

15.1. Conhecendo soluções para envio de e-mails transacionais

quarta-feira, 5 de abril de 2023 10:14

- Emails transacionais são tipos de emails que são disparados por eventos na aplicação
- Soluções:
 - Amazon Simple Email Service
 - Mandrill
 - MailGun
 - SendGrid

15.2. Configurando o projeto para envio de e-mails usando servidor SMTP

quarta-feira, 5 de abril de 2023

10:41

- Utilizar o SendGrid
- dependência no pom.xml

<!-- <https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-mail> -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-mail</artifactId>

<version>3.0.5</version>

</dependency>

- Configurações no application.properties
 - Criar key no SenGrid

```
#15.2. Configurando o projeto para envio de e-mails usando servidor SMTP
spring.mail.host=smtp.sendgrid.net
spring.mail.port=587
spring.mail.username=apikey
#spring.mail.password=environment variables in intellij
```

15.3. Implementando o serviço de infraestrutura de envio de e-mails com Spring

quarta-feira, 5 de abril de 2023 10:45

- Código de infraestrutura e de serviço
 - interface no domínio do serviço

```
package com.algaworks.algafood.domain.service;

import ...

2 usages 1 implementation
public interface EnvioEmailService {

    1 implementation
    void enviar(Mensagem mensagem);

    @Getter
    @Builder
    class Mensagem{
        private Set<String> destinatarios;
        private String assunto;
        private String corpo;
    }
}
```

- implementação no pacote Email em serviço de infrastructure

```
package com.algaworks.algafood.infrastructure.service.email;

import ...

@Service
public class SmtplibEnvioEmailService implements EnvioEmailService {
```

- Criar classe para representar o envio de Email (dentro da interface)
 - Set String de destinatários
 - assunto e corpo
- Beans da classe SmtplibEnvioEmailService

```
2 usages
5 @Autowired
6 private JavaMailSender javaMailSender;
7
8 1 usage
8 @Autowired
9 private EmailProperties emailProperties;
```

- JavaMailSender classe do pacote spring-boot-starter-mail
 - Classes para envio e construção do e-mail
- EmailProperties
 - informações do remetente do e-mail

```
package com.algaworks.algafood.core.email;

import ...

2 usages
@Validated
@Getter
@Setter
@Component
@ConfigurationProperties(prefix = "algafood.email")
public class EmailProperties {

    @NotNull
    private String remetente;
}
```

```
#🔧.3. Implementando o serviço de infraestrutura de envio de e-mails com Spring
algafood.email.remetente=AlgaFood <naoresponder@algafood.com.br>
```

- método de envio de e-mail na implementação da interface

```
@Override
public void enviar(Mensagem mensagem) {
    /*Objeto de email*/
    MimeMessage mimeMessage = javaMailSender.createMimeMessage();
    final MimeMessageHelper helperMimeMessage = new MimeMessageHelper(mimeMessage, encoding: "UTF-8");/* encapsula mimeMessage */

    try {
        /* construindo mensagem */
        helperMimeMessage.setTo(mensagem.getDestinatarios().toArray(new String[0]));
        helperMimeMessage.setSubject(mensagem.getAssunto());
        helperMimeMessage.setText(mensagem.getCorpo(), html: true);
        helperMimeMessage.setFrom(emailProperties.getRemetente());

        /* envio da mensagem construida */
        javaMailSender.send(mimeMessage);
    } catch (Exception e) {
        throw new EmailException("Não foi possível enviar e-mail", e);
    }
}
```

- MimeMessage
 - objeto de criação, construção e encapsulamento do e-mail
- helperMimeMessage
 - classe do Spring que ajuda a construção de um MimeMessage

15.4. Usando o serviço de envio de e-mails na confirmação de pedidos

quarta-feira, 5 de abril de 2023 17:04

- Logo após confirmar o pedido

```
@Transactional
public void confirmar(String codigoPedido){
    final Pedido pedido = pedidoService.buscar(codigoPedido);
    pedido.confirmar();

    Mensagem mensagem = Mensagem.builder()
        .assunto(pedido.getRestaurante().getNome() + " - Pedido confirmado.")
        .corpo("O pedido de código <strong>" + pedido.getId() + "</strong> foi confirmado!")
        .destinatario(pedido.getCliente().getEmail())
        .build();

    envioEmailService.enviar(mensagem);
}
```

- a propriedade

```
#💡.3. Implementando o serviço de infraestrutura de envio de e-mails com Spring
algafood.email.rementente=Elementar <wgustavo.dev@gmail.com>
```

```
@Getter
@Builder
class Mensagem{
    @Singular
    private Set<String> destinatarios;
    private String assunto;
    private String corpo;
}
}
```

