01

# Configurando mailsender

### Transcrição

Agora que já fizemos uma boa parte do que devia ser feito em relação à negociação de conteúdo, conversas com sistemas externos, vamos trabalhar com outra, com a qual o sistema já está acostumado.

Logo após o usuário finalizar uma compra ele registra os dados, mas não exite nenhum retorno dizendo a ele que a compra foi um sucesso. Isso existe no sistema, mas o usuário final não recebe nenhuma informação. Seria bom, nessas situações, enviar um e-mail ao usuário.

Em `PagamentoService.java`, logo depois que o pagamento é executado, podemos ver a resposta:

```
public void pagar(@suspended final AsyncResponse ar, @QueryParam("uuid") ...) {
    //...
    executor.submit(() -> {
        try {
            //...
            Response response = Response.seeOther(responseUri).build();
            ar.resume(response);
        }
    });
}
```

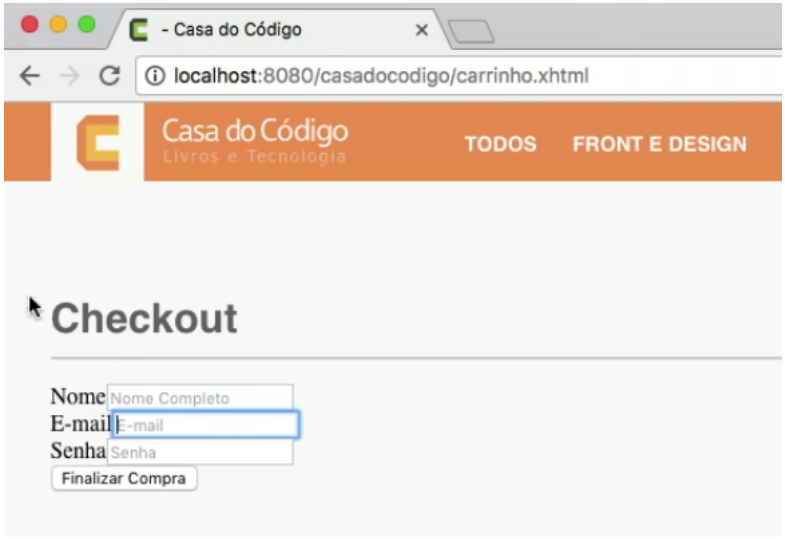
Logo depois disso seria interessante fazer o envio de um e-mail. Lembrando que no `PagamentoService` estamos utilizando o `executor` que faz uma requisição assíncrona e o `AsyncResponse ar` fazendo o processo de `resume`. ele abre uma nova thread no servidor, como discutimos nos cursos anteriores, e ela é continuada a partir do `resume`. Então antes dele e depois da resposta, implementamos o envio de e-mail. É comum usarmos o `mailSender`:

```
Response response = Response.seeOther(responseUri).build();

mailSender.send();

ar.resume(response);
```

Usamos o método `send()` com parâmetros específicos, como quem envia e quem recebe. Quem envia será "compras@casacodigo.com.br", já quem recebe será aquele usuário que preencheu os dados de checkout:




Essas informações chegam para nós como parâmetro, então fazemos:

```
mailSender.send("compras@casacodigo.com.br", compra.getUsuario().getEmail());
```

Também precisamos informar o assunto e o corpo da mensagem que virá o número do pedido:

```
mailSender.send("compras@casacodigo.com.br", compra.getUsuario().getEmail(), "Nova Compra na CD
```

Vamos deixar o corpo da mensagem como variável local:

```
String messageBody = "Sua compra foi realizada com sucesso, com o número de pedido " + compra.g

mailSender.send("compras@casacodigo.com.br", compra.getUsuario().getEmail(), "Nova Compra na CD
```

Se o corpo fosse um texto muito grande teríamos usado o *String Builder*, mas não é o caso, já que é um texto simples.

Podemos fazer o `mailSender` virar um campo ("Ctrl + 1" > "F3"). E temos um *object*:

```
private MailSender mailSender;
```

Só que ainda não temos o objeto `MailSender` , então criaremos a sua Classe dentro do pacote "br.com.casadocodigo.loja.infra":

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

public class MailSender {

}
```

Como queremos injetá-o, colocamos a anotação `@ApplicationScoped`

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

import javax.enterprise.context.ApplicationScoped;

@ApplicationScoped
public class MailSender {

}
```

Voltando para o `PagamentoService.java`, o injetamos:

```
@Inject
private MailSender mailSender
```

Ele é `Application Scoped` porque ele irá durar todo o escopo da aplicação. O CDI vai criar o objeto e o manterá vivo durante todo o contexto da aplicação. O próximo passo é criar o método `send()` nomeando as strings:

```
public class MailSender {

    public void send(String from, String to, String subject, String body) {

    }

}
```

Uma vez que o método está preparado precisamos fazer o código de implementação de e-mail dele:

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

import javax.enterprise.context.ApplicationScoped;
import javax.mail.internet.MimeMessage;

