PROGRAMAÇÃO (BACKEND)

Tags: crud, Eclipse, Java, jpa, matera, MySQL, postgresql, Spring, thyemeleaf

CRUD com Spring e Thymeleaf

SPRING BOOT

Ao passo que a configuração de infraestrutura e definição da arquitetura estão entre as etapas que mais consomem recursos durante o desenvolvimento de aplicações web com Java, emergiram iniciativas buscando mitigar questões dessa ordem. Nesse sentido, o Spring Boot provê o *bootstrap* e configuração automática, sempre que possível, de aplicações Spring *standalone* rodando em Apache Tomcat embarcado, além de suportar integração com uma gama diversa de ferramentas de mercado.

Uma representação simplista de um cadastro de postagens em blog será usada como exemplo. Para inciar um projeto Spring Boot, acesse a página do **Spring Initializr (http://start.spring.io),** defina as configurações de acordo com a Figura 1 e clique em "Generate Project".

(http://www.matera.com/br/wp-content/uploads/2017/02/1.jpg)
Figura 1 – Configuração do Spring Initializr

Extraia o arquivo zip gerado e importe para o **Eclipse** pelo menu "*File > Import... > Maven > Existing Maven Projects*". A estrutura base do projeto, em suma, contém:

- **BlogApplication:** Classe usada para inicialização da aplicação.
- **BlogApplicationTests:** Classe base para codificação dos testes .
- **pom.xml:** Definições da *build.*
- **application.properties (https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html):** Aonde são realizadas configurações diversas.

SPRING DATA JPA

O **Spring Data** (http://projects.spring.io/spring-data/) é um conjunto de projetos que objetam simplificar e uniformizar o acesso a bancos de dados relacionais, não relacionais e soluções congêneres, sendo o **Spring Data JPA** (https://docs.spring.io/spring-data/jpa/docs/current/reference/html/) responsável pelos primeiros. Esse artigo usa o **PostgreSQL** em seus exemplos, contudo outras

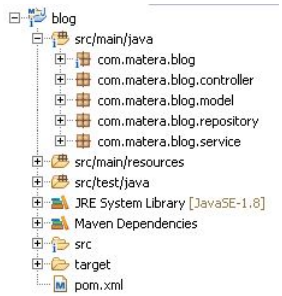
soluções, como o **MySQL**, podem ser usadas bastando que o driver **JDBC** adequado seja declarado como dependência no arquivo *pom.xml* e a configuração do *datasource* esteja em conformidade com a instância desejada. O Spring Data JPA, por default, usa do **Hibernate** no seu plano de fundo, mas possibilita o uso de outras soluções ORM, como o **EclipseLink**.

Para configurar um *datasource* básico acessando o PostgreSQL, abra o arquivo *application.properties* localizado no diretório *src/main/resources* do projeto e altere as configurações de acordo com o seu banco de dados.

```
#datasource
spring.datasource.url = jdbc:postgresql://localhost:5432/blog
spring.datasource.username = postgres
spring.datasource.password = admin
spring.datasource.driver-class-name = org.postgresql.Driver
spring.datasource.validationQuery = SELECT 1

#jpa/hibernate
spring.jpa.database=POSTGRESQL
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
```

Execute o método main da classe "BlogApplication" para subir a aplicação e validar a conexão com o banco de dados. Se tudo estiver certo, ao final, uma mensagem informado que a aplicação foi iniciada será exibida no console do Eclipse.

Como a codificação será feita sob o **Spring MVC**, é coerente que sejam definidas camadas. Essas podem estar separadas em projetos ou módulos, de modo a restringir o acesso indevido entre as mesmas. Não obstante, o uso de *packages* será empregado para facilitar o entendimento. Crie novas *packages* conforme a Figura 2.



(http://www.matera.com/br/wp-content/uploads/2017/02/4.jpg)
Figura 2 – Camadas da Aplicação

Model – Aonde ficam as entidades que representam tabelas de bancos de dados e tipos.
Repository – Repositórios do Spring Data JPA responsáveis pelo acesso a dados.
Service – Classes de serviço e/ou negócio.
Controller – Camada intermediária entre a view e a camada de serviços.

