Exercício 7 - JSTL/EL

Este exercício deve ser feito como **alteração do Exercício 6** feito anteriormente. As alterações são:

- Proibido o uso de Scriptles. Todas as páginas JSP devem ter somente código HTML, CSS, JS e JSTL/EL
- Não é permitida chamada de métodos do Façade ou DAO nas JSPs
- Todos os campos dos formulários de entrada de dados devem ter:
    - Tamanhos máximos conforme o dado e tabela
    - Máscara
    - Validação de obrigatório ou não
- Todos os campos de Data devem ser entrados através de calendários (por exemplo: jQuery Datepicker). É TAREFA DOS ALUNOS PESQUISAR COMO FAZER ESTA ALTERAÇÃO.
- Todos os campos monetários e datas devem ser entrados e mostrados em formato brasileiro
- Deve-se normalizar os campos cidade e estado do cliente. Isto é, criar as seguintes tabelas:
    - **tb_cidade** contendo os campos **id_cidade** (PK, inteiro), **nome_cidade** (VARCHAR(100)), **id_estado** (FK para tabela de estado). Carregar pelo menos 10 cidades de cada estado.
    - **tb_estado** contendo os campos **id_estado** (PK, inteiro), **nome_estado** (VARCHAR(100)), **sigla_estado** (CHAR(2). Carregar com todos os estados do Brasil.
- Deve-se alterar a tabela de cliente, conforme apresentado abaixo
- As telas **clientesNovo.jsp** e **clientesAlterar.jsp** deixarão de existir e suas funcionalidades estarão em uma nova tela, chamada **clientesForm.jsp**, que é usada tanto na edição como na alteração do cliente
- Deve-se incluir as seguintes regras de negócio:
    - login_usuario, email_cliente e cpf_cliente são únicos. Isto é, não se deve permitir a inclusão de elementos repetidos
    - CPF deve ser válido
- As senhas dos usuários devem ser criptografadas no banco de dados (Pode ser feito, de forma simples, usando MessageDigest. PESQUISAR)
- Demais alterações realçadas ao longo da especificação.

Agora todas as classes Java deverão estar dentro dos pacotes conforme indicado nas descrições abaixo.

Deve-se usar o banco de dados já criado contendo uma tabela chamada **tb_usuario** contendo os campos **id_usuario** (PK, inteiro), **login_usuario** (VARCHAR(50)), **senha_usuario** (VARCHAR(50) e **nome_usuario** (VARCHAR(100)).

Deve-se ter pelo menos 3 usuários nessa tabela. Esta tabela será usada para efetuar login no sistema.

Deve-se adicionar uma tabela chamada **tb_cliente** contendo os campos **id_cliente** (PK, inteiro), **cpf_cliente** (CHAR(11)), **nome_cliente** (VARCHAR(100)), **email_cliente** (VARCHAR(100), **data_cliente** (DATE), **rua_cliente** (VARCHAR(100)), **nr_cliente** (INTEGER), **cep_cliente** (CHAR(8)), **id_cidade** (FK para tabela cidade).

Adicionar pelo menos 10 clientes nessa tabela.

Todas as telas JSPs e HTMLs devem usar Bootstrap. Link para básico do BOOTSTRAP: https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp

1) index.jsp

- Deve-se incluir BOOTSTRAP para formatação desta página
- Deve receber um parâmetro no escopo da requisição (chamado "msg"). Se este parâmetro for passado, deve ser apresentado em vermelho, como uma mensagem de aviso, junto com a página de login. Se não for passado, somente a página de login é apresentada.

No rodapé da página deve ser colocada a seguinte mensagem: **"Em caso de problemas contactar o administrador: "** juntamente com o e-mail armazenado no escopo da aplicação.

2) com.ufpr.tads.web2.servlets.StartupServlet

É uma servlet que deve ser inicializada no Startup da aplicação. Dentro do seu método **init()**, deve-se armazenar no escopo da aplicação (ServletContext) uma instância preenchida de um ConfigBean, com o nome "configuracao".

3) com.ufpr.tads.web2.servlets.LoginServlet

É uma servlet que verifica o login do usuário.
Se o par login/senha estiver no banco de dados:

- Preenche uma instância de LoginBean com os dados do usuário
- Armazena o LoginBean na sessão (indicando que o usuário está logado);
- Redireciona para a servlet **portal.jsp**.

