**CadeVice doc**

Ferramentas usadas:

* NodeJs
* MongoDb
* angularJs

Instalação:

Para instalar o Sistema, instale o MongoDb e o NodeJs (pode ser a versão mais atual de ambos).

Após instalar o MongoDb, verifique no site, como fazer para rodá-lo como serviço no sistema operacional que você escolheu.

Feito isso, baixe o projeto usando o git (link do projeto: <https://github.com/omiguel/DMNute>) e coloque-o na pasta de sua preferência no servidor.

Em seguida instale o bower usando o comando “npm install –g bower”.

Dentro da pasta suaPasta\DMNute\public use o comando “bower install”, com esse comando todas as dependências que estão sendo usadas no angularjs serão instaladas.

Dentro da pasta suaPasta\DMNute use o camando “npm install”, com esse comando, todas as dependências que estão sendo usadas para o nodeJs serão instaladas.

Feito isso, o sistema estará pronto para rodar. Para coloca-lo em produção use o comando “node index.js”. Assim que estiver pronto para uso, aparecerá na interface do cmd em qual porta o sistema esta rodando, para alterá-la basta modificar o numero da porta no arquivo index.js, que se encontra na raiz do projeto.

Existe uma outra opção para rodar o sistema, segue abaixo instruções:

Baixe o supervisor usando o comando “npm install –g supervisor”, feito isso execute o mesmo passo citado anteriormente para rodar o projeto, porém ao invés do comando “node index.js” use o comando “supervisor index.js”.

Se optar por rodar o sistema com o supervisor, em casa de algum bug em que o serviço fosse cair, o supervisor se responsabiliza por subir novamente o serviço, ele também tem a opção de ser iniciado como serviço, mas para isso pesquise como fazer isso no sistema operacional que você está usando.

Descrição do sistema:

O CadeVice, foi todo desenvolvido usando a linguagem javascript, tanto no frontend quanto no backend.

O sistema é todo baseando em eventos, segue uma breve descrição de como funciona essa estrutura de eventos:

Quando um cliente entra no sistema, ele recebe um socket, esse socket possui uma chave única, ou seja, cada cliente tem o seu socket para comunicação com o servidor, a partir daí, ele recebe no servidor o rtcLogim, esse documento é responsável por restringir todas as permissões do cliente dependendo do tipo de usuário, todas as solicitações do cliente, passam pelo rtc que ele recebeu. Nesse primeiro momento, antes do login, ele não tem nenhuma permissão a não ser a de solicitar entrada no sistema (explicarei mais sobre o rtc posteriormente), assim que o sistema verificar que o usuário tem acesso, ele é responsável por verificar que tipo é esse usuário e definir as suas restrições, feito isso, o usuário terá acesso ao sistema e de acordo com o tipo de usuário, ele pode ou não executar determinadas funções.

RTC:

Real Time Connection, é o documento responsável por restringir os acessos do cliente, ele funciona com evento, ou seja, ele tem eventos que ele escuta do lado do cliente, que vem através do canal do socket e também escuta eventos do lado do servidor, que ele escuta através do hub, a função principal dele é repassar os eventos, por exemplo: o rtcLogin é responsável por tratar de login apenas, se o usuário conseguir burlar as regras da interface e acessar uma tela de mapas, nada acontecerá, pois para pegar os mapas, o cliente precisa mandar um evento fazendo essa solicitação, porem o rtcLogin intende apenas o evento de logar e ignorará qualquer outro evento que passar por ele.

Quando o usuário informa suas credenciais no e solicita o login ao rtcLogin, ele pede para o banco de dados verificar se aquele usuário esta cadastrado, caso sim, o rtcLogin verifica qual é o tipo de usuário, avisa a interface do cliente que ele está cadastrado e passa as informações dele, e ao mesmo tempo, de acordo com o tipo de usuário, ele determina qual é o rtc apropriado a esse usuário, cria, passa o socket e se destrói. Enquanto isso a interface pega os dados do cliente cadastrado e verifica quais funcionalidades ele tem acesso e quais as views ele também pode acessar.

SIOM:

No cliente temos o Socket IO Manager, todos os eventos do cliente passam por esse documento, se o documento é uma solicitação ao servidor, usa-se a função emitirServer() dentro dele, ele é um documento sigleton, ou seja, ele pode ser acessado de qualquer lugar da interface, caso queira usar um evento de comunicação apenas do lado do cliente, usa-se a função emit().

Hub:

Nome derivado do hardware hub, assim como no fronte, esse documento é por esse documento que passam todos os eventos do server, porem a grande diferença é que todos os eventos são enviados através da função emit(), quem se responsabiliza em verificar se o evento tem que passar para a interface ou não é o próprio rtc do cliente.

Os três citados acima são os principais meios de comunicação nesse projeto, pode-se interpretar como tendo 2 mundos, o lado do cliente, onde temos o SIOM e o lado do server onde temos o Hub, esses dois documentos nem se conhecem, sendo assim o RTC é o documento responsável pela comunicação entre os dois mundos.

Estrutura:

A partir daqui, farei uma breve descrição do que contem em cada pasta e a responsabilidade de cada documento.

db:

Dentro desta pasta, temos todos os documentos responsáveis por criar e gerenciar o banco de dados, nela tempos documentos para inicialização dos bancos e para fazer pesquisas nele, abaixo descrição de subpastas e documentos encontrados dentro dela.

managers: