DESENVOLVIMENTO WEB BACK END

AULA 5

Prof. Rafael Moraes

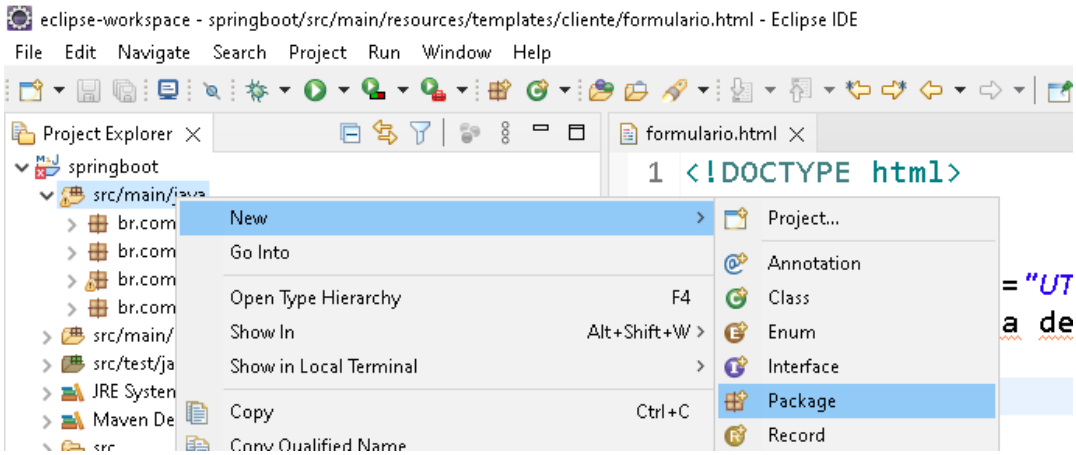
CONVERSA INICIAL

Desenvolvida a tela de cadastro do cliente, na sequência precisamos exibir o formulário de cadastro do cliente para o usuário, a fim de que ele possa realizar a persistência de dados. Além disso, temos que disponibilizar uma tela para que o usuário possa gerenciar os cadastros existentes, permitindo que ele possa alterar os dados cadastrais caso seja necessário. Para tornar isso possível, teremos que implementar os controladores para atender às requisições efetuadas pelo usuário, além de prover a tela de gerenciamento de cadastro do cliente.

TEMA 1 – CRIANDO O CONTROLLER

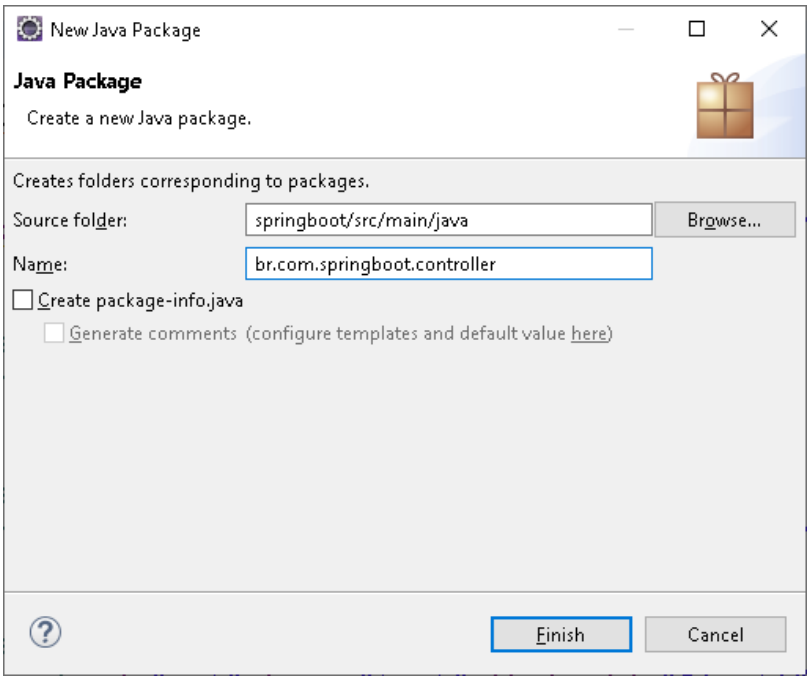
Finalizado o desenvolvimento do formulário de cadastro do cliente, precisamos disponibilizá-lo para o usuário por meio de uma url. Para isso, devemos implementar um controller, componente responsável por tratar as requisições http. A fim de manter o projeto organizado, vamos criar um pacote específico para os controllers da aplicação denominado br.com.springboot.controller. Isso posto, clique com o botão direito sobre a pasta src/main/java e acesse o menu New > Package, conforme mostra a figura a seguir:

Figura 1 – Menu para criação do pacote



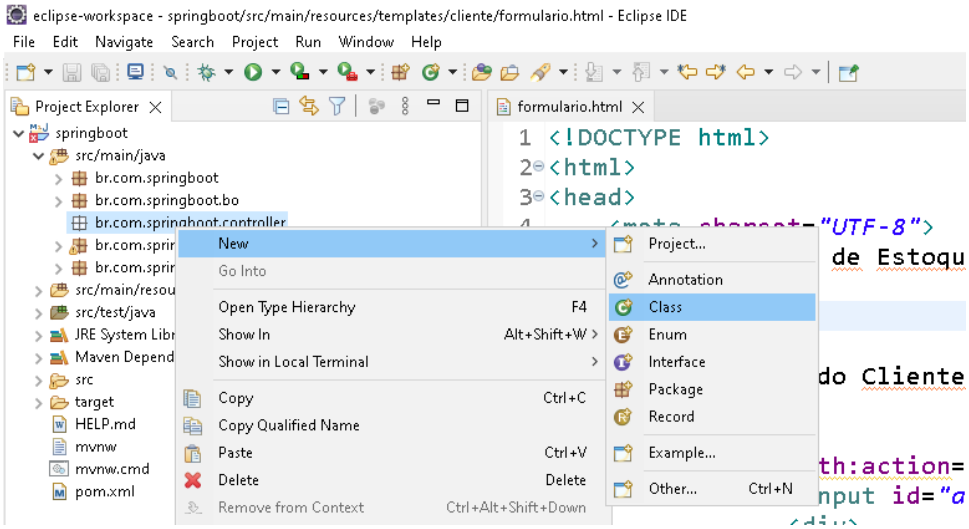
Na tela de cadastro de pacote, informe no campo Name o nome do pacote e clique no botão Finish, conforme mostra a figura a seguir:

