Olá!

Nesta videoaula, você estudará acerca da API REST e dos métodos HTTP. Além disso, você ainda aprenderá a adequar o controller FilmeController ao padrão REST.

O termo API REST é bastante popular no contexto dos desenvolvedores, principalmente, quando se estuda a integração de dados, comunicação entre aplicações e disponibilização de recursos.

Para que você faça uso de uma API REST, é preciso realizar requisições através do protocolo HTTP. Estas são compostas por três pilares: a URL, endereço dos recursos disponibilizados; o método de requisição, o que deve ser realizado com os dados enviados; e o corpo, os dados propriamente ditos a serem processados.

Diante disso, é necessário que, primeiramente, você conheça os principais métodos do HTTP e onde eles se aplicam.

Bem, o protocolo HTTP possui vários métodos de requisição, que muitas vezes também são chamados de verbos, por realizarem alguma ação em relação aos dados enviados ao servidor. Os principais são GET, POST, PUT e DELETE, mas há outros além desses. Contudo, não serão estudados nesta videoaula. Portanto, para cada processamento realizado, por meio do protocolo HTTP, existe um código de status de respostas associadas. Isso é uma forma de identificar se foi ou não bem sucedida. Então, os métodos GET, POST, PUT e DELETE estão diretamente ligados ao CRUD, acrônimo para Create (criação), Read (consulta), Update (atualização) e Delete (deleção).

Para a troca de dados, as API REST utilizam um formato de representação de dados. Dentre os principais, têm-se o XML, JSON e YAML. Cada um com suas estruturas e características.

O método GET é responsável por consultar ou recuperar dados. Em caso de sucesso, ele retorna às informações requisitadas em um dos formatos de representação de dados citados anteriormente (XML, JSON ou YAML) e, juntamente a esses dados, retorna o status de resposta 200 do HTTP, indicando que a requisição foi bem sucedida.

Em seguida, o método POST é encarregado de criar recursos. Assim, os dados são enviados dentro do corpo do HTTP e, em caso de sucesso, pode ou não retornar às informações persistidas, ou criadas, em um dos formatos de representação de dados citados anteriormente (XML, JSON ou YAML) e, juntamente a estes dados, retorna o status de resposta 201 do HTTP, indicando que um novo recurso foi criado. Agora, o método PUT tem o objetivo de atualizar recursos existentes, por isso seu uso é semelhante ao POST. Porém, a principal diferença entre eles é o status de resposta do HTTP que, no caso do PUT, é o 200, informando que a requisição foi bem sucedida.

Por último, tem-se o método DELETE. Este tem o propósito de remover recursos existentes. Em caso de sucesso, pode ou não retornar às informações deletadas em um dos formatos de representação de dados citados anteriormente (XML, JSON ou YAML) e, juntamente a esses dados, retorna o status de resposta 200 do HTTP, indicando que a requisição foi bem sucedida.

Como você estudou, até aqui, sobre cada um dos métodos explicados anteriormente, chegou o momento de aplicá-los ao seu projeto. Para isso, você irá reformular o controller FilmeController, de acordo com os padrões REST. Assim, abra o Eclipse e, no projeto de catálogo de filmes, localize a classe FilmeController no pacote br.com.lead.controller (br ponto com ponto lead ponto controller). Depois, você iniciará pela refatoração do método persistirFilme do controller FilmeController, que é responsável por cadastrar novos filmes. Para isso, você deve alterar o método de requisição de GET para POST da anotação @RequestMapping (arroba request mapping), que se encontra antes da declaração do método. Ah, lembre-se de que o método GET é utilizado para consultar informações, enquanto o POST cria recursos. Portanto, o código, a seguir, apresenta o método persistirFilme da classe FilmeController.java (filme controller ponto java) com a modificação realizada.

Código

```
@RequestMapping(value  = "/persistir-filme", method  = RequestMethod.POST)
public String persistirFilme(@RequestParam String nome, @RequestParam String genero,
@RequestParam String ano,  @RequestParam String nomeAutor,  @RequestParam String
dataNascimentoAutor)  {
        Autor  autor = new Autor( );
        autor.setDataNascimento(LocalDate.parse(dataNascimentoAutor));
        autor.setNome(nomeAutor);
        Filme filme = new Filme(nome, genero, Integer.valueOf(ano));
        filme.setAutor(autor);
        EntityManager entityManager = JPAUtil.getEntityManager();
        entityManager.getTransaction().begin();
        entityManager.presist(autor);
        entityManager.persist(filme);
        entityManager.getTransaction().commit();
        entityManager.close();
        return "persistir-filme-view";
}
```

Ainda no método persistirFilme, da classe FilmeController.java (filme controller ponto java), você deve adicionar a anotação (ou diretiva) @ReponseBody (arroba response body) logo abaixo de @RequestMapping (arroba request mapping). Isso vai indicar que a resposta da requisição será enviada no corpo do HTTP. Em seguida, remova todos os parâmetros do método persistirFilme e adicione somente um, de nome filme do tipo Filme, OK?

Nesse momento, você deve estar se perguntando, “ah, como vou enviar esses dados?” A resposta é simples, através de um JSON (JavaScript Object Notation)!

O JSON é um modelo de representação de dados no formato atributo e valor, utilizado para a troca de informações entre aplicações. Na prática, você tem um objeto em formato JavaScript com os dados que definem seu estado e pode ser transmitido via web em formato de texto.

Portanto, cada linguagem de programação possui mecanismos que analisam os dados, em formato JSON, e os converte para objetos, seja ela, Java, JavaScript, Dart ou GoLang. Essa operação é chamada de data-binding (data binding).

Por padrão, o Spring MVC não possui essa funcionalidade, contudo é possível que você a adicione, utilizando as três dependências: jackson-core (jackson core), jackson-databind (jackson databind) e jackson-datatype-jsr310 (jackson datatype jsr310), aplicadas ao arquivo pom.xml (pom ponto xml). Elas serão as encarregadas pela conversão dos dados. A última dependência trata, especificamente, dos novos tipos de dados implementados no Java 8.

Para isso, acesse o repositório central do maven a partir da URL https://mvnrepository.com/ (https dois pontos barra barra mvn repository ponto com barra) e, no campo de pesquisa, digite jackson-core (jackson core). Em seguida, uma lista de opções de dependências vai ser apresentada a você. Então, escolha a opção jackson-core (jackson core) na versão 2.10.2 (dois ponto dez ponto dois), copie o conteúdo da aba Maven e o adicione entre as tags dependencies do arquivo pom.xml (pom ponto xml) do projeto de catálogo de filmes.

Faça o mesmo procedimento para as demais dependências jackson-databind (jackson databind) e jackson-datatype-jsr310 (jackson datatype jsr310), utilizando o nome da dependência para realizar a pesquisa; e escolha a alternativa que a disponibiliza na versão 2.10.2 (dois ponto dez ponto dois).

O trecho, a seguir, apresenta o conteúdo do arquivo pom.xml do projeto com as novas dependências jackson-core (jackson core), jackson-databind (jackson databind) e jackson-datatype-jsr310 (jackson datatype jsr310).

Código

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>CatalogoDeFilmes</groupId>
	<artifactId>CatalogoDeFilmes</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>war</packaging>

	<dependencies>
		<!-- HIBERNATE -->
		<dependency>
		   <groupId>org.hibernate</groupId>
		   <artifactId>hibernate-core</artifactId>
		   <version>5.2.6.Final</version>
		</dependency>
		<dependency>
		   <groupId>org.hibernate</groupId>
		   <artifactId>hibernate-entitymanager</artifactId>
		   <version>5.2.6.Final</version>
		</dependency>

		<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>8.0.13</version>
		</dependency>
```

```
        <!-- SPRING MVC -->
        <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.2.6.RELEASE</version>
        </dependency>

        <!-- JACKSON -->
        <dependency>
           <groupId>com.fasterxml.jackson.core</groupId>
           <artifactId>jackson-core</artifactId>
           <version>2.10.2</version>
        </dependency>
        <dependency>
           <groupId>com.fasterxml.jackson.core</groupId>
           <artifactId>jackson-databind</artifactId>
           <version>2.10.2</version>
        </dependency>
        <dependency>
           <groupId>com.fasterxml.jackson.datatype</groupId>
           <artifactId>jackson-datatype-jsr310</artifactId>
           <version>2.10.2</version>
        </dependency>
</dependencies>

<build>
        <sourceDirectory>src</sourceDirectory>
        <plugins>
                <plugin>
                        <artifactId>maven-compiler-plugin</artifactId>
                        <version>3.8.0</version>
                        <configuration>
                                <source>1.8</source>
```

```
                        <target>1.8</target>
                    </configuration>
                </plugin>
                <plugin>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>3.2.3</version>
                    <configuration>
                        <warSourceDirectory>WebContent</warSourceDirectory>
                    </configuration>
                </plugin>
            </plugins>
        </build>
</project>
```

Até aqui, você viu como aplicar as dependências jackson-core (jackson core), jackson-databind (jackson databind) e jackson-datatype-jsr310 (jackson datatype jsr310) ao seu arquivo pom.xml (pom ponto xml). Observe, então, que, para utilizar as bibliotecas dessas dependências, não é necessário configurá-las previamente**.** Isso ocorre porque o Spring MVC trabalha em parceria com o Jackson, convertendo dados em JSON para novas instâncias em Java.

Assim, é necessário ajustar a assinatura do método persistirFilme da classe FilmeController.java (filme controller ponto java) para receber, por meio do corpo da requisição HTTP, os dados que irão formar seu objeto a ser cadastrado.

No parâmetro filme, adicione a anotação @RequestBody (arroba request body). Isso vai informar ao Spring que o conteúdo que forma o objeto virá no corpo da requisição HTTP, e que o JSON será utilizado para criar uma instância de filme. Dessa maneira, o código, a seguir, apresenta o método persistirFilme da classe FilmeController.java (filme controller ponto java) com o seu único parâmetro que é filme do tipo Filme.

Código

```
@RequestMapping(value  = "/persistir-filme", method  = RequestMethod.POST)
public String persistirFilme(@RequestParam String nome, @RequestParam String genero,
```

```
@RequestParam String ano, @RequestParam String nomeAutor, @RequestParam String dataNascimentoAutor) {
        Autor autor = new Autor( );
        autor.setDataNascimento(LocalDate.parse(dataNascimentoAutor));
        autor.setNome(nomeAutor);
        Filme filme = new Filme(nome, genero, Integer.valueOf(ano));
        filme.setAutor(autor);
        EntityManager entityManager = JPAUtil.getEntityManager();
        entityManager.getTransaction().begin();
        entityManager.presist(autor);
        entityManager.persist(filme);
        entityManager.getTransaction().commit();
        entityManager.close();
        return "persistir-filme-view";
}
```

Por fim, você irá pedir ao Spring MVC que retorne ao requisitante a própria instância do filme em formato JSON e não mais o redirecione a view persistir-filme-view.jsp persistir traço filme traço view ponto jsp, que se encontra no diretório WebContent/WEB-INF/views (webcontent barra web inf barra views). Para que isso aconteça, você deve modificar o retorno do método persistirFilme da classe FilmeController.java (filme controller ponto java) para o tipo Filme. Em seguida, na anotação, ou na diretiva, o @RequestMapping (arroba request mapping) deve incluir outra propriedade chamada produces com o valor application/json (application barra json). Isso vai indicar, ao Spring MVC, que ele deve retornar um objeto no formato JSON.

O código, a seguir, apresenta a versão final do método persistirFilme e da classe FilmeController.java (filme controller ponto java), após as modificações para se adequar ao padrão REST.

Código

```
@RequestMapping(value = "/persistir-filme", method = RequestMethod.POST, produces = "application/json")
        @ResponseBody
        public Filme persistirFilme(@RequestBody Filme filme) {
```

```
        EntityManager entityManager = JPAUtil.getEntityManager();

        entityManager.getTransaction().begin();
        entityManager.persist(filme.getAutor());
        entityManager.persist(filme);
        entityManager.getTransaction().commit();
        entityManager.close();

        return filme;
    }
```

Você fará um pequeno ajuste na classe Autor.java (autor ponto java) do pacote br.com.lead.modelo (br ponto com ponto lead ponto modelo). Para isso, inclua a anotação, ou diretiva, @JsonFormat (arroba jsonformat) sobre o atributo dataNascimento da classe Autor e, dentro dos parênteses, atribua à propriedade pattern o valor yyyy-MM-dd (yyyy traço mm traço dd) entre aspas duplas, como uma String. Isso vai indicar ao Jackson que ele deve formatar a data em um determinado padrão de escolha. Em seguida, você deve adotar o formato estabelecido pela ISO 8601, ou seja, yyyy-MM-dd (yyyy traço mm traço dd). Isso é só para melhorar a legibilidade de dados, certo?

Código

```
package br.com.lead.modelo;
import java.time.LocalDate;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import com.fasterxml.jackson.annotation.JsonFormat;

@Entity
public class Autor {
```

```
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer id;
	private String nome;
	@Column(name="data_nascimento")
	@JsonFormat(pattern = "yyyy-MM-dd")
	private LocalDate dataNascimento;
	public Integer getId() {
		return id;
	}
	public void setId(Integer id) {
		this.id = id;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public LocalDate getDataNascimento() {
		return dataNascimento;
	}
	public void setDataNascimento(LocalDate dataNascimento) {
		this.dataNascimento = dataNascimento;
	}
}
```

O próximo passo, na configuração do seu projeto, é adaptar o método consultarFilme da classe FilmeController.java (filme controller ponto java). Portanto, fique tranquilo! Com tudo o que você aprendeu até aqui, essa atividade será simples. Além disso, o método consultarFilme já está de acordo com os princípios REST, ou seja, usa o método GET para consultar os dados dos filmes cadastrados em seu projeto, e, por fim, os repassa a alguém. Sendo que esta última etapa é uma forma de exibir as informações encontradas em uma página HTML. Contudo, o que você deve

modificar, de fato, no método consultarFilme, é o formato dos dados que serão enviados como JSON, e remover o repasse dessas informações a view consultar-filme-view.jsp (consultar traço filme traço view ponto jsp).

No método consultarFilme, você deve adicionar, na anotação, ou na diretiva, @RequestMapping (arroba request mapping), a mesma propriedade incluída no método persistirFilme. Ou seja, produces com o valor application/json (application barra json), para indicar, ao Spring MVC, que ele deve retornar um objeto no formato JSON. Além disso, ele não irá retornar mais um objeto ModelAndView, e sim uma instância do tipo Filme. Então, modifique o tipo de retorno do método persistirFilme para Filme. Por fim, remova a instância da classe ModelAndView para que os dados encontrados não sejam mais encaminhados para view consultar-filme-view.jsp (consultar traço filme traço view ponto jsp), ou seja, o retorno vai ser realizado no corpo do HTTP em formato JSON, utilizando os dados encontrados na consulta.

Código

```
package br.com.lead.controller;
import javax.persistence.EntityManager;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import br.com.lead.modelo.Filme;
import br.com.lead.util.JPAUtil;
@Controller
public class FilmeController {

	@RequestMapping(value = "/persistir-filme", method = RequestMethod.POST, produces = "application/json")
	@ResponseBody
	public Filme persistirFilme(@RequestBody Filme filme) {
		EntityManager entityManager = JPAUtil.getEntityManager();
```

```
        entityManager.getTransaction().begin();
        entityManager.persist(filme.getAutor());
        entityManager.persist(filme);
        entityManager.getTransaction().commit();
        entityManager.close();

        return filme;
    }


    @RequestMapping(value = "/consultar-filme", method = RequestMethod.GET)
    @ResponseBody
    public Filme consultarFilme(@RequestParam Integer id) {
            EntityManager entityManager = JPAUtil.getEntityManager();

        Filme filme = entityManager.find(Filme.class, id);

        return filme;
    }

}
```

Até aqui, você aprendeu sobre API REST e métodos HTTP. Além disso, aplicou esses conhecimentos para adequar a classe FilmeController aos padrões REST. Na próxima videoaula, você irá conhecer o Angular e criará seu primeiro projeto.

Até a próxima aula e bons estudos!

Referências

Códigos de status de respostas HTTP. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>. Acesso em: 16 jun 2020.

Create, read, update and delete. Disponível em: <https://en.wikipedia.org/wiki/Create,_read,_update_and_delete>. Acesso em: 16 jun 2020.

Date format by country. Disponível em: <https://en.wikipedia.org/wiki/Date_format_by_country>. Acesso em: 16 jun 2020.

DIAS, Emílio. **4 Conceitos sobre REST que Qualquer Desenvolvedor Precisa Conhecer**. Disponível em: <https://blog.algaworks.com/4-conceitos-sobre-rest-que-qualquer-desenvolvedor-precisa-conhecer/#:~:text=REST%20%C3%A9%20acr%C3%B4nimo%20de%20Representational,1. https://rockcontent.com/blog/rest-api/>. Acesso em: 16 jun 2020.

Jackson Date. Disponível em: <https://www.baeldung.com/jackson-serialize-dates>. Acesso em: 16 jun 2020.

JSON. Disponível em: <https://pt.wikipedia.org/wiki/JSON>. Acesso em: 16 jun 2020.

MARINHO, Agnaldo. Disponível: < https://medium.com/@agnaldom/introdu%C3%A7%C3%A3o-ao-apis-rest-5c920c8fc58f>. Acesso em: 16 jun 2020.

PARASCHIV, Eugen. **Spring RequestMapping**. Disponível em: <https://www.baeldung.com/spring-requestmapping>. Acesso em: 16 jun 2020.

PIRES, Jacskon. **O que é API?** Disponível em: <https://becode.com.br/o-que-e-api-rest-e-restful/>. Acesso em: 16 jun 2020.

RODRIGUES, Joel. **Testando serviços Web API com Postman**. Disponível em: <http://www.linhadecodigo.com.br/artigo/3712/testando-servicos-web-api-com-postman.aspx>. Acesso em: 16 jun 2020.

Spring MVC Content Negotiation. Disponível em: <https://www.baeldung.com/spring-mvc-content-negotiation-json-xml>. Acesso em: 16 jun 2020.

TILKOV, Stefan. Disponível em: <https://www.infoq.com/br/articles/rest-introduction/>. Acesso em: 16 jun 2020.

Web Services Architecture. Disponível em: <https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>. Acesso em: 16 jun 2020.

What is REST. Disponível em: <https://restfulapi.net/>. Acesso em: 16 jun 2020.