



Curso de Back-End con Node.js (Avanzado)

Clase 08



Temario



Temario

- WebSockets & socket.IO
 - WebSockets.
 - Socket.IO.
 - Integración en Express.
 - Establecimiento de conexión.
 - Emisión y difusión de mensajes.
- Ejercicios.



Aplicaciones Real-Time



Aplicaciones Real-Time

Problema: ¿Cómo hace un cliente para enterarse si “*hay algo nuevo*” en el servidor? Por ejemplo: ¿cómo sabe la aplicación de Front-End de Gmail que existe un email nuevo en el servidor? En definitiva, ¿cómo podemos construir aplicaciones *real-time*?

Una posible solución es que el cliente realice *requests* cada cierta cantidad de tiempo (ej: cada 2 segundos), consultando por datos nuevos.

Esta técnica se conoce como *long-polling*. Si bien funciona, no es una buena práctica dado que tiene muy mala performance (y no es elegante).

Una solución mucho mejor se logra utilizando *WebSockets*, y la cual veremos a continuación.

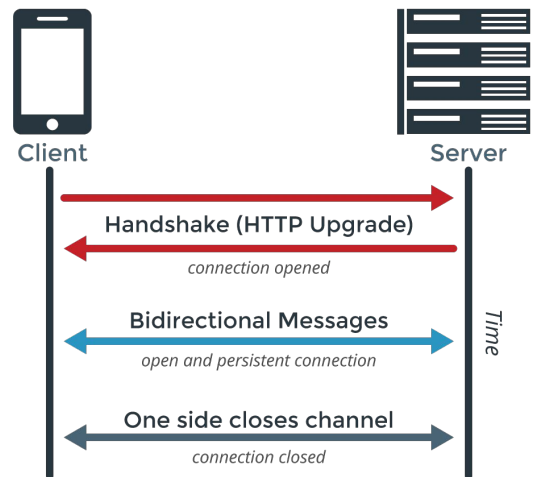


WebSockets

WebSockets

WebSocket es un protocolo **bidireccional** de comunicación cliente-servidor para permitir flujos en tiempo real, que utiliza una única conexión. Es un estándar.

En comparación, los protocolos HTTP y HTTPS son unidireccionales. Todos los *requests* son iniciados por el cliente, y cada *request* tiene su *response* correspondiente.






Socket.io

Socket.IO

[Socket.IO](#) es una pequeña librería para **simplificar y estandarizar ciertas interacciones** relativas a WebSockets.

Además, Socket.IO provee una serie de *fallbacks* para permitir el uso de la librería (de forma transparente) en navegadores viejos que no cuentan con implementación de WebSockets.

 Socket.IO provee tanto de una librería para el lado del servidor como para el lado del cliente. Son librerías distintas → Tener especial atención al momento de instalarlas.



Socket.IO: Integración en Express (1/2)

Integración de Socket.IO en una aplicación de Express:

En index.js o server.js:

```
const express = require("express");
const socket = require("socket.io");
const app = express();

const server = app.listen(3000, () =>
  console.log("¡Servidor corriendo en el puerto 3000!");
);

const io = socket(server);

io.on("connection", () => console.log("¡Usuario conectado mediante socket!"));
```



Socket.IO: Integración en Express (2/2)

Por un tema de orden, el código anterior se puede *refactorear* de la siguiente manera:

En index.js o server.js:

```
const express = require("express");
const socket = require("./socket");
const app = express();
const server = app.listen(3000, () => console.log("¡Servidor corriendo en el puerto 3000!"));
socket(server);
```

En socket.js:

```
const socket = require("socket.io");
module.exports = (server) => {
  const io = socket(server);
  io.on("connection", () => console.log("¡Usuario conectado mediante socket!"));
};
```



Socket.IO: Establecimiento de conexión

Para **establecer conexión** con un socket, también usamos la librería Socket.IO pero en su versión cliente.

En el navegador, por defecto, no tenemos un “manejador de paquetes” ni nada similar. A los efectos de demostrar el funcionamiento, dependeremos de la versión servida en un CDN.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.1.3/socket.io.js"></script>

<script>

  var socket = io.connect("http://localhost:3000");

</script>
```



Socket.IO: Emisión de mensajes

Cuando hablamos de **emisión** de mensajes nos referimos a los mensajes entre **un cliente** y el servidor, a través de **un socket** (existe un socket por cliente conectado).

Para hacerlo:

```
// Cliente
socket.emit(
  "Nuevo-mensaje",
  "Un mensaje..."
);
```

```
// Servidor
io.on("connection", (socket) => {
  socket.on(
    "Nuevo-mensaje",
    (msg) => console.log("message: " + msg)
  );
});
```



Socket.IO: Difusión de mensajes

Cuando hablamos de **difusión** (*broadcast*) de mensajes nos referimos a los mensajes enviados desde el servidor a **todos los clientes** conectados.

Para hacerlo:

```
// Servidor
io.emit(
  "New-message",
  "Hola desde server"
);
```

```
// Cliente
socket.on("new-message", function (msg) {
  console.log("message: " + msg);
});
// > Un mensaje...
```



Ejercicio



Ejercicio (1/2)

Crear una app de Express que:

- Imprima en consola cuando se establece una conexión socket.
- Imprima en consola cuando recibe un evento de tipo `new-message` con el contenido del mismo y los difunde al resto de los usuarios conectados al mismo socket.
- Imprima en consola cuando un usuario se desconecta (documentación).

Crear una app de Front-End (HTML, CSS y JS) que:

- Se conecte a un Websocket.
- Envíe eventos `new-message`, cuyo contenido es introducido en un `input` por el usuario (incluir un botón llamado “Enviar”). Al hacer esto se debe reenviar el mensaje hacia el resto de los usuarios conectados.
- Cuando detecta nuevos mensajes, los agrega a la pantalla (DOM).



Ejercicio (2/2)

Opcionales:

- Agregar soporte para nombres de usuario.
- Agregar fecha y hora en la que se envió el mensaje.
- Informar cuando alguien se conecta o desconecta, en el cliente.
- Persistir mensajes más allá de la sesión usando MongoDB y Mongoose.