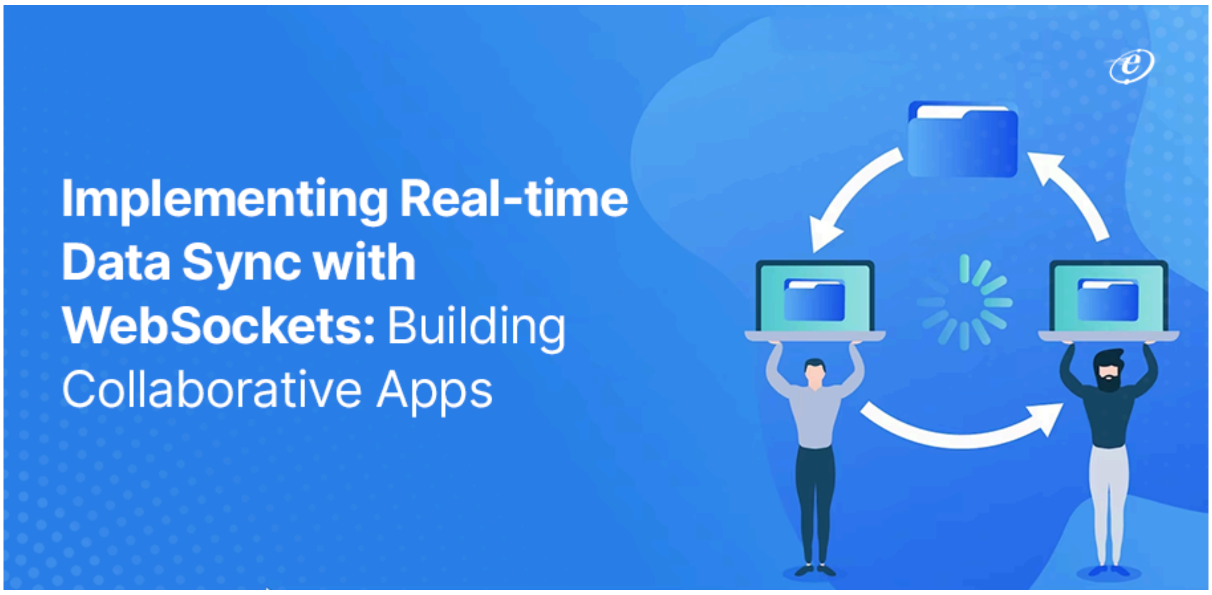

Implementing Real-time Data Sync with WebSockets: Building Collaborative Apps

Use websockets to sync data in realtime

Boigantso Mabena

1. Sprint Goals

- Develop a real-time status update system using WebSockets.
- Set up a MySQL database for storing status updates.
- Implement a Node.js server with `socket.io` for real-time communication.
- Create a simple web interface to add and display status updates.

2. Sprint Backlog

User Stories and Tasks

1. Database Setup
    - **User Story**: As a developer, I need to set up a MySQL database to store status updates so that the application can persist and retrieve statuses.
    - **Tasks**:
        - Create a MySQL database and table for storing statuses.
        - Verify database schema and test basic CRUD operations.
2. Server Setup
    - **User Story**: As a developer, I need to set up a Node.js server to handle WebSocket connections and interact with the MySQL database.
    - **Tasks**:
        - Set up a Node.js project with `express`, `mysql`, and `socket.io`.
        - Implement server-side logic to handle WebSocket connections (`io.on`).
        - Implement database interactions for storing and retrieving statuses.
        - Implement real-time updates using `socket.io`.
3. Client Interface
    - **User Story**: As a user, I want to be able to submit status updates and see updates from other users in real-time.
    - **Tasks**:
        - Create a web interface with an input field and a submit button.
        - Implement JavaScript to handle WebSocket communication (`socket.emit` and `socket.on`).
        - Display status updates on the web page in real-time.
4. Testing and Debugging
    - **User Story**: As a developer, I need to test the system to ensure it functions correctly and fix any bugs that arise.
    - **Tasks**:
        - Write test cases for the server-side logic and WebSocket interactions.
        - Perform manual testing of the client interface.
        - Fix any issues identified during testing.
5. Documentation and Deployment
    - **User Story**: As a developer, I need to document the setup and deployment instructions so that the application can be easily set up and deployed in different environments.
    - **Tasks**:

        - Document the setup instructions for the MySQL database and Node.js server.
        - Create deployment instructions for deploying the application to a cloud service or production environment.

3. Sprint Timeline

Week 1

- **Day 1-2**:
    - Database setup and schema design.
    - Initial server setup with Node.js, `express`, and `mysql`.
- **Day 3-4**:
    - Implement server-side logic for WebSocket communication and database interaction.
    - Develop client-side interface with basic status input and display functionality.
- **Day 5**:
    - Integrate WebSocket functionality in the client interface.
    - Test basic real-time updates and debug any initial issues.

Week 2

- **Day 6-7**:
    - Continue testing and debugging, focusing on edge cases and performance.
    - Implement additional features or improvements based on feedback.
- **Day 8-10**:
    - Finalise testing, including both manual and automated tests.
    - Prepare documentation and deployment instructions.
- **Day 11-12**:
    - Deploy the application to a cloud service or production environment.
    - Conduct final review and ensure all tasks are completed.
- **Day 13-14**:
    - Sprint review and retrospective.
    - Gather feedback and plan for any additional tasks or improvements for the next sprint.

4. Team Roles and Responsibilities

- **Project Manager**: Oversee sprint progress, ensure milestones are met, and facilitate communication among team members.
- **Backend Developer**: Responsible for server-side development, database interactions, and WebSocket implementation.
- **Frontend Developer**: Develop the client interface, handle WebSocket communication on the client side, and ensure real-time updates are properly displayed.
- **QA Tester**: Perform testing of the application, report bugs, and assist in debugging and final validation.

- **Technical Writer**: Document setup, deployment instructions, and any other necessary documentation.

5. Risk Management

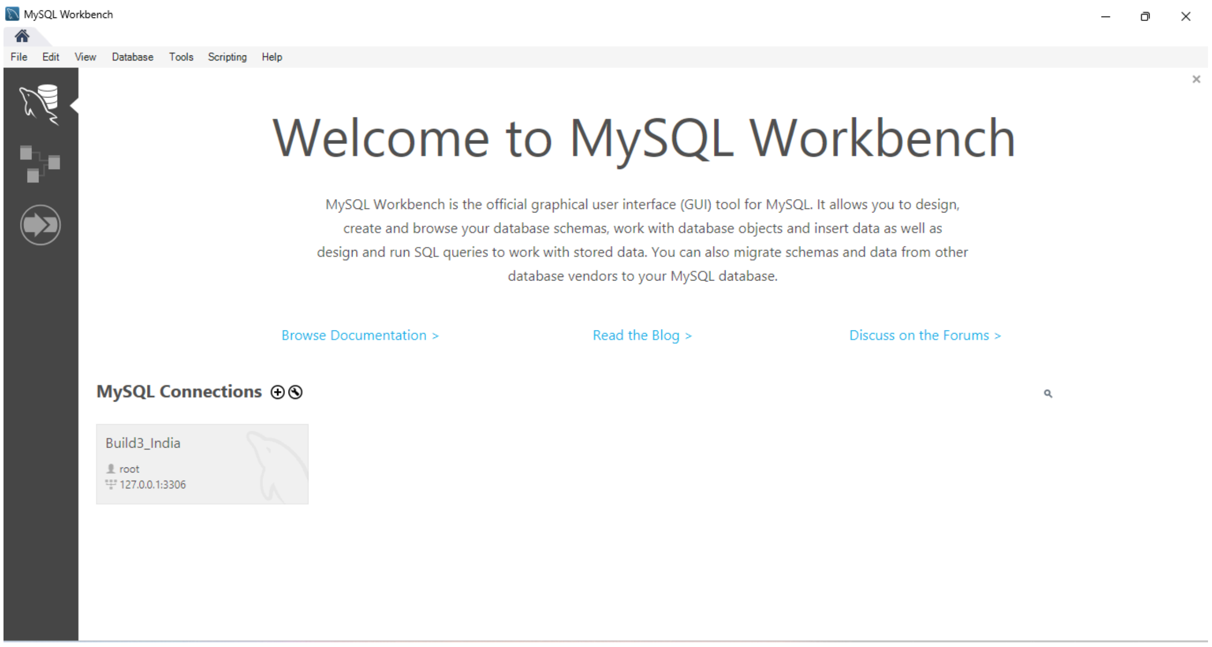
- **Risk**: Potential issues with WebSocket connectivity or performance.
    - **Mitigation**: Implement thorough testing and use performance monitoring tools.
- **Risk**: Delays in database setup or schema design.
    - **Mitigation**: Ensure database setup is prioritised early in the sprint and allocate buffer time for unexpected issues.

