JavaScript - FetchAPI, WebSocket

Fetch API

- Funkce v prohlížeči pro HTTP požadavky na server / API.
- Používá se pro získání / odesílání dat
- fetch() vrací Promise
- **Základní GET:**

```
fetch(url)
    .then(res => res.json())
    .then(data => console.log(data))
    .catch(err => console.log(err));
```

  - res.json() převede odpověď na JS objekt
  - .catch() chytá hlavně chyby sítě
- **GET přes async/await:**

```
async function load() {
    try {
      const res = await fetch(url);
      const data = await res.json();
      console.log(data);
    } catch (err) {
      console.log(err);
    }
}
```

  - await čeká na Promise
  - try/catch je pro chyby

- **POST:**

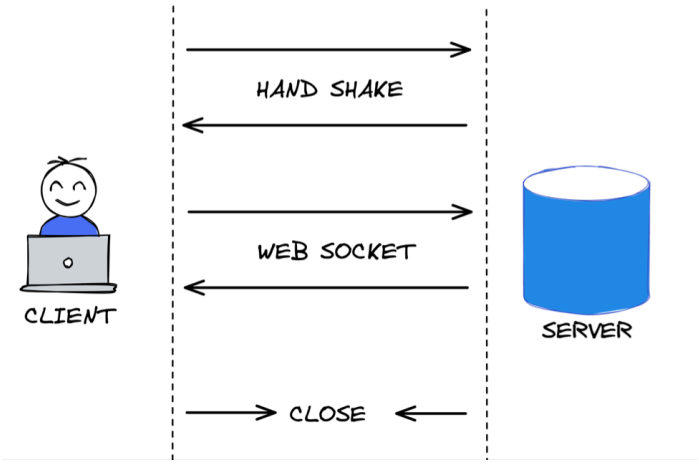
```
await fetch(url, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ name: "Anna", age: 18 })
});
```

  - method určuje typ požadavku
  - headers říká, že posílám JSON
  - body must but string -> JSON.stringify()

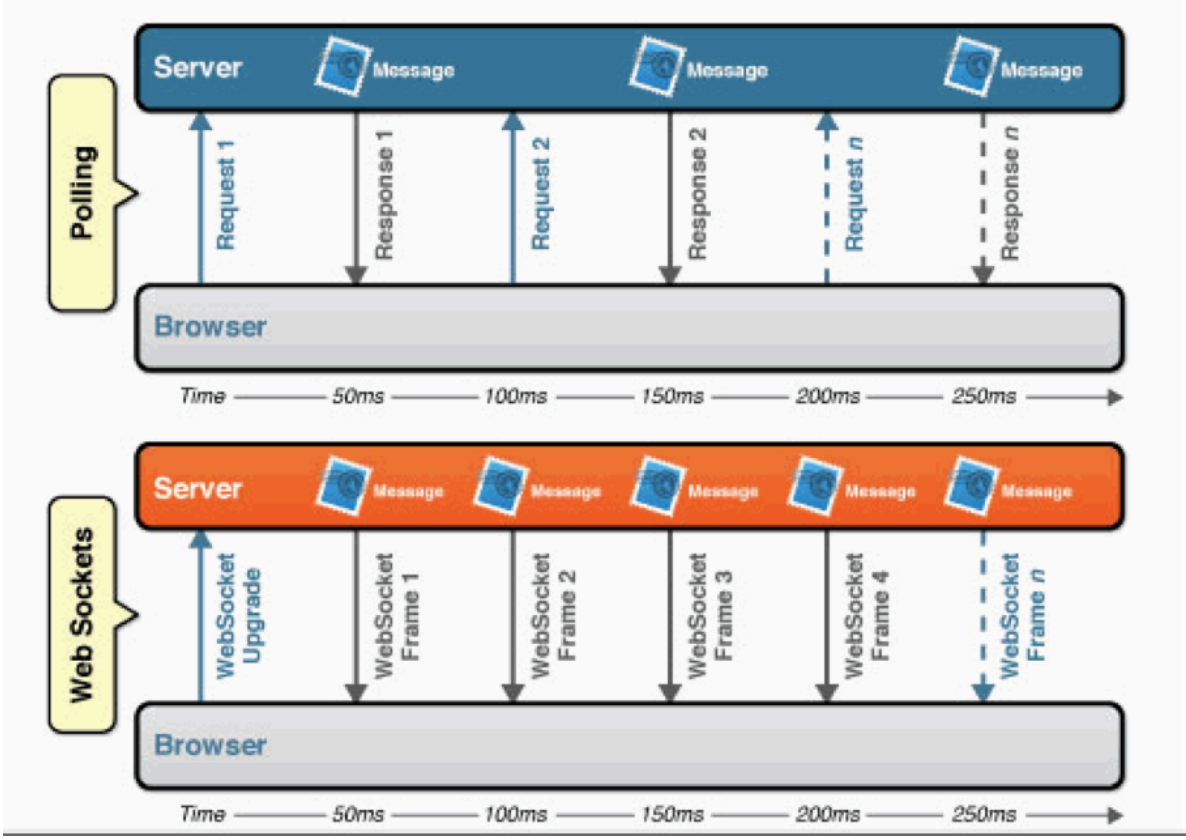
- **Přehled HTTP metod:**
  - **GET**
  - **POST**
  - **PUT/PATCH**
  - **DELETEResponse**
  - status
  - ok
  - Metody pro tělo odpovědi:
    - json()
    - text()

