JESUÏTES educació

DAWM06UF4
COMUNICACIÓ ASÍNCRONA CLIENT-SERVIDOR

UF4.4 HTML5. WEBSOCKETS

CFGS Desenvolupament d'Aplicacions Web
M06. Desenvolupament web en entorn client
Fundació Jesuïtes Educació - Escola del Clot
Sergi Grau sergi.grau@fje.edu

OBJECTIUS

- Aprendre les característiques del funcionament de les comunicacions basades en el nou protocol de websockets

- Entrendre les aplicacions del nou protocol així com els avantatges i inconvenients que suposa

- Utilitzar la biblioteca socket.io, per a fer comunicacions amb websockets entre frontend en JS i backend en Node.js

QUÈ SÓN ELS WEBSOCKETS?

- WebSockets és una tecnologia avançada que fa possible obrir una sessió de comunicació interactiva entre el navegador de l'usuari i un backend.

- Amb aquesta API es poden enviar i rebre missatges a un servidor controlats per esdeveniments sense haver de consultar de manera repetitiva al servidor per a una resposta.

APLICACIONS CLÀSSIQUES HTTP/1.1

- Internet s'ha creat en gran part a partir de l'anomenat paradigma petició / resposta d'HTTP.
- Un client carrega una pàgina web i no passa res fins que l'usuari fa clic a la pàgina següent. Al 2005, **AJAX** va començar a fer que Internet semblés més dinàmic. Tot i això, totes **les comunicacions HTTP eren dirigides pel client,** el que requeria la interacció de l'usuari o feia necessari preguntar periòdicament cada vegada que es carregaven noves dades del servidor.

COMET I TECNOLOGIES PUSH

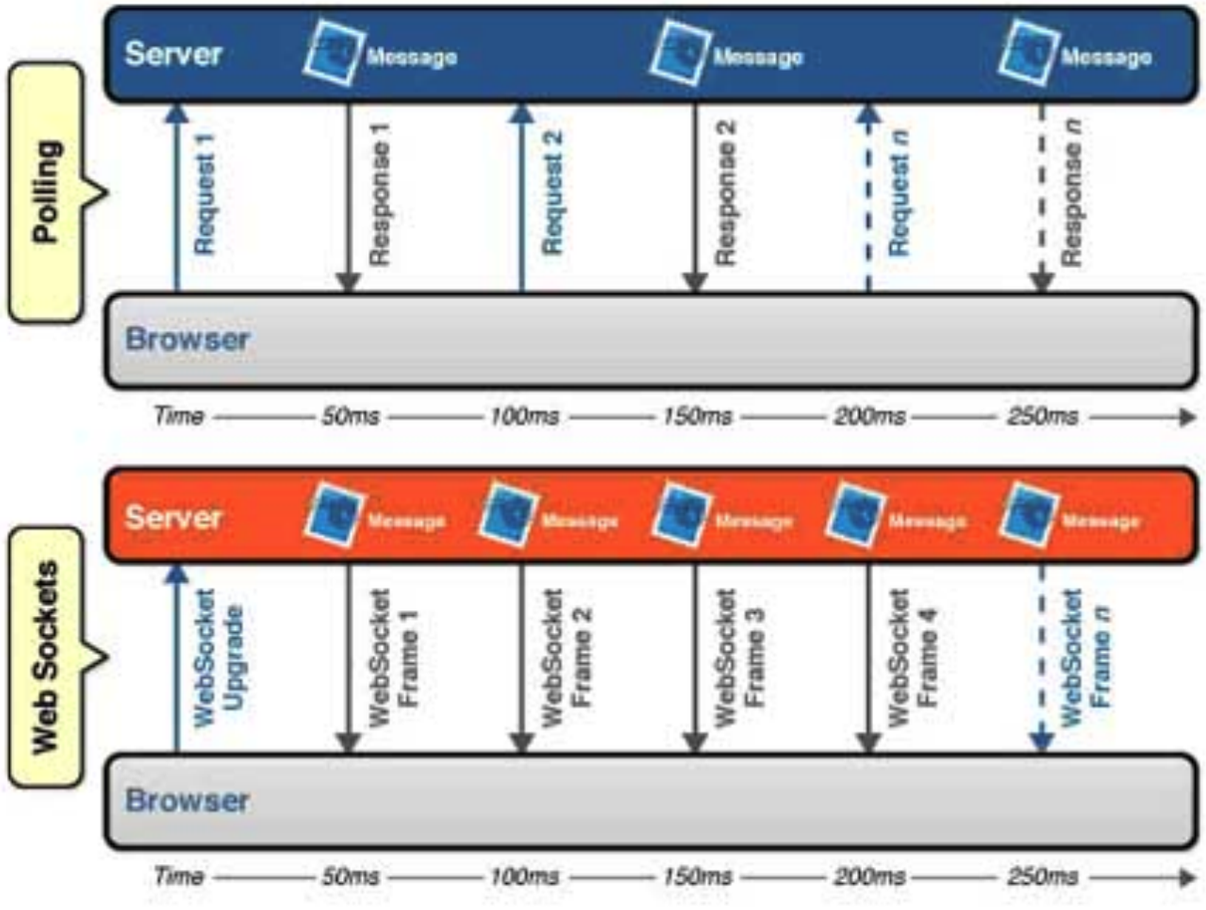
- Ja fa algun temps que existeixen tecnologies que permeten al servidor enviar dades al client en el mateix moment que detecta que hi ha noves dades disponibles. Es coneixen com a **"Push" o "Comet"** .

- Un dels trucs més comuns per crear la il·lusió d'una connexió iniciada pel servidor s'anomena Long Polling (sondeig llarg).

- Amb el **Long polling**, el client obre una connexió HTTP amb el servidor, el qual la manté oberta fins que s'enviï una resposta. Cada vegada que el servidor tingui dades noves, enviarà la resposta (altres tècniques impliquen Flash , sol·licituds XHR de diverses parts i els anomenats htmlfiles). El Long Polling i les altres tècniques funcionen bastant bé.

INCONVENIENTS SOLUCIONS CLÀSSIQUES

- No obstant això, totes aquestes solucions comparteixen un problema: l'excés de l'HTTP, el que no les fa aptes per a aplicacions de baixa latència. Pensa per exemple en els jocs multijugador de tir en primera persona del navegador o en qualsevol altre joc en línia amb un component en temps real.



QUÈ SÓN ELS WEBSOCKETS?

- Els WebSocket és un **nou estàndard per a la comunicació en temps real al web**, que forma part del paraigües de HTML5. WebSocket és un senzill API de JavaScript i el protocol d'acompanyament que li permet crear "sockets web", amb capacitat per a la comunicació full-duplex bidireccional a través d'una connexió TCP persistent (socket). Aquestes "sockets web", a diferència de sockets TCP tradicionals, són molt fàcils d'utilitzar.

- Amb aquesta API es poden enviar i rebre missatges a un servidor **controlats per esdeveniments** sense haver de consultar de manera repetitiva al servidor per a una resposta.

COM FUNCIONA?

- Com l'estàndard HTTP, WebSocket per defecte utilitza el port 80 en el clar i 443 a través de SSL.

- El client WebSocket estableix una connexió HTTP i fa una petició per a canviar el protocol utilitzant el mecanisme d'actualització d'HTTP

- A continuació, segueix un protocol d'enllaç per assegurar tant el client com el suport de servidor WebSocket. Atès que les connexions WebSocket comencen com HTTP, WebSocket pot treballar a través de molts proxies i tallafocs existents, a diferència d'altres protocols.

COM FUNCIONA?

