**Socket** é um mecanismo de comunicação, usado normalmente para implementar um modelo cliente/servidor, que permite a troca de mensagens entre os processos de uma máquina/aplicação servidor e de uma máquina/aplicação cliente.

**WebSocket** é uma tecnologia muito mais especifica, que permite a comunicação bidirecional entre browsers, clientes e servidores web sobre um único socket por canais full-duplex. Em outras palavras, podemos criar uma conexão persistente entre o cliente e o servidor e ambas as partes podem enviar dados a qualquer momento.

Com WebSocket podemos fugir do paradigma de requisição resposta tradicional da web. Em que um browser cliente sempre faz uma requisição e fica aguardando uma resposta do servidor. Conceito novo é que o próprio servidor pode fazer uma requisição para o cliente (inverter os papeis).

O browser e o servidor deve suportar trabalhar com websocket. O chrome em uma das recentes versões já possui esse suporte e o nodejs (nosso servidor), já tem um modulo para tratar disso. É o **socket.IO.**

Instalando o socket.io no projeto.

- npm install socket.io - -save

Devemos preparar nosso servidor a não apenas responder HTTP, deve responder também a websocket (não tem problema as duas requisições chegarem na mesma porta, pois são protocolos diferentes). Vou deixar a mesmo porta: 3000

- Incluindo modulo do socket.io dentro a aplicação

/\* importar as configurações do servidor \*/

var app = require('./config/server');

/\* importar modulo do Socket.IO \*/

var socketIO = require('socket.io');

/\* parametrizar porta de escuta \*/

var server = app.listen(3000, function () {

console.log('Servidor rodando porta 3000');

});

socketIO.listen(server);

Agora tanto o socket como o http estão escutando na mesma porta 3000

- Abrindo a conexão via websocket do cliente com o servidor

Na view chat.ejs

        <!-- rota importada automaticamente, tem tudo que precisamos do socket do lado do cliente -->

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

- Na view chat.ejs

        <script>

            // instancia do obj que se conecta com websocket

            var socket = io('http://localhost:3000');

        </script>

- No servidor app.js

var io = socketIO.listen(server);

/\* criar a conexão por websocket \*/

// funcao ON vai estar escutando req lado do cliente e vice-versa

// estamos escutando eventos de connection, evento padrão do socket.io (propria conexao por parametro [socket])

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

console.log('Usuario conectou');

socket.on('disconnect', function () {

console.log('Usuario desconectou');

});

});

Fazer o teste se o usuário conectou e desconectou

- Vou dividir a tela em 3 chats. Eliminar os diálogos de exemplo da view.

On(‘nome’, function(data){} );

Emit(‘nome’ {} );

On: Ouvindo pedidos de execução

Emit: Faz um pedido para executar uma ação

Dessa forma é o dialogo tanto de um lado como o do outro.

- No cliente eu vou escutar ‘msgParaCliente’ através da função on()

        <script>

            // instancia do obj que se conecta com websocket

            var socket = io('http://localhost:3000');

            // escutar msgParaCliente

            socket.on('msgParaCliente', function(data){

                alert(data);

            });

        </script>

- No servidor, vou amitir uma mensagem através da função emit(), logo após fazer as validações do chat.js no controller

/\* emit msgParaCliente (websocket)

utilizando obj do socket.io dentro do controller \*/

application.get('io').emit(

'msgParaCliente',

'Teste'

);

- Um fator muito importante é que a variável IO tem que estar global para eu poder ter acesso a ela; (app.js)

// deixando a variavel Global para ser acessado na controller -> chat

app.set('io', io);

- fazer o teste se o alert apareceu em todos que estão conectados no websocket do chat

- Tirando alert de teste, informando quem de fato entrou no chat.

Na função emit() feita anteriormente do lado do servidor, vamos passar um obj json no 2º parâmetro

/\* emit msgParaCliente (websocket)

utilizando obj do socket.io dentro do controller \*/

application.get('io').emit(

'msgParaCliente',

{ apelido: dadosForm.apelido, mensagem: ' acabou de entrar no chat' }

);

Agora na view do cliente vamos recuperar escutando e formatando em um dialogo já padronizado no css (lembrar de colocar um id onde queremos colocar o elemento novo criado)

                <div class="col-md-10" id="dialogos">

                </div>

        <script>

            // instancia do obj que se conecta com websocket

            var socket = io('http://localhost:3000');

            // escutar msgParaCliente

            socket.on('msgParaCliente', function(data){

                var html = '';

                html += '<div class="dialogo">';

                    html += '<h4>' + data.apelido + '</h4>';

                    html += '<p>' + data.mensagem + '</p>';

                html += '</div>';

                $('#dialogos').append(html);

            });

        </script>

Fazer o teste e verificar o resultado abaixo...

