Engenharia de Softwares Escaláveis

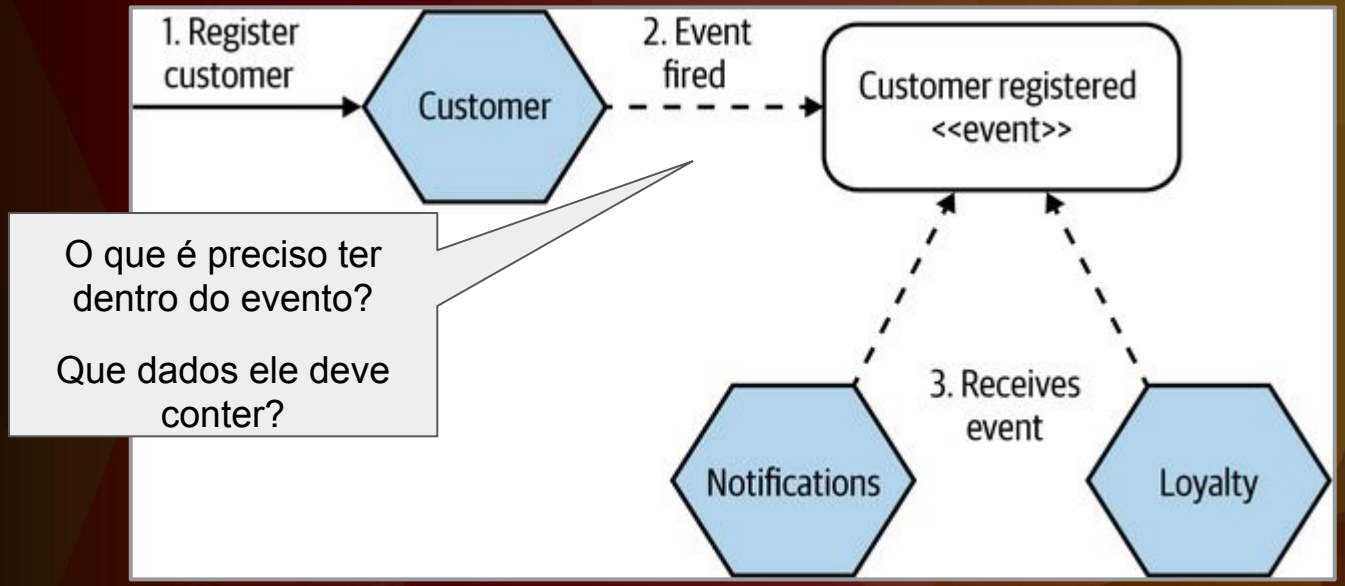
Domain-Driven Design (DDD) e Arquitetura de Softwares Escaláveis com Java

Agenda

Etapa 4: Publicação e Armazenamento de Domain Events.

- Revisando a Estratégia de Eventos.
- Google Cloud Pub/Sub - Parte 2.
- Modelo de Arquitetura Pet Friends: E-Commerce, Almoxarifado, Transporte.

