

WebSocketek

Kétirányú kommunikáció az AJAX-on túl

Derzsi Dániel

Bevezető – Milyen a kétoldalú kommunikáció?

- A HTTP protokoll a World Wide Web alapját képezi.
- Viszont, csupán half-duplex kommunikációt tesz lehetővé (kétoldalú kommunikáció, de nem egy időben)
- A HTTP szerver válaszát mindig egy kliens kérése előzi meg
- A HTTP szerver nem küldhet később üzeneteket a kliensnek
- Mi lenne, ha a HTTP szerver és a kliens (böngésző) között szabad kétoldalú kommunikáció lenne?

Történelmi megvalósítások - Flash

- A legelső próbálkozás: Macromedia Flash 5 – XMLSocket bevezetése 2000-ben
- XML alapú üzenetek küldése és fogadása egy időben
- Előnye: Ez volt az egyetlen megoldás
- A Flash már 2017 óta nem támogatott, 2021 óta az Adobe világszerte letiltotta
- Hátrányai:
 - Az XML használata kötelező
 - Bináris adatokat hogy küldhetünk? Kódolás nélkül lehetetlen – nagyobb sávszélesség szükséges!
 - Nem nyílt webes szabvány – Böngésző kiterjesztést igényel

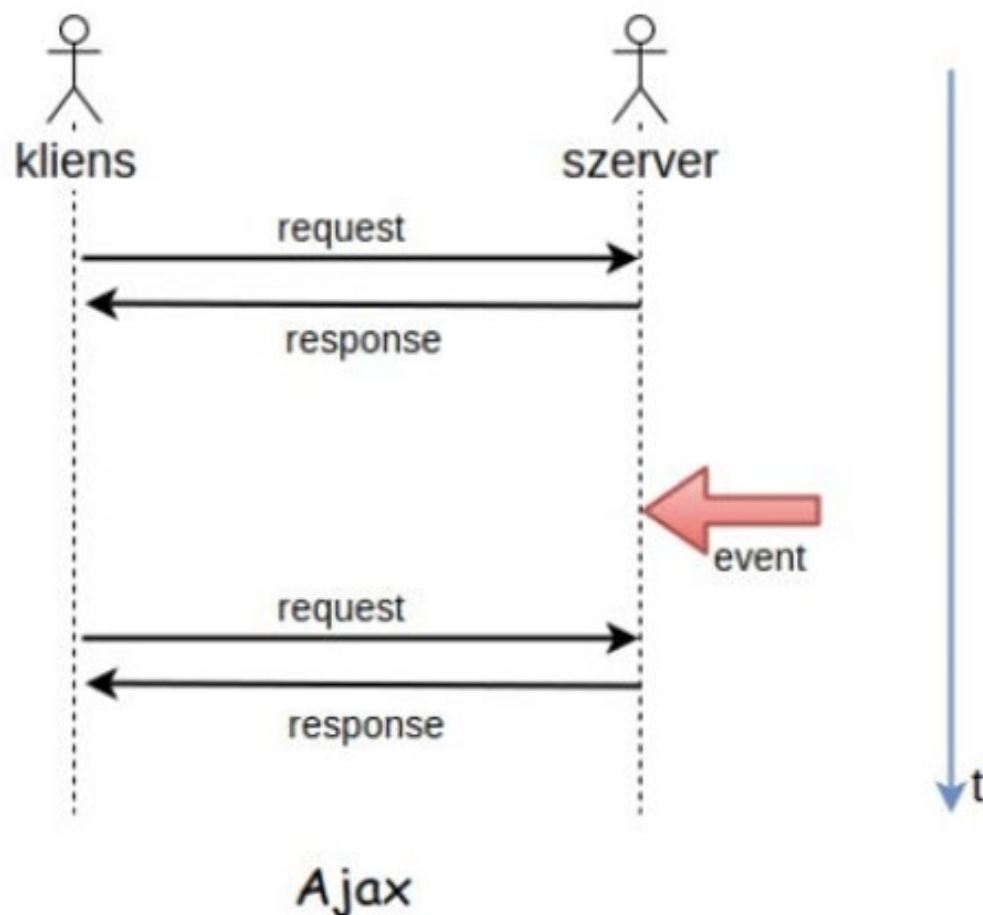
Történelmi megvalósítások - Flash

- A Flash már 2017 óta nem támogatott, 2021 óta az Adobe világszerte letiltotta
- A Flash XMLSocket képezi az alapját a modern megoldásoknak
- Eseményalapú megközelítés: hasonlóan a modern webes szabványokhoz

