FACULDADE DE TECNOLOGIA SENAI JARAGUÁ DO SUL
GRADUAÇÃO TECNOLÓGICA EM SISTEMAS PARA INTERNET

MANUAL TÉCNICO SISTEMA REAL MANAGER

JARAGUÁ DO SUL, SC
2020

Sumário

1 Apresentação / Descrição

O sistema Real Manager foi desenvolvido com o intuito de automatizar e agilizar os processos de abertura, acompanhamento e finalização das informações referentes às ordens de manutenção.
Presentes dentro da empresa Duas Rodas, às ordens de manutenção servem para garantir que os procedimentos de manutenção sejam executados de forma com que cada operação, componente, apontamento e outras diversas informações importantes deste processo sejam armazenadas e após sua finalização, inseridas no sistema SAP da empresa.
Sendo assim foram feitos estudos e desenvolvimentos para que este sistema atenda às necessidades dos nossos clientes a fim que o processo elimine o uso de papel e aumente a produtividade da equipe Duas Rodas.

2 Tecnologias

Às tecnologias utilizadas durante o desenvolvimento do Real Manager estarão descritas nesta seção, seus sites e documentações respectivas também serão disponibilizadas junto aos conteúdos referentes.

2.1 AngularJs

O AngularJs é um framework JavaScript de código aberto que tem como principal objetivo aumentar aplicativos para serem acessados por um navegador web, a biblioteca lê o HTML que contém atributos especiais e então os compila, executando a diretiva na qual a tag em específico pertence.
O AngularJs é mantido pela equipe do Google e possibilita que o valor das variáveis JavaScript podem ser setadas manualmente ou via recursos JSON estáticos ou dinâmicos.
Às diretivas do AngularJs permitem que os desenvolvedores especifiquem as tags HTML personalizadas e reusáveis, personalizando assim o comportamento dos elementos.
O AngularJs também abstrai o acoplamento entre o lado cliente e o lado servidor da aplicação, permitindo que o desenvolvimento do aplicativo evolua em ambos os lados e que também haja um reaproveitamento de código.

Mais informações de instalação, preparação de ambiente de utilização, documentação e criação de projetos estão disponíveis no site oficial do AngularJs: angularjs.org .

2.2 Node.Js

O Node.Js é um ambiente de execução JavaScript a nível backend e frontend, trabalhando em conjunto dos frameworks e bibliotecas disponíveis utilizados pelos desenvolvedores, o Node.Js foi implementado é baseado no motor de interpretação JavaScript V8 do Google, com desenvolvimento mantido pela Fundação Node.Js em parceria com a Fundação Linux.
As vantagens de utilizar o Node.Js são a flexibilidade, pois possui um gerenciador de pacotes reusáveis NPM, como o Express.Js e o maior repositório de softwares, possibilitando ao interpretador um potencial de ser utilizado em qualquer situação. A leveza, pois o ambiente Node.Js não exige muitos recursos computacionais para ser executado.
O suporte, que conta com empresas de serviços de armazenamento em nuvem, como Microsoft Azure, AWS e Google Cloud.

Mais informações sobre utilização do Node.Js e sua documentação podem ser encontrados no site oficial: nodejs.org .

2.3 MySQL

O MySQL é um sistema de gerenciamento de banco de dados, que utiliza a linguagem SQL como interface e atualmente é um dos sistemas de gerenciamento mais populares da Oracle. Empresas como a NASA, Bradesco, HP, Nokia e Sony utilizam o MySQL para gerenciar seus bancos de dados.
Dentre as principais características do MySQL podemos citar, a portabilidade, a compatibilidade, o excelente desempenho e a facilidade de uso, ele também contempla a utilização de vários Storage Engines, como o InnoDB, Falcon, CSV e MyISAM.

Mais informações sobre instalação, utilização e aplicação podem ser encontradas no site oficial do MySQL: mysql.com .

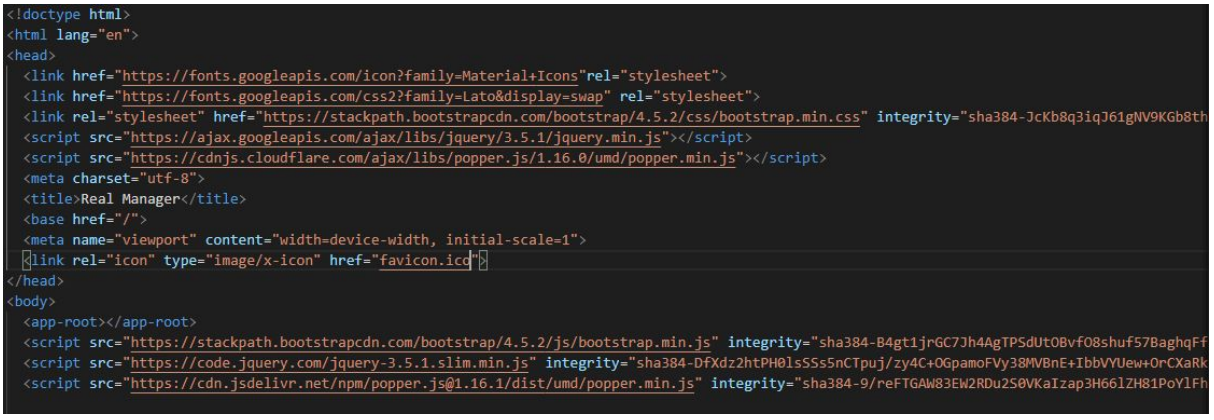
3 Estruturas

É de suma importância apresentar às estruturas presentes no sistema Real Manager, pois são nelas que às informações contidas são tratadas e direcionadas da forma adequada, assim cumprindo às funcionalidades do sistema.

3.1 Front-End

O desenvolvimento Front-End de forma resumida, é a prática de converter os elementos de dados em interface gráfica, utilizando HTML, CSS, JavaScript e frameworks baseados nestes. Ou seja, utilizamos do Front-End para criarmos interfaces gráficas para sites e sistemas web, no caso do Real Manager foram utilizadas para às interfaces gráficas, HTML 5, CSS 3 e o framework AngularJs. Foram desenvolvidas todas as telas presentes no sistema web, e também os elementos presentes na aplicação móvel.

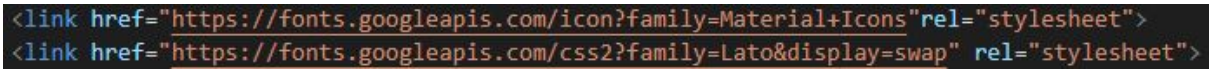
Figura 1 - Código Principal HTML.



Na figura 1 encontra-se o arquivo HTML principal dentro do framework AngularJs, que serve para referenciar os outros arquivos HTML presentes, definindo assim os recursos, elementos, bibliotecas e implementações Front-End que serão utilizadas no desenvolvimento das interfaces gráficas.

As tags “link” são responsáveis por trazer os recursos desejados para utilização até o código HTML para depois serem interpretados e convertidos na interface gráfica desejada pelos desenvolvedores.

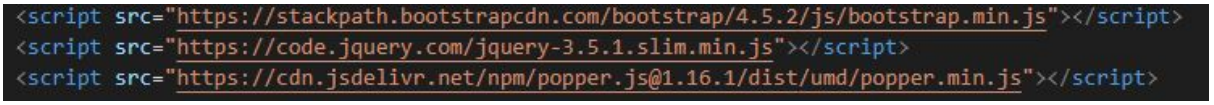
Figura 2 - Tag “Link”.



Na figura 2, estão representadas duas das tags ”link” presentes no código HTML, estas duas em específico estão possibilitando a utilização de ícones e fontes disponíveis no Material Icons e no Google Fonts.

Temos também neste código, a parte das importações lógicas, passando os recursos que desejamos utilizar junto ao HTML, como o framework Bootstrap, o Jquery e o Popper.Js, que serão utilizados durante a execução das interfaces gráficas.

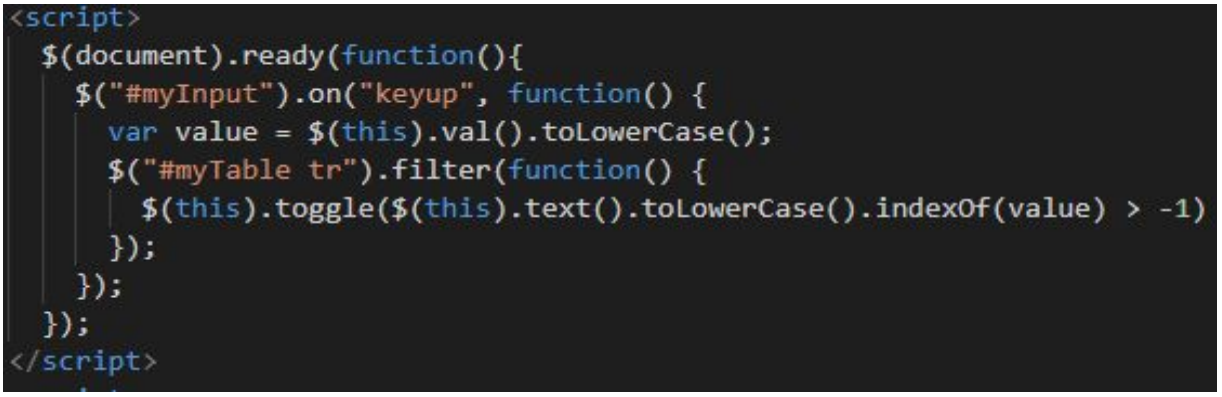
Figura 3 - Importações dos Scripts.



Na figura 3 estão apresentadas às importações de alguns recursos utilizados na execução da interface gráfica do sistema, em específico, estamos chamando o Bootstrap, o Jquery e o Popper.Js por meio da tag “script”, utilizando “src” para chamar estes recursos através do link de acesso.

Podemos também criar lógicas de programação dentro de um arquivo HTML, no caso, temos uma pequena lógica que vai filtrar os elementos da página por meio de JavaScript, presente na seção “script” dentro do arquivo.

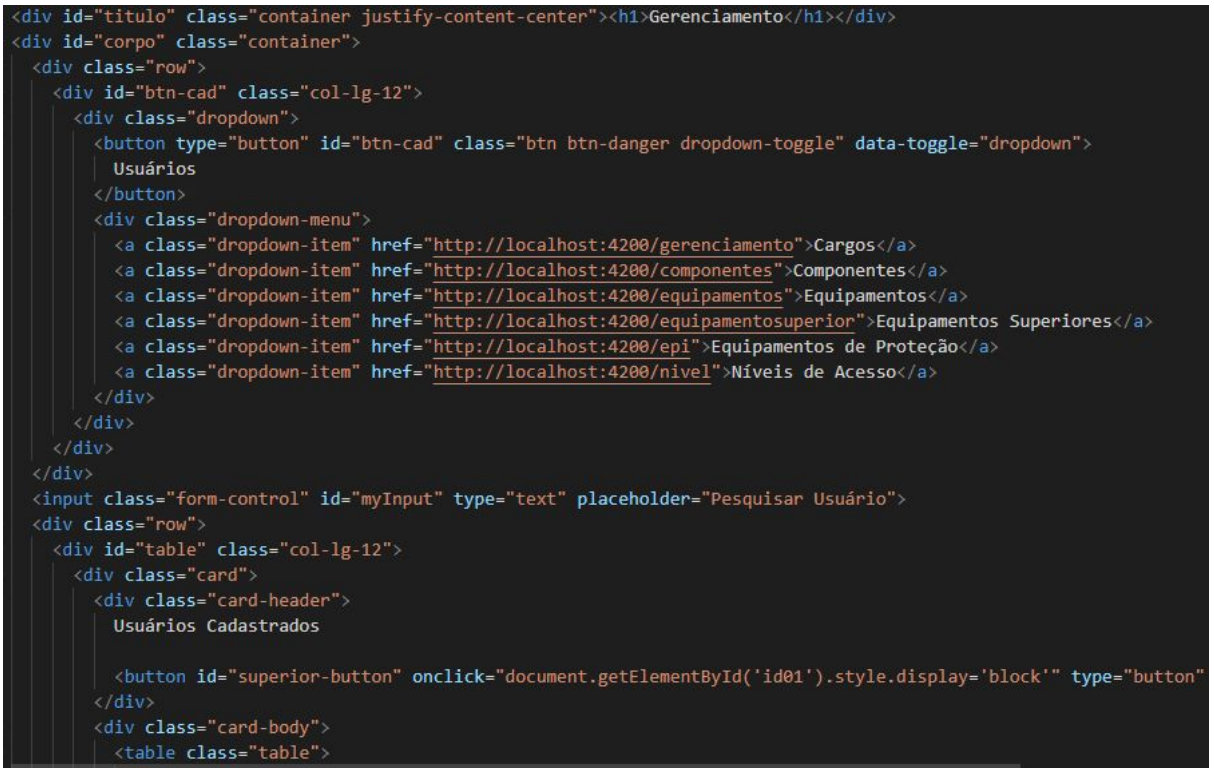
Figura 4 - Lógica e funções JavaScript.



Na figura 4 está sendo demonstrada uma pequena lógica JavaScript que faz a busca de elementos da página de acordo com o caractere digitado, presentes dentro de uma tabela alimentada dinamicamente pelo Back-End com as informações fornecidas pelo banco de dados através da API.

Às outras páginas desenvolvidas para às interfaces do Real Manger utilizam dos recursos apresentados no arquivo HTML principal do AngularJs, estas páginas, como por exemplo a página de gerenciamento de usuários, herdam todas às importações das tags “link” e “script”, permitindo assim a execução destes recursos.

Figura 5 - Arquivo HTML da página de gerenciamento de Usuários.



```
<div id="titulo" class="container justify-content-center"><h1>Gerenciamento</h1></div>
<div id="corpo" class="container">
  <div class="row">
    <div id="btn-cad" class="col-lg-12">
      <div class="dropdown">
        <button type="button" id="btn-cad" class="btn btn-danger dropdown-toggle" data-toggle="dropdown">
          Usuários
        </button>
        <div class="dropdown-menu">
          <a class="dropdown-item" href="http://localhost:4200/gerenciamento">Cargos</a>
          <a class="dropdown-item" href="http://localhost:4200/componentes">Componentes</a>
          <a class="dropdown-item" href="http://localhost:4200/equipamentos">Equipamentos</a>
          <a class="dropdown-item" href="http://localhost:4200/equipamentosuperior">Equipamentos Superiores</a>
          <a class="dropdown-item" href="http://localhost:4200/epi">Equipamentos de Proteção</a>
          <a class="dropdown-item" href="http://localhost:4200/nivel">Níveis de Acesso</a>
        </div>
      </div>
    </div>
  </div>
  <input class="form-control" id="myInput" type="text" placeholder="Pesquisar Usuário">
  <div class="row">
    <div id="table" class="col-lg-12">
      <div class="card">
        <div class="card-header">
          Usuários Cadastrados

          <button id="superior-button" onclick="document.getElementById('id01').style.display='block'" type="button"
        </div>
        <div class="card-body">
          <table class="table">
```

Na figura 5 está sendo apresentado o arquivo HTML da página de gerenciamento de usuários, aqui neste arquivo, todos os elementos gráficos desta página, algumas alimentações, funções, efeito e a estrutura presentes nesta página estão escritos por meio de código, para que sejam interpretadas e transformadas em interface gráfica.

Na parte superior, estão estruturados os elementos que fazem parte do título da página, o menu de navegação com botão drop down, que por meio dos links de acesso, redirecionam paras às outras telas referentes a gerenciamento de informações gerais. Já para as próximas seções do arquivo, teremos os elementos referentes à tabela de usuários cadastrados, os modais para cadastro, visualização e edição das informações referentes aos usuários.

Figura 6 - Código do Elemento da Tabela de Usuários Cadastrados.

```
<input class="form-control" id="myInput" type="text" placeholder="Pesquisar Usuário">
<div class="row">
  <div id="table" class="col-lg-12">
    <div class="card">
      <div class="card-header">
        Usuários Cadastrados

        <button id="superior-button" onclick="document.getElementById('id01').style.display='block'"
      </div>
      <div class="card-body">
        <table class="table">
          <thead class="thead-light">
            <tr>
              <th scope="col">Usuário</th>
              <th>Cargo</th>
              <th>Nível de Acesso</th>
              <th>Opções</th>
            </tr>
          </thead>
          <tbody id="myTable" *ngFor="let usuario of usuarios">
            <tr>
              <td>{{usuario.nome}}</td>
              <td>{{usuario.cargo}}</td>
              <td>{{usuario.nivel}}</td>
              <td><button (click)="informacoesAlterarUsuario(usuario)" onclick="document.getElementB
            </tr>
          </tbody>
        </table>
```

Na figura 6 estão presentes às tags referentes aos elementos da barra de pesquisa para filtro dos usuários cadastrados, abaixo deste elemento, temos os referentes a tabela dos usuários cadastrados, onde é montada uma estrutura, onde as colunas e linhas da tabela ficam prontas para receber as informações requisitadas ao banco de dados, no caso é feita uma interpolação onde nome do usuário, o seu cargo e o seu nível de acesso são mostrados na tela respectivamente nos recursos {{usuario.nome}}, {{usuario.cargo}} e {{usuario.nivel}}.

Figura 7 - Código do Elemento Modal de Cadastros de Usuários.

```
<div id="id01" class="w3-modal">
  <div  class="w3-modal-content">
    <div id="cont" class="w3-container">
      <div class="row">
        <div id="cadastro" class="col-sm">
          <div class="card">
            <div class="card-header">
              Cadastro de Usuários
            </div>
            <div class="card-body">
                <div class="row">
                  <div class="col-md-12">
                    <div class="input-group mb-3">
                      <div class="input-group-prepend">
                        <span class="input-group-text" id="basic-addon1">Nome</span>
                      </div>
                      <input [(ngModel)]="nome"  type="text" class="form-control" aria-label="Username" aria-describedby="basic-addon1">
                    </div>
                    <div class="input-group mb-3">
                      <div class="input-group-prepend">
                        <span class="input-group-text" id="basic-addon1">Senha</span>
                      </div>
                      <input [(ngModel)]="senha"  type="password" class="form-control" aria-label="Username" aria-describedby="basic-addon1">
                    </div>
                    <div class="input-group mb-3">
                      <div class="input-group-prepend">
                        <span class="input-group-text" id="basic-addon1">Crachá</span>
                      </div>
                      <input [(ngModel)]="cracha"  type="text" class="form-control" aria-label="Username" aria-describedby="basic-addon1">
                    </div>
```

Na figura 7 temos os elementos que fazem a construção da interface do modal de cadastro de usuários, neste elemento modal podem ser vistos alguns dos campos que estão responsáveis por enviar as informações inseridas para o Back-End da página, onde estes dados serão tratados e enviados por meio da API até o Banco de Dados.

Além destas duas estruturas na página, nós temos os elementos referentes a edição das informações de cada um dos usuários, onde diferentemente do cadastro, às informações colocadas são alteradas na tabela responsável por armazenar os dados dos usuários.

Figura 8 - Elemento Modal para Edição das Informações dos Usuários.

```
<div id="id02" class="w3-modal">
  <div  class="w3-modal-content">
    <div id="cont" class="w3-container">
      <div class="row">
        <div id="cadastro" class="col-sm">
          <div class="card">
            <div class="card-header">
              Editar Informações do Usuário {{nomeA}}
              <div class="card-body">
                  <div class="row">
                    <div class="col-md-12">
                      <div class="input-group mb-3">
                        <div class="input-group-prepend">
                          <span class="input-group-text" id="basic-addon1">Senha</span>
                        </div>
                        <input [(ngModel)]="senhaA" type="password" class="form-control" aria-label="Username" aria-describedby="basic-addon1">
                      </div>
                      <div class="input-group mb-3">
                        <div class="input-group-prepend">
                          <span class="input-group-text" id="basic-addon1">Cargo</span>
                        </div>
                        <input [(ngModel)]="cargoA" type="text" class="form-control" aria-label="Username" placeholder="Novo Cargo" aria-descri
                      </div>
                      <div class="input-group mb-3">
                        <div class="input-group-prepend">
                          <span class="input-group-text" id="basic-addon1">Nível de Acesso</span>
                        </div>
                        <input [(ngModel)]="nivelA"  type="text" class="form-control" aria-label="Username" aria-describedby="basic-addon1" pla
                      </div>
```

Na figura 8 observa-se a estrutura dos elementos do modal de edição de um usuário, aqui os campos disponíveis com as informações já existentes do usuário são mostrados, para que nu lugar deste, seja adicionada uma nova informação para substituir a antiga.

Por fim, em ambas às estruturas nós temos os elementos que representam os botões, nestes elementos nós definimos qual será o comportamento do software por meio da função “(click)”, esta função vai fazer com que outra função seja executada no Back-End, onde às variáveis referentes aos campos recebam às informações, que a lógica aplicada seja executada e enviar estas informações pelo caminho designado até elas chegarem ao Banco de Dados.

Figura 9 - Elementos dos Botões.

```
<button id="voltar"  type="button" class="btn btn-danger" (click)="cadastraUsuario()">Cadastrar</button>
<button id="voltar" (click)="alterarUsuario()"  type="button" class="btn btn-danger">Alterar</button>
<button id="voltar" onclick="document.getElementById('id02').style.display='none'" type="button" class="btn btn-danger">Voltar</button>
```

Na figura 9 temos os elementos dos botões referentes às funções de cadastro e edição das informações, que fazem o envio das mesmas ao Banco de Dados por meio do Back-End, e também o botão “Voltar”, que fecha o modal por meio de uma função “(click)” que também executa uma função JavaScript para fechar o modal aberto baseado-se no seu ID.

Às demais telas de cadastro do sistema, às quais não possuem as funcionalidades de edição, funcionam da mesma maneira qual foi demonstrada na página de gerenciamento de usuários, cada uma delas com suas respectivas mudanças em nomenclatura de campos e classes, porém funcionando dinamicamente da mesma forma, com elementos modais, funções JavaScript e variáveis de armazenamento, como pode-se ver nas figuras 10, 11 e 12, onde estão representados os elementos HTML da página de gerenciamento de componentes.

Figura 10 - Arquivo HTML da página de Gerenciamento de Componentes.

```
<div id="titulo" class="container justify-content-center"><h1>Gerenciamento</h1></div>
<div id="corpo" class="container">
  <div class="row">
    <div id="btn-cad" class="col-lg-12">
      <div class="dropdown">
        <button type="button" id="btn-cad" class="btn btn-danger dropdown-toggle" data-toggle="dropdown">
          Componentes
        </button>
        <div class="dropdown-menu">
          <a class="dropdown-item" href="http://localhost:4200/gerenciamento">Cargo</a>
          <a class="dropdown-item" href="http://localhost:4200/equipamentos">Equipamentos</a>
          <a class="dropdown-item" href="http://localhost:4200/equipamentosuperior">Equipamentos Superiores</a>
          <a class="dropdown-item" href="http://localhost:4200/epi">Equipamentos de Proteção</a>
          <a class="dropdown-item" href="http://localhost:4200/nivel">Níveis de Acesso</a>
          <a class="dropdown-item" href="http://localhost:4200/usuarios">Usuários</a>
        </div>
      </div>
    </div>
  </div>
    <input class="form-control" id="myInput" type="text" placeholder="Pesquisar Componente">
    <div class="row">
      <div id="table" class="col-lg-12">
        <div class="card">
          <div class="card-header">
            Componentes Cadastrados

            <button id="superior-button" onclick="document.getElementById('id01').style.display='block'" type="button"
          </div>
          <div class="card-body">
            <table class="table">
              <thead class="thead-light">
                <tr>
```

Na figura 10 são vistos parte dos elementos que compõem a estrutura da página de gerenciamento de componentes do sistema Real Manager.

Figura 11 - Elementos da Tabela de Componentes Cadastrados.

```
<input class="form-control" id="myInput" type="text" placeholder="Pesquisar Componente">
<div class="row">
  <div id="table" class="col-lg-12">
    <div class="card">
      <div class="card-header">
        Componentes Cadastrados

        <button id="superior-button" onclick="document.getElementById('id01').style.display='block'
      </div>
      <div class="card-body">
        <table class="table">
          <thead class="thead-light">
            <tr>
              <th scope="col">Componente</th>
              <th scope="col">Valor Unitário</th>
              <th scope="col">Unidade de Medida</th>
            </tr>
          </thead>
          <tbody id="myTable" *ngFor="let componente of componentes">
            <tr>
              <td>{{componente.componente}}</td>
              <td>{{componente.valor}}</td>
              <td>{{componente.unidademedida}}</td>
            </tr>
          </tbody>
        </table>
```

Figura 11, representando a tabela qual possibilita que os usuários tenham acesso às informações dos componentes cadastrados no Banco de Dados por meio das interpolações {{componente.componente}}, {{componente.valor}} e {{componente.unidademedia}}, trazendo respectivamente às informações do nome do componente, o valor de número inteiro ou real e sua unidade, a qual pode ser representada por unidades como Quilograma, grama, litro, ml ou peça.

Figura 12 - Elemento de Modal para Cadastro de um novo Componente.

```
<div class="card-header">
  Cadastro de Componentes
</div>
<div class="card-body">
    <div class="row">
      <div class="col-md-12">
        <div class="input-group mb-3">
          <div class="input-group-prepend">
            <span class="input-group-text" id="basic-addon1">Novo Componente</span>
          </div>
          <input [(ngModel)]="novoComponente"  type="text" class="form-control" aria-label="Username" aria-describedby="basic-
        </div>
        <div class="input-group mb-3">
          <div class="input-group-prepend">
            <span class="input-group-text" id="basic-addon1">Quantidade</span>
          </div>
          <input [(ngModel)]="valor"  type="text" class="form-control" aria-label="Username" aria-describedby="basic-addon1">
        </div>
        <div class="input-group flex-nowrap">
          <div class="input-group-prepend">
            <label class="input-group-text" for="inputGroupSelect01">Unidade de Medida</label>
          </div>
          <select  class="custom-select" id="inputGroupSelect01" [(ngModel)]="unidademedida">
            <option value="g">g(Grama)</option>
            <option value="Kg">Kg(Kilograma)</option>
```

Na figura 12 pode-se observar os elementos referentes a estrutura do modal para cadastro de um novo componente, nele os campos para cadastro e suas respectivas variáveis estão dispostas para que sejam inseridas no Banco de Dados, às informações referentes ao componente que está sendo cadastrado.

3.2 Back-End

Falando agora da estrutura Back-End, entramos na parte mais importante dentro de um sistema, onde as estruturas de todas as funcionalidades lógicas, que envolvem as funções e executam toda a parte funcional do Real Manager estão escritas por meio das linhas de código, cada variável, cada comportamento, função ou rotas designadas. Utilizando o TypeScript como linguagem de programação, às informações inseridas nos campos de cadastro e edição são tratados com objetivo de passarem pelas etapas do Back-End, até de fato chegarem às tabelas e suas linhas e colunas dentro do Banco de Dados.

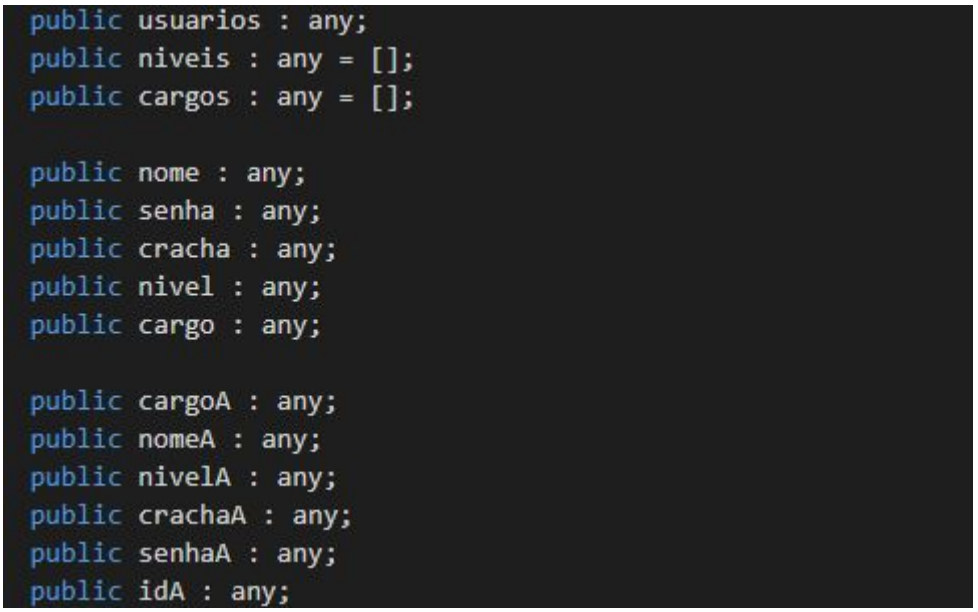
Figura 13 - Arquivo TypeScript da Página de Gerenciamento de Usuários.

```
constructor(private apiService: ApiserviceService) { }

ngOnInit(): void {
  this.apiService.getUSU().subscribe((data:any)=>{
    this.usuarios = data;
  });
  this.apiService.getCargos().subscribe((data:any)=>{
    this.cargos = data;
  });
  this.apiService.getNiveis().subscribe((data:any)=>{
    this.niveis = data;
  });
}

cadastraUsuario() {
  this.apiService.cadastraUsuario(this.nome, this.senha, this.cracha, this.nivel, this.cargo)
}

alterarUsuario() {
  this.apiService.alterarUsuario(this.nomeA, this.senhaA, this.crachaA, this.nivelA, this.cargoA, this.idA);
}

informacoesAlterarUsuario(usuarios) {
  this.nivelA = usuarios.nivel;
  this.nomeA = usuarios.nome;
  this.crachaA = usuarios.cracha;
  this.cargoA = usuarios.cargo;
  this.idA = usuarios.idUS;
```

Na figura 13 estão apresentados os elementos lógicos, as variáveis e funções necessárias para executar as funcionalidades da página de Gerenciamento de Usuários.

Às variáveis são objetos que tem a capacidade de reter e representar um valor ou expressão, cada variável tem um identificador, que geralmente é um nome dado a ela e um tipo, existem diversos tipos de variáveis como “string” para caracteres alfabéticos, “int” para números inteiros e entre outras. Às variáveis podem ter duas classificações, às globais que são declaradas fora do escopo das funções e que podem ou não serem acessadas de qualquer lugar do código em específico, e também às variáveis locais, estas são acessíveis somente dentro do bloco onde seus escopos foram declarados.

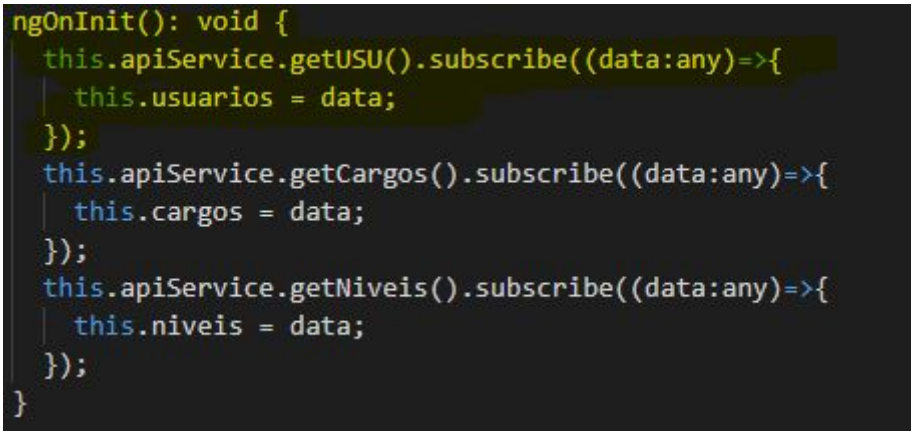
Figura 14 - Variáveis no Back-End da Página de Gerenciamento de Usuários.



Na figura 14 observam-se às variáveis declaradas no arquivo TypeScript de forma global, pois podem ser acessadas por qualquer função dentro deste arquivo, as variáveis nome, senha, crachá, nível e cargo, são responsáveis por armazenar todas as informações inseridas nos campos referentes a elas no Front-End da página. Já as variáveis usuarios, niveis e cargos, recebem os valores que chegam do Banco de Dados, passando ela por meio de interpolação para o Front-End onde serão disponibilizadas e mostradas aos usuários do sistema. E por último às variáveis cargoA, nomeA, nivelA, crachaA, senhaA e idA, que são responsáveis pela parte das informações de alteração de um usuário.

As funções são um conjunto de comandos que realizam uma ou mais tarefas específicas com seus objetivos designados, por meio delas aplicamos funcionalidades lógicas dentro do sistema, podemos chamar variáveis dentro das funções para executar e atender às necessidades solicitadas, como por exemplo cadastrar e inserir informações em um Banco de Dados.

Figura 15 - Funções da Página de Gerenciamento de Usuários.



Na figura 15 é mostrada a função que requisita às informações para a API, que por sua vez pega às informações do Banco de Dados e as retorna para a função, assim a função recebe estas informações direcionando-as para a variável responsável fazer o armazenamento destas informações que neste caso, estão vindo no formato de lista, após isso a variável será chamada no Front-End, assim mostrando a lista de informações para os usuários quando a página é carregada.

Para efetuarmos a comunicação entre o código TypeScript das páginas, e a API responsável por comunicar com o Banco de Dados, utiliza-se um Provider, o qual vai realizar a comunicação das informações do Back-End da página, com a API, é por meio dele que por exemplo, um “get” é executado, neste caso, o Back-End que solicita ao Provider que seja executada uma determinada lógica passando as informações e parâmetros necessários para a API, onde estes são atribuídos ao endpoint responsável por fazer a requisição ao Banco de Dados.

Figura 16 - Arquivo TypeScript do Provider.

```typescript
@Injectable({
  providedIn: 'root'
})
export class ApiserviceService {

  apiUrl = 'http://localhost:3000';

  constructor(private http: HttpClient) { }

  /*Gerenciamentos*/
  /*Epi´s*/
  /*Lista de Epi´s*/
  public getEpis(): Observable<any>{
    return this.http.get(this.apiUrl + '/getEPI')
  };
  /*Cadastra Epi*/
  public cadastraEPI(epi, catergoria, corpo, numero){
    let EPIS: any = null;

    EPIS={
      epi: epi,
      categoria: catergoria,
      corpo: corpo,
      numero: numero
    };

    this.http.post(this.apiUrl + '/cadastraEPI', EPIS).subscribe(
      (data:any)=>{
        setTimeout(function(){document.location.reload(true);; }, 1500);
        Swal.fire({
          position: 'center',
          icon: 'success',
```

A figura 16 demonstra a estrutura do Provider que é responsável pela comunicação com a API, nele são executadas às lógicas que trocam informações com a API, definindo o que deve ser requisitado, cadastrado ou alterado dentro do Banco de Dados.

Nas próximas imagens temos algumas funções do Provider que são responsáveis pela lógica que envia as informações recebidas do arquivo TypeScript de uma página até a API do sistema Real Manager, utilizando o método “POST”.

Figura 17 - Função de Cadastro de Usuários do Provider.

```
public cadastraUsuario(nome, senha, cracha, nivel, cargo){
  let Usuarios: any = null;

  Usuarios={
    nome: nome,
    senha: senha,
    cracha: cracha,
    nivel: nivel,
    cargo: cargo

  };

  this.http.post(this.apiUrl + '/cadastroUSU', Usuarios).subscribe(
    (data:any)=>{
      setTimeout(function(){document.location.reload(true);; }, 1500);
      Swal.fire({
        position: 'center',
        icon: 'success',
        title: 'Novo Usuário Cadastrado!',
        showConfirmButton: false,
        timer: 1500
      });
    },
    (error:any)=>{
      Swal.fire({
        position: 'center',
        icon: 'error',
        title: 'Ocorreu um Erro!',
        showConfirmButton: false,
        timer: 1500
      });
    }
```

Nesta figura, observa-se a função que executa a lógica responsável por receber os dados inseridos nos campos atribuídos às variáveis do arquivo TypeScript da página de Gerenciamento de Usuários, especificamente dos campos presentes no modal de Cadastro de Usuários, dados quais vão ser enviados ao endpoint responsável por inserir estes no Banco de Dados.

Todos os dados que são fornecidos pelo código TypeScript de uma página, são passados para dentro de uma lógica no Provider, depois são inseridos dentro do objeto definido, o qual será mandando junto com o endereço do endpoint, para que quando chegarem a esse endpoint dentro da API, serem utilizados e passados para os campos e tabelas definidos. Observe a utilização das lógicas no decorrer das etapas através das figuras a seguir!

Etapa número 1: Inserção dos dados nos campos referentes às informações de uma Ordem Preventiva.

Figura 18 - Cadastro de Ordem Preventiva arquivo HTML.

```
<div class="row">
  <div id="cadastro" class="col-sm">
    <div class="card">
      <div class="card-header">
        Informações Principais
      </div>
      <div class="card-body">
        <div class="row">
          <div id="cadastro" class="col-sm-6">
            <div class="input-group flex-nowrap">
              <input type="text" class="form-control" placeholder="Número da Ordem" [(ngModel)]="numeroordemP">
            </div>
          </div>
          <div id="cadastro" class="col-sm-6">
            <div class="input-group flex-nowrap">
              <input type="text" class="form-control" placeholder="Título" [(ngModel)]="tituloP">
            </div>
          </div>
      </div>
      <div class="row">
        <div id="cadastro" class="col-sm-6">
          <div class="input-group flex-nowrap">
            <input type="text" class="form-control" placeholder="Solicitante" [(ngModel)]="solicitanteP">
          </div>
        </div>
        <div id="cadastro" class="col-sm-6">
          <div class="input-group flex-nowrap">
            <div class="input-group-prepend">
              <span class="input-group-text" id="basic-addon3">Emissão</span>
            </div>
            <input type="date" class="form-control" placeholder="Emissão" [(ngModel)]="emissaoP">
          </div>
        </div>
    </div>
    <div class="row">
      <div id="cadastro" class="col-sm-12">
        <div class="input-group flex-nowrap">
          <div class="input-group-prepend">
            <label class="input-group-text" for="inputGroupSelect01">Responsável</label>
          </div>
          <select  class="custom-select" id="inputGroupSelect01" [(ngModel)]="responsavelP">
            <option *ngFor="let manutentor of manutentor">{{manutentor.nome}}</option>
          </select>
        </div>
      </div>
  </div>
```

Nesta figura é possível ver a estrutura modal e os elementos e campos que recebem as informações inseridas desde o número da ordem até o mecânico responsável, de forma com que esses dados vão ser atribuídos às suas respectivas variáveis dentro do arquivo TypeScript da Página de Abertura de Ordem Preventiva.

Figura 19 - Datas e Horários de uma Ordem Preventiva.

```
<div id="cadastro" class="col-sm">
  <div class="card">
    <div class="card-header">
      Datas e Horários
    </div>
    <div class="card-body">
        <div class="row">
          <div class="col-md-6">
            <div class="input-group mb-3">
              <div class="input-group-prepend">
                <span class="input-group-text" id="basic-addon3">Inicio Planejado</span>
              </div>
              <input type="date" class="form-control" id="basic-url" aria-describedby="basic-addon3" [(ngModel)]="inicioP">
            </div>
          </div>
          <div class="col-md-6">
            <div class="input-group mb-3">
              <div class="input-group-prepend">
                <span class="input-group-text" id="basic-addon3">Fim Planejado</span>
              </div>
              <input type="date" class="form-control" id="basic-url" aria-describedby="basic-addon3" [(ngModel)]="fimpP">
            </div>
          </div>
        </div>
        <div class="row">
          <div class="col-md-6">
            <div class="input-group mb-3">
              <div class="input-group-prepend">
                <span class="input-group-text" id="basic-addon3">Inicio Programado</span>
              </div>
              <input type="date" class="form-control" id="basic-url" aria-describedby="basic-addon3" [(ngModel)]="inicioproP">
            </div>
          </div>
          <div class="col-md-6">
            <div class="input-group mb-3">
              <div class="input-group-prepend">
                <span class="input-group-text" id="basic-addon3">Fim Programado</span>
              </div>
              <input type="date" class="form-control" id="basic-url" aria-describedby="basic-addon3" [(ngModel)]="fimproP">
            </div>
          </div>
        </div>
```

Na figura 13 estão sendo apresentados os elementos para os campos de inserção de dados referentes às datas e horários de uma Ordem Preventiva.

Etapa número 2: Os dados inseridos nos campos do Front-End tem os elementos e campos atribuídos às variáveis responsáveis por armazenar estes dados.

Figura 20 - Variáveis referentes aos campos de cadastro de uma Ordem Preventiva.

```
export class NewpreventivaComponent implements OnInit {

  public numeroordemP: any;
  public tituloP: any;
  public solicitanteP: any;
  public emissaoP: any;
  public descricaoproblemaP: any;
  public responsavelP: any;
  public localdeinstalacaoP: any;
  public equipamentoP: any;
  public equipamentoSuP: any;
  public prioridadeP: any;
  public requerparadaP: any;
  public horainicioproP: any;
  public horafimproP: any;
  public iniciopP: any;
  public fimpP: any;
  public inicioproP: any;
  public fimproP: any;
  public tipodemanutencaoP = 'Preventiva';
  public statusordemP = 'Aberta';
```

Na figura 20 estão dispostas às variáveis que recebem as informações que chegam do Front-End por meio dos campos atribuídos a estas variáveis, com exceção do tipo da ordem, que já vem pré-definida e o status da ordem, que ao final do cadastro é caracterizado como “Aberta”.

Etapa número 3: Às variáveis são chamadas dentro da função responsável por executar a lógica que irá realizar o envio destas informações para o Provider do sistema.

Figura 21 - Função que realiza o envio das variáveis até o Provider.

```
cadastraOrdemP(){
  this.apiService.cadastraOrdemP(this.numeroordemP, this.tituloP, this.solicitanteP, this.emissaoP,
  this.descricaoproblemaP, this.responsavelP, this.localdeinstalacaoP, this.equipamentoP, this.equipamentoSuP,
  this.prioridadeP, this.requerparadaP, this.horainicioproP, this.horafimproP, this.iniciopP, this.fimpP, this.inicioproP,
  this.fimproP, this.tipodemanutencaoP, this.statusordemP)
};
```

Na figura 21 pode ser observada a função que vai realizar o envio de todas as variáveis que armazenam as informações do cadastro de uma Ordem Preventiva até o Provider.

Etapa número 4: Dentro do Provider às informações recebidas são atribuídas às suas nomenclaturas, quais serão utilizadas para atribuir às suas posições dentro da estrutura do endpoint, dentro da função que executa toda a lógica necessária para isso acontecer.

Figura 22 - Objeto que define as variáveis das informações dentro de um Objeto.

```
public cadastraOrdemP(numeroordemP, tituloP, solicitanteP, emissaoP, descricaoproblemaP
  let ordemP: any = null;

  ordemP={
    numeroordemP: numeroordemP,
    tituloP: tituloP,
    solicitanteP: solicitanteP,
    emissaoP: emissaoP,
    descricaoproblemaP: descricaoproblemaP,
    responsavelP: responsavelP,
    localdeinstalacaoP: localdeinstalacaoP,
    equipamentoP: equipamentoP,
    equipamentosuperiorP: equipamentoSuP,
    prioridadeP: prioridadeP,
    requerparadaP: requerparadaP,
    iniciopP: iniciopP,
    fimpP: fimpP,
    inicioproP: inicioproP,
    fimproP: fimproP,
    horainicioproP: horainicioproP,
    horafimproP: horafimproP,
    tipodemanutencaoP: tipodemanutencaoP,
    statusordemP: statusordemP

  };
```

Nesta figura são mostradas todas as variáveis responsáveis pelas informações dentro do objeto “ordemP”, referente a Ordem Preventiva.

Etapa número 5: O objeto junto ao endereço do endpoint entram em uma lógica que vai aplicar o método "POST", enviando às informações para a API caso esteja tudo certo, caso haja algum erro, ele não envia os dados e avisa que ocorreu um erro.

Figura 23 - Lógica aplicada ao endereço do endpoint e ao Objeto com as Variáveis.

```
this.http.post(this.apiUrl + '/cadastroUSU', Usuarios).subscribe(
  (data:any)=>{
    setTimeout(function(){document.location.reload(true);; }, 1500);
    Swal.fire({
      position: 'center',
      icon: 'success',
      title: 'Novo Usuário Cadastrado!',
      showConfirmButton: false,
      timer: 1500
    });
  },
  (error:any)=>{
    Swal.fire({
      position: 'center',
      icon: 'error',
      title: 'Ocorreu um Erro!',
      showConfirmButton: false,
      timer: 1500
    });
  }
);
```

Na figura 23 observa-se a lógica aplicada ao endereço do endpoint e ao Objeto que contém as variáveis, esta lógica vai enviar todos a API caso esteja tudo ocorrendo da forma adequada, caso ocorra algum tipo de erro, relacionada às variáveis, o envio não ocorrerá e uma mensagem de erro será exibida ao Usuário.

Etapa número 6: Dentro da API, todas as variáveis recebidas do Provider entram no endpoint endereçado na função que executou o envio, dentro deste endpoint nós temos a query SQL para passar ao banco de dados todas as informações para serem cadastradas nas colunas, passando o nome das colunas e o valor que será inserido nela, finalizando este processo, a API manda a query para o gerenciador do banco de dados realizar a inserção dos mesmos no banco.

Figura 24 - Endpoint responsável por cadastrar as informações no Banco de Dados.

```
app.post("/cadastroOrdemPreventiva", (req, res)=>{
    let sql = 'insert into ordempreventiva(titulo, solicitante, equipamentosuperior,equipamento,
    new MySQLFactory().getConnection().select(sql).subscribe(
        (data:any)=>{
            res.send(data);
        },
        (error:any)=>{
            res.status(405).send('Erro')
        }
    );
});
```

A figura 24 mostra o endpoint responsável por cadastrar as informações no Banco de Dados, por meio da query, são definidas as colunas da tabela e os valores que serão inseridos em cada uma delas, como por exemplo, coluna tìtulo recebe o valor título proveniente da variável título presente dentro do objeto enviado.

A parte de busca das informações no banco de dados também possui suas etapas a serem executadas para que estas cheguem até o Usuário por meio da interface gráfica, este processo ocorre quando todas as informações requisitadas são guardadas em variáveis do tipo lista, e interpoladas no Front-End.

Ao carregar a tela, a função pré definida dentro do “ngOnInit” de um arquivo TypeScript, para que a função aconteça junto a renderização da página, para que as informações estejam prontas e inseridas nos elementos corretos.

Figura 25 - Função que faz a busca das informações de Ordens e suas variáveis.

```
export class OrdensComponent implements OnInit {

  public omsP: any;
  public omsC: any;
  public omsPA: any;
  public omsCA: any;
  public omsPF: any;
  public omsCF: any;
  public teste: any;
  data=[];


  public insertC: any;
  public insertP: any;

  constructor(private apiserviceService: ApiserviceService) { }

  ngOnInit(): void {
    this.apiserviceService.getOmC().subscribe((data: []) =>{
      this.omsC = data;
    });
    this.apiserviceService.getOmP().subscribe((data: []) =>{
      this.omsP = data;
    });
  }
```

Na figura 25 observam-se as variáveis que armazenam as informações requisitadas da API, que chegam no arquivo TypeScript da página por meio do Provider, informações estas que estão sendo disponibilizadas por meio das variáveis a serem chamadas nos elementos do Front-End da página.

É requisitada ao provider a execução da função responsável com o método “GET”, junto ao endereço do endpoint responsável.

Figura 26 - Função de busca e endereçamento do endpoint.

```
public getOmP(): Observable<any>{
  return this.http.get(this.apiUrl + '/getOrdemPreventiva');
};
```

Na figura 26 situa-se a função responsável por requisitar as informações ao endpoint, e retorná-las ao arquivo TypeScript.

Na API o endpoint responsável executa as lógicas e a query que vão solicitar ao Banco de Dados às informações a serem retornadas ao Provider e consequentemente ao arquivo TypeScript da página.

Figura 27 - Endpoint dentro da API.

```
app.get("/getOrdemPreventiva", (req, res)=>{
    let sql = 'select * from ordempreventiva where status="Aberta"';
    new MySQLFactory().getConnection().select(sql).subscribe(
        (data:any)=>{
            res.send(data);
        },
        (error:any)=>{
            res.status(405).send('Erro');
        }
    );
});
```

A figura 27 demonstra o endpoint responsável por requisitar ao Banco de Dados todas as Ordens de Manutenção Preventivas com status “Aberta”, cadastradas na tabela presente no Banco de Dados.

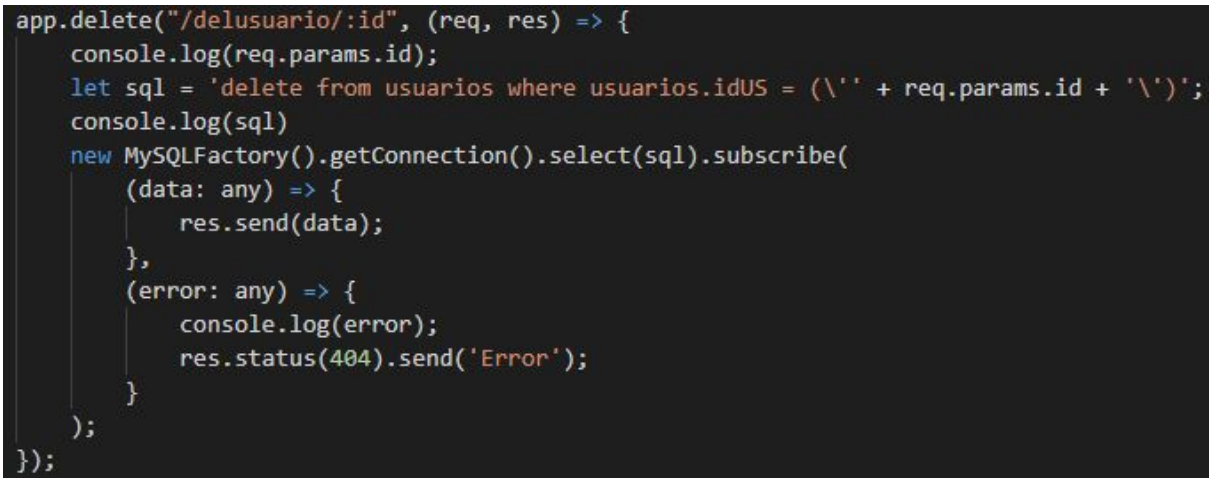
Falando especificamente dos endpoints presentes dentro da API, nós temos mais dois tipos de métodos utilizados nas queries, o método “UPDATE”, que nos permite encaixar novos valores a serem subscritos no lugar dos antigos, utilizado para realizar a edição de informações do sistema, e o “DELETE”, que como o nome já diz, servem para excluir informações dentro do Banco de Dados, geralmente utilizando como parâmetro de comparação, para podermos editar ou excluir determinado cadastro. Como observado na Figura 28 e 29.

Figura 28 - Query com método “UPDATE” no endpoint.

```
app.post("/alteraUSU/:id", (req, res)=>{
    let sql = 'update usuarios set nome=(\'' + req.body.nome + '\'), senha=(\'' + req.body.senha + '\'),
    new MySQLFactory().getConnection().select(sql).subscribe(
        (data:any)=>{
            res.send(data);
        },
        (error:any)=>{
            res.status(405).send('Erro');
        }
    );
});
```

Na figura 28 temos o endpoint dentro da API, que realiza a alteração das informações passadas por variáveis vindas da página de Gerenciamento de Usuários, especificamente da parte de alteração de usuários, nesta query podemos ver o método UPDATE sendo utilizado para sobrescrever as informações nas colunas da tabela de usuários.

Figura 29 - Query com método “DELETE” no endpoint.



```
app.delete("/delusuario/:id", (req, res) => {
    console.log(req.params.id);
    let sql = 'delete from usuarios where usuarios.idUS = (\'' + req.params.id + '\')';
    console.log(sql)
    new MySQLFactory().getConnection().select(sql).subscribe(
        (data: any) => {
            res.send(data);
        },
        (error: any) => {
            console.log(error);
            res.status(404).send('Error');
        }
    );
});
```

Figura 29 demonstrando um endpoint que faz a exclusão dos dados de determinada tabela do banco de dados baseando-se na coluna que armazena os id´s de usuários.

4 Requisitos Mínimos

Os requisitos mínimos para rodar o sistema Real Manager seriam acesso a um navegador, pois é um sistema Web construído graficamente em HTML5 e CSS3, com linguagem de programação TypeScript e utilizar um servidor gerenciado para realizar todos os procedimentos que envolvem informações do sistema.