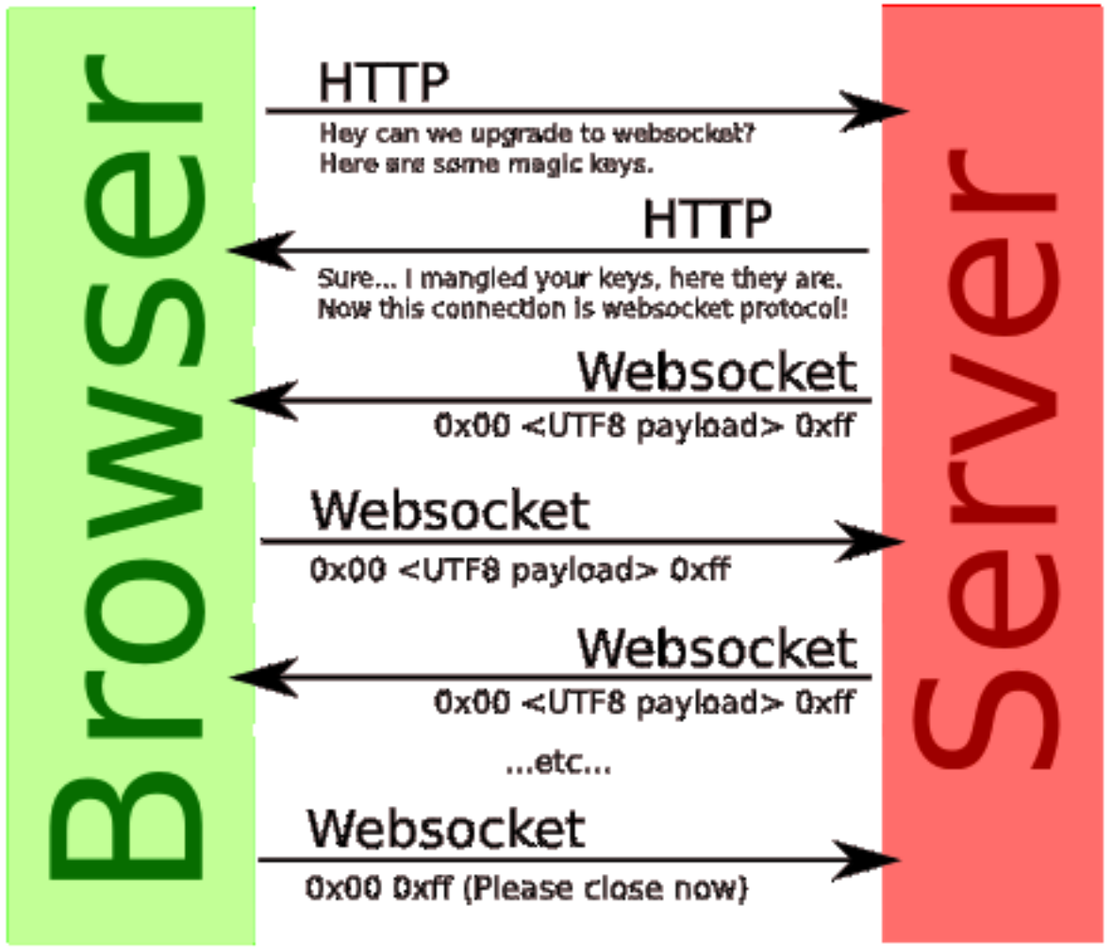
- Un cop establerta la connexió, els missatges s'envien com "frames", ja sigui de text o en format binari, en ambdues direccions. Aquestes són les cadenes de dades que envia i rep JavaScript en el frontend.

- Les URI WebSocket tenen el mateix format bàsic com HTTP , però amb un esquema URI diferent: **ws://domini:port/ruta**.

- La ruta pot ser utilitzada per distingir la finalitat de la connexió, però, alguns servidors la ignoren.

INCORPORACIÓ DE SOCKETS A LA WEB

- L'especificació WebSocket defineix una API que estableix connexions "socket" entre un navegador web i un servidor. Dit amb altres paraules: hi ha una connexió persistent entre el client i el servidor, i ambdues parts poden començar a enviar dades en qualsevol moment.



OBRIR UNA CONNEXIÓ

- Per obrir una connexió WebSocket, només cal executar el constructor WebSocket:

```
var connexio = new WebSocket('ws://domini/echo', ['soap',
'xmpp']);
```

- Observa els elements ws: . Aquest és el nou esquema d'URL per a les connexions WebSocket. També hi ha wss: per a connexions WebSocket segures, de la mateixa manera que s'utilitza https: per a les connexions HTTP segures.

- Adjuntar immediatament diversos manegadors d’esdeveniments a la connexió et permet saber quan està oberta la connexió, quan ha rebut missatges entrants o quan hi ha un error.

OBRIR UNA CONNEXIÓ

- El segon argument accepta subprotocolos opcionals. Pot ser una cadena o una matriu de cadenes. Cada cadena ha de representar un nom de subprotocol i el servidor accepta només un dels subprotocols de la matriu. Per determinar el subprotocol acceptat, accedeix a la propietat protocol de l'objecte WebSocket.

- Els noms subprotocols han de ser un dels registrats en el registre de IANA . A data de febrer de 2012, només hi ha un nom de subprotocol registrat (soap).

OBRIR UNA CONNEXIÓ

client.js

```
// Quan la connexió està oberta envia alguna dada al servidor
connexio.onopen = function () {
  connexio.send('Ping'); // Envia al servidor el missatge
'Ping'
};

// Log errors
connexio.onerror = function (error) {
  console.log('WebSocket Error ' + error);
};

// missatges rebuts des del servidor
connection.onmessage = function (e) {
  console.log('Server: ' + e.data);
};
```

COMUNICACIÓ AMB EL SERVIDOR

- Quan s'estableixi una connexió amb el servidor (quan l'esdeveniment open s'activi), podem començar a enviar dades al servidor amb el mètode send(missatge') en l'objecte de connexió.

- Abans només s'admetien cadenes, però l'última especificació també permet enviar missatges binaris. Per enviar dades binàries, pots utilitzar un objecte Blob o ArrayBuffer .

client.js

```
// enviar una cadena
connexio.send ('missatge');

// enviar una imatge, ImageData es ArrayBuffer
var img = canvas_context.getImageData (0, 0, 400, 320);
var binary = new Uint8Array (img.data.length);
 for (var i = 0; i <img.data.length; i + +) {
   binary [i] = img.data [i];
 }
 connexio.send (binary.buffer);

 // enviar un fitxer, es Blob
 var fitxer = document.querySelector ('input [type = "file"]').
files[0];
 connexio.send (fitxer);
```

COMUNICACIÓ AMB EL SERVIDOR

- **De la mateixa manera, el servidor pot enviar missatges en qualsevol moment**. Cada vegada que això passi, es crida el callback onmessage . Aquest callback rep un objecte "event", el que permet accedir al missatge actual mitjançant la propietat data .

- L'última especificació de WebSocket també permet rebre missatges binaris. Els marcs binaris es poden rebre en format Blob o ArrayBuffer . Per especificar el format del binari rebut, estableix la propietat binaryType l'objecte WebSocket a "blob" o "arraybuffer". El format per defecte és "blob" (no cal alinear el paràmetre binaryType en enviar).

```
connexio.binaryType = 'arraybuffer';
connexio.onmessage = function (i) {
  console.log (e.data.byteLength);  // ArrayBuffer si és
binari };
```

- Una altra funció recentment incorporada a WebSocket són les **extensions**.

- Les extensions permeten enviar marcs comprimits , multiplexats , etc. Per enviar extensions que accepti el servidor, examina la propietat de les extensions de l'objecte WebSocket després d'un esdeveniment "open".

```
console.log (connexio.extensions);
```

COMUNICACIONS D'ORIGEN CREUAT

+ Les comunicacions d'origen creuat són un protocol modern creat directament per WebSocket. Encara que segueix sent necessari que t'asseguris que només et comuniques amb clients i servidors de confiança, WebSocket permet la comunicació entre les parts de qualsevol domini.

+ El servidor decideix si vol posar el seu servei a disposició de tots els clients o sol dels que estan allotjats en un grup de dominis ben definits.