nota: no site sendgrid o email tem que ser verificado

15.5. Processando template do corpo de e-mails com Apache FreeMarker

quarta-feira, 5 de abril de 2023

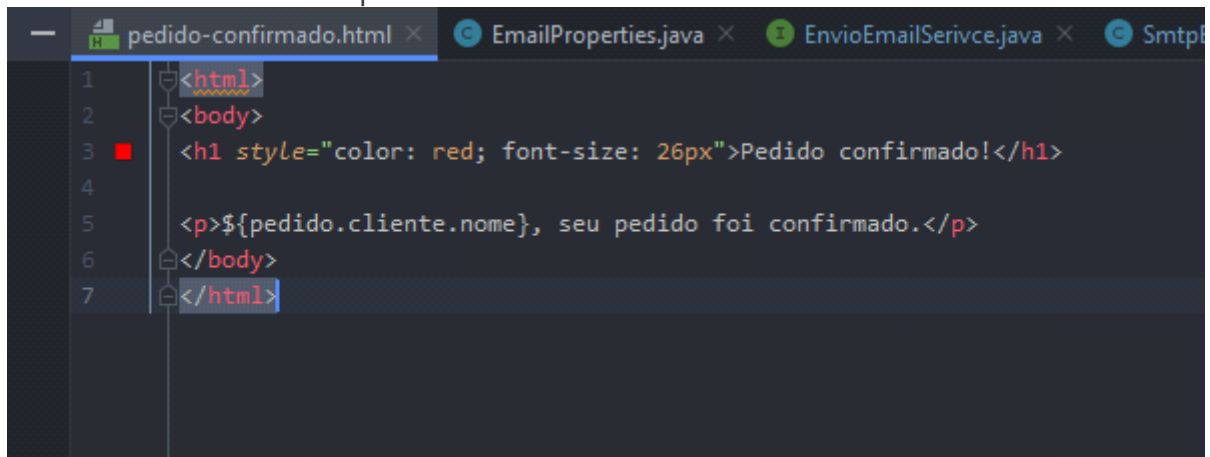
17:09

- dependência do freeMarker

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-freemarker -->
```

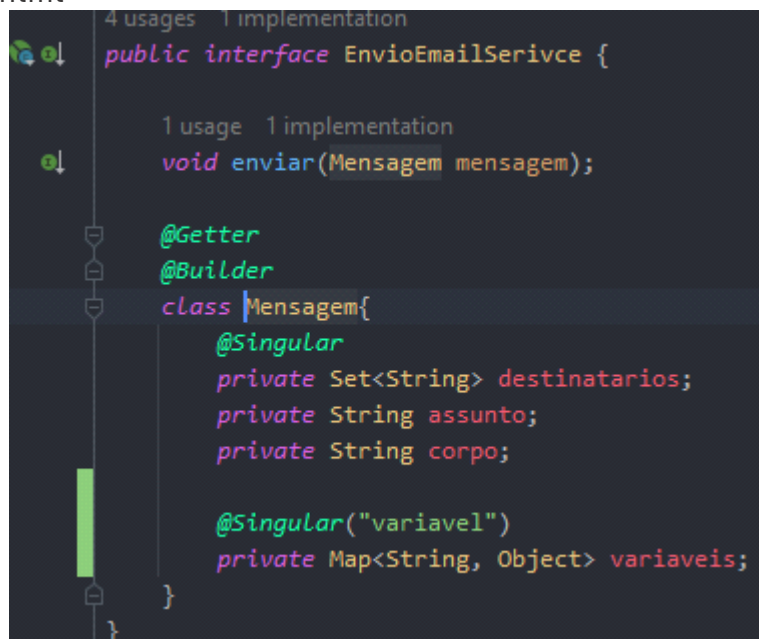
```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-freemarker</artifactId>  
  <version>3.0.5</version>  
</dependency>
```

- Caminho do template
- resources/templates



```
1 <html>  
2 <body>  
3   <h1 style="color: red; font-size: 26px">Pedido confirmado!</h1>  
4  
5   <p>${pedido.cliente.nome}, seu pedido foi confirmado.</p>  
6 </body>  
7 </html>
```

- construção do template do email a partir da mensagem
@Autowired
private Configuration freemarkerConfig;
adicionar conjunto de chaves e valores mapeados para serem substituídos no template html



```
4 usages 1 implementation  
public interface EnvioEmailService {  
  
  1 usage 1 implementation  
  void enviar(Mensagem mensagem);  
  
  @Getter  
  @Builder  
  class Mensagem {  
    @Singular  
    private Set<String> destinatarios;  
    private String assunto;  
    private String corpo;  
  
    @Singular("variavel")  
    private Map<String, Object> variaveis;  
  }  
}
```

- Em FluxoPedidoService

```

@Transactional
public void confirmar(String codigoPedido) {
    final Pedido pedido = pedidoService.buscar(codigoPedido);
    pedido.confirmar();

    Mensagem mensagem = Mensagem.builder()
        .assunto(pedido.getRestaurante().getNome() + " - Pedido confirmado.")
        .corpo("pedido-confirmado.html")
        .variavel( variavelKey: "pedido", pedido)
        .destinatario(pedido.getCliente().getEmail())
        .build();

    envioEmailService.enviar(mensagem);
}

```

o corpo da mensagem guarda o nome do template adicionado em resources/templates para o freemarker mapear a variavel adiciona o conjunto de chave/valor para o template do freemarker

- Em SmtEnvioEmailService