Se o par login/senha não estiver no banco de dados:

- Redireciona para **index.jsp** (via *forward*), passando como parâmetro a mensagem de erro a ser apresentada (parâmetro "msg") "**Usuário/Senha inválidos."**

4) portal.jsp

A primeira coisa que **portal.jsp** faz é verificar se o usuário está logado.

- Verifica através de JSTL/EL
- O JSP já possui um objeto **session** instanciado, basta acessá-lo
- Se não estiver logado:
    - Redireciona para **index.jsp** (via *forward*), passando como parâmetro a mensagem de erro a ser apresentada (parâmetro "msg") "**Usuário deve se autenticar para acessar o sistema."**
- Se o usuário estiver logado:
    - Apresenta uma tela com o nome do usuário logado usando EL
    - Deve-se incluir um MENU contendo as opções:
        - **Cadastro de Clientes**: que direciona para a servlet **ClientesServlet**
        - **Sair**: que direciona para a servlet **LogoutServlet**

No rodapé da página deve ser colocada a seguinte mensagem: **"Em caso de problemas contactar o administrador: "** juntamente com o e-mail armazenado no escopo da aplicação.

5) com.ufpr.tads.web2.servlets.LogoutServlet

Efetua o logout do usuário e direciona, via **forward** para **index.jsp**, passando como parâmetro a mensagem **"Usuário desconectado com sucesso"**

6) erro.jsp

Esta agora será uma página de erro, portanto adicione a diretiva que faz esta indicação.

Apresenta, em vermelho, a mensagem de erro de dentro da exceção que gerou a sua chamada (objeto implícito **exception**). Devem ser mostrado a mensagem e logo após o StackTrace:

Mostrar Mensagem:

${pageContext.exception.message}

Mostrar StackTrace:

${pageContext.out.flush()}
${pageContext.exception.printStackTrace(pageContext.response.writer)}

No rodapé da página deve ser colocada a seguinte mensagem: "Em caso de problemas contactar o administrador: " juntamente com o e-mail armazenado no escopo da aplicação.

7) com.ufpr.tads.web2.servlets.ClientesServlet

Inicialmente verifica se o usuário está logado. Se não estiver, redireciona via **forward** para **index.jsp** passando como parâmetro a mensagem **"Usuário deve se autenticar para acessar o sistema"** (parâmetro "msg").

Esta servlet processa TODAS as requisições relativas a clientes. Ela deve SEMPRE receber um parâmetro chamado **action**. A seguir os valores possíveis e ações a serem tomadas

1. Valor: "**list**" ou nulo (não passado)

Invoca a Façade para buscar todos os clientes da base de dados, como um **List<Cliente>**, e os coloca no escopo da requisição. Em seguida, efetua **forward** para **clientesListar.jsp**.

2. Valor: "**show**"

    - Recebe um parâmetro (via parameter, GET) com o Id do cliente a ser mostrado
    - Invoca a Façade para buscar no banco de dados o objeto do cliente
    - Adiciona o objeto no escopo da requisição
    - Efetua forward para **clientesVisualizar.jsp**

3. Valor: "**formUpdate**"

    - Recebe um parâmetro (via parameter, GET) com o Id do cliente a ser alterado
    - Invoca a Façade para busca no banco de dados o objeto do cliente
    - Carrega a lista de estados, para apresentar na Combo
    - Adiciona o objeto no escopo da requisição
    - Efetua forward para **clientesForm.jsp**

4. Valor: "**remove**"

    - Recebe um parâmetro (via parameter, GET) com o Id do cliente a ser removido
    - Invoca a Façade para remover o cliente do banco de dados
    - Efetua forward para **ClientesServlet**

5. Valor: "**update**"

    - Recebe todos os dados do cliente passados via POST pelo formulário
    - Invoca a Façade para realizar uma atualização na base de dados
    - Efetua um redirecionamento para **ClientesServlet**.

6. Valor: "**formNew**"

    - Carrega a lista de estados, para apresentar na Combo
    - Efetua um redirecionamento para **clientesForm.jsp**.

7. Valor: "**new**"

    - Recebe todos os dados do cliente passados via POST pelo formulário
    - Invoca a Façade para realizar uma inserção na base de dados
    - Efetua um redirecionamento para **ClientesServlet**.

8) com.ufpr.tads.web2.beans.Cliente

Um Java Bean que contém os dados da tabela **tb_cliente**. Este bean deve ser usado juntamente com as classes DAO para acessar o banco de dados.

