

Laporan
Pemrograman Berorientasi Objek dan Pemrograman Web



Mini Project
“CRUD, Rest API dan WebSocket”

Dosen Pengampu:
Wildan Afrizal Arifin, S.Pd., M.Kom

Disusun Oleh:
Nelita Maharani (2307052)
Rakhshanda Shaina Ahava Hidayat (2307262)

**PROGRAM STUDI SISTEM INFORMASI KELAUTAN
UNIVERSITAS PENDIDIKAN INDONESIA**

2024

I. PENDAHULUAN

Dalam era digital yang semakin berkembang, kebutuhan akan komunikasi real-time dalam aplikasi web menjadi semakin penting. Aplikasi modern seperti chat, game online, dan sistem monitoring membutuhkan pertukaran data yang cepat dan efisien antara client dan server. Protokol HTTP tradisional, yang berbasis request-response, memiliki keterbatasan dalam memenuhi kebutuhan ini karena sifatnya yang unidirectional dan connectionless.

WebSocket hadir sebagai solusi untuk mengatasi keterbatasan tersebut dengan menyediakan protokol komunikasi yang mendukung pertukaran data dua arah (full-duplex) melalui koneksi tunggal yang persisten. Teknologi ini memungkinkan server untuk mengirim data ke client tanpa perlu menunggu request, sehingga sangat cocok untuk aplikasi yang membutuhkan pembaruan data secara real-time.

Dalam praktikum ini, kita akan mempelajari dan mengimplementasikan protokol WebSocket untuk membangun aplikasi web yang mendukung komunikasi real-time. Fokus utama praktikum adalah memahami cara kerja WebSocket, melakukan setup koneksi, menangani events, dan mengimplementasikan pertukaran data antara client dan server.

II. ALAT DAN BAHAN

1. Laptop/Komputer
2. Aplikasi Visual Studio Code
3. XAMPP
4. Database mysql

III. LANGKAH KERJA

Berikut adalah langkah kerja websocket, node js dan integrasi dengan mysql:

1. Buka terminal dan ketik `npm init -y` dan `npm install ws mysql`.
2. Membuat file `server.js` berikut,

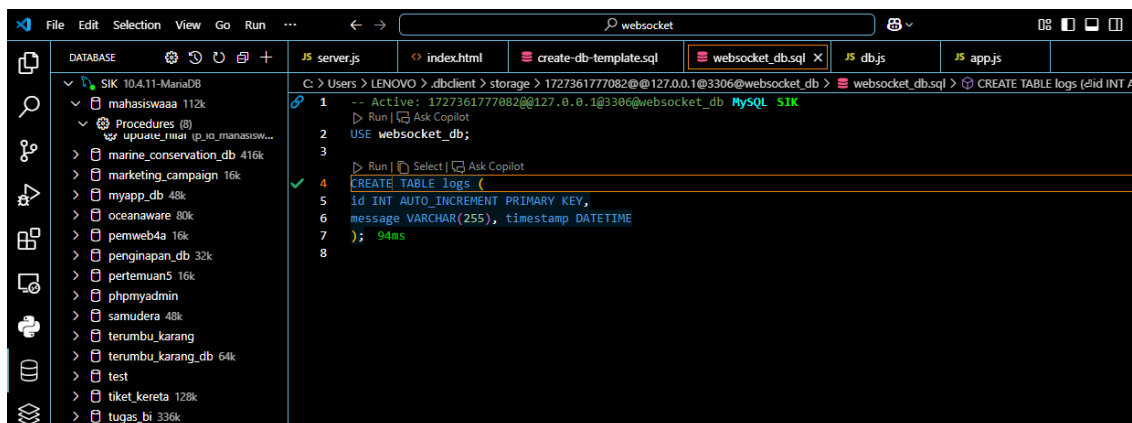
```
1. const WebSocket = require('ws');
2. const wss = new WebSocket.Server({ port: 3000 });
3.
4. wss.on('connection', (ws) => {
5.   console.log('Client connected');
6. }
```

```

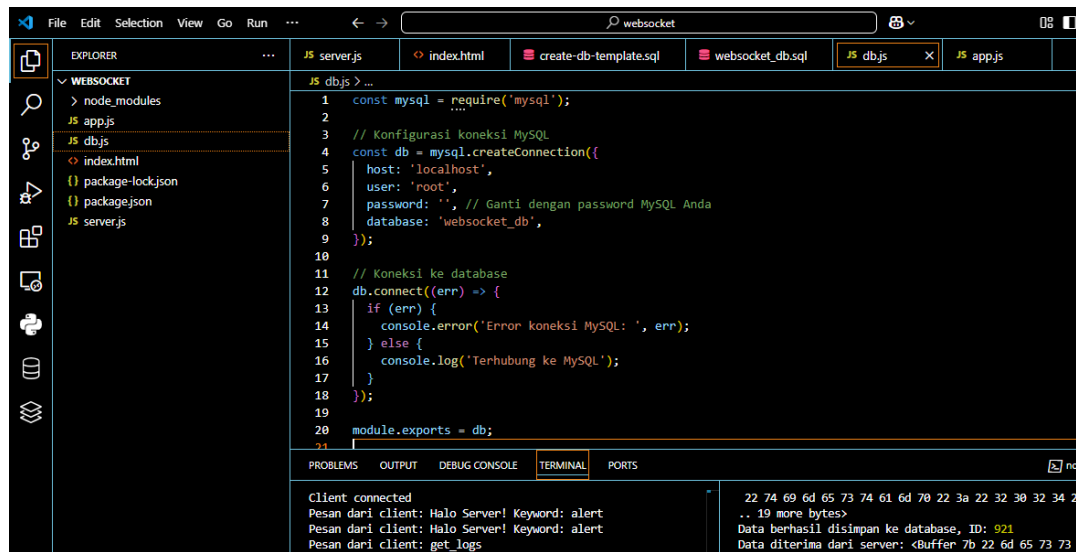
7. // Kirim pesan ke klien setiap 3 detik
8. setInterval(() => {
9.     ws.send(JSON.stringify({
10.         message: 'Data dari server',
11.         timestamp: new Date()
12.     }));
13. }, 3000);
14.
15. // Terima pesan dari klien
16. ws.on('message', (message) => {
17.     console.log(`Pesan dari client: ${message}`);
18.
19.     // Deteksi keyword "alert"
20.     if (message.includes('alert')) {
21.         ws.send(JSON.stringify({ notification: 'Keyword "alert"
22.             terdeteksi!' }));
23.     }
24. });
25.
26. // Ketika klien terputus
27. ws.on('close', () => {
28.     console.log('Client disconnected');
29. });
30.
31. console.log('WebSocket server berjalan di ws://localhost:3000');
32.

```

3. Buat database dan table di mysql,



4. Buat file db.js berikut,



The screenshot shows a VS Code editor with a project named 'websocket'. The Explorer sidebar on the left shows the file structure: 'node_modules', 'app.js', 'db.js', 'index.html', 'package-lock.json', 'package.json', 'server.js', and 'websocket_db.sql'. The 'db.js' file is open in the editor, showing the following code:

```
1 const mysql = require('mysql');
2
3 // Konfigurasi koneksi MySQL
4 const db = mysql.createConnection({
5   host: 'localhost',
6   user: 'root',
7   password: '', // Ganti dengan password MySQL Anda
8   database: 'websocket_db',
9 });
10
11 // Koneksi ke database
12 db.connect((err) => {
13   if (err) {
14     console.error('Error koneksi MySQL: ', err);
15   } else {
16     console.log('Terhubung ke MySQL');
17   }
18 });
19
20 module.exports = db;
```

The bottom panel shows the 'TERMINAL' tab with the following output:

```
Client connected
Pesan dari client: Halo Server! Keyword: alert
Pesan dari client: Halo Server! Keyword: alert
Pesan dari client: get_logs
```