```

@Override
public void enviar(Mensagem mensagem) {
    /*Objeto de email*/
    MimeMessage mimeMessage = javaMailSender.createMimeMessage();
    final MimeMessageHelper helperMimeMessage = new MimeMessageHelper(mimeMessage,

    try {
        /* construindo mensagem */
        String corpo = processarTemplate(mensagem);
        helperMimeMessage.setTo(mensagem.getDestinatarios().toArray(new String[0]));
        helperMimeMessage.setSubject(mensagem.getAssunto());
        helperMimeMessage.setText(corpo, html: true);
        helperMimeMessage.setFrom(emailProperties.getRemetente());

        /* envio da mensagem construida */
        javaMailSender.send(mimeMessage);
    } catch (Exception e) {
        throw new EmailException("Não foi possível enviar e-mail", e);
    }
}

```

o método processarTemplate recebe a mensagem construída para o MimeMessage e processa o template baseado nas variáveis de chave/valor para a construção do template

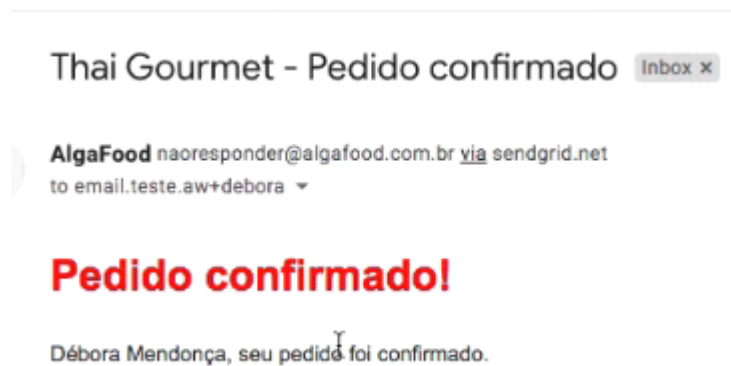
```

1 usage
public String processarTemplate(Mensagem mensagem){
    try {
        final Template template = freemarkerConfig.getTemplate(mensagem.getCorpo());
        return FreeMarkerTemplateUtils.processTemplateIntoString(template, mensagem.getVariaveis());
    } catch (Exception e) {
        throw new EmailException("Não foi possível montar o template do e-mail.", e);
    }
}

```

o objeto injetado `freemarkerConfig.getTemplate` recebe como `String` o nome do template em html

a classe utilitária `FreeMarkerTemplateUtils.processTemplateIntoString` processa o template e a coleção de chaves/valores



15.6. Melhorando o texto do e-mail com FTL (FreeMarker Template Language)

quarta-feira, 5 de abril de 2023 17:44

```
pedido-confirmado.html x EmailProperties.java x EnvioEmailServce.java x SmtpeEnvioEmailSer
1 <html>
2 <body style="font: 14px Arial, Helvetica, sans-serif">
3   <h1 style="color: red; font-size: 26px">Pedido confirmado!</h1>
4
5   <p>${pedido.cliente.nome}, seu pedido foi confirmado pelo restaurante e já
6     está sendo preparado.</p>
7
8   <h2 style="font-size: 20px">${pedido.restaurante.nome}</h2>
9
10  <table width="100%" border="0" cellpadding="0"
11    style="max-width: 400px; color: #6F6F6F">
12    <#list pedido.itensPedido as item>
13      <tr>
14        <td style="padding: 10px 0">${item.quantidade}x ${item.produto.nome}</td>
15        <td style="width: 30px">${item.precoTotal}</td>
16      </tr>
17    </#list>
18
19    <tr>
20      <td style="padding: 10px 0">Frete</td>
21      <td>${pedido.taxaFrete}</td>
22    </tr>
23    <tr style="font-weight: bold">
24      <td style="padding: 10px 0">Total</td>
25      <td>${pedido.valorTotal}</td>
26    </tr>
27  </table>
28
29  <h2 style="font-size: 20px">Forma de pagamento</h2>
30  <p style="color: #6F6F6F">${pedido.formaPagamento.descricao}</p>
31</body>
32</html>
```


15.7. Formatando valores monetários com FTL

quarta-feira, 5 de abril de 2023 20:08

```
<tr style="font-weight: bold">
  <td style="padding: 10px 0">Total</td>
  <td>${pedido.valorTotal?string.currency}</td>
</tr>
```

```
#15.7. Formatando valores monetários com FTL
spring.freemarker.settings.locale=pt_BR
```

15.8. Desafio: implementando serviço de envio de e-mail fake

quarta-feira, 5 de abril de 2023 20:24

- Criar classe mock que implementa a interface EnvioEmailService

```
package com.algaworks.algafood.infrastructure.service.email;

import ...