Websocket

- Websocket je protokol, který umožňuje dvousměrnou komunikaci mezi webovým klientem a serverem přes jedno, dlouho otevřené spojení
- Stavový protokol
- **Vlastnosti:**
  - Dvousměrná komunikace
  - Nízká latence
  - Jedno spojení pro celou komunikaci
- **Použití:**
  - Real-time aplikace jako chaty, hry, finanční tickery, atd.
- **Porty:**
  - Využívá porty 80 a 443 pro spojení, což usnadňuje průchod firewally
- **Websocket Handshake:**
  - Komunikace začíná jako HTTP/1.1 požadavek s hlavičkou **Upgrade: websocket**, kterým klient požádá server o přechod z HTTP na WebSocket
  - Server odpoví kódem **101 Switching Protocols** a potřebnými hlavičkami (např. **Sec-WebSocket-Accept**), čímž potvrdí navázání WebSocket spojení
  - Po úspěšném handshaku se mezi klientem a serverem udržuje jediné TCP spojení a data se dále vyměňují prostřednictvím WebSocket rámců



- **Rámce ve WebSocketu:**
  - **Definice:**
    - Rámec je základní jednotka datové komunikace ve WebSocketu
  - **Struktura:**
    - Skládá se z hlavičky a nákladu (payload). Hlavička obsahuje metadata jako délku nákladu a typ zprávy
  - **Typy rámců:**
    - Textové, binární, uzavírací, ping a pong
  - **Fragmentace:**
    - Velká zpráva může být rozdělena do více rámců (fragmentů)
  - **Efektivita:**
    - Rámcování umožňuje efektivní a rychlý přenos dat po streamu
  - **Bezpečnost:**
    - Každý rámec může být maskován pro zajištění bezpečnosti



- **Ping-pong:**
  - **Definice:**
    - Jsou speciální krátké rámce používané k udržení spojení a monitorování latence
  - **Ping:**
    - Server nebo klient může odeslat ping rámec k zjištění odezvy
  - **Pong:**
    - Odpověď na ping - potvrzuje, že spojení je stále aktivní
  - **Automatická odpověď:**
    - WebSocket API standardně reaguje na přijatý ping automaticky odesláním pong
  - **Využití:**
    - Udržuje spojení otevřené a pomáhá detekovat neaktivní spojení
  - **Latence:**
    - Doba mezi odesláním ping a přijetím odpovědi pong měří síťovou latenci
  - **Bezpečnost:**
    - Ping/pong zprávy nemají vliv na maskování ani šifrování dat
- **Maskování:**
  - **Co to je?:**
    - Proces, při němž jsou data v rámci zkombinována s náhodně generovaným 32bitovým klíčem
    - Povinné pro všechny rámce odesílané klientem směrem k serveru
    - Volitelné pro rámce odesílané serverem ke klientovi

- **Jak funguje?:**
  - Přečte se maskovací klíč z hlavičky rámce
  - Pro každý bajt datového pole se provede operace XOR s odpovídajícím bajtem maskovacího klíče
  - Maskovací klíč se aplikuje cyklicky; opakuje se, pokud je datové pole delší než 4 bajty
- **Důsledky:**
  - Zvyšuje bezpečnost (eliminuje riziko zneužití mezilehlými proxy)
  - Má minimální dopad na výkon (XOR operace je velmi rychlá)

- **Výhody a nevýhody:**

Výhody:

- **Nízká latence**: řádově jednotky ms
- **Plně duplexní provoz**: data mohou proudit oběma směry současně
- **Nízká režie**: odpadá opakované navazování spojení a posílání HTTP hlaviček

Nevýhody:

- **Kompatibilita**: některé proxy a firewally (zejm. starší) nepodporují WebSocket
- **Zátěž na server**: nutnost udržovat velké množství otevřených spojení

- **Využití Websocketu:**
  - Webové chatovací aplikace pro komunikaci v reálném čase
  - **Streamování dat**: finanční trhy (tickery cen), sportovní výsledky aj. s okamžitými aktualizacemi
  - **Online hry**: synchronizace stavu hry mezi klienty v reálném čase
  - Spolupráce v reálním čase: společná editace dokumentů nebo tabulí
- **Websocket a IoT:**
  - Ovládání IoT zařízení přes web
  - Streamování dat ze senzorů
  - MQTT přes WebSocket
-

- **Přijetí WebSocketu na trhu:**
  - Běžná součást webových aplikací
  - Růst popularity
  - IoT boom
- **Příklad:**

klient - html

```html
<!DOCTYPE html>
<html>
<head>
  <title>WebSocket Klient</title>
</head>
<body>
  <!-- Tlačítko pro odeslání zprávy -->
  <button onclick="sendMessage()">Odeslat zpravu</button>

  </body>
</html>
```

klient - připojení

```javascript
// Vytvoření nového WebSocket spojení
const ws = new WebSocket('ws://localhost:8080');
// Událost při úspěšném připojení
ws.addEventListener('open', () => {
  console.log('Pripojeno k serveru');
});
```

klient - přijetí zprávy

```javascript
// Událost při přijetí zprávy od serveru
ws.addEventListener('message', (event) => {
  console.log(`Zprava od serveru: ${event.data}`);
});
```

Klient - odeslání zprávy

```javascript
// Funkce pro odeslání zprávy
function sendMessage() {
  ws.send('Ahoj, server!');
}
```