9) clientesListar.jsp

Inicialmente verifica se o usuário está logado. Se não estiver, redireciona via **forward** para **index.jsp** passando como parâmetro a mensagem **"Usuário deve se autenticar para acessar o sistema"** (parâmetro "msg").

Esta tela apresenta o mesmo formato que **portal.jsp**, alterando somente seu conteúdo. Deve apresentar, em forma de tabela, os dados de Cliente: CPF, Nome e E-mail. Adicionalmente, em cada linha, deve-se apresentar 3 pequenas imagens representando as ações: Visualizar, Alterar e Remover; com os seguintes links:

- Visualizar: para a Servlet **ClientesServlet?action=show&id=10** (onde id=10 é o id do cliente naquela linha)
- Alterar: para a Servlet **ClientesServlet?action=formUpdate&id=10** (idem)
- Remover: para a Servlet **ClientesServlet?action=remove&id=10** (idem). Ao pressionar Remover:
    - Primeiro deve-se, via JavaScript, perguntar ao usuário se ele deseja realmente fazer a remoção
    - Em o usuário indicando positivamente, pode então executar a ação

Use **<c:forEach items="${lista}" var="cliente" />** para apresentar esta lista. Para adicionar o id do cliente nos links, use por exemplo: **ClientesServlet?action=remove&id=${cliente.id}**

Esta tela também deve apresentar um botão de Novo, com um link para a servlet **ClientesServlet?action=formNew**.

10) clientesVisualizar.jsp

Inicialmente verifica se o usuário está logado. Se não estiver, redireciona via **forward** para **index.jsp** passando como parâmetro a mensagem **"Usuário deve se autenticar para acessar o sistema"** (parâmetro "msg").

Apresenta todos os dados do cliente (label e valor), exceto o ID. Apresenta também um botão Voltar com um link para **ClientesServlet?action=list**

11) clientesForm.jsp

Inicialmente verifica se o usuário está logado. Se não estiver, redireciona via **forward** para **index.jsp** passando como parâmetro a mensagem **"Usuário deve se autenticar para acessar o sistema"** (parâmetro "msg").

Esta tela deve funcionar tanto para alteração de cliente como para criação de cliente. Para isso, deve receber um parâmetro via request chamado "form". Se este parâmetro tiver o conteúdo "alterar", então o formulário se comportará como alteração de cliente. Caso "form" tenha outro conteúdo ou não for passado (null), então o formulário deve se comportar como fomulário para novo cliente.

Se for um formulário de alteração de clientes:

- Deve apresentar um formulário com todos os dados do cliente, já preenchidos com os dados recebidos via request, um botão **Alterar** e um botão **Cancelar**.
- Use EL (ex.: **${cliente.nome}**) para apresentar estes dados. Não é mais necessário usar a tag **<jsp:useBean />**
- Ao pressionar **Alterar**, deve-se submeter para **ClientesServlet?action=update**.
- Ao pressionar **Cancelar**, deve-se direcionar para **ClientesServlet**.

Se for um formulário de criação de clientes:

- Deve apresentar um formulário com todos os dados do cliente (limpos para uma adição), um botão **Salvar** e um botão **Cancelar**.
- Ao pressionar Salvar, deve-se submeter para **ClientesServlet&action=new**.
- Ao pressionar Cancelar, deve-se direcionar para **ClientesServlet**.

OS CAMPOS DO FORMULÁRIO **NÃO** PODEM SER DUPLICADOS. VOCÊ DEVE USAR CONDICIONAIS PARA CONTROLAR O CONTEÚDO DELES (ATRIBUTO VALUE) E PARA CONTROLAR O DESTINO DA SUBMISSÃO.

Em ambos os casos, deve-se apresentar duas ComboBoxes (campo <select />), uma para seleção de Estado e outra para seleção de Cidade. Primeiramente o usuário seleciona o Estado e, logo em seguida via AJAX, o segundo combo é preenchido com todas as Cidades daquele estado.

