

👑 Sistema de Petshop 👑

Grupo nº3 - Padawans



Gabriel Luppi, Mateus Castro & Jean Silva

App:

A nossa API foi desenvolvida com base nas necessidades e dificuldades de comércios petshop.

Trouxemos a tona funcionalidades de cadastro dos dados de clientes e seus pets, permitindo criar/alterar/deletar seus dados.

Também criamos um sistema que gera automaticamente o valor de cada pedido dos clientes, usando como base seu pet.

App:

Permitimos que os clientes acessassem seus próprios dados, podendo gerar novos pedidos de forma on-line

Implementamos relatórios de dados relevantes para o dia-a-dia, além de registrar o lucro-bruto e quantidade de pedidos por mês

A API tem restrições de autenticações de acordo com o tipo de usuário que está acessando ela

Sobre o módulo:

Criamos duas API's extras: uma para processar e registrar os dados para o balanço mensal, e a segunda para processar e enviar os emails da aplicação




Vale pontuar que também consumimos os dados da API petshop-consumer a partir de chave estrangeira na petshop-api.
Adicionando endpoints para recuperar os dados de balanço mensal

Sobre o módulo:

Utilizamos o particionamento para separar os tipos de cada envio de email, e também adicionamos um 4º tipo, o MARKETING

O email de marketing está programado para uma vez ao mês enviar emails para os nossos clientes, realizando pausas de 5 minutos entre cada envio para evitar o spam

Construção do corpo de envio de mensagem kafka

```
private void envioMensagemGenerica(String mensagem, MessageBuilder<String> stringMessageBuilder) {  
    Message<String> stringMessage = stringMessageBuilder.build();  
  
    ListenableFuture<SendResult<String, String>> future = kafkaTemplate.send(stringMessage);  
  
     Gabriel Luppi  
    future.addCallback(new ListenableFutureCallback<>() {  
         Gabriel Luppi  
        @Override  
        public void onFailure(Throwable ex) { log.info("Erro ao enviar mensagem: '{}', para o Kafka", mensagem); }  
         Gabriel Luppi  
        @Override  
        public void onSuccess(SendResult<String, String> result) {  
            log.info("Mensagem: '{}', enviada para o Kafka com sucesso", mensagem);  
        }  
    });  
}
```

Métodos de envio de mensagens Kafka


```
public void enviarMensagemSemParticao(String mensagem, String topico) {  
    MessageBuilder<String> stringMessageBuilder = MessageBuilder.withPayload(mensagem)  
        .setHeader(KafkaHeaders.TOPIC, topico)  
        .setHeader(KafkaHeaders.MESSAGE_KEY, UUID.randomUUID().toString());  
    envioMensagemGenerica(mensagem, stringMessageBuilder);  
}
```

1 usage  Gabriel Luppi

```
public void enviarMensagemParticionada(String mensagem, String topico, Integer particao) {  
    MessageBuilder<String> stringMessageBuilder = MessageBuilder.withPayload(mensagem)  
        .setHeader(KafkaHeaders.TOPIC, topico)  
        .setHeader(KafkaHeaders.PARTITION_ID, particao)  
        .setHeader(KafkaHeaders.MESSAGE_KEY, UUID.randomUUID().toString());  
    envioMensagemGenerica(mensagem, stringMessageBuilder);  
}
```

Métodos de envio de pedidos e emails para o producer

```
public void sendPedido(PedidoDTOConsumer pedidoDTOConsumer) throws JsonProcessingException {  
    String mensagemFinal = objectMapper.writeValueAsString(pedidoDTOConsumer);  
    enviarMensagemSemParticao(mensagemFinal, topicoBanco);  
}
```

2 usages  Gabriel Luppi

```
public void sendMessageEmail(ClienteEmailMessageDTO clienteDadosEmail, TipoRequisicao requisicao)  
    String mensagemDadosEmail = objectMapper.writeValueAsString(clienteDadosEmail);  
    enviarMensagemParticionada(mensagemDadosEmail, topicoEmail, requisicao.ordinal());  
}
```


