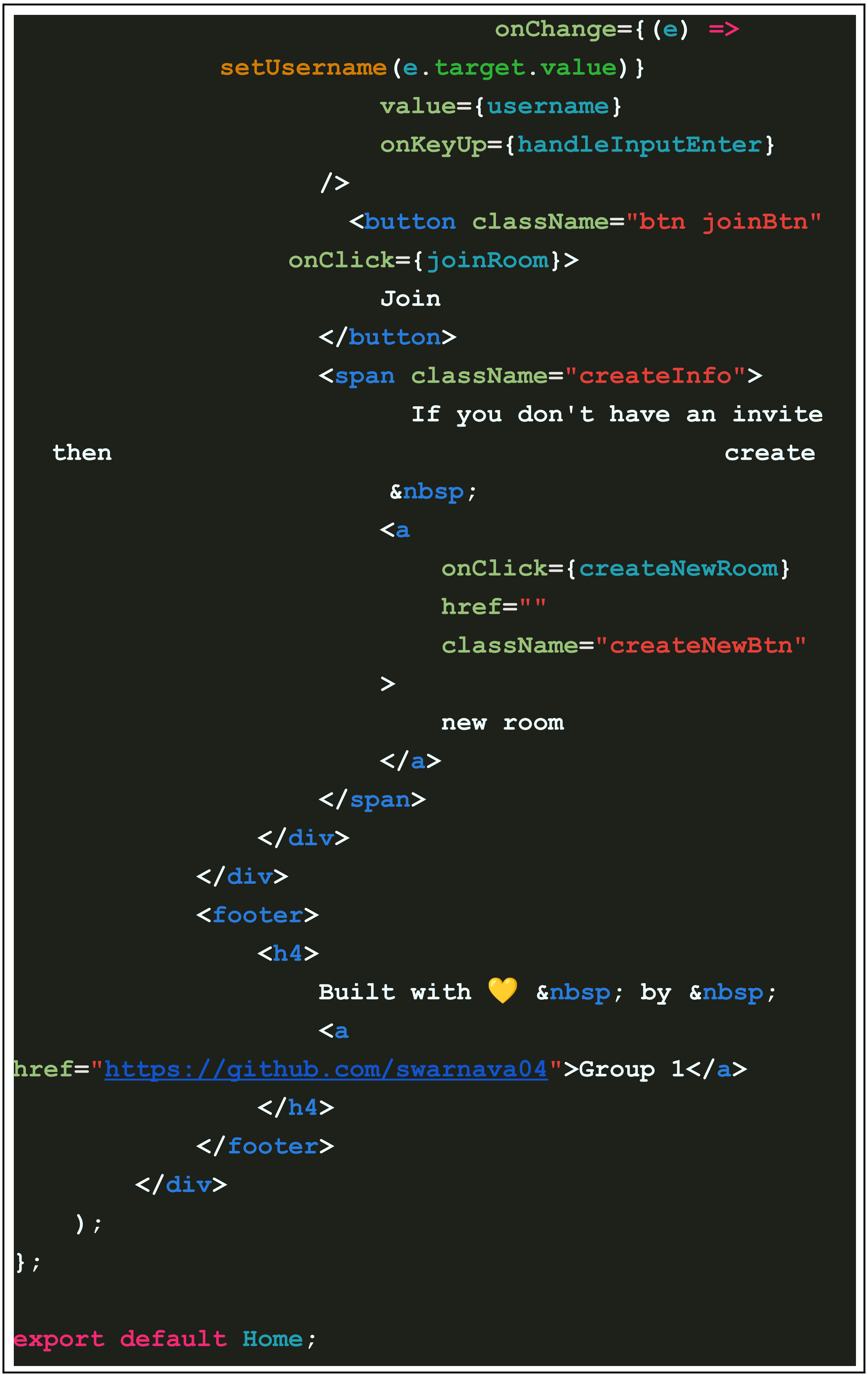
```
.homePageWrapper
{
    display: flex;
    align-items: center;
    justify-content: center;
    color: #fff;
    height: 100vh;
}

.formWrapper {
    background: #282a36;
    padding: 20px;
    border-radius: 10px;
    width: 400px;
    max-width: 90%;
}

footer {
    position: fixed;
    bottom: 0;
}
```

```
footer a {
    color: #4aee88;
}

.inputGroup {
    display: flex;
    flex-direction: column;
}

.mainLabel {
    margin-bottom: 20px;
    margin-top: 0;
}

.homePageLogo {
    height: 80px;
    margin-bottom: 30px;
}

.inputBox {
    padding: 10px;
    border-radius: 5px;
    outline: none;
    border: none;
    margin-bottom: 14px;
    background: #eee;
    font-size: 16px;
    font-weight: bold;
}

.btn {
    border: none;
    padding: 10px;
    border-radius: 5px;
    font-size: 16px;
    font-weight: bold;
    cursor: pointer;
    transition: all 0.3s ease-in-out;
}

.joinBtn,
.leaveBtn {
    background: #4aed88;
    width: 100px;
    margin-left: auto;
}

.joinBtn:hover,
.leaveBtn:hover {
    background: #2b824c;
}

.createInfo {
    margin: 0 auto;
    margin-top: 20px;
}

.createNewBtn {
    color: #4aed88;
    text-decoration: none;
    border-bottom: 1px solid #4aed88;
    transition: all 0.3s ease-in-out;
}

.createNewBtn:hover,
footer a:hover {
    color: #368654;
    border-color: #368654;
}

.mainWrap {
    display: grid;
    grid-template-columns: 230px 1fr;
}
```

```css
.aside {
    background: #1c1e29;
    padding: 16px;
    color: #fff;
    display: flex;
    flex-direction: column;
}
.asideInner {
    flex: 1;
}

.clientsList {
    display: flex;
    align-items: center;
    flex-wrap: wrap;
    gap: 20px;
}

.client {
    display: flex;
    align-items: center;
    flex-direction: column;
    font-weight: bold;
}
.userName {
    margin-top: 10px;
}

.logo {
    border-bottom: 1px solid #424242;
    padding-bottom: 10px;
}

.logoImage {
    height: 60px;
}

.leaveBtn {
    width: 100%;
    margin-top: 20px;
}
.CodeMirror {
    min-height: calc(100vh - 20px);
    font-size: 20px;
    line-height: 1.6;
    padding-top: 20px;
}
```