Történelmi megvalósítások - AJAX

- A második próbálkozás az AJAX: Asynchronous JavaScript and XML – 2002-ben XMLHttpRequest
- Célja a sávszélesség csökkentése és a sebesség növelése → Felhasználói élmény fokozása
- Hogyan jut el a klienshez a szerveroldalon frissült adat?
- Megoldás: a kliens oldalon időzített, ismétlődő kérések (requestek) a szerveroldal felé – AJAX Polling
- A tartalom JavaScript-el módosul az oldalon

Történelmi megvalósítások - AJAX



Történelmi megvalósítások - AJAX

- Legnagyobb hátránya: Még mindig a kliens kezdeményez
- Felesleges hálózati forgalom
- Leterhelődik a szerver gép
- Nem skálázható
- A felhasználó csak meghatározott időközökben kap visszajelzést az alkalmazástól ➔ not responsive!

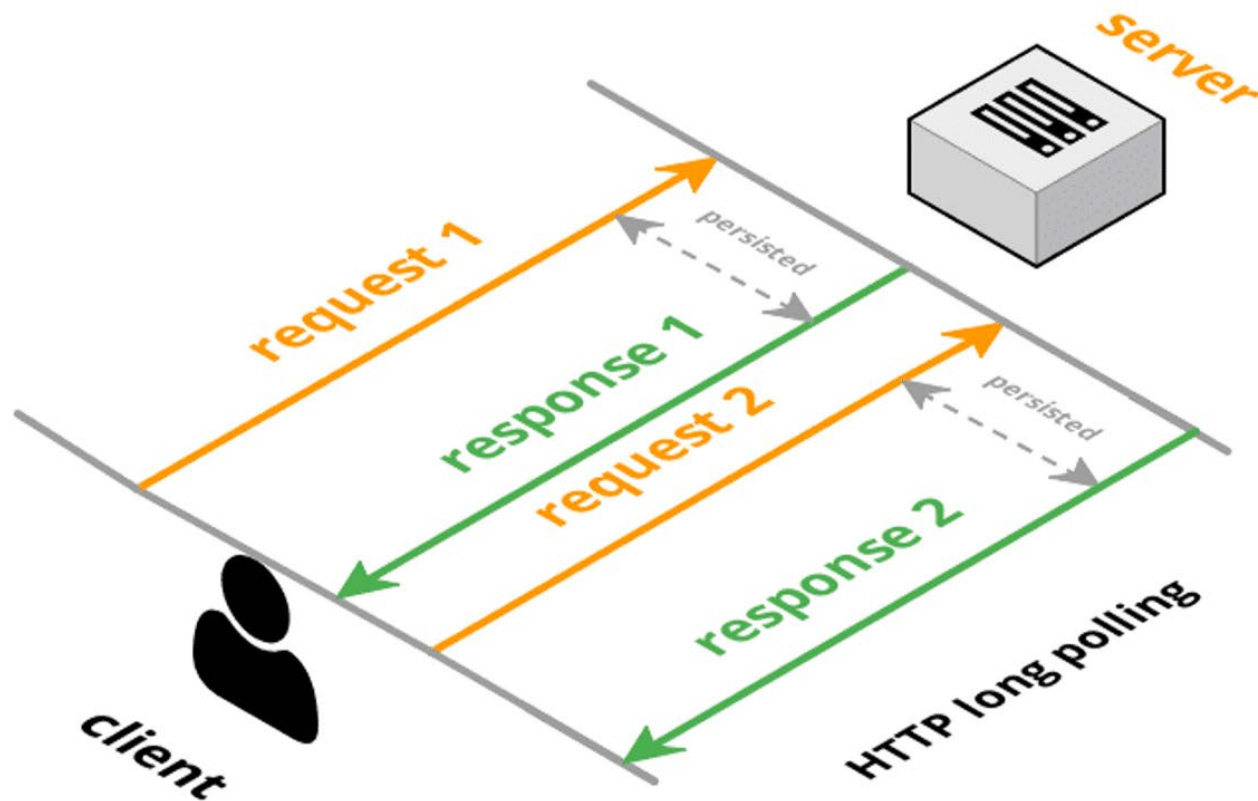
Történelmi megvalósítások - AJAX

```
1 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
2 <script>
3   var gameState = $('.game-state');
4
5   var poll = function() {
6     $.ajax({
7       url: 'game.json',
8       dataType: 'json',
9       type: 'get',
10      success: function(data) {
11        gameState.innerHTML = data.newGameState;
12      }
13    });
14  };
15
16  var pollInterval = setInterval(function() {
17    poll();
18  }, 500);
19</script>
```