SERVIDORS PROXY

+ Tota nova tecnologia comporta una nova sèrie de problemes. En el cas de WebSocket, es tracta de la compatibilitat amb els servidors proxy que intervenen les connexions HTTP en la majoria de les xarxes corporatives. El protocol WebSocket utilitza el sistema d'actualització d'HTTP (normalment utilitzat per HTTP / SSL) per a "actualitzar" una connexió HTTP a una connexió WebSocket. A alguns servidors proxy no els agrada i cancel·la la connexió. Per tant, encara que un client donat utilitzi el protocol WebSocket, és possible que no pugui establir una connexió.

US ACTUAL

- WebSocket segueix sent una tecnologia jove i no està implementada completament en tots els navegadors. Actualment, però, pots utilitzar WebSocket amb les biblioteques que utilitzen una de les alternatives esmentades anteriorment si WebSocket no està disponible.

- A causa de la sol·licitud HTTP extra, sempre hi haurà un excés en comparació amb el WebSocket pur.

- https://en.wikipedia.org/wiki/Comparison_of_WebSocket_implementations

- Per establir una connexió WebSocket, el client envia una petició de negociació WebSocket, i el servidor envia una resposta de negociació WebSocket, com es pot veure en el següent exemple:

- Petició del navegador al servidor:

```
GET /demo HTTP/1.1
Host: example.com
Connection: Upgrade
Sec-WebSocket-Key2: 12998 5 Y3 1 .P00
Sec-WebSocket-Protocol: sample
Upgrade: WebSocket
Sec-WebSocket-Key1: 4 @1 46546xW%0l 1 5
Origin: http://example.com

^n:ds[4U
```

- Resposta del servidor:

```
HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Location: ws://example.com/demo
Sec-WebSocket-Protocol: sample

8jKS'y:G*Co,Wxa-
```

- Els 8 bytes amb valors numèrics que acompanyen els camps Secció WebSocket-Key1 i Secció WebSocket-Key2 són tokens aleatoris que el servidor utilitzarà per construir un símbol de 16 bytes al final de la negociació per confirmar que heu llegit correctament la petició de negociació del client.

- La negociació o handshake es construeix concatenant els números que acompanyen al primer camp, i dividint pel nombre d'espais en blanc en el propi camp. Això mateix es repeteix per al segon camp. Els dos nombre resultants es concatenen entre si i amb els 8 bytes que van després dels camps. El resultat final és una suma MD5 de la cadena concatenada.

PROTOCOL

- Aquesta negociació pot semblar-se a la negociació HTTP, però no és així. Permet al servidor interpretar part de la petició de negociació com HTTP i llavors canviar a WebSocket.

- Un cop establerta, les trames WebSocket de dades poden començar a enviar-se en tots dos sentits entre el client i el servidor en mode full-duplex. Les trames de text poden ser enviades en mode full-duplex també, en ambdues direccions al mateix temps. La informació es segmenta en trames d'únicament 2 bytes. Cada trama comença amb un byte 0x00, acaba amb un byte 0xFF, i conté dades UTF-8 entre ells. Trames de dades binàries no contemplades encara en l'API. Les trames WebSocket de text utilitzen un terminador, mentre que les trames binàries utilitzen un prefix de longitud.

- Aquesta negociació pot semblar-se a la negociació HTTP, però no és així. Permet al servidor interpretar part de la petició de negociació com HTTP i llavors canviar a WebSocket.

- Un cop establerta, les trames WebSocket de dades poden començar a enviar-se en tots dos sentits entre el client i el servidor en mode full-duplex. Les trames de text poden ser enviades en mode full-duplex també, en ambdues direccions al mateix temps. La informació es segmenta en trames d'únicament 2 bytes. Cada trama comença amb un byte 0x00, acaba amb un byte 0xFF, i conté dades UTF-8 entre ells. Trames de dades binàries no contemplades encara en l'API. Les trames WebSocket de text utilitzen un terminador, mentre que les trames binàries utilitzen un prefix de longitud.

SOCKET.IO

- Socket.IO és una biblioteca JS per a aplicacions web en temps real. Consta de dues parts: una biblioteca de client que s'executa en el navegador, i una biblioteca al servidor per Node.js. Tots dos components tenen una API de gairebé idèntica. Com Node.js, està orientat a esdeveniments. http://socket.io/

- Socket.IO utilitza el protocol WebSocket, però si cal pot utilitzar mètodes, com ara sòcols Flash, polling (sondeig) JSONP i sondeig AJAX llarg, amb la mateixa interfície.

- Proporciona moltes més característiques, incloent la radiodifusió a múltiples sòcols, l'emmagatzematge de dades associades amb cada client, i E/S asíncrona.

- npm install socket.io

SERVIDOR NODE AMB SOCKET IO

JESUÏTES educació
DAWM06UF4 sergi.grau@fje.edu

servidor.js

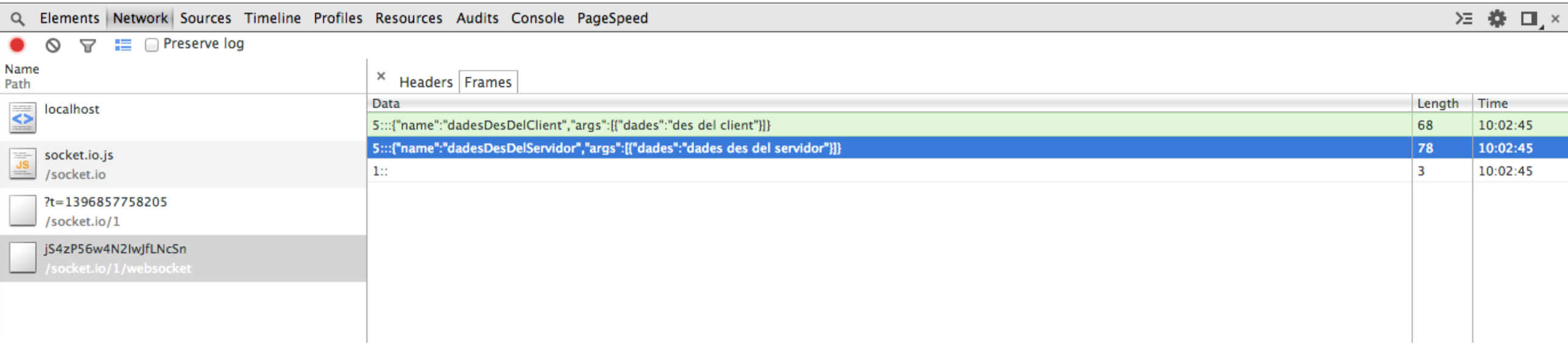
```
var app = require('http').createServer(handler)
  , io = require('socket.io').listen(app)
  , fs = require('fs')

app.listen(8888);
function handler (req, res) {
  fs.readFile(__dirname + '/index.html',
  function (err, data) {
    if (err) {
      res.writeHead(500);
      return res.end('Error loading index.html');
    }
    res.writeHead(200);
    res.end(data);
  });
}
io.sockets.on('connection', function (socket) {
  socket.emit('dadesDesDelServidor', { dades: 'dades des del servidor' });
  socket.on('dadesDesDelClient', function (data) {
    console.log('SERVIDOR -> dades rebudes del client->'+data.dades);
  });
});
```

CLIENT JS AMB SOCKET IO

JESUÏTES educació
DAWM06UF4 sergi.grau@fje.edu

client.js

```
var socket = io.connect('http://localhost:8888');
    socket.on('dadesDesDelServidor', function(data) {
      console.log('CLIENT -> dades rebudes del servidor-
          >'+data.dades);
      socket.emit('dadesDesDelClient', {
        dades : 'des del client'
      });
    });
```



- npm install socket.io

- http://socket.io/

BIBLIOGRAFIA

+ http://www.w3.org/TR/websockets/
+ http://caniuse.com/websockets
+ http://www.websocket.org/demos.html
+ https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
+ https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers