

Curso: Análise e Desenvolvimento de Sistemas – ADS

Ano: 2023/1

Orientação Técnica (OT) – 23

Edição REST - Preparando formulário

Introdução

A alteração é a última etapa do nosso CRUD (mas não a última parte da trilha...). Para realizá-la por completo, vamos fazer o seguinte: assim que o usuário clicar no botão de alterar, ele verá um formulário similar ao de inserção, porém já preenchido com os dados do produto selecionado, para que possa alterar apenas o que deseja, e salvar as alterações realizadas.

Este documento tem o intuito de lhe guiar na primeira parte do processo: até a exibição dos dados do produto selecionado no formulário. *Simbora!*

Codificação HTML e CSS do formulário

Quanto ao formulário, poderíamos fazer de dois modos diferentes: criar um novo formulário, ou carregar os dados no formulário que já possuímos. Nesse momento, para evitar confusões e também devido ao nosso layout, é mais interessante fazermos a criação de um novo formulário, exclusivo para a alteração. Destarte, abra a página HTML desse CRUD, a **product/index.html**, e ao final dela (conforme destacado na imagem), insira o formulário de alteração conforme as imagens seguintes:

```
<div id="listaProdutos" class="listaRegistros">
</div>
<div id="modalEditaProduto" class="modalEditar">
  <form name="frmEditaProduto" id="editaProduto" class="frmEditar">
    <table>
      <tr>
        <th>Categoria</th>
        <td>
          <select name="categoria" id="selCategoriaEdicao">
            <option value="">Selecione</option>
            <option value="1">Geladeira</option>
            <option value="2">Freezer</option>
          </select>
        </td>
      </tr>
      <tr>
        <th>Marca</th>
        <td>
          <select name="marcaId" id="selMarcaEdicao">
            <option value="" class="buscando">Aguarde, buscando marcas...</option>
          </select>
        </td>
      </tr>
      <tr>
        <th>Modelo</th>
        <td>
          <input type="text" name="modelo">
        </td>
      </tr>
    </table>
  </form>
</div>
```

Aqui, criamos uma **div** para que nosso formulário seja aberto em uma **janela modal**; abrimos o **formulário**; criamos uma **tabela** para organizá-lo; criamos os **campos** para **categoria** (select), para **marca** (select, que como no outro formulário buscará no BD as marcas registradas) e para o **modelo** do produto (texto).

Continuando...

```
<tr>
  <th>Capacidade(l)</th>
  <td>
    <input type="text" name="capacidade">
  </td>
</tr>
<tr>
  <th>Valor (R$)</th>
  <td>
    <input type="text" name="valor">
  </td>
</tr>
</table>
<input type="hidden" name="idProduto">
</form>
</div>
```

Aqui, criamos **mais 3 campos**: **capacidade** (texto, e antes que vacile, aquilo é uma letra L, de litro, e não o número um...), **valor** (texto) e um carinha novo: um **input type hidden** chamado **idProduto**.

Pesquise sobre esse tipo de input, e discutiremos na validação...

Além disso, fechamos a tabela, o formulário e a div.

Agora, precisamos estilizar esse formulário. Abra o arquivo **admin.css** e apenas complemente o código que faz as modais ficarem ocultas por padrão, adicionando a classe da div da nova modal.

```
/****** Modais *****/
#modalAviso, .modalEditar{
    display: none;
}
```

Feito isto, finalizamos as codificações HTML e CSS da alteração.

Função JS de carregamento do formulário

Em um processo parecido com o que fizemos na exclusão, é o botão ao lado do produto na lista de produtos que deve ativar a função que carregará o formulário de edição na tela. Assim, no **product.js**, encontre a parte em que o botão de alteração é criado e adicione nele o código em destaque:

```
for (var i=0; i<listaDeProdutos.length; i++){
    tabela += "<tr>" +
        "<td>"+listaDeProdutos[i].categoria+"</td>" +
        "<td>"+listaDeProdutos[i].marcaNome+"</td>" +
        "<td>"+listaDeProdutos[i].modelo+"</td>" +
        "<td>"+listaDeProdutos[i].capacidade+"</td>" +
        "<td>R$ "+COLDIGO.formatarDinheiro(listaDeProdutos[i].valor)+"</td>" +
        "<td>" +
            "<a onclick='\"COLDIGO.produto.exibirEdicao(\""+listaDeProdutos[i].id+"')\"'><img src='../imgs/edit.png'" +
            "<a onclick='\"COLDIGO.produto.excluir(\""+listaDeProdutos[i].id+"')\"'><img src='../imgs/delete.png' al" +
        "</td>" +
        "</tr>"
}
```