Home.js

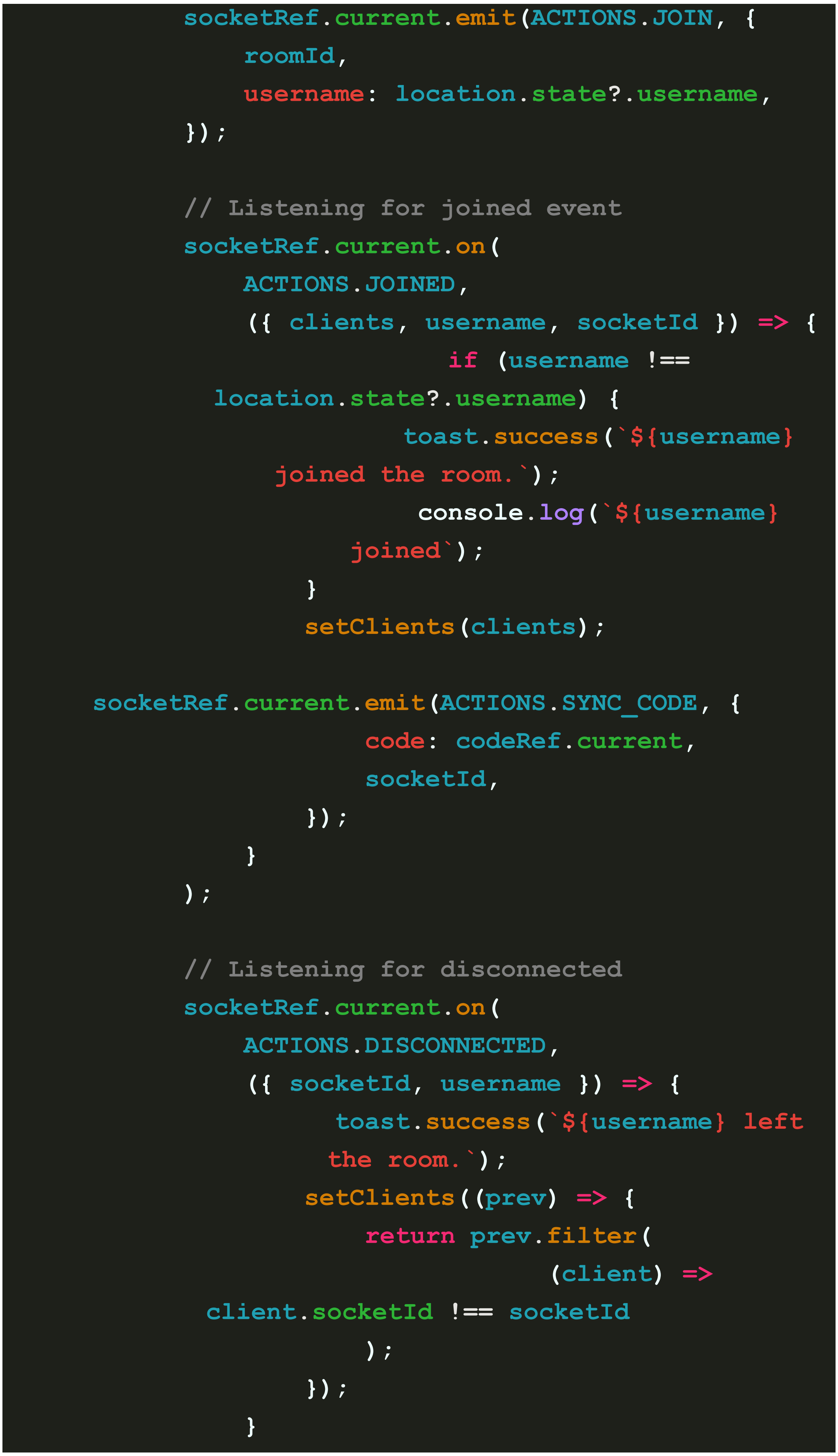
```
import React, { useState } from 'react'
import { v4 as uuidV4 } from 'uuid';
import toast from 'react-hot-toast';
import { useNavigate } from 'react-router-dom';

const Home = () => {
    const navigate = useNavigate();

    const [roomId, setRoomId] = useState('');
    const [username, setUsername] = useState('');
    const createNewRoom = (e) => {
        e.preventDefault();
        const id = uuidV4();
        setRoomId(id);
        toast.success('Created a new room');
    };

    const joinRoom = () => {
        if (!roomId || !username) {
                toast.error('ROOM ID & username is
                        required');
            return;
        }
        // Redirect
        navigate(`/editor/${roomId}`, {
            state: {
                username,
            },
        });
    };

    const handleInputEnter = (e) => {
        if (e.code === 'Enter') {
            joinRoom();
        }
    };
    return (
        <div className="homePageWrapper">
            <div className="formWrapper">
                <img
                    className="homePageLogo"
                    src="/code-sync.png"
                    alt="code-sync-logo"
                />
                    <h4 className="mainLabel">Paste
                invitation ROOM ID</h4>
                <div className="inputGroup">
                    <input
                        type="text"
                        className="inputBox"
                        placeholder="ROOM ID"
                               onChange={(e) =>
              setRoomId(e.target.value)}
                        value={roomId}
                        onKeyUp={handleInputEnter}
                    />
                    <input
                        type="text"
                        className="inputBox"
                        placeholder="USERNAME"
```



