

# Exercício 8 - Atendimento

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

- Proibido o uso de Scriptlets. 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
- Deve-se usar Façade para a Servlet efetuar as operações: **LoginFacade**, **ClientesFacade**, **AtendimentoFacade**.
- Todos os campos dos formulários de entrada de dados devem ter:
  - Tamanhos máximos conforme o dado e tabela
  - Máscaras para entrada de dados formatadas (ex, CEP, CPF, Telefone, etc)
  - Validação de campos obrigatório ou não
  - Calendários quando for entrada de datas
  - Campos monetários e datas devem ser entrados e mostrados em formato brasileiro
- 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)
- Todas as classes Java deverão estar dentro dos pacotes conforme indicado nas descrições abaixo.
- Deve-se adicionar uma tela de Atendimento ao Cliente contendo: O cliente a ser atendido, a data/hora do atendimento (deve vir como default da data/hora atual), tipo do atendimento (reclamação, dúvida, sugestão ou elogio), produto, observação.
- Todos os métodos das Façades deverão ter o seguinte comportamento:
  - Em caso de sucesso, OU não retornar nada OU retornar os dados solicitados (ex, quando solicitado um cliente por ID)
  - Em caso de erro, levantar uma exceção da aplicação, criada para o projeto. Cada método deve levantar suas exceções específicas (não deve levantar uma exceção genérica).
- Demais alterações realçadas ao longo da especificação.

## Tabelas

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.

Deve-se adicionar 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 adicionar as seguintes tabelas:

- **tb\_produto**, contendo **id\_produto** (PK, inteiro), **nome\_produto** (VARCHAR(100)).
- **tb\_tipo\_atendimento**, contendo **id\_tipo\_atendimento** (PK, inteiro), **nome\_tipo\_atendimento** (VARCHAR(50)).
- **tb\_atendimento**, contendo **id\_atendimento** (PK, inteiro), **dt\_hr\_atendimento** (data/hora atendimento DATETIME/TIMESTAMP), **dsc\_atendimento** (VARCHAR(255)), **id\_produto** (FK para tabela de produto), **id\_tipo\_atendimento** (FK para tabela tb\_tipo\_atendimento), **id\_usuario** (FK para tabela de usuário, usuário que efetuou o atendimento), **id\_cliente** (FK para tabela de cliente, cliente para o qual se efetuou o atendimento), **res\_atendimento** (CHAR(1) contendo 'S' para atendimento resolvido e 'N' para não resolvido).

Adicione pelo menos 10 produtos e adicione os quatro tipos de atendimento. Para estas tabelas não é necessário fazer CRUD. A tabela de atendimentos não possui característica de CRUD, por ser um processo de negócio.

Para gerenciar atendimentos deve-se:

- Implementar **AtendimentoServlet**, tratando todas as ações para atendimento (seguindo os mesmos moldes de ClientesServlet);
- Implementar **AtendimentoFacade**, contendo todas as operações necessárias para atendimento (seguindo os mesmos moldes de ClientesFacade). Aqui também faz-se as buscas por tipo de atendimento, produtos;

- Implementar **atendimento.jsp**, que é uma tela usada para se efetuar o atendimento. Ela deve vir preenchida com: data/hora default, lista de produtos, lista de tipos de atendimento e lista de clientes. Para saber se o atendimento foi ou não resolvido deve-se usar um *checkbox*;
- Implementar **atendimentoListar.jsp**, que é uma tela usada para mostrar a lista de atendimentos efetuados pelo usuário logado. Deve apresentar na lista data/hora, produto e nome do cliente. Deve apresentar um botão de detalhes que apresenta a tela **atendimentoDetalhes.jsp**;
- Implementar **atendimentoDetalhes.jsp**, que mostra todos os dados daquele atendimento selecionado. Não se pode editar ou remover um atendimento.

Na tela **portal.jsp** deve-se adicionar mais um Menu de Atendimentos, contendo as seguintes opções:

- Efetuar Atendimento: que abre a tela **atendimento.jsp**, para efetuar atendimento
- Mostrar Atendimentos: que abre a tela **atendimentoListar.jsp**, que mostra a lista de todos os atendimentos efetuados pelo usuário logado.

Todas as telas JSPs e HTMLs devem usar Bootstrap. Link para básico do BOOTSTRAP: [https://www.w3schools.com/bootstrap4/bootstrap\\_get\\_started.asp](https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp)

## Exceções

Deve-se possuir uma exceção chamada **AppException** dentro do pacote (**com.ufpr.tads.web2.exceptions**), que é a classe mãe de todas as demais exceções do sistema. Para cada caso no projeto deverá ser criada uma exceção (filha de AppException):

- UsuarioSenhaInvalidosException
- ClienteNaoExisteException
- ErroInserindoClienteException
- ErroRemovendoClienteException
- ErroBuscandoClienteException
- Etc.

Todas as servlets deverão tratar as exceções e decidir se são exceções tratáveis ou exceções impeditivas, para levantar erro. Ex. Erro de conexão com o banco é uma exceção impeditiva; já UsuarioSenhaInvalidosException é uma exceção tratável que se mostra uma mensagem em uma tela específica.

Em caso de Exceções impeditivas, deve-se direcionar para a página de erro. Em Servlets pode-se redirecionar para página de erro (marcada com `isErrorPage`) da seguinte forma:

```
request.setAttribute("javax.servlet.jsp.jspException", <objeto Exceção a ser mostrada> );
request.setAttribute("javax.servlet.error.status_code", 500);
RequestDispatcher rd = getServletContext().getRequestDispatcher("/erro.jsp");
rd.forward(request, response);
```

Em caso de Exceções tratáveis, deve-se direcionar para a página certa passando como parâmetro a mensagem específica (ex. em caso de cliente inexistente, deve-se direcionar para **clientesListar.jsp** passando uma mensagem a ser mostrada para o usuário).

## Arquivos

### 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>**,
- Coloca a lista retornada no escopo da requisição.

- 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 formulário 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>

```

### 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.

### **23) com.ufpr.tads.web2.dao.TipoAtendimentoDAO**

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

### **24) com.ufpr.tads.web2.dao.ProdutoDAO**

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

### **25) com.ufpr.tads.web2.beans.TipoAtendimento**

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

### **26) com.ufpr.tads.web2.beans.Produto**

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

### **27) com.ufpr.tads.web2.beans.Atendimento**

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

### **28) com.ufpr.tads.web2.beans.Atendimento**

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