LOVELY PROFESSIONAL UNIVERSITY

Transforming Education Transforming India

ADVANCED WEB DEVELOPMENT

COURSE CODE : INT222

CONTINUOUS ASSESSMENT - 2

SUBMITTED BY :-

Name – Bhavesh Singh

Roll no – 19

Reg. No. – 11703793

Section – KM014

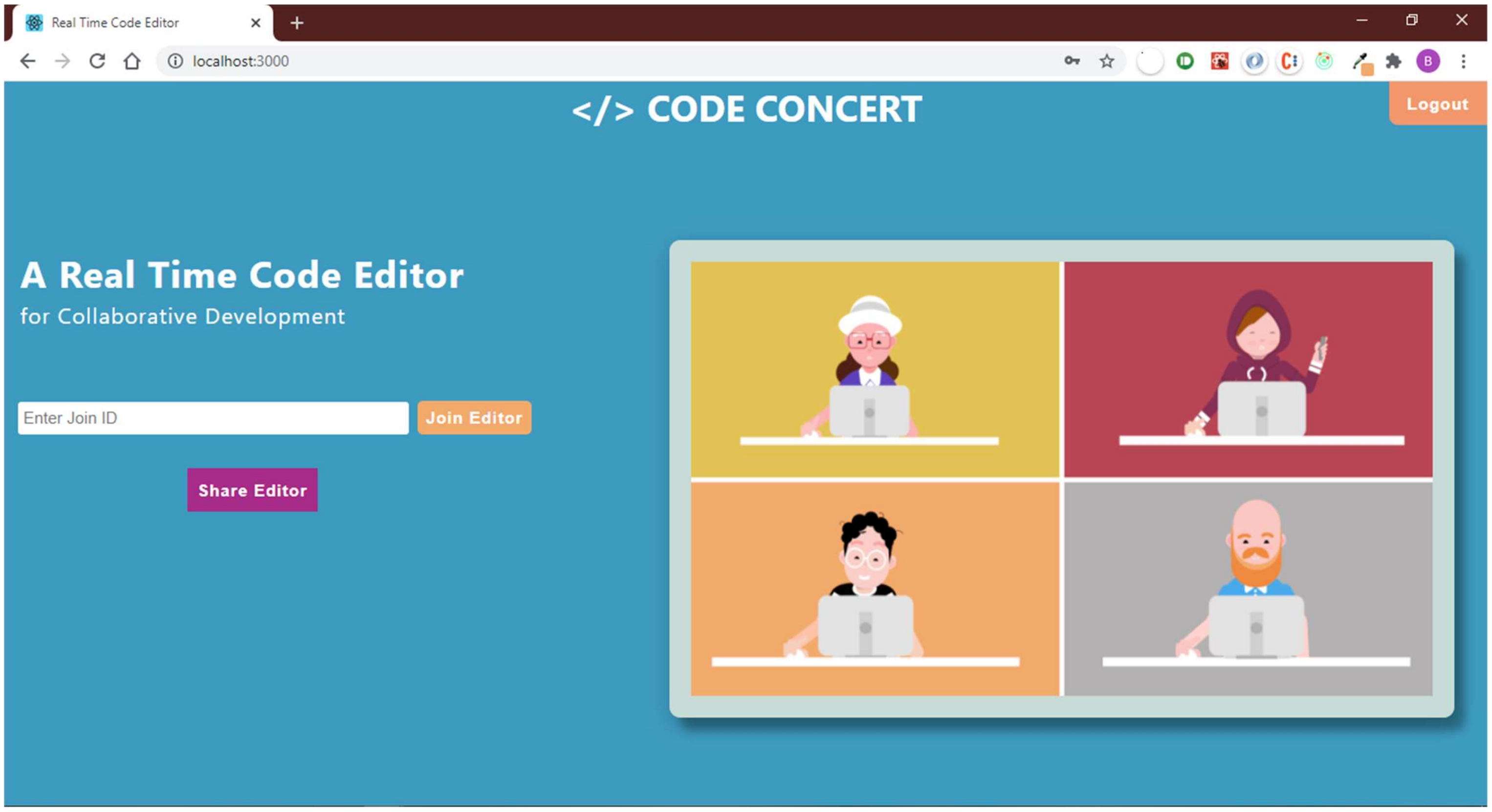
INDEX

INTRODUCTION

This project is a real time code editor which uses web sockets for connecting users to a particular room where they can use code editor collaboratively.

GitHub Link :- https://github.com/Bhaveshsingh022000/RealtimeCodeSharing

Home Page



This is the is the home page of the application. User can get to this home page after logging in into the website. Home page consist of an Input Field where user can enter Room Id to which the user wants to connect and can get connected to the room.

It also has a “Share Editor” Button that will generate a random Room Id which a user can share with peers for joining the room.

Code for Server Side Socket Connection :-

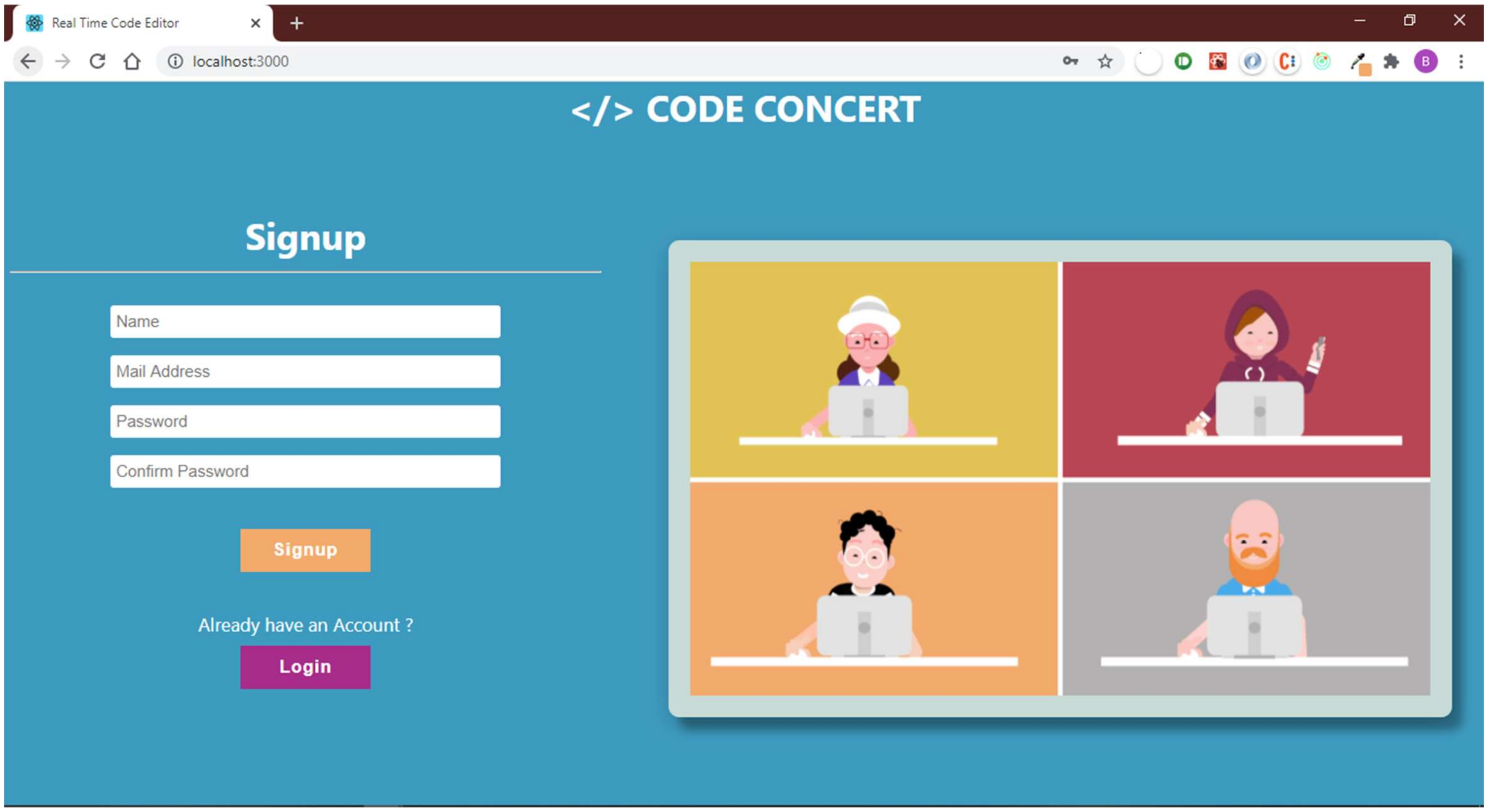
```
const server = app.listen(3005);
        const io = require('./socket').init(server);
        io.on("connection", socket => {
            console.log("Client Connected");
            socket.on("disconnect", () => console.log(`Disconnected ${socket.id}`));
            socket.on('join', (room) => {
                console.log(`Socket ${socket.id} joined ${room}`);
                socket.join(room);
            });
            socket.on('content', data => {
                const { data, room } = data;
                // console.log(message);
                io.to(room).emit('content', data);
            })
        })
```

The above code joins the user to a particular Room. An event “content ” is registered which is emitted whenever a user types some thing on the Editor “io.(room).emit(‘content’, data)” is also emitted which will send the data to all the users who are connected to a particular room

Code for Client-Side Socket Connection

```
import io from 'socket.io-client';
const socket = io('http://localhost:3005/');

componentDidMount() {
        socket.emit('join', this.props.roomName);
        socket.on('content', msg => {
            this.setState({ code: msg });
        })
    }

onChange = (newValue, e) => {
        this.setState({ code: e });
        socket.emit('content', { message: this.state.code, room: this.props.roomName });
    }
```

Signup Page



Code for Signup Route

```
const { body } = require('express-validator/check');
const authController = require('../controller/authController');
const User = require('../model/userModel');

router.post('/signup', [
    body('email').isEmail()
        .withMessage("Please Enter valid Email")
        .custom((value, { req }) => {
            return User.findOne({ email: value }).then(userData => {
                if (userData) {
                    return Promise.reject("Email Already Exist");
                }
            });
        }).normalizeEmail(),
    body('password').trim().isLength({ min: 6 }),
    body('name').trim().not().isEmpty()
], authController.postSignUp);
```

In the above code user's credentials is validated using Express Validator and if there is any error it is passed to the signup controller.