2 usages
@Slf4j
public class MockEmailService implements EnvioEmailService {
    1 usage
    @Override
    public void enviar(Mensagem mensagem) {

        final Set<String> destinatarios = mensagem.getDestinatarios();

        final String destinatario = destinatarios.stream().findFirst().get();
        final Map<String, Object> variaveis = mensagem.getVariaveis();
        final Pedido pedido = (Pedido) variaveis.get("pedido");

        log.info("Alteração de Pedido");
        log.info(mensagem.getAssunto());
        log.info("Destinatário: " + destinatario);
        log.info("Olá, " + pedido.getClient().getNome() + ", seu pedido foi confirmado!");
        log.info("O pedido de código " + pedido.getId() + " foi confirmado!");

        pedido.getItemsPedido().forEach(item -> {
            log.info("Quantidade \t\t Produto");
            log.info(item.getQuantidade() + "\t\t" + item.getProduto().getNome());
        });

        log.info(String.format("Taxa frete: R$ %.2f", pedido.getRestaurante().getTaxaFrete()));
        log.info(String.format("Subtotal: R$ %.2f", pedido.getSubTotal()));
        log.info(String.format("Valor total: R$ %.2f", pedido.getValorTotal()));
    }
}
```

- Criar classe de configuração de Bean

```
@Configuration
public class EmailConfig {

    1 usage
    @Autowired
    private EmailProperties emailProperties;

    @Bean
    public EnvioEmailService envioEmailService(){

        return switch(emailProperties.getImplementacao()){
            case MOCK -> new MockEmailService();
            case SENDGRID -> new SmtplibEnvioEmailService();
            case SANDBOX -> new SandboxEmailService();
        };
    }
}
```

- Definição no application.properties

```
spring.profiles.active=mock

#15.8. Desafio: implementando serviço de envio de e-mail fake
algafood.email.implementacao=mock
```

- EmailProperties

```
package com.algaworks.algafood.core.email;

import ...

3 usages
@Validated
@Getter
@Setter
@Component
@ConfigurationProperties(prefix = "algafood.email")
public class EmailProperties {

    @NotNull
    private String remetente;

    private ImplEmail implementacao = ImplEmail.MOCK;

    2 usages
    public enum ImplEmail{
        2 usages
        MOCK,
        1 usage
        SENDGRID;
    }
}
```

15.9. Desafio: Implementando serviço de envio de e-mail sandbox

quinta-feira, 6 de abril de 2023 09:58

- Criar propriedade que representa o destinatário do sandbox e um atributo da enumeração para configurar o Bean

```
#15.8. Desafio: implementando serviço de envio de e-mail fake
algafood.email.implementacao=sandbox

#15.9. Desafio: Implementando serviço de envio de e-mail sandbox
algafood.email.sandbox.destinatario=wgustavo.dev@gmail.com
```

- Criar classe que estende a SmtEnvioEmailService

```
2 usages
public class SandboxEmailService extends SmtEnvioEmailService {

    @Autowired
    private JavaMailSender javaMailSender;

    1 usage
    @Autowired
    private EmailProperties emailProperties;

    @Autowired
    private Configuration freemarkerConfig;

    2 usages
    @Override
    protected MimeMessage criarMimeMessage(Mensagem mensagem) throws MessagingException {
        final MimeMessage mimeMessage = super.criarMimeMessage(mensagem);

        MimeMessageHelper mimeMessageHelper = new MimeMessageHelper(mimeMessage, encoding: "UTF-8");
        mimeMessageHelper.setTo(emailProperties.getSandbox().getDestinatario());
        return mimeMessage;
    }
}
```

somente o método sobrescrito vai ser chamado. O fluxo é:

classe FluxoPedidoService vai receber uma implementação da interface EnvioEmailService através da definição do Bean de SandboxEmailService

```

1 usage
@Transactional
public void confirmar(String codigoPedido) {
    final Pedido pedido = pedidoService.buscar(codigoPedido);
    pedido.confirmar();

    Mensagem mensagem = Mensagem.builder()
        .assunto(pedido.getRestaurante().getNome() + " - Pedido confirmado.")
        .corpo("pedido-confirmado.html")
        .variavel(variavelKey: "pedido", pedido)
        .destinatario(pedido.getCliente().getEmail())
        .build();

    envioEmailService.enviar(mensagem);
}

```

a classe vai chamar o método enviar de SandboxEmailService que é o mesmo da classe pai por não estar sobrescrito (pois herdou todos os métodos da classe pai)

```

2 usages
public class SandboxEmailService extends SmtEnvioEmailService {

    @Autowired
    private JavaMailSender javaMailSender;

    1 usage
    @Autowired
    private EmailProperties emailProperties;

    @Autowired
    private Configuration freemarkerConfig;

    2 usages
    @Override
    protected MimeMessage criarMimeMessage(Mensagem mensagem) throws MessagingException {
        final MimeMessage mimeMessage = super.criarMimeMessage(mensagem);

        MimeMessageHelper mimeMessageHelper = new MimeMessageHelper(mimeMessage, encoding: "UTF-8");
        mimeMessageHelper.setTo(emailProperties.getSandbox().getDestinatario());
        return mimeMessage;
    }
}

```

mas, na classe pai, dentro do método enviar(), vai ser invocado a chamada pro método criarMimeMessage

```

@Override
public void enviar(Mensagem mensagem) {
    try {
        MimeMessage mimeMessage = criarMimeMessage(mensagem);

        /* envio da mensagem construida */
        javaMailSender.send(mimeMessage);
    } catch (Exception e) {
        throw new EmailException("Não foi possível enviar e-mail", e);
    }
}

