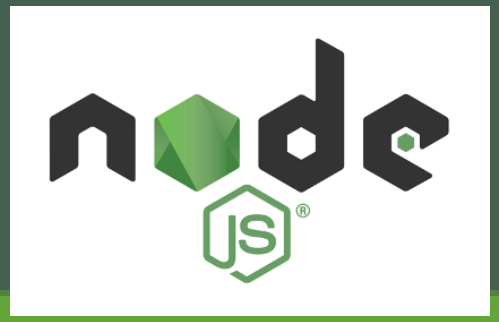


socket.io

Contenido

- WebSocket
- socket.io

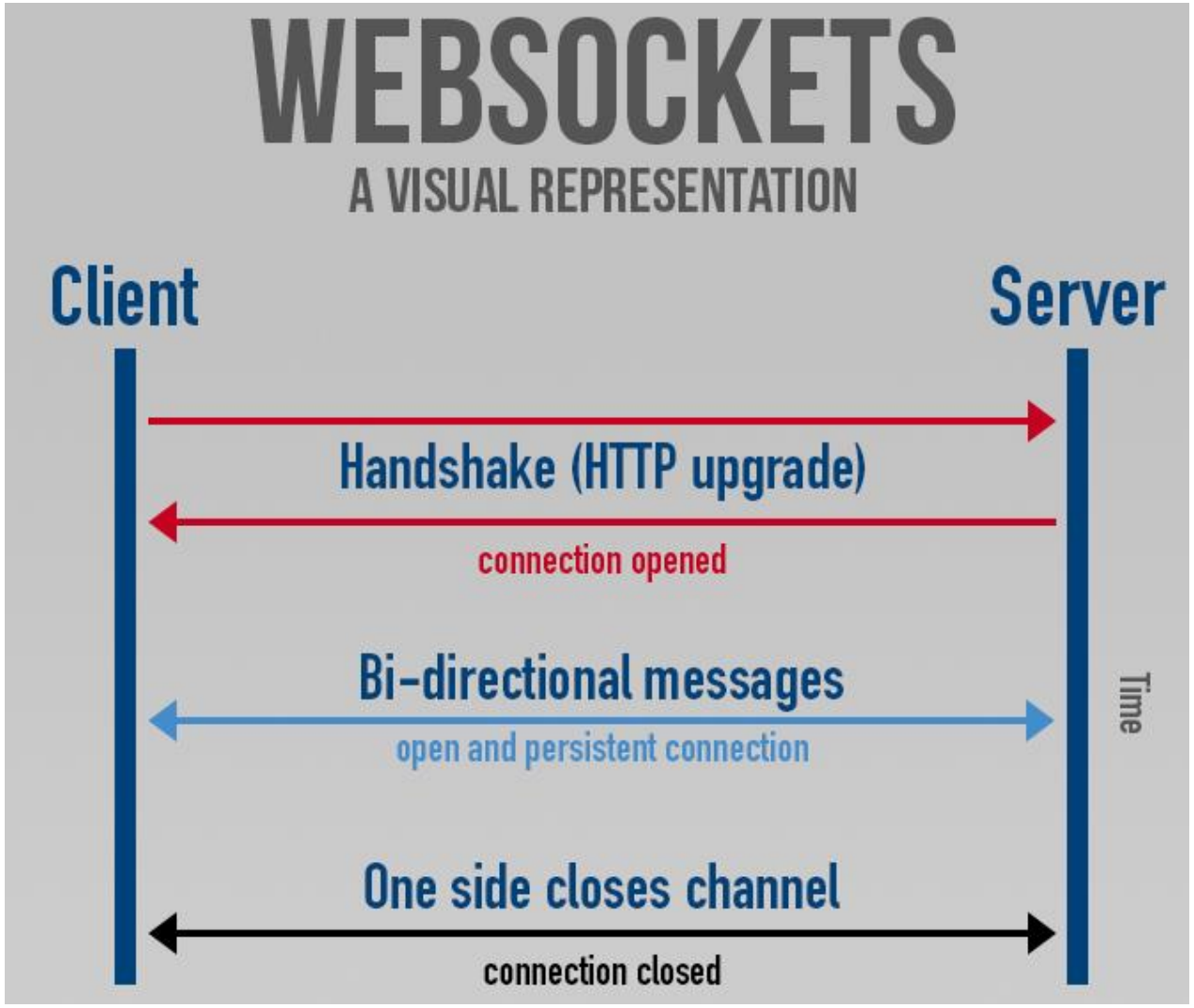
Introducción

- Dar respuesta al problema de subir cambios del servidor a los clients
- Asegurar la actualización en tiempo real de las aplicaciones clients
- Alternativas:
  - **AJAX** - request → response. Crea una conexión al servidor, envía una cabeceras de peticiones y recibe una respuesta del servidor. Después Cierra la conexión
  - **Long poll** - request → wait → response. Crea una conexión al servidor como AJAX, pero la mantiene viva durante un tiempo. Durante la conexión el cliente recibe datos del servidor. El cliente se va reconectando periódicamente. En la parte del servidor la petición es HTTP, aunque la respuesta puede ser en el momento o postpuesta según la lógica de la aplicación

WebSocket

- Tecnología que proporciona un canal de comunicación **persistente**, **bidireccional** y **full-duplex** sobre un único socket TCP
- WebSocket del lado del cliente aparece en HTML5
- Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse en cualquier tipo de aplicación cliente/servidor.
- Proporciona una solución al bloqueo de puertos diferentes del 80, proveyendo una funcionalidad similar a la apertura de varias conexiones en distintos puertos, pero multiplexando diferentes servicios WebSocket sobre un único puerto TCP (a costa de una pequeña sobrecarga del protocolo).
- En el lado del cliente, WebSocket está implementado en Mozilla Firefox > 8, Google Chrome > 4 y Safari > 5, así como la versión móvil de Safari en el iOS 4.2.1.

WebSocket


WEBSOCKETS
A VISUAL REPRESENTATION