On the right side of the terminal, there is a hex dump of the received data:

```
22 74 69 64 65 73 74 61 64 70 22 3a 22 32 30 32 34 20
.. 19 more bytes>
Data berhasil disimpan ke database, ID: 921
Data diterima dari server: <Buffer 7b 22 64 65 73 73
```

5. Implementasi websocket dengan membuat file app.js

```
1. const WebSocket = require('ws');
2. const ws = new WebSocket('ws://localhost:3000');
3.
4. // Koneksi ke WebSocket Server
5. ws.on('open', () => {
6.   console.log('Terhubung ke WebSocket server');
7.
8.   // Kirim pesan dengan keyword "alert"
9.   ws.send('Halo Server! Keyword: alert');
10.
11.   // Minta data logs
12.   ws.send('get_logs');
13.});
14.
15.// Terima pesan dari server
16.ws.on('message', (data) => {
17.  const response = JSON.parse(data);
18.
19.  // Tangani respons dari server
20.  if (response.logs) {
21.    console.log('Data logs dari server:', response.logs);
22.  } else if (response.notification) {
23.    console.log('Notifikasi dari server:', response.notification);
24.  } else {
25.    console.log('Pesan lain dari server:', response);
26.  }
```

```

27.});
28.
29.// Tangani error
30.ws.on('error', (err) => {
31.  console.error('Error WebSocket:', err);
32.});
33.
34.// Tangani koneksi ditutup
35.ws.on('close', () => {
36.  console.log('Koneksi WebSocket ditutup');
37.});
38.

```

6. Setelah server berhasil dijalankan, tambahkan fitur untuk membaca data dari mysql dan mengembalikannya ke websocket server

The screenshot shows a terminal window with the following content:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Client connected
Pesan dari client: Halo Server! Keyword: alert
Client connected
Pesan dari client: Halo Server! Keyword: alert
Pesan dari client: Halo Server! Keyword: alert
Pesan dari client: get_logs
Client disconnected
Client connected
22 3a 22 44 61 74 61 20 64 61 72 69 20 73 65 72
22 74 69 6d 65 73 74 61 6d 70 22 3a 22 32 30 32
.. 19 more bytes>
Data berhasil disimpan ke database, ID: 1044
Data diterima dari server: <Buffer 7b 22 6d 65 7
22 3a 22 44 61 74 61 20 64 61 72 69 20 73 65 72
22 74 69 6d 65 73 74 61 6d 70 22 3a 22 32 30 32
.. 19 more bytes>

```

7. Kembangkan notifikasi real-time, buat frontend sederhana menggunakan html dan javascript yang berkomunikasi dengan server websocket

```

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.   <meta charset="UTF-8">
5.   <meta name="viewport" content="width=device-width, initial-
scale=1.0">
6.   <title>WebSocket Client</title>
7.   <link href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0-beta3/css/all.min.css" rel="stylesheet">
8.   <style>
9.     body {
10.      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
11.      margin: 0;
12.      padding: 0;
13.      background-color: #e0e5ec;
14.      color: #333;
15.    }
16.
17.    .container {

```

```
18.     max-width: 900px;
19.     margin: 50px auto;
20.     padding: 40px;
21.     background-color: white;
22.     border-radius: 15px;
23.     box-shadow: 0 10px 30px rgba(0, 0, 0, 0.1);
24.     transition: transform 0.3s ease;
25. }
26.
27. .container:hover {
28.     transform: scale(1.02);
29. }
30.
31. h1 {
32.     color: #4A90E2;
33.     font-size: 2.8em;
34.     margin-bottom: 25px;
35.     text-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
36. }
37.
38. .status {
39.     font-size: 1.2em;
40.     padding: 12px;
41.     color: #28a745;
42.     background-color: #d4edda;
43.     border-radius: 5px;
44.     margin-bottom: 20px;
45.     transition: background-color 0.3s ease;
46.     font-weight: bold;
47. }
48.
49. .status.error {
50.     color: #dc3545;
51.     background-color: #f8d7da;
52. }
53.
54. .status.closed {
55.     color: #ffc107;
56.     background-color: #fff3cd;
57. }
58.
59. .buttons {
60.     display: flex;
61.     justify-content: space-around;
62.     gap: 30px;
```

```
63.     margin-bottom: 30px;
64. }
65.
66. button {
67.     padding: 15px 30px;
68.     font-size: 1.2em;
69.     color: white;
70.     background: linear-gradient(135deg, #4A90E2, #357ABD);
71.     border: none;
72.     border-radius: 8px;
73.     cursor: pointer;
74.     transition: background 0.3s ease, transform 0.3s ease;
75.     display: flex;
76.     align-items: center;
77. }
78.
79. button:hover {
80.     background: linear-gradient(135deg, #357ABD, #4A90E2);
81.     transform: scale(1.05);
82. }
83.
84. button i {
85.     margin-right: 12px;
86. }
87.
88. #output {
89.     margin-top: 20px;
90.     border-radius: 10px;
91.     padding: 20px;
92.     background-color: #fafafa;
93.     height: 350px;
94.     overflow-y: auto;
95.     border: 1px solid #ddd;
96.     font-family: 'Courier New', monospace;
97.     font-size: 1.1em;
98.     color: #555;
99.     white-space: pre-wrap;
100.     box-shadow: inset 0 0 12px rgba(0, 0, 0, 0.1);
101.     background: #f1f1f1;
102. }
103.
104. #output pre {
105.     margin: 0;
106. }
107.
```

```

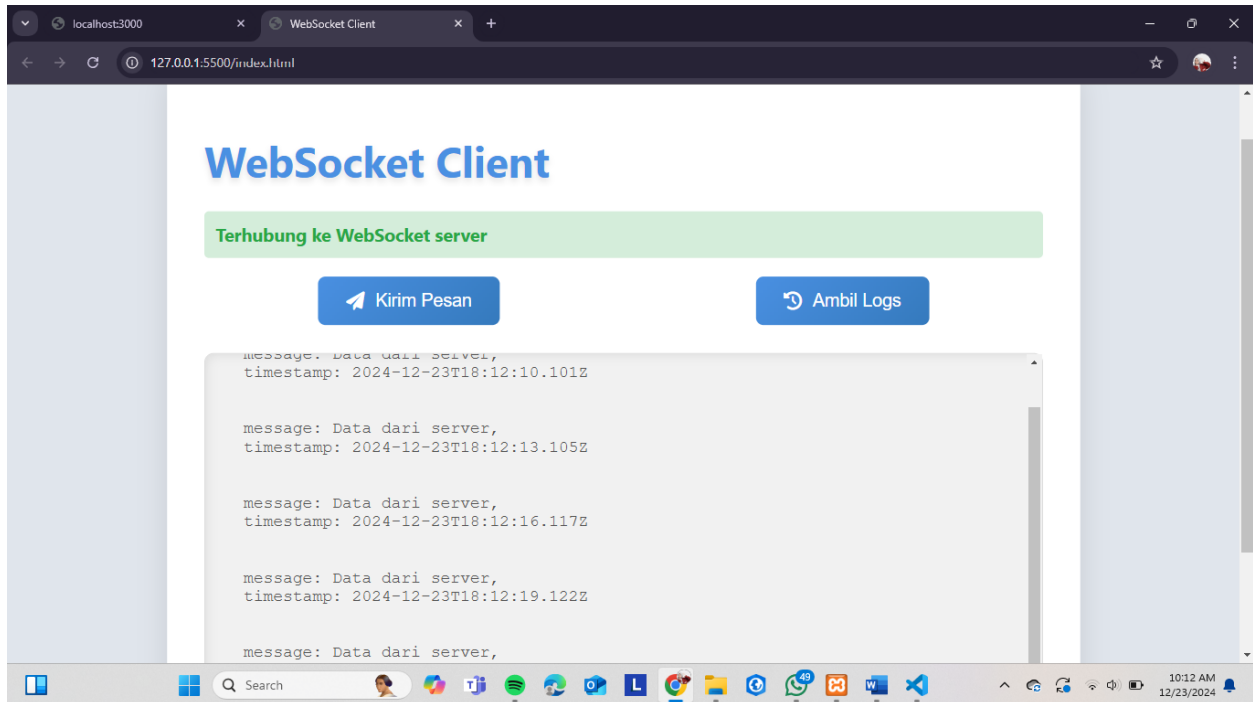
108.     @media screen and (max-width: 768px) {
109.         .container {
110.             padding: 20px;
111.         }
112.
113.         button {
114.             width: 100%;
115.             padding: 18px;
116.             font-size: 1.1em;
117.         }
118.
119.         #output {
120.             height: 250px;
121.         }
122.     }
123. </style>
124. </head>
125. <body>
126.     <div class="container">
127.         <h1>WebSocket Client</h1>
128.         <div class="status" id="status">Mencoba
menghubungkan...</div>
129.         <div class="buttons">
130.             <button id="sendMessage"><i class="fas fa-paper-
plane"></i>Kirim Pesan</button>
131.             <button id="getLogs"><i class="fas fa-history"></i>Ambil
Logs</button>
132.         </div>
133.         <div id="output"></div>
134.     </div>
135.
136.     <script>
137.         const ws = new WebSocket('ws://localhost:3000');
138.         const output = document.getElementById('output');
139.         const status = document.getElementById('status');
140.
141.         ws.onopen = () => {
142.             status.textContent = 'Terhubung ke WebSocket server';
143.             status.classList.remove('error', 'closed');
144.             status.classList.add('status');
145.         };
146.
147.         ws.onmessage = (event) => {
148.             const data = JSON.parse(event.data);
149.

```



