Tutorial + Exercício Web Sockets e Server-sent Events

Tutorial Server-sent Events (EventSource)

1. Crie uma pasta chamada sse para este exemplo.

2. Crie um arquivo sse-server.js que será utilizado como aplicação servidora, responsável por enviar dados, colocando o seguinte conteúdo:

```
var http = require("http");
http.createServer(function(req,res){
  res.writeHeader(200,  {"Content-Type":"text/event-stream"
                        , "Cache-Control":"no-cache"
                        , "Connection":"keep-alive"
                        , "Access-Control-Allow-Origin": "*"});

  var interval = setInterval( function() {
  res.write("data: " + randomInt(100,127) + "\n\n");
  },2000);

}).listen(9090);

console.log('SSE-Server started!');

function randomInt (low, high) {
  return Math.floor(Math.random() * (high - low) + low);
}
```

O código acima cria uma conexão que ficará aberta com o cliente, enquanto este estiver conectado. O servidor enviará a cada dois segundos o timestamp como conteúdo da mensagem.

3. Execute o código através da linha de comando: `node sse-server.js`

4. Crie o arquivo sse-client.html que será o cliente de sse-server:

```
<html>
<script language="javascript">
var source = new EventSource("http://localhost:9090");
source.onmessage = function(event) {
    document.getElementById("result").innerHTML = "<h3>Voltagem medida: " +
event.data + "</h3>";
};
</script>
<body>
  <form>
    <div id="result"></result>
  </form>
</body>
</html>
```

O código acima cria uma conexão que ficará aberta com o servidor. A cada mensagem recebida, ele atualizará o conteúdo do div.

5. Abra a página utilizando o browser.

Tutorial Web Sockets

1. Crie uma pasta chamada websockets para este exemplo.

2. Vá até a linha de comando e entre na pasta criada acima. Efetue o download do pacote da biblioteca node.js que encapsula o acesso a websockets e será utiliza em nosso exemplo, digitando: npm install ws

O código acima irá baixar e instalar a biblioteca localmente.

3. Crie o arquivo ws-server.js que será utilizado como aplicação servidora, responsável por enviar dados, colocando o seguinte conteúdo:

```
var WebSocketServer = require('ws').Server;
wss = new WebSocketServer({port: 8080, path: '/testing'});
wss.on('connection', function(ws) {
    ws.on('message', function(message) {
        console.log('Msg received in server: %s ', message);
    });
    console.log('new connection');
    ws.send('Msg from server');
});
```

O código acima criará um WebSocket na porta 8080, escutando no caminho especificado (testing)

4. Execute o código através da linha de comando: node ws-server.js

5. Crie o arquivo ws-client.html que será o cliente de ws-server:

```
<html>
<script language="javascript">
var connection = new WebSocket('ws://localhost:8080/testing');

connection.onopen = function(){
   console.log('Connection open!');
   connection.send('Hey server, whats up?');
}
connection.onclose = function(){
   console.log('Connection closed');
}
connection.onmessage = function(e){
   var server_message = e.data;
   console.log(server_message);
   document.getElementById("result").innerHTML += server_message + "<br>";
}
</script>
<body>
  <form>
    <div id="result"></result>
  </form>
</body>
</html>
```

O código acima cria uma conexão full duplex que ficará aberta com o servidor. A cada mensagem recebida, ele atualizará o conteúdo do div.

6. Abra a página criada acima utilizando o browser

Exercício: Chat com Web Sockets + EventSource

1. Semelhante aos exercícios com as tecnologias vistas anteriormente, escreva um chat em que clientes, através de um browser, usam WebSockets para se conectar a um servidor que media as mensagens entre os diferentes clientes.

Alguns requisitos:
a. O servidor deve pedir ao usuário cliente para digitar um nome de modo que as mensagens de cada usuário possam ser identificadas.
b. As mensagens enviadas por um cliente são replicadas para todos os outros clientes.
Exemplo hipotético de conversação:

```
Alex: Muito bom esse servidor.
Ze: vsf muito complicado programar com websockets
Maria: Pra mim foi facinho. Vcs são enrolados.
Alex: kkkkk
Ze: LOL
```

2. Crie um recurso HTTP (i.e. uma URL) que representará uma visão “read-only” do chat. A implementação deverá utilizar Server-sent events para disponibilizar as mensagens da conversa do chat à medida que vão sendo enviadas pelos clientes que estão conectados ao chat, funcionando como um feed.

Questões para discussão:
Quais as principais diferenças entre esta implementação e a implementação com sockets TCP?
Quais as principais dificuldades com a implementação usando EventSource?