Curso: Análise e Desenvolvimento de Sistemas – ADS

Ano: 2023/1

Orientação Técnica (OT) – 22

Exclusão REST

Introdução

A exclusão é um processo relativamente simples, onde selecionaremos um registro e ele deverá ser excluído do BD. Repare que, visualmente, a única coisa necessária é o botão de excluir, que já possuímos, então já poderemos ir direto para a programação da mesma! Vamos lá?

Função JS de exclusão

Neste momento, vamos criar a função JS que enviará a informação do produto a ser excluído ao BD. Mas como já dito, essa função deve ser acionada através do **botão de exclusão** ao lado de cada registro na tabela de produtos acessível pelo usuário. Assim, nossa primeira ação será fazer essa codificação no local onde criamos o botão.

Veja que, apesar de ser um código HTML, esse botão não é criado em **products/index.html**, e sim no local onde criamos a tabela em si: no arquivo **product.js**, no método **COLDIGO.produto.exibir**, dentro do ***for*** responsável por criar cada linha da tabela. Assim, encontre esse código e **adicione a ele a parte destacada** da imagem abaixo:

```
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><img src='../../imgs/edit.png' alt='Editar registro'></a> " +
                "<a onclick=\"COLDIGO.produto.excluir('"+listaDeProdutos[i].id+"')\"><img src='../../imgs/delete.png'
            "</td>" +
            "</tr>"
}
```

Agora sim, ao clicarmos no botão de exclusão a função será chamada. ***Mas ela nem existe ainda!*** Para resolver isso, vá até o final deste mesmo arquivo e, entre os locais destacados abaixo, insira o novo código:

```
    //Executa a função de busca ao carregar a página
    COLDIGO.produto.buscar();

    //Exclui o produto selecionado
    COLDIGO.produto.excluir = function(id){
        $.ajax({
            type:"DELETE",
            url: COLDIGO.PATH + "produto/excluir/"+id,
            success: function(msg){
                COLDIGO.exibirAviso(msg);
                COLDIGO.produto.buscar();
            },
            error: function(info){
                COLDIGO.exibirAviso("Erro ao excluir produto: "+ info.status + " - " + info.statusText);
            }
        });
    };

});
```

Já colocamos nesta imagem o **código COMPLETO desta função** (repare dentro do ***success*** que ele já chama a função de exibição de aviso, para dizer se houve erro ou sucesso, e logo depois refaz a busca dos produtos para que o produto excluído não apareça mais).

Além disso, repare que temos uma **novidade na URL do Ajax:** o **id do produto** a ser excluído **está sendo enviado através dela!** Exemplificando, se iremos excluir o **produto de id 23**, acessaremos a URL **"produto/excluir/23"**. Guarde isso em mente, isso causará diferenças no backend!

Além dessa, acreditamos não ter outra parte desse código que precisemos explicar, *então nos avise na validação caso estejamos enganados*.

Programação backend para exclusão

Como sempre, vamos buscar uma ordem que seja o mais próximo possível dos acontecimentos reais, então para começar abra a classe **ProdutoRest** e, *após* os métodos já existentes, adicione o código da imagem a seguir:

```
@DELETE
@Path("/excluir/{id}")
@Consumes("application/*")
public Response excluir(@PathParam("id") int id){

}
```

Explicando: além da anotação **@DELETE**, que é novidade, mas você é quem irá nos explicar na validação, **outras duas partes são novas**: no **@Path**, guardamos o que está depois de "/excluir/" em um **parâmetro de caminho chamado id** (por isso **colocamos a palavra id entre chaves...**); e na **declaração do método**, indicamos que **esse parâmetro de caminho deve ser guardado** em uma **variável do tipo int chamada id** para que possa ser usado dentro do método. Assim, em 2 passos, podemos usar a parte da URL onde enviamos o id do usuário dentro

do método.
Sobre os dois erros que apareceram, acreditamos que já entende como resolvê-los, certo? Isso mesmo, são só os ***imports*** que ainda não fizemos... Veja como e onde fazê-los abaixo:

```
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
```

Erros corrigidos e explicações feitas, vamos em frente... Começaremos o corpo do método excluir da ProdutoRest, então insira o código entre os destaques:

```
@DELETE
@Path("/excluir/{id}")
@Consumes("application/*")
public Response excluir(@PathParam("id") int id){

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

}
```

Sem novidades, né? Realizamos o **try-catch** para tratarmos as possíveis exceções e, dentro do try, já fazemos as **ações relativas à banco de dados**: **nos conectamos a ele** e criamos um **objeto de JDBCProdutoDAO** que recebe o objeto com a conexão aberta.
Seguindo, vamos **chamar o método responsável pela exclusão do produto do BD**, conforme destacado na imagem abaixo:

```
public Response excluir(@PathParam("id") int id){

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

        boolean retorno = jdbcProduto.deletar(id);

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

}
```

Agora, para criarmos esse método que acabamos de chamar, primeiro vamos até a **interface ProdutoDAO**, indicar que esse método deve ser criado por quem a implementar:

```
public interface ProdutoDAO {

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

}
```

Como, no nosso caso, quem implementa esta interface é a classe **JDBCProdutoDAO**, vamos a ela. Logo abaixo dos métodos já existentes, crie o código a seguir:

```
public boolean deletar(int id) {
    String comando = "DELETE FROM produtos WHERE id = ?";
    PreparedStatement p;
    try {
        p = this.conexao.prepareStatement(comando);
        p.setInt(1, id);
        p.execute();
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}
```

Veja que o código é bem simples:

- Recebemos o id como **parâmetro**;
- Começamos criando o **comando SQL** de exclusão deixando o id em aberto com a "?";
- Criamos o **PreparedStatement p**;
- **Preparamos o comando com a conexão**;
- **Trocamos** no comando **a "?" pelo id** do produto a ser excluído;
- **Executamos** o comando;
- **Retornamos verdadeiro** caso não haja nenhuma exceção.

ATENÇÃO: porque usamos o PreparedStatement ao invés do Statement? Faz alguma diferença?

Feita a interação com o BD, só nos falta retornar uma mensagem ao lado cliente, informando o status da operação de exclusão. Para isso, volte ao método excluir da **ProdutoRest** para terminarmos sua codificação, inserindo nele o código entre as partes destacadas:

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

    boolean retorno = jdbcProduto.deletar(id);

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

    conec.fecharConexao();

    return this.buildResponse(msg);

}catch(Exception e){
```

Repare que primeiro **criamos uma variável para receber uma mensagem**, depois, **de acordo com o retorno** do método deletar da JDBCProdutoDAO, **guardamos nela a mensagem** respectiva, então **fechamos a conexão** e **construímos a resposta** para retornar ao lado cliente.

Testes

Desta vez, não vamos mostrar imagens, nem trabalhar os exemplos. Acreditamos que você seja capaz de perceber o que deve aparecer ou não, e que saiba como resolver os erros presentes. Todavia, em caso de necessidade, *não hesite em nos chamar*!

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!**
- **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

Devido à ordem em que fizemos as atividades, essa etapa é a mais simples, mesmo tendo suas novidades. Porém, agora, vamos para a etapa que individualmente é a mais trabalhosa e a que temos mais chances de erro: a alteração! Assim, não abaixe a guarda, estamos quase acabando o nosso CRUD! Até a próxima OT!

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