Websockets

Comunicación bidireccional en aplicaciones web cliente-servidor

Sonsoles López Pernas

HTTP

https://www.flickr.com/photos/psd/4967856349 by Paul Downey

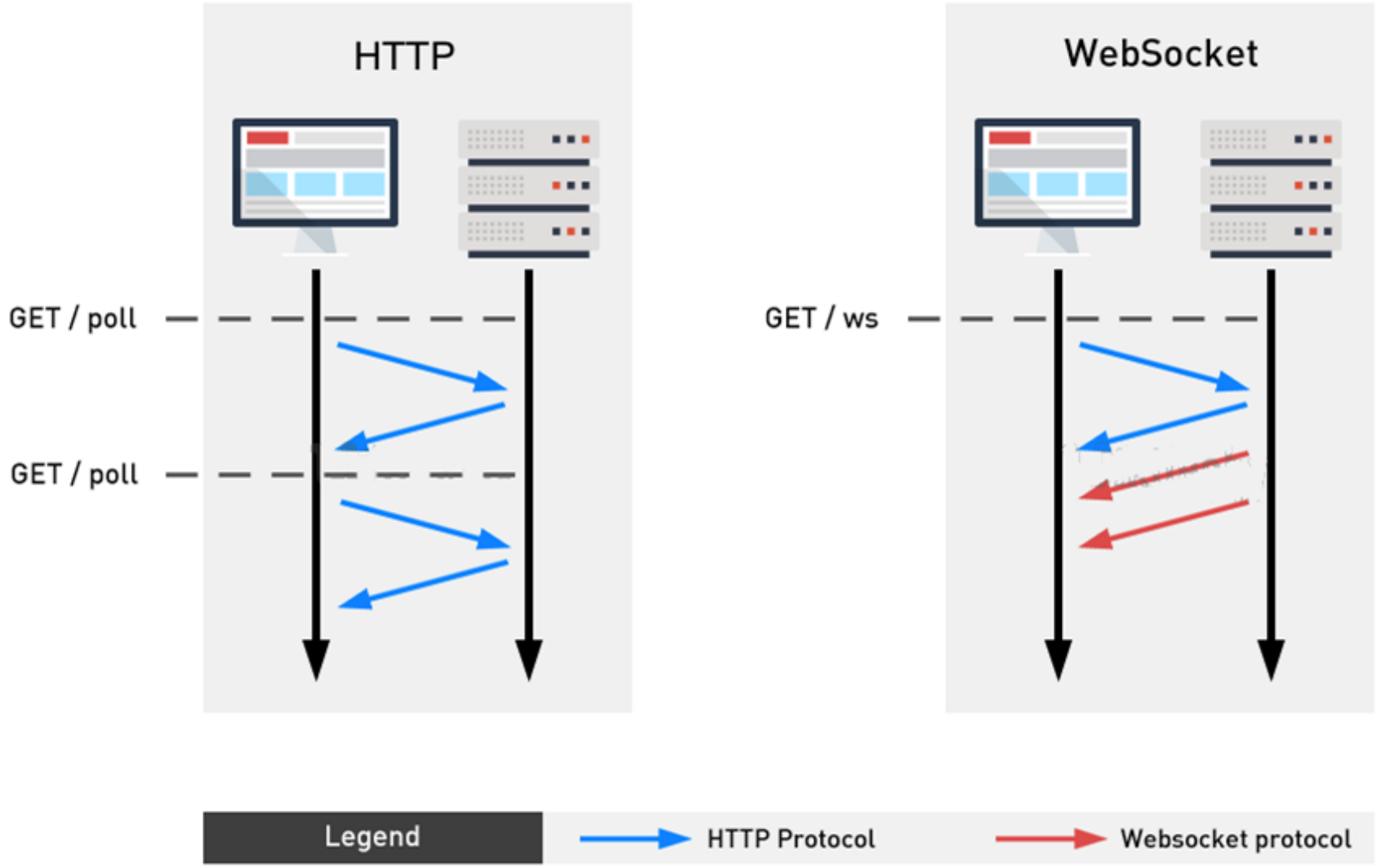
- El cliente siempre inicia la conexión
- El servidor solo puede mandar información en respuesta a una petición
- Múltiples conexiones TCP
- Mensajes de gran tamaño