Történelmi megvalósítások – Long polling

- A harmadik próbálkozás a Long polling megközelítés
- Ugyancsak a kliens kezdeményez
- A szerver fenntartja a kliens kapcsolatát, amíg nem történik timeout (időtúllépés)
- Az adatok frissítése esetén a szerver választ küld és a kapcsolatot egyből lezárja
- Az időtúllépés vagy válaszküldés után új kapcsolat kezdődik
- Előnye: Hatékonyabb, mint a hagyományos polling
- Hátránya: A szerver több kapcsolatot kell egyszerre fenntartsion – erőforrás éheztetés (resource starvation)

Történelmi megvalósítások – Long polling



Történelmi megvalósítások – Silverlight

- A negyedik próbálkozás a Microsoft Silverlight - 2007
- Hasonlóan a Flash-hez, böngésző kiterjesztést igényel
- A Windows Communication Framework (WCF) lehetővé teszi a kétoldalú, full-duplex kommunikációt
- Nem JavaScript alapú megközelítés: C# kód írása szükséges
- Ezt a technológiát aligha használták, nem kapták fel a fejlesztők



WebSocket

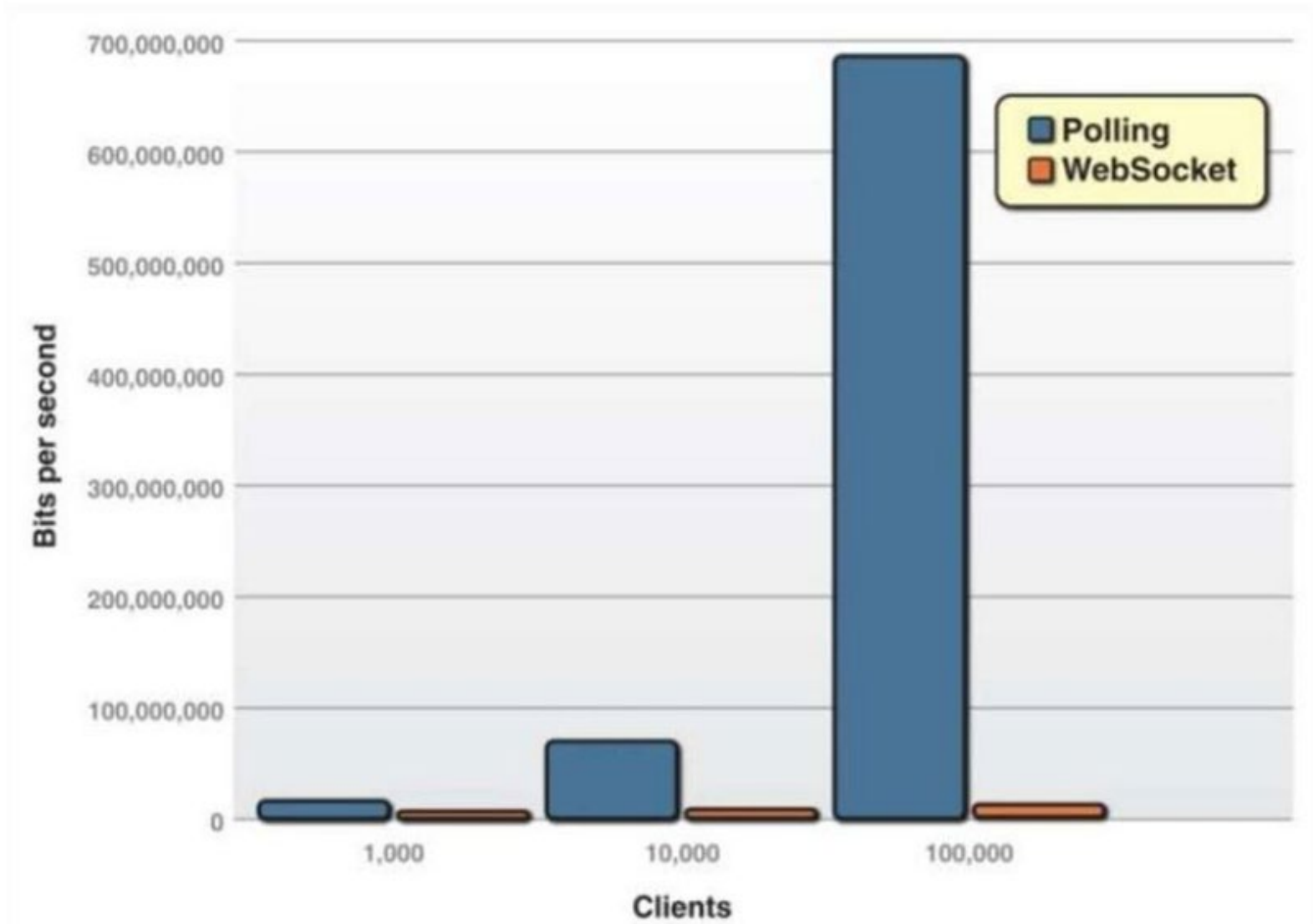
Isten hozott a WebSocket-ek világában!

- A legmodernebb, legújabb megvalósítás: a WebSocketek
- 2011 óta szabványosított protokoll: RFC 6455
- Kétirányú, perzsisztens, full-duplex csatorna a kliens és a szerver között
- HTTP-től független TCP protokoll, de HTTP protokollról felfejleszthetjük a kapcsolatainkat WebSocket (WS) protokollra

Isten hozott a WebSocket-ek világában!

- Alapvetően bináris protokoll: optimalizált sávszélesség
- Eseményorientált megközelítés, szerver és kliens oldalon egyaránt
- A küldés/fogadás üzenet alapú
- Az üzenetek technikailag frame-ekre bontva utaznak – csupán 6 byte-os header!
- Megoldja az AJAX hiányosságait:
 - Azonnali válasz
 - Optimalizált sávszélesség
 - Kevés erőforrást igényel – nincs TCP kapcsolat újraépítés
 - Bináris kommunikáció lehetőségét is biztosítja
 - **Stateful**: Nincs szükség külső adatbázisra az adatok perzisztenssé tételéhez

WebSocket – polling összehasonlítás



WebSocket kapcsolat felépítése

- HTTP kérés:

```
GET /wstest HTTP/1.1
```

```
Host: server.example.com
```

```
Sec-WebSocket-Version:"13"
```

```
Sec-WebSocket-Key:"2yJIeg5iwroBBmCpUPCy+A=="
```

```
Connection: keep-alive, Upgrade
```

```
Upgrade: websocket
```

- HTTP válasz: 258EAF5A5-E914-47DA- 95CA-C5AB0DC85B11

```
HTTP/1.1 101 Switching Protocols
```

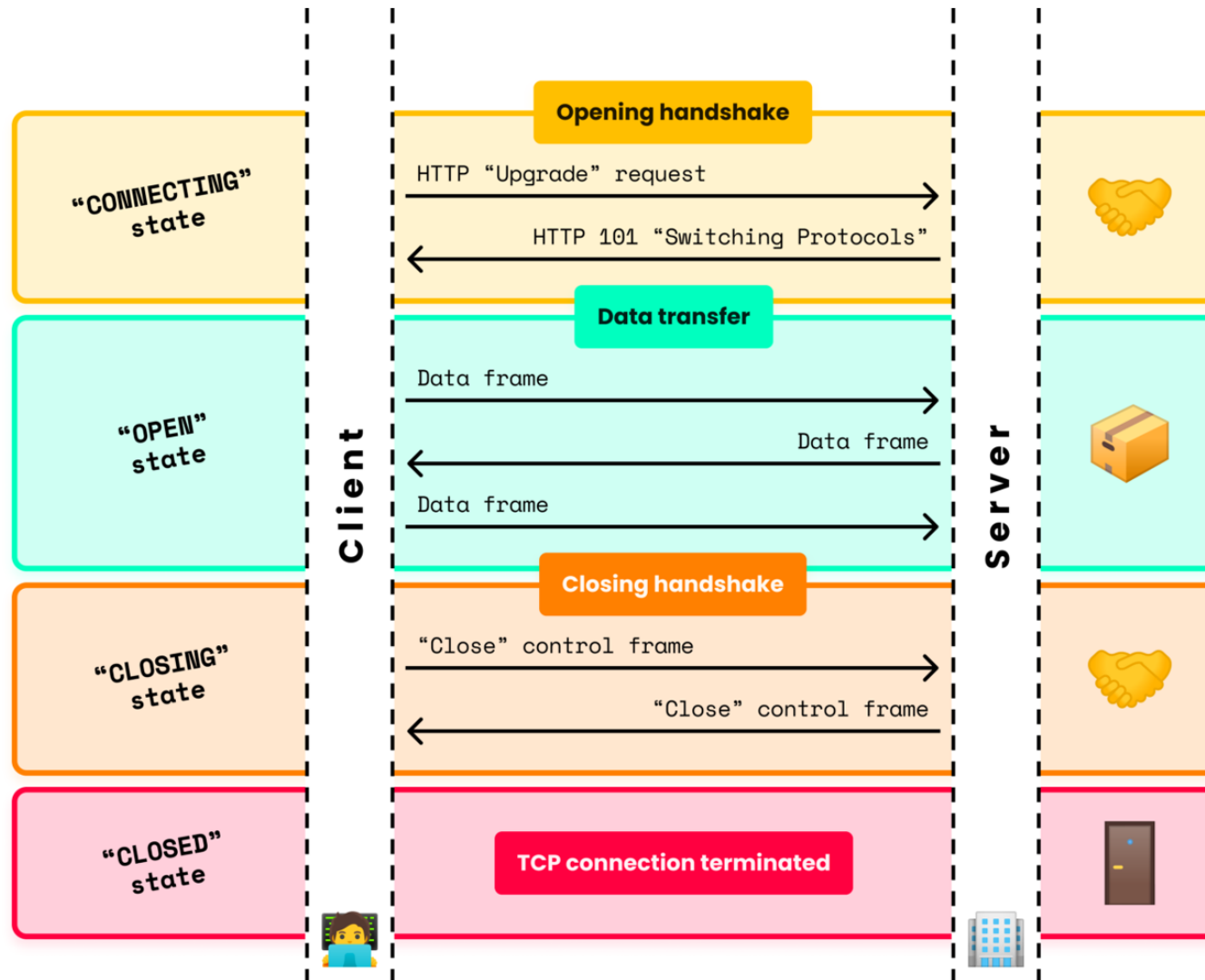
```
Sec-WebSocket-Accept:"jT2uT0a6MYwabx3iWngsre6+Gpw="
```

```
Connection: Upgrade
```

```
Upgrade: WebSocket
```

- A Sec-WebSocket-Key generálásáról:
<https://rfc-editor.org/rfc/rfc6455#page-7>

WebSocket kapcsolatok életciklusa



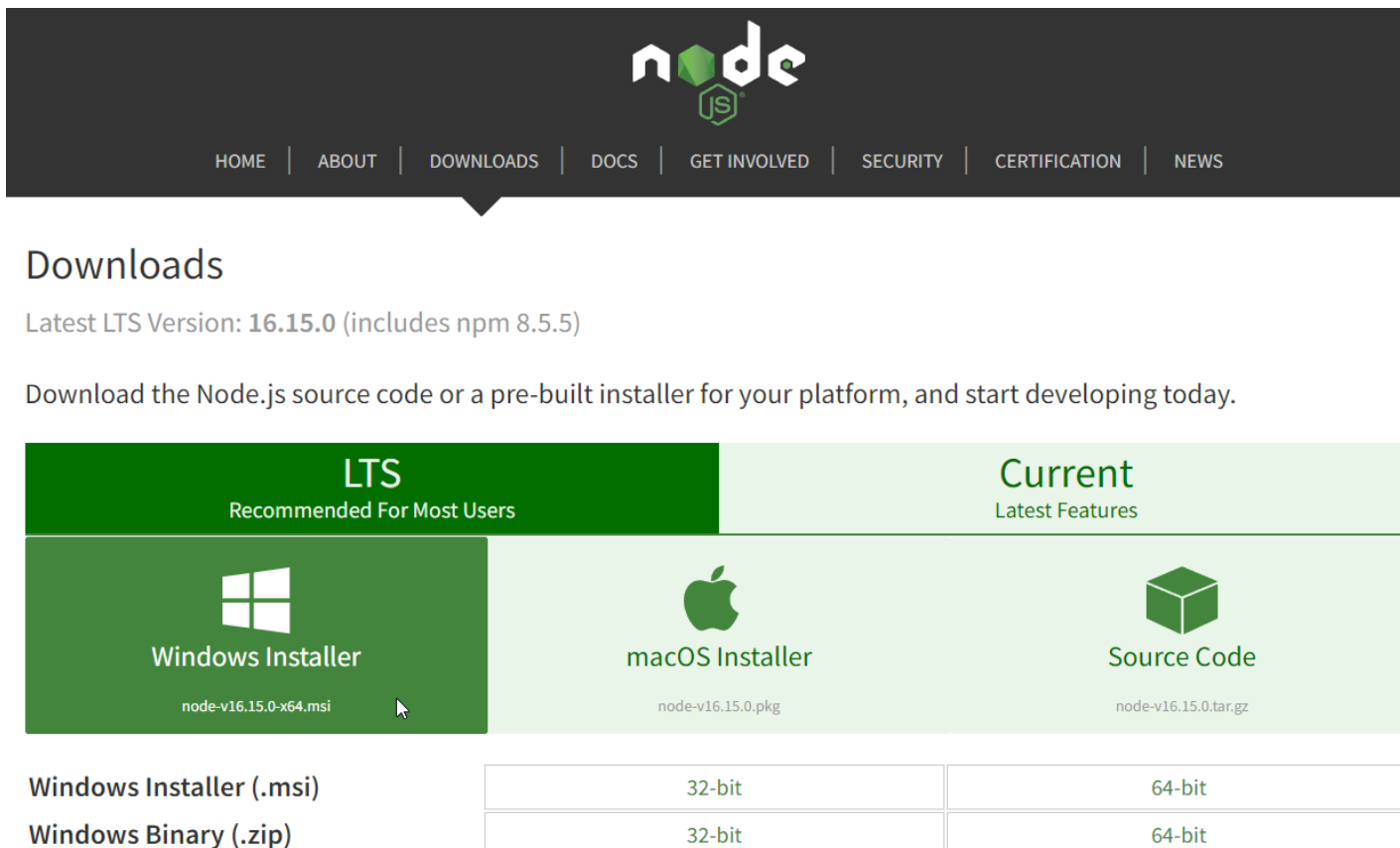
WebSocket kapcsolatok – Könyvtárak

- A WebSocketek köré széles körű ekosisztéma épül
- Nem csupán JavaScriptes implementációk léteznek
- PHP: Ratchet - <http://socketo.me>
- Rust: websocket - <https://docs.rs/websocket>
- JavaScript:
 - ws - <https://github.com/websockets/ws>
 - Natív böngésző implementáció – Chrome, Firefox, Safari
- Python:
 - Websockets - <https://websockets.readthedocs.io>
 - Autobahn - <https://autobahn.readthedocs.io>

WebSocket kapcsolatok – Kliens és szerver

- Mivel a WebSocket kapcsolatok kétoldalú, párhuzamos kommunikációt tesznek lehetővé, ezért sok különbség nincs a kliens és a szerver között
- A kliens is, illetve a szerver is a következő eseményekre figyelhet:
 - open – Mikor létrejön egy kapcsolat
 - close – Mikor lezáródik egy kapcsolat
 - message – Mikor érkezik egy üzenet a másik féltől
 - error – Mikor hiba történik a kommunikáció során
 - ping / pong – Tesztelésre és heartbeat mechanizmusokra használt üzenetek

WebSocket kapcsolatok – Szerver létrehozása



The screenshot shows the Node.js download page. At the top is the Node.js logo and a navigation bar with links: HOME, ABOUT, DOWNLOADS, DOCS, GET INVOLVED, SECURITY, CERTIFICATION, and NEWS. Below the navigation bar is the 'Downloads' section, which states the latest LTS version is 16.15.0 (including npm 8.5.5) and provides instructions to download the source code or a pre-built installer. The page is divided into two main columns: 'LTS Recommended For Most Users' and 'Current Latest Features'. The 'LTS' column contains a 'Windows Installer' button with the file name 'node-v16.15.0-x64.msi'. The 'Current' column contains buttons for 'macOS Installer' (file: 'node-v16.15.0.pkg') and 'Source Code' (file: 'node-v16.15.0.tar.gz'). Below these buttons is a table with two rows: 'Windows Installer (.msi)' and 'Windows Binary (.zip)', each with columns for '32-bit' and '64-bit'.




node

HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | CERTIFICATION | NEWS

Downloads

Latest LTS Version: **16.15.0** (includes npm 8.5.5)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS Recommended For Most Users	Current Latest Features	
 Windows Installer node-v16.15.0-x64.msi	 macOS Installer node-v16.15.0.pkg	 Source Code node-v16.15.0.tar.gz
Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit

<https://nodejs.org/en/download/>

WebSocket kapcsolatok – Új szerver létrehozása

```
1 [tohka@sundae ~]$ mkdir project
2 [tohka@sundae ~]$ cd project
3 [tohka@sundae ~/project]$ npm init
4   This utility will walk you through creating a package.json file.
5   It only covers the most common items, and tries to guess sensible defaults.
6 [tohka@sundae ~/project]$ npm install --global yarn
7   changed 1 package, and audited 2 packages in 714ms
8   found 0 vulnerabilities
9 [tohka@sundae ~/project]$ yarn add ws
10  yarn add v1.22.18
11  success Saved 1 new dependency
12  Done in 0.77s.
```

WebSocket kapcsolatok – Demo telepítése

```
1 [tohka@sundae ~]$ git clone https://github.com/darktohka/websocket-demos
2 [tohka@sundae ~]$ cd websocket-demos
3 [tohka@sundae ~/websocket-demos]$ npm install --global yarn
4   changed 1 package, and audited 2 packages in 2s
5   found 0 vulnerabilities
6 [tohka@sundae ~/websocket-demos]$ yarn install
7   yarn install v1.22.18
8   [1/4] Resolving packages...
9   success Already up-to-date.
10  Done in 0.17s.
11 [tohka@sundae ~/websocket-demos]$ yarn ping-demo
12   yarn run v1.22.18
13   $ node src/ping-demo-server.js
14   The WebSocket server is now running on port 8080!
```

WebSocket kapcsolatok – Szerver létrehozása

```
1 import { WebSocketServer } from 'ws';
2
3 const wss = new WebSocketServer({ port: 8000 });
4
5 wss.on('connection', function connection(ws) {
6   ws.on('message', function message(data) {
7     console.log('Received message from client: %s', data);
8     ws.send(`Hey, you sent me this message recently: ${data}`);
9   });
10
11   ws.send('A welcome message to our new client!');
12 });
```

WebSocket kapcsolatok – Kliens létrehozása (NodeJS)

```
1 import WebSocket from 'ws';  
2  
3 const ws = new WebSocket('ws://127.0.0.1:8000');  
4  
5 ws.on('open', function open() {  
6   ws.send('Some sort of message, anything really!');  
7 });  
8  
9 ws.on('message', function message(data) {  
10  console.log('Received message: %s', data);  
11 });
```


WebSocket kapcsolatok – Kliens létrehozása (Böngésző)

```
1 const ws = new WebSocket('ws://127.0.0.1:8000');  
2  
3 ws.addEventListener('open', (event) => {  
4   console.log('Connection established with the server!');  
5   ws.send('Hello world! How are you, server?');  
6 });  
7  
8 ws.addEventListener('close', (event) => {  
9   alert('Our WebSocket connection has been closed!');  
10});  
11  
12 ws.addEventListener('message', (event) => {  
13   const message = event.data;  
14   alert(`Server has responded: ${message}`);  
15});
```

WebSocket kapcsolatok – Periodikus üzenetek

```
1 import WebSocket, { WebSocketServer } from 'ws';  
2  
3 const wss = new WebSocketServer({ port: 8000 });  
4  
5 setInterval(function sendPeriodicMessage() {  
6   wss.clients.forEach(function each(client) {  
7     if (client.readyState === WebSocket.OPEN) {  
8       client.send("Hey guys, I'm still alive! How are you all?");  
9     }  
10  });  
11}, 1000);
```



WebSocket kapcsolatok – Titkosítás

- Biztonsági szempontok miatt egy HTTPS szerverről nem lehet egy nem biztonságos WebSocket kapcsolatot létesíteni
- Ha biztonságos WebSocket szervert szeretnénk létre hozni, be kell állítsuk az SSL bizonyítványunkat
- Ha nincs SSL bizonyítványunk, kérhetünk egyet ingyen a LetsEncrypt tanúsító hatóságtól:
<https://letsencrypt.org/>

WebSocket kapcsolatok – Titkosítás

```
1 import { createServer } from 'https';
2 import { readFileSync } from 'fs';
3 import { WebSocketServer } from 'ws';
4
5 const server = createServer({
6   cert: readFileSync('cert.pem'),
7   key: readFileSync('key.pem')
8 });
9 const wss = new WebSocketServer({ server });
10
11 wss.on('connection', function connection(ws) {
12   ws.on('message', function message(data) {
13     console.log('Received a message from a client: %s', data);
14   });
15
16   ws.send('Welcome to our server!');
17 });
18
19 server.listen(8000);
```

WebSocket kapcsolatok – Titkosítás

```
1 // Encrypted WebSocket server connection
2 const ws = new WebSocket("wss://example.org:8000");
3
4 ws.addEventListener('open', (event) => {
5   ws.send('Hello there! Nobody will be able to eavesdrop on us...');
6 });
```

WebSockets – Demo GitHub



<https://github.com/darktohka/websocket-demos>