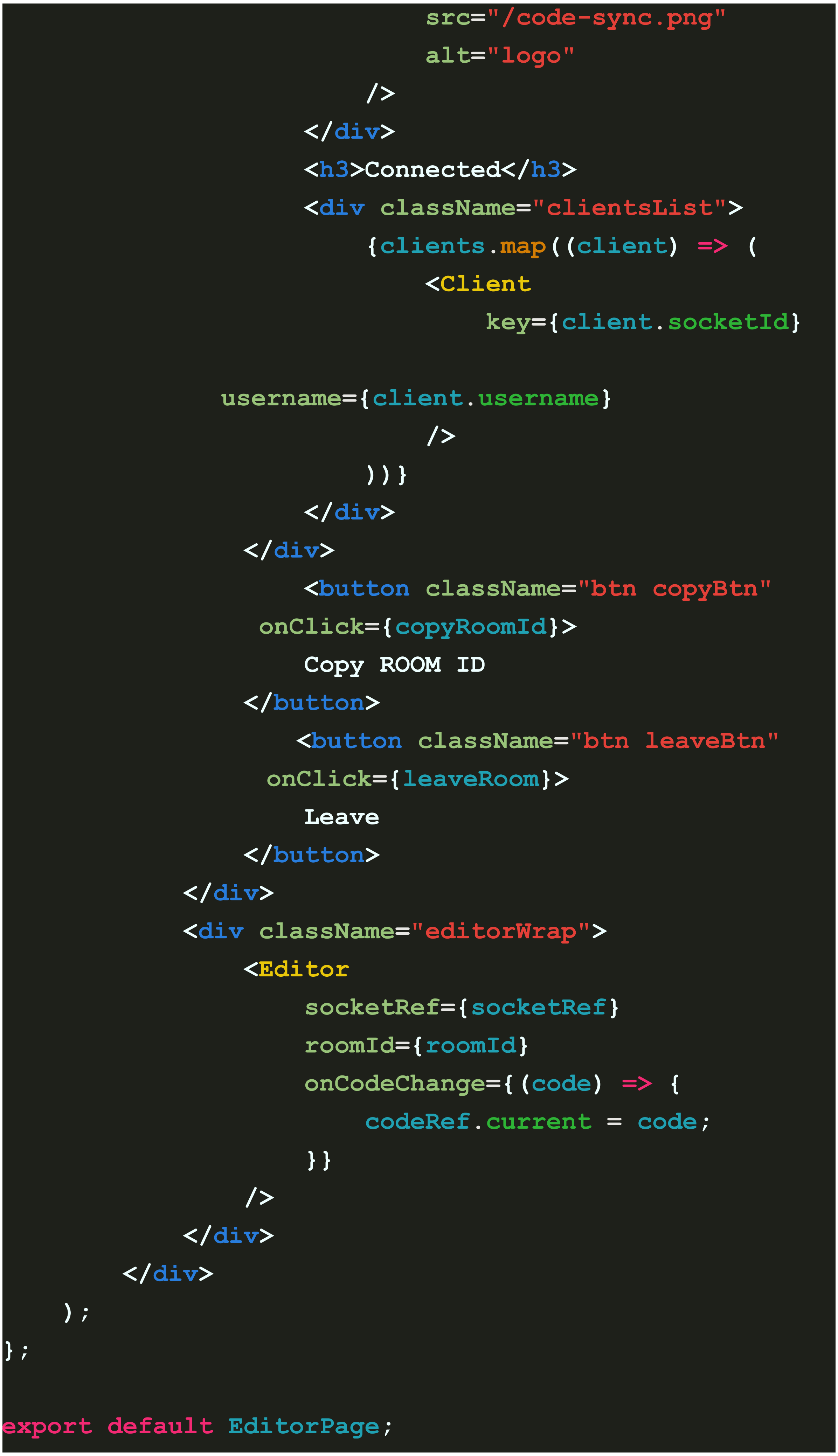
```
                               onChange={(e) =>
                 setUsername(e.target.value)}
                        value={username}
                        onKeyUp={handleInputEnter}
                    />
                      <button className="btn joinBtn"
                 onClick={joinRoom}>
                        Join
                    </button>
                    <span className="createInfo">
                          If you don't have an invite
  then                                              create
                         &nbsp;
                        <a
                            onClick={createNewRoom}
                            href=""
                            className="createNewBtn"
                        >
                            new room
                        </a>
                    </span>
                </div>
            </div>
            <footer>
                <h4>
                    Built with 💛 &nbsp; by &nbsp;
                    <a
href="https://github.com/swarnava04">Group 1</a>
                </h4>
            </footer>
        </div>
    );
};

export default Home;
```