¿Qué pasa si necesitamos que el servidor inicie la comunicación? Ejemplo: Notificaciones

Websockets

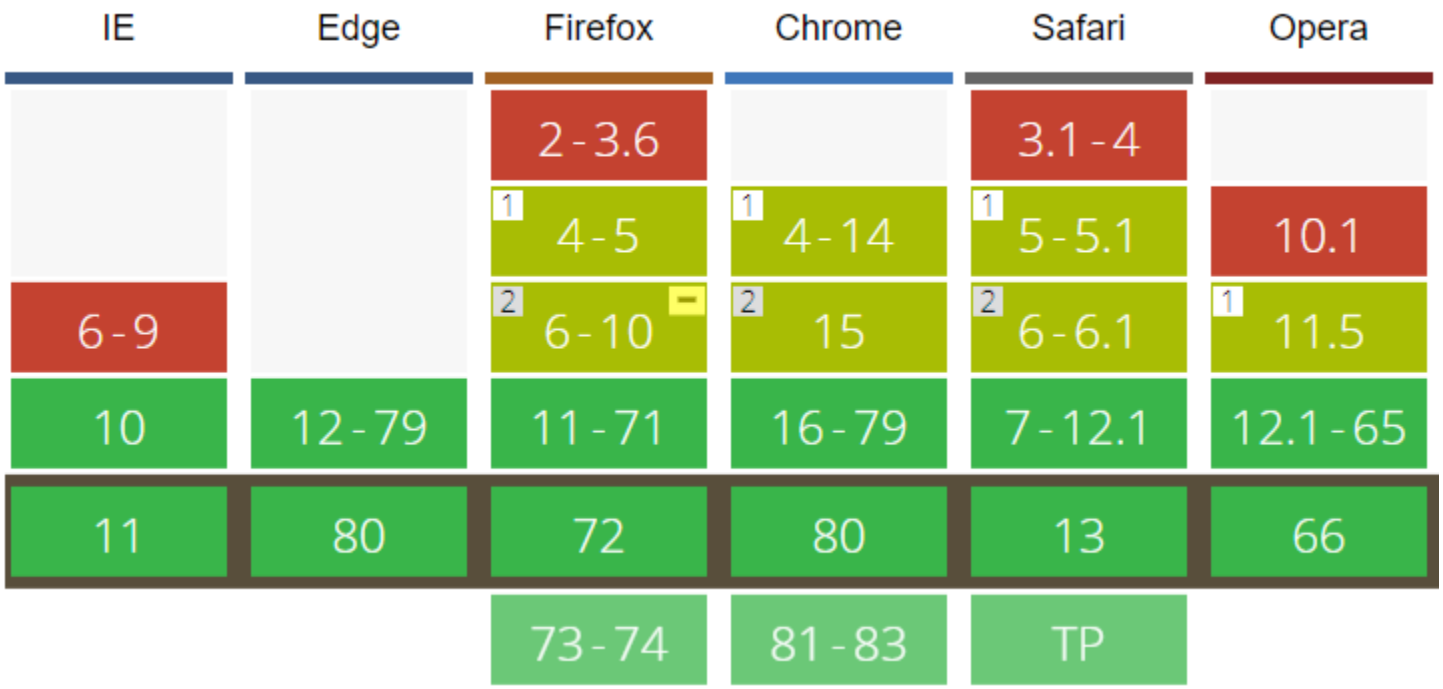
- Tecnología que proporciona un canal de comunicación bidireccional entre cliente y servidor
- El cliente establece una única conexión inicial vía HTTP
- Una vez se ha establecido la conexión, se abre un canal de comunicación en el que tanto el servidor como el cliente pueden enviar mensajes utilizando el protocolo WebSocket
- Se usa una única conexión TCP
- Mensajes ligeros

HTTP vs. WebSocket



Soporte de navegadores

- Aunque el protocolo WebSocket es bastante actual, se trata de una especificación madura.
- Todos los navegadores modernos soportan WebSockets



Ejemplos de usos de Websockets

- Notificaciones
- Chat
- Edición colaborativa (Google Drive, Trello, etc.)
- Juegos online multijugador
- Geolocalización

socket.io

Socket.io es una librería que facilita el uso de Websockets en aplicaciones web cliente-servidor. Tiene dos partes:

- Un **servidor** que se integra con un servidor HTTP implementado con node.js: socket.io
- Una librería de **cliente** que se integra en el navegador: socket.io-client

Además, provee un fallback a HTTP para navegadores que no soporten WebSockets.

