Protokół WebSocket
Node.js

Technologie Internetowe

Co zastosować, gdy chcemy mieć aktualne dane korzystając z protokołu HTTP?

1. HTTP Polling to technika komunikacji, w której klient (np. przeglądarka internetowa) regularnie wysyła żądania HTTP do serwera w celu sprawdzenia, czy są dostępne nowe dane. Jest to jeden z najprostszych sposobów implementacji komunikacji między klientem a serwerem w czasie quasi-rzeczywistym.

Klient wysyła żądanie HTTP: Klient (np. aplikacja webowa) wysyła żądanie HTTP GET lub POST do serwera w ustalonych odstępach czasu.

Serwer odpowiada: Serwer przetwarza żądanie i zwraca aktualne dane (jeśli są dostępne) lub pustą odpowiedź, jeśli nic się nie zmieniło.

Cykliczne żądania: Klient w krótkim czasie ponawia żądanie, aby ponownie sprawdzić dostępność nowych danych.

2. HTTP/2 Server Push to mechanizm wprowadzony w protokole HTTP/2, który pozwala serwerowi na wysyłanie danych do klienta bez konieczności wcześniejszego żądania tych danych przez klienta. Jest to sposób na poprawę wydajności i optymalizację ładowania zasobów w aplikacjach webowych.

3. Może MQTT???, który powstał w 1999 roku.

4. Zastosowanie protokołu WebSocket!!!, który powstał w 2009 roku.

WebSocket

- Protokół komunikacyjny zapewniający dwukierunkowy kanał wymiany danych poprzez pojedyncze połączenie TCP.
- Pierwsza przeglądarka Google Chrome obsługuje C od 2009 roku.
- Aktualnie wszystkie popularne przeglądarki wspierają tę technologię.
- NodeRED wykorzystuje WebSocket i można na nim stworzyć serwer oraz klientów.
- Klienci mogą publikować i subskrybować wiadomości.
- WebSocket umożliwia przesyłanie danych z minimalnym narzutem.
- Może pracować na tym samym porcie co przeglądarka.
- Wykorzystywany często w Internecie Rzeczy.
- Serwer WebSocket bardzo łatwo jest uruchomić na platformie Node.js.

Platforma Node.js

Node.js to środowisko uruchomieniowe, które umożliwia wykonywanie kodu JavaScript po stronie serwera. Powstało w 2009 roku i od tego czasu stało się jednym z najpopularniejszych narzędzi do tworzenia aplikacji webowych, API, a także narzędzi serwerowych.

Podstawowe informacje o Node.js

1. **Środowisko uruchomieniowe:** Node.js opiera się na silniku V8 Google, który służy do interpretacji i wykonywania kodu JavaScript w przeglądarkach. Dzięki Node.js, kod JavaScript może być wykonywany poza przeglądarką. *„JavaScript everywhere”.*
2. **Asynchroniczność:** Node.js jest asynchroniczne i oparte na zdarzeniach. Wykorzystuje model programowania oparty na zdarzeniach, co pozwala obsługiwać wiele żądań jednocześnie bez blokowania wątków.
3. **Jednowątkowość:** Node.js działa w jednym wątku, ale wykorzystuje **pętlę zdarzeń** (event loop), dzięki której może obsługiwać wiele żądań równocześnie.
4. **Non-blocking I/O:** Operacje wejścia/wyjścia (np. odczyt plików, zapytania do bazy danych) są nieblokujące, co oznacza, że aplikacja może kontynuować wykonywanie innych zadań, podczas gdy czeka na wynik operacji.
5. **Instalowanie pakietów**, globalnie z opcją –g: npm install -g nazwa-pakietu.
6. Node-RED działa na platformie Node.js.

Serwer HTTP na platformie Node.js

1. Instalacja platformy Node.js,
   jak jest zainstalowany Node-RED ,
   to jest zainstalowana platforma Node.js
2. Utworzenie i zapisanie pliku z kodem serwera
   „serwer.js” jak po prawej stronie
3. Uruchomienie serwera
   node serwer.js
4. Połączenie z serwerem.

serwer.js

```
const http = require('http');

const server = http.createServer((req, res) => {
   res.writeHead(200, { 'Content-Type': 'text/html' });
   res.write('<h1>Witaj w Node.js!</h1>');
   res.end('<h3>Tu jest koniec.</h3>');
});

   server.listen(3333, () => {
   console.log('Serwer działa na porcie 3333');
});
```

Praktyczne zastosowania Node.js

1. **Tworzenie API:** Dzięki Node.js możesz szybko stworzyć RESTful API do obsługi aplikacji frontendowych i mobilnych.
2. **Aplikacje działajace w czasie rzeczywistym:** Narzędzia takie jak **WebSockets** (np. pakiet socket.io) pozwalają na tworzenie aplikacji takich jak czaty czy gry online. Node-RED.
3. **Streaming danych:** Node.js świetnie sprawdza się w aplikacjach, które przetwarzają dane strumieniowe, takich jak serwery wideo czy audio.

