Socket.IO

Akash Pundir

System Programming –I

School of Computer Science and Engineering

Problems faced by traditional HTTP requests

- **Unidirectional communication**: HTTP is inherently a request-response protocol, where the client sends a request to the server, and the server responds. This model doesn't support real-time updates initiated by the server without the client explicitly requesting them.

- **Polling**: One approach to achieve real-time updates with HTTP is through polling, where the client repeatedly sends requests to the server at regular intervals to check for updates. This method can be inefficient, leading to unnecessary network traffic and increased server load, especially if updates are infrequent.

- **Latency**: Even with polling, there's often a delay between when an event occurs on the server and when the client receives the update. This latency can degrade the user experience, especially in applications where real-time interaction is critical.

- **Scalability**: Traditional HTTP communication can be challenging to scale for real-time applications with a large number of concurrent users. Each HTTP request creates a new connection to the server, which can strain server resources and limit scalability.

What is **Socket.IO**?

Socket.IO is a library that enables low-latency, **bidirectional** and event-based communication between a client and a server.

Socket.IO enables bidirectional communication between clients and servers. Unlike traditional HTTP requests, where communication is initiated by the client, Socket.IO allows servers to send updates to clients without waiting for a request. This bidirectional capability is essential for real-time applications like chat rooms, online gaming, and collaborative editing

Install Socket.IO

```
npm install socket.io
```

Importing Required Modules

```javascript
const express = require('express');
const socketIO = require('socket.io');
const path = require('path');
```

Creating Express App and Starting Server

```
const app = express();
const server = app.listen(3000, () => {
    console.log('Server is running on port 3000');
});
```

The server variable holds the server instance returned by calling app.listen()

```
const io = socketIO(server);

// Serve static files from the 'public' directory
app.use(express.static(path.join(__dirname,
'public')));
```

It creates a Socket.IO instance attached to the Express server.

and serves static files (like HTML, CSS, JavaScript) located in the 'public' directory.

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

    // Listen for 'chat message' events from clients
    socket.on('chat message', msg => {
        // Broadcast the message to all clients in the same
room
        io.emit('chat message', msg);
    });

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

- It listens for a 'connection' event when a client connects via Socket.IO.
- Upon connection, it logs a message to the console.
- It sets up event listeners for the 'chat message' event from clients.
- When it receives a 'chat message' event from a client, it broadcasts the message to all connected clients.
- It also listens for a 'disconnect' event, which triggers when a client disconnects, and logs a message to the console.

Now, let’s make a basic client-side chat interface

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Socket.IO Chat</title>
</head>
<body>
    <ul id="messages"></ul>
    <form id="form" action="">
        <input id="input"/><button>Send</button>
    </form>
```

```
<script src="/socket.io/socket.io.js"></script>
    <script>
        const socket = io();

        const form = document.getElementById('form');
        const input = document.getElementById('input’);
```

Initial script tag includes the Socket.IO client library, allowing the client-side JavaScript to connect to the Socket.IO server

a Socket.IO client instance is initialized and assigned to the socket constant

```
form.addEventListener('submit', e => {
            e.preventDefault();
            if (input.value) {
                // Emit 'chat message' event with the input value
                socket.emit('chat message', input.value);
                input.value = '';
            }
        });
```

This adds a submit event listener to the form. When the form is submitted (e.g., by clicking the send button or pressing Enter), this function executes.

e.preventDefault();: This prevents the default form submission behavior, which would cause the page to reload

```
// Listen for 'chat message' events from the server
        socket.on('chat message', msg => {
            const item = document.createElement('li');
            item.textContent = msg;
            document.getElementById('messages').appendChild(item);
        });
    </script>
</body>
</html>
```

This sets up a listener for 'chat message' events from the server. When the client receives a message from the server, this function executes

Test your Knowledge

Write a simple Socket.IO chat application where multiple users can connect and exchange messages in real-time. The application should have the following features:

- **Display a chat interface where users can see all the messages sent by other users.**
- **Allow users to enter their name when they join the chat.**
- **Display the names of users along with their messages.**
- **Implement a feature to notify all users when a new user joins or leaves the chat.**
- **Ensure that the chat interface scrolls automatically to show the latest messages.**

```javascript
const express = require('express');
const socketIO = require('socket.io');
const path=require('path');

const app = express();
const server = app.listen(3000,()=>{
    console.log("Server started on port 3000");
});

const io = socketIO(server);

app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'test.html'));
});
```

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

    // Listen for 'join' event when a new user joins
    socket.on('join', username => {
        socket.username = username;
        io.emit('chat message', { type: 'notification', message: `${username} has joined the chat` });
    });

    // Listen for 'chat message' events from clients
    socket.on('chat message', msg => {
        io.emit('chat message', { type: 'message', username: socket.username, message: msg });
    });

    // Listen for 'disconnect' event when a user leaves
    socket.on('disconnect', () => {
        console.log('A user disconnected');
        if (socket.username) {
            io.emit('chat message', { type: 'notification', message: `${socket.username} has left the chat`
});
        }
    });
});
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Socket.IO Chat</title>
</head>
<body>
    <ul id="messages"></ul>
    <form id="form" action="">
        <input id="input" autocomplete="off"
/><button>Send</button>
    </form>
```

```
<script src="/socket.io/socket.io.js"></script>
    <script>
        const socket = io();

        const form = document.getElementById('form');
        const input = document.getElementById('input');
        const messages = document.getElementById('messages');

        form.addEventListener('submit', e => {
            e.preventDefault();
            if (input.value) {
                socket.emit('chat message', input.value);
                input.value = '';
            }
        });
```

```
// Prompt the user to enter their name
        const username = prompt('Please enter your name:');
        socket.emit('join', username);

        socket.on('chat message', data => {
            const item = document.createElement('li');
            if (data.type === 'notification') {
                item.textContent = data.message;
                item.style.fontWeight = 'bold';
            } else {
                item.textContent = `${data.username}: ${data.message}`;
            }
            messages.appendChild(item);
            // Scroll to the bottom of the chat window
            messages.scrollTop = messages.scrollHeight;
        });
    </script>
</body>
</html>
```

THANKS SOCKET.IO