Adicione uma nova classe chamada "Post" na camada **Model** e não se esqueça de gerar os métodos *getters e setters*.

```
package com.matera.blog.model;

import java.io.Serializable;
import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.validation.constraints.NotNull;
import org.hibernate.validator.constraints.NotBlank;
import org.springframework.format.annotation.DateTimeFormat;

@Entity(name = "tbl_post")
public class Post implements Serializable {

	private static final long serialVersionUID = 1L;

	@Id
	@SequenceGenerator(name = "post_seq", sequenceName = "post_seq")
	@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "post_seq")
	private Long id;

	@Column(nullable = false, length = 50)
	@NotBlank(message = "Autor é uma informação obrigatória.")
	private String autor;

	@Column(nullable = false, length = 150)
	@NotBlank(message = "Título é uma informação obrigatória.")
	private String titulo;

	@Column(nullable = false, length = 2000)
	@NotBlank(message = "Texto é uma informação obrigatória.")
	private String texto;

	@Column(nullable = false)
	@Temporal(TemporalType.DATE)
	@DateTimeFormat(pattern = "yyyy-MM-dd")
	@NotNull(message = "Data é uma informação obrigatória.")
	private Date data;
}
```

Essa classe representa a tabela de banco de dados “tbl_post”, observe que as anotações do JPA definem características das colunas, enquanto anotações que recebem a string *message* tem objetivo de validar o preenchimento desses atributos.

Na camada **Repository**, crie uma **interface** com nome de “PostRepository” que estenda “JpaRepository”. Repositórios tem como finalidade facilitar as operações entre tabelas de bancos de dados e objetos Java. A interface deve estar anotada com @Repository. Cumpre informar que anotações nesse sentido são intituladas *stereotypes* e tem a finalidade de registrar e instanciar uma classe automaticamente no contexto do Spring. Genericamente, a anotação @Component tem o mesmo efeito, contudo, para semântica aprimorada, é recomendado anotar cada classe em concordância a camada (http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/package-summary.html) pertencente.

```
package com.matera.blog.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.matera.blog.model.Post;

@Repository
public interface PostRepository extends JpaRepository<Post, Long> { }
```

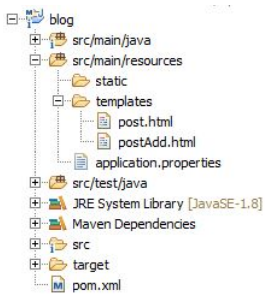
Feito isso, crie a classe “PostService” anotada com @Service na camada **Service** e use da anotação @Autowired para realizar a injeção do repositório criado anteriormente. O propósito de @Autowired é injtetar uma instância da classe declarada como atributo no escopo da classe trabalhada sem que a segunda saiba como instanciar a primeira, reduzindo o acoplamento. Pensando em um CRUD, a classe de serviço deve fornecer as opções de recuperação, inserção, edição e exclusão.

```
@Service
public class PostService {

	@Autowired
	private PostRepository repository;

	public List<Post> findAll() {
		return repository.findAll();
	}

	public Post findOne(Long id) {
		return repository.findOne(id);
	}

	public Post save(Post post) {
		return repository.saveAndFlush(post);
	}

	public void delete(Long id) {
		repository.delete(id);
	}
}
```

THYMELEAF

Trata-se de um *template engine* para Java e uma alternativa ao JSP. Os templates são escritos, em sua maioria, com código **HTML5** e tem boa integração com o **Spring Framework**, além da capacidade de processar código **CSS** e **JavaScript**.

Para criação das telas, adicione dois arquivos no diretório preexistente *src/main/resources/templates* com os nomes "post.html" e "postAdd.html" conforme Figura 3.



(http://www.matera.com/br/wp-content/uploads/2017/02/5.jpg)
Figura 3 – Diretório dos templates

Na camada Controller crie uma nova classe anotada de forma homônima e denominada "PostController" e responsável por intermediar a comunicação entre as views do **Thyemeleaf (http://www.thymeleaf.org/documentation.html)** e a camada de serviços. Essa classe recebe "PostService" injetada com @Autowired e define as URLs de acesso aos recursos da aplicação. Repare que o construtor de ModelAndView recebe o nome dos templates criados anteriormente e transfere de forma bilateral dados entre view e controller.

```
package com.matera.blog.controller;

import javax.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.servlet.ModelAndView;
import com.matera.blog.model.Post;
import com.matera.blog.service.PostService;

@Controller
public class PostController {

	@Autowired
	private PostService service;

	@GetMapping("/")
	public ModelAndView findAll() {

		ModelAndView mv = new ModelAndView("/post");
		mv.addObject("posts", service.findAll());

		return mv;
	}

	@GetMapping("/add")
	public ModelAndView add(Post post) {

		ModelAndView mv = new ModelAndView("/postAdd");
		mv.addObject("post", post);

		return mv;
	}

	@GetMapping("/edit/{id}")
	public ModelAndView edit(@PathVariable("id") Long id) {

		return add(service.findOne(id));
	}

	@GetMapping("/delete/{id}")
	public ModelAndView delete(@PathVariable("id") Long id) {

		service.delete(id);

		return findAll();
	}

	@PostMapping("/save")
	public ModelAndView save(@Valid Post post, BindingResult result) {

		if(result.hasErrors()) {
			return add(post);
		}

		service.save(post);

		return findAll();
	}

}
```

Codifique as telas e observe que as diretivas que o Thymeleaf adiciona às estruturas do HTML são iniciadas pelo prefixo **th**, como os **comandos** comentados na sequência:

- **th:each**: Percorre uma coleção de objetos enviada pelo *controller*.
- **th:if**: Habilita e desabilita controles do HTML de acordo com a condição recebida.
- **th:object**: Define o objeto que o *controller* irá receber e enviar por meio de um formulário.
- **th:field**: Faz *bind* dos atributos do objeto do formulário com os *inputs*.

post.html

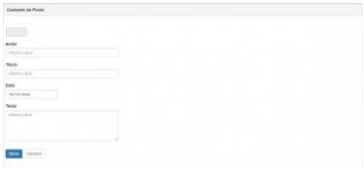
```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml (http://www.w3.org/1999/xhtml)" xmlns:th="http://www.thymele
af.org (http://www.thymeleaf.org)">
<head>
        <meta charset="UTF-8" />
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <meta name="viewport" content="width=device-width" />
        <title>Posts</title>
        <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css (https://max
cdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css)" rel="stylesheet"></link>
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js (https://maxc
dn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js)"></script>
</head>
<body>
        <div class="panel panel-default">
                <div class="panel-heading">
                        <strong>Posts</strong>
                </div>
                <div class="panel-body">
                        <div class="table-responsive">
                                <table class="table table-sm table-striped table-hover table-bordered"
>
                                        <thead>
                                                <tr>
                                                        <th>ID</th>
                                                        <th>Autor</th>
                                                        <th>T&iacute;tulo</th>
                                                        <th>Data</th>
                                                        <th></th>
                                                </tr>
                                        </thead>
                                        <tbody>
                                                <tr th:each="post : ${posts}">
                                                        <td th:text="${post.id}"></td>
                                                        <td th:text="${post.autor}"></td>
                                                        <td th:text="${post.titulo}"></td>
                                                        <td th:text="${#dates.format(post.data, 'dd/MM
/yyyy')}"></td>
                                                        <td>
                                                                <div class="btn-group pull-right">
                                                                        <a class="btn btn-sm btn-prima
ry" th:href="@{/edit/{id}(id=${post.id})}" >Editar</a>
                                                                        <a class="delete btn btn-sm bt
n-danger" th:href="@{/delete/{id}(id=${post.id})}">Excluir</a>
                                                                </div>
                                                        </td>
                                                </tr>
                                        </tbody>
                                </table>
                        </div>
                </div>
                <div class="panel-footer">
                        <a class="btn btn-sm btn-success" th:href="@{/add/}" >Adicionar</a>
                </div>
        </div>
</body>
</html>
```

postAdd.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml (http://www.w3.org/1999/xhtml)" xmlns:th="http://www.thymele
af.org (http://www.thymeleaf.org)">
<head>
        <meta charset="UTF-8" />
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <meta name="viewport" content="width=device-width" />
        <title>Cadastro de Posts</title>
        <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css (https://max
cdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css)" rel="stylesheet"></link>
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js (https://maxc
dn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js)"></script>
</head>
<body>
        <div class="panel panel-default">
                <div class="panel-heading">
                        <strong>Cadastro de Posts</strong>
                </div>
                <div class="panel-body">
                        <form class="form-horizontal" th:object="${post}" th:action="@{/save}" method=
"POST" style="margin: 10px">
                                <div class="form-group">
                                        <fieldset>
                                                <div class="form-group row">
                                                        <div class="alert alert-danger" th:if="${#fiel
ds.hasAnyErrors()}">
                                                                <div th:each="detailedError : ${#field
s.detailedErrors()}">
                                                                        <span th:text="${detailedError
.message}"></span>
                                                                </div>
                                                        </div>
                                                </div>
                                                <div class="form-group row">
                                                    <div class="col-md-1">
                                                        <input type="text" class="form-control input-s
m" id="id" th:field="*{id}" readOnly="readonly"/>
                                                    </div>
                                                 </div>
                                                <div class="form-group row">

                                                        <div class="col-md-4" th:classappend="${#field
s.hasErrors('autor')}? 'has-error'">
                                                            <label>Autor</label>
                                                            <input type="text" class="form-control inp
ut-sm" th:field="*{autor}" autofocus="autofocus" placeholder="Informe o autor" maxlength="50"/>
                                                        </div>
                                                </div>
                                                <div class="form-group row">
                                                        <div class="col-md-4" th:classappend="${#field
s.hasErrors('titulo')}? 'has-error'">
                                                                <label>T&iacute;tulo</label>
                                                                <input type="text" class="form-control
 input-sm" th:field="*{titulo}" maxlength="150" placeholder="Informe o título"/>
                                                        </div>
                                                </div>
                                                <div class="form-group row">
                                                        <div class="col-md-2" th:classappend="${#field
s.hasErrors('data')}? 'has-error'">
                                                                <label>Data</label>
                                                                <input type="date" class="form-control
 input-sm" th:field="*{data}"/>
                                                        </div>
                                                </div>
                                                <div class="form-group row">
                                                        <div class="col-md-4" th:classappend="${#field
s.hasErrors('texto')}? 'has-error'">
                                                                <label>Texto</label>
                                                                <textarea class="form-control input-sm
" th:field="*{texto}" cols="5" rows="5" placeholder="Informe o texto"></textarea>
                                                        </div>
                                                </div>
                                        </fieldset>
                                </div>
                                <div class="form-group row">
                                        <button type="submit" class="btn btn-sm btn-primary">Salvar</b
utton>
                                        <a th:href="@{/}" class="btn btn-sm btn-default">Cancelar</a>
                                </div>
                        </form>
                </div>
        </div>
</body>
</html>
```

Ao término, execute novamente a aplicação pelo método main da classe "BlogApplication.java" e acesse o endereço http://localhost:8080 (http://localhost:8080) pelo navegador. O resultado final pode ser observado nas Figuras 4 e 5.

(http://www.matera.com/br/wp-content/uploads/2017/02/6.jpg)
Figura 4 – Tela Inicial do CRUD



(http://www.matera.com/br/wp-content/uploads/2017/02/7.jpg)
Figura 5 – Tela de Edição do CRUD

CONCLUSÃO

Quando combinados, o **Spring** e o **Thymeleaf,** são capazes de reduzir a complexidade de tópicos inerentes à configuração, acesso a dados e criação de páginas web, possibilitando o desenvolvimento produtivo e focado no negócio. O código completo e comentado desse artigo está disponível no **GitHub (https://github.com/materasystems/spring-thymeleaf-tutorial)**.

REFERÊNCIAS

http://docs.spring.io/spring-boot/docs/current/reference/html/ (http://docs.spring.io/spring-boot/docs/current/reference/html/)

https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ (https://docs.spring.io/spring-data/jpa/docs/current/reference/html/)

https://docs.spring.io/spring/docs/current/spring-framework-reference/html/ (https://docs.spring.io/spring/docs/current/spring-framework-reference/html/)

http://www.thymeleaf.org/documentation.html (http://www.thymeleaf.org/documentation.html)

https://www.postgresql.org/docs/ (https://www.postgresql.org/docs/)

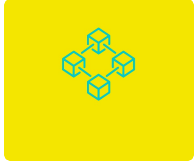
Por STÉFFANO BONAIVA BATISTA

Analista de Sistemas com quase uma década de experiência e interesse em Java e Infraestrutura.

Postado em: 16 de fevereiro de 2017

Confira outros artigos do nosso blog

Voltar para o blog (/blog)



Aprendendo Spring Framework

Sistemas poliglotas

27 de junho de 2017

Como funciona a Blockchain?

11 de junho de 2018
Alan Cesar Elias

Tutorial: Carga de dados com o SQL Loader

17 de abril de 2018
Ricardo Silveira

09 de abril de 2018
Hivison Moura

Ronaldo Chicareli

Deixe seu comentário

TAMBÉM NO **BLOG MATERA**

Carreira (/career) **Conteúdos (/blog)** **Suporte (https://cas2.matera.com/)** **Contato (/contact)**

(https://www.linkedin.com/company/matera)

(http://www.mat

be.com/channel/UCMgX8Pjw7_PEJMkaJVGsKqw)

(https://www.instagram.com/matera.oficial)

(https://www.facebook.com/matera.oficial/)