Client
Server
Handshake (HTTP upgrade)
connection opened
Bi-directional messages
open and persistent connection
One side closes channel
connection closed
Time

Socket.io

- Facilita la comunicación en tiempo real, bidireccional y basada en eventos entre el navegador y el servidor. Consiste en:
  - Un servidor Node.js
  - Un cliente Javascript para el navegador
- "Socket.io no es una implementación de WebSocket"
- Socket.io utiliza WebSocket como transporte pero añade metadatos a cada paquete: tipo de paquete, namespace y el id de reconocimiento del mensaje
- Clientes y servidores WebSocket no son compatibles

Socket.io: Características

- Confiabilidad
  - La conexión se establece incluso en presencia de proxies o firewalls.
- Soporte de auto-reconocimiento
  - Un cliente desconectado intenta reconectarse de manera continua hasta que el servidor está disponible nuevamente
- Detección de desconexión
  - Un mecanismo de pings (heartbeat) permite al servidor y al cliente saber cuando uno de los dos no está respondiendo
- Soporte binario
  - Se puede emitir ArrayBuffer, Blob (navegador) o Buffer (Node.js)
- Soporte multiplexado
  - Se pueden crear varios Namespaces que actúan como canales separados, sobre una misma conexión
- Soporte room
  - Sobre cada namespace se pueden definir canales arbitrarios (Rooms) a los cuales se pueden unir los sockets. Los mensajes se emiten a un room determinado llegando a los sockets adheridos al room

Socket.io: Instalación

- Instalar socket.io en el servidor: `npm install --save socket.io`
- Se puede utilizar un cliente desde socket.io: `npm install --save socket.io-client`