6. Review and Retrospective

- **Review Meeting**: Conducted at the end of the sprint to assess completed tasks, review achievements, and gather feedback.
- **Retrospective Meeting**: Discuss what went well, what could be improved, and plan actions for the next sprint.

This sprint planning document outlines a structured approach to developing a real-time status update application using WebSockets and Node.js. Adjust the details based on your team's specific needs, resources, and feedback during the sprint.

1. Create a MySQL database



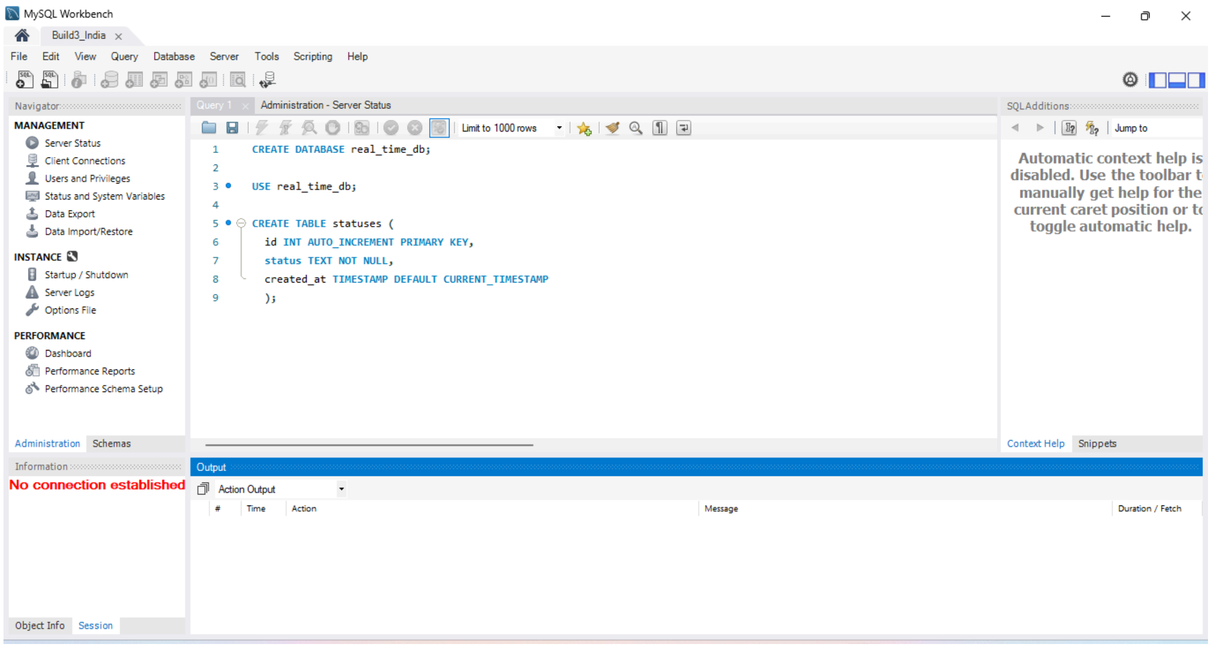