2 usages 1 override
protected MimeMessage criarMimeMessage(Mensagem mensagem) throws MessagingException {
    /*Objeto de email*/
}

```

esse método está sobrescrito na classe filha, e muda a trajetória do código

```

2 usages
public class SandboxEmailService extends SmtEnvioEmailService {

    @Autowired
    private JavaMailSender javaMailSender;

    1 usage
    @Autowired
    private EmailProperties emailProperties;

    @Autowired
    private Configuration freemarkerConfig;

    2 usages
    @Override
    protected MimeMessage criarMimeMessage(Mensagem mensagem) throws MessagingException {
        final MimeMessage mimeMessage = super.criarMimeMessage(mensagem);

        MimeMessageHelper mimeMessageHelper = new MimeMessageHelper(mimeMessage, encoding: "UTF-8");
        mimeMessageHelper.setTo(emailProperties.getSandbox().getDestinatario());
        return mimeMessage;
    }
}

```

dentro do método sobrescrito, chamamos o método original e instanciamos um MimeMessage, logo a baixo sobrescrevemos o valor em setTo para o destinatário do sandbox



[Domain Events](#)

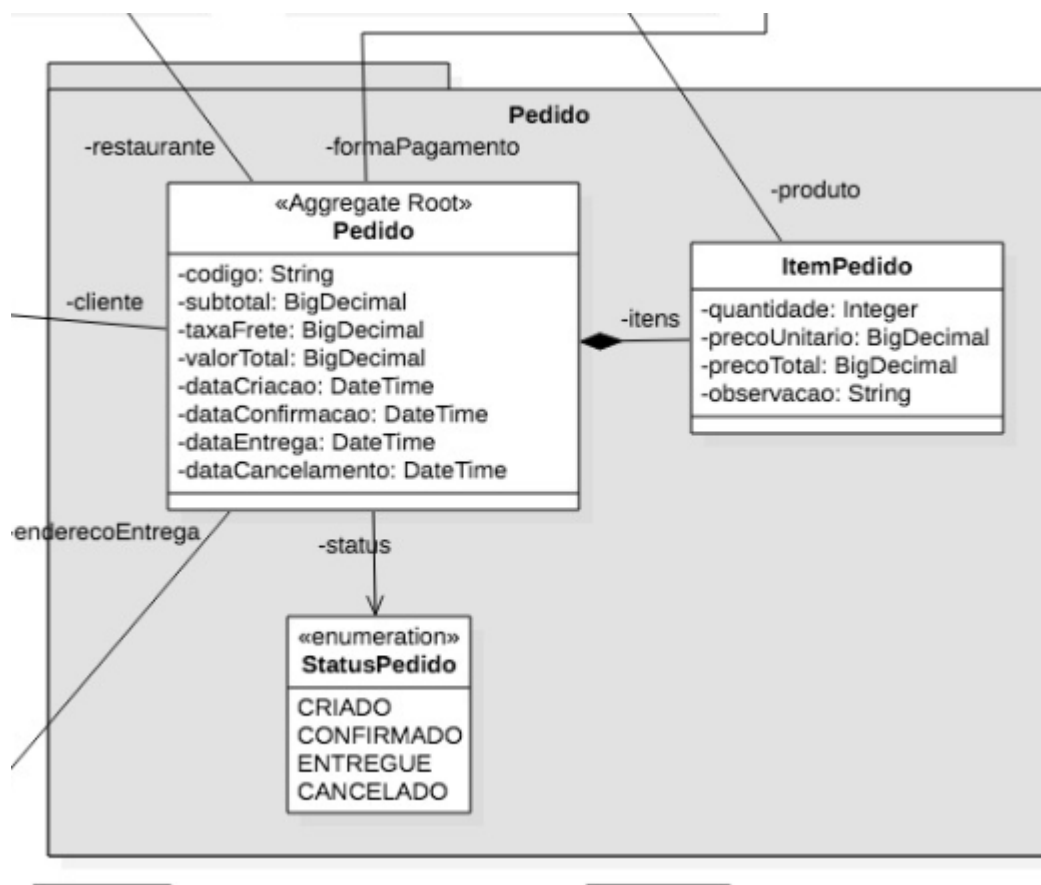
15.11. Publicando Domain Events a partir do Aggregate Root

quinta-feira, 6 de abril de 2023

11:00

- Disparo de evento de domínio a partir de um Aggregate Root para Pedido Confirmado
 - Um evento descreve o que aconteceu no passado, que é de interesse do negócio, para que outros componentes possam reagir baseado no evento
- Domínio

```
FluxoPedidoService.java x
17
18 1 usage
19 @Transactional
20 public void confirmar(String codigoPedido) {
21     final Pedido pedido = pedidoService.buscar(codigoPedido);
22     pedido.confirmar();
23
24     Mensagem mensagem = Mensagem.builder()
25         .assunto(pedido.getRestaurante().getNome() + " - Pedido confirmado.")
26         .corpo("pedido-confirmado.html")
27         .variavel( variavelKey: "pedido", pedido)
28         .destinatario(pedido.getCliente().getEmail())
29         .build();
30     envioEmailService.enviar(mensagem);
31 }
```



No que diz respeito a definição de eventos de domínio, devemos publicar eventos apenas em Aggregates Roots do domínio, no caso de Pedido. Não podemos registrar esses eventos em entidades secundárias do domínio.