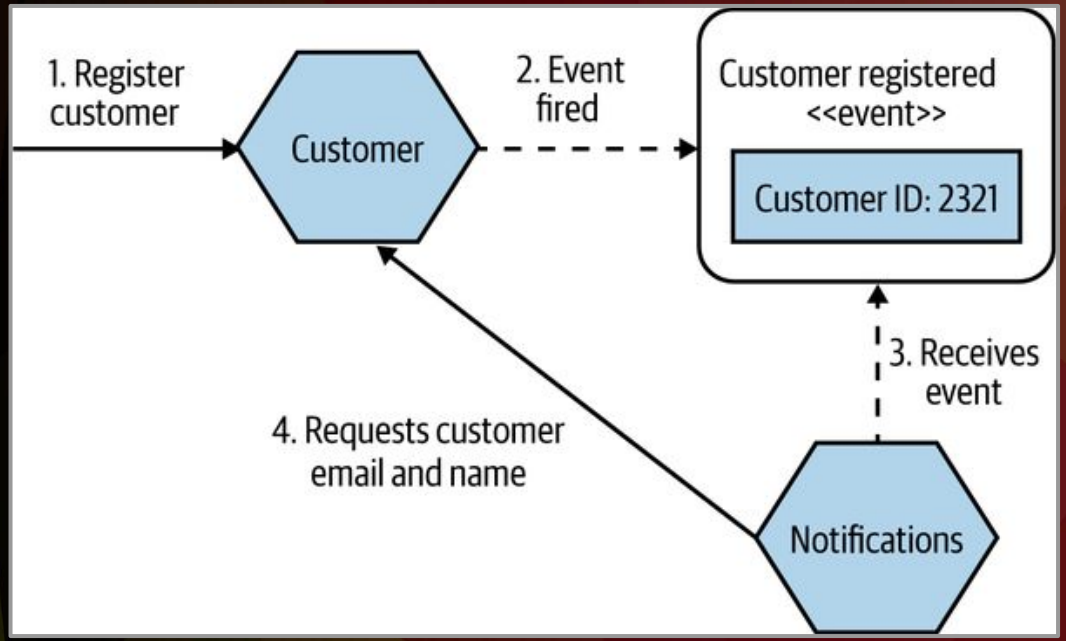
Revisando a Estratégia de Eventos



Um evento sendo transmitido de **Customer**, informando as partes interessadas que um novo cliente se registrou no sistema.

Dois dos microsserviços downstream, Loyalty e Notifications, se importam com esse evento.

O **Loyalty** reage ao receber o evento configurando uma conta para o novo cliente para que ele possa começar a ganhar pontos, enquanto o **Notifications** envia um e-mail ao cliente recém-registrado dando-lhe as boas-vindas.



Uma opção é que o evento contenha apenas um identificador para o cliente recém-registrado.

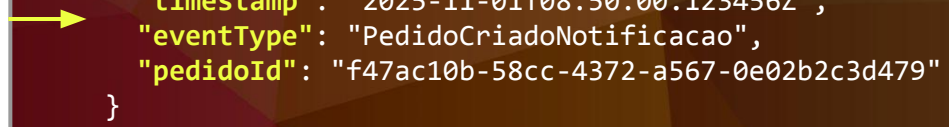
O **Loyalty** precisa apenas desse identificador para criar a conta de fidelidade correspondente, então ele tem todas as informações de que precisa.

No entanto, embora o **Notifications** saiba que precisa enviar um e-mail de boas-vindas quando esse tipo de evento for recebido, ele precisará de informações adicionais para fazer seu trabalho.

