05

Configurando a JAX-RS

Transcrição

Vamos corrigir algumas coisas no nosso código. A primeira dela é o `contextName` estar vindo como *null*. em `finalizar()` façamos o seguinte:

```
String contextName = facesContext = facesContext.getExternalContext().getRequestContextPath();
//...
response.setHeader("Location", contextName+ ...)
```

Próxima questão: quem irá atender a URL `/service` ? Ela precisa ser atendida por um serviço do tipo REST, o que faremos através de uma configuração da própria API do JAX-RS.

Dentro do pacote ".conf" criaremos a Classe "JaxRsConfiguration" que será extendida de `application` :

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

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/service")
public class JaxRsConfiguration extends Application {

}
```

O `@ApplicationPath` indica o caminho que será atendido. E em `finalizar()` , não precisamos chamar manualmente o 307:

```
response.setStatus(HttpServletResponse.SC_TEMPORARY_REDIRECT);
```

Agora é a vez de resolvermos quem irá atender a URL `/pagamento` . Dessa vez será uma Classe dentro do pacote de serviços, com a seguinte configuração inicial:

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

import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.core.Response;

@Path("/pagamento")
public class PagamentoService {

    @POST
    public Response pagar() {

    }
}
```

Vamos colocar o parâmetro `uuid` como argumento do método. Como ele vem pela URL, é um *query param

```
@POST
public Response pagar(@QueryParam("uuid") String uuid) {
    return null;
}
```

Vamos fazer um teste para ver se está funcionando mandando imprimir o uuid:

```
pagar(@QueryParam("uuid") String uuid) {
    System.out.println(uuid);
    return null;
}
```

Com o servidor no ar, hora de testar! Dessa vez sem o erro 404 e no console poderemos ver a uuid impressa!

Agora está tudo preparado para evoluir o serviço de maneira assíncrona.