```
150.
151.         const formattedData = JSON.stringify(data, null,
152.           2).replace(/[\{\}]/g, ''); // Hapus tanda kurung dan kutip
153.         output.textContent += formattedData + '\n';
154.         output.scrollTop = output.scrollHeight;
155.     };
156.     ws.onerror = (err) => {
157.         status.textContent = `Error: ${err.message}`;
158.         status.classList.remove('status', 'closed');
159.         status.classList.add('error');
160.     };
161.
162.     ws.onclose = () => {
163.         status.textContent = 'Koneksi WebSocket ditutup';
164.         status.classList.remove('status', 'error');
165.         status.classList.add('closed');
166.     };
167.
168.     document.getElementById('sendMessage').addEventListener('c
169.       lick', () => {
170.         ws.send('Halo Server! Keyword: alert');
171.       });
172.     document.getElementById('getLogs').addEventListener('click
173.       ', () => {
174.         ws.send('get_logs');
175.       });
176.     </script>
177.   </body>
178. </html>
```

8. Berikut merupakan tampilan websocket ketika sudah diimplementasikan dengan html dan javascript,



Akan muncul notifikasi real-time pada web tersebut

IV. KESIMPULAN

Berdasarkan praktikum implementasi WebSocket yang telah dilakukan, dapat disimpulkan bahwa protokol WebSocket merupakan solusi yang sangat efektif untuk komunikasi real-time dalam pengembangan aplikasi web modern. Melalui praktikum ini, telah dibuktikan bahwa WebSocket mampu memberikan performa yang lebih baik dibandingkan HTTP tradisional, terutama dalam hal kemampuannya untuk memfasilitasi komunikasi dua arah secara instan antara client dan server melalui koneksi tunggal yang persisten.

Implementasi WebSocket menggunakan Node.js di sisi server dan JavaScript di sisi client menunjukkan proses yang relatif straightforward dan efisien. Pendekatan event-driven programming yang diterapkan dalam penanganan berbagai events seperti 'onopen', 'onmessage', dan 'onclose' memungkinkan pengembangan aplikasi yang lebih terstruktur dan mudah dimaintain. Keunggulan WebSocket dalam menangani pertukaran data real-time terbukti sangat signifikan, mengingat protokol ini hanya memerlukan satu koneksi

TCP yang dapat digunakan secara terus-menerus, sehingga mengurangi overhead yang umumnya terjadi pada metode polling tradisional.

Link github:

<https://github.com/maharaninelita/websocket.git>