

Curso: Spring Boot com Ionic - Estudo de Caso Completo

Capítulo: Serviço de email

Objetivo geral:

- 🕒 Criar um serviço de email
 - Criar uma operação de envio de confirmação de pedido
- 🕒 Implementar o serviço em modo de desenvolvimento e produção
 - Criar o MockEmailService com Logger
 - Criar o SmtEmailService com SMTP do Google
- 🕒 Demonstrar uma implementação flexível e elegante com padrões de projeto (Strategy e Template Method)

Implementando toString de Pedido

ATUALIZAÇÃO

Se você criou o projeto usando Spring Boot versão 2.x.x:

-

- Na classe PedidoService, usar ClienteService ao invés de ClienteRepository

-

Checklist:

- 🕒 Mudar o profile do projeto para test
- 🕒 Implementar toString para ItemPedido e Pedido
- 🕒 Ajustes na operação de insert em PedidoService:
 - Instanciar os objetos relacionados (Cliente e Produto) a partir do banco de dados
 - Instanciar a data do pedido com base na data do sistema

MockEmailService com Logger. Padrões Strategy e Template Method

Checklist:

- ⌚ Adicionar a dependência no POM.XML
- ⌚ Remetente e destinatário default no application.properties
- ⌚ Criar a interface EmailService (padrão Strategy)
- ⌚ Criar a classe abstrata AbstractEmailService
 - Criar método prepareSimpleMailMessageFromPedido
 - Sobrescrever o método sendOrderConfirmationEmail (padrão Template Method)
- ⌚ Implementar o MockEmailService
- ⌚ Em TestConfig, criar um método @Bean EmailService que retorna uma instância de MockEmailService

Implementando SmtplibEmailService com servidor do Google

Checklist:

1) Acrescentar os seguintes dados em application-dev.properties:

```
spring.mail.host=smtp.gmail.com
spring.mail.username=
spring.mail.password=
spring.mail.properties.mail.smtp.auth = true
spring.mail.properties.mail.smtp.socketFactory.port = 465
spring.mail.properties.mail.smtp.socketFactory.class = javax.net.ssl.SSLSocketFactory
spring.mail.properties.mail.smtp.socketFactory.fallback = false
spring.mail.properties.mail.smtp.starttls.enable = true
spring.mail.properties.mail.smtp.ssl.enable = true
```

2) Mudar o profile do projeto para dev

3) **ATENÇÃO:** em DBService mude o email do cliente para algum email seu

4) Implementar o SmtplibEmailService utilizando nele uma instância de MailSender

5) Em DevConfig, criar um método @Bean EmailService que retorna uma instância de SmtplibEmailService

Notas:

1) Para testar não se esqueça de subir o MySQL

2) Na primeira tentativa de envio de email você vai receber um erro porque o Google por padrão bloqueia tentativa de email por app:

```
{
  "timestamp": 1508679645657,
  "status": 500,
  "error": "Internal Server Error",
  "exception": "org.springframework.mail.MailAuthenticationException",
  "message": "Authentication failed; nested exception is javax.mail.AuthenticationFailedException: 534-5.7.14 <https://accounts.google.com/signin/continue?sarp=1&sc=1&plt=AKgsbs9\\n534-5.7.14 ivkWPXaax269ytMBAjwqX64k9z1kUau3YqvsidPaVQdEYui-jDCJrBwfw9GVefHEwt_D3\\n534-5.7.14 XuushXsRFYwIOJuSZwByd80lu-6CnI37CmrEsCwwS3_Stc2c3bAXjm35D-7dtxhyrbq1Vo\\n534-5.7.14 o4AWqsUG1nbQuoPgJ-yGcPp0hXD2MVV_ix6ijUpFeqamNk39Jh1R0Pfm8-34FbjABbr0s1\\n534-5.7.14 3qhr-bn_DIAJjJhvr4-jNWrYvj3-4> Please log in via your web browser and\\n534-5.7.14 then try again.\\n534-5.7.14 Learn more at\\n534 5.7.14 https://support.google.com/mail/answer/78754 p31sm3472334qtj.12 - gsmtp\\n",
  "path": "/pedidos"
}
```

Verifique seu email do Google. Haverá a seguinte mensagem:

Clique em "*permitindo o acesso a apps menos seguros*" e habilite o envio de emails:



ATENÇÃO: ao fazer o commit, APAGUE SUAS CREDENCIAIS DE EMAIL

Atualização - outro link para liberar acesso ao Gmail

Referência:

<https://stackoverflow.com/questions/25341198/javax-mail-authenticationfailedexception-is-thrown-while-sending-email-in-java/33801654>

Libere o acesso ao app por meio de dois links:

<https://www.google.com/settings/security/lesssecureapps>

<https://accounts.google.com/b/0/DisplayUnlockCaptcha>

Email HTML

Checklist:

1) Incluir a dependência do Thymeleaf:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

2) Criar o template Thymeleaf para o email (código no final deste documento). Criar o arquivo em:
resources/templates/email/confirmacaoPedido.html

3) Em EmailService, incluir os seguintes métodos:

```
void sendOrderConfirmationHtmlEmail(Pedido obj);
void sendHtmlEmail(MimeMessage msg);
```

4) Em AbstractEmailService, incluir o seguinte método, que será responsável por retornar o HTML preenchido com os dados de um pedido, a partir do template Thymeleaf:

```
protected String htmlFromTemplatePedido(Pedido obj)
```

5) Em AbstractEmailService, implementar o novo contrato:

```
void sendOrderConfirmationHtmlEmail(Pedido obj);
```

6) Em MockEmailService, implementar os novos contratos de EmailService

7) Em SmtplibEmailService, implementar os novos contratos de EmailService

8) Em PedidoService, mudar a chamada para o método sendOrderConfirmationHtmlEmail

ATENÇÃO: ao fazer o commit, APAGUE SUAS CREDENCIAIS DE EMAIL

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title th:remove="all">Order Confirmation</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
    <div>

        <h1>
            Pedido número: <span th:text="${pedido.id}"> </span>
        </h1>
        <p>
            Instante: <span
                th:text="${#dates.format(pedido.instante, 'dd/MM/yyyy hh:mm')}"></span>
        </p>
        <p>
            Cliente: <span th:text="${pedido.cliente.nome}"></span>
        </p>
        <p>
            Situação do pagamento: <span
                th:text="${pedido.pagamento.estado.descricao}"></span>
        </p>
        <h3>Detalhes do pedido:</h3>
        <table border="1">
            <tr>
                <th>Produto</th>
                <th>Quantidade</th>
                <th>Preço unitário</th>
                <th>Subtotal</th>
            </tr>
            <tr th:each="item : ${pedido.itens}">
                <td th:text="${item.produto.nome}">nome</td>
                <td th:text="${item.quantidade}">quantidade</td>
                <td th:text="${#numbers.formatDecimal(item.preco, 0, 'POINT', 2,
'COMMA')}">preco</td>
                <td th:text="${#numbers.formatDecimal(item.subTotal, 0, 'POINT', 2,
'COMMA')}">subTotal</td>
            </tr>
        </table>
        <p>
            Valor total: <span th:text="${#numbers.formatDecimal(pedido.valorTotal, 0,
'POINT', 2, 'COMMA')}"></span>
        </p>
    </div>
</body>
</html>

```