Figura 2 – Tela de criação do pacote



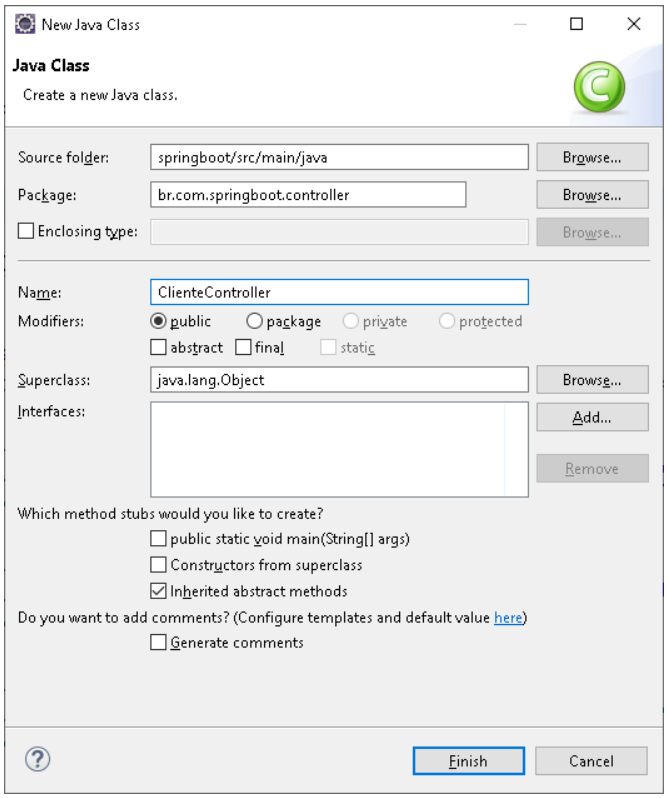
Adicionado o pacote referente aos controllers da aplicação, vamos adicionar a classe ClienteController a esse pacote. Para tal, clique com o botão direito sobre o pacote e acesse o menu New > Class, conforme mostra a figura a seguir:

Figura 3 – Menu para criação da classe



Na tela de cadastro da classe, informe o nome da classe no campo Name e clique no botão Finish, conforme mostra a figura a seguir:

Figura 4 – Tela de criação da classe



Criada a classe ClienteController, vamos anotá-la utilizando duas anotações do Spring: @Controller e @RequestMapping, conforme mostra o quadro a seguir:

Quadro 1 – Configuração da classe ClienteController

```
package br.com.springboot.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/clientes")
public class ClienteController {

}
```

A anotação @Controller fará com que o Spring reconheça essa classe como um controller e passe a gerenciar o seu ciclo de vida. Já a anotação @RequestMapping irá configurar a url path inicial das requisições para esse controller. Dessa forma, ao tentar acessar o endereço <**Erro! A referência de hiperlink não é válida.**http://localhost:8080/clientes> por meio de um navegador, essa requisição será encaminhada para que a classe ClienteController realize o seu processamento.

Como queremos exibir o formulário de cadastro do cliente para o usuário, devemos criar um método dentro da classe ClienteController para realizar essa tarefa. Esse formulário será disponibilizado por meio da url <**Erro! A referência de hiperlink não é válida.**http://localhost:8080/clientes/novo> por meio do método GET. Portanto, vamos adicionar à classe ClienteController o método novo, conforme mostra o quadro a seguir:

Quadro 2 – Método novo da classe ClienteController

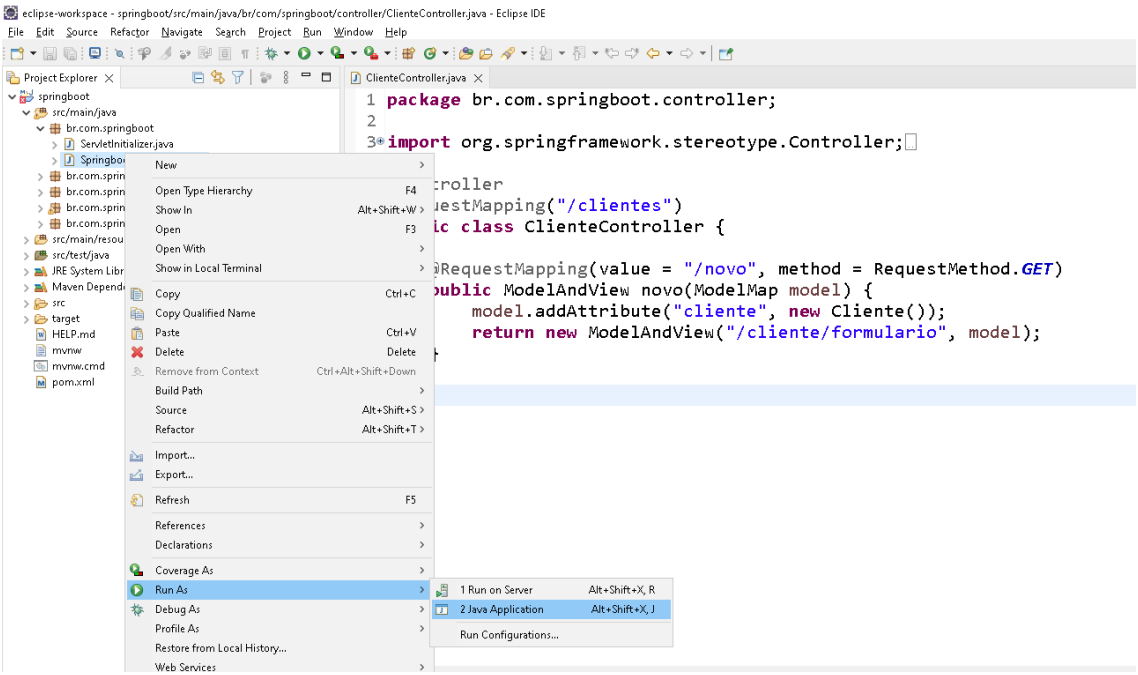
```
package br.com.springboot.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import br.com.springboot.model.Cliente;

@Controller
@RequestMapping("/clientes")
public class ClienteController {

	@RequestMapping(value = "/novo", method = RequestMethod.GET)
	public ModelAndView novo(ModelMap model) {
		model.addAttribute("cliente", new Cliente());
		return new ModelAndView("/cliente/formulario", model);
	}
}
```

O método novo irá retornar um objeto do tipo ModelAndView, ou seja, um objeto com a página Web (view) e os dados que serão populados na tela (model). A página Web que será exibida no navegador é o formulário de cadastro do cliente, portanto, no primeiro parâmetro do objeto ModelAndView, deve-se informar o caminho do arquivo que implementar essa tela com base na pasta templates, sem a necessidade de informar a extensão do arquivo. O segundo parâmetro a ser fornecido consiste nos dados que serão populados na tela por meio de um objeto do tipo ModelMap.