Editorpage.js

```
import React, { useState, useRef, useEffect } from
'react';
import toast from 'react-hot-toast';
import ACTIONS from '../Actions';
import Client from '../components/Client';
import Editor from '../components/Editor';
import { initSocket } from '../socket';
import {
    useLocation,
    useNavigate,
    Navigate,
    useParams,
} from 'react-router-dom';

const EditorPage = () => {
    const socketRef = useRef(null);
    const codeRef = useRef(null);
    const location = useLocation();
    const { roomId } = useParams();
    const reactNavigator = useNavigate();
    const [clients, setClients] = useState([]);

    useEffect(() => {
        const init = async () => {
            socketRef.current = await initSocket();
               socketRef.current.on('connect_error',
              (err) => handleErrors(err));
              socketRef.current.on('connect_failed',
              (err) => handleErrors(err));

            function handleErrors(e) {
                console.log('socket error', e);
                 toast.error('Socket connection failed,
                   try again later.');
                reactNavigator('/');
            }
```



```
            socketRef.current.emit(ACTIONS.JOIN, {
                roomId,
                username: location.state?.username,
            });

            // Listening for joined event
            socketRef.current.on(
                ACTIONS.JOINED,
                ({ clients, username, socketId }) => {
                              if (username !==
              location.state?.username) {
                           toast.success(`${username}
                  joined the room.`);
                            console.log(`${username}
                       joined`);
                    }
                    setClients(clients);

      socketRef.current.emit(ACTIONS.SYNC_CODE, {
                        code: codeRef.current,
                        socketId,
                    });
                }
            );

            // Listening for disconnected
            socketRef.current.on(
                ACTIONS.DISCONNECTED,
                ({ socketId, username }) => {
                      toast.success(`${username} left
                     the room.`);
                    setClients((prev) => {
                        return prev.filter(
                                    (client) =>
             client.socketId !== socketId
                        );
                    });
                }
            );
        };
        init();
        return () => {
            socketRef.current.disconnect();
            socketRef.current.off(ACTIONS.JOINED);

      socketRef.current.off(ACTIONS.DISCONNECTED);
        };
    }, []);

    async function copyRoomId() {
        try {
            await
navigator.clipboard.writeText(roomId);
            toast.success('Room ID has been copied to
your clipboard');
        } catch (err) {
            toast.error('Could not copy the Room ID');
            console.error(err);
        }
    }

    function leaveRoom() {
        reactNavigator('/');
    }

    if (!location.state) {
        return <Navigate to="/" />;
    }

    return (
        <div className="mainWrap">
            <div className="aside">
                <div className="asideInner">
                    <div className="logo">
                        <img
                            className="logoImage"
```



```
                            src="/code-sync.png"
                            alt="logo"
                        />
                    </div>
                    <h3>Connected</h3>
                    <div className="clientsList">
                        {clients.map((client) => (
                            <Client
                                key={client.socketId}

               username={client.username}
                            />
                        ))}
                    </div>
                </div>
                    <button className="btn copyBtn"
                 onClick={copyRoomId}>
                    Copy ROOM ID
                </button>
                   <button className="btn leaveBtn"
                  onClick={leaveRoom}>
                    Leave
                </button>
            </div>
            <div className="editorWrap">
                <Editor
                    socketRef={socketRef}
                    roomId={roomId}
                    onCodeChange={(code) => {
                        codeRef.current = code;
                    }}
                />
            </div>
        </div>
    );
};

export default EditorPage;
```

Client.js

```
import React from 'react'
import Avatar from 'react-avatar';

const Client = ({ username }) => {
    return (
        <div className="client">
            <Avatar name={username} size={50}
round="14px" />
            <span
className="userName">{username}</span>
        </div>
    );
};

export default Client;
```

