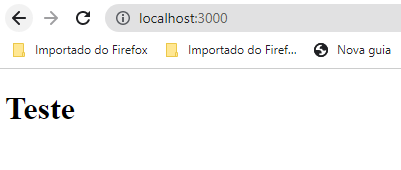
**UTILIZANDO O SOCKET.IO**

**CONFIGURAÇÃO DO PROJETO PARA O SOCKET IO**

1. Inicie o projeto npm na pasta do projeto
2. Digite npm init.
3. Instalando as bibliotecas.
4. Express: npm install express –save
5. Nodemon: npm install nodemon --save
6. Para as views vamos instalar a ejs: npm install ejs --save
7. Socket-IO: npm install socket.io –save
8. Crie o arquivo index.js
9. importe o express e crie a instancia da aplicação.
10. *var* express = require('express');
11. *var* app = express();
12. vamor montar um servidor http nativo do node, isso exige que agente mude a configuração original do express.
13. //colocando a aplicação express para rodar dentro do servidor http nativo do node
14. *var* http = require('http').createServer(app);
15. isso foi feito poqure que a aplicação express e a aplicação socket-io precisam estar rodando no mesmo servidor http. Para isso primeiro colocamos a aplicação express dentro do servidor http nativo do node.
16. Depois importamos o socket.io e passamos como parâmetro o servidor http que foi criado anteriormente.
17. *var* io = require('socket.io')(http);

**CONFIGURANDO A VIEW NO PROJETO**

1. iremos utilizar as views do ejs.
2. app.set('view engine','ejs');
3. vamos criar uma rota principal essa rota irá renderizar um arquivo html chamado index.ejs
4. esse arquivo será criado dentro da pasta views.
5. //criando rota principal que irá renderizar o arquvio index.ejs
6. app.get('/', (*req*, *res*) *=>* {
7. *res*.render('index')
8. })
9. No arquivo index.ejs vamos criar uma estrutura html e um h1 com uma mensagem
10. <!DOCTYPE html>
11. <html lang="en">
12. <head>
13. <meta charset="UTF-8">
14. <meta http-equiv="X-UA-Compatible" content="IE=edge">
15. <meta name="viewport" content="width=device-width, initial-scale=1.0">
16. <title>Document</title>
17. </head>
18. <body>
19. <h1>Teste</h1>
20. </body>
21. </html>
22. Agora vamos configurar a aplicação para rodar utilizando o servidor nativo que criamos
23. http.listen(3000, () *=>* {
24. console.log("Servidor rodando");
25. })
26. Colocando a aplicação para rodar, no terminal na pasta do projeto digite: nodemon index.js
27. Agora digite no navegador localhost:3000 e aparecerá a mensagem que criamos na view index.ejs
28. 

**Como Iniciar com o socket.io**

1. A primeira coisa que temos que fazer é abrir um evento de conexão, pois os web sockets são baseados em eventos.
2. Esse evento é um padrão do socket.io e é chamado sermpre que um cliente se conecta a sua aplicação web socket.
3. //evento de conexão do socket.io
4. io.on("connection",(*clienteBackEnd*) *=>* {
5. console.log(*clienteBackEnd*);
6. console.log(*clienteBackEnd*.id)
7. })
8. Para testar esse cliente temos que ir no fron end no arquivo index.ejs e importar o script front end do socket.io
9. <script src="">
10. </script>
11. O socket.io possui tanto a biblioteca para back end como para front end. E são utilizadas para comunicar o backend com o frontend em tempo real.
12. Para isso podemos ir no goolgle e pesquisar socket.io cdn.
13. 
14. Vamos copiar o link e colar no src do script criado no arquivo index.ejs.
15. <script src="https://cdn.socket.io/4.4.1/socket.io.min.js" integrity="sha384-fKnu0iswBIqkjxrhQCTZ7qlLHOFEgNkRmK2vaO/LbTZSXdJfAu6ewRBdwHPhBo/H" crossorigin="anonymous"></script>"></script>
16. Vamos criar uma variável em um script abaixo do link que importamos.
17. Essa variável irá receber o io() com o endereço do servidor que criamos e está rodando o socket.io
18. <script>
19. *var* clienteFrontEnd = io("http://localhost:3000");
20. </script>
21. Fazendo isso estaremos conectando o socket do frontend com o backend.
22. Vamos rodar novamente a aplicação e irá aparecer a mensagem do backend quando recarregarmos a página no front end.