Para adicionar dados a esse objeto, deve-se utilizar o método addAttribute, no qual o primeiro parâmetro é o nome do atributo que será acessado pelo Thymeleaf e, no segundo parâmetro, será fornecido o conteúdo desse atributo, que pode ser uma variável ou objeto. No caso, vamos adicionar ao objeto ModelMap o atributo cliente, que irá conter os dados de uma nova instância da classe Cliente. Dessa forma, iremos iniciar o formulário com o campo “Registro ativo” selecionado e os demais campos em branco.

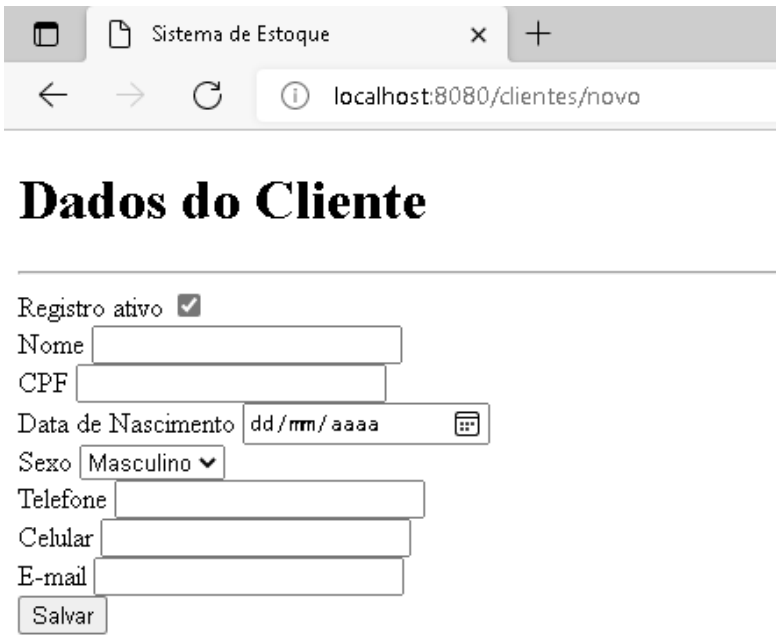
Inicie a aplicação clicando com o botão direito sobre a classe SpringApplication e acesse o menu Run As > Java Application, conforme mostra a figura a seguir:

Figura 5 – Menu para iniciar a aplicação



Iniciada a aplicação, abra o navegador e acesse a url <**Erro! A referência de hiperlink não é válida.**http://localhost:8080/clientes/novo> para que seja carregado o formulário de cadastro do cliente, conforme mostra a figura a seguir:

Figura 6 – Formulário de cadastro do cliente



TEMA 2 – REALIZANDO O PRIMEIRO CADASTRO VIA APLICAÇÃO

Exibido o formulário de cadastro do cliente, precisamos agora implementar a ação que será executada quando o usuário clicar no botão salvar. Para isso, na tag form desse formulário deve-se informar, obrigatoriamente, dois atributos para que possamos submeter esse formulário para o servidor da aplicação. O primeiro atributo é a ação que será executada, identificada por meio do atributo *th:action*. Por sua vez, o segundo atributo é o método da ação que será executada, identificado por meio do atributo method. No quadro a seguir, temos a configuração que foi adicionada anteriormente à tag form do formulário de cadastro do cliente.

Quadro 3 – Configuração da tag form

```
<form th:action="@{/clientes}" th:object="${cliente}" method="POST">

</form>
```

Ao submeter o formulário de cadastro do cliente, ou seja, quando o usuário clicar no botão salvar, será enviada uma requisição HTTP do tipo POST para a url <**Erro! A referência de hiperlink não é válida.**http://localhost:8080/clientes>, conforme foram parametrizados os atributos *th:action* e *method*. Portanto, deve-se adicionar mais um método dentro da classe ClienteController para que possamos realizar a persistência de dados assim que o usuário submeter o formulário de cadastro à aplicação. Dentro da classe ClienteController, vamos adicionar o método salva para realizar essa tarefa, conforme mostra o quadro a seguir:

Quadro 4 – Método salva da classe ClienteController

```
@RequestMapping(value = "", method=RequestMethod.POST)
public String salva(@ModelAttribute("cliente") Cliente cliente) {

}
```

Novamente, faremos o uso da anotação @RequestMapping para definir a url path e o método da requisição HTTP, conforme foram parametrizados os atributos *th:action* e *method* na tag form do formulário de cadastro do cliente. Além dessa anotação, vamos utilizar também a anotação @ModelAttribute do Spring, a qual será responsável em converter o objeto de comando do Thymeleaf, atributo *th:object* da tag form, em um objeto Java.

Dessa forma, o método salva irá prover ao desenvolvedor, por parâmetro, um objeto do tipo Cliente já populado com os valores digitados pelo usuário no formulário de cadastro do cliente.

Como o método salva será responsável tanto por inserir um novo registro quanto por atualizar um registro já existente, devemos implementar essa regra de negócio baseado no atributo id do objeto cliente. Caso o id seja nulo, será invocado o método insere da classe ClienteBO, do contrário, será invocado o método atualiza dessa mesma classe. Ao realizar a persistência dos dados, vamos redirecionar a aplicação, por ora, novamente para a tela de cadastro do cliente. Portanto, conforme mostra quadro a seguir, teremos a seguinte implementação para a classe ClienteController:

Quadro 5 – Classe ClienteController

```
package br.com.springboot.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import br.com.springboot.bo.ClienteBO;
import br.com.springboot.model.Cliente;

@Controller
@RequestMapping("/clientes")
public class ClienteController {

	@Autowired
	private ClienteBO clienteBO;

	@RequestMapping(value = "/novo", method = RequestMethod.GET)
	public ModelAndView novo(ModelMap model) {
		model.addAttribute("cliente", new Cliente());
		return new ModelAndView("/cliente/formulario", model);
	}

	@RequestMapping(value = "", method=RequestMethod.POST)
	public String salva(@ModelAttribute("cliente") Cliente cliente) {
		if (cliente.getId() == null)
			clienteBO.insere(cliente);
		else
			clienteBO.atualiza(cliente);
		return "redirect:/clientes/novo";
	}
}
```

Inicie novamente a aplicação e, por meio do navegador, acesse a url <**Erro! A referência de hiperlink não é válida.**http://localhost:8080/clientes/novo>, preencha todos os campos da tela e clique no botão salvar. Ao salvar, caso não tenha ocorrido qualquer erro, acesse o MySQL e verifique se o registro foi inserido com sucesso. Na figura 7 é exibido o formulário de cadastro do cliente preenchido e, na figura 8, os dados que foram persistidos no banco de dados ao acionar o botão salvar do formulário.

Figura 7 – Formulário de cadastro do cliente preenchido

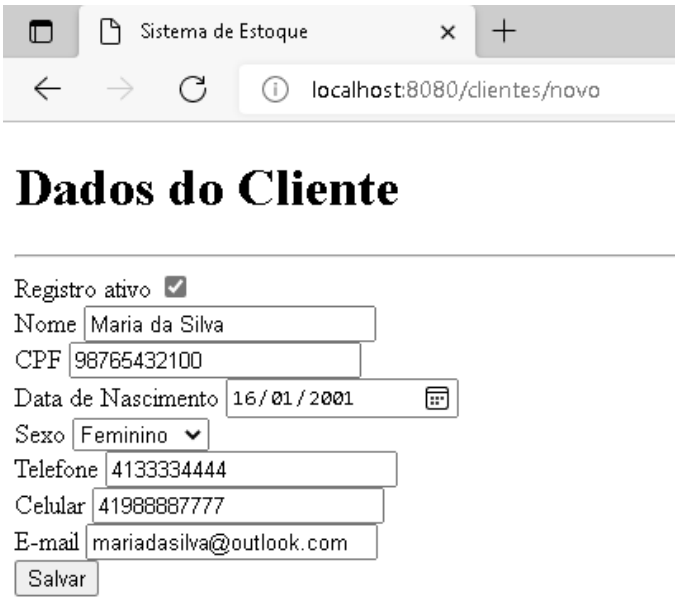
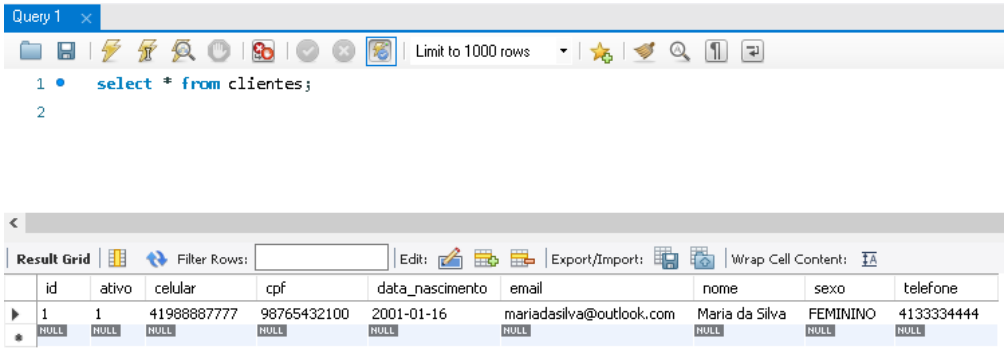


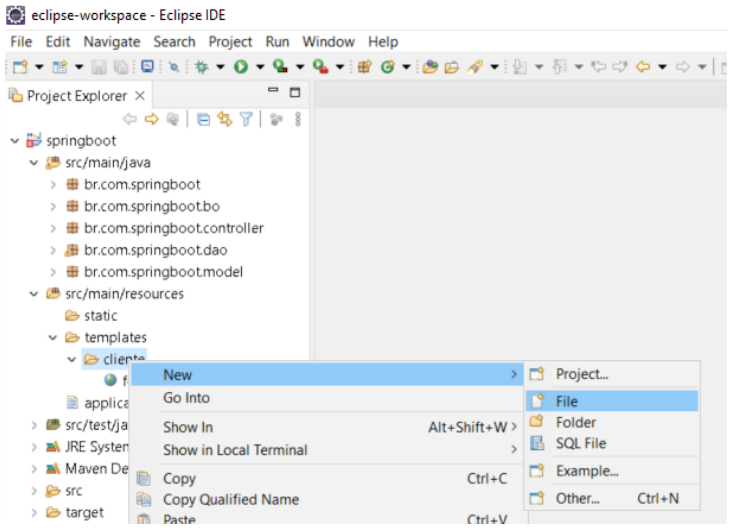
Figura 8 – Registro persistido



TEMA 3 – CRIANDO A TELA DE GERENCIAMENTO DE CADASTRO DO CLIENTE

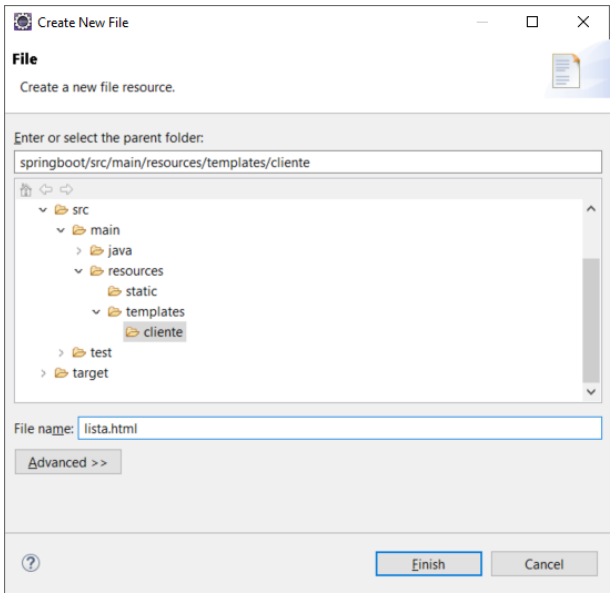
Após criar a tela de cadastro do cliente, torna-se necessário o desenvolvimento da tela de gerenciamento dos cadastros para que o usuário possa visualizar todos os clientes cadastrados no sistema, além de ter acesso às ações de cadastrar, editar e ativar/inativar um cliente. Para isso, adicione o arquivo lista.html dentro da pasta cliente, clique com o botão direito sobre ela e acesse o menu New > File, conforme mostra a imagem a seguir:

Figura 9 – Adicionar um novo arquivo



Após clicar sobre o item do menu citado anteriormente, será aberta a tela para criação do arquivo. Informe no campo File name o nome do arquivo e clique no botão Finish, conforme a figura a seguir:

Figura 10 – Tela de criação do arquivo



Além das tags HTML abordadas anteriormente, visando ao desenvolvimento da tela de cadastro do cliente, serão utilizadas mais algumas tags para o desenvolvimento da tela de gerenciamento de cadastro, conforme mostra a tabela a seguir:

Tabela 1 – Tags para o desenvolvimento da tela de gerenciamento

Tag	Descrição
<a>	Adiciona um link ao documento
<table>	Adiciona uma tabela ao documento
<thead>	Define um cabeçalho para a tabela
<tbody>	Define o corpo da tabela
<tr>	Adiciona uma linha à tabela
<td>	Adiciona uma célula à linha da tabela
<b>	Deixa o texto em negrito

No quadro a seguir, temos a estrutura base da tela de gerenciamento de cadastro do cliente.