Em seguida, abaixo da função que criou para a exclusão, mas ainda dentro do `$(document).ready`, crie a função que deve realizar a requisição dos dados do registro selecionado ao servidor conforme abaixo:

```
//Carrega no BD os dados do produto selecionado para alteracao e coloca-os no formulário de alteração
COLDIGO.produto.exibirEdicao = function(id){
    $.ajax({
        type: "GET",
        url: COLDIGO.PATH + "produto/buscarPorId",
        data: "id="+id,
        success: function(produto){

        },
        error: function(info){
            COLDIGO.exibirAviso("Erro ao buscar produto para edição: " + info.status + " - " + info.statusText);
        }
    });
};
```

Veja que o código não tem segredos, já fizemos o que está aqui em outros momentos, então dispensamos comentários. Apenas repare que deixamos a função *success* vazia, para podermos fazer sua codificação posteriormente.

Lado servidor: Carregando os dados do produto e retornando ao cliente

Para começar, vá até a classe **ProdutoRest** e no final dela, crie o método de busca por id, conforme imagem abaixo:

```
@GET
@Path("/buscarPorId")
@Consumes("application/*")
@Produces(MediaType.APPLICATION_JSON)
public Response buscarPorId(@QueryParam("id") int id){

    try{

    }catch(Exception e){
        e.printStackTrace();
        return this.buildErrorResponse(e.getMessage());
    }
}
```

Veja que não há erros devido a falta de *import*, pois tudo o que está aqui já foi usado antes nesta classe! Justamente por isso, já colocamos desde o início também o famoso try-catch.

Agora, iniciaremos o código do try. Copie nele o que está destacado na imagem abaixo, então:

```
public Response buscarPorId(@QueryParam("id") int id){

    try{
        Produto produto = new Produto();
        Conexao conec = new Conexao();
        Connection conexao = conec.abrirConexao();
        JDBCProdutoDAO jdbcProduto = new JDBCProdutoDAO(conexao);
    }catch(Exception e){
        e.printStackTrace();
        return this.buildErrorResponse(e.getMessage());
    }
}
```

Veja que, além do relativo ao BD de sempre, **criamos um objeto da classe Produto**.

Consegue identificar o porquê?

Uma vez conectados ao banco, podemos chamar o método responsável pela busca do produto a ser alterado. Crie assim a linha entre os dois destaques em seu código:

```
try{
    Produto produto = new Produto();
    Conexao conec = new Conexao();
    Connection conexao = conec.abrirConexao();
    JDBCProdutoDAO jdbcProduto = new JDBCProdutoDAO(conexao);

    produto = jdbcProduto.buscarPorId(id);
} catch (Exception e){
```

Mais uma vez, antes de agirmos no BD, precisamos primeiramente sinalizar a necessidade do método de busca de um produto por seu id na interface **ProdutoDAO**, portanto insira nele a linha destacada na imagem abaixo.

```
public interface ProdutoDAO {

    public boolean inserir(Produto produto);
    public List<JsonObject> buscarPorNome(String nome);
    public boolean deletar(int id);
    public Produto buscarPorId(int id);
}
```

Agora sim, vamos à classe **JDBCProdutoDAO** criar o método ao qual estamos nos referindo: o **buscarPorId**. Para isso, logo abaixo do último método programado nessa classe, insira o método da imagem a seguir:

```
public Produto buscarPorId(int id) {
    String comando = "SELECT * FROM produtos WHERE produtos.id = ?";
    Produto produto = new Produto();
    try {
        PreparedStatement p = this.conexao.prepareStatement(comando);
        p.setInt(1, id);
        ResultSet rs = p.executeQuery();
        while (rs.next()) {

        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return produto;
}
```

Veja que não há nada de muito especial aqui, com apenas uma exceção (algo que ainda não havíamos feito desse jeito). Consegue reconhecer o que é?

Continuando... Vamos agora tratar da parte que deve ser feita a cada registro correspondente à pesquisa, então copie o código entre as áreas destacadas:

```
try {
    PreparedStatement p = this.conexao.prepareStatement(comando);
    p.setInt(1, id);
    ResultSet rs = p.executeQuery();
    while (rs.next()) {

        String categoria = rs.getString("categoria");
        String modelo = rs.getString("modelo");
        int capacidade = rs.getInt("capacidade");
        float valor = rs.getFloat("valor");
        int marcaId = rs.getInt("marcas_id");

        produto.setId(id);
        produto.setCategoria(categoria);
        produto.setMarcaId(marcaId);
        produto.setModelo(modelo);
        produto.setCapacidade(capacidade);
        produto.setValor(valor);

    }
} catch (Exception e) {
```

Veja que pegamos **todos os dados do produto, menos o id**. Já da **marca respectiva, pegamos justamente o id**: isso será **importante na exibição** dela no formulário.

Agora, voltemos à classe **ProdutoRest**, para fechamos a conexão e retornarmos os dados do produto encontrado (ou um possível erro) ao usuário:

```
try{
    Produto produto = new Produto();
    Conexao conec = new Conexao();
    Connection conexao = conec.abrirConexao();
    JDBCProdutoDAO jdbcProduto = new JDBCProdutoDAO(conexao);

    produto = jdbcProduto.buscarPorId(id);

    conec.fecharConexao();

    return this.buildResponse(produto);
}catch(Exception e){
```

Assim, encerramos a programação relativa ao backend até o momento!



Primeiro teste

Voltemos ao lado frontend, pelo arquivo **product.js**. Nele, insira o comando **console.log** dentro da função de **success do Ajax**, conforme abaixo:

```
COLDIGO.produto.exibirEdicao = function(id){
  $.ajax({
    type: "GET",
    url: COLDIGO.PATH + "produto/buscarPorId",
    data: "id="+id,
    success: function(produto){
      console.log(produto);
    },
    error: function(info){
      COLDIGO.exibirAviso("Erro ao buscar produ
    }
  });
};
```

Como teste, **clique no botão de edição de um produto**, e observe a **console do navegador**: se os dados do registro referentes ao botão de alterar que você clicou aparecerem, estamos bem!

Para ter uma exemplo de referência, veja na imagem abaixo um registro de produto:

Categoria	Marca	Modelo	Cap.(l)	Valor	Ações
Geladeira	Prosdócimo	SP500	255	R\$ 1.200,00	 

Se clicarmos no botão de alterar dele, aparecerá o seguinte na console do navegador:.

```
VM33:180
{id: 4, categoria: "1", marcaId: 3, modelo: "SP500", capacidade: 255, ...}
  capacidade: 255
  categoria: "1"
  id: 4
  marcaId: 3
  modelo: "SP500"
  valor: 1200
  __proto__: Object
```

Claro, confira se os dados batem: se tudo o que aparece é realmente daquele registro, afinal de contas só isso vai garantir que não aconteceu algo como você ter clicado em um produto e outro aparecer...

ATENÇÃO: tá, eu já sei, se der algo errado devo corrigir ou avisar... Isso mesmo, muito bom, jovem padawan!

Exibindo os dados no formulário de edição

Agora que sabemos que os dados estão chegando no lado cliente, **retire o console.log** e **insira o que colocamos na imagem abaixo** em seu lugar:

```
success: function(produto){  
  
    document.frmEditaProduto.idProduto.value = produto.id;  
    document.frmEditaProduto.modelo.value = produto.modelo;  
    document.frmEditaProduto.capacidade.value = produto.capacidade;  
    document.frmEditaProduto.valor.value = produto.valor;  
  
},  
error: function(info){
```

Veja que **colocamos nos 4 campos do formulário** seus valores: id, modelo, capacidade e valor do produto. Assim, **faltam apenas os 2 campos select**, que são **um pouco diferentes**, afinal não será suficiente escrever um texto nos campos, temos que **fazer com que a opção correta já apareça selecionada** assim que o formulário for carregado!

O primeiro campo select que prepararemos, a categoria, é o mais simples, pois ele **NÃO VEM DO BD!** Assim, insira o novo código conforme destacado na imagem:

```
document.frmEditaProduto.capacidade.value = produto.capacidade;  
document.frmEditaProduto.valor.value = produto.valor;  
  
var selCategoria = document.getElementById('selCategoriaEdicao');  
for(var i=0; i < selCategoria.length; i++){  
    if (selCategoria.options[i].value == produto.categoria){  
        selCategoria.options[i].setAttribute("selected", "selected");  
    }else{  
        selCategoria.options[i].removeAttribute("selected");  
    }  
}  
  
},  
error: function(info){
```

Explicando, primeiro recebemos o **select** existente em uma variável, usando seu id. Depois, criamos **uma estrutura de repetição** que, **para cada option** desse select, verificará **se o valor dela é igual ao valor da categoria do produto carregado do BD**: se sim, insere nela o **atributo selected**; se não, remove esse atributo.

Aliás, para que serve o atributo selected mesmo?

Como próximo passo, **vamos pular o select de marcas** (já vai entender o motivo...), e ir para a **criação da janela modal na qual exibiremos o formulário**.

Assim, insira o código entre as linhas destacadas em amarelo no seu projeto:

```
    }else{
        selCategoria.options[i].removeAttribute("selected");
    }
}

var modalEditaProduto = {
    title: "Editar Produto",
    height: 400,
    width: 550,
    modal: true,
    buttons:{
        "Salvar": function(){

        },
        "Cancelar": function(){
            $(this).dialog("close");
        }
    },
    close: function(){
        //caso o usuário simplesmente feche a caixa de edição
        //não deve acontecer nada
    }
};

$("#modalEditaProduto").dialog(modalEditaProduto);
},
error: function(info){
```

Veja que **configuramos a janela modal** (só não dissemos ainda o que deve ser feito ao se clicar no botão “Salvar”), e, com a função jQuery UI **Dialog**, transformamos a **div de id modalEditaProduto** (que criamos acima ainda nesta OT) em uma janela **modal**.

Agora **salve, teste**, e se tudo estiver OK, o formulário irá aparecer com todos os dados exceto marcas carregados e a modal terá botões de Salvar e Cancelar, como na imagem abaixo:

Categoria	Marca	Modelo	Cap.(l)	Valor	Ações
Geladeira	Prosdócimo	SP500	255	R\$ 1.200,00	

Veja em nossa imagem que os dados do formulário são os mesmos do registro que aparece esmaecido ao fundo.

ATENÇÃO: Testou? Não funcionou? Parou e arrumou!

Agora só nos falta exibir a marca correta...

Exibir a marca já cadastrada na edição



Precisamos fazer com que **no campo select apareça desde o começo a marca** a qual esse produto já pertence, mas **também devem nele ser carregadas todas as marcas**, para que o usuário possa trocar se necessário...

Já **temos uma função que carrega as marcas**: a **carregarMarcas**, que usamos anteriormente nessa trilha para exibir as marcas em um campo select do formulário de cadastro. **Ou seja, já temos meio caminho andado!**

Essa função só não está pronta para exibir a marca selecionada... Assim, nossa tarefa no momento será **alterar a função carregarMarcas**, para que ela também dê conta desse recado!

Agora que já tem conhecimento do **que** realizaremos, vamos a **como** isso será feito. Primeiro, antes de configurar a modal de edição, vamos **chamar a função carregarMarcas**, mas dessa vez **passando um parâmetro**: o **id da marca** que desejamos que apareça. Veja no código abaixo o destaque e copie-o para o mesmo local em seu código:

```
var selCategoria = document.getElementById('selCategoriaEdicao');
for(var i=0; i < selCategoria.length; i++){
    if (selCategoria.options[i].value == produto.categoria){
        selCategoria.options[i].setAttribute("selected", "selected");
    }else{
        selCategoria.options[i].removeAttribute("selected");
    }
}

COLDIGO.produto.carregarMarcas(produto.marcaId);

var modalEditaProduto = {
    title: "Edita Produto",
    height: 400,
    width: 550,
    modal: true,
    buttons:{
```

Depois, vamos **alterar a função carregarMarcas** para que ela possa **receber esse id**, e vamos também **verificar** se ao se chamar a função o **id foi ou não recebido**. Veja abaixo:

```
//Carrega as marcas registradas no BD no select do formulário de inserir ou editar
COLDIGO.produto.carregarMarcas = function(id){
    if(id!=undefined){
        select = "#selMarcaEdicao";
    }else{
        select = "#selMarca";
    }
    $.ajax({
        type: "GET",
        url: COLDIGO.PATH + "marca/buscar",
        success: function (marcas) {
```

Repare na **variável select**: caso **seja recebido um id**, ela terá como valor **"#selMarcaEdicao"**, ou seja, a identificação do **campo select de marca do formulário de edição**; caso **não seja recebido um id**, é definido **"#selMarca"** como seu valor, referente ao **campo select de marca do formulário de cadastro**.

Agora, ainda dentro desta função, vamos usar essa variável no lugar de todas as vezes em que, anteriormente, fazíamos uma edição em “#selMarca”, trocando esse identificador pela variável **select** para que seja alterado o campo correto. Veja onde fazer essa alteração nos locais destacados abaixo:

```
success: function (marcas) {  
    if (marcas!="") {  
        $(select).html("");  
        var option = document.createElement("option");  
        option.setAttribute("value", "");  
        option.innerHTML = ("Escolha");  
        $(select).append(option);  
        for (var i = 0; i < marcas.length; i++) {  
            var option = document.createElement("option");  
            option.setAttribute("value", marcas[i].id);  
            option.innerHTML = (marcas[i].nome);  
            $(select).append(option);  
        }  
    }else{  
        $(select).html("");  
        var option = document.createElement("option");  
        option.setAttribute("value", "");  
        option.innerHTML = ("Cadastre uma marca primeiro!");  
        $(select).append(option);  
        $(select).addClass("aviso");  
    }  
},  
error: function (info) {  
    COLDIGO.exibirAviso("Erro ao buscar as marcas: "+ info.stat  
    $(select).html("");  
    var option = document.createElement("option");  
    option.setAttribute("value", "");  
    option.innerHTML = ("Erro ao carregar marcas!");  
    $(select).append(option);  
    $(select).addClass("aviso");  
}
```

Por fim, vamos fazer o código que será responsável por **deixar selecionada a marca** correspondente ao produto que será alterado. Para isso, observe com atenção a imagem abaixo, que destaca o *for* responsável por criar as opções do **select** de marcas, e adicione as linhas entre os destaques:

```
for (var i = 0; i < marcas.length; i++) {  
    var option = document.createElement("option");  
    option.setAttribute("value", marcas[i].id);  
  
    if ((id!=undefined)&&(id==marcas[i].id))  
        option.setAttribute("selected", "selected");  
  
    option.innerHTML = (marcas[i].nome);  
    $(select).append(option);  
}
```

Veja que agora, se o id não estiver indefinido e for igual ao id da marca para a qual estamos criando a opção, definimos para ela o **atributo selected** com o **valor selected**, fazendo assim com que ela apareça selecionada.

Teste final

Mais uma vez, nosso teste final não terá imagens. Para conferir se o funcionamento está OK, você deve fazer com que, **ao ser selecionado um produto para edição, a marca correta apareça por padrão no campo select do formulário de alteração**. Além disso, **isso não deve alterar o funcionamento do formulário de cadastro**, ou seja, nele as opções devem aparecer, mas a opção a ser exibida deve ser a padrão.

Reforçando o conhecimento adquirido

Quanta novidade, né? Porém, é tudo interconectado, então não tem como separar mais se quisermos manter sentido em cada parte do código... Assim, pedimos com muito carinho que execute as seguintes ações:

- **Releia o documento com calma!**
- **Comente o código!**
- Quanto à **alteração da função carregarMarcas do JS**, essa parte ficou bem clara para você? E em relação à **criação da modal com o formulário de edição e o carregamento dos dados nela**, alguma dúvida?
- **Através de um fluxograma ou outra forma que não seja um texto corrido, esquematize o funcionamento do processo criado nesta OT**, desde o momento em que carregamos a página até o momento em que os dados são mostrados na tabela. Represente os arquivos, quando um chama o outro, como o servidor encontra a Rest, enfim, **desenhe o fluxo criado nesta OT. O esquema criado será usado na validação.**

Conclusão

Essa é apenas a primeira parte da edição, onde preparamos o formulário para que o usuário altere os dados. Salvar as alterações realizadas pelo usuário, a outra metade da laranja, é tarefa para a próxima OT. Então, *#partiu!*

Após finalizar a OT, crie um novo commit no Git com o nome da OT e comunique um orientador para novas instruções.