

Redes de Computadores



Luiz Fernando Vieira de Castro Ferreira
13/0013757

Introdução

O trabalho consiste na implementação de um chat para troca de mensagens em tempo real. Esse chat desenvolvido é baseado em um dos primeiros sistemas de troca de mensagem o Internet Relay Chat (IRC). A implementação utilizou o protocolo da camada de transporte TCP (Transmission Control Protocol), protocolo que fornece transferência confiável de dados, detecção de erro, retransmissões, reconhecimentos cumulativos, temporizadores e cabeçalhos com número de sequência e de reconhecimento. Foi escolhido fazer um sistema web utilizando a linguagem Javascript junto a diversos frameworks e bibliotecas. A mais importante dentre essas bibliotecas é Socket.IO. Ela auxilia no desenvolvimento da aplicação proporcionando uma comunicação baseada em evento bidirecional e real-time.

Arquitetura

O sistema utiliza uma arquitetura Cliente Servidor para troca de mensagens entre diferentes usuários. Nessa arquitetura, a aplicação Cliente pede por serviços fornecidos pela aplicação servidora.

Para implementação deste chat foi escolhido fazer um sistema web com uma única página, ou seja, não ocorrem redirecionamentos em nenhum momento durante a execução do programa. Visando alcançar o objetivo do trabalho foram utilizadas uma série de linguagens de programação e marcação assim como uma série de frameworks e bibliotecas.

Por parte do cliente, a linguagem responsável pela estruturação da página foi o HTML e a linguagem escolhida para estilização foi a linguagem CSS juntamente ao framework Bootstrap. Além delas, foi utilizada a linguagem Javascript juntamente ao framework jQuery para dar dinamicidade aos dados e à interface.

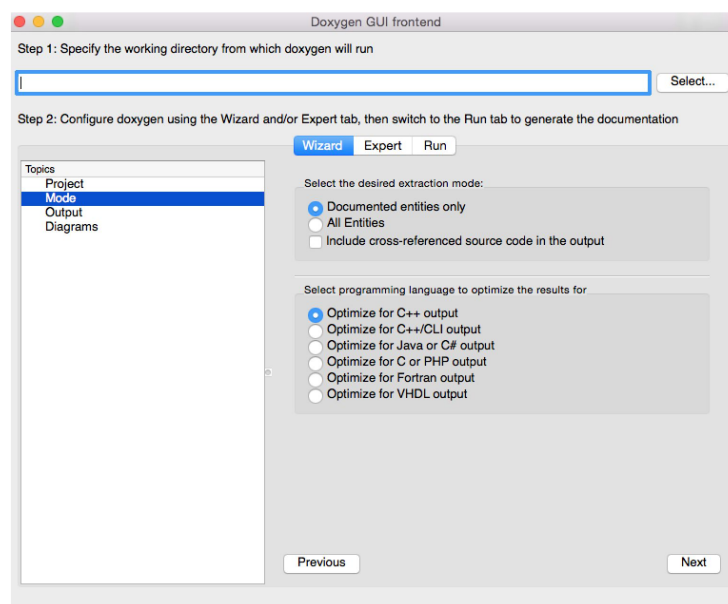
No lado do Servidor foi utilizada a linguagem Javascript juntamente à plataforma Node.js e ao framework express. Além do framework foram utilizadas algumas bibliotecas e módulos como por exemplo http, socket.io, socket.io-file, fs, e functional.js.

A maior parte da aplicação tem como base a orientação a eventos disponibilizada tanto pela linguagem Javascript quanto pela biblioteca Socket.IO. Outra parte considerável do código utiliza fundamentos da programação funcional para manipulação dos grupos e dos usuários da aplicação servidora. Justamente pela facilidade de manipulação de eventos e pela característica multi-paradigma da

linguagem Javascript acredita-se que ela foi uma ótima escolha de linguagem para realização deste trabalho.

Documentação

Como ferramenta de documentação foi sugerido o uso do Doxygen. Infelizmente não é possível gerar documentação Doxygen a partir de código Javascript. A ferramenta possui uma quantidade bastante limitada de linguagens suportadas, e por esse motivo, foi adicionada documentação no estilo JavaDoc em todo código, mas não foi possível gerar o HTML solicitado na especificação do trabalho.



```
1  /**
2   * @Author Luiz Fernando Vieira de Castro Ferreira
3   * @date Novembro, 2016
4   * @brief Arquivo do servidor
5   *
6   * Arquivo responsável pelo armazenamento em memória e
7   * pela lógica da manipulação dos dados presentes
8   * no lado servidor da aplicação. Aqui estão presentes
9   * as diferentes funções responsável pelo tratamento
10  * dos eventos enviados por diferentes clientes
11  */
12
13  var express = require('express');
14  var app = express();
15  var server = require('http').createServer(app);
16  var io = require('socket.io').listen(server);
17  var fjs = require('functional.js');
18  var SocketIOFile = require('socket.io-file');
19  var fs = require('fs');
20
21  /**
22   * Variáveis responsáveis por guardar o estado
23   * do sistema. Com apenas as variáveis que
24   * guardam os usuários, grupos e suas respectivas
25   * sockets é possível simular a existência de
26   * um banco de dados em memória que guarda
27   * os estados dos diferentes grupos
28  */
```

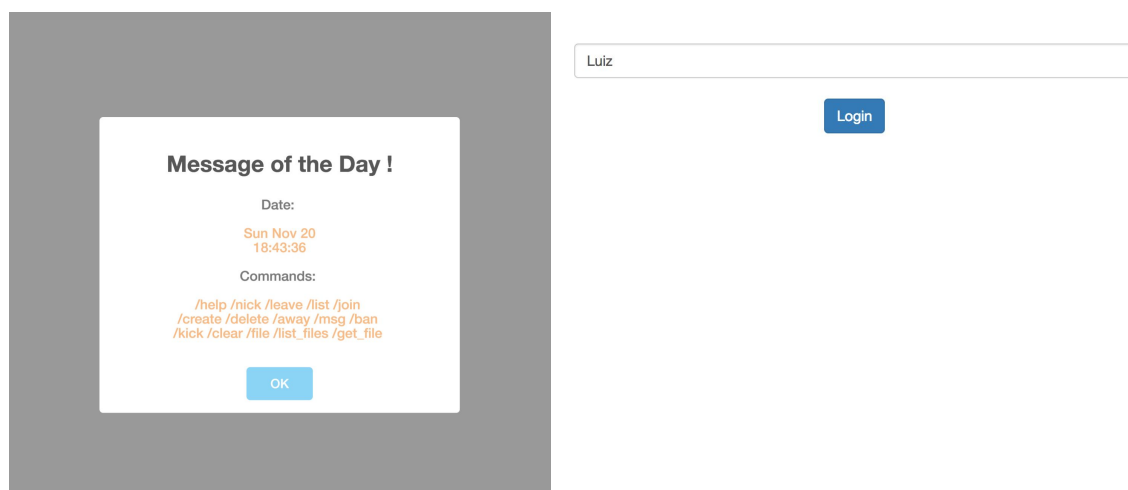
```
1  /**
2   * @Author Luiz Fernando Vieira de Castro Ferreira
3   * @date Novembro, 2016
4   * @brief Arquivo do cliente
5   *
6   * Arquivo responsável pela manipulação dos elementos
7   * da interface e pelo envio de mensagens ao servidor.
8   * A aplicação cliente guarda apenas dados relativos ao
9   * nome do usuário que está utilizando a aplicação
10  * e o nome do grupo que aquele usuário faz parte
11  */
12
13  /**
14   * Variáveis responsáveis por guardar informações
15   * relativas aos clientes da aplicação
16   */
17  var socket = io.connect();
18  var userSessionName = '';
19  var currentGroup = '';
20
21  /**
22   * Todo conteúdo dentro desse procedimento será
23   * inicializado assim que o evento document.ready
24   * for disparado, ou seja, só quando o documento
25   * HTML tiver sido totalmente carregado
26   */
27  $(function() {
```

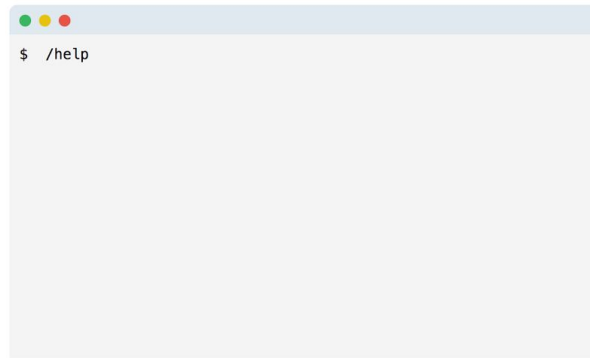
Screenshots e Funcionalidades

Nessa seção serão apresentadas as telas com todas as funcionalidades e comandos pedidos no trabalho. Note que a maioria dos comandos podem ser executados por meio da interface e por meio do terminal desenhado na tela do browser. **Conforme acordado com o professor**, alguns dos comandos não faziam muito sentido juntamente com a interface gráfica, logo, suas lógicas estão implementadas mas não são chamadas por meio de comandos no terminal, e sim pela interação do usuário com a interface existente.

Nas imagens a seguir foram cadastradas três pessoas: Luiz, Lucas e Diogo. O usuário Luiz é o administrador de um grupo e os demais usuários pedem para entrar no grupo de Luiz. O usuário Diogo é administrador de outro grupo, assim como o usuário Lucas.

A primeira coisa que aparece quando o usuário entra no sistema é a Message of the Day indicando a data, hora, e comandos que o usuário pode executar no terminal. Depois de clicar no botão OK lhe é mostrada uma tela com um campo de input onde ele digita seu nome. Posterior a essa tela onde o usuário digita o nome é mostrada uma tela com um terminal onde é possível digitar diversos comandos e é apresentada uma lista com todos os grupos do sistema.





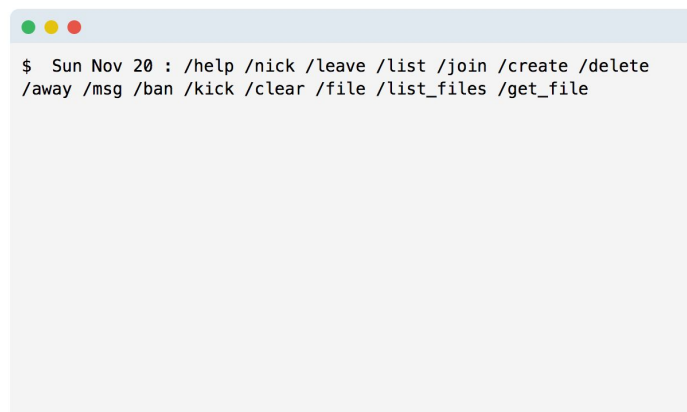
```
$ /help
```

Nome do Grupo

Criar Grupo

A partir desse momento o usuário pode digitar os comandos descritos na especificação do trabalho.

/help

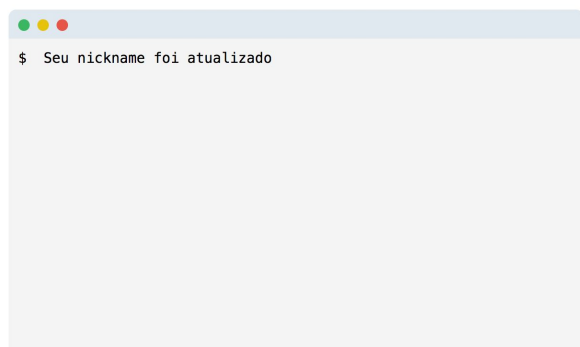


```
$ Sun Nov 20 : /help /nick /leave /list /join /create /delete  
/away /msg /ban /kick /clear /file /list_files /get_file
```