Link para baixar o Google GSON:
https://repo1.maven.org/maven2/com/google/code/gson/gson/2.8.2/gson-2.8.2.jar

Para implementar esta funcionalidade, você vai precisar de mais uma Servlet, aqui chamada de AJAXServlet, cujo conteúdo pode ser mais ou menos este:

```
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        String estado = request.getParameter("estadoId");

        // Vai no BD buscar todas as cidades deste estado, em uma lista

        List<Cidade> lista = new ArrayList<Cidade>();
        Cidade c = new Cidade();
        c.setId(1);
        c.setNome("Curitiba");
        lista.add(c);
        c = new Cidade();
        c.setId(2);
        c.setNome("Campo Largo");
        lista.add(c);


        // transforma o MAP em JSON
        String json = new Gson().toJson(lista);

        // retorna o JSON
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");
        response.getWriter().write(json);
    }
```

Também vai precisar alterar a JSP que contém as comboboxes, adicionando JQuery (se já não estiver adicionado por causa do framework) e criando os seguintes códigos dentro da seção <HEAD>:

```
        <script type="text/javascript" >

$(document).ready(function() {
    $( "#estado" ).change(function() {
      getCidades();
    });
});

function getCidades(){
    var estadoId = $("#estado").val();
    var url = "AJAXServlet";
    $.ajax({
            url : url, // URL da sua Servlet
            data : {
                estadoId : estadoId
            }, // Parâmetro passado para a Servlet
            dataType : 'json',
            success : function(data) {
                // Se sucesso, limpa e preenche a combo de cidade
                // alert(JSON.stringify(data));
                $("#cidade").empty();
                $.each(data, function(i, obj) {
                    $("#cidade").append('<option value=' + obj.id + '>' + obj.nome +
'</option>');
                });
```

```
            },
            error : function(request, textStatus, errorThrown) {
                alert(request.status + ', Error: ' + request.statusText);
                // Erro
            }
        });
}
</script>
```

~~11) clientesAlterar.jsp~~

~~Inicialmente verifica se o usuário está logado. Se não estiver, redireciona via **forward** para **index.jsp** passando como parâmetro a mensagem **"Usuário deve se autenticar para acessar o sistema"** (parâmetro "msg").~~

~~Deve apresentar um formulário com todos os dados do cliente, já preenchidos com os dados recebidos via request, um botão **Alterar** e um botão **Cancelar**.~~

~~Use EL (ex.: **${cliente.nome}**) para apresentar estes dados. Não é mais necessário usar a tag **<jsp:useBean />**~~

~~Ao pressionar **Alterar**, deve-se submeter para **ClientesServlet?action=update**. Ao pressionar **Cancelar**, deve-se direcionar para **ClientesServlet**.~~

~~12) clientesNovo.jsp~~

~~Inicialmente verifica se o usuário está logado. Se não estiver, redireciona via **forward** para **index.jsp** passando como parâmetro a mensagem **"Usuário deve se autenticar para acessar o sistema"** (parâmetro "msg").~~

~~Deve apresentar um formulário com todos os dados do cliente (limpos para uma adição), um botão **Salvar** e um botão **Cancelar**. Ao pressionar Salvar, deve-se submeter para **ClientesServlet&action=new**. Ao pressionar Cancelar, deve-se direcionar para **ClientesServlet**.~~

13) com.ufpr.tads.web2.beans.LoginBean

Um Java Bean que contém os dados de login a serem armazenados na sessão. Contém o id do usuário, e o nome do usuário.

14) com.ufpr.tads.web2.beans.ConfigBean

Um Java Bean que contém dados de configuração da aplicação, no caso o e-mail do administrador.

15) com.ufpr.tads.web2.beans.Usuario

Um Java Bean que contém os dados da tabela **tb_usuario**. Este bean deve ser usado juntamente com as classes DAO para acessar o banco de dados.

16) com.ufpr.tads.web2.dao.UsuarioDAO

É a classe DAO de acesso do usuário ao banco de dados.

17) com.ufpr.tads.web2.dao.ConnectionFactory

É a classe que controla a conexão com o banco de dados.

18) com.ufpr.tads.web2.dao.ClienteDAO

É a classe DAO de acesso do usuário ao banco de dados para manipular os dados de clientes.

19) com.ufpr.tads.web2.dao.CidadeDAO

É a classe DAO de acesso do usuário ao banco de dados para manipular os dados de cidades.

20) com.ufpr.tads.web2.beans.Cidade

Um Java Bean que contém os dados da tabela **tb_cidade**. Este bean deve ser usado juntamente com as classes DAO para acessar o banco de dados.

21) com.ufpr.tads.web2.dao.EstadoDAO

É a classe DAO de acesso do usuário ao banco de dados para manipular os dados de estados.

22) com.ufpr.tads.web2.beans.Estado

Um Java Bean que contém os dados da tabela **tb_estado**. Este bean deve ser usado juntamente com as classes DAO para acessar o banco de dados.