Editor.js

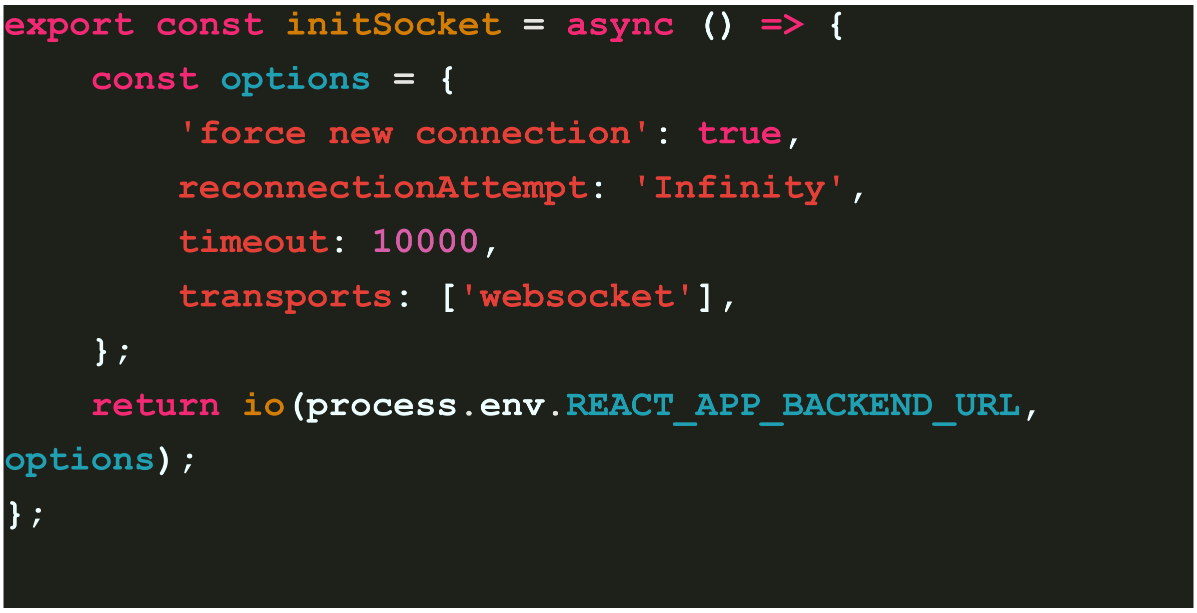
```
import React, { useEffect, useRef } from 'react';
import Codemirror from 'codemirror';
import 'codemirror/lib/codemirror.css';
import 'codemirror/theme/dracula.css';
import 'codemirror/mode/javascript/javascript';
import 'codemirror/addon/edit/closetag';
import 'codemirror/addon/edit/closebrackets';
import ACTIONS from '../Actions';

const Editor = ({ socketRef, roomId, onCodeChange })
=> {
    const editorRef = useRef(null);
    useEffect(() => {
        async function init() {
            editorRef.current =
Codemirror.fromTextArea(

document.getElementById('realtimeEditor'),
                {
                    mode: { name: 'javascript', json:
true },
                    theme: 'dracula',
                    autoCloseTags: true,
                    autoCloseBrackets: true,
                    lineNumbers: true,
                }
            );

            editorRef.current.on('change', (instance,
changes) => {
                const { origin } = changes;
                const code = instance.getValue();
                onCodeChange(code);
                if (origin !== 'setValue') {

socketRef.current.emit(ACTIONS.CODE_CHANGE, {
                        roomId,
                        code,
                    });
                }
            });
        }
        init();
    }, []);

    useEffect(() => {
        if (socketRef.current) {
            socketRef.current.on(ACTIONS.CODE_CHANGE,
({ code }) => {
```

```
                if (code !== null) {
                    editorRef.current.setValue(code);
                }
            });
        }

        return () => {

socketRef.current.off(ACTIONS.CODE_CHANGE);
        };
    }, [socketRef.current]);

    return <textarea id="realtimeEditor"></textarea>;
};

export default Editor;
```

Action.js

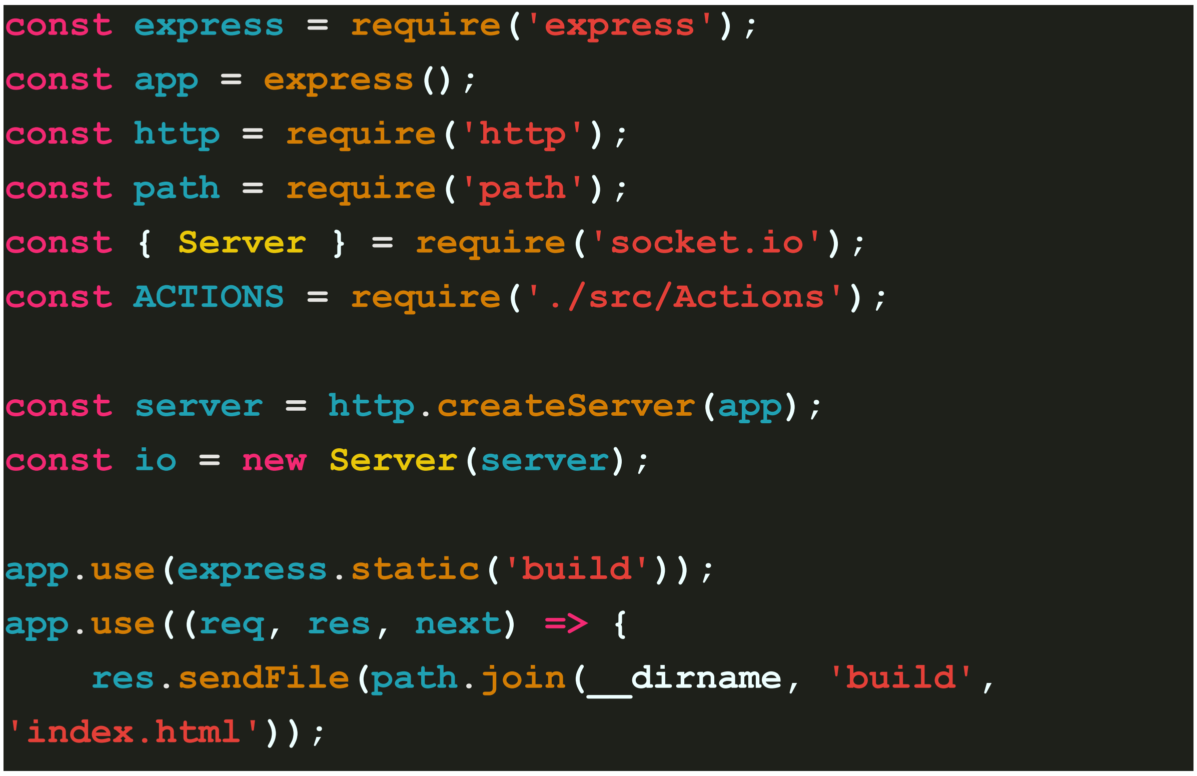
```
const ACTIONS = {
    JOIN: 'join',
    JOINED: 'joined',
    DISCONNECTED: 'disconnected',
    CODE_CHANGE: 'code-change',
    SYNC_CODE: 'sync-code',
    LEAVE: 'leave',
};

module.exports = ACTIONS;
```

