



CRUD Stripes / MyBatis / Spring

Delivoro Sistemas



Passos - 1 - Armazenamento de Contatos

- Criar Entidade - para representar um contato
 - Anotar com `@Data`, `@ToString` do Lombok
 - Atributos: nome, telefone, genero (M/F), ativo, estado civil, observações
- Criar Data Access Object ou Mapper - para armazenar dados de contatos
 - Num primeiro momento armazenaremos dados na memória
 - Depois integraremos com o MyBatis para armazenar em uma tabela no MySQL
 - Criar a interface ContatoMapper em `training.mapper1`
 - Usar 1 no package name para evitar ser detectada pelo myBatis
 - copiar de `ExemploMapper` sem o `@Param`
 - 5 operações básicas: insert, update, delete, find, list



Passo - 1 - Armazenamento de Contatos

- Criar Data Access Object ou Mapper - para armazenar dados de contatos
 - Criar classe `training.mapper.impl.ContatoMapperImpl` que implemente `ContatoMapper`
 - Use um `HashMap` como backend de armazenamento de dados
 - Anotar com `@Component` para ser registrada no Spring
 - Disponível na pasta `códigos do google Drive`
- Criar teste unitário para `ContatoMapperImpl`
 - Início disponível na pasta `códigos do google Drive`



Passo - 2 - Interface Web

- Criar uma JSP chamada form.jsp em WEB-INF/jsp/contato
 - Obter ponto de partida na pasta códigos do Google Drive
 - Acrescentar um título ao mesmo usando a tag H1 - Inserir Contato
 - A JSP deverá conter um formulário HTML
 - Veja exemplo no Google Drive (form.html)
 - Usar tags fieldset, legend, label
 - Criar os campos a serem preenchidos pelo usuário.
 - O nome de cada campo deverá começar com "contato."
 - depois do ponto vem o nome do atributo - igual da entidade Contato
 - O formulário deverá possuir um botão submit
 - O nome do botão deverá ser "inserir"

Obs: Certificar que todos os campos do contato estejam na JSP



Passo - 3 - Interface Web - Inserção

- Criar ContatoActionBean em training.actionbeans
 - extends BaseActionBean
 - @UrlBinding("/contato.action")
 - Criar atributo Contato contato;
 - Criar atributo ContatoMapper contatoMapper e anotar com @SpringBean
 - Criar um método chamado exibir
 - Anotá-lo com @DefaultHandler - mudaremos isso depois
 - Responsável por encaminhar a request para a JSP
 - /WEB-INF/jsp/cadastro/form.jsp
 - Criar um método chamado "inserir" que insira o contato em ContatoMapper
 - Retornar um RedirectResolution para o próprio caso de uso
 - Usar breakpoints para verificar valores
- Testar tudo o que foi feito até funcionar basicamente



Passo - 4 - Interface Web - Tags do Stripes

- Mensagem de sucesso
 - Adicionar uma mensagem de sucesso usando o `super.addMessage` e
 - Exibir mensagem de sucesso na `form.jsp` usando o `<s:messages/>`
- Exibir quantidade de elementos inseridos na JSP
 - Criar atributo `int` `quantidade` na `ContatoActionBean`
 - Anotar com `@Getter`
 - Exibir esse atributo da `Action` na JSP



Passo - 5 - Interface Web - Tags do Stripes

- Implementar Validation usando annotations do Stripes
 - Todos os campos são obrigatórios
 - menos o campo ativo
 - Definir tamanhos mínimos e máximos para fins de exercício
 - Usar `<s:errors field="" />` nos campos
 - Usar `<s:errors globalErrorsOnly="true" />` para o formulário



Passo - 6 - Integração com Banco de Dados

- Mover a interface ContatoMapper para a package training.mapper
- Anotar os argumentos dos métodos remove e buscaPorId com o @Param
- Criar uma tabela chamada contato no banco de dados treinamentoX em 192.168.1.90 usando o DBeaver
 - X é o último dígito do seu user ID no Gitlab
 - Colunas: id_contato int, nome (100) not null, telefone (25) not null, genero (1), ativo (boolean), estado_civil (10), observacoes (text)
- Criar o arquivo XML ContatoMapper.xml a partir do ExampleMapper.xml
 - Salvar em src/main/resources
 - Fazer as modificações necessárias
- Remover o @Component da classe ContatoMapperImpl
- Testar a aplicação

