03

Disponibilizando livros JSON

Transcrição

Para que nossos serviços sejam expostos de maneira que uma aplicação externa consiga acessar essa lista é disponibilizar os serviços como recursos de nossa aplicação. Para isso criaremos um novo pacote dentro de `src/main/java` com o nome `br.com.casadocodigo.loja.resources`. E, como queremos que um recurso externo veja nossos livros, criaremos uma nova classe com o nome `LivroResource`. Dentro dela que faremos a leitura dos livros. Podem ser os principais, os últimos lançamentos, o que quisermos.

```
package br.com.casadocodigo.loja.resources;

public class LivroResource {

}
```

Criemos um método para os últimos lançamentos, pois é o que chama mais atenção:

```
package br.com.casadocodigo.loja.resources;

public class LivroResource {

    public void ultimosLancamentosJson() {

    }

}
```

Dentro dele teremos uma Classe que terá a consulta dos últimos lançamentos que já fazemos no DAO. Para isso precisaremos do `LivroDao`:

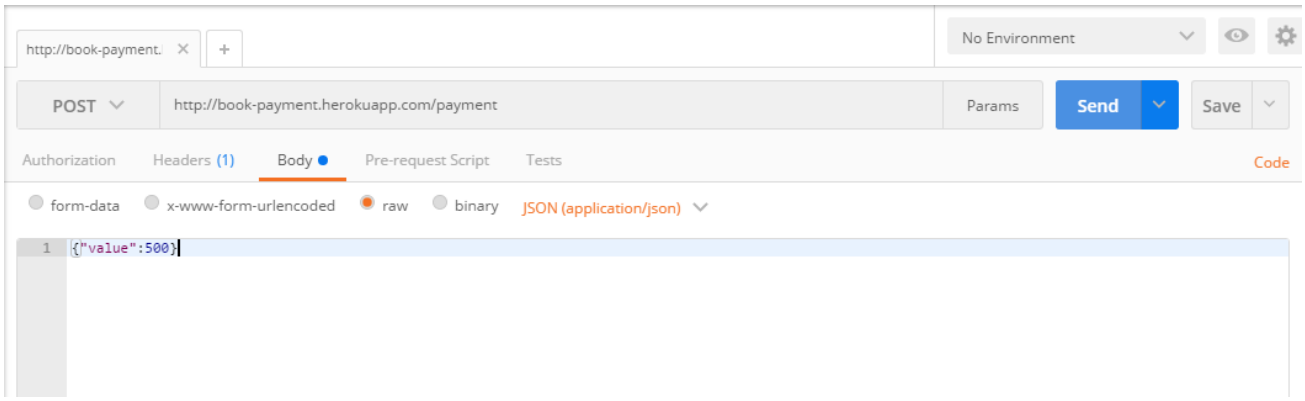
```
package br.com.casadocodigo.loja.resources;

import br.com.casadocodigo.loja.daos.LivroDao;

public class LivroResource {

    private LivroDao dao;

    public void ultimosLancamentosJson() {
        dao.ultimosLancamentos();
    }

}
```

Porém, se pegarmos esses últimos lançamentos e não obter os objetos e lançá-los para fora do método, eles não servirão para nada. Por isso é interessante que retornemos o resultado da lista de livros:

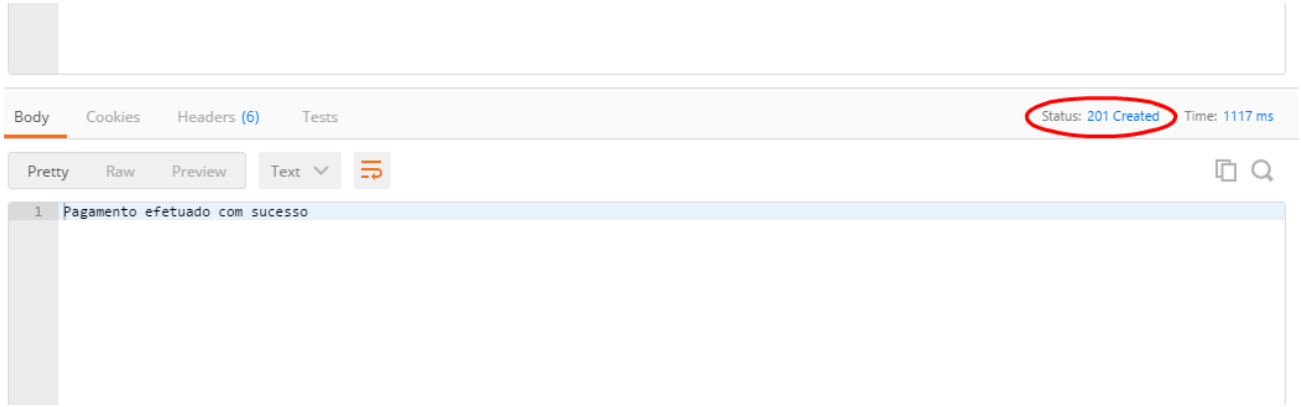
```
package br.com.casadocodigo.loja.resources;

import br.com.casadocodigo.loja.daos.LivroDao;
import br.com.casadocodigo.loja.models.Livro;

public class LivroResource {

    private LivroDao dao;

    public List<Livro> ultimosLancamentosJson() {
        return dao.ultimosLancamentos();
    }

}
```

Com "Ctrl + 1" mudamos o retorno do método. Agora temos um `LivroResource` com o método `ultimosLancamentosJson`, no qual pegaremos todos os últimos lançamentos que vêm do banco de dados e retorná-los como JSON. Mas o que torna essa classe um JSON? Onde estão os mapeamentos? Em que caminho eu chamo?

No `PagamentoGateway.java` temos uma url de pagamento `http://book-payment.herokuapp.com/payment`. Vamos copiá-la para o Postman e faremos um POST onde no corpo - selecionada a opção *raw* - passaremos o valor de 500 (lembrando de indicar que o código é em JSON):



É feito o envio dos dados e retorna o status 201, de criado:



Temos, então, uma requisição. Perceba que temos um endereço, a url `payment`, o método POST e o JSON. Precisamos exatamente das mesmas coisas na nossa classe `LivroResource` com uma url de formato mais completo.

No `LivroResource` precisaremos informar qual o `Path` que o nosso usuário poderá atender, importando-o do pacote `javax.ws.rs` que vem da própria especificação do Java EE para REST, com a *anotation* `@Path`, que por sua vez exige um

nome de caminho:

```
package br.com.casadocodigo.loja.resources;

import javaxx.ws.rs.Path

import br.com.casadocodigo.loja.daos.LivroDao;
import br.com.casadocodigo.loja.models.Livro;

@Path("livros")
public class LivroResource {

    private LivroDao dao;

    public List<Livro> ultimosLancamentosJson() {
        return dao.ultimosLancamentos();
    }

}
```

Esses livros serão carregados em nosso sistema e para isso precisamos do `livroDao` fazendo `@Inject` :

```
@Inject
private LivroDao dao;
```

E precisaremos publicar esse método. Perceba que "livros" é o recurso completo, mas queremos usar algo mais específico, por exemplo, os últimos lançamentos. Logo usamos o `@Path` novamente:

```
@Path("livros")
public class LivroResource {

    @Inject
    private LivroDao dao;

    @Path("lancamentos")
    public List<Livro> ultimosLancamentosJson() {
        return dao.ultimosLancamentos();
    }

}
```

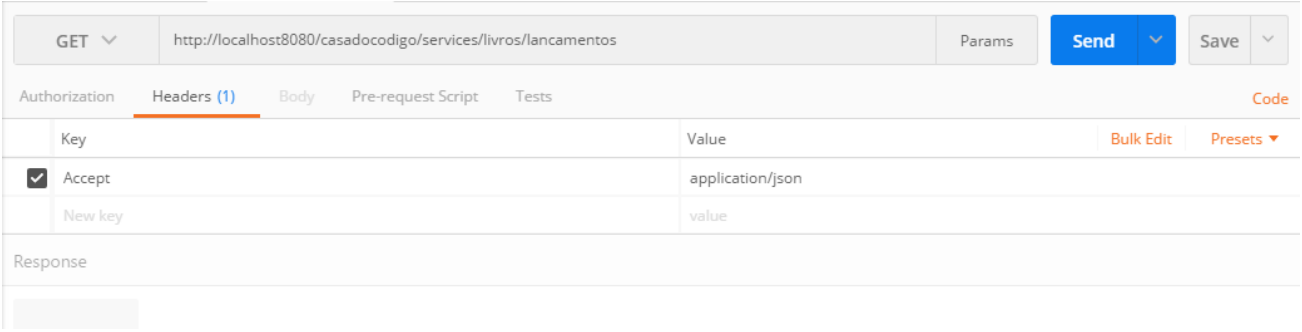
Os últimos lançamentos (`lançamentos`) virão através de uma requisição `/livros/lançamentos` retornando a informação. Para `/livros/lancamentos` ainda temos que definir qual será o método (GET, POST, PUT, etc) HTTP. Os principais métodos HTTP são GET, PUT, POST e DELET, mas existem outros para outros motivos dentro do próprio REST. Então temos a definição que os últimos lançamentos serão chamados por um GET:

```
@GET
@Path("lancamentos")
public List<Livro> ultimosLancamentosJson() {
    return dao.ultimosLancamentos();
}
```

Estamos falando que serão os últimos lançamentos carregados em JSON. Como faremos para que eles sejam retornados em JSON? Precisaremos configurar que queremos produzir (`Produce` do javax.ws.rs) direto de um *array*:

```
@GET
@Path("lancamentos")
@Produces({MediaType.APPLICATION_JSON})
public List<Livro> ultimosLancamentosJson() {
    return dao.ultimosLancamentos();
}
```

Agora temos nossos últimos lançamentos sendo transformados em JSON e sendo retornados para nosso usuário. Vamos testar para ver se, de fato, está tudo funcionando. Com o projeto salvo, reiniciamos o Wildfly e no Postman testaremos via GET a requisição que criamos com os seguintes valores pro *header*:



Fazendo o envio podemos perceber que recebemos uma resposta com o JSON dos últimos lançamentos. Nosa configuração funcionou! E tudo isso está acontecendo devido à preconfiguração do JAX-RS de autoconversão.

Mas será que o JSON é o único formato que serve para configuração? Será que é interessante que nossa loja retorne mais de um tipo de formato como XML?