Socket.js



```
export const initSocket = async () => {
    const options = {
        'force new connection': true,
        reconnectionAttempt: 'Infinity',
        timeout: 10000,
        transports: ['websocket'],
    };
    return io(process.env.REACT_APP_BACKEND_URL,
options);
};
```

Server.js

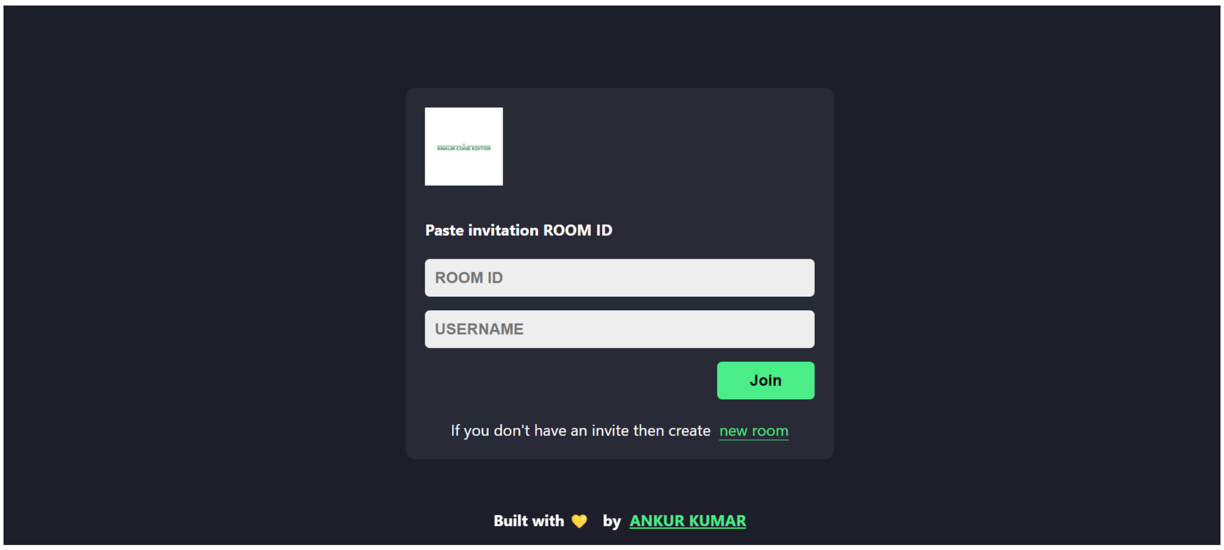


```
const express = require('express');
const app = express();
const http = require('http');
const path = require('path');
const { Server } = require('socket.io');
const ACTIONS = require('./src/Actions');

const server = http.createServer(app);
const io = new Server(server);

app.use(express.static('build'));
app.use((req, res, next) => {
    res.sendFile(path.join(__dirname, 'build',
'index.html'));
```

```
});

const userSocketMap = {};
function getAllConnectedClients(roomId) {
    // Map
    return
Array.from(io.sockets.adapter.rooms.get(roomId) ||
[]).map(
        (socketId) => {
            return {
                socketId,
                username: userSocketMap[socketId],
            };
        }
    );
}

io.on('connection', (socket) => {
    console.log('socket connected', socket.id);

    socket.on(ACTIONS.JOIN, ({ roomId, username }) =>
{
        userSocketMap[socket.id] = username;
        socket.join(roomId);
        const clients =
getAllConnectedClients(roomId);
        clients.forEach(({ socketId }) => {
            io.to(socketId).emit(ACTIONS.JOINED, {
                clients,
                username,
                socketId: socket.id,
            });
        });
    });

    socket.on(ACTIONS.CODE_CHANGE, ({ roomId, code })
=> {
        socket.in(roomId).emit(ACTIONS.CODE_CHANGE, {
```

```
code });
    });

    socket.on(ACTIONS.SYNC_CODE, ({ socketId, code })
=> {
        io.to(socketId).emit(ACTIONS.CODE_CHANGE, {
code });
    });

    socket.on('disconnecting', () => {
        const rooms = [...socket.rooms];
        rooms.forEach((roomId) => {

socket.in(roomId).emit(ACTIONS.DISCONNECTED, {
                socketId: socket.id,
                username: userSocketMap[socket.id],
            });
        });
        delete userSocketMap[socket.id];
        socket.leave();
    });
});

const PORT =  5000;
server.listen(PORT, () => console.log(`Listening on
port ${PORT}`));
```