Uruchomienie serwera WebSocket na platformie Node.js

1. Instalacja platformy Node.js,
   jak jest zainstalowany Node-RED ,
   to jest zainstalowana platforma Node.js
2. Dogranie pakietu (biblioteki ?) ws
   npm install ws
3. Utworzenie i zapisanie pliku z kodem serwera „serwer.js” jak po prawej stronie
4. Uruchomienie serwera
   node serwer.js
5. Połączenie z serwerem np. programem Postman

Sewer uproszczony

```
const WebSocket = require('ws');

// Tworzenie serwera WebSocket
const wss = new WebSocket.Server({ port: 8080 });

wss.on('connection', (ws) => {
 console.log('Nowe połączenie WebSocket');

 ws.on('message', (message) => {
  console.log('Otrzymano wiadomość:', message);
  ws.send('Odpowiedź serwera: ' + message);
 });

 ws.on('close', () => {
  console.log('Połączenie zamknięte');
 });
});
```

Plik serwer.js

```
// Importowanie biblioteki ws
const WebSocket = require('ws');

// Tworzenie serwera WebSocket na porcie 8080
const wss = new WebSocket.Server({ port: 8080 });

console.log('Serwer WebSocket uruchomiony na porcie 8080');

// Obsługa zdarzenia "connection" – nowe połączenie
wss.on('connection', (ws) => {
 console.log('Nowe połączenie WebSocket');

 // Obsługa wiadomości od klienta
 ws.on('message', (message) => {
  console.log('Otrzymano wiadomość:', message);

  // Wysłanie odpowiedzi do klienta
  ws.send(`Otrzymano: ${message}`);
 });

 // Obsługa zamknięcia połączenia
 ws.on('close', () => {
  console.log('Połączenie zamknięte');
 });

 // Obsługa błędów
 ws.on('error', (error) => {
  console.error('Błąd WebSocket:', error);
 });

 // Wysłanie wiadomości powitalnej do klienta
 ws.send('Witaj na serwerze WebSocket!');
});
```

Ten serwer odsyła wiadomości tylko do tego kto ją wysłał!?

Serwer odsyłający wiadomość do wszystkich klientów

Plik serwerAll.js

```
// Funkcja do wysyłania wiadomości do wszystkich klientów
function broadcast(message) {
  wss.clients.forEach((client) => {
    // Sprawdzenie, czy klient jest w stanie otwartym
    if (client.readyState === WebSocket.OPEN) {
      client.send(message);
    }
  });
```

wss.clients:
Zwraca zbiór (kolekcję) wszystkich aktywnych połączeń WebSocket.

client.readyState:
Każde połączenie WebSocket ma różne stany
(CONNECTING, OPEN, CLOSING, CLOSED).
Sprawdzany jest stan klienta, gdy połączenie jest otwarte (WebSocket.OPEN),
to do klienta wysyłana jest wiadomość.

client.send(message);
Wysłane wiadomości do klienta

Serwer WebSocket trzeba napisać samemu lub skorzystać z gotowego np Node-RED

```
const WebSocket = require('ws');

// Tworzenie serwera WebSocket na porcie 8080
const wss = new WebSocket.Server({ port: 8080 });

console.log('Serwer WebSocket uruchomiony na porcie 8080');

wss.on('connection', (ws) => {
 console.log('Nowe połączenie WebSocket');
 // Wysłanie powitalnej wiadomości do nowego klienta
 ws.send('Witaj na serwerze WebSocket!');

 // Obsługa wiadomości od klienta
 ws.on('message', (message) => {
  console.log('Otrzymano wiadomość:', message);
  // Rozesłanie wiadomości do wszystkich klientów
  broadcast(`Klient powiedział: ${message}`);
 });

 // Obsługa zamknięcia połączenia
 ws.on('close', () => {
  console.log('Połączenie zamknięte');
 });

 // Obsługa błędów
 ws.on('error', (error) => {
  console.error('Błąd WebSocket:', error);
 });
});

// Funkcja do wysyłania wiadomości do wszystkich klientów
function broadcast(message) {
 wss.clients.forEach((client) => {
  // Sprawdzenie, czy klient jest w stanie otwartym
  if (client.readyState === WebSocket.OPEN) {
   client.send(message);
  }
 });
}
```

Rozbudowa serwera

- Obsługa subskrypcji i publikacji wiadomości na różne tematy
- Identyfikacja klientów.
- …

Klient WebSocket na na stronie html

Klient Websocket

```
// Tworzenie połączenia WebSocket
const socket = new WebSocket('ws://localhost:8080');

// Obsługa otwarcia połączenia
socket.onopen = () => {
  console.log('Połączono z serwerem WebSocket');
  socket.send('Wiadomość od klienta');
};

// Obsługa wiadomości od serwera
socket.onmessage = (event) => {
  console.log('Wiadomość od serwera:', event.data);
};

// Obsługa błędów
socket.onerror = (error) => {
  console.error('Błąd WebSocket:', error);
};

// Obsługa zamknięcia połączenia
socket.onclose = () => {
  console.log('Połączenie zamknięte');
};
```

Elementy strony html i kodu JavaScript

Umieszczenie wiadomości w obiekcie (paragrafie)

```
<p id="p1"> Tu będzie tekst</p>
 …
document.getElementById('p1').innerHTML = event.data;
```

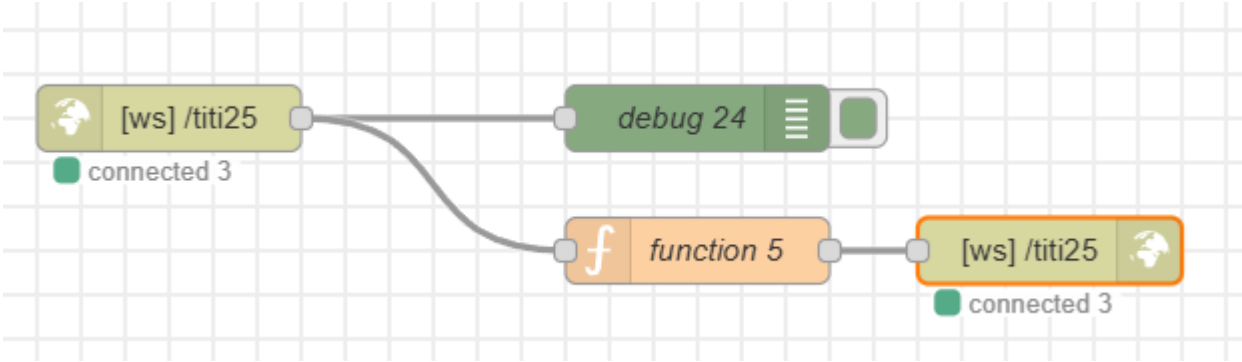
Pobranie wiadomości do wysłania

```
<input type="text" id="t1" value="Tu wpisujemy wiadomość">
 …
document.getElementById("t1").value
```

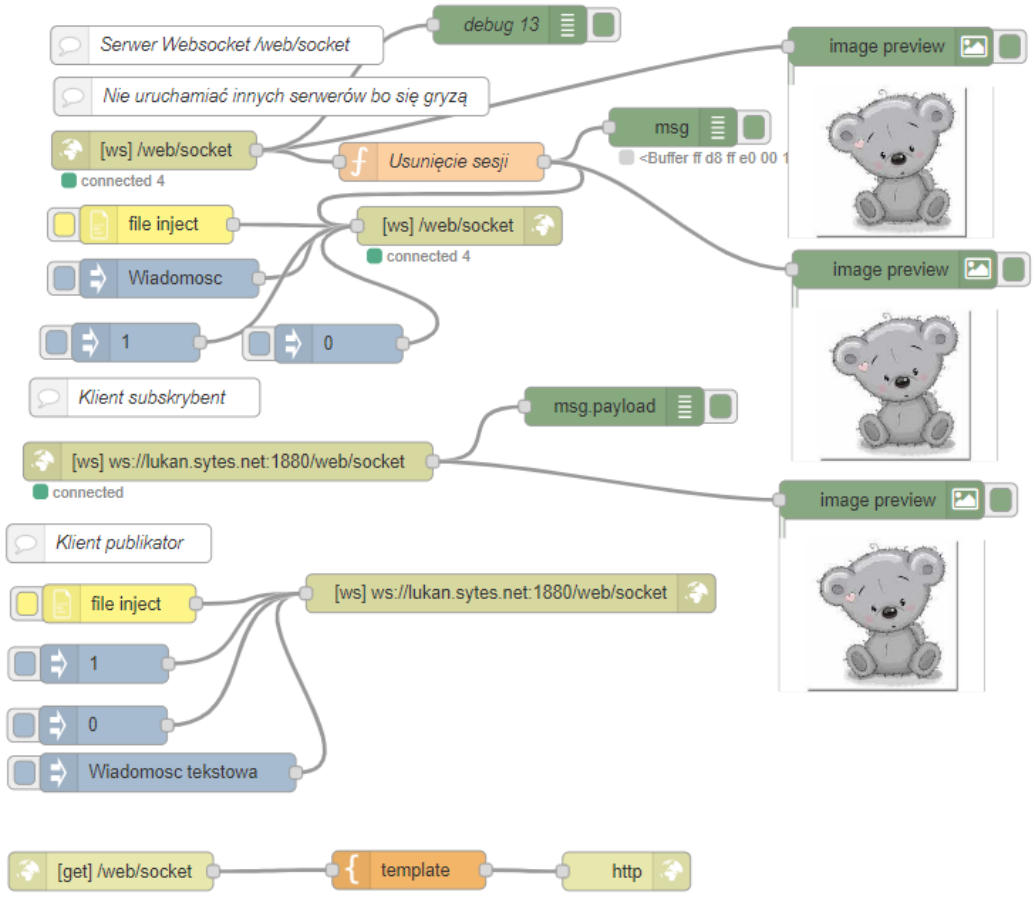
Wysłanie wiadomości po kliknięciu przycisku

```
<input id="Button1" type="button" value="Wyślij wiadomość"/>
 …
document.getElementById('Button1').addEventListener('click', buttonClik);
function buttonClik() {
            socket.send(document.getElementById("t1").value);
         }
Albo:
document.getElementById('Button1').addEventListener('click', () =>
socket.send(document.getElementById("t1").value) );
```

Serwer WebSocket w Node-RED



Przykład



- http://lukan.sytes.net:1880/#flow/45100628.515128
- http://lukan.sytes.net:1880/web/socket
- https://lukan.sytes.net:1881/#flow/efe381a3.121a7
- Strona WebSocket.html lokalnie

WebSocket na tym samym porcie co serwer WWW

WebSocket może działć na porcie 80 (HTTP) lub 443 (HTTPS), ale różni się od protokołu HTTP.
Używa specjalnego mechanizmu nazywanego **"handshake"**, który zamienia standardowe połączenie HTTP w pełnoprawne połączenie WebSocket.
Proces handshake:
Klient inicjuje połączenie, wysyłając zapytanie HTTP z nagłówkiem Upgrade.
Serwer odpowiada, akceptując żądanie połączenia WebSocket.
Po ustanowieniu połączenia, dane mogą być przesyłane w obu kierunkach w czasie rzeczywistym.

Żądanie

```
GET /chat HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
```

Odpowiedź

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

Cechy WebSocket

Dwukierunkowość: WebSocket pozwala zarówno klientowi, jak i serwerowi wysyłać dane w czasie rzeczywistym bez konieczności ponownego nawiązywania połączenia.
Stałe połączenie: Po ustanowieniu połączenia, komunikacja odbywa się za pomocą lekkiej ramki, co redukuje opóźnienia.
Efektywność: WebSocket eliminuje konieczność wielokrotnego przesyłania nagłówków HTTP, co obniża narzut protokołu i zużycie zasobów.
Obsługa binarna i tekstowa: WebSocket obsługuje zarówno dane tekstowe (np. JSON), jak i binarne (np. pliki lub strumienie audio/wideo).

Przykłady zastosowań

WebSocket, podobnie jak MQTT, znajduje zastosowanie w wielu dziedzinach, w których wymagana jest szybka, dwukierunkowa komunikacja:

1. **Aplikacje czatu:** Messenger, WhatsApp Web czy aplikacje obsługujące komunikację w czasie rzeczywistym.
2. **Gry online:** Synchronizacja stanu gry pomiędzy graczami.
3. **Notyfikacje w czasie rzeczywistym:** Powiadomienia o zmianach w aplikacjach, takich jak e-maile czy media społecznościowe.
4. **Systemy handlowe:** Przesyłanie cen akcji w czasie rzeczywistym.
5. **Aplikacje IoT:** Integracja urządzeń Internetu Rzeczy z centralnymi serwerami.
6. **Współdzielone edytory dokumentów:** Jednoczesne edytowanie plików przez wielu użytkowników.
7. **Node-RED**

WebSocket to wszechstronny i wydajny protokół komunikacyjny, idealny do zastosowań w czasie rzeczywistym. Jego zalety, małe opóźnienia i pełna dwukierunkowość, czynią go doskonałym wyborem w projektach wymagających szybkiej i niezawodnej wymiany danych.