Curso: Análise e Desenvolvimento de Sistemas – ADS

Ano: 2023/1

Orientação Técnica (OT) – 24

Edição REST - Programação

Introdução

Essa OT visa finalizar o CRUD de produtos, realizando a segunda parte da edição, onde os dados contidos no formulário serão enviados ao servidor, que alterará o registro e retornará um aviso ao lado cliente.

Não há muitas novidades nesse momento, então não deverá encontrar dificuldades. Mesmo assim, sua atenção deve ser alta, pois qualquer erro pode se tornar problemático...

Envio dos dados do lado cliente ao servidor

Na OT passada, fizemos no arquivo **produto.js** uma função chamada **COLDIGO.produto.exibirEdicao** contendo, entre outros, a **criação de uma modal** com o **formulário de edição** e dois botões: **Salvar** e **Cancelar**.

Encontre-a em seu projeto, e na seção de código relativa ao **botão Salvar**, adicione a linha destacada na imagem abaixo:

```
var modalEditaProduto = {
    title: "Editar Produto",
    height: 400,
    width: 550,
    modal: true,
    buttons:{
        "Salvar": function(){
            COLDIGO.produto.editar();
        },
        "Cancelar": function(){
            $(this).dialog("close");
        }
    },
    close: function(){
        //caso o usuário simplesmente feche a caixa de edição
        //não deve acontecer nada
    }
};
```

Veja que estamos chamando uma função que ainda não existe, mas já vamos resolver isso! No final do arquivo **produto.js**, crie a nova função conforme imagem abaixo:

```
//Realiza a edição dos dados no BD
COLDIGO.produto.editar = function(){

    var produto = new Object();
    produto.id = document.frmEditaProduto.idProduto.value;
    produto.categoria = document.frmEditaProduto.categoria.value;
    produto.marcaId = document.frmEditaProduto.marcaId.value;
    produto.modelo = document.frmEditaProduto.modelo.value;
    produto.capacidade = document.frmEditaProduto.capacidade.value;
    produto.valor = document.frmEditaProduto.valor.value;

};
```

Veja que, inicialmente, criamos um **objeto produto** e adicionamos nele todos os **dados do formulário**.

Agora continuaremos, criando o código **Ajax** que fará a requisição ao servidor. Para isso, abaixo do código que acabamos de criar, ainda dentro da função (veja na imagem abaixo) insira os códigos novos da imagem a seguir.

```
    produto.capacidade = document.frmEditaProduto.capacidade.value;
    produto.valor = document.frmEditaProduto.valor.value;

    $.ajax({
        type:"PUT",
        url: COLDIGO.PATH + "produto/alterar",
        data:JSON.stringify(produto),
        success: function(msg){

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

        }
    });

};
```

Definimos aqui: **tipo** do envio, **url** destino, **dados** a serem enviados, **função de sucesso** (sem corpo ainda) e **função de erro**. Agora, é hora de fazer o código *backend* para essa url!

Programação backend de alteração

Primeiro, vamos à **ProdutoRest**, criar o método que **receberá as requisições do Ajax que criamos nesta OT**. Faça isso seguindo a imagem, colocando-o como último método desta classe.

```
@PUT
@Path("/alterar")
@Consumes("application/*")
public Response alterar(String produtoParam){

}
```

Até aqui, nada de novo, certo? Também não será surpresa o que pediremos para inserir agora, em destaque na imagem abaixo:

```
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
```

Agora vamos iniciar a codificar o corpo deste método:

```
@PUT
@Path("/alterar")
@Consumes("application/*")
public Response alterar(String produtoParam){
    try{
        Produto produto = new Gson().fromJson(produtoParam, Produto.class);
        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 realizamos um ***try-catch***, e **dentro do *try*, convertemos o Json** recebido do lado cliente em um **objeto da classe Produto**, pois é assim que esses dados serão tratados no Java. Depois, **criamos a conexão** e a repassamos na criação do **objeto jdbcProduto**. Ou seja, nada que não tenhamos feito antes.
Agora, conforme imagem abaixo, vamos chamar o método que vai interagir com o BD para realizar a alteração dos dados:

```
try{
    Produto produto = new Gson().fromJson(produtoParam, Produ
    Conexao conec = new Conexao();
    Connection conexao = conec.abrirConexao();
    JDBCProdutoDAO jdbcProduto = new JDBCProdutoDAO(conexao);

    boolean retorno = jdbcProduto.alterar(produto);

}catch(Exception e){
```

Mais uma vez, antes de criar o método em si, vamos definir na interface **ProdutoDAO** que ele será necessário:

```
public interface ProdutoDAO {

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

}
```

Agora sim, vamos à classe **JDBCProdutoDAO** iniciar a codificação do método que fará as alterações no BD. Assim, crie no final dela o método, conforme exibido abaixo:

```
public boolean alterar(Produto produto) {

    String comando = "UPDATE produtos "
            + "SET categoria=?, modelo=?, capacidade=?, valor=?, marcas_id=?"
            + " WHERE id=?";
    PreparedStatement p;

}
```

Veja que indicamos que ele **recebe um objeto de Produto**, criamos em uma variável o **comando SQL UPDATE** para alteração e também um **objeto de PreparedStatement**.

IMPORTANTE: você se lembra como o id do produto chega ao lado servidor, já que não damos ao usuário a possibilidade de alterá-lo no formulário?

A seguir, continue o método, inserindo abaixo do código anterior **o *try-catch*** e seu código interno, **e o *return true*,** conforme imagem a seguir:

```
    PreparedStatement p;
    try {
        p = this.conexao.prepareStatement(comando);
        p.setString(1, produto.getCategoria());
        p.setString(2, produto.getModelo());
        p.setInt(3, produto.getCapacidade());
        p.setFloat(4, produto.getValor());
        p.setInt(5, produto.getMarcaId());
        p.setInt(6, produto.getId());
        p.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
    return true;

}
```

No ***try***, usamos a conexão para **preparar o comando SQL** criado, depois **setamos os 6 valores** variantes desse comando com **os valores do objeto produto**, e por fim **executamos o UPDATE**.

No ***catch***, o de sempre: **mostramos o erro e quem o gerou**, e **retornamos false**. **Abaixo do try-catch, retornamos verdadeiro**, pois se chegar nessa parte do código é porque a alteração deu certo.
Terminado este método, vamos retornar à classe ProdutoRest e finalizar o método de edição:

```
    boolean retorno = jdbcProduto.alterar(produto);

    String msg = "";
    if (retorno){
        msg = "Produto alterado com sucesso!";
    }else{
        msg = "Erro ao alterar produto.";
    }

    conec.fecharConexao();
    return this.buildResponse(msg);
}catch(Exception e){
```

Mais uma vez, apenas códigos cuja lógica e funcionamento já viu anteriormente, então abstemo-nos de explicações.

Recebendo o resultado no frontend

Agora que o lado backend foi programado por completo, voltemos ao lado frontend, para indicar o que deve acontecer se a chamada ao servidor for realizada com sucesso. Para isso, copie então o código entre as partes em destaque no código abaixo:

```
$.ajax({
    type:"PUT",
    url: COLDIGO.PATH + "produto/alterar",
    data:JSON.stringify(produto),
    success: function(msg){
        COLDIGO.exibirAviso(msg);
        COLDIGO.produto.buscar();
        $("#modalEditaProduto").dialog("close");
    },
    error: function(info){
        COLDIGO.exibirAviso("Erro ao editar produ

    }
});
```

Veja que o código é bem simples:

- **Exibimos** em uma modal a **mensagem de aviso** vinda do servidor;
- **Recarregamos** a **lista de produtos**;
- **Fechamos a modal com o formulário** de edição de produto.

Com isso, finalizamos a programação da edição!!!

Testando o pacote completo!

Bem, se já acabou a edição, já acabou o CRUD!

Agora é só testar. Porém, justamente como temos o CRUD completo, recomendamos um teste mais completo. Afinal, e se alguma programação realizada mais recentemente afetou o desempenho do que estava funcionando antes? Assim, sugerimos como teste:

- Primeiro excluir todos os produtos que possui para “recomeçar”;
- Fazer as ações do CRUD de maneira mais aleatória, ou seja, não criar vários, depois alterar vários, e então excluir vários... Mescle as ações.

Se testar tudo direitinho, de modos diferentes, verá...

O problema que você ~~(nem)~~ sabia que existia...

Mas o que? Você não achou nenhum erro? Vish...

Faça o seguinte então:

- Recarregue a página no navegador;
- Acesse a página de produtos;
- Edite um produto com sucesso;
- Vá até o menu, e clique no link para a página de produtos (sim, a mesma em que você já está...);
- Clique no botão de editar de um produto qualquer...

Cadê a modal, John?

Repare na console do navegador:

```
Uncaught TypeError: Cannot set property 'value' of undefined      VM134:176
    at Object.success (<anonymous>:176:46)
    at u (jquery-3.3.1.min.js:2)
    at Object.fireWith [as resolveWith] (jquery-3.3.1.min.js:2)
    at k (jquery-3.3.1.min.js:2)
    at XMLHttpRequest.<anonymous> (jquery-3.3.1.min.js:2)
```

Não, você não virou o Ciclope dos X-men. Está tudo vermelho mesmo!
Vamos entender o problema: **recarregue a página**, e **abra a página de produtos**. Abra a parte de **inspeção**, deixando na **aba Elements**. Veja que a modal de edição foi criada dentro da section:

```
▼<section> == $0
    <script src="../../js/product.js"></script>
    <h2>Registro de produtos</h2>
  ▶<form name="frmAddProduto" id="addProduto" class="frmInserir">…</form>
    <h3>Produtos registrados</h3>
  ▶<form id="filtraProduto" class="frmFiltrar">…</form>
  ▶<div id="listaProdutos" class="listaRegistros">…</div>
  ▶<div id="modalEditaProduto" class="modalEditar">…</div>
  </section>
```

Posicione-se **abaixo do footer**. **Abra a modal de edição**. Ela vai criar um novo código **entre o </footer> e o </body>**:

```
    </footer>
  ▶<div tabindex="-1" role="dialog" class="ui-dialog ui-corner-all ui-widget
  ui-widget-content ui-front ui-dialog-buttons ui-draggable ui-resizable" aria-
  describedby="modalEditaProduto" aria-labelledby="ui-id-1" style="position:
  absolute; height: auto; width: 550px; top: 141.516px; left: 395.493px; z-
  index: 101; display: none;">…</div>
  ▶<div tabindex="-1" role="dialog" class="ui-dialog ui-corner-all ui-widget
  ui-widget-content ui-front ui-dialog-buttons ui-draggable ui-resizable" aria-
  describedby="modalAviso" aria-labelledby="ui-id-2" style="position: absolute;
  height: auto; width: 400px; top: 185.504px; left: 470.48px; z-index: 102;
  display: none;">…</div>
  </body>
</html>
```

Nesta imagem mostramos **2 divs**, o que você também verá caso conclua a edição, que são **as modais de edição e de aviso**, respectivamente (veja pelo atributo **"aria-describedby"**). Repare que estão **abaixo do footer**, certo? **Procure** agora a **modalEditaProduto dentro da section**: ela foi **movida**! Isso é feito automaticamente **pelo jQueryUI** ao exibir as divs como modais.

Agora abra no **Eclipse** o arquivo **admin.js,** e veja que em **COLDIGO.carregaPagina**, a função que exibe a página correta conforme nossa escolha no menu, começamos **esvaziando a tag section**, certo? **Execute essa função**, clicando novamente no menu, no link de produtos.

Como essas divs **sairam da section** graças ao jQueryUI, elas **não foram apagadas.** E como solicitamos o **carregamento da página de produtos**, a **modalEditaProduto foi criada novamente dentro da section**, o que resulta em **2 formulários iguais**!

Se essa duplicidade confunde até o sentido aranha do Spidey, imagina o pobre JS! **Ele não sabe em qual dos dois formulários carregar os dados para a edição...**

Mas como é um rapaz inteligente, Pet... Quer dizer, o Homem Aranha já descobriu como resolver isso: **é só apagar sua cópia**! Para isso, na função **COLDIGO.carregaPagina** da página **admin.js**, adicione o código em destaque na imagem abaixo:

```
//Função para carregamento de páginas de conteúdo, que
//recebe como parâmetro o nome da pasta com a página a ser carregada
COLDIGO.carregaPagina = function(pagename){
    //Remove o conteúdo criado na abertura de uma janela modal pelo jQueryUI
    if ($(".ui-dialog"))
        $(".ui-dialog").remove();
    //Limpa a tag section, excluindo todo o conteúdo de dentro dela
    $("section").empty();
```

Agora sim, **ao carregar uma página, excluímos as divs criadas pelo jQueryUI** (caso elas existam) e não teremos mais esse problema! Teste tudo novamente, e agora não deve encontrar mais problemas...

Calma, Spidey, folga ainda não....

ATENÇÃO: ainda tem algo não funcionando? Vish... Tente resolver, mas se precisar, é só chamar o amigão da vizinhança, ou um orientador.

Reforçando o conhecimento adquirido

Como já fazemos há algumas OTs, pedimos com muito carinho que execute as seguintes ações para reforçar o que aprendeu:

- **Releia o documento com calma!**
- **Comente o código!**
- **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.**
- Depois desta OT, você desenvolveu por completo um processo que: abre uma modal, e a partir dela, abre uma segunda modal... Aplique essa mesma prática para **aprimorar a exclusão**, fazendo com que, ao clicar no botão de exclusão, **apareça primeiro uma modal com uma frase "Deseja realmente excluir este produto?" e dois botões (OK e Cancelar)**, e apenas se o usuário confirmar a exclusão, ela acontece e o usuário visualiza a modal com a mensagem da notificação de produto excluído.

Conclusão

Aê! Finalizamos o CRUD de produtos!!! Pode comemorar!

Mas claro, o que vai te dar maior direito de comemoração é sua compreensão sobre o que fizemos nestas OTs. Criar um CRUD é básico para um desenvolvedor, assim absorver esse conhecimento é o que mais importa. Até porque você acabou agora apenas o primeiro de muitos que ainda fará conosco. Falando nisso, vamos à próxima OT?

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