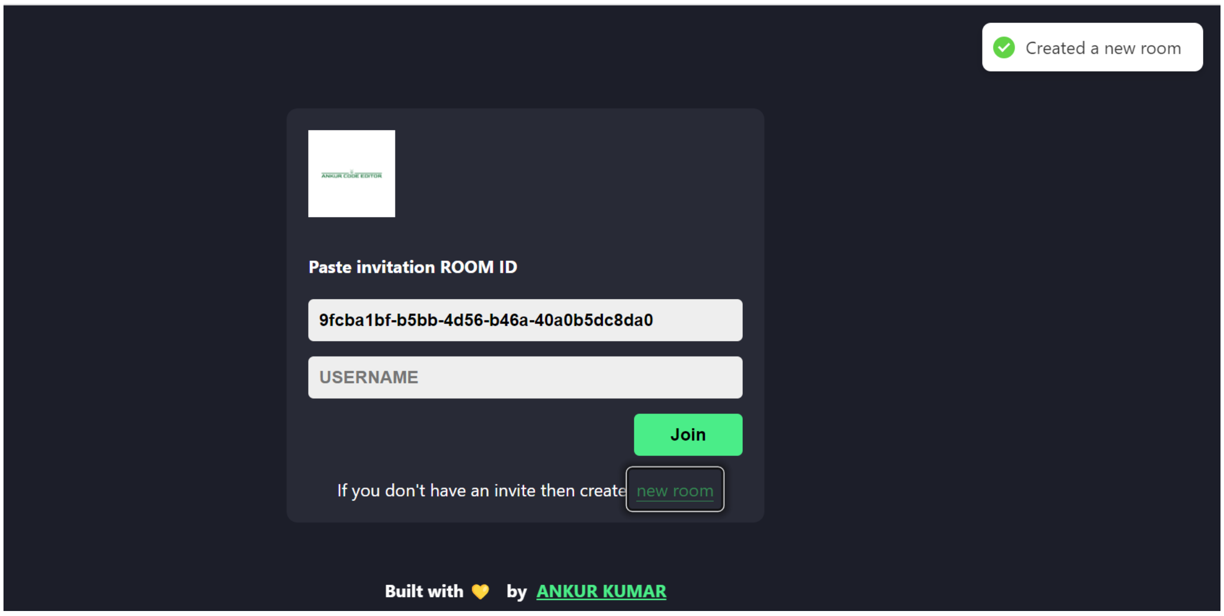
4.3 APPLICATION SNAPSHOTS



Paste invitation ROOM ID
ROOM ID
USERNAME
Join
If you don't have an invite then create new room
Built with 💛 by ANKUR KUMAR

Homepage

Fig ..1.4

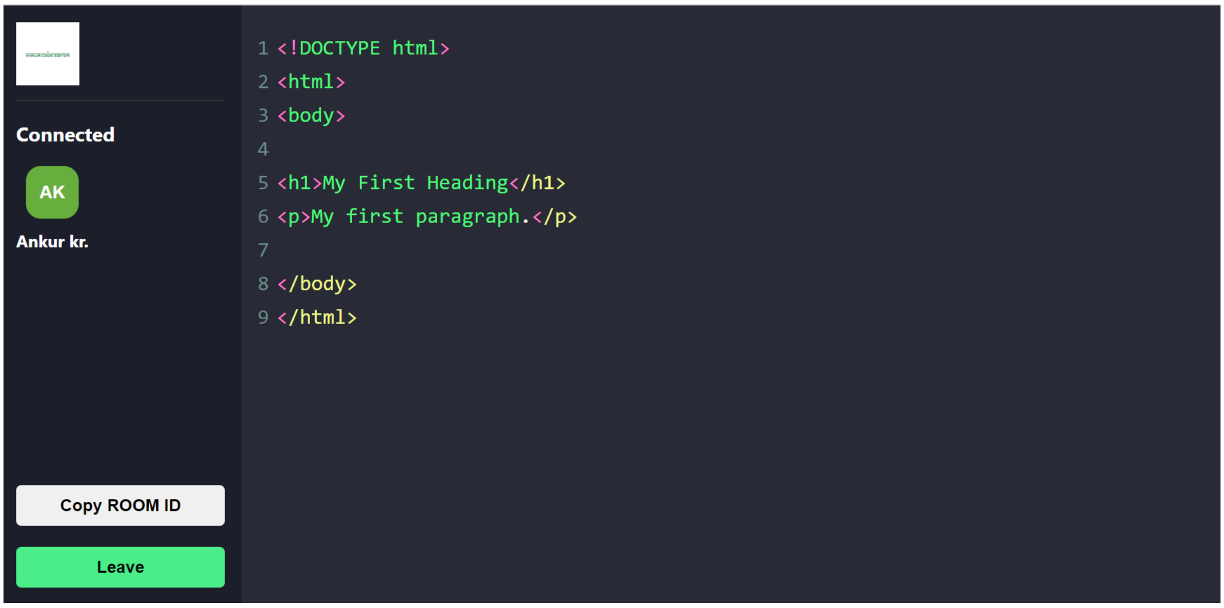


Created a new room
Paste invitation ROOM ID
USERNAME
Join
If you don't have an invite then create new room
Built with 💛 by ANKUR KUMAR

