04

Gerando um ID único para a compra

Transcrição

Quando clicamos em "Finalizar compra" após inserir os dados do cliente, as informações são enviadas para o banco de dados, a requisição é feita em um sistema externo de pagamento e retornando para nosso sistema. Este é um fluxo ao qual damos o nome de *síncrono*. A ideia aqui será transformá-lo em *assíncrono*.

Acontece que o JSF não suporta esse tipo de fluxo, a não ser que criemos uma nova *Thread utilizando a criação de objeto dinâmico do Java 8:

```
new Thread(() -> {
    String response = pagamentoGateway.pagar(getTotal());
    System.out.println(response);
}).start();
```

Essa solução é bem primitiva então não vamos utilizá-la. Em vez disso, façamos algumas alterações no código:

```
public void finalizar(Compra compra) {
    compra.setItens(this.toJson());
    compraDao.salvar(compra);

    //código de pagamento
}
```

A parte que retiramos do código irá para o `CheckoutBean.java`:

```
@Transactional
public void finalizar() {
    Compra compra = new Compra();
    compra.setUsuario(usuario);
    carrinho.finalizar(compra);
}
```

A mesma coisa para o código de pagamento, podemos retirá-lo do método `finalizar()`. Iremos utilizá-lo em outro local, em outro momento.

O próximo passo é avisar para o JSF que queremos realizar uma requisição, como se fosse um componete externo, mas dentro da própria aplicação. Faremos isso através do *faces-redirect*:

```
public void finalizar() {
    Compra compra = new Compra();
    compra.setUsuario(usuario);
    carrinho.finalizar(compra);

    return "service/pagamento?faces-redirect=true"
}
```

Mas um detalhe: o `service/pagamento` não será uma página do JSF. Não é `pagamento.xhtml` e o JSF só trata chamadas de um outro serviço se este for gerenciado pelo servlet do próprio JSF. Logo, o que queremos fazer de fato, é pegar o `response`, tratá-lo e enviá-lo para outro local. Para isso podemos usar o `facesContext`:

```
@Inject
private FacesContext facesContext;

@Transactional
public void finalizar() {
    //...

    String contextName = facesContext.getExternalContext().getContextName();
}
```

Mas iremos utilizar o JSF para que ele possa nos entregar o `response`:

```
public void finalizar() {
    //...

    String contextName = facesContext.getExternalContext().getContextName();
    HttpServletResponse response = (HttpServletResponse)
    facesContext.getExternalContext().getResponse();
}
```

O próximo passo é dizer para o `response` qual a URL:

```
String contextName = facesContext.getExternalContext().getContextName();
HttpServletResponse response = (HttpServletResponse)
    facesContext.getExternalContext().getResponse();

response.setHeader("Location", "/"+contextName+"/service/pagamento?id="+compra.getId());
```

Para que o navegador não apresente para o usuário essa URL com o Id, podemos utilizar um código:

```
response.setStatus(307);
response.setHeader("Location", "/"+contextName+"/service/pagamento?id="+compra.getId());
```

O código *307* vai nos permitir fazer um *redirect* temporário mantendo o método que foi invocado.

Apesar de não termos o serviço que atende a requisição `service/pagamento` - ficará para um segundo momento - vamos testar essa chamada de serviço. Passando por todo o processo de compra novamente caimos nessa tela:

Perceba que se temos ali na URL `id=7` , provavelmente existirá ids com numeração mais baixa, por exemplo. Ou seja, podemos modificar a URL com outros ids e descobrir informações de pagamentos de outros usuários!

Para resolver este problema de segurança podemos utilizar outro tipo de identificador, o **uuid** (*Universally Unique Identifier*):

```
response.setHeader("Location", "/"+contextName+"/service/pagamento?uuid="+compra.getUuid());
```

Esse identificador único possui uma probabilidade baixíssima de se repetir!

Vamos entrar em `Compra.java` e criar um campo para ele junto com seus getters e setters:

```
private String uuid

//...

public String getUuid() {
    return uuid;
}

public void setUuid(String uuid) {
    this.uuid = uuid;
}
```

O UUID será gerado no momento em que pedimos para que o objeto compra persista. Existe uma Classe "UUID" no Java:

```
private String uuid

public void createUUID() {
    this.uuid = UUID.randomUUID().toString();
}
```

Dessa forma será gerado o UUID de maneira aleatória e transformá-lo em String.

Agora precisamos informar para o Entity Manager que, antes de persistir, é necessário chamar o método que acabamos de criar:

```
@PrePersist
public void createUUID() {
```

```
        this.uuid = UUID.randomUUID().toString();
    }
```

Isso é feito através da API de *callback* do Entity Manager, a qual possui outras diversas *anotations*.

Testando e passando novamento por todo o processo de finalizar compra, temos o UUID na URL:

Inclusive este código também foi salvo no banco de dados, basta fazermos `select * from compra;` no terminal.