Quadro 6 – Estrutura base da tela de gerenciamento de cadastro do cliente

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>Sistema de Estoque</title>
</head>
<body>
   <div>
      <h1>Clientes</h1>
      <hr>
      <div>
         <a th:href="@{/clientes/novo}">Novo</a>
      </div>
      <hr>
      <table>
      </table>
   </div>
</body>
</html>
```

Note que foram adicionados à estrutura base da tela de gerenciamento de cadastro de cliente um link e uma tabela. O link permitirá que o usuário possa, por meio dessa tela, acessar a tela de cadastro do cliente e efetuar um novo cadastro. Para isso, deve-se informar no atributo th:href do link, a url que irá redirecionar a aplicação para a tela de cadastro do cliente. A tabela deverá exibir todos os clientes cadastrados no sistema, exibindo suas principais informações e, para cada registro listado na tabela, disponibilizar opções para que o usuário possa editar ou ativar/inativar um determinado registro. Portanto, deve-se adicionar a tag table mais duas tags: thead e tbody.

Na *tag thead* será definido o cabeçalho da tabela, ou seja, nela serão especificadas as colunas que irão compor essa tabela. Para isso, deve-se acionar uma linha ao cabeçalho da tabela por meio da tag tr e especificar suas colunas por meio da tag td. A tabela da tela de gerenciamento de clientes será composta pelas seguintes colunas.

- **Nome**: coluna que irá exibir o nome do cliente.
- **Data de nascimento**: coluna que irá exibir a data de nascimento do cliente.
- **CPF**: coluna que irá exibir o CPF do cliente.
- **Editar**: coluna que irá disponibilizar um link para que o usuário possa editar o cadastro.
- **Ativar/Inativar**: coluna que irá disponibilizar um link para que o usuário possa ativar/inativar o cadastro.

Já na tag tbody serão adicionadas à tabela as informações referentes a cada cliente cadastrado no sistema. Para isso, iremos acessar o objeto que contém a lista de clientes cadastrados no sistema, utilizando para isso o atributo th:each do Thymeleaf, a fim de acessar cada elemento da lista por meio de um laço de repetição. A cada laço, será adicionada uma linha à tabela e o conteúdo de cada objeto populado à coluna correspondente, por meio do atributo th:text da tag td. As colunas responsáveis pelas operações de editar e ativar/inativar irão conter os links referentes às operações pertinentes. A estrutura da tag table da tela de gerenciamento de cadastro do cliente é exibida no quadro a seguir:

Quadro 7 – Estrutura da tag table

```
<table>
  <thead>
    <tr>
      <td><b>NOME</b></td>
      <td><b>DATA NASCIMENTO</b></td>
      <td><b>CPF</b></td>
      <td></td>
      <td></td>
    </tr>
  </thead>
  <tbody>
```

```
    <tr th:each="cliente : ${clientes}">
      <td th:text="${cliente.nome}"></td>
      <td th:text="${cliente.dataDeNascimento}"></td>
      <td th:text="${cliente.cpf}"></td>
      <td>
        <a th:href="@{/clientes/edita/{id}(id=${cliente.id})}">Editar</a>
      </td>
      <td>
        <a th:if="${cliente.ativo == false}"
th:href="@{/clientes/ativa/{id}(id=${cliente.id})}">Ativar</a>
        <a th:unless="${cliente.ativo == false}"
th:href="@{/clientes/inativa/{id}(id=${cliente.id})}">Inativar</a>
      </td>
    </tr>
  </tbody>
</table>
```

Repare que nos links foram adicionados os atributos th:if e th:unless. Dessa forma, conseguimos habilitar a opção de ativar para o registro cujo atributo ativo seja falso, e caso o atributo ativo seja verdadeiro, habilitar a opção de inativar o registro.

TEMA 4 – IMPLEMENTANDO AS FUNCIONALIDADES DA TELA DE GERENCIAMENTO

Criada a tela de gerenciamento de cadastro de clientes, devemos implementar as funcionalidades dessa tela, adicionando à classe ClienteController os métodos responsáveis pelas tarefas de editar e ativar/inativar um cadastro, além do método que irá exibir a tela de gerenciamento para o usuário.

Primeiramente, vamos implementar o método lista, o qual será responsável por redirecionar a aplicação para a tela de gerenciamento de cadastro de clientes, que será acessada por meio de uma requisição do tipo GET utilizando a url <**Erro! A referência de hiperlink não é válida.**http://localhost:8080/clientes>. O método lista deverá retornar um objeto do tipo ModelAndView contendo a página de gerenciamento e a lista de clientes cadastrados no sistema, conforme o quadro a seguir:

Quadro 8 – Método lista da classe ClienteController

```
@RequestMapping(value = "", method=RequestMethod.GET)
public ModelAndView lista(ModelMap model) {
   model.addAttribute("clientes", clienteBO.listaTodos());
   return new ModelAndView("/cliente/lista", model);
}
```

Na sequência, iremos adicionar o método edita à classe ClienteController, o qual será responsável por exibir a tela de cadastro do cliente preenchida com os dados do registro que será editado pelo usuário. A edição do registro será feita por meio de uma requisição do tipo GET utilizando a url <**Erro! A referência de hiperlink não é válida.**http://localhost:8080/clientes/edita/{id}>, na qual o termo {id} será substituído pelo id do cliente que será editado. Para que tenhamos acesso ao id do cliente presente na url da requisição, devemos utilizar a anotação PathVariable do Spring, a qual irá extrair esse dado da url e atribuir a uma variável.

Em posse do id, podemos obter os dados do cliente por meio do método pesquisaPeloId do objeto da classe ClienteBO injetado na classe ClienteController. Dessa forma, conseguimos retornar um objeto do tipo ModelAndView contendo o formulário de cadastro do cliente e os dados que serão utilizados pelo Thymeleaf para renderizar a tela. O código referente ao método edita pode ser visto no quadro a seguir:

Quadro 9 – Método edita da classe ClienteController