Añadir socket.io a una aplicación web

Servidor (node.js)

Añadir dependencia

```
$ npm install --save socket.io
```

Importar dependencia en app.js

```
var io = require('socket.io')(http);
```

Inicializar servidor de socket.io

```
io.on('connection', function(socket){
  console.log('Nueva conexión');
});
```

el objeto socket representa cada cliente que se conecta al servidor

Añadir socket.io a una aplicación web

Cliente

Descargar librería socket.io-client y guardar en directorio público (ej. public/js/socket.io.js)

https://github.com/socketio/socket.io-client/blob/master/dist/socket.io.js

Incluir librería en la página donde deseemos añadir Websockets (ej. index.html)

```
<script      src  ="/js/socket.io.js"            ></script>
```

Inicializar conexión

```
<script>var       socket = io();        </script>
```

Intercambio de mensajes usando socket.io

Enviar

```
socket.emit('msg', 'hello');
```

tipo de mensaje

contenido del mensaje (puede ser JSON, string o contenido binario)

Recibir

```
socket.on('msg', function (msg) {
    console.log('msg: ' + msg);
});
```

tipo de mensaje

Se emplea la misma sintaxis para el servidor que para el cliente. Sin embargo el servidor puede mandar un mensaje tanto a un solo usuario `socket.emit(...)` como a todos los usuarios conectados `io.emit(...)`

Mensajes predefinidos

- El desarrollador se encarga de definir el tipo y contenido de los mensajes que se envían.
- Sin embargo, socket.io tiene una serie de mensajes predefinidos que envía automáticamente y que podemos escuchar para actuar de determinada manera en el caso de que ocurran. Los más relevantes son:

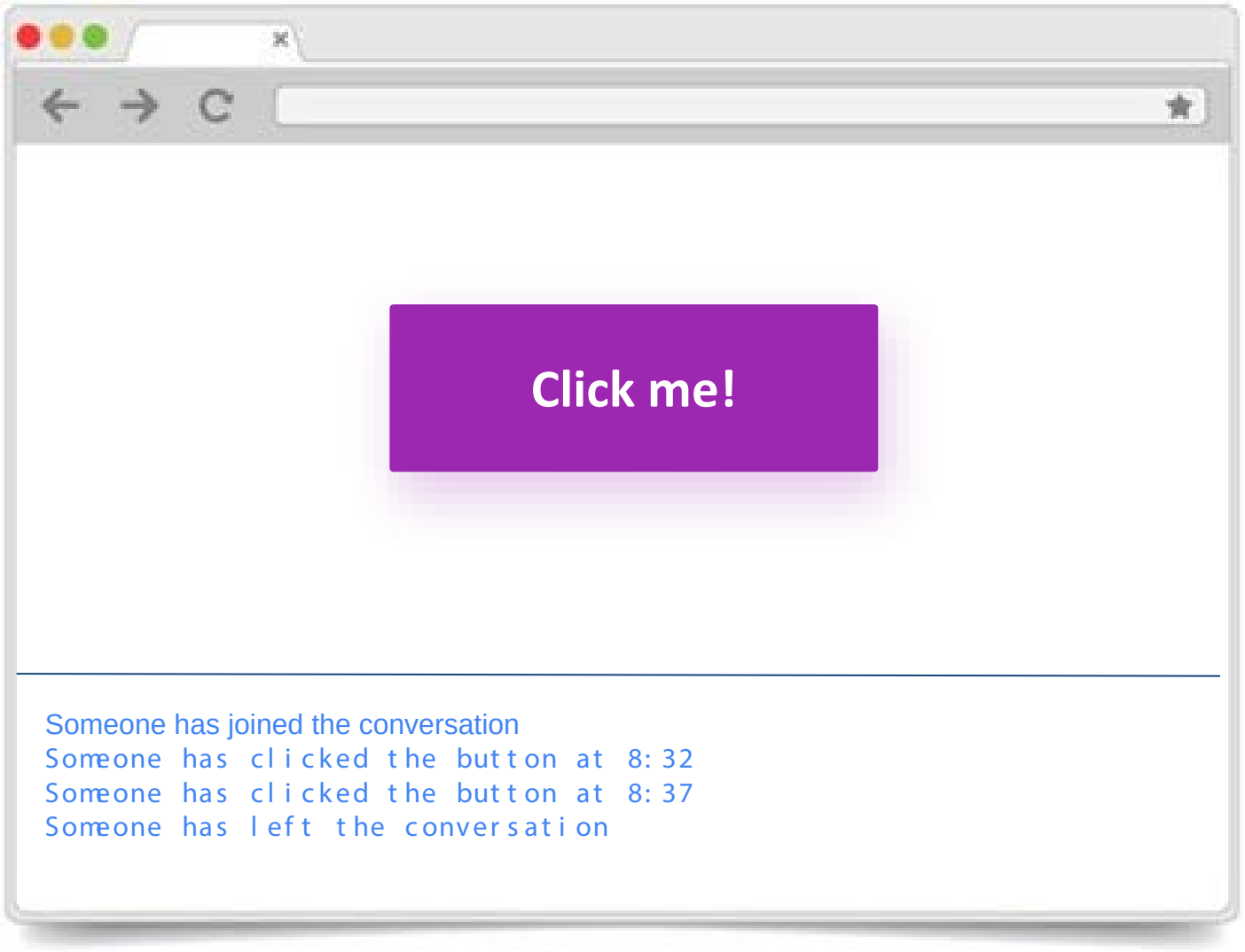
Servidor

- **connection** : Un nuevo cliente se ha conectado al servidor
- **disconnect** : Un cliente se ha desconectado del servidor

Cliente

- **connect:** La conexión al servidor se ha realizado con éxito
- **disconnect:** Se ha terminado la conexión con el servidor

Ejemplo: Click colaborativo



Ejemplo: Click colaborativo (servidor)

index.js

```
var express = require('express');
var app = express();   var path = require('path');
var http = require('http').Server(app);
var io = require('socket.io')(http);
app.use(express.static(path.join(__dirname, "public")));

io.on('connection', function(socket){
    io.emit('new_connection');
    socket.on('disconnect', function(msg){
        io.emit('new_disconnection');
    });
    socket.on('click', function(time){
        io.emit('new_click', time);
    });
});
http.listen(3000);
```

Recibir: Nueva conexión

Enviar: Avisamos a todos de que alguien se ha conectado

Recibir: Alguien se ha desconectado

Enviar: Avisamos a todos de que alguien se ha desconectado

Recibir: Alguien ha hecho click en el botón

Enviar: Avisamos a todos de que alguien ha hecho click

Ejemplo: Click colaborativo (cliente)

public/index.html

```
<!doctype html>
<html>
  <head>
    <title>Socket.IO chat</title>
    <script src="...../socket.io.dev.js"></script>
    <script src="...../jquery-1.11.1.js"></script>
   </head>
   <body>
    <button id="clickme">Click me!</button>
    <script>
      $(function () {
        const socket = io.connect('http://localhost:3000/');

        socket.on('new_connection', function(){
          console.log('Someone has joined the conversation');
        });
        socket.on('new_disconnection', function(){
          console.log('Someone has left the conversation');
        });
        $('#clickme').on('click', function(){
          socket.emit('click', new Date());
        });
        socket.on('new_click', function(time){
          console.log('Someone has clicked the button at ' + time);
        });
      });
    </script>
  </body>
</html>
```