```
1 usage
@Transactional
public void confirmar(String codigoPedido) {
    final Pedido pedido = pedidoService.buscar(codigoPedido);
    pedido.confirmar();

    /*Chamada para envio de email retirada para publicação
    de eventos no aggregate root da entidade*/
}
```

- Estender AbstractAggregateRoot<Entidade>

```
@Entity
@Data
@EqualsAndHashCode(onlyExplicitlyIncluded = true, callSuper = false)
public class Pedido extends AbstractAggregateRoot<Pedido> {
```

callSuper = false para não aplicar EqualAndHashCode para a superclasse

```
public void confirmar(){
    setStatus(StatusPedido.CONFIRMADO);
    setDataConfirmacao(OffsetDateTime.now());

    registerEvent(event);
}

1 usage
```

A entidade agora possui métodos de eventos, e um deles, para registrar eventos que devem ser disparados quando um evento ocorre, no caso, quando um pedido é confirmado. A instância do evento deve conter todas as informações relevantes do evento a ser disparado.

- Criar classe que representa o Evento

```
package com.algaworks.algafood.domain.event;

import ...

@Getter
@AllArgsConstructor
public class PedidoConfirmadoEvent {

    private Pedido pedido;

}
```



```

1 usage
public void confirmar(){
    setStatus(StatusPedido.CONFIRMADO);
    setDataConfirmacao(OffsetDateTime.now());

    registerEvent(new PedidoConfirmadoEvent(this));
}

```

Neste momento, não estamos disparando de fato um evento, mas apenas registrando um evento que deve ser disparado assim que a entidade for salva no banco de dados.

Para disparar eventos de pedidos confirmados, a entidade tem que ser gerenciada pelo Spring Data e a partir do método save, o evento será registrado para que qualquer componente que esteja interessado possa acessar o evento disparado. Nesse momento temos um **ponto de extensão** da nossa aplicação.

```

1 usage
@Autowired
private PedidoRepository pedidoRepository;

1 usage
@Transactional
public void confirmar(String codigoPedido) {
    final Pedido pedido = pedidoService.buscar(codigoPedido);
    pedido.confirmar();
    pedidoRepository.save(pedido);
}

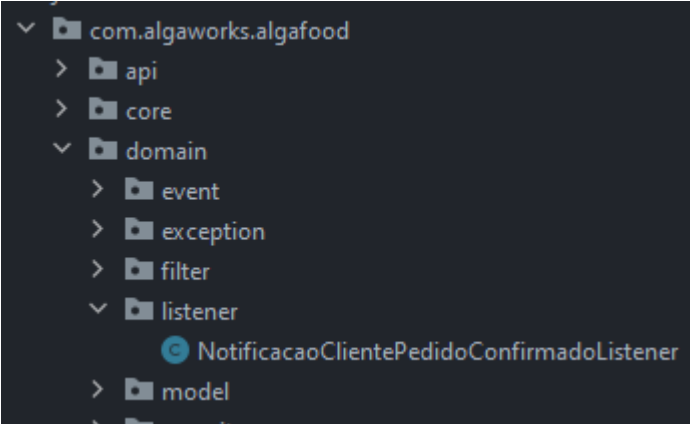
```

15.12. Observando e reagindo a Domain Events

quinta-feira, 6 de abril de 2023 12:01

Dado um evento registrado, em toda a aplicação pode-se capturar a instância desse evento em diferentes partes, para desacoplar a responsabilidade de outras classes

- Criar pacote Listener para adicionar componentes que tem a responsabilidade de ficar "escutando eventos"



- Criar classe que captura o evento

```
@Component
public class NotificacaoClientePedidoConfirmadoListener {

    1 usage
    @Autowired
    private EnvioEmailServivce envioEmailServivce;

    @EventListener
    public void aoConfirmarPedido(PedidoConfirmadoEvent event){
        final Pedido pedido = event.getPedido();
        EnvioEmailServivce.Mensagem mensagem = EnvioEmailServivce.Mensagem.builder()
            .assunto(pedido .getRestaurante().getNome() + " - Pedido confirmado.")
            .corpo("pedido-confirmado.html")
            .variavel( variavelKey: "pedido", pedido )
            .destinatario(pedido .getCliente().getEmail())
            .build();

        envioEmailServivce.enviar(mensagem);
    }
}
```

Tiramos a responsabilidade de enviar emails do método confirmar da classe FluxoPedidoService.

Caso haja outro Listener esperando um evento usado por outro Listener, ambos os Listeners serão chamados com um único evento disparado, ou seja, podemos ter vários Listerners para um evento

15.13. Reagindo a Domain Events em fases específicas da transação

quinta-feira, 6 de abril de 2023

19:00

- Os eventos ocorrem antes da efetivação da transação no banco de dados, como:

```
FluxoPedidoService.java x NotificacaoClientePedidoConfirmadoListener.java x Pe
18      @Transactional
19      public void confirmar(String codigoPedido) {
20          System.out.println("Buscando pedidos no banco de dados");
21          final Pedido pedido = pedidoService.buscar(codigoPedido);
22          pedido.confirmar();
23          System.out.println("Antes do método save do Spring Data");
24          pedidoRepository.save(pedido);
25          System.out.println("Depois do método save do Spring Data");
26
27          /*Chamada para envio de email retirada para publicação
28             de eventos no aggregate root da entidade*/
29      }
```

```
public void confirmar(){
    setStatus(StatusPedido.CONFIRMADO);
    setDataConfirmacao(OffsetDateTime.now());

    registerEvent(new PedidoConfirmadoEvent(this));
    System.out.println("Evento registrado");
}
```

log

```
Buscando pedidos no banco de dados
Hibernate: select pedido0_.id as id1_8_, pedido0_.usuario_cliente_id as usuario16_8_, pedido0_.codigo as codigo2_8_, pedido0_.data_criacao as data_criacao2_8_, pedido0_.data_cancelamento as data_cancelamento2_8_, pedido0_.data_confirmacao as data_confirmacao2_8_, pedido0_.data_atualizacao as data_atualizacao2_8_ from pedido0_ where pedido0_.codigo=?
Hibernate: select usuario0_.id as id1_14_0_, usuario0_.data_cadastro as data_cad2_14_0_, usuario0_.email as email3_14_0_, usuario0_.senha as senha3_14_0_ from usuario0_ where usuario0_.id=?
Hibernate: select restaurant0_.id as id1_11_0_, restaurant0_.aberto as aberto2_11_0_, restaurant0_.ativo as ativo3_11_0_ from restaurant0_ where restaurant0_.id=?
Evento registrado
Antes do método save do Spring Data
2023-04-06 22:28:04.519 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Alteração de Pedido
2023-04-06 22:28:04.519 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Thai Gourmet - Pedido
2023-04-06 22:28:04.519 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Destinatário: guto15s
2023-04-06 22:28:04.520 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Olá, Débora Mendonça,
2023-04-06 22:28:04.520 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : O pedido de código 6
Hibernate: select itenspedido0_.pedido_id as pedido_i6_7_0_, itenspedido0_.id as id1_7_0_, itenspedido0_.id as id1_7_1_, itenspedido0_.quantidade as quantidade2_7_0_, itenspedido0_.descricao as descricao2_7_0_ from itenspedido0_ where itenspedido0_.pedido_id=?
2023-04-06 22:28:04.527 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Quantidade
2023-04-06 22:28:04.528 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : 1 Camarão tailandês
2023-04-06 22:28:04.528 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Taxa frete: R$ 10,00
2023-04-06 22:28:04.529 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Subtotal: R$ 87,20
2023-04-06 22:28:04.529 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Valor total: R$ 97,20
Depois do método save do Spring Data
Hibernate: update pedido set usuario_cliente_id=?, codigo=?, data_cancelamento=?, data_confirmacao=?, data_criacao=?, data_atualizacao=? where pedido.id=?
```

e no final do método anotado com `@Transactional` o commit ocorre, mesmo se chamar o método `flush`, pois a implementação dos eventos são disparados sempre antes do Flush e/ou Commit do Spring Data

Nota: o objeto adicionado no evento tem que ser gerenciado pelo contexto de persistência

Para mudar esse comportamento, podemos usar `@TransactionalEventListener`

```
// @EventListener
@TransactionalEventListener
public void aoConfirmarPedido(PedidoConfirmadoEvent event){
    final Pedido pedido = event.getPedido();
    EnvioEmailService.Mensagem mensagem = EnvioEmailService.Mensagem.builder()
        .assunto(pedido.getRestaurante().getNome() + " - Pedido confirmado.")
        .corpo("pedido-confirmado.html")
        .variavel( variavelKey: "pedido", pedido )
        .destinatario(pedido.getCliente().getEmail())
        .build();

    envioEmailService.enviar(mensagem);
}
```

Agora o evento é publicado depois de finalizar o método anotado com `@Transactional`, depois do commit no banco de dados

```
Buscando pedidos no banco de dados
Hibernate: select pedido0_.id as id1_8_, pedido0_.usuario_cliente_id as usuario16_8_, pedido0_.codigo as codigo2_8_, ped
Hibernate: select usuario0_.id as id1_14_0_, usuario0_.data_cadastro as data_cad2_14_0_, usuario0_.email as email3_14_0_
Hibernate: select restaurant0_.id as id1_11_0_, restaurant0_.aberto as aberto2_11_0_, restaurant0_.ativo as ativo3_11_0_
Evento registrado
Antes do método save do Spring Data
Depois do método save do Spring Data
Hibernate: update pedido set usuario_cliente_id=?, codigo=?, data_cancelamento=?, data_confirmacao=?, data_criacao=?, da
2023-04-06 22:50:16.915 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Alteração de Pedido
2023-04-06 22:50:16.915 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Thai Gourmet - Pedi
```

Porém, caso acontecer um erro dentro do método ouvinte, não será feito o rollback na transação com o banco de dados, logo, o evento não será disparado mas a transação será feita

propriedades phase na anotação

```
// @EventListener
@TransactionalEventListener(phase = TransactionPhase.BEFORE_COMMIT)
public void aoConfirmarPedido(PedidoConfirmadoEvent event){
    if (true) throw new IllegalArgumentException();
    final Pedido pedido = event.getPedido();
    EnvioEmailService.Mensagem mensagem = EnvioEmailService.Mensagem.builder()
        .assunto(pedido.getRestaurante().getNome() + " - Pedido confirmado.")
        .corpo("pedido-confirmado.html")
        .variavel( variavelKey: "pedido", pedido )
        .destinatario(pedido.getCliente().getEmail())
        .build();
}
```

phase = TransactionalPhase.BEFORE_COMMIT

- Manipula o evento antes do commit da transação, e caso um erro ocorrer dentro do método da manipulação do evento.