@ApplicationScoped
public class MailSender {

    public void send(String from, String to, String subject, String body) {

        MimeMessage message = new  MimeMessage(session);
    }

}
```

Esse `MimeMessage` será nossa mensagem. Poderíamos estar usando frameworks até mais simples que essa implementação de enviar e-mail, com HTML por exemplo. Não tem problema, mas o nosso foco é a especificação do Java EE, por isso estamos fazendo manualmente. E o `MimeMessage` já é provido pelo Java EE através da especificação de JavaMail. Este nos fornece o objeto `session`, o qual deve ser carregado por um recurso da aplicação:

```
@Resource(mappedName = "java:/jboss/mail")
private Session session;
```

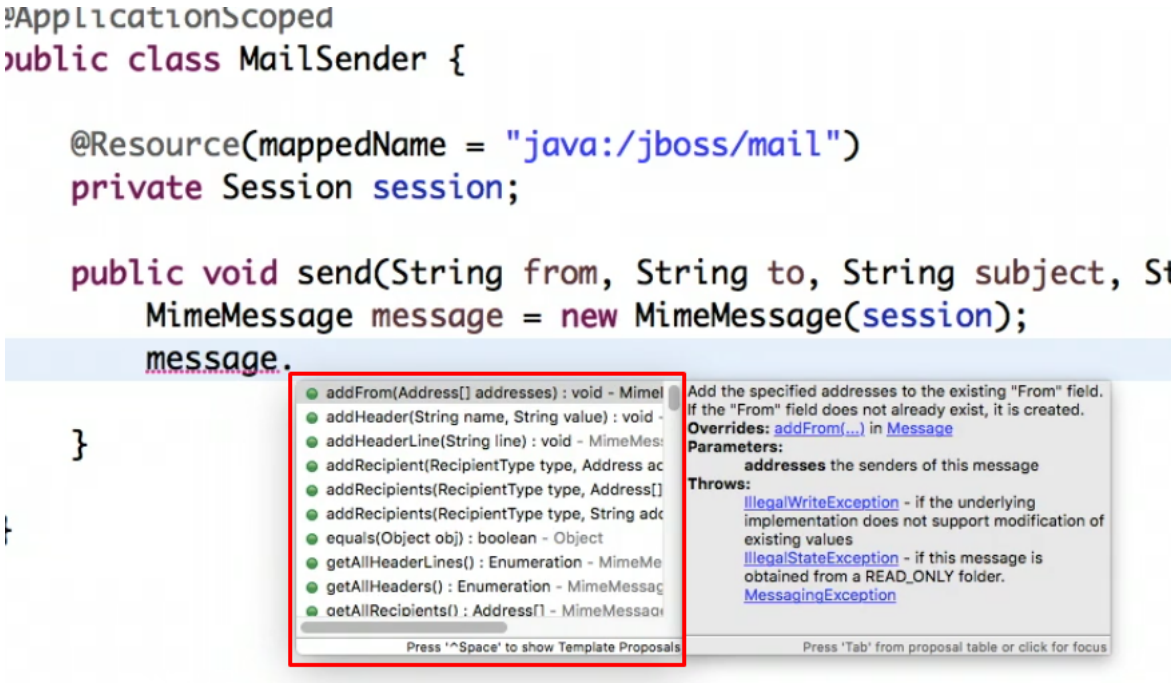
```
public void send(String from, String to, String subject, String body) {

        MimeMessage message = new  MimeMessage(session);
}
```

Vamos importar o `session` como sendo o `javax.mail.Session`:

```
import javax.mail.Session;
```

Todas as importações de e-mail são do Java EE. Agora construiremos a mensagem. Perceba que existem métodos bem simples:




Comecemos pelo `setRecipients()`:

```
public void send(String from, String to, String subject, String body) {

    MimeMessage message = new  MimeMessage(session);
    message.setRecipients(javax.mail.Message.RecipientType.TO, InternetAddres.parse(to));
}
```

Vamos adicionar um *Try/Catch*:

```
public void send(String from, String to, String subject, String body) {

    MimeMessage message = new  MimeMessage(session);
    try {
        message.setRecipients(javax.mail.Message.RecipientType.TO, InternetAddres.parse(to));
    } catch (MessagingException e) {
        e.printStackTrace();
    }
}
```

Já temos para quem queremos enviar. Precisamos informar as demais operações, relembrando que estamos utilizando emplementações da especificação, existem sim formas mais simples utilizando frameworks. Façamos agora o *from* (remetente), o *subject* (assunto) e o *content* (corpo, sendo do tipo HTML):

```
try {
    message.setRecipients(javax.mail.Message.RecipientType.TO, InternetAddres.parse(to));

    message.setFrom(new InternetAddres(from));
    message.setSubject(subject);
    message.setContent(body, "text/html");
    }
```

Temos, assim, nosso e-mail com as informações configuradas. Precisamos fazer o envio propriamente dito, usando o `Transport`:

```
try {
    message.setRecipients(javax.mail.Message.RecipientType.TO, InternetAddres.parse(to));

    message.setFrom(new InternetAddres(from));
    message.setSubject(subject);
    message.setContent(body, "text/html");

    Transport.send(message);
    }
```

Para que tudo isso funcione ainda temos algumas implementações a fazer, porém já configuramos muitas coisas.