```
@RequestMapping(value = "/edita/{id}", method = RequestMethod.GET)
public ModelAndView edita(@PathVariable("id") Long id, ModelMap model) {
   try {
      model.addAttribute("cliente", clienteBO.pesquisaPeloId(id));
   } catch (Exception e) {
      e.printStackTrace();
   }
   return new ModelAndView("/cliente/formulario", model);
}
```

Outro recurso dessa tela que devemos implementar é a funcionalidade de inativar um cadastro. Portanto, deve-se implementar dentro da classe ClienteController o método inativa, que será responsável por realizar tal tarefa. Esse método irá funcionar de forma semelhante ao método edita, especificando na url a operação que será realizada e o id do cadastro, de tal modo que a url terá o seguinte padrão <**Erro! A referência de hiperlink não é válida.**http://localhost:8080/clientes/inativa/{id}>. No quadro a seguir temos a implementação do método inativa:

Quadro 10 – Método inativa da classe ClienteController

```
@RequestMapping(value = "/inativa/{id}", method = RequestMethod.GET)
public String inativa(@PathVariable("id") Long id) {
   try {
      Cliente cliente = clienteBO.pesquisaPeloId(id);
      clienteBO.inativa(cliente);
   } catch (Exception e) {
      e.printStackTrace();
   }
   return "redirect:/clientes";
}
```

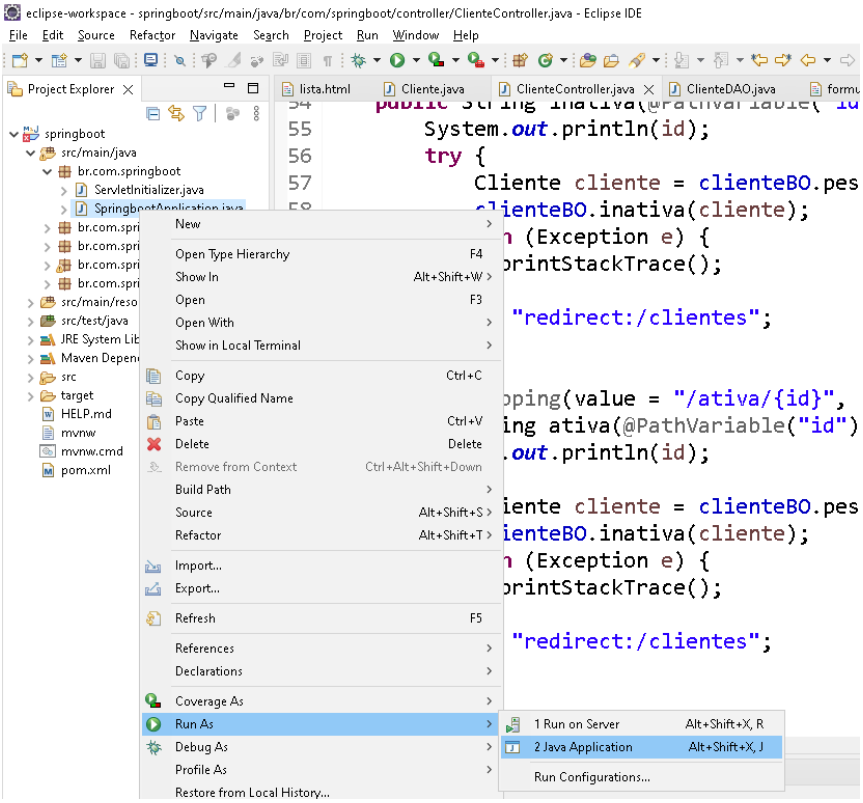
Repare que o retorno desse método não é um objeto do tipo ModelAndView, e sim uma String, indicando que, após inativar o registro, a aplicação será redirecionada para a tela de gerenciamento de cadastro do cliente. Vamos adotar essa mesma abordagem para o método salva, dessa forma, sempre que o usuário cadastrar ou alterar um cadastro, a aplicação será redirecionada para a tela de gerenciamento.

Quadro 11 – Método salva da classe ClienteController

