WEBSOCKETS

How to “Real Time” on the web

SWIPE

What is it?

Websockets are a communication protocol that enables real-time, two-way communication between a client (like a web browser) and a server over a single, long-lived connection. They are often used in web applications where real-time updates, interactive features, or instant messaging are required.

Handshake

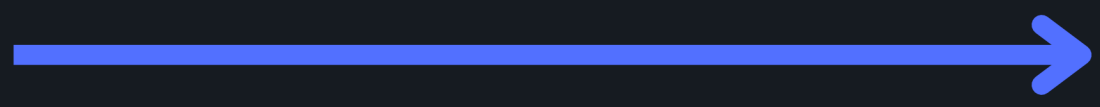
The process starts with an initial HTTP handshake between the client and the server, just like when you request a web page. This handshake is used to establish the WebSocket connection. The client sends a WebSocket handshake request, and if the server supports WebSockets, it responds with a WebSocket handshake response, upgrading the connection from HTTP to WebSocket.

Persistent Connection

Unlike traditional HTTP, which is stateless and involves opening a new connection for each request, WebSockets maintain a persistent connection. This means that once the connection is established, both the client and server can send data to each other at any time without the overhead of repeatedly establishing new connections.

Bi-Directional Communication

WebSockets allow both the client and the server to send data to each other whenever they want. This is different from HTTP, where the client initiates requests and the server responds. With WebSockets, data can flow in both directions, enabling real-time updates.



WebSocket Messages

Data sent over a WebSocket connection is encapsulated into messages. Messages can be text or binary data. Each message can be sent as a whole or in smaller chunks for large payloads. The WebSocket protocol ensures that messages are properly framed and delivered reliably.

Event Driven

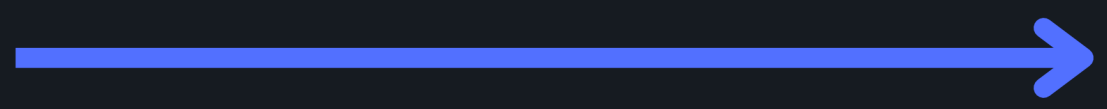
WebSockets are event-driven. The client and server can define event handlers to process incoming messages or respond to specific events. This makes it easy to create interactive applications that react in real-time to user actions or data changes on the server.

Closing the connection

Either the client or server can initiate the closure of the WebSocket connection when it's no longer needed. This is done through a closing handshake to ensure a graceful termination of the connection.

Pros

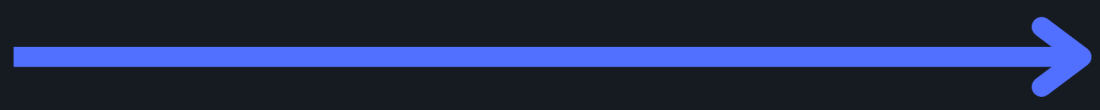
Either the client or server can initiate the closure of the WebSocket connection when it's no longer needed. This is done through a closing handshake to ensure a graceful termination of the connection.



Server Setup Example (Socket.io)

We create a basic Express.js web server and an HTTP server.
Socket.IO is integrated with the Express server, allowing WebSocket communication.
We serve a simple HTML file (index.html) that will be used to interact with the WebSocket.

```
const http = require('http');
const express = require('express');
const socketIo = require('socket.io');
const app = express();
const server = http.createServer(app);const io = socketIo(server);
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});
io.on('connection', (socket) => {
  console.log('A user connected');
  socket.on('chat message', (msg) => {
    console.log(`Message: ${msg}`);
    io.emit('chat message', msg);
  });
  socket.on('disconnect', () => {
    console.log('A user disconnected');
  });
});
server.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
});
```

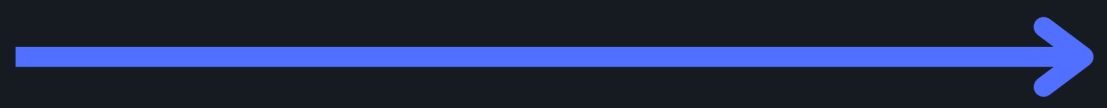


Client Setup Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Socket.IO Chat Example</title>
  <script src="/socket.io/socket.io.js"></script>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
  <ul id="messages"></ul>
  <input id="message-input" autocomplete="off" /><button onclick="sendMessage()">Send</b
  <script>
    // Connect to the WebSocket server
    const socket = io();

    // Listen for incoming messages
    socket.on('chat message', (msg) => {
      const li = document.createElement('li');
      li.textContent = msg;
      document.getElementById('messages').appendChild(li);
    });

    // Function to send a message
    function sendMessage() {
      const message = document.getElementById('message-input').value;
      socket.emit('chat message', message);
      document.getElementById('message-input').value = '';
    }
  </script>
</body>
</html>
```



Support

```
# Python
more_content = like and follow and comment

# Javascript
const moreContent = like && follow && comment;

# Java
boolean moreContent = like && follow && comment;

# PHP
$moreContent = $like && $follow && $comment;

# C++
bool moreContent = like && follow && comment;
```

Mario Ruci
https://marioruci.com
http://havenojob.me

More to come...