```

Buscando pedidos no banco de dados
Hibernate: select pedido0_.id as id1_8_, pe
Hibernate: select usuario0_.id as id1_14_0,
Hibernate: select restaurant0_.id as id1_11
Evento registrado
Antes do método save do Spring Data
Depois do método save do Spring Data
java.lang.IllegalArgumentException Create bre
at com.algaworks.algafood.domain.liste

```

```

Depois do método save do Spring Data
2023-04-06 23:16:23.139 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Altera
2023-04-06 23:16:23.139 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Thai O
2023-04-06 23:16:23.139 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Desti
2023-04-06 23:16:23.141 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Olá, U
2023-04-06 23:16:23.141 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : O ped
Hibernate: select itenspedido_.pedido_id as pedido_i6_7_0_, itenspedido_.id as id1_7_0_, itenspedido_.id as
2023-04-06 23:16:23.146 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Quant
2023-04-06 23:16:23.150 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : 1
2023-04-06 23:16:23.151 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Taxa
2023-04-06 23:16:23.151 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Subto
2023-04-06 23:16:23.151 INFO 13864 --- [nio-8080-exec-2] c.a.a.i.service.email.MockEmailService : Valor
Hibernate: update pedido set usuario_cliente_id=?, codigo=?, data_cancelamento=?, data_confirmacao=?, data

```

phase = TransactionalPhase.AFTER_COMMIT:(padrão) primeiro o commit, depois o evento

phase = TransactionalPhase.BEFORE_COMMIT : primeiro o evento, depois o commit

15.14. Desafio: enviando e-mails no cancelamento de pedidos

sexta-feira, 7 de abril de 2023 08:20

- Criar template html do pedido

```
<html>
<body style="font: 14px Arial, Helvetica, sans-serif">
<h1 style="color: red; font-size: 26px">Pedido cancelado!</h1>

<p>${pedido.cliente.nome}, seu pedido foi cancelado pelo restaurante. :(</p>

<h2 style="font-size: 20px">${pedido.restaurante.nome}</h2>

<table width="100%" border="0" cellpadding="0" cellspacing="0"
      style="max-width: 400px; color: #6F6F6F">
  <#list pedido.itensPedido as item>
    <tr>
      <td style="padding: 10px 0">${item.quantidade}x ${item.produto.nome}</td>
      <td style="width: 30px">${item.precoTotal?string.currency}</td>
    </tr>
  </#list>

  <tr>
    <td style="padding: 10px 0">Frete</td>
    <td>${pedido.taxaFrete?string.currency}</td>
  </tr>
  <tr style="font-weight: bold">
    <td style="padding: 10px 0">Total</td>
    <td>${pedido.valorTotal?string.currency}</td>
  </tr>
</table>

<h2 style="font-size: 20px">Forma de pagamento</h2>
<p style="color: #6F6F6F">${pedido.formaPagamento.descricao}</p>
</body>
</html>
```

- Criar classe do Evento

```
package com.algaworks.algafood.domain.event;

import ...

@AllArgsConstructor
@Getter
public class PedidoCanceladoEvent {
    private Pedido pedido;
}
```

- Criar classe ouvinte do evento

```
package com.algaworks.algafood.domain.listener;

import ...

@Component
public class NotificacaoClientePedidoCanceladoListener {

    @Autowired
    private EnvioEmailService envioEmailService;

    @TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)
    public void aoCancelarPedido(PedidoCanceladoEvent event) {
        final Pedido pedido = event.getPedido();

        EnvioEmailService.Mensagem mensagem = EnvioEmailService.Mensagem.builder()
            .assunto(pedido.getRestaurante().getNome() + " - Pedido Cancelado.")
            .corpo("pedido-cancelado.html")
            .variavel( variavelKey: "pedido", pedido)
            .destinatario(pedido.getCliente().getEmail())
            .build();

        envioEmailService.enviar(mensagem);
    }
}
```

- Registrar evento
 - Entidade Pedido (Aggregate Root e gerenciado pelo contexto de persistência)

```
1 usage
public void cancelar(){
    setStatus(StatusPedido.CANCELADO);
    setDataCancelamento(OffsetDateTime.now());
    registerEvent(new PedidoCanceladoEvent(this));
}
```

```
1 usage
@Transactional
public void cancelar(String codigoPedido) {
    final Pedido pedido = pedidoService.buscar(codigoPedido);
    pedido.cancelar();
    pedidoRepository.save(pedido);
}
```