Passo - 7 - Fazer a inserção funcionar com AJAX

- Exemplo

The screenshot displays the Delivoro dashboard interface. At the top, there is a navigation bar with the Delivoro logo and several action buttons: DELIVERY (F1), MESAS (F2), RETAGUARDA (F4), MINHA CONTA, and LOGOUT (F10). Below this is a secondary navigation bar with links to various sections: Ver seu site, FINANCEIRO, PRODUTOS, ESTOQUE, RELATÓRIOS, DIVULGAÇÃO, CLIENTES, and CONFIGURAÇÕES. The CONFIGURAÇÕES menu is open, showing a list of options: ÁREA E TAXAS DE ENTREGA, OPÇÕES, FINANCEIRO, OPERADORES, AUTH API, and COMANDA/MESA. The OPÇÕES sub-menu is also open, displaying FORMAS DE PAGAMENTO, ADQUIRÊNCIA, CAIXAS, and another option partially visible. The CAIXAS option is highlighted. In the background, the 'CADASTRO DE CAIXAS' form is visible, featuring a text input field for 'Nome do Caixa' and two buttons: 'Novo Caixa' and 'Configurar atribuições aos caixas'. Below the form, a table header is partially visible with columns: Nome, Tipo, Data Cadastro, and Ativo.

Passo - 7 - Fazer a inserção funcionar com AJAX

- Exemplo

Novo Caixa

Nome do Caixa [?] ☒ Ativo Data de Cadastro

Tipo de Caixa
 Normal

Caixa tipo Normal - Este tipo de caixa se refere aos caixas do operador, não possui nenhuma característica específica.

Tipo de fechamento
 ☒ Fechamento cego [?]

Configurações opcionais

Valor do fundo de caixa [?] Margem do fundo de caixa [?] Valor limite do caixa [?]

☒ Utilizar sugestão de transporte no valor do caixa [?] ☐ Obrigar transporte no valor do caixa [?]

☐ Vincular usuário(s) específico(s) [?]

Configurações de impressão

☐ Imprime fechamento do caixa
 ☐ Imprime produtos vendidos
 ☐ Imprime resumo de categorias

Salvar



Passo - 7 - Fazer a inserção funcionar com AJAX

Preparar bibliotecas Javascript para AJAX

- Criar pasta js em src/main/webapp e copiar os arquivos abaixo para ela
 - Copiar o arquivo lojaWeb/src/main/webapp/js/gmxbbox.js do Delivoro
 - Copiar o arquivo lojaWeb/src/main/webapp/node/jquery/dist/jquery.min.js do Delivoro
 - Copiar o arquivo lojaWeb/src/main/webapp/node/sweetalert/dist/sweetalert.min.js do Delivoro
- Criar pasta css em src/main/webapp e copiar os arquivos abaixo para ela
 - Copiar arquivo lojaWeb/src/main/webapp/node/sweetalert/dist/sweetalert.css do Delivoro
 - Copiar arquivo lojaWeb/src/main/webapp/css/gmxbbox.css do Delivoro



Passo - 7 - Fazer a inserção funcionar com AJAX

Preparar bibliotecas Javascript para AJAX

- Carregar os arquivos js e css na tela de listagem
- Adicione as linhas abaixo no arquivo layout.jsp entre as tags <head> e </head> do HTML

```
<script src="js/jquery.min.js"></script>
```

```
<script src="js/gmxbbox.js"></script>
```

```
<script src="js/sweetalert.js"></script>
```

```
<link rel="stylesheet" href="css/sweetalert.css">
```

- Certificar que não tem nenhum erro de javascript no console do Google Chrome



Passo - 7 - Fazer a inserção funcionar com AJAX

Fazer formulário de edição aparecer em uma div parecida com um popup

- Duplicar a JSP de inserir contato, salvando em `inserirAjax.jsp` e remover o código relacionado com layout
- Criar o método `preparaInserirAjax` na `ContatoActionBean` com o mesmo código que `preparaInserir` só que fazendo o return da resolution com a nova JSP.
 - Ex: `new ForwardResolution("inserirAjax.jsp")`
- Criar um link para abrir o `preparaInserirAjax` em uma `gmxBBox`
- Ex: `<s:link beanclass="ContatoActionBean" event="preparaInserirAjax" class="gmxbbox">`
- Testar e ver se ao clicar nesse link se abre o popup de inserção em uma div flutuante no meio da tela



Passo - 8 - Atualizar a table de contatos via AJAX

Fazer botão que envie dados sem dar refresh na tela.

Vai precisar de código Javascript para enviar dados para o server e obter o que o server retornar e colocar na div de listagem de dados

- Colocar a display table dentro de uma div HTML com `id="listagem"`
 - O conteúdo dessa div será substituído pela função Javascript assim que a resposta chegar do server.
- Extrair o código com a `display:table` para seu próprio arquivo JSP
 - `list-table.jsp`
 - Não vai precisar de layout nessa JSP
 - Apenas faz o `include` das taglibs no topo
 - Fazer o `<jsp:include page="list-table.jsp" />` dentro da div onde você tirou a display table
- Criar o método `inserirAjax` na `ContatoActionBean` com o mesmo código do método `inserir` existente
 - Só que mudar o return para `ForwardResolution` ao invés de `RedirectResolution`
 - O `ForwardResolution` deve ser para a `list-table.jsp`



