**Was ist ein Web Socket**

WebSockets sind ein Kommunikationsprotokoll, das eine dauerhafte, bidirektionale Verbindung zwischen Client und Server ermöglicht. Die Verbindung bleibt bestehen und wird nicht wie bei http für jeden request neu aufgebaut und dann wider geschlossen

**Vorteile**

* Netzwerknutzung ist effizienter weil die Verbindung für mehrere Requests benutzt werden kann und man dann auch nicht dauernd einen neuen overhead benötigt der Bandbreit und Verarbeitungszeit einnimmt
* Die dauerhafte Verbindung ermöglicht Echtzeit Kommunikation, weil Web Sockets dauerhaft erlauben (solange die Verbindung läuft) das der Server oder Client Daten schicken. Das ist gut für Chat oder Spiele.

**Wie funktionieren Web Sockets**

1. Der Client schickt eine Anfrage an den Server
2. Ein sogenannter handshake wird gemacht. Dieser handshake legt den Header fest für die gesamte Verbindung hier ein Beispiel wie so ein Header aussieht von HTTP und Web Sockets

HTTP Header

GET /data HTTP/1.1

Host: example.com

User-Agent: Mozilla/5.0

Accept: application/json

Connection: keep-alive

Dieser Header wir bei jeder HTTP anfrage gebraucht dazu werden noch kleine Daten Mengen zusätzlich gebraucht meistens bekommt man ein einfaches OK zurück

Web Socket Header

GET /chat HTTP/1.1// Das ist unsere Get Anfrage

Host: example.com //Ziel Server

Upgrade: websocket//Das fordert den Server auf das es die HTTP Anfrage zu einen Websocket umwandelt

Connection: Upgrade//Signalisiert erneut das es auf Web Socket umgestellt werden soll

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9YZrdMw==//Dieser Code stellt sicher das die Anfrage nicht einfach von irgendeinen HTTP Request kommt weil das immer ein zufällig generirter Web Socket key vom Client ist

Sec-WebSocket-Version: 13//Protokoll Version

Es muss umgewandelt werden weil Web Sockets eine Erweiterung eines HTTP Requests sind

Ab den Punkt Upgrade wird also unser Handshake gemacht der festlegt das es ein Web Socket Protokoll ist und ab dann wird die Verbindung auch gehalten

1. Es werden Bidirectional Daten hin und her geschickt also der Client kann während der Verbindung dauerhaft Anfragen stellen und der Server kann dauerhaft Antworten schicken

✅ **3️⃣. Serverprüfung**

* Der Server prüft, ob er WebSockets unterstützt und ob der Pfad (z. B. /scoreboard) passt.
* Wenn ja, antwortet er mit:
* makefile
* CopyEdit
* HTTP/1.1 101 Switching Protocols
* Upgrade: websocket
* Connection: Upgrade
* Sec-WebSocket-Accept: <hashed-key>
* **Code 101** heißt: Ich schalte jetzt auf WebSocket um.

✅ **4️⃣. Verbindung erfolgreich**

* Nach der 101-Antwort wird die HTTP-Verbindung „gekappt“, und stattdessen läuft nun ein **dauerhafter WebSocket-Datenstrom** über dieselbe TCP-Verbindung.
* Jetzt wird das onopenEvent ausgelöst.

**❌ Wann wird die Verbindung beim Handshake gebrochen?**

Der Handshake bricht ab, wenn:

* Der Server keine WebSockets akzeptiert (z. B. weil der Pfad falsch ist).
* Der Server eine andere Antwort als **101** sendet (z. B. 400, 404, 500).
* Ein Load Balancer, Proxy oder Firewall die UpgradeHeaders blockiert.
* Bei wss://: Das TLS-Zertifikat ungültig ist oder der Handshake auf der SSL-Ebene scheitert.
* Netzwerkfehler oder Timeout auftreten (z. B. Server ist nicht erreichbar).

In diesen Fällen:

* Kein onopen.
* Stattdessen wird **onerror** aufgerufen.
* Danach wird automatisch die Verbindung geschlossen, und onclose läuft.

**Wie kann ich sowas aber erstellen?**

Client Seite

<!DOCTYPE html>

<script>

// Verbindung zum Server aufbauen

ws = new WebSocket("ws://127.0.0.1/scoreboard") // Local server

// ws = new WebSocket("wss://game.example.com/scoreboard") // Remote server

ws.onopen = () => {

console.log("Connection opened")

ws.send("Hi server, please send me the score of yesterday's game")

}

ws.onmessage = (event) => {

console.log("Data received", event.data)

ws.close() // We got the score so we don't need the connection anymore

}

ws.onclose = (event) => {

console.log("Connection closed", event.code, event.reason, event.wasClean)

}

ws.onerror = () => {

console.log("Connection closed due to error")

}

</script>

* Bei ws:// → unverschlüsselt (Port 80).
* Bei wss:// → TLS/SSL-verschlüsselt (Port 443).