```
@RequestMapping(value = "", method=RequestMethod.POST)
public String salva(@ModelAttribute("cliente") Cliente cliente) {
   if (cliente.getId() == null)
      clienteBO.insere(cliente);
   else
      clienteBO.atualiza(cliente);
   return "redirect:/clientes";
}
```

Ao finalizar a implementação dos métodos citados anteriormente, execute a aplicação clicando com o botão direito sobre a classe SpringbootApplication, localizada dentro do pacote br.com.springboot, e acesse o menu Run As > Java Application, conforme a figura a seguir:

Figura 11 – Iniciando a aplicação



Assim que a aplicação for iniciada, abra o navegador e acesse a url <**Erro! A referência de hiperlink não é válida.**http://localhost:8080/clientes> para visualizar a tela de gerenciamento de cadastro do cliente, conforme mostra a figura a seguir:

Figura 12 – Tela de gerenciamento de cadastro do cliente

Na tela de gerenciamento de cadastro do cliente, navegue pelos links e verifique se a aplicação está efetuando as requisições solicitadas corretamente. Inative um cadastro utilizando o link Inativar e, na sequência, tente ativar o mesmo cadastro por meio do link Ativar. Note que a operação solicitada não foi efetuada, pois faltou implementar esse recurso.

Tomando como base a opção de inativar um cadastro, implemente a opção para ativar o cadastro. Lembre-se de que você terá que adicionar um método na classe ClienteController para atender a essa requisição e outro método na classe ClienteBO para realizar essa tarefa.

TEMA 5 – UTILIZANDO O BOOTSTRAP

Desenvolvido por um designer e desenvolvedor do Twitter em meados de 2010, o Bootstrap tornou-se um dos frameworks de front-end e projetos de código aberto mais populares do mercado. Utilizando suas bibliotecas, podemos desenvolver, de forma fácil e rápida, aplicações responsivas com um visual elegante. Para isso, devemos adicionar ao código HTML dois arquivos.

- **Bootstrap.min.css**: folha de estilo com o código CSS para estilizar os componentes da tela. Esse arquivo deve ser adicionado dentro da tag head da página HTML.
- **Bootstrap.bundle.min.js**: arquivo de script para o correto funcionamento dos componentes visuais. Esse arquivo deve ser adicionado antes do fechamento da tag body da página HTML.

Vamos adicionar os dois arquivos tanto na página de cadastro do cliente quanto na página de gerenciamento de cadastro do cliente por meio dos links da CDN fornecidos pelo próprio Bootstrap.

Portanto, nos arquivos HTML referentes a essas páginas, adicione o código do quadro 12 dentro da tag head e o código do quadro 13 antes do fechamento da tag body.

Quadro 12 – Folha de estilo CDN

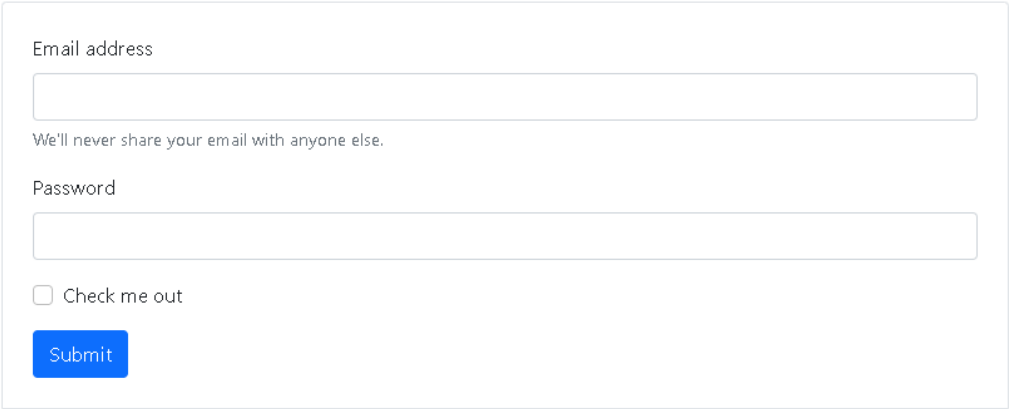
```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.c
ss" rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
```

Quadro 13 – Arquivo de script CDN

```
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.
min.js" integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>
```

Adicionados os arquivos às páginas, basta aplicar aos componentes HTML às classes do Bootstrap para estilizá-los. Por meio do site <https://getbootstrap.com/>, você tem acesso à documentação desse framework, em que são apresentados diversos exemplos de estilos e o seu respectivo código HTML. Na figura a seguir, temos uma tela estilizada com as classes do Bootstrap e, no quadro 14, o seu código HTML, ambos extraídos da documentação do framework.

Figura 13 – Formulário estilizado com Bootstrap



Quadro 14 – Código fonte do formulário estilizado com Bootstrap

```
<form>
  <div class="mb-3">
    <label for="exampleInputEmail1" class="form-label">Email
address</label>
    <input type="email" class="form-control" id="exampleInputEmail1" aria-
describedby="emailHelp">
    <div id="emailHelp" class="form-text">We'll never share your email with
anyone else.</div>
  </div>
  <div class="mb-3">
    <label for="exampleInputPassword1" class="form-label">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword1">
  </div>
  <div class="mb-3 form-check">
    <input type="checkbox" class="form-check-input" id="exampleCheck1">
    <label class="form-check-label" for="exampleCheck1">Check me
out</label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Veja que, de forma simples, conseguimos estilizar a interface de uma página HTML, bastando para isso importar as bibliotecas do framework e vincular as classes do Bootstrap aos componentes da tela.

5.1 Formulário de cadastro do cliente

No arquivo referente ao formulário de cadastro do cliente, vamos adicionar algumas classes aos componentes que compõem essa tela. Dentre elas, destacam-se:

Tabela 2 – Classes aplicadas ao formulário de cadastro

Classe	Descrição
container	Classe que centraliza o formulário com espaçamentos laterais
mb	Classe que define o espaçamento entre elementos
form-check	Classe para estilizar o elemento que contém a label e o campo de entrada do tipo checkbox
form-switch	Classe para estilizar o checkbox no formato switch
form-check-label	Classe para estilizar a label do checkbox
form-check-input	Classe para estilizar um campo de entrada do tipo checkbox

form-control	Classe para estilizar um campo de entrada do tipo texto
row	Define o elemento com uma linha, tornando possível definir o tamanho e demais elementos que compõem essa linha
col	Define o tamanho de um elemento dentro de uma linha
btn	Deixa o elemento com formato de um botão, podendo essa classe ser aplicada a links
btn-primary	Deixa o botão na cor azul (botão principal da tela)

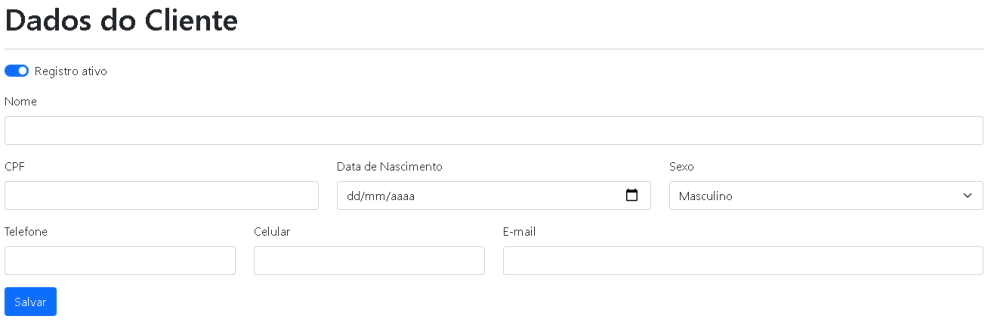
No quadro a seguir, temos o código fonte do formulário de cadastro do cliente utilizando as classes do Bootstrap, que irão resultar na interface vista na Figura 14.

