SOCKET.IO

Agenda

- Implement connected oriented web applications
- Discuss techniques
- WebSocket Protocol
- Using Socket.IO

The Challenge: A Chat Application

- Polling is expensive
- Long polling is better but still expensive on busy server
- WebSocket is great but
  - Need modern browser
  - No built-in support inside Node.js

Web Socket Protocol

- Enables two-way communication between browser and server
- Does not rely on opening multiple HTTP connections
- Replacement for older techniques like long polling and forever frame
- A simple abstract over TCP socket
- A totally new application protocol
  - No HTTP headers
- Managed by IETF

Web Socket API

- A JavaScript API
- Is used by the browser to initiate a Web Socket communication with the server
- Is all about the WebSocket class
  - send
  - close
  - readyState
  - onopen, onmessage, onclose, onerror
- Managed by W3C

Getting Started

- Create a new WebSocket object
- Specify a URL and sub protocols (Optional)
  - Must use ws or wss protocols
- Register to open and error events

```
var ws = new WebSocket("ws://localhost:5481/socket/connect");

ws.onerror = function (e) {
     console.log("ERROR");
     console.log(e);
}

ws.onopen = function (e) {
     console.log("OEPN");
     console.log(e);
}
```

The Handshake

- Upon a WebSocket object creation the browser sends an Upgrade request to the server

```
GET http://localhost:5481/socket/connect HTTP/1.1
Upgrade: websocket
Connection: Upgrade
Host: localhost:5481
Origin: http://localhost:5481
Sec-WebSocket-Key: rHeA9NhKxIuFX85mxpTOfQ==
Sec-WebSocket-Version: 13
```

- Server must respond with appropriate headers

Server Response


```
HTTP/1.1 101 Switching Protocols
Cache-Control: private
Upgrade: Websocket
Server: Microsoft-IIS/8.0
Sec-WebSocket-Accept: JLSwLlhPkGnb4qOJOnYzl957T7I=
Connection: Upgrade
```

- After a successful handshake, the data transfer part starts
- This is a two-way communication channel where each side can, independently from the other, send data at will

Send and Receive Messages

- WebSocket object supports
  - send – Can only be used after onopen event was fired
  - onmessage

```
var ws = new WebSocket("ws://localhost:5481/api/socket/connect");

ws.onclose = function (e) {
     ...
}

ws.onopen = function (e) {
    ws.send(message);
}

ws.onmessage = function (e) {
     console.log("MESSAGE: " + e.data);
}
```

Framing

- WebSocket is message based protocol
- The data being sent by the client is considered a message
- A message consists of multiple frames
- Each frame has slight overhead over the original payload
  - 2 bytes – FIN + Opcode + Payload Length + More
  - 4 bytes – Masking key
  - Above is true only for messages <= 125 bytes

Socket.IO

- Real time bidirectional event based communication
- Supports browser & node platforms
- Uses WebSocket as the transport protocol
- Supports fallbacks
  - XHR
  - JSONP
  - Long Polling
- Must use Socket.IO on both sides

When to use

- Real time analytics
- Instant messaging
- Chat
- Binary streaming

Getting Started

- Socket.IO is composed of two parts
- socket.io: A server that integrates with Node.js
- socket.io-client: A client library that loads on the browser

Server Side

- yarn add express socket.io

Socket.IO does not connect directly to express

```
const express = require('express');
const http = require('http');
const socketIO = require('socket.io');
const path = require("path");

const app = express();
const server = http.Server(app);
const io = socketIO(server);

io.on('connection', function(socket){
    console.log('A user connected');
});

server.listen(3000, function(){
    console.log('Serve is running on port
3000');
});
```

Client Side

- Include the client library
- Start using the global io object

```
<body>
   <script src="socket.io-client/dist/socket.io.dev.js"></script>
   <script>
      var socket = io();
      socket.emit('chat message', 'hello');
   </script>
</body>
```

Transmission Types


```
// unicast to current client
socket.emit('message', 'hey there client');

// broadcast to all clients including sender
io.sockets.emit('message', 'hello everyone!');

// short form, same thing
io.emit('message', 'hello everyone!');

// broadcast to all clients excluding sender
socket.broadcast.emit('message', 'surprise party');

// unicast by socketid
io.to(socketid).emit('message', 'your eyes only');
```

Namespace

- Allows for grouping of connections
- By default all clients are part of the default namespace
- Define new namespace

```
const ns = io.of("/chat");
ns.on('connection', function(socket){
   socket.on('chat message', function(msg){
      ns.emit('chat message', msg);
   });
});
```

```
var socket = io("/chat");
```

Server vs. client

Rooms

- Within a namespace clients can join/leave rooms
- Rooms are automatically created when a socket join them
- Only the server can add a client to a room

```
ns.on('connection', function(socket){
   console.log('a user connected', socket.id);

   socket.join("room1");
});
```

Rooms

- The server can decide to send messages only to a specific room

```
socket.on('chat message', function(msg){
   ns.in("room1").emit('chat message', msg);
});
```

- As before, the sender can be excluded

```
socket.on('chat message', function(msg){
   socket.broadcast.to("room1").emit('chat message', msg);
});
```

Query Clients


```
// all connected clients
io.sockets.clients((err,clients)=>{});

// all clients in namespace 'final-frontier'
io.of('/final-frontier').clients((err,clients)=>{});

// all users in namespace 'final-frontier' room ‘spock'
io.of('/final-frontier').in("room1").clients((err,clients)=>{});

// all clients in room 'myroom'
io.in("room1").clients((err,clients)=>{});
```

socket.id

- Every connected client has a unique id. For example,

```
/chat#jFWs9corXp_9jOF3AAAA
```

- Might change because of reconnection
- Can use it to maintain a “session” state table

Clustering

- IE < 11 does not support web socket
- Socket.IO will use long polling
- Must use sticky load balancing
- PM2 does not support that ☹
- Use fork_mode with different port per process and good balancer like Ngnix

Passing Event Between Nodes

- Assuming cluster configuration
- You want to broadcast event to everyone
- Need to implement some IPC infra
- Or use socket.io-redis

```
var io = require('socket.io')(3000);
var redis = require('socket.io-redis');
io.adapter(redis({ host: 'localhost', port: 6379 }));
```

Summary

- Very easy to use
- Not to be confused with WebSocket API