

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

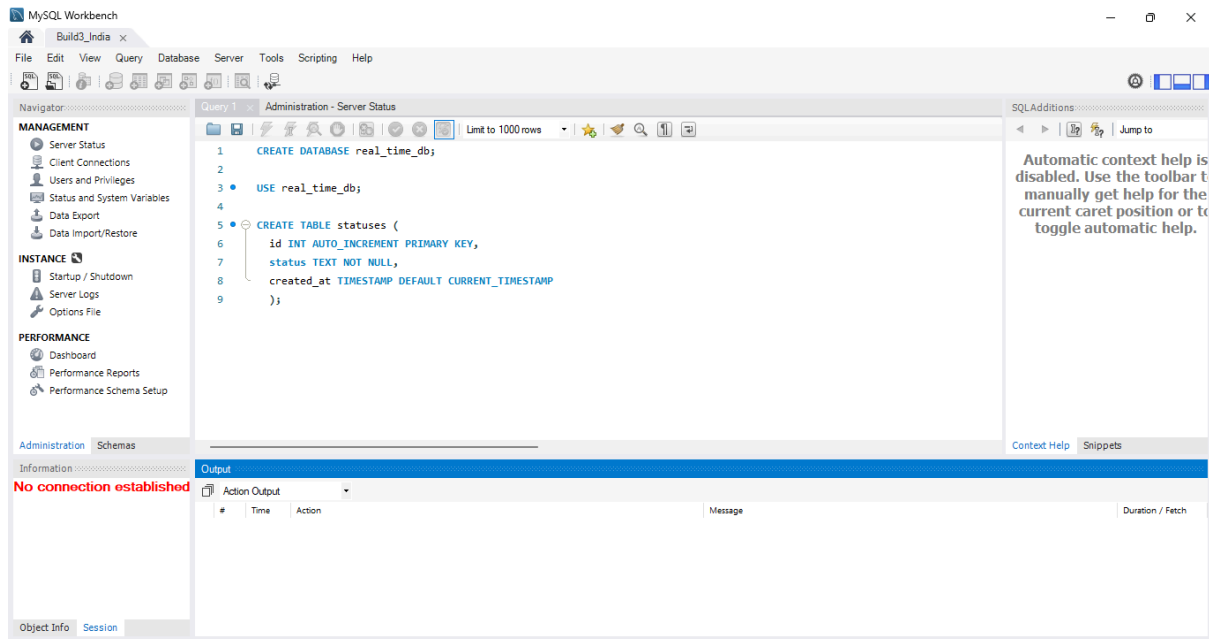
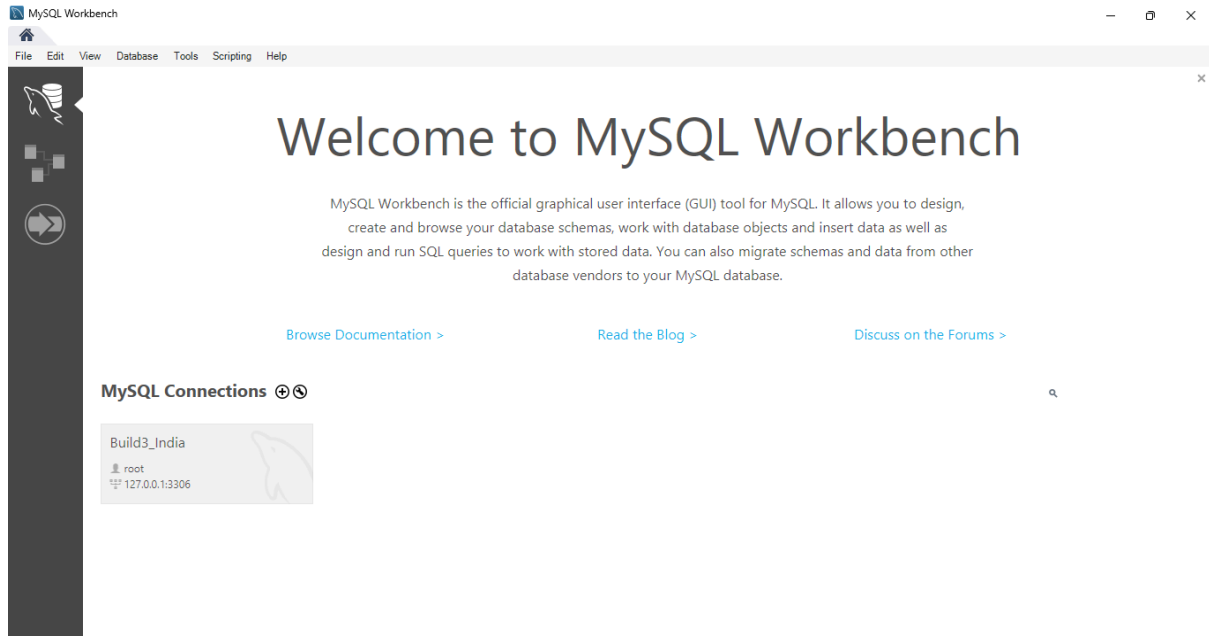
- **Risk:** Potential issues with WebSocket connectivity or performance.
 - **Mitigation:** Implement thorough testing and use performance monitoring tools.
- **Risk:** Delays in database setup or schema design.
 - **Mitigation:** Ensure database setup is prioritised early in the sprint and allocate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