Welcome to MySQL Workbench

MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

Browse Documentation > Read the Blog > Discuss on the Forums >

MySQL Connections

Build3_India
root
127.0.0.1:3306

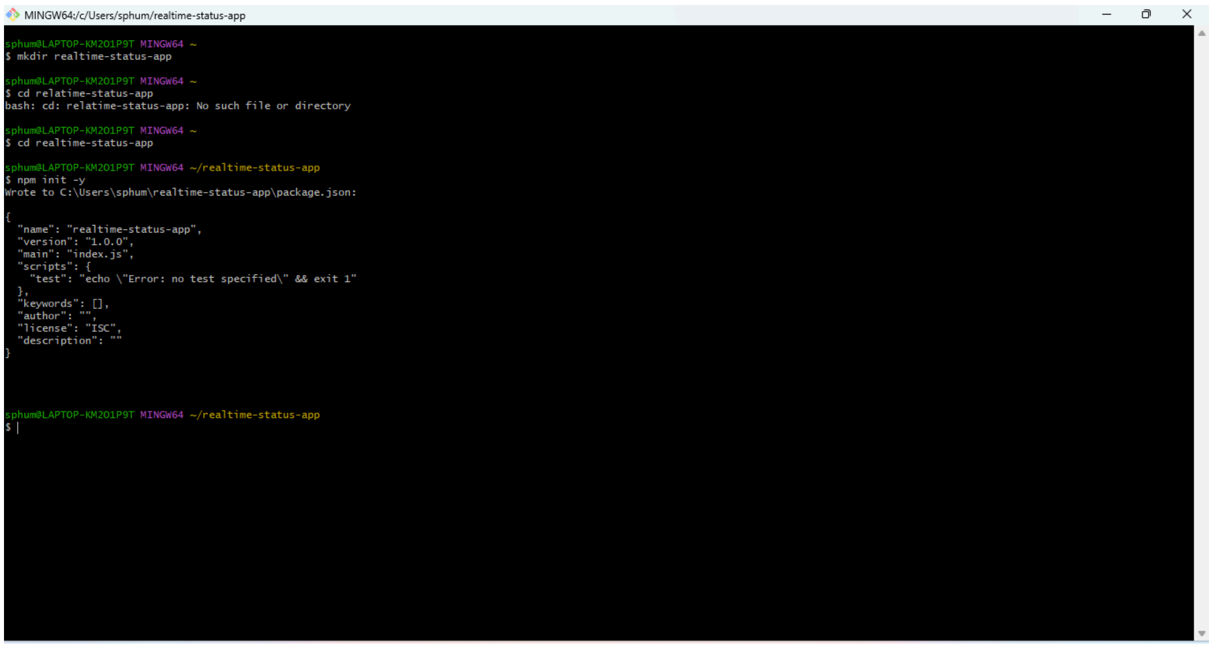


```sql
CREATE DATABASE real_time_db;

USE real_time_db;

CREATE TABLE statuses (
  id INT AUTO_INCREMENT PRIMARY KEY,
  status TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
  );
```

No connection established

Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or tc toggle automatic help.