Establecer conexión

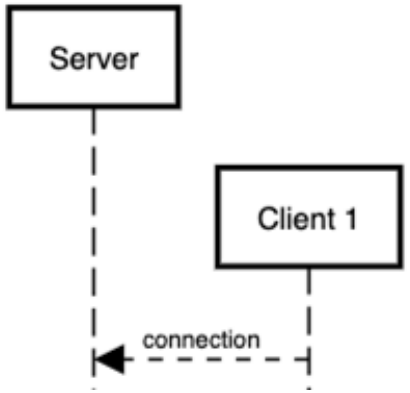
Recibir: Alguien se ha conectado

Recibir: Alguien se ha desconectado

Enviar: Click en el botón

Recibir: Alguien ha hecho click en el botón

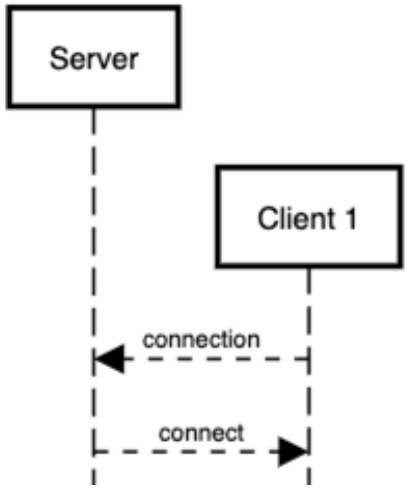
Ejemplo: Click colaborativo - Mensajes (1)



Cuando un cliente se conecta al servidor, se envía automáticamente el evento connection .

```
io.on('connection',
      function(socket){. . .})
```

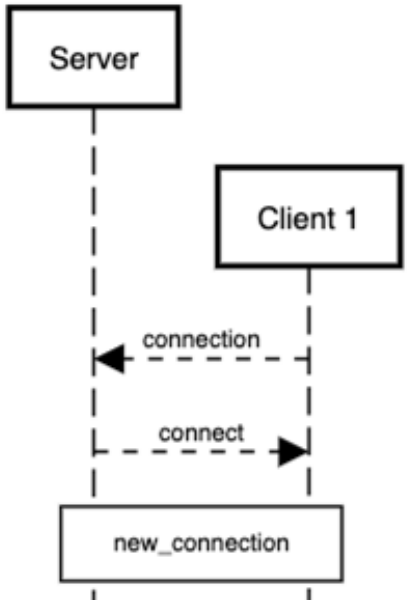
Ejemplo: Click colaborativo - Mensajes (2)



Si la conexión ha tenido éxito, el servidor le responde al cliente con el evento `connect`

Tanto el evento `connection` como el evento `connect` los genera automáticamente la librería socket.io

Ejemplo: Click colaborativo - Mensajes (3)