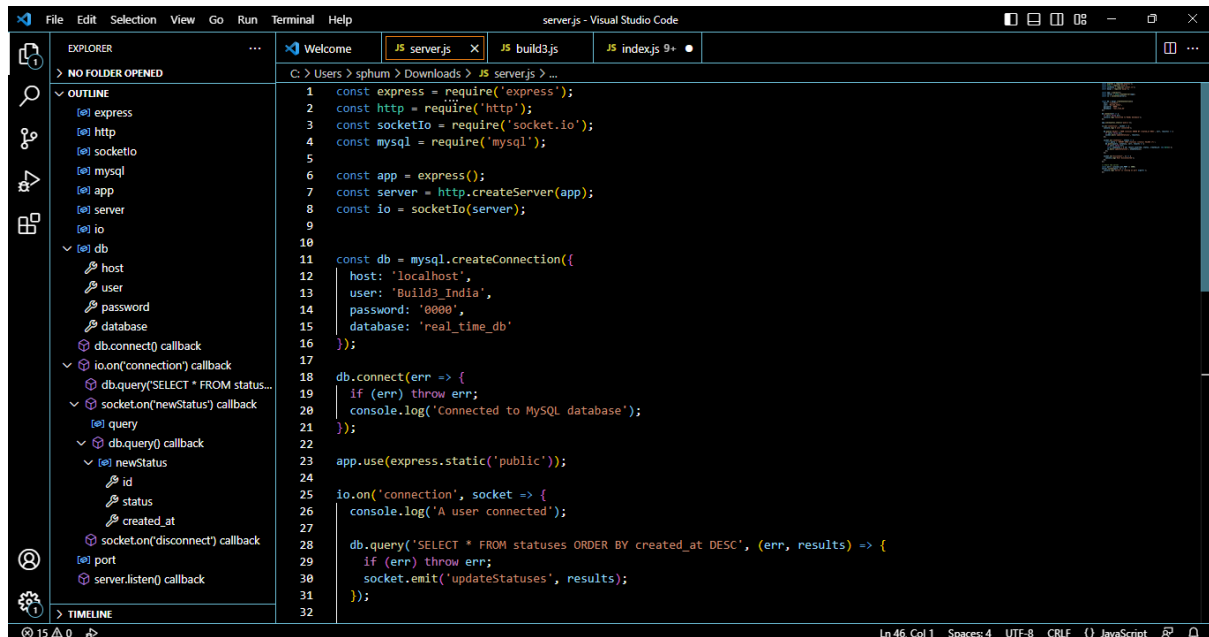


Initialise NodeJS Project

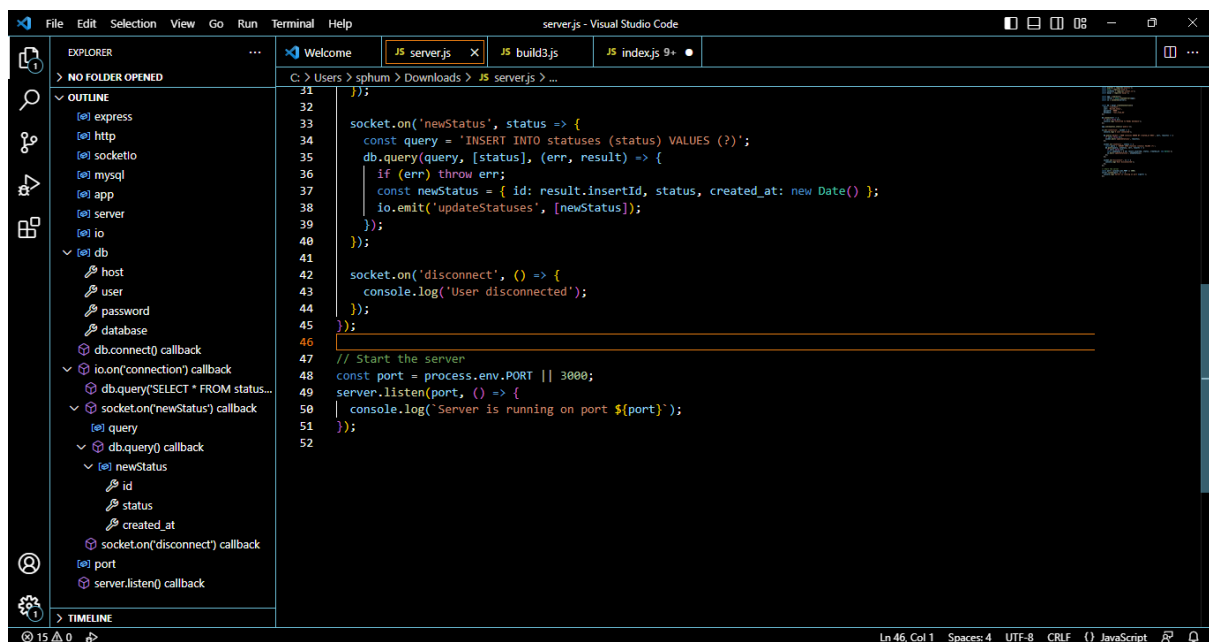
```
MINGW64/c/Users/sphum/realtime-status-app
sphum@LAPTOP-KM201P9T MINGW64 ~
$ mkdir realtime-status-app
sphum@LAPTOP-KM201P9T MINGW64 ~
$ cd realtime-status-app
bash: cd: realtime-status-app: No such file or directory
sphum@LAPTOP-KM201P9T MINGW64 ~
$ cd realtime-status-app
sphum@LAPTOP-KM201P9T MINGW64 ~/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



```
1 const express = require('express');
2 const http = require('http');
3 const socketIo = require('socket.io');
4 const mysql = require('mysql');
5
6 const app = express();
7 const server = http.createServer(app);
8 const io = socketIo(server);
9
10
11 const db = mysql.createConnection({
12   host: 'localhost',
13   user: 'Build3_India',
14   password: '0000',
15   database: 'real_time_db'
16 });
17
18 db.connect(err => {
19   if (err) throw err;
20   console.log('Connected to MySQL database');
21 });
22
23 app.use(express.static('public'));
24
25 io.on('connection', socket => {
26   console.log('A user connected');
27
28   db.query('SELECT * FROM statuses ORDER BY created_at DESC', (err, results) => {
29     if (err) throw err;
30     socket.emit('updateStatuses', results);
31   });
32 }
```

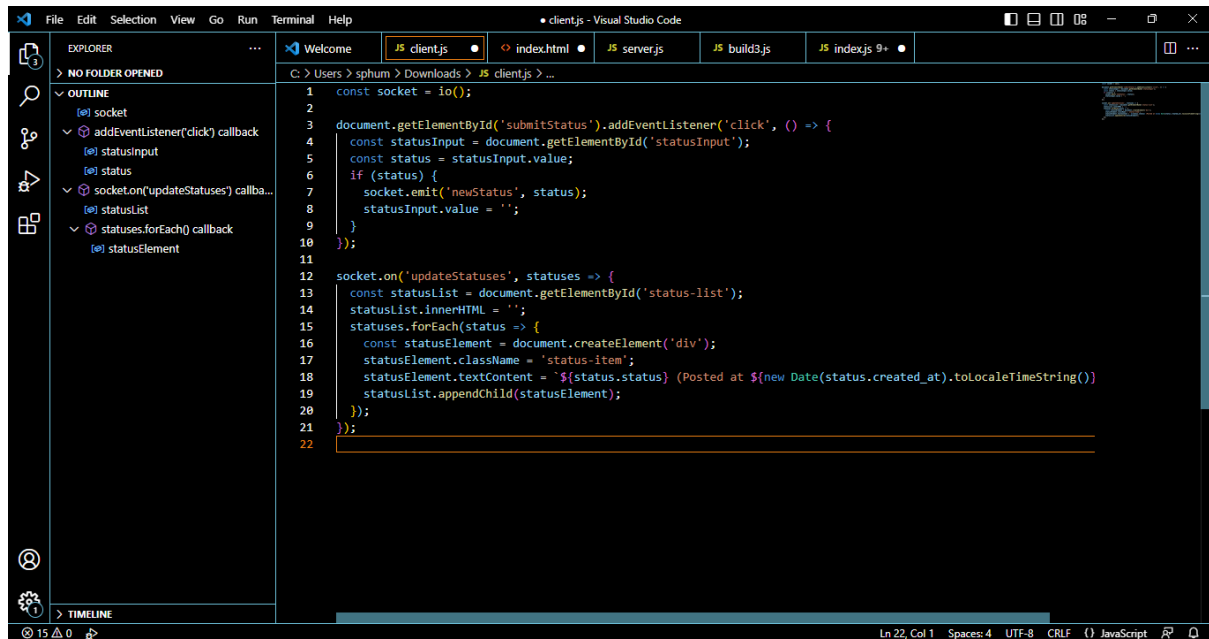


```
31 });
32
33 socket.on('newStatus', status => {
34   const query = 'INSERT INTO statuses (status) VALUES (?)';
35   db.query(query, [status], (err, result) => {
36     if (err) throw err;
37     const newStatus = { id: result.insertId, status, created_at: new Date() };
38     io.emit('updateStatuses', [newStatus]);
39   });
40 });
41
42 socket.on('disconnect', () => {
43   console.log('User disconnected');
44 });
45
46
47 // Start the server
48 const port = process.env.PORT || 3000;
49 server.listen(port, () => {
50   console.log('Server is running on port ${port}');
51 });
52 }
```

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

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Real-Time Status Updates</title>
5   <style>
6     #status-list {
7       margin-top: 20px;
8     }
9     .status-item {
10      margin-bottom: 10px;
11      padding: 10px;
12      border: 1px solid #ddd;
13    }
14  </style>
15 </head>
16 <body>
17   <h1>Real-Time Status Updates</h1>
18   <input id="statusInput" type="text" placeholder="Enter your status" />
19   <button id="submitStatus">Submit</button>
20   <div id="status-list"></div>
21
22   <script src="/socket.io/socket.io.js"></script>
23   <script src="client.js"></script>
24 </body>
25 </html>
26
```


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



```
1 const socket = io();
2
3 document.getElementById('submitStatus').addEventListener('click', () => {
4   const statusInput = document.getElementById('statusInput');
5   const status = statusInput.value;
6   if (status) {
7     socket.emit('newStatus', status);
8     statusInput.value = '';
9   }
10 });
11
12 socket.on('updateStatuses', statuses => {
13   const statusList = document.getElementById('status-list');
14   statusList.innerHTML = '';
15   statuses.forEach(status => {
16     const statusElement = document.createElement('div');
17     statusElement.className = 'status-item';
18     statusElement.textContent = `${status.status} (Posted at ${new Date(status.created_at).toLocaleTimeString()})`;
19     statusList.appendChild(statusElement);
20   });
21 });
22
```

Github Link

[Boigantso/Websocket-RealtimeData \(github.com\)](https://github.com/Boigantso/Websocket-RealtimeData)

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.