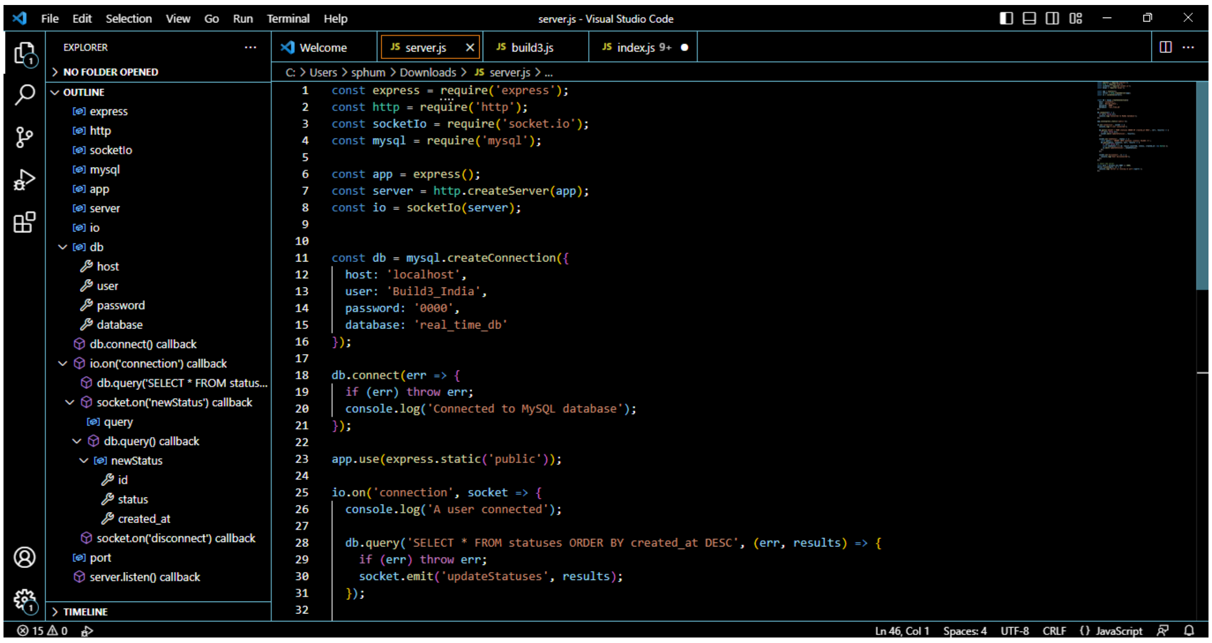
Initialise NodeJS Project



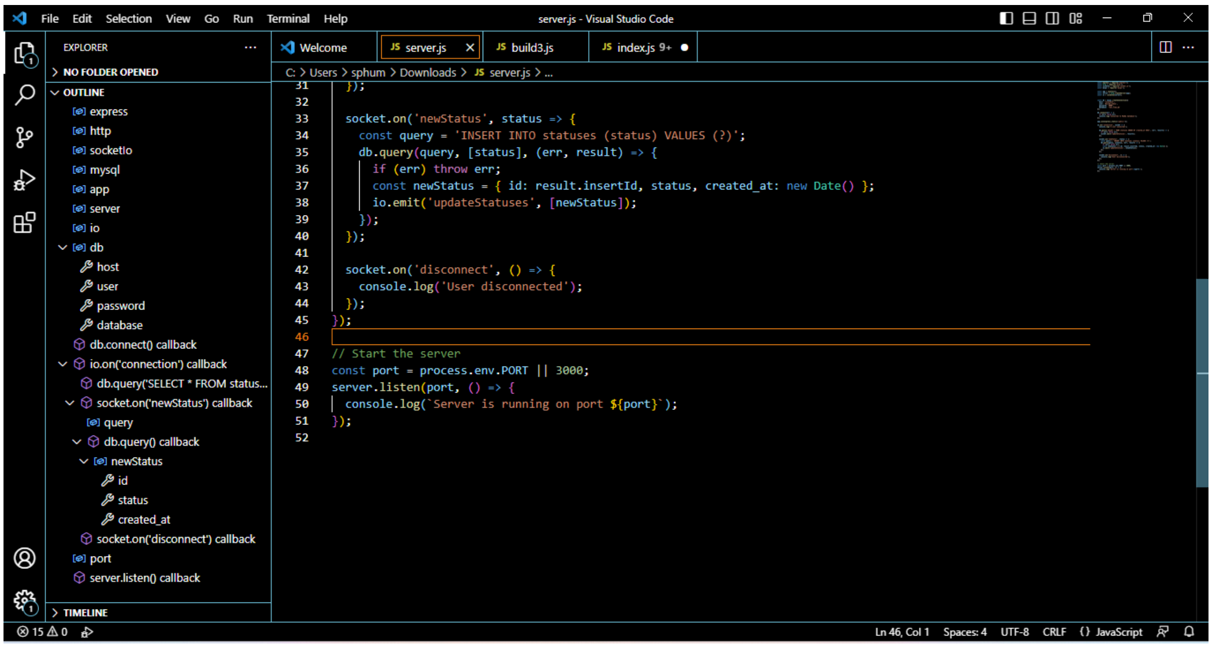
MINGW64:/c/Users/sphum/realtime-status-app

```
sphum@LAPTOP-KM201P9T MINGW64 ~
$ mkdir realtime-status-app

sphum@LAPTOP-KM201P9T MINGW64 ~
$ cd relatime-status-app
bash: cd: relatime-status-app: No such file or directory

sphum@LAPTOP-KM201P9T MINGW64 ~
$ cd realtime-status-app

sphum@LAPTOP-KM201P9T MINGW64 ~/realtime-status-app
$ npm init -y
Wrote to C:\Users\sphum\realtime-status-app\package.json:

{
  "name": "realtime-status-app",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}



sphum@LAPTOP-KM201P9T MINGW64 ~/realtime-status-app
$ |
```

2. Setup the server.js file with all the dependencies

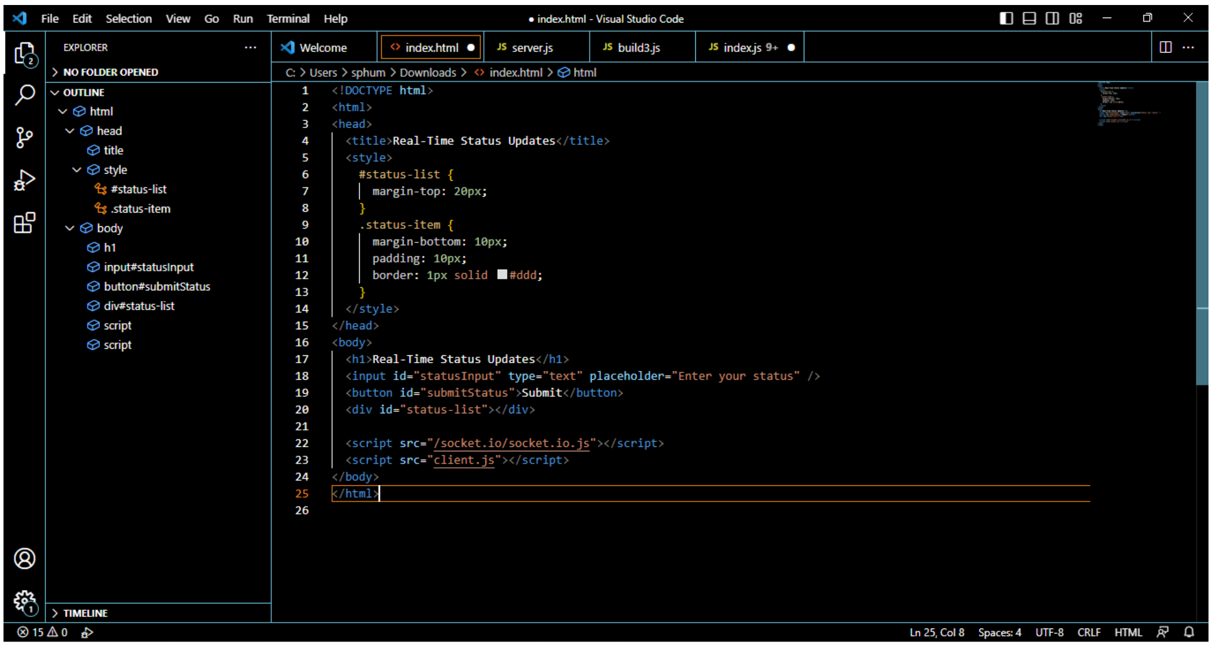

File Edit Selection View Go Run Terminal Help
server.js - Visual Studio Code
EXPLORER
NO FOLDER OPENED
OUTLINE
Welcome
server.js
build3.js
index.js 9+
C: > Users > sphum > Downloads > server.js > ...
TIMELINE
Ln 46, Col 1 Spaces: 4 UTF-8 CRLF {} JavaScript

```
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');
const mysql = require('mysql');

const app = express();
const server = http.createServer(app);
const io = socketIo(server);

const db = mysql.createConnection({
  host: 'localhost',
  user: 'Build3_India',
  password: '0000',
  database: 'real_time_db'
});

db.connect(err => {
  if (err) throw err;
  console.log('Connected to MySQL database');
});

app.use(express.static('public'));

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

  db.query('SELECT * FROM statuses ORDER BY created_at DESC', (err, results) => {
    if (err) throw err;
    socket.emit('updateStatuses', results);
  });

```


File Edit Selection View Go Run Terminal Help
server.js - Visual Studio Code
EXPLORER
NO FOLDER OPENED
OUTLINE
Welcome
server.js
build3.js
index.js 9+
C: > Users > sphum > Downloads > server.js > ...
TIMELINE
Ln 46, Col 1 Spaces: 4 UTF-8 CRLF {} JavaScript

```
  });

  socket.on('newStatus', status => {
    const query = 'INSERT INTO statuses (status) VALUES (?)';
    db.query(query, [status], (err, result) => {
      if (err) throw err;
      const newStatus = { id: result.insertId, status, created_at: new Date() };
      io.emit('updateStatuses', [newStatus]);
    });
  });

  socket.on('disconnect', () => {
    console.log('User disconnected');
  });
});

// Start the server
const port = process.env.PORT || 3000;
server.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});

```

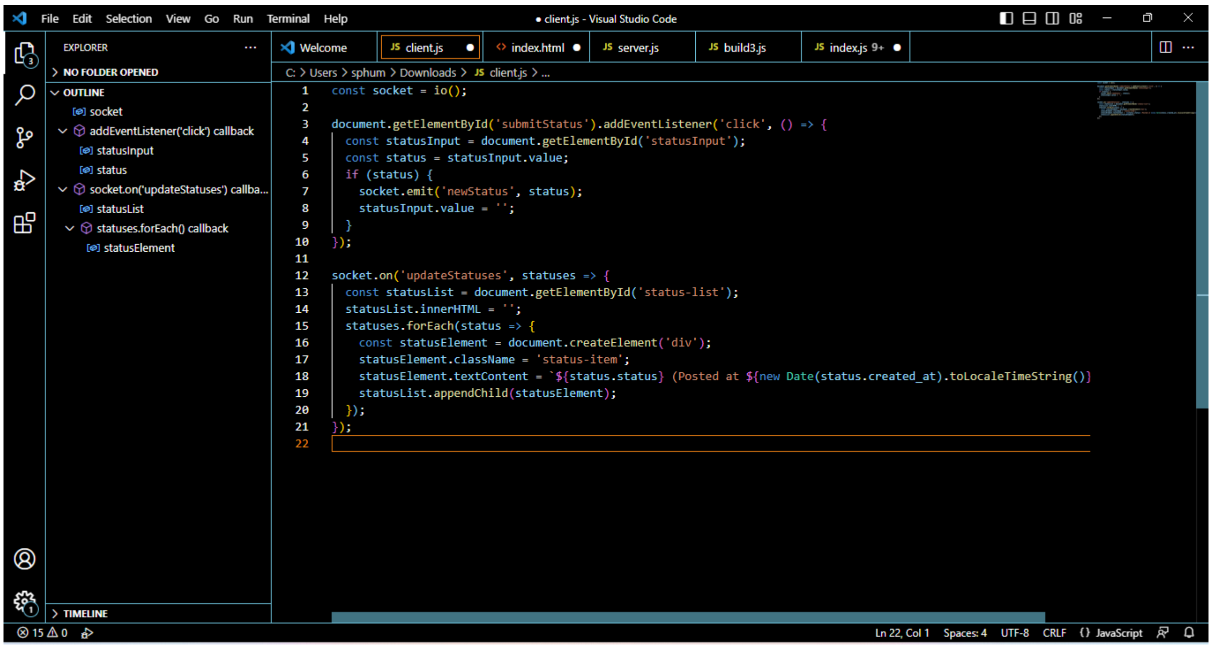
Directory with `index.html` and `client.js`