**ENVIANDO ENVENTO DO FRONT PARA O BACK END E VICE VERSA**

1. Vamos enviar um evento para o back end informando o nome do cliente.
2. No arquivo index.ejs no script que contem a variável de cliente criada, vamos utilizar o evento emit.
3. <script>
4. *var* clienteFrontEnd = io("http://localhost:3000");
5. clienteFrontEnd.emit("mensagem",{nome: 'Claudisnei Bello'});
6. </script>
7. C
8. No arquivo index.js do backend vamos receber os dados enviados pelo frontEnd.
9. Utilizamos o método on que irá ficar escutando e trará qualquer dado enviado pelo evento chamado “mensagem”.
10. //evento de conexão do socket.io
11. io.on("connection",(*clienteBackEnd*) *=>* {
12. *clienteBackEnd*.on("mensagem", (*data*) *=>* {
13. console.log(*data*);
14. })
15. })
16. Para melhorar o teste na view index.ejs, vamos inserir um input e um botão.
17. <h1> Teste de Eventos com socket.io </h1>
18. <br>
19. <input type="text" name="" id="palavra">
20. <button>Enviar</button>
21. Vamos ainda criar uma função na view index.ejs, chamada enviar.
22. Nesta função vamos criar uma variável chamada input que irá receber o input de id ‘palavra’, através da método document.getElementById do Javascript puro.
23. Chamaremos ainda nesta função o método emit do clienteFrontEnd e criaremos o evento chamado ‘palavra’, passando o valor do input toda vez que o evento for chamado.
24. *function* enviar(){
25. *var* input = document.getElementById('palavra');
26. console.log(input.value);
27. clienteFrontEnd.emit("palavra", input.value)
28. }
29. Agora temos que ir no arquivo do backEnd o index.js e chamar o método ‘on’ do clienteBackEnd para escutar toda comunicação que ocorrer pelo evento ‘palavra’.
30. *clienteBackEnd*.on('palavra',(*data*) *=>* {
31. console.log(*data*)
32. })
33. Para testar a resposta em tempo real vamos melhorar o html da view index.ejs
34. Vamos inserir um parágrafo com id chamado resultado.
35. <hr>
36. <p id="resultado"></p>
37. Vamos voltar agora no index.js, no listen do evento ‘palavra’ e vamos chamar o método emit de clienteBackEnd para emitir um evento chamado ‘resultado’ do servidor para o cliente, passando os dados que foram passados pelo cliente e a mensagem que quiser.
38. *clienteBackEnd*.on('palavra',(*data*) *=>* {
39. console.log(*data*);
40. *clienteBackEnd*.emit("resultado", *data* + "Guia do programador" );
41. })
42. Na view index.ejs vamos chamar o método ‘on’ do clienteFrontEnd para receber o evento ‘resultado’ emitido no clienteBackEnd e vamos passa-lo para dentro de um parágrafo chamado ‘resultado’
43. clienteFrontEnd.on('resultado', (*data*) *=>* {
44. console.log(*data*);
45. *var* p = document.getElementById('resultado');
46. p.innerHTML = *data*;
47. })

**Realizando a desconexão**

1. É possível verificar se o cliente realizou a desconexão com o servidor, porém deve ser uma desconexão limpa, ou seja, clincando para fechar o navedor, desligar o computador normalmente. Agora em casos de perda de conexão ou queda de energia, o método não funciona bem.
2. Para detectar quando um cliente se desconecta da aplicação temos que criar um método de desconexão no front end.
3. clienteFrontEnd.on("disconnect", ()*=>*{
4. console.log("Desconectado")
5. })
6. Temos também que ir no backend e criar outro método dentro do evento de conexão que irá escutar a comunicação quando o evento ‘disconnect’ for chamado no fronted.
7. //evento de conexão do socket.io
8. io.on("connection",(*clienteBackEnd*) *=>* {
9. *clienteBackEnd*.on("disconnect",() *=>* {
10. console.log("Cliente X desconectou "+ *clienteBackEnd*.id);
11. })
13. *clienteBackEnd*.on("mensagem", (*data*) *=>* {
14. console.log(*data*);
15. })
16. *clienteBackEnd*.on('palavra',(*data*) *=>* {
17. console.log(*data*);
18. *clienteBackEnd*.emit("resultado", *data* + " Guia do programador" );
19. })
21. })
22. Quando o cliente se desconectar será enviado a mensagem com o id do cliente.