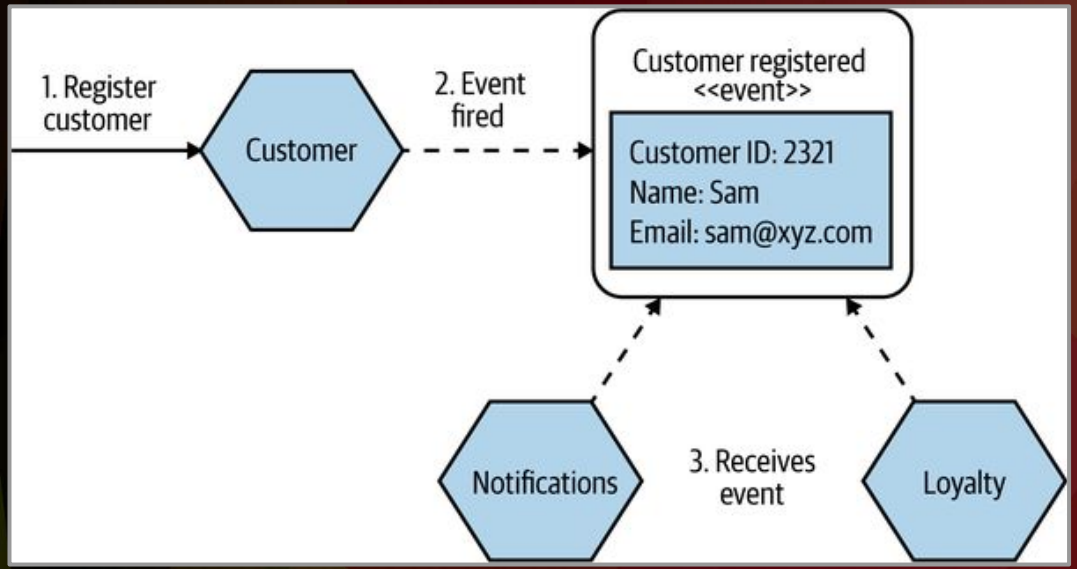
PedidoCriadoEvent.java

Source | History

```java
package br.edu.infnet.pedidos.eventos;

import ...2 lines

// Evento Tipo 1: Apenas notifica a criação e envia o ID
public class PedidoCriadoEvent {

    private final UUID eventId = UUID.randomUUID();
    private final Instant timestamp = Instant.now();
    private final String eventType = "PedidoCriadoNotificacao";

    // O único dado de negócio é o ID do agregado
    private final UUID pedidoId;

    // Construtor
    public PedidoCriadoEvent(UUID pedidoId) {
        this.pedidoId = pedidoId;
    }

    // Getters
    public UUID getEventId() {
        return eventId;
    }

    public Instant getTimestamp() {
        return timestamp;
    }

    public String getEventType() {
        return eventType;
    }

    public UUID getPedidoId() {
```



No protótipo da aula passada ficaria assim



A alternativa é colocar todos os dados necessários em um evento.

Se você deixasse **Notifications** pedir o endereço de e-mail e o nome de um determinado cliente, por que não colocar essas informações no evento em primeiro lugar?

Notifications agora é autossuficiente e capaz de fazer seu trabalho sem precisar se comunicar com o **Customer**.

Na verdade, ele pode nunca precisar saber que o **Customer** existe.

PedidoCriadoPayloadEvent.java

Source | History

```java
package br.edu.infnet.pedidos.eventos;

import ...4 lines

// Evento Tipo 2: Carrega o estado completo necessário ("Fat Event")
public class PedidoCriadoPayloadEvent {

    private final UUID eventId = UUID.randomUUID();
    private final Instant timestamp = Instant.now();
    private final String eventType = "PedidoCriadoPayload";

    // --- Dados completos do Agregado Pedido ---
    private UUID pedidoId;
    private UUID clienteId;
    private BigDecimal valorTotal;
    private List<ItemPayload> itens;

    // Dado "extra" (de outro contexto, ex: Cliente)
    // que é útil para os consumidores (ex: Serviço de Notificação)
    private String clienteEmail;

    // Classe interna para os itens
    public static class ItemPayload {

        private UUID produtoId;
        private String nomeProduto;
        private int quantidade;
        private BigDecimal precoUnitario;

        // Construtor e Getters dos Itens...
    }

    // Construtor e Getters do Evento Principal...
```


```
{
  "eventId": "b4c5d6e7-f8g9-1234-b5c6-d7e8f9g01234",
  "timestamp": "2025-11-01T08:51:00.987654Z",
  "eventType": "PedidoCriadoPayload",
  "pedidoId": "e58bd21c-67ac-4133-b8f9-1a03c3e4f580",
  "clienteId": "c90ab88f-12a3-4b45-8c76-2d34e5f6a7b8",
  "valorTotal": 249.90,
  "clienteEmail": "cliente.feliz@petfriends.com",
  "itens": [
    {
      "produtoId": "p1a2b3c4-...",
      "nomeProduto": "Ração Premium Cães Adultos 15kg",
      "quantidade": 1,
      "precoUnitario": 219.90
    },
    {
      "produtoId": "p5d6e7f8-...",
      "nomeProduto": "Ossinho Dental Care",
      "quantidade": 3,
      "precoUnitario": 10.00
    }
  ]
}
```


No protótipo da aula passada ficaria assim

Google Cloud Pub/Sub - Parte 2

Existem várias **estratégias para converter os Agregados em JSON** para enviar para o Google Pub/Sub e futuramente para trânsito REST.

- Criar DTOs.
- Serialização / Desserialização.
- Anotações Jackson.

	PRÓS	CONTRAS
DTOs	• fácil e direto para classes simples. • fácil verificar o formato JSON apenas olhando o código-fonte.	• pode causar duplicação de estruturas de dados quando o objeto de valor precisa ser usado em mais de um aplicativo ou módulo. • requer testes adicionais para verificar se a serialização está correta em cada módulo separadamente. • você pode escrever um conversor DTO/VO comum e compartilhá-lo com outros módulos, mas os usuários da API precisam estar cientes da conversão e usá-la explicitamente sempre que quiserem serializar ou desserializar mensagens.

PedidoCriadoEventDTO.java

Source | History

```java
package br.edu.infnet.pedidos.eventos;

import ...4 lines

// Este é o DTO. É a classe que será convertida em JSON.
// Nota: Em Java 16+, isso seria idealmente um 'record'.
public class PedidoCriadoEventDTO {

    // --- Metadados do Evento ---
    private UUID eventId;
    private Instant timestamp;
    private String eventType;

    // --- Dados do Payload ---
    private UUID pedidoId;
    private UUID clienteId;
    private BigDecimal valorTotal;
    private String clienteEmail;
    private List<ItemDTO> itens;

    // Construtor principal
    public PedidoCriadoEventDTO(UUID pedidoId, UUID clienteId, BigDecimal valorTotal, String clienteEmail, List<ItemDTO> itens) {...11 lines }

    // DTO aninhado para os itens
    public static class ItemDTO {

        private UUID produtoId;
        private String nomeProduto;
        private int quantidade;
        private BigDecimal precoUnitario;

        // Construtor e Getters para ItemDTO...
    }
```

No protótipo da aula passada ficaria assim

	PRÓS	CONTRAS
Anotações Jackson	• fácil de usar. • permite reutilizar o mapeamento entre módulos internos.	• requer dependência adicional para anotações de Jackson – não aceitável quando exposto a sistemas externos.

PedidoCriadoPayloadEvent.java

Source | History

```java
package br.edu.infnet.pedidos.eventos;

import ...8 lines

// A classe de evento é a mesma, mas agora "decorada" com anotações
// Informa ao Jackson para não incluir campos nulos no JSON
@JsonInclude(JsonInclude.Include.NON_NULL)
public class PedidoCriadoPayloadEvent {

    // Renomeia o campo no JSON
    @JsonProperty("identificador_evento")
    private UUID eventId;

    // Formata o timestamp para um padrão específico
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'", timezone = "UTC")
    private Instant timestamp;

    private String eventType;

    // --- Payload ---
    @JsonProperty("id_pedido") // Renomeia
    private UUID pedidoId;

    @JsonProperty("id_cliente") // Renomeia
    private UUID clienteId;

    // Formata o BigDecimal para ter sempre 2 casas decimais no JSON
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "#.00")
    private BigDecimal valorTotal;

    @JsonProperty("email_contato") // Renomeia
    private String clienteEmail;
```

No protótipo da aula passada ficaria assim

	PRÓS	CONTRAS
Serializador / Desserializador personalizado	• fácil de reutilizar entre módulos – escreva e teste uma vez, use em qualquer lugar. • fácil de distribuir como um módulo Jackson adicional.	• requer implementação de baixo nível (mas é muito mais simples do que o esperado). • mais difícil definir campos obrigatórios/opcionais.

PedidoCriadoPayloadEventSerializer.java

Source | History

```java
    @Override
    public void serialize(PedidoCriadoPayloadEvent event, JsonGenerator gen, SerializerProvider provider)
            throws IOException {

        // Inicia o objeto JSON {
        gen.writeStartObject();
        // Escreve os campos, um por um
        gen.writeStringField("eventId", event.getEventId().toString());
        // O Jackson lida bem com Instant, mas você pode formatar se quiser
        // Para usar o formato padrão (ISO-8601 string):
        provider.defaultSerializeField("timestamp", event.getTimestamp(), gen);
        gen.writeStringField("eventType", event.getEventType());
        gen.writeStringField("pedidoId", event.getPedidoId().toString());
        gen.writeStringField("clienteId", event.getClienteId().toString());
        gen.writeStringField("clienteEmail", event.getClienteEmail());
        // Escreve um número BigDecimal
        gen.writeNumberField("valorTotal", event.getValorTotal());
        // Inicia o array "itens" [
        gen.writeArrayFieldStart("itens");
        for (ItemPayload item : event.getItens()) {
            // --- Serialização manual do sub-objeto ---
            gen.writeStartObject();
            gen.writeStringField("produtoId", item.getProdutoId().toString());
            gen.writeStringField("nomeProduto", item.getNomeProduto());
            gen.writeNumberField("quantidade", item.getQuantidade());
            gen.writeNumberField("precoUnitario", item.getPrecoUnitario());
            gen.writeEndObject();
            // --- Fim da serialização manual ---
        }
        // Fecha o array ]
        gen.writeEndArray();
        // Fecha o objeto JSON }
        gen.writeEndObject();
```

No protótipo da aula passada ficaria assim

PedidoCriadoPayloadEventDeserializer.java

Source | History

```java
    @Override
    public PedidoCriadoPayloadEvent deserialize(JsonParser jp, DeserializationContext ctxt)
      throws IOException, JsonProcessingException {
        // Lê a árvore JSON inteira
        JsonNode node = jp.getCodec().readTree(jp);
        // Extrai os campos do nó JSON
        UUID eventId = UUID.fromString(node.get("eventId").asText());
        Instant timestamp = Instant.parse(node.get("timestamp").asText());
        String eventType = node.get("eventType").asText();
        UUID pedidoId = UUID.fromString(node.get("pedidoId").asText());
        UUID clienteId = UUID.fromString(node.get("clienteId").asText());
        String clienteEmail = node.get("clienteEmail").asText();
        BigDecimal valorTotal = node.get("valorTotal").decimalValue();
        // Extrai o array de "itens"
        List<ItemPayload> itens = new ArrayList<>();
        JsonNode itensNode = node.get("itens");
        if (itensNode.isArray()) {
            for (JsonNode itemNode : itensNode) {
                // Extrai os campos de cada item
                UUID produtoId = UUID.fromString(itemNode.get("produtoId").asText());
                String nomeProduto = itemNode.get("nomeProduto").asText();
                int quantidade = itemNode.get("quantidade").asInt();
                BigDecimal precoUnitario = itemNode.get("precoUnitario").decimalValue();

                // Cria o objeto ItemPayload
                itens.add(new ItemPayload(produtoId, nomeProduto, quantidade, precoUnitario));
            }
        }
        // Cria o objeto principal usando o construtor "cheio"
        return new PedidoCriadoPayloadEvent(
            eventId, timestamp, eventType, pedidoId, clienteId,
            valorTotal, clienteEmail, itens
        );
```

Modelo de Arquitetura Pet Friends: E-Commerce, Almoxarifado, Transporte

Pet Friends

Atendimento ao Cliente

Atendimento ao Franqueado

Produtos

Serviços Agendados

Venda

Assinatura de Ração

Veterinário

Tosa e Banho

Passeio

Produtos

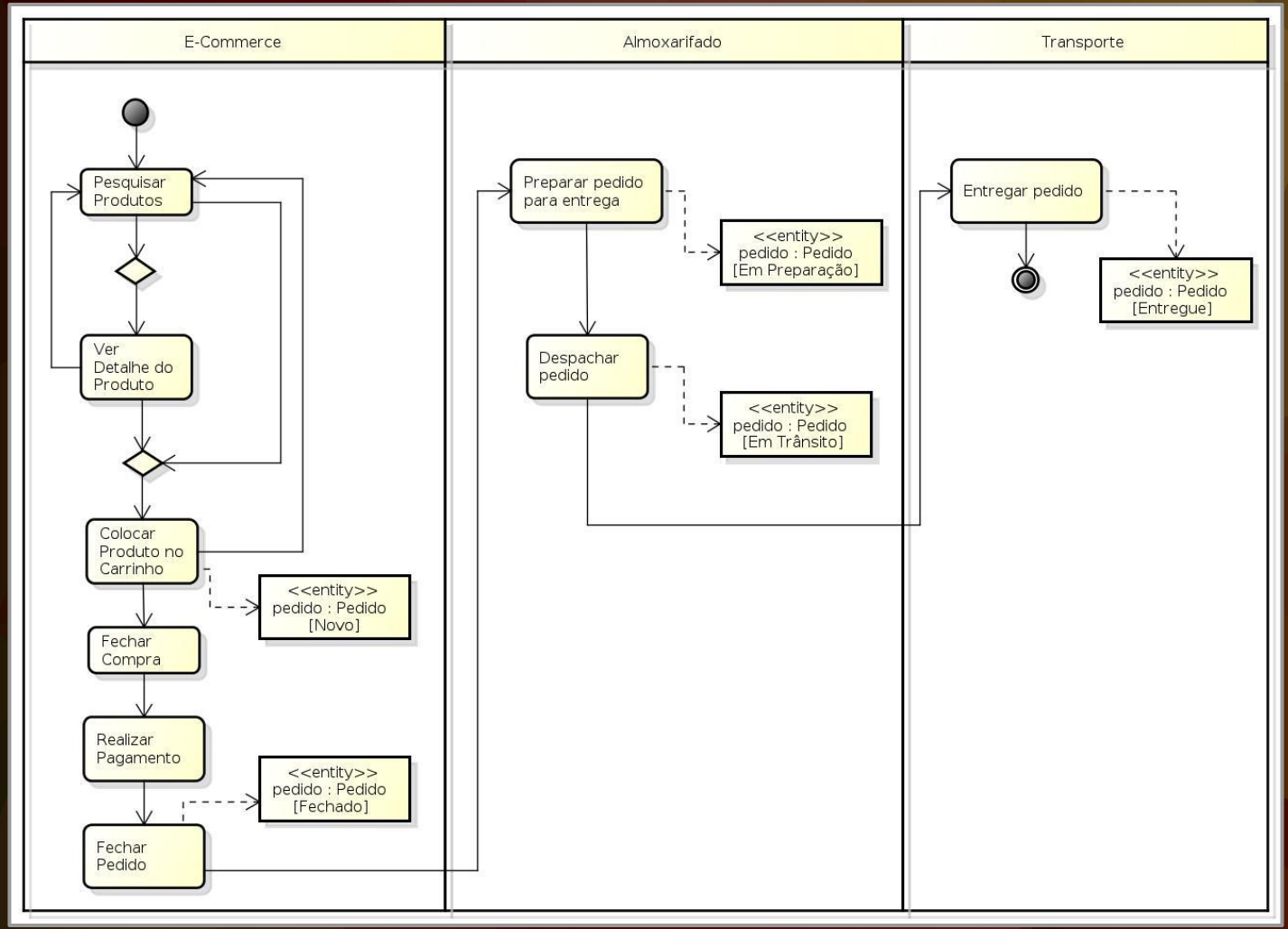
A venda de produtos segue o modelo de e-commerce onde o cliente pode encontrar os grandes distribuidores credenciados e seus produtos.

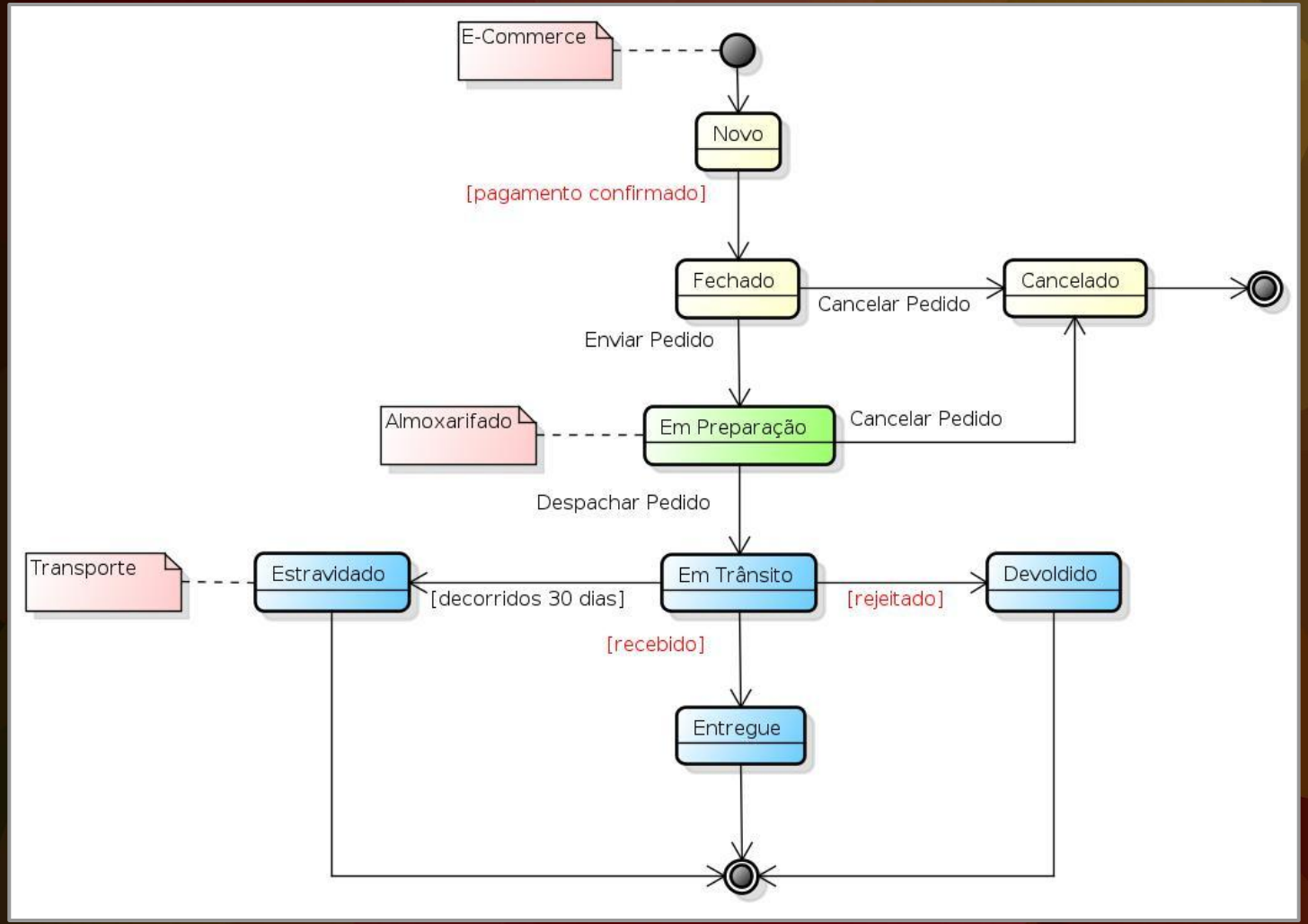
Os diversos produtos serão ditados pela matriz da **Pet Friends**.

