Atividade prática - Web Assíncrona

Server-sent Events (EventSource)

1. Crie uma pasta chamada sse para este exemplo.

2. Crie o 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"></div>
  </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.

6. Efetue alterações na aplicação servidora construída anteriormente no exercício da aula de HTTP/REST para que cada recurso envie atualizações da temperatura através de Server-sent events. Note que existem pequenos detalhes tanto na requisição quanto na resposta (ex: qual tipo de conteúdo o cliente deve requisitar para obter server-sent events?)
Sua URL deverá tanto responder à requisições diretamente no browser como a requisições feitas através do objeto EventSource.
Você terá certeza que está funcionando se adaptar a questão 4 acima para testar o objeto EventSource usando qualquer URL do seu servidor HTTP/REST.

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