Envio de email mensal de marketing (schedule)

```
@Scheduled(cron = "* 30 12 15 * *" )
private void sendEmailDeMarketingPorMes() throws JsonProcessingException, InterruptedException {
    List<ClienteEntity> clienteEntities = clienteRepository.findAll();
    for(ClienteEntity cliente : clienteEntities){
        ClienteEmailMessageDTO clienteEmailMessageDTO = new ClienteEmailMessageDTO();
        clienteEmailMessageDTO.setNome(cliente.getNome());
        clienteEmailMessageDTO.setEmail(cliente.getEmail());
        clienteEmailMessageDTO.setIdCliente(cliente.getIdCliente());
        kafkaProducer.sendMessageEmail(clienteEmailMessageDTO, TipoRequisicao.MARKETING);
        Thread.sleep( millis: 300000);
    }
}
```

Interface de consulta aos dados externos de balanço mensal

```
2 usages  👤 Jean
@FeignClient(value="consumer-petshop", url="localhost:8070")
@Headers({"Content-Type: application/json"})
public interface ConsumerPetshopClient {

    1 usage  👤 Jean
    @RequestLine("GET /balanco-mensal/mes-atual")
    BalancoMensalDTO getMesAtual(@HeaderMap Map<String, String> headerMap);

    1 usage  👤 Jean
    @RequestLine("GET /balanco-mensal/mes/{mes}/ano/{ano}")
    BalancoMensalDTO getByMesAno(@Param("mes") Integer mes, @Param("ano") Integer ano, @HeaderMap Map<String, String> headerMap);

    1 usage  👤 Jean
    @RequestLine("GET /balanco-mensal/pedidos-mes-atual")
    PedidoMensalDto getPedidosMesAtual(@HeaderMap Map<String, String> headerMap);

    1 usage  👤 Jean
    @RequestLine("GET /balanco-mensal/pedidos/mes/{mes}/ano/{ano}")
    PedidoMensalDto getPedidosByMesAno(@Param("mes") Integer mes, @Param("ano") Integer ano, @HeaderMap Map<String, String> headerMap);

    1 usage  👤 Jean
    @RequestLine("POST /auth")
    String auth(LoginCreateDTO loginCreateDTO);
}
```

Listener do consumer-petshop. Chama a atualização do balanço sempre que escuta uma nova mensagem de pedido CONCLUÍDO

👤 Mateus de Castro

```
@KafkaListener(  
    topics = "${kafka.balanco-topic}",  
    groupId = "group1",  
    containerFactory = "listenerContainerFactory",  
    clientIdPrefix = "balanco-mensal")  
public void consumir(@Payload String mensagem,  
    @Header(KafkaHeaders.RECEIVED_MESSAGE_KEY) String key,  
    @Header(KafkaHeaders.OFFSET) Long offset) throws JsonProcessingException {  
    PedidoDTOConsumer pedido = objectMapper.readValue(mensagem, PedidoDTOConsumer.class);  
    balançoMensalService.atualizarBalanço(pedido);  
    pedidosMensalService.atualizarPedidos(pedido);  
}
```

Listener do email-consumer. Filtra tipos de email pela partição e realiza o envio

```
@KafkaListener(  
    topics = "${kafka.email-topic}",  
    groupId = "${kafka.client-id}",  
    containerFactory = "listenerContainerFactory",  
    clientIdPrefix = "chat"  
)  
public void enviarEmail(@Payload String payload,  
                        @Header(KafkaHeaders.RECEIVED_MESSAGE_KEY) String key,  
                        @Header(KafkaHeaders.RECEIVED_PARTITION_ID) Integer partition,  
                        @Header(KafkaHeaders.OFFSET) Long offset) throws JsonProcessingException {  
    ClienteEmailMessageDTO dadosEnvioEmail = objectMapper.readValue(payload, ClienteEmailMessageDTO.class);  
    if(partition == TipoRequisicao.POST.ordinal()) {  
        log.info("ENVIANDO EMAIL TIPO POST");  
  
        emailService.sendEmail(  
            dadosEnvioEmail.getNome(),  
            dadosEnvioEmail.getIdCliente(),  
            dadosEnvioEmail.getEmail(),  
            TipoRequisicao.POST);  
  
        getLog(dadosEnvioEmail);  
    }  
}
```

Diagrama de arquitetura