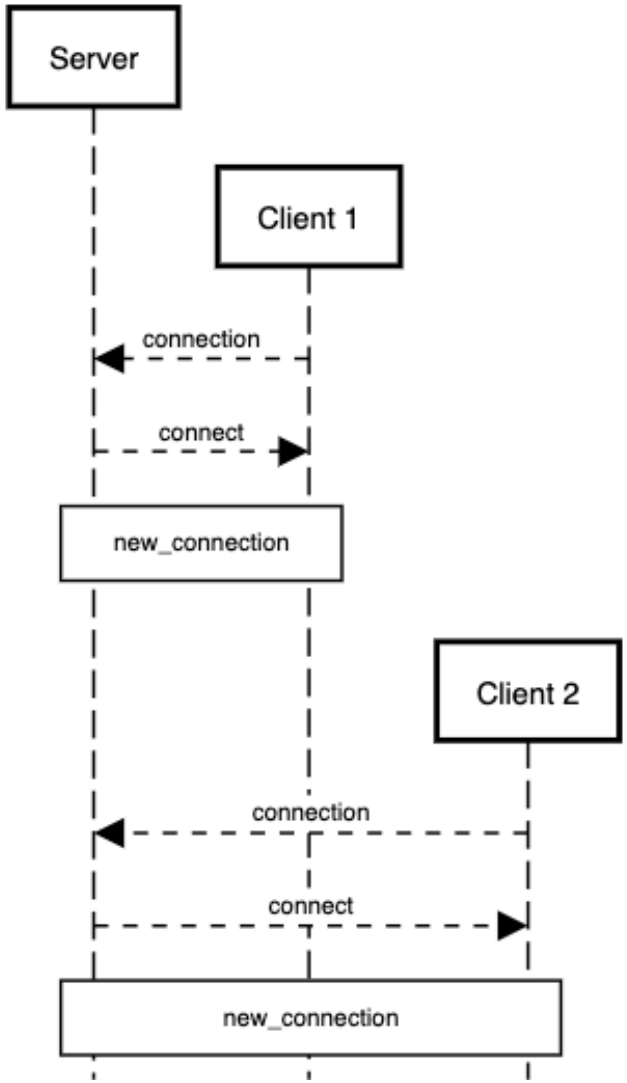


Cuando se conecta un cliente nuevo, el servidor avisa a todos los clientes conectados de que hay un nuevo cliente.

```
io.emit('new_connection');
```

Como solo hay un cliente conectado, sólo lo recibirá éste.

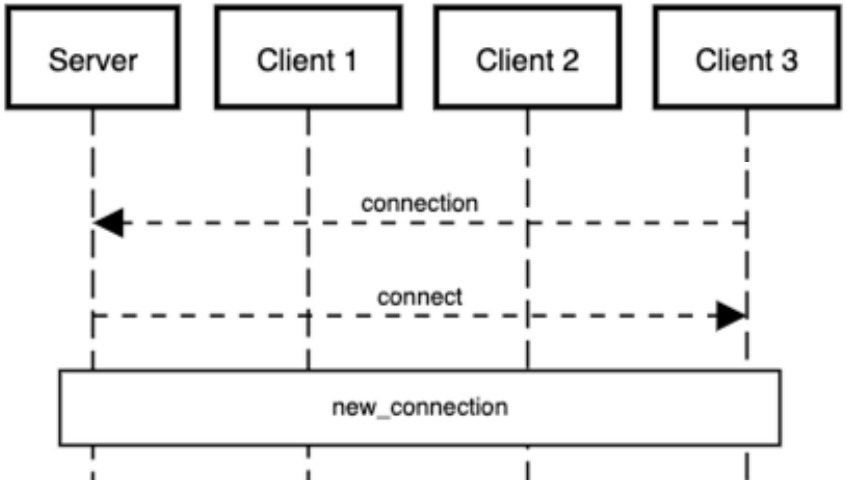
Ejemplo: Click colaborativo - Mensajes (4)



Si se conecta un cliente nuevo se repite el mismo proceso.

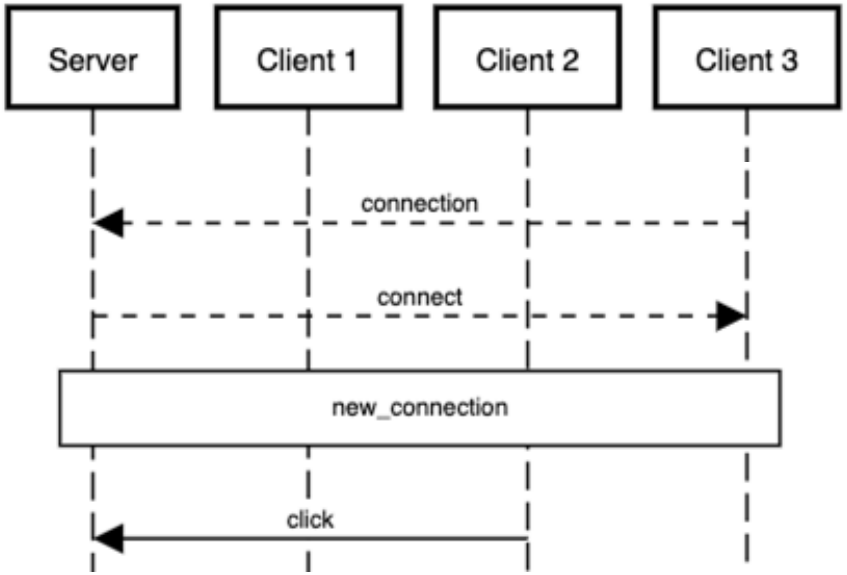
Esta vez se avisa de la nueva conexión tanto al Cliente 1 como al Cliente 2 a través del mensaje new_connection

