## Secure Chat Room Implementation Reference Document

### Overview

This guide outlines the procedures for completing the Secure Chat Room Implementation assignment, using JavaScript, Node.js, MySQL, and Express. The application must ensure security, user authentication, and interactive communication.

**Objectives:**

- Implement secure user registration and authentication using JWTs.

- Allow prime members to create chat rooms.

- Enable real-time communication within chat rooms using WebSockets.

- Implement profile viewing and friend request functionalities.

### Step-by-Step Instructions

#### 1. Setting Up the Project

**Prerequisites:**

- Node.js and npm installed.

- MySQL database set up.

- Git for version control.

**Steps:**

1. **Initialize the project:**

```sh

mkdir secure-chat-room

cd secure-chat-room

npm init -y

```

2. **Install dependencies:**

```sh

npm install express mysql2 bcrypt jsonwebtoken ws

```

3. **Set up version control:**

```sh

git init

git remote add origin <your\_github\_repository\_url>

```

#### 2. User Registration and Authentication

1. **Create necessary files:**

- `server.js`: Main entry point of the application.

- `routes/`: Folder to store route handlers.

- `models/`: Folder to store database schema and queries.

- `middleware/`: Folder to store authentication and error handling middleware.

2. **User Model and Registration Endpoint:**

- Create a user model in `models/user.js`:

```javascript

const bcrypt = require('bcrypt');

const db = require('../database');

// User registration example:

const registerUser = async (userData) => {

const hash = await bcrypt.hash(userData.password, 10);

// Save user with hashed password

await db.query('INSERT INTO users SET ?', { ...userData, password: hash });

};

```

- In `routes/auth.js`:

```javascript

const express = require('express');

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

const router = express.Router();

router.post('/register', async (req, res) => {

// Validation and user creation logic here...

});

router.post('/login', async (req, res) => {

// User login and JWT token generation process...

});

module.exports = router;

```

3. **Add authentication middleware for route protection:**

```javascript

const jwt = require('jsonwebtoken');

const authenticateJWT = (req, res, next) => {

const token = req.header('Authorization');

if (!token) {

return res.status(401).send('Access Denied');

}

try {

const verified = jwt.verify(token, 'secretKey');

req.user = verified;

next();

} catch (err) {

res.status(400).send('Invalid Token');

}

};

module.exports = authenticateJWT;

```

#### 3. Chat Room Creation

1. **Create `chatroom.js` in `models`:**

```javascript

const createChatRoom = async (roomData) => {

// Insert chat room details into the database

await db.query('INSERT INTO chatrooms SET ?', roomData);

};

module.exports = { createChatRoom };

```

2. **Create room endpoint in `routes/chatroom.js`:**

```javascript

const express = require('express');

const { createChatRoom } = require('../models/chatroom');

const authenticateJWT = require('../middleware/authenticateJWT');

const router = express.Router();

router.post('/chatrooms', authenticateJWT, async (req, res) => {

// Room creation logic

});

module.exports = router;

```

3. **Validation for prime members and room capacity:**

Add logic in the route handler to check if the user is prime and ensure room does not exceed 6 participants.

#### 4. Inviting Participants

1. **Token generation for invitations:**

Use JWT to create secure tokens:

```javascript

const inviteToken = jwt.sign({ roomId, userId }, 'secretKey', { expiresIn: '1h' });

```

2. **Endpoint for inviting participants:**

Add an endpoint in `routes/chatroom.js`:

```javascript

router.post('/invite', authenticateJWT, (req, res) => {

// Invitation logic here

});

```

#### 5. Chat Functionality

1. **WebSocket setup in `server.js`:**

```javascript

const WebSocket = require('ws');

const wss = new WebSocket.Server({ server });

wss.on('connection', (ws) => {

ws.on('message', (message) => {

// Broadcast message to all clients

wss.clients.forEach(client => {

if (client.readyState === WebSocket.OPEN) {

client.send(message);

}

});

});

});

```

2. **Client-side setup for WebSocket:**

Example code for initiating WebSocket connection on the client side.

#### 6. Profile Viewing and Friend Requests

1. **Profile retrieval in `models/user.js`:**

```javascript

const getUserProfile = async (userId) => {

return await db.query('SELECT \* FROM users WHERE userId = ?', [userId]);

};

```

2. **Profile endpoint in `routes/profile.js`:**

```javascript

router.get('/profile/:userId', authenticateJWT, async (req, res) => {

const profile = await getUserProfile(req.params.userId);

res.json(profile);

});

```

3. **Friend request functionality:**

Add logic for sending/receiving friend requests in `routes/friendRequests.js`.

#### 7. Database Management

1. **Design database schema:**

- Create tables for users, chatrooms, messages, and friend requests.

- Example for users:

```sql

CREATE TABLE users (

userId INT PRIMARY KEY AUTO\_INCREMENT,

deviceId VARCHAR(255),

name VARCHAR(255),

phone VARCHAR(15),

availCoins INT,

password VARCHAR(255)

);

```

2. **Configure MySQL connection:**

```javascript

const mysql = require('mysql2');

const connection = mysql.createConnection({

host: 'localhost',

user: 'root',

password: 'password',

database: 'secure\_chat'

});

connection.connect();

module.exports = connection;

```

#### 8. Security Measures

- Use bcrypt to hash passwords before storing them.

- Ensure all endpoints that require authentication check the JWT token.

- Secure sensitive information using environment variables for secret keys and database credentials.

#### 9. Error Handling and Validation

- Implement proper validation for all endpoints using express validators.

- Catch and handle errors, sending appropriate responses to the client.

#### 10. Version Control

- Use Git for version control.

- Commit changes frequently with clear, descriptive messages.

- Push to your GitHub repository.

### Best Practices

- **Code Quality:** Write clean, modular code. Use meaningful variable names.

- **Security:** Never expose sensitive information like API keys or secrets.

- **Testing:** Test each component thoroughly before moving to the next step.

- **Documentation:** Comment your code and maintain updated documentation.

### Submission Guidelines

- **Submission Deadline:** 6th June, 2024, till 7 P.M.

- **Submit to:** rishav@leadmint.io, cc: shabaj@leadmint.io

- **Include:**

- GitHub repository link with complete code.

- A README file with setup and running instructions.

- Database schema in a separate `.sql` file.

- Detailed documentation of API endpoints.

### FAQ Section

**Q1:** What should I do if I face issues setting up my environment?

**A:** Ensure all dependencies are properly installed. Check the official documentation of Node.js, MySQL, and any other libraries you are using.

**Q2:** How do I test my JWT tokens?

**A:** Use tools like Postman to test your API endpoints, including generating and using JWT tokens.

**Q3:** What happens if a chat room exceeds the maximum capacity?

**A:** Ensure the server-side logic prevents any additional participants from joining once the limit is reached.

For any further queries, drop an email at rishav@leadmint.io or shabaj@leadmint.io. Good luck!