Signup Controller

```
const UserModel = require('../model/userModel');
const bcrypt = require('bcryptjs');
const {validationResult} = require('express-validator/check');
exports.postSignUp = (req,res,next)=>{
    const errors = validationResult(req);
    if(!errors.isEmpty()){
        const error = new Error("Validation Failed");
        error.statusCode = 422;
        error.data = errors.array()[0].msg;
        throw error
    }
    const email = req.body.email;
    const password = req.body.password;
    const name = req.body.name;
    bcrypt.hash(password, 12)
        .then(hashedPwd =>{
            const user = new UserModel({
                email: email,
                password: hashedPwd,
                name: name
            });
            return user.save();
        })
        .then(result =>{
            res.status(201)
                .json({
                    message: 'User Created',
                    userId: result._id
                })
        })
        .catch(err =>{
            if (!err.statusCode) {
                err.statusCode = 500;
            }
            next(err);
        })
};
```

In signup controller “validationResut(req)” is called which will return array and if the array is empty then no error are generated during validation, and if array is not empty then there is validation error this generated error is thrown with a status code of 422 and error message is stored in “errors.data”

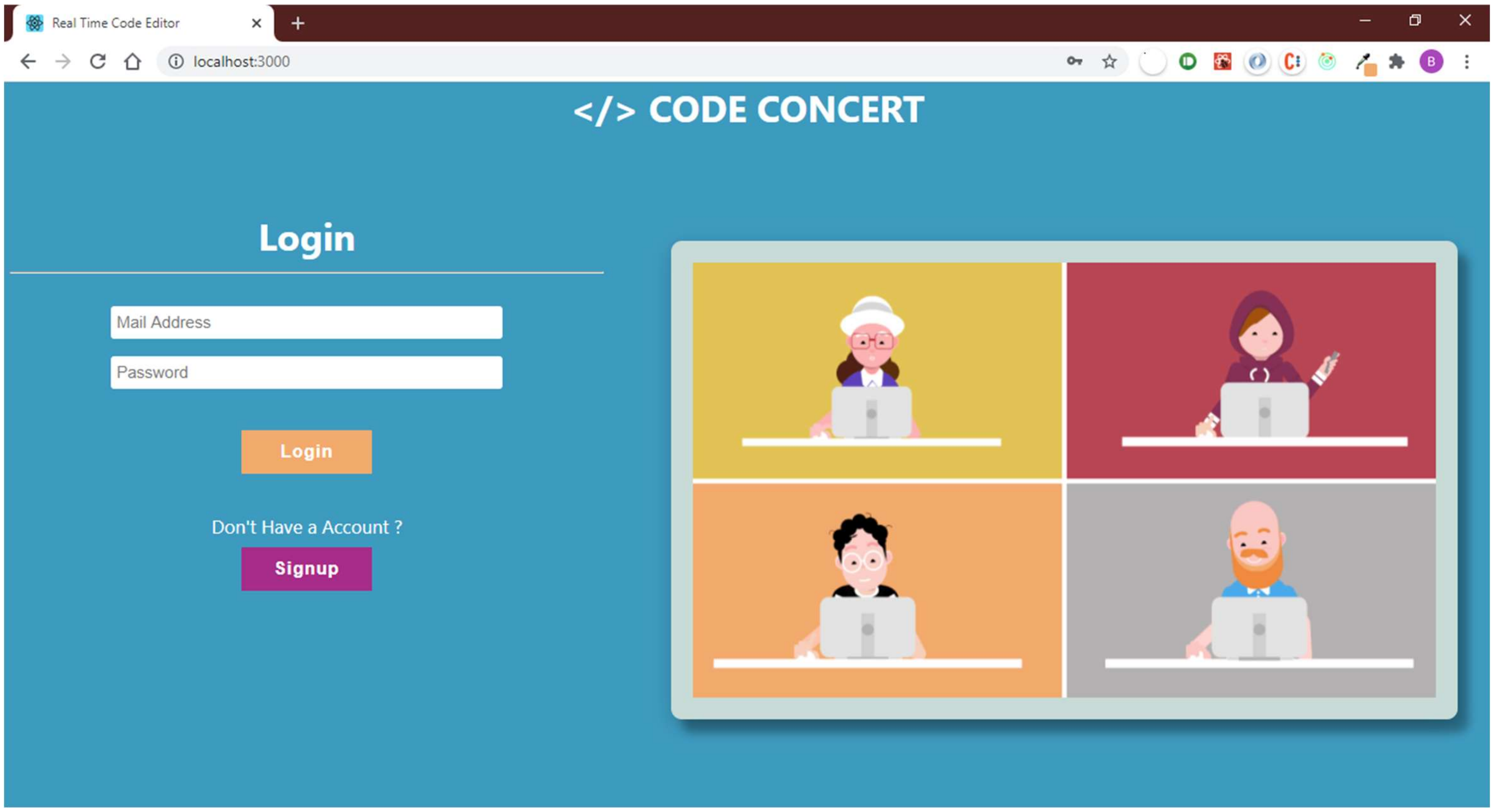
If validation is successful then password is hashed using “bcrypt” library and then it is stored in MongoDB Database. A response with status code 201 with message “User Created” is sent to the frontend.

User Model

```
const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const userSchema = new Schema({
    name: {
        type: String,
        minlength: 3,
        required: true
    },
    email: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    }
});

module.exports = mongoose.model('Users', userSchema);
```

The above code defines the schema for Users collection in database.

Login Page

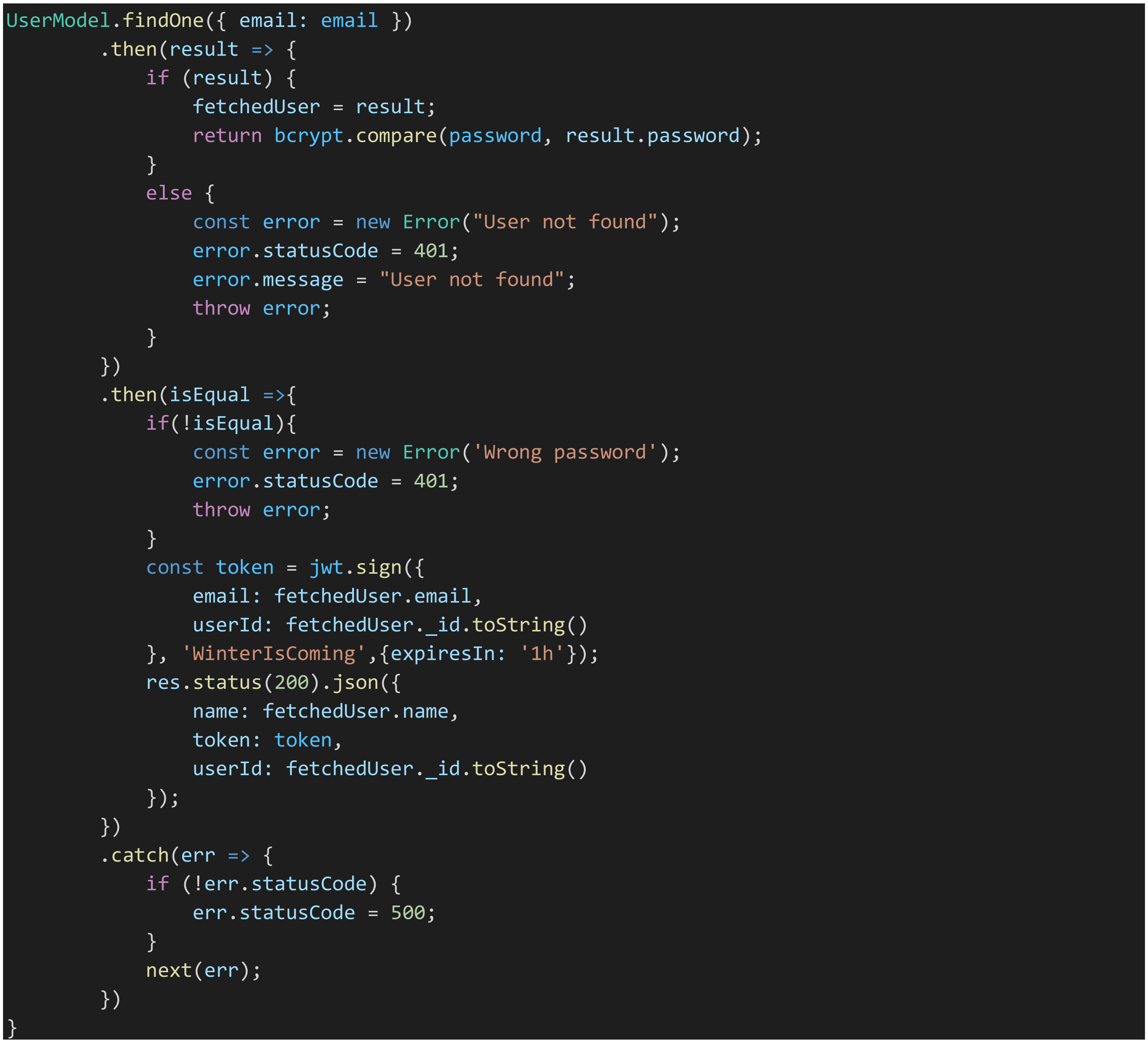


Login Route

```
const router = express.Router();

router.post('/login', authController.postLogin);
```

Login Controller

```
const jwt = require('jsonwebtoken');

exports.postLogin = (req, res, next) => {
    const email = req.body.email;
    const password = req.body.password;
    let fetchedUser;
```



```
UserModel.findOne({ email: email })
        .then(result => {
            if (result) {
                fetchedUser = result;
                return bcrypt.compare(password, result.password);
            }
            else {
                const error = new Error("User not found");
                error.statusCode = 401;
                error.message = "User not found";
                throw error;
            }
        })
        .then(isEqual =>{
            if(!isEqual){
                const error = new Error('Wrong password');
                error.statusCode = 401;
                throw error;
            }
            const token = jwt.sign({
                email: fetchedUser.email,
                userId: fetchedUser._id.toString()
            }, 'WinterIsComing',{expiresIn: '1h'});
            res.status(200).json({
                name: fetchedUser.name,
                token: token,
                userId: fetchedUser._id.toString()
            });
        })
        .catch(err => {
            if (!err.statusCode) {
                err.statusCode = 500;
            }
            next(err);
        })
}
```

In login controller email id entered by user id searched in the database then database returns the fetched user. After this password entered by the user is compared with password stored in database using “bcrypt.compare()” if password does not matches it will return false and error will be thrown. If “bcrypt.compare()” return true then

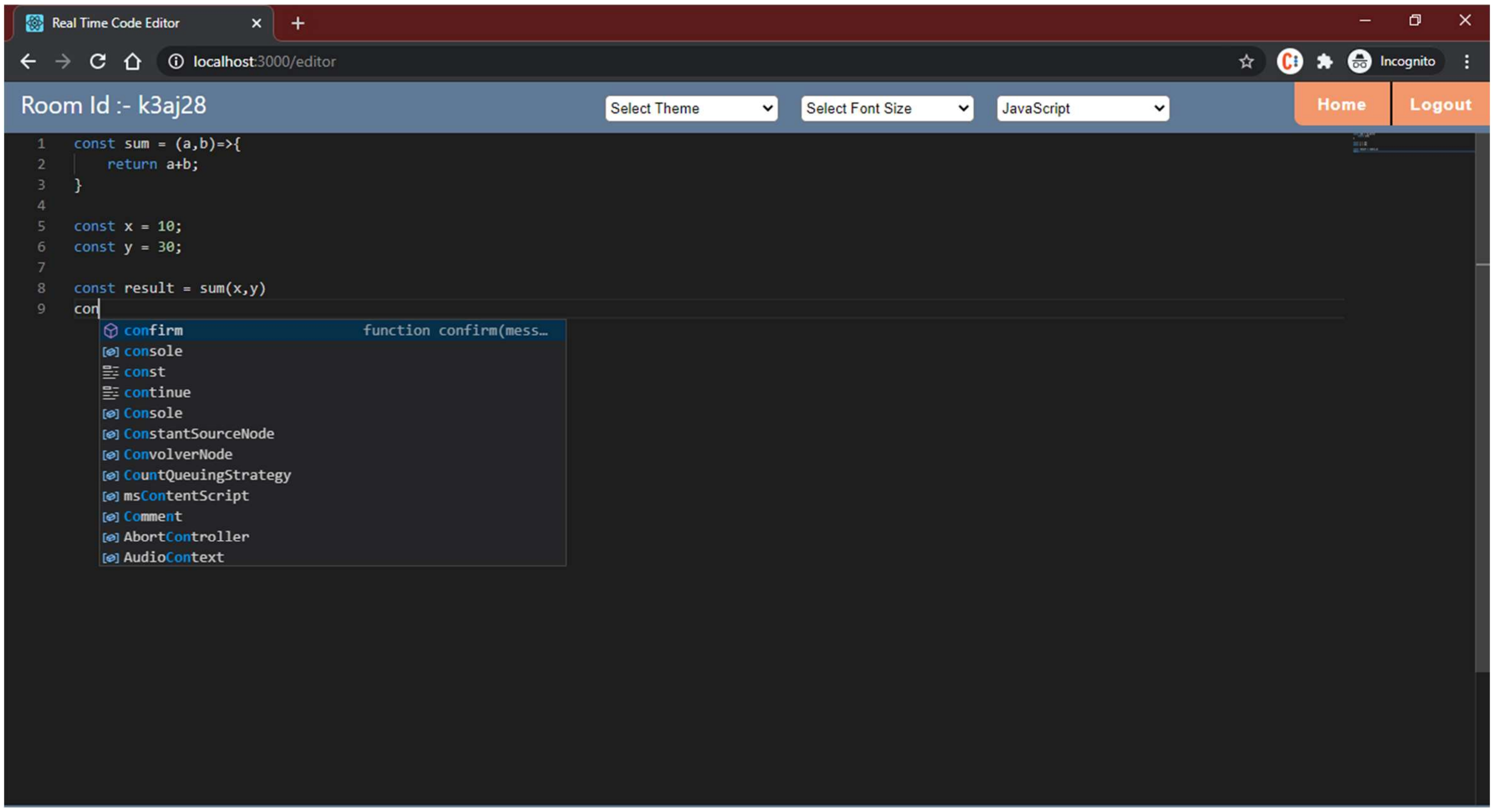
JSON Web Token (JWT) token is generated which will be active for 1 hour and its sent to the user with status code 200 along with name of user and id of the user.

Code for Login request sent from Frontend

```
export const startLogin = (email, password) => {
    return dispatch => {
        dispatch(startAuth());
        const formData = {
            email: email,
            password: password
        }
        axios.post('http://localhost:3005/login', formData)
            .then(res => {
                console.log(res.data);
                if (res.status === 422) {
                    throw new Error('Validation failed.');
                }
                if (res.status !== 200 && res.status !== 201) {
                    console.log('Error!');
                    throw new Error('Could not authenticate you!');
                }
                localStorage.setItem('token', res.data.token);
                localStorage.setItem('userId', res.data.userId);
                const expiresIn = 60 * 60 * 1000;
                const expiryDate = new Date(new Date().getTime() + expiresIn);
                localStorage.setItem('expiryDate', expiryDate.toISOString());
                dispatch(setAuth(res.data.name));
                dispatch(autoLogout(expiresIn));
            })
            .catch(err => {
                dispatch(loginFailed(err.response.data.message));
            })
    }
}
```

The above code after send the request we will receive token, user Id, and user name from the server. Token received from the backend will be stored in the local storage and will be active for 1 hour after that user will be logged out automatically and local storage will be cleared. If the user logout of the session before 1 hour this token will be deleted from the local storage.

Code Editor



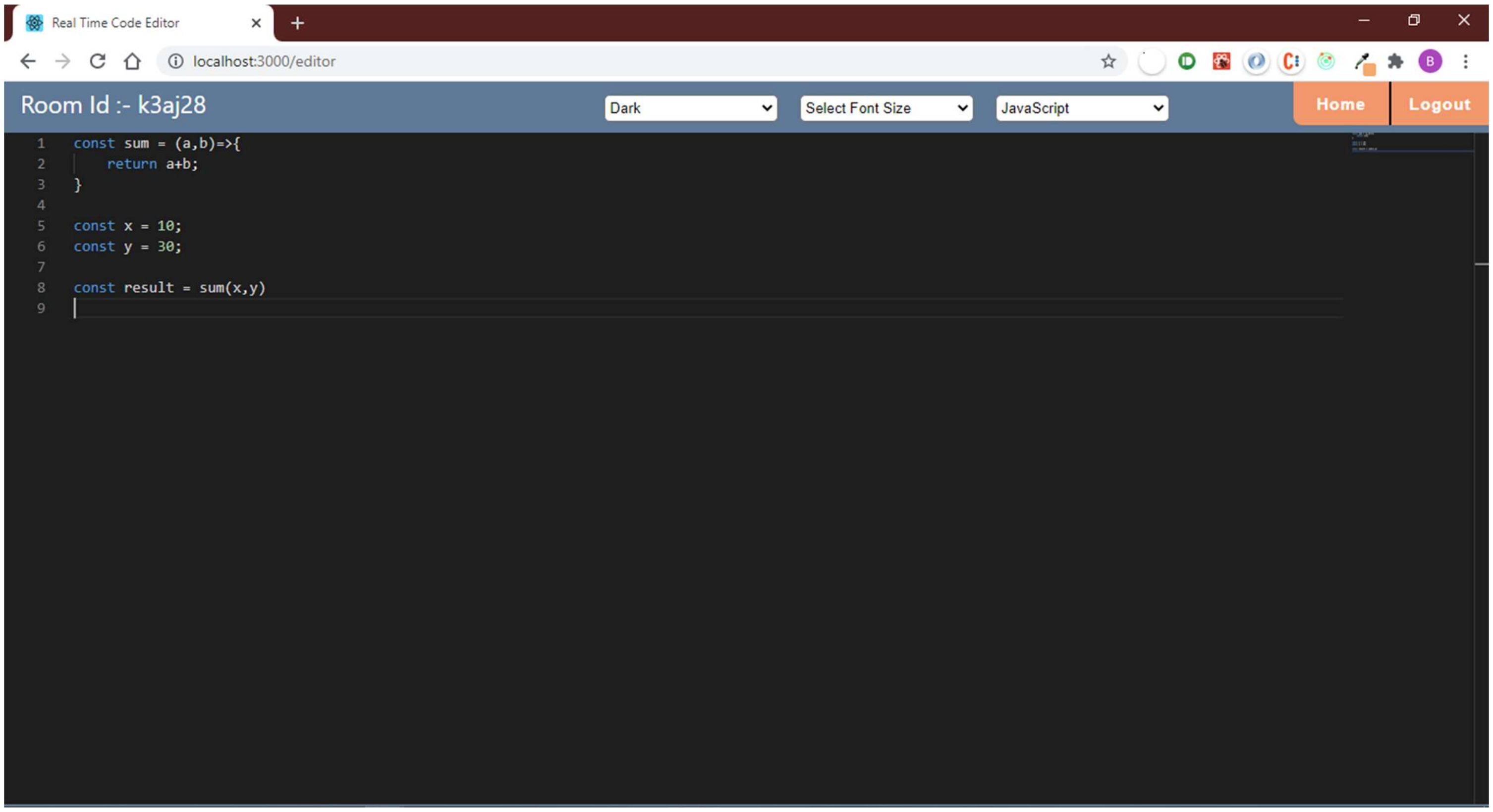
```
const sum = (a,b)=>{
    return a+b;
}

const x = 10;
const y = 30;

const result = sum(x,y)
con
```

This following editor is built with “@monaco-editor/react” library. User can change theme of the Editor to either dark or light, can increase and decrease the size of the text, and user can also select a lagugage in which the user wants to write the code.

This editor also has the syntax suggestion feature.

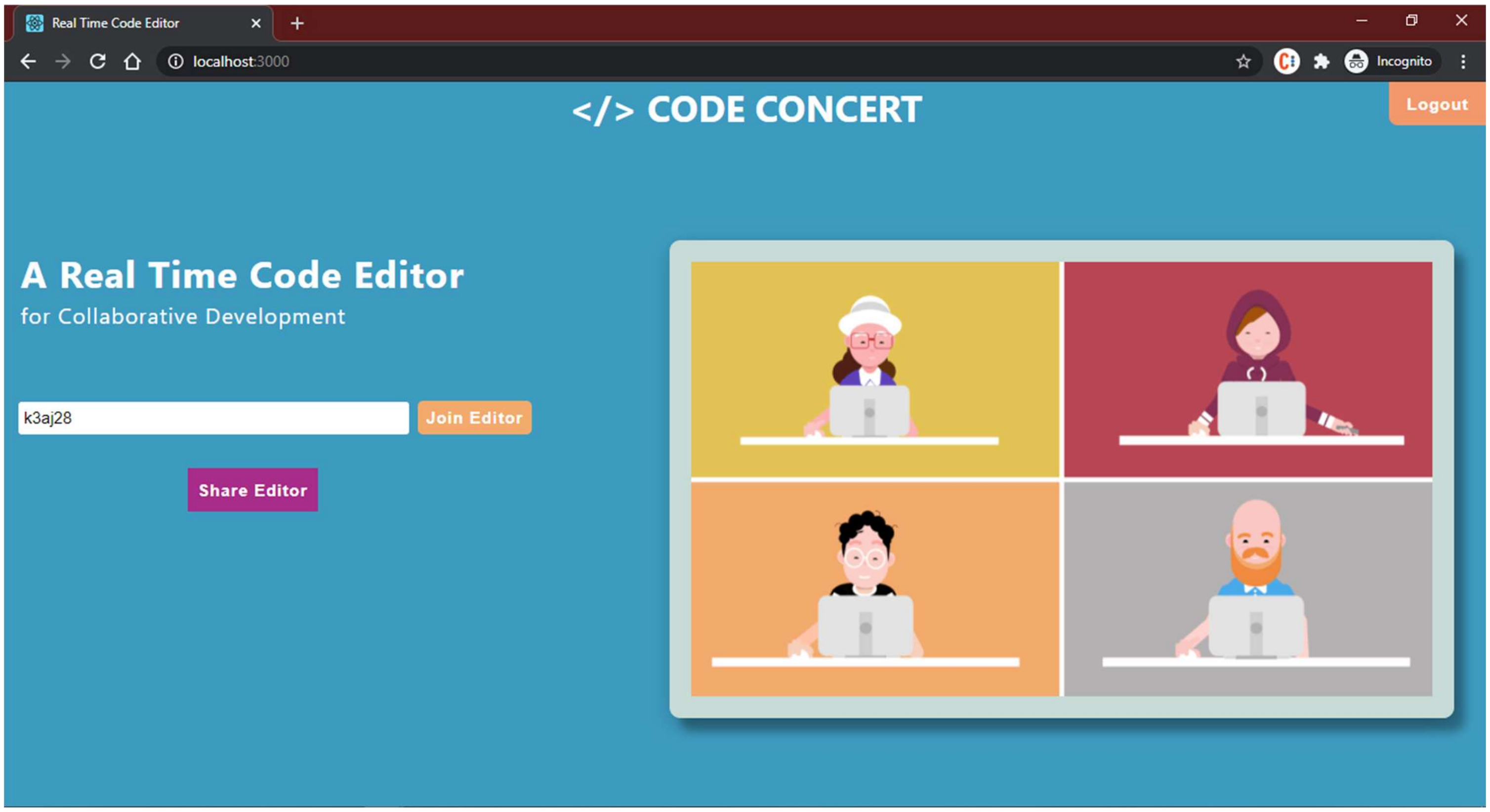
This page also contains a Room Id which can be shared across the user for joining them to the editor.

Connecting 2 Users to Editor

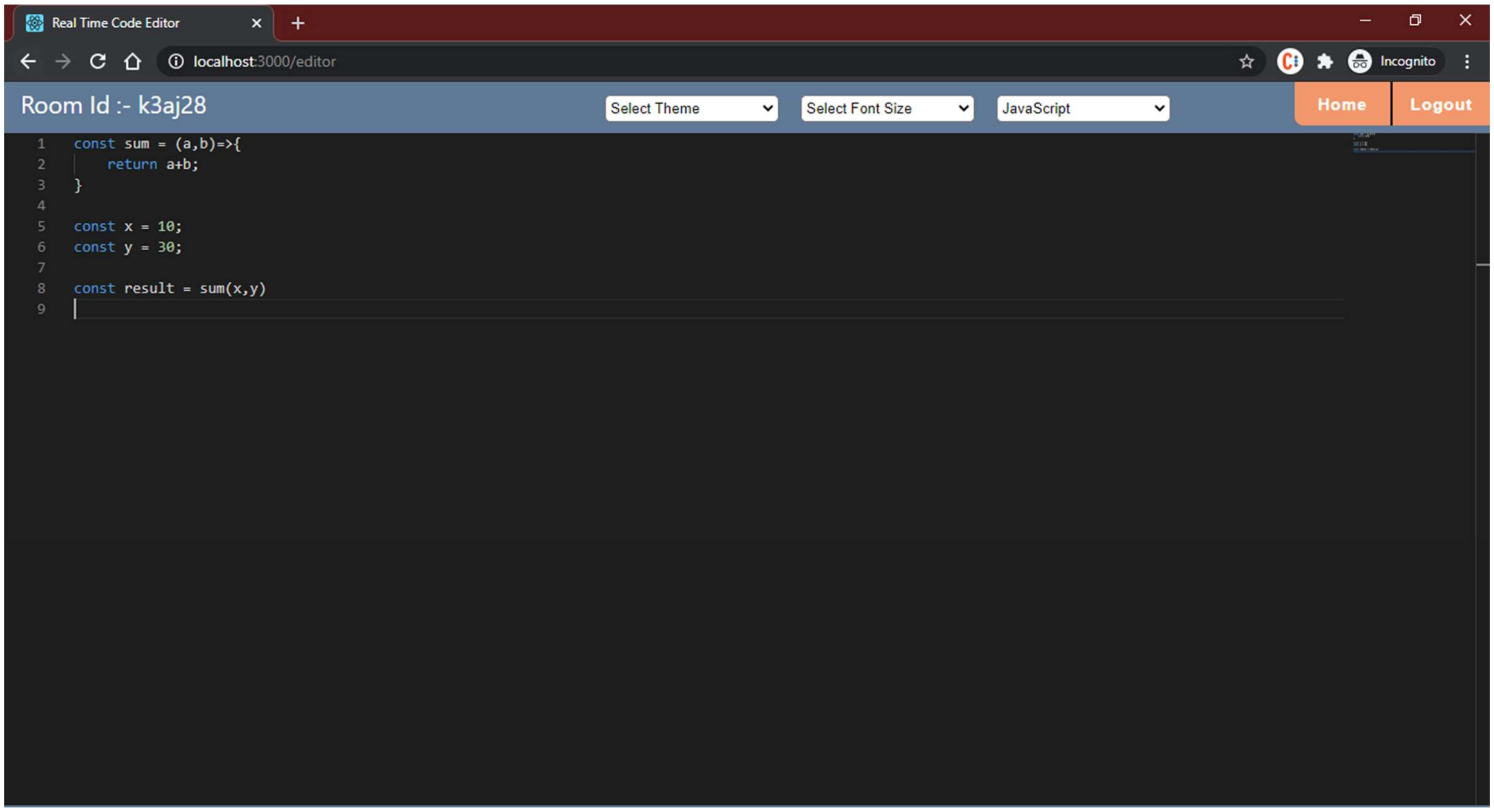


```
const sum = (a,b)=>{
    return a+b;
}

const x = 10;
const y = 30;

const result = sum(x,y)

```

User A



USER B



```
const sum = (a,b)=>{
    return a+b;
}

const x = 10;
const y = 30;

const result = sum(x,y)

```

USER B

USECASE OF THIS PROJECT

- This project can be used for taking coding interviews remotely and interviewer can observe the candidate in real-time.
- This project can be used by colleges and universities for teaching student.
- Can also be used by developers to debug errors in a project collaboratively by writing code on the editor or copy-pasting the code.