Ejemplo: Click colaborativo - Mensajes (5)



Si se conecta un tercer cliente, se avisará a los tres de la nueva conexión a través del mensaje new_connection

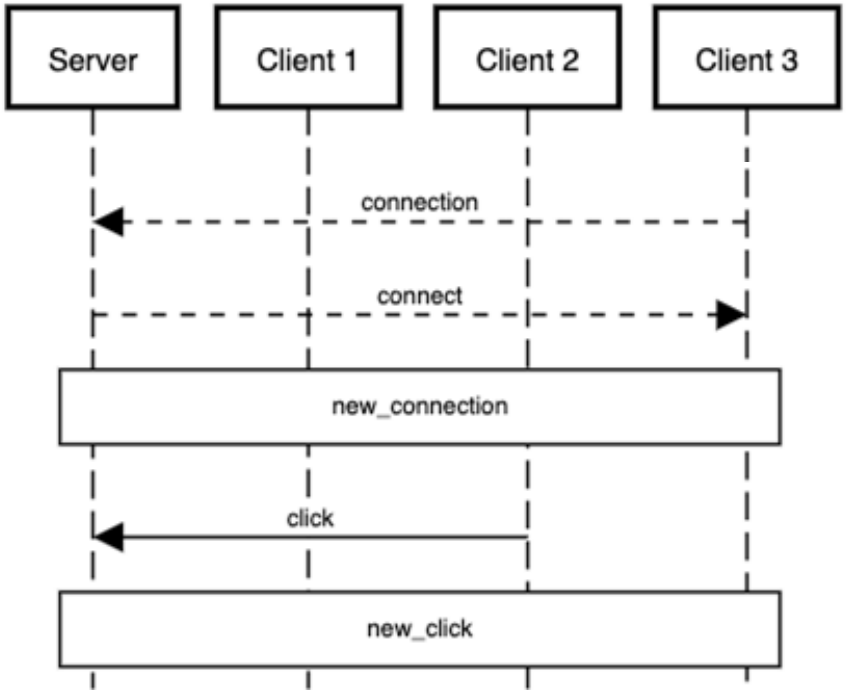
Ejemplo: Click colaborativo - Mensajes (6)



A continuación el cliente 2 hace click sobre el botón con id “clickme”

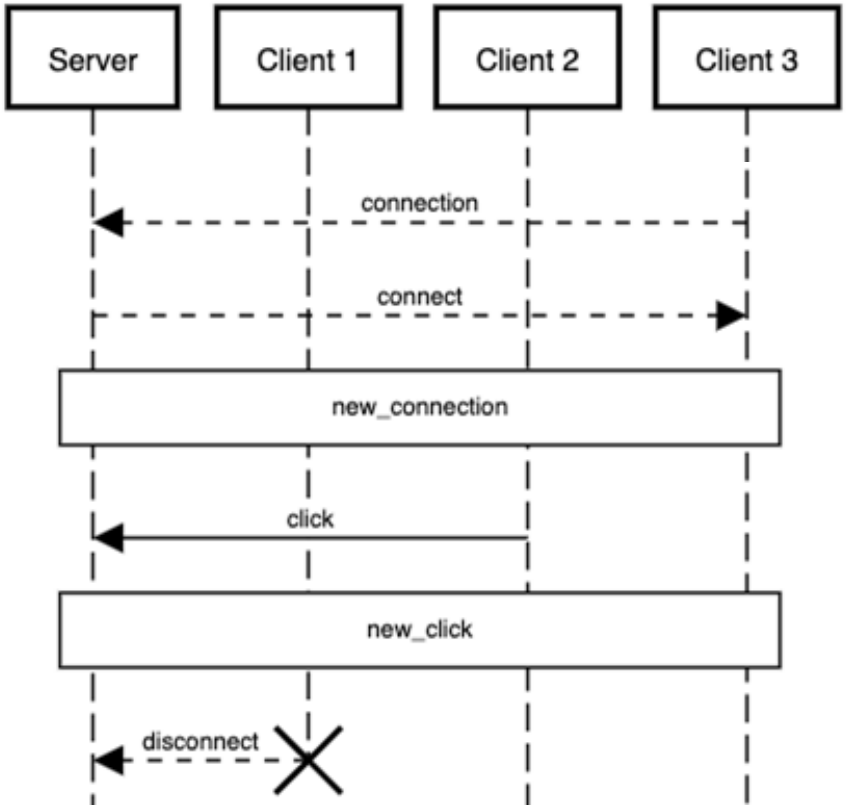
Acto seguido se envía un mensaje al servidor del tipo click

