Socket.io

¿Qué son websockets?

Protocolo de comunicación

- Full-duplex
- Una sola conexión permanente
- Stream de mensajes
- Contenido en tiempo real

¿Qué son websockets?

Es decir...

- El cliente puede enviar y recibir datos en tiempo real
- Orientado a “eventos” (mensajes)
- Siempre conectado
- Baja latencia

Websockets y Node.js

Funcionan especialemente bien con Node.js

- El servidor maneja muchas conexiones simultáneas
- Buena integración con JSON
- Eventos

¿Para qué sirven?

Fundamentalmente, para:

- Actividades colaborativas
- Juegos multijugador
- Acelerar ciertas operaciones
  - Enviar datos
  - Cargar recursos
- En resumen: tiempo real en vez de “a petición”

Socket.io

Vamos a usar Socket.io

- Una librería para manipular websockets
- Muy popular
- Fallback para navegadores obsoletos
- Muy fácil de usar

➡ http://socket.io/

Socket.io

Socket.io tiene dos partes:

- Servidor (Node.js):

```
var express = require("express"),
    server = require("http").createServer(),
    io = require("socket.io").listen(server),
    app = express();

server.on("request", app).listen(3000);
```

- Cliente:

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

Socket.io

Los sockets emiten eventos

- Un evento = un “mensaje”
- Se pueden pasar parámetros
- `socket.on(mensaje, callback)`
- `socket.emit(mensaje, [param1, param2, ...])`

Socket.io

server.js

```
var express = require("express"),
    app = express(),
    server = require("http").createServer(app),
    io = require("socket.io").listen(server);

app.use(express.static(__dirname + "/public"));

io.sockets.on("connection", function(socket) {
  socket.emit("ping");
  socket.on("pong", function() {
    console.log("PONG!");
  });
});

server.listen(3000);
```

Socket.io

server.js

```
var express = require("express"),
    app = express(),
    server = require("http").createServer(app),
    io = require("socket.io").listen(server);

app.use(express.static(__dirname + "/public"));

io.sockets.on("connection", function(socket) {
  socket.emit("ping");
  socket.on("pong", function() {
    console.log("PONG!");
  });
});

server.listen(3000);
```

Socket.io

server.js

```
var express = require("express"),
    app = express(),
    server = require("http").createServer(app),
    io = require("socket.io").listen(server);

app.use(express.static(__dirname + "/public"));

io.sockets.on("connection", function(socket) {
  socket.emit("ping");
  socket.on("pong", function() {
    console.log("PONG!");
  });
});

server.listen(3000);
```

Socket.io

index.html

```
<html>
  <head>
    <script src="/socket.io/socket.io.js"></script>
    <script type="text/javascript">
      var socket = io.connect("http://localhost:3000");

      socket.on("ping", function() {
        console.log("PING!");
        socket.emit("pong");
      });
    </script>
  </head>
  <body></body>
</html>
```

Socket.io

index.html

```
<html>
  <head>
    <script src="/socket.io/socket.io.js"></script>
    <script type="text/javascript">
      var socket = io.connect("http://localhost:3000");

      socket.on("ping", function() {
        console.log("PING!");
        socket.emit("pong");
      });
    </script>
  </head>
  <body></body>
</html>
```

Socket.io

index.html

```
<html>
  <head>
    <script src="/socket.io/socket.io.js"></script>
    <script type="text/javascript">
      var socket = io.connect("http://localhost:3000");

      socket.on("ping", function() {
        console.log("PING!");
        socket.emit("pong");
      });
    </script>
  </head>
  <body></body>
</html>
```

Socket.io

Eventos reservados (servidor):

- `io.sockets.on("connection", cb)`
- `socket.on("message", cb)`
- `socket.on("disconnect", cb)`

Cliente:

- `socket.on("connect", cb)`
- `socket.on("disconnect", cb)`
- `socket.on("error", cb)`
- `socket.on("message", cb)`

Socket.io

Métodos (servidor)

- `socket.broadcast.emit(msg)`
  - les llega a todos menos el emisor
- `socket.disconnect()`
- `socket.emit(msg) / socket.on(msg)`

Métodos (client)

- `var socket = io.connect(host)`
- `socket.disconnect()`
- `socket.emit(msg) / socket.on(msg)`

Un Chat! (simple)

Vamos a hacer un chat sencillo:

- Los usuarios se loguean eligiendo un nick
- Todo el mundo escribe en la misma sala común
- No tenemos indicador de presencia

Un Chat! (simple)

En el cliente:

- `Chat.registerHandler(cb)`: callback cuando el usuario escribe
- `Chat.postMsg(user, msg)`: Muestra un mensaje de otro
- `Chat.showMyMsg(user, msg)`: Muestra un mensaje propio

Donde:

- `user: {avatar: <string>, name: <string>}`
- `msg: {text: <string>, time: <date>}`

Canales

Con Socket.io podemos crear canales o *namespaces* para agrupar los receptores

```
var express = require("express"),
    app = express(),
    server = require("http").createServer(app),
    io = require("socket.io").listen(server);

app.use(express.static(__dirname + "/public"));

io.of("/canal").on("connection", function(socket) {
  socket.emit("ping");
});

server.listen(3000);
```

Canales

En el cliente:

```
<script src="/socket.io/socket.io.js"></script>
<script type="text/javascript">
  var socket = io.connect("http://localhost:3000/canal");

  socket.on("ping", function() {
    console.log("PING!");
  });

</script>
```

Canales

Podemos tener varios canales simultáneos (multiplexando el mismo websocket)

```
io.of("/canal").on("connection", function(socket) {
  socket.emit("ping");
});

io.of("/otro").on("connection", function(socket) {
  socket.emit("bang!");
});
```

Canales

En el cliente:

```
var canal = io.connect("http://localhost:3000/canal"),
    otro = io.connect("http://localhost:3000/otro");

canal.on("ping", function() {
  console.log("PING!");
});

otro.on("bang!", function() {
  console.log("Estoy herido!");
});
```

Ahora, multisala

Utilizando namespaces, los usuarios pueden:

- Loguearse/registrarse (simpleauth)
- Crear salas
- Unirse y salirse de las salas creadas
- Escribir en la sala en la que estén

Ahora, multisala

Consejos:

- Guarda los sockets de cada usuario en un objeto
- Utiliza la sesión (o req.user) para saber en qué sala está un usuario (clave del objeto de sockets + canal)
- Crea mensajes para:
  - Un usuario ha entrado en la sala
  - Un usuario ha salido de la sala
  - Alguien postea un mensaje