Socket.io: Configuración del servidor

- Para instalar socket.io es necesario tener instanciado un servidor http
- Se carga el módulo de socket.io y se pasa el servidor como parámetro
- Para gestionar si se está produciendo una nueva conexión de websockets al servidor se añade un listener al evento 'connection' sobre sockets.io
- La conexión devuelve un stream socket (igual que la conexión sobre TCP) que se utiliza para gestionar la conexión con el cliente
- Se utiliza express para gestionar las rutas de las peticiones http al servidor.
- En este caso se carga el cliente en el navegador con res.sendfile(__dirname + '/index.html');

```
const app = require('express')();
const server = require('http').Server(app);
const io = require('socket.io')(server);

server.listen(80);
// WARNING: app.listen(80) no funciona

app.get('/', function (req, res) {
res.sendFile(__dirname + '/index.html');
});

// Eventos en servidor:
'connection','anything','disconnect','message')
io.on('connection', function (socket) {
socket.emit('news', { hello: 'world' });
socket.on('my other event', function (data) {
console.log(data);
});
});
```

Socket.io: Configuración del cliente

- El cliente se puede configurar en el mismo servidor a partir del cliente de socket.io ("/socket.io/socket.io.js">)
- También puede configurarse mediante cnd:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js"></script>
```

- O también instalando el package **socket.io-client** para otros servidores nodeJS

```
<script src="/socket.io/socket.io.js"></script>
<script>
var socket = io.connect('http://localhost:80');
socket.on('news', function (data) {
document.write(`${JSON.stringify(data)}`);
socket.emit('my other event', { my: 'data' });
});
</script>
```

Socket.io: Namespaces

- Los *namespaces* permiten asignar diferentes *endpoints* o *rutas* a un socket
- Utilidad:
  - Minimiza el número de recursos (conexiones TCP)
  - Separa conceptos entre canals de comunicación
- Namespace por DEFECTO: "/"
- Se pueden personalizar:

```
var nsp = io.of('/my-namespace');
nsp.on('connection', function(socket){
console.log('someone connected');
});
nsp.emit('hi', 'everyone!');
```

- El cliente se conecta a la ruta expuesta:

```
var socket = io('/my-namespace');
```

Socket.io: Namespaces

- Los *namespaces* permiten asignar diferentes *endpoints* o *rutas* a un socket
- Utilidad:
  - Minimiza el número de recursos (conexiones TCP)
  - Separa conceptos entre canals de comunicación
- Namespace por DEFECTO: "/"
- Se pueden personalizar:

```
var nsp = io.of('/my-namespace');
nsp.on('connection', function(socket){
console.log('someone connected');
});
nsp.emit('hi', 'everyone!');
```

- El cliente se conecta a la ruta expuesta:

```
var socket = io('/my-namespace');
```

Socket.io: Rooms

- Son canals de emission definidos dentro de cada namespace. Los sockets pueden unirse o salir.
- **Joining**
  - Se utiliza socket.join para subscribirse al canal:

```
io.on('connection', function(socket){
socket.join('some room');
});
```

  - Y se utiliza **to** o **in** para emitir un mensaje:

```
io.to('some room').emit('some event');
```

- **Default Room**
  - Cada socket está identificado por un id único: Socket#id. Y cada socket se subscribe automáticamente al canal (room) identificado por su id

```
io.on('connection', function(socket){
socket.on('say to someone', function(id, msg){
socket.broadcast.to(id).emit('my message', msg);
});
});
```

Socket.io: Real-time API

- En el marco de Desarrollo de una API RESTful la emission de eventos debería recaer en el servidor y no en el cliente.
- El servidor difunde los mensajes a los clientes
- Ej.:

```
const app = require('express')();
const server = require('http').Server(app);
const io = require('socket.io')(server);

app.post('/webhook/orders/updated', function(req, res, next) {
io.sockets.emit('order', "Order Id " + req.body.data.id + " :
Updated");
});
```

Ejercicio 28

- Implementar comunicación vía socket.io en el ejercicio 22. Cada vez que se inserte un nuevo documento en la colección student debe verse en el navegador de un cliente.
- BONUS: configurar un segundo cliente en un servidor aparte.