Passo - 8 - Atualizar a table de contatos via AJAX

Criar função Javascript que será chamada ao clicar em Salvar no popup de inserção

- Fazer alguma mudança na `inserirAjax.jsp` para que o botão Salvar não faça o submit do formulário
 - Transformar o `s:submit` em um `s:button`
- Adicionar o código abaixo no `s:submit`
 - Ele deve parar de salvar e dar um erro no console do Google Chrome

```
onclick="salvarContato(); return false;"
```



Passo - 8 - Atualizar a table de contatos via AJAX

Criar função Javascript que será chamada ao clicar em Salvar no popup de inserção

- Criar uma variável na JSP no topo da `list.jsp` com a URL que a função `salvarContato` deverá chamar
 - ex: `<s:url beanclass="ContatoActionBean" event="inserirAjax" var="inserirAjaxURL" />`
- Adicionar um id ao `s:form` `id="formContato"`
- Escrever a função `salvarContato()` na `list.jsp`

```
<script>
```

```
function salvarContato() {  
}
```

```
</script>
```


Passo - 8 - Atualizar a table de contatos via AJAX

```
<script>
function salvarContato() {
    params = {
        "contato.nome": jQuery('#formContato input["name=contato.nome"]').val(),
        "contato.telefone": jQuery('#formContato
input[name="contato.telefone"]').val(),
    }; // Adicionar outros campos que faltam acima
    jQuery.ajax({
        contentType: "application/x-www-form-urlencoded; charset=UTF-8",
        url: ${inserirAjaxURL},
        type: "POST",
        datatype: "HTML",
        data: params,
        success: function(data) {
            jQuery("#listagem").html(data);
        },
        error: function(xhr, ajaxOptions, thrownError) {
            alert("Um erro ocorreu ao salvar o contato no servidor");
        }
    });
}
</script>
```



Passo - 9 - Fazer Edição e Remoção por AJAX

A edição pode ser feita de forma bem parecida com a inserção.

Para fazer a remoção

- Modificar método que faz a remoção da ContatoActionBean para fazer da mesma forma que o inserirAjax (carregar dados e Forward para list-table.jsp)
- Criar um s:url para a remocao como foi feito para inserção e colocar na list.jsp
- Trocar o s:link por ``
 - `return false` vai impedir que o navegador tente ir para o endereço #
 - Criar a função `removerContato` a partir da `salvarContato`.
 - O id do contato a ser removido deverá vir como argumento
 - Enviar apenas chave primária do contato através do params

Passo - 8 - Atualizar a table de contatos via AJAX

```
<script>
function removerContato(int id) { // note que essa função recebe um argumento
    params = {
        "contato.id": // preencher essa parte
    };
    jQuery.ajax({
        contentType: "application/x-www-form-urlencoded; charset=UTF-8",
        url: // preencher essa parte,
        type: "POST",
        datatype: "HTML",
        data: params,
        success: function(data) {
            jQuery("#listagem").html(data);
        },
        error: function(xhr, ajaxOptions, thrownError) {
            alert("Um erro ocorreu ao remover o contato no servidor");
        }
    });
}
</script>
```



Passo - 10 - Adicionar um campo

Adicionar um campo a mais no contato: cidade

- Criar uma tabela cidade no banco de dados com as colunas
 - id_cidade - int - Definir como chave primária
 - cidade - varchar(50)
 - uf - varchar(2)
 - Cadastrar algumas cidades nessa tabela usando o DBeaver
- Criar uma entidade no código Java para representar uma cidade (ex: Contato)
- Criar CidadeMapper.java e CidadeMapper.xml com apenas uma operação para listar as cidades presentes no banco de dados (ex: ContatoMapper.java e .xml)
- Adicionar o CidadeMapper no mybatis.xml
- Injetar o CidadeMapper na ContatoActionBean
 - como foi feito para o ContatoMapper



Passo - 10 - Adicionar um campo

- Criar um atributo `List<Cidade> cidades` na `ContatoActionBean`
 - Anotar ele com `@Getter` para a JSP poder consultar
- Nos métodos de preparação para editar e inserir
 - Consultar as cidades via `CidadeMapper` e salvar em `cidades`
 - ex: `cidades = cidadeMapper.lista();`
- Adicionar o campo `contato.cidade` nas JSPs de inserção e remoção
 - O campo `cidade` deverá ser um campo de `s:select` do Stripes
 - [Exemplo no Delivoro](#)
- Adicionar o campo `contato.cidade` nas funções Javascript de AJAX
- Testar se funciona



Passo - 11 - Melhorias

Melhorias de código na ActionBean

- Remover métodos que não são mais necessários devido a migração para AJAX
- Criar métodos privados que possam ser chamados para evitar duplicidade de código.

Unificar a inserção e a edição para usarem a mesma JSP e o mesmo método que salva os dados no banco

- Para o seu código saber se deve inserir ou atualizar basta verificar se já a chave primária está presente nos dados do contato