/nick



```
$ /nick LuFernando
```

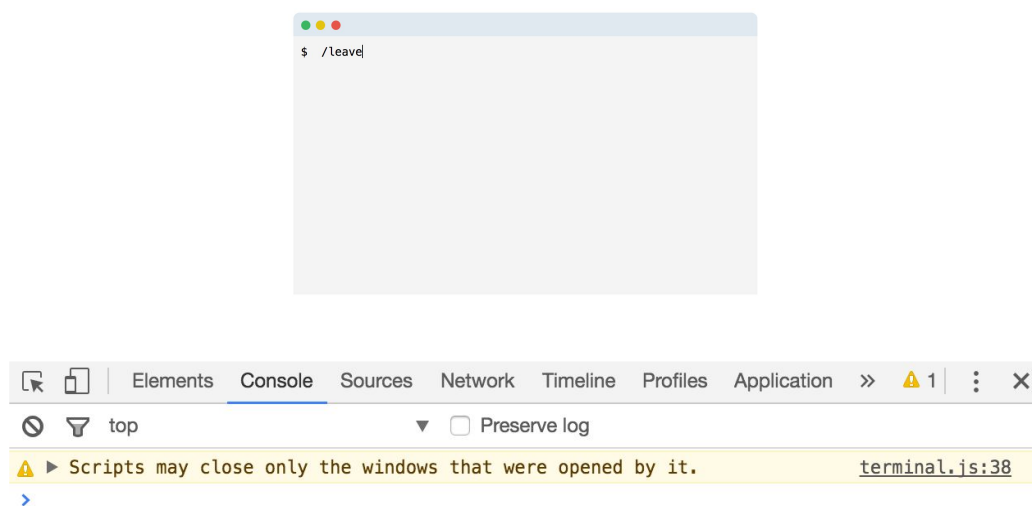


```
$ Seu nickname foi atualizado
```

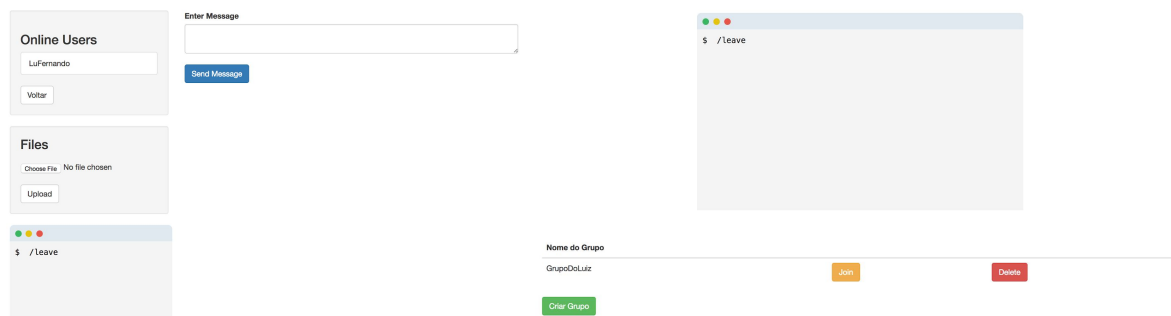
/leave

Primeiramente é mostrado um exemplo onde o comando /leave foi utilizado com o usuário fora de qualquer grupo. Aqui nota-se que devido a limitação dos browsers não é possível fechar a janela por meio de código Javascript a menos que a página tenha sido criada usando Javascript (não é o caso deste sistema). Essa limitação é discutida no seguinte tópico do famoso site StackOverflow.

<http://stackoverflow.com/questions/19761241/window-close-and-self-close-do-not-close-the-window-in-chrome>



A próxima imagem é referente ao comando /leave quando o usuário está dentro de um grupo. Nela é possível ver que o comando funcionou corretamente.



/list

\$ /list

Nome do Grupo		
GrupoDoLuiz	Join	Delete
GrupoDoLucas	Join	Delete
GrupoDoDiego	Join	Delete

Create Grupo

\$ GrupoDoLuiz GrupoDoLucas GrupoDoDiego

Nome do Grupo		
GrupoDoLuiz	Join	Delete
GrupoDoLucas	Join	Delete
GrupoDoDiego	Join	Delete

Create Grupo

/join

\$ /join GrupoDoLuiz

Online Users

Luiz

Voltar

Files

Choose File No file chosen

Upload

\$ /help

Enter Message

Send Message

/create

\$ /create OutroGrupo

Nome do Grupo		
GrupoDoLuiz	Join	Delete
OutroGrupo	Join	Delete

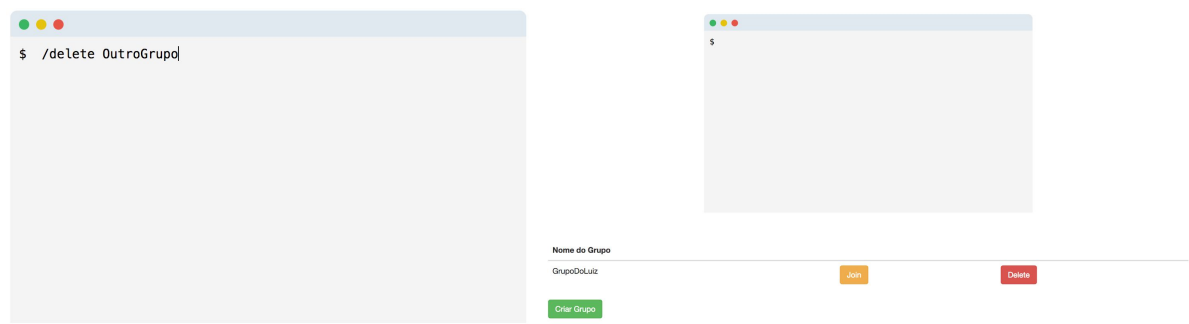
Create Grupo

\$

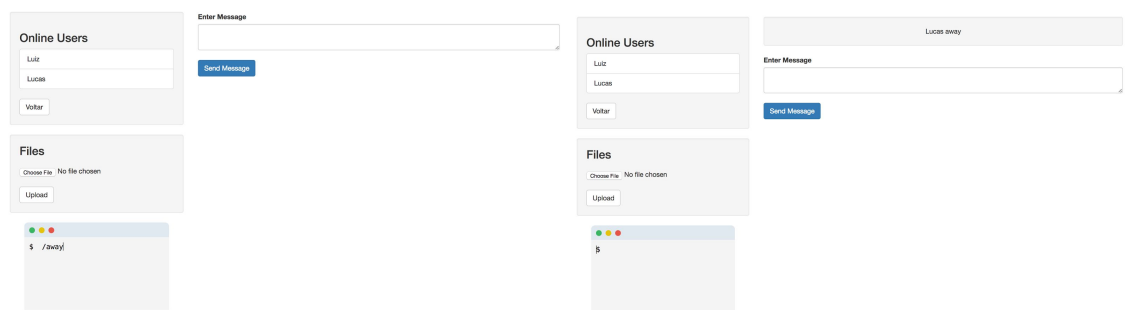
Nome do Grupo		
GrupoDoLuiz	Join	Delete
OutroGrupo	Join	Delete

Create Grupo

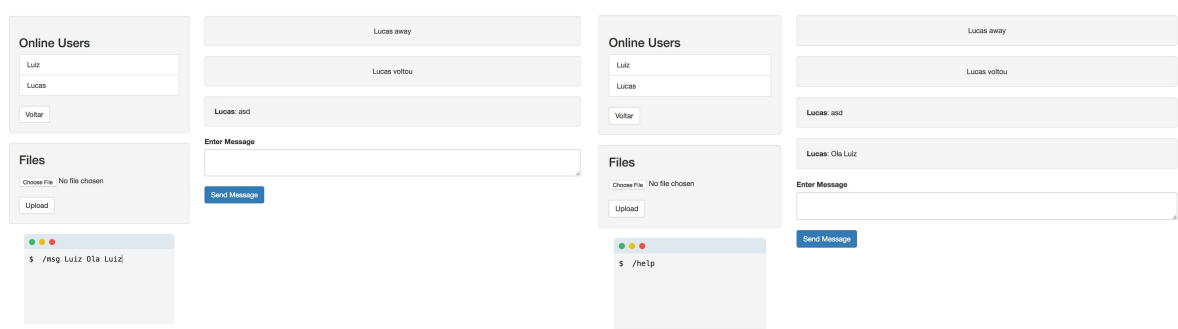
/delete



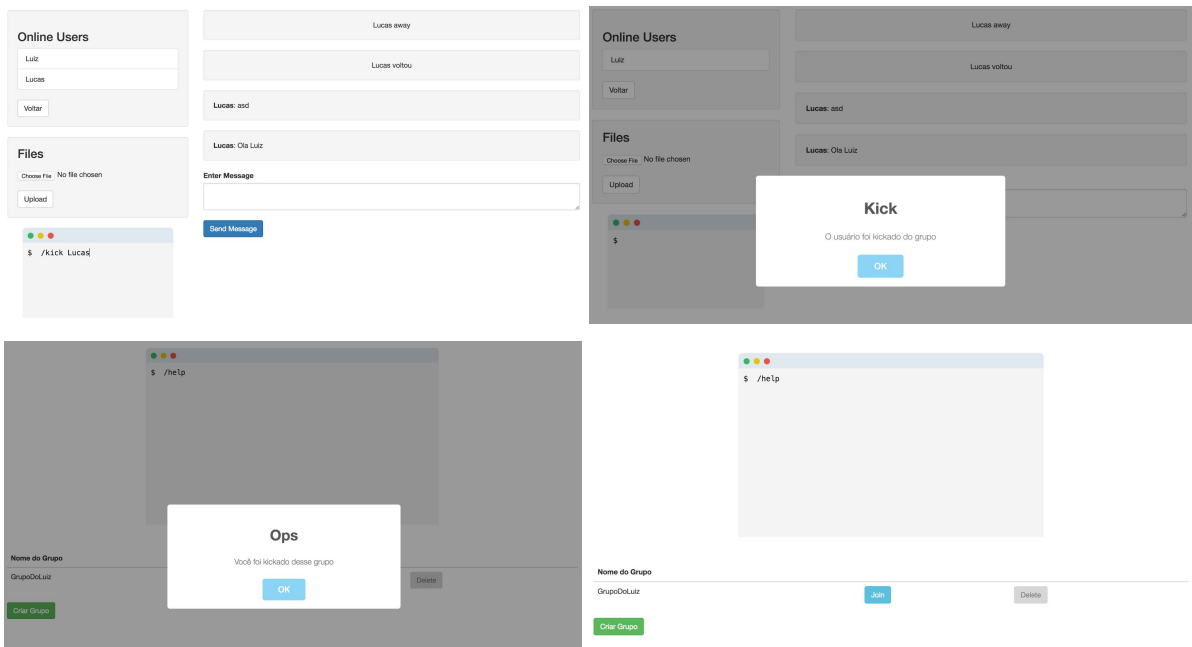
/away



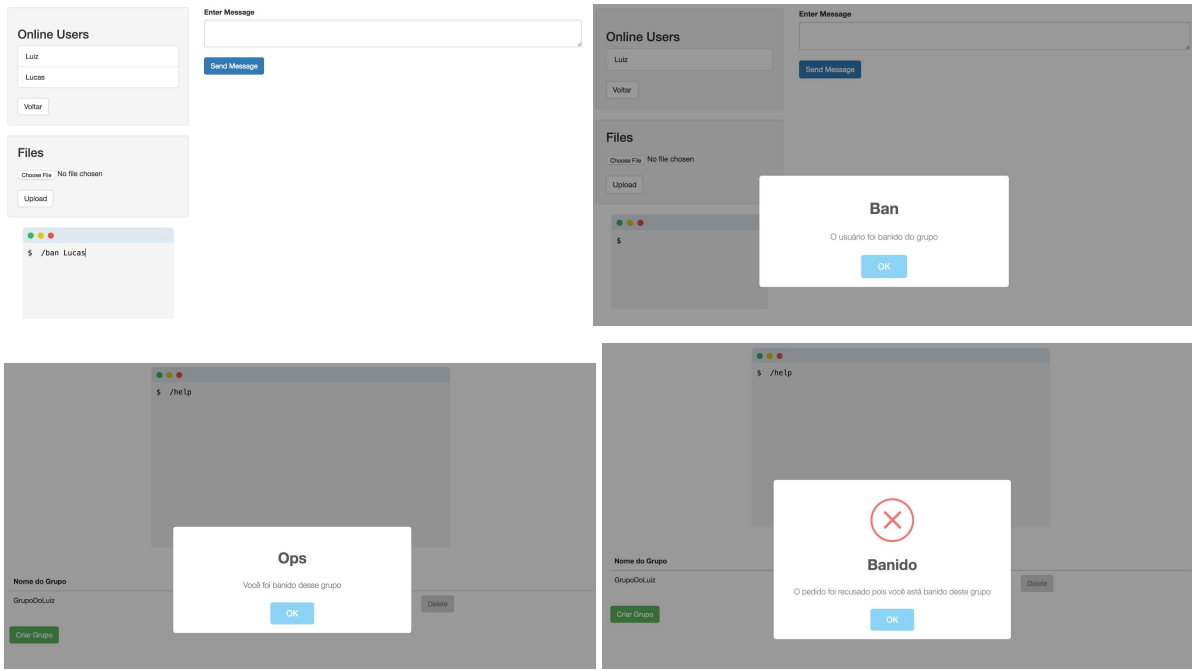
/msg



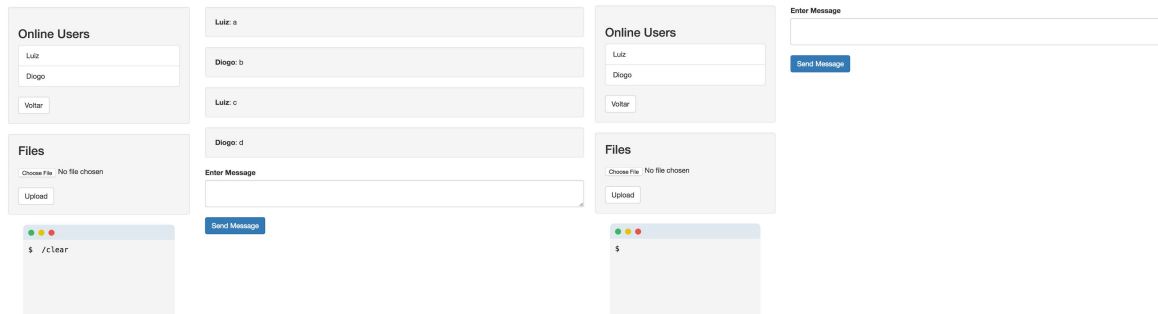
/kick



/ban



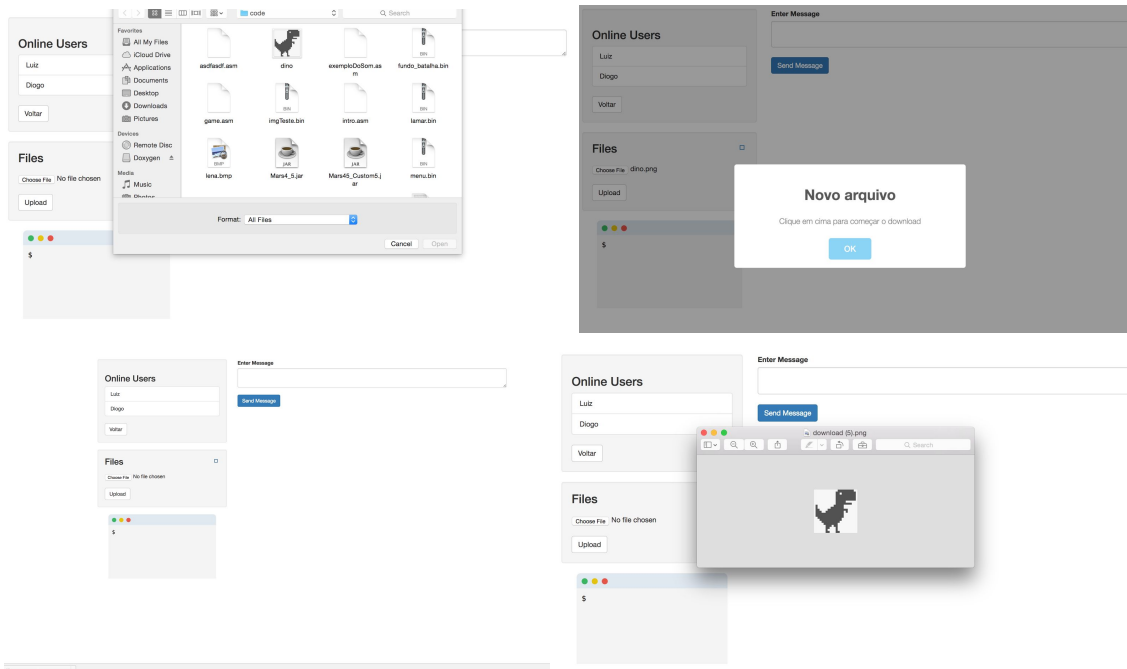
/clear



/file

/list_files

/get_file



Informações para Execução

Como dito anteriormente, durante a realização do trabalho foram utilizadas algumas bibliotecas com o intuito de agilizar o processo de desenvolvimento do software. Abaixo é apresentada uma lista com tudo que deve ser configurado na sua máquina para que possa rodar o programa e uma breve explicação de como configurar cada biblioteca.

Para executar o projeto é necessário que o Node.js esteja configurado. Para isso, vá até o site <https://nodejs.org/en/> e baixe o software. Para garantir que o Node.js foi

corretamente configurado vá até o terminal e digite “node -v”. Depois de configurado, navegue até a pasta do projeto e digite o comando “node server”, ele é o responsável por subir a aplicação servidora. Agora abra o navegador Google Chrome e acesse o endereço “<http://localhost:3000/>”. O sistema está pronto para uso. Para abrir vários clientes basta abrir novas abas no navegador e colocar o mesmo endereço mostrado acima.

Além de executar o sistema localmente foi necessário colocar o sistema na cloud para garantir que toda sua parte dependente da rede funcionasse perfeitamente. Para essa tarefa foi utilizado o heroku. Heroku é uma plataforma como serviço usada para computação na nuvem. Segue o link do site do heroku <https://www.heroku.com/> . O sistema ficou hospedado na seguinte URL <https://limitless-thicket-38257.herokuapp.com/> . Para abrir vários clientes o procedimento é semelhante ao descrito previamente, a única diferença é a URL que será acessada.

Para facilitar a execução foram colocadas todas as outras bibliotecas e frameworks dentro da pasta do projeto, dessa forma não é necessário configurá-las. Segue uma lista de todas as outras tecnologias utilizadas e onde encontrá-las.

Express: <http://expressjs.com/pt-br/>

npm install express --save-dev

SocketIO: <https://www.npmjs.com/package/socket.io>

npm install socket.io --save-dev

SocketIOFile: <https://www.npmjs.com/package/socket-file>

npm install socket-file --save

FunctionalJs: <http://functionaljs.com/installation/>

npm install functional.js --save

jQuery: <https://jquery.com/>

Bootstrap: <http://getbootstrap.com/>

SweetAlert: <http://t4t5.github.io/sweetalert/>

Conclusão

O trabalho foi bem interessante e ajudou muito na fixação do conteúdo visto em sala a respeito do TCP e principalmente de Sockets. Praticamente todos os requisitos do projeto foram cumpridos com exceção de alguns que foram limitados pela linguagem ou pelo ambiente de desenvolvimento, como por exemplo, o fechamento da janela do browser pelo Javascript que não é possível por questões de segurança das aplicações web.