File Edit Selection View Go Run Terminal Help
● index.html - Visual Studio Code
EXPLORER
NO FOLDER OPENED
OUTLINE
TIMELINE
Welcome
index.html
server.js
build3.js
index.js 9+
C: > Users > sphum > Downloads > index.html > html
Ln 25, Col 8 Spaces: 4 UTF-8 CRLF HTML

```html
<!DOCTYPE html>
<html>
<head>
  <title>Real-Time Status Updates</title>
  <style>
    #status-list {
      margin-top: 20px;
    }
    .status-item {
      margin-bottom: 10px;
      padding: 10px;
      border: 1px solid #ddd;
    }
  </style>
</head>
<body>
  <h1>Real-Time Status Updates</h1>
  <input id="statusInput" type="text" placeholder="Enter your status" />
  <button id="submitStatus">Submit</button>
  <div id="status-list"></div>

  <script src="/socket.io/socket.io.js"></script>
  <script src="client.js"></script>
</body>
</html>
```

3. In the client.js file, use ‘socket.emit’ to emit commands which will be caught by the ‘io.on’ on the server.side


File Edit Selection View Go Run Terminal Help
• client.js - Visual Studio Code
EXPLORER
NO FOLDER OPENED
OUTLINE
socket
addEventListener('click') callback
statusInput
status
socket.on('updateStatuses') callba...
statusList
statuses.forEach() callback
statusElement
TIMELINE
Welcome
client.js
index.html
server.js
build3.js
index.js 9+
C: > Users > sphum > Downloads > client.js > ...
15 0
Ln 22, Col 1 Spaces: 4 UTF-8 CRLF {} JavaScript

```
const socket = io();

document.getElementById('submitStatus').addEventListener('click', () => {
  const statusInput = document.getElementById('statusInput');
  const status = statusInput.value;
  if (status) {
    socket.emit('newStatus', status);
    statusInput.value = '';
  }
});

socket.on('updateStatuses', statuses => {
  const statusList = document.getElementById('status-list');
  statusList.innerHTML = '';
  statuses.forEach(status => {
    const statusElement = document.createElement('div');
    statusElement.className = 'status-item';
    statusElement.textContent = `${status.status} (Posted at ${new Date(status.created_at).toLocaleTimeString()}
    statusList.appendChild(statusElement);
  });
});

```

Summary

- **MySQL Database**: Set up a database and table to store statuses.
- **Node.js Project**: Initialise the project, install dependencies, and set up the server to handle real-time communication and database interactions.
- **WebSocket Integration**: Use `socket.io` to enable real-time updates for all connected clients.
- **Client Side**: Implement a simple web interface to submit statuses and display real-time updates.

This basic setup provides a foundation for real-time data synchronisation using WebSockets and can be expanded with more features and robust error handling as needed.