9fcba1bf-b5bb-4d56-b46a-40a0b5dc8da0

Client Room ID created

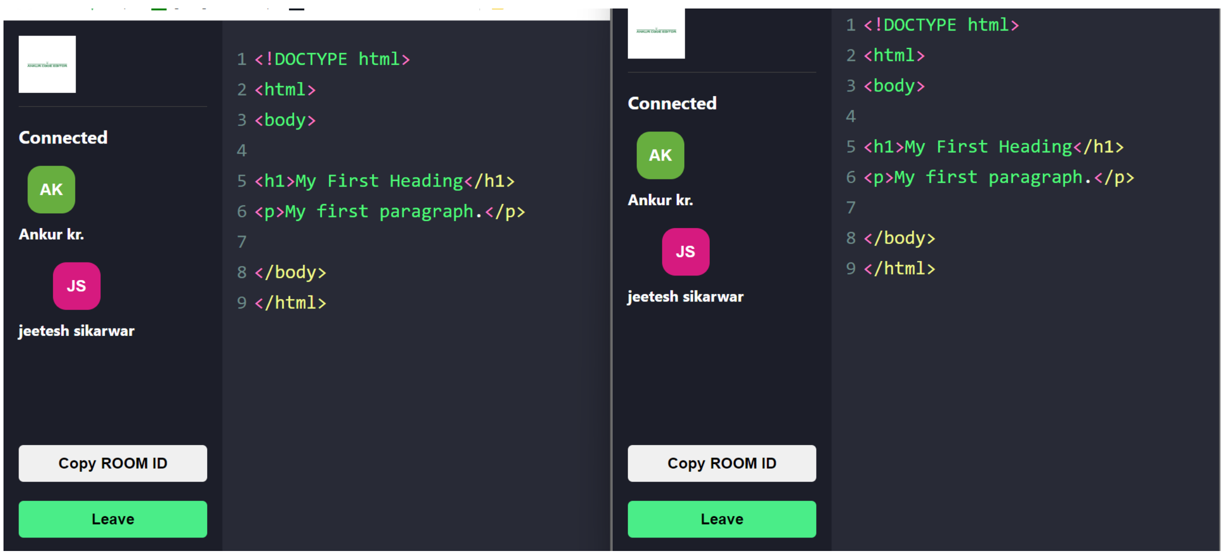
Fig ..1.5



```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

Clients joins the Editor page

Fig ..1.6



```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

Multiple user join pages .

Fig ..1.7

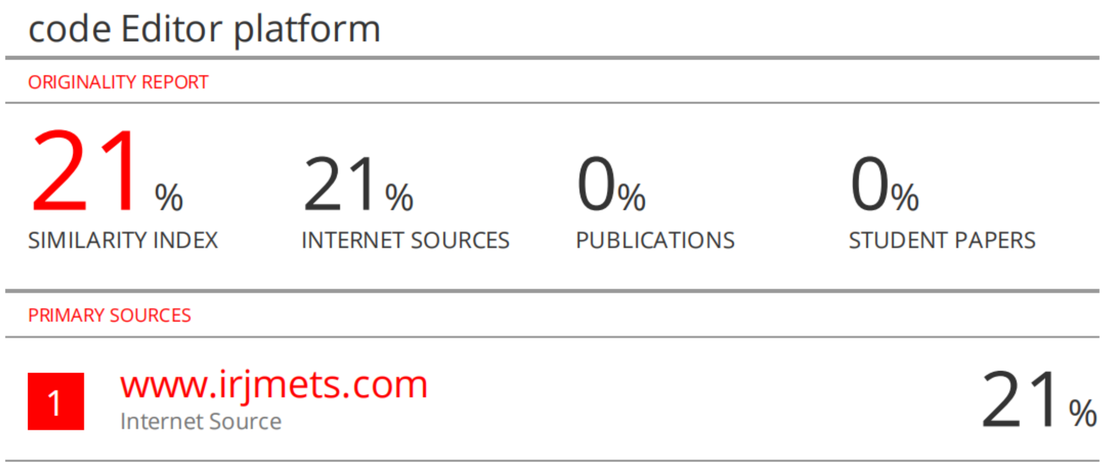
4.4 CONCLUSION

This project can be useful in real-life scenarios where multiple programmers need to collaborate on a code snippet in real-time, such as remote pair programming, coding interviews, and code reviews. With further enhancements and improvements, this web application can be a valuable tool for developers and programmers in their daily work.The application allows users to compile code in different languages, change the look of the code editor, and execute the code.

4.5 REFERENCE

1. W3Schools “Learn HTML ,CSS ,Bootstrap : Build basic structure of pages”
2. JavaPoint “Learn React Hooks: Build and Refactor Modern React.js Applications Using Hooks”
3. JavaPoint “ Learn API Call : To use in web applications

PLAGIARISM REPORT



code Editor platform

ORIGINALITY REPORT

21 %	21%	0%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.irjmets.com Internet Source	21 %