Quadro 15 – Código fonte do formulário estilizado com Bootstrap

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Sistema de Estoque</title>

  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.c
ss" rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
</head>
<body>
  <div class="container">
    <h1>Dados do Cliente</h1>
    <hr>
    <div>
      <form th:action="@{/clientes}" th:object="${cliente}" method="POST">
        <input id="ativo" type="hidden" th:field="*{id}"/>
        <div class="mb-3 form-check form-switch">
```

```
          <label class="form-check-label" for="ativo">Registro ativo</label>
          <input class="form-check-input" role="switch" id="ativo"
                 type="checkbox" th:field="*{ativo}"/>
        </div>
        <div class="mb-6">
          <label class="form-label" for="nome">Nome</label>
          <input class="form-control" id="nome" type="text"
                 th:field="*{nome}"/>
        </div>
        <div class="row">
          <div class="col-4 mb-3">
            <label class="form-label" for="cpf">CPF</label>
            <input class="form-control" id="cpf" type="text"
                   th:field="*{cpf}">
          </div>
          <div class="col-4 mb-3">
            <label class="form-label" for="dataDeNascimento">
            Data de Nascimento</label>
            <input class="form-control" id="dataDeNascimento" type="date"
                   th:field="*{dataDeNascimento}" />
          </div>
          <div class="col-4 mb-3">
            <label class="form-label" for="sexo">Sexo</label>
            <select class="form-select" id="sexo" th:field="*{sexo}">
              <option value="MASCULINO">Masculino</option>
              <option value="FEMININO">Feminino</option>
            </select>
          </div>
        </div>
        <div class="row">
          <div class="col-3 mb-3">
            <label class="form-label" for="telefone">Telefone</label>
            <input class="form-control" id="telefone" type="text"
                   th:field="*{telefone}"/>
          </div>
          <div class="col-3 mb-3">
            <label class="form-label" for="celular">Celular</label>
            <input class="form-control" id="celular" type="text"
                   th:field="*{celular}"/>
          </div>
          <div class="col-6 mb-3">
            <label class="form-label" for="email">E-mail</label>
            <input class="form-control" id="email" type="text"
                   th:field="*{email}"/>
          </div>
        </div>
        <div class="mb-3">
          <input class="btn btn-primary" type="submit" value="Salvar"/>
        </div>
      </form>
    </div>
  </div>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.
min.js" integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>
</body>
</html>
```

Figura 14 – Interface do formulário de cadastro do cliente



5.2 Tela de gerenciamento de cadastro do cliente

No arquivo referente à tela de gerenciamento de cadastro do cliente, além das classes citadas anteriormente, vamos adicionar outras classes, das quais destacam-se:

Tabela 3 – Classes aplicadas na tela de gerenciamento de cadastro do cliente

Classe	Descrição
table	Classe para estilizar uma tabela
table-hover	Destaca a linha da tabela na qual o cursor está posicionado
btn-secundary	Deixa o botão na cor cinza (botão secundário da tela)
btn-sm	Reduz o tamanho do botão

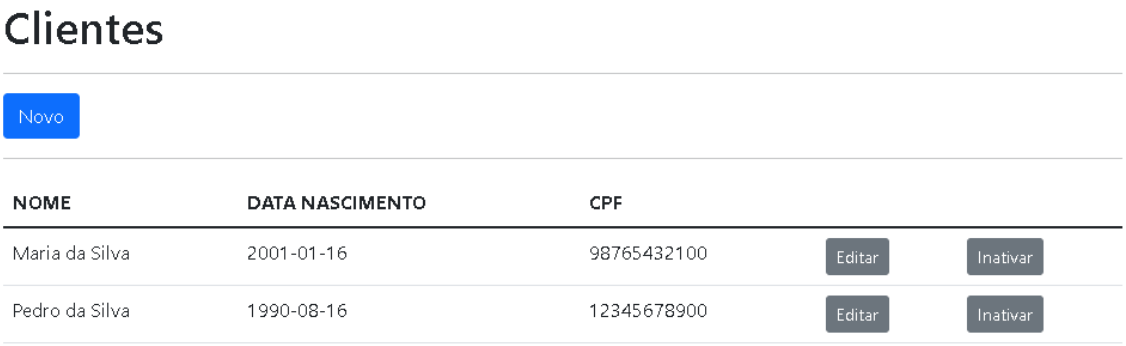
No quadro a seguir, temos o código fonte da tela de gerenciamento de cadastro do cliente utilizando as classes do Bootstrap, que irão resultar na interface vista na figura 15.

Quadro 16 – Código fonte da tela de gerenciamento estilizado com Bootstrap

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Sistema de Estoque</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.c
ss" rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
</head>
<body>
  <div class="container">
    <h1>Clientes</h1>
    <hr>
    <div>
      <a class="btn btn-primary" th:href="@{/clientes/novo}">Novo</a>
    </div>
    <hr>
```

```
    <table class="table table-hover">
      <thead>
        <tr>
          <td><b>NOME</b></td>
          <td><b>DATA NASCIMENTO</b></td>
          <td><b>CPF</b></td>
          <td></td>
          <td></td>
        </tr>
      </thead>
      <tbody>
        <tr th:each="cliente : ${clientes}">
          <td th:text="${cliente.nome}"></td>
          <td th:text="${cliente.dataDeNascimento}"></td>
          <td th:text="${cliente.cpf}"></td>
          <td>
            <a class="btn btn-sm btn-secondary"
              th:href="@{/clientes/edita/{id}(id=${cliente.id})}">Editar</a>
          </td>
          <td>
          <a th:href="@{/clientes/ativa/{id}(id=${cliente.id})}"
             class="btn btn-sm btn-secondary"
             th:if="${cliente.ativo == false}" >Ativar</a>
          <a th:href="@{/clientes/inativa/{id}(id=${cliente.id})}"
             class="btn btn-sm btn-secondary"
             th:unless="${cliente.ativo == false}" >Inativar</a>
          </td>
        </tr>
      </tbody>
    </table>
  </div>
</body>
</html>
```

Figura 15 – Interface da tela de gerenciamento de cadastro do cliente



FINALIZANDO

Nesta aula, aprendemos a desenvolver os controladores da aplicação, componente responsável por atender às requisições efetuadas pelo usuário. Além disso, desenvolvemos também a tela de gerenciamento de cadastro do cliente, fornecendo recursos para que o usuário possa realizar um cadastro, editar os dados cadastrais de um determinado cliente, além de ativar/inativar um cadastro específico. Ademais, adicionamos as bibliotecas do Bootstrap às páginas HTML com o intuito de torná-las mais atrativas comercialmente e recursivas, fazendo com que a interface gráfica da aplicação se adapte ao dispositivo no qual ela está sendo executada.

REFERÊNCIAS

THYMELEAF. **Tutorial**: Using Thymeleaf. Disponível em: <https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html>. Acesso em: 24 mar. 2022.

BOOTSTRAP. **Introduction**. Disponível em: <https://getbootstrap.com/docs/5.1/getting-started/introduction/>. Acesso em: 24 mar. 2022.