Ejemplo: Click colaborativo - Mensajes (7)



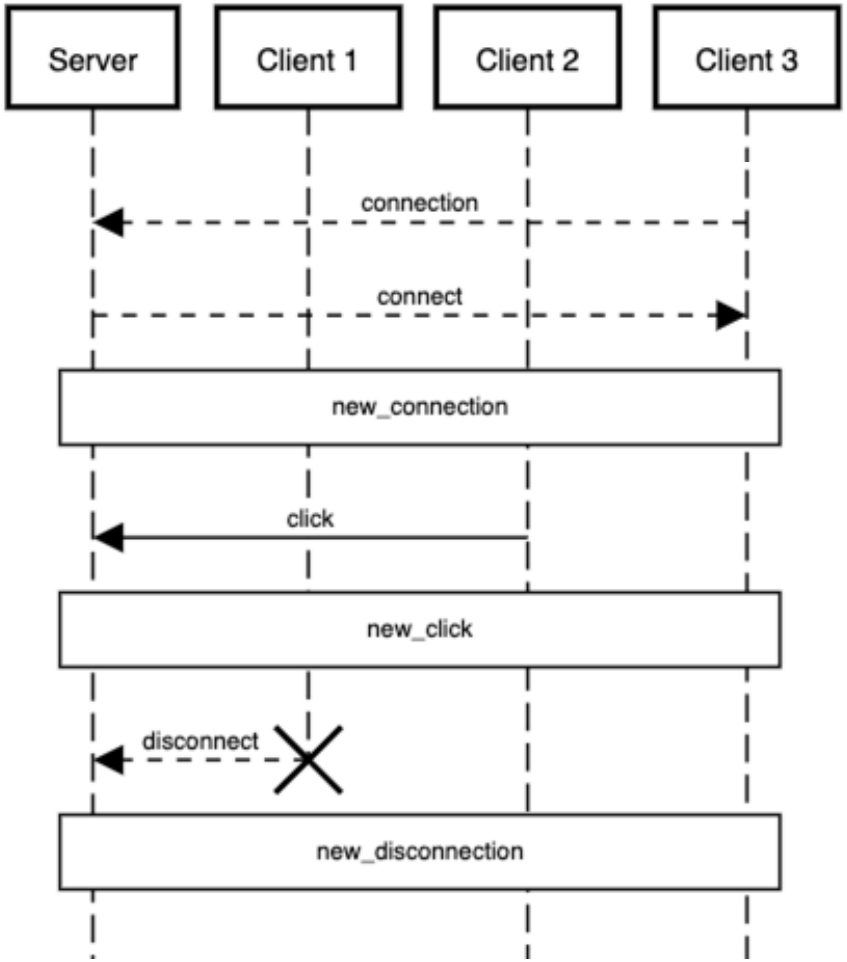
El servidor reconoce el mensaje click y avisa a todos los clientes de que alguien ha hecho click en el botón mediante el mensaje new_click

Ejemplo: Click colaborativo - Mensajes (8)



El cliente 1 cierra la página web, lo que provoca que se reciba el mensaje disconnect en el servidor

Ejemplo: Click colaborativo - Mensajes (9)



Por último, el servidor avisa a todos los clientes que quedan de que alguien se ha desconectado

Websockets

Comunicación bidireccional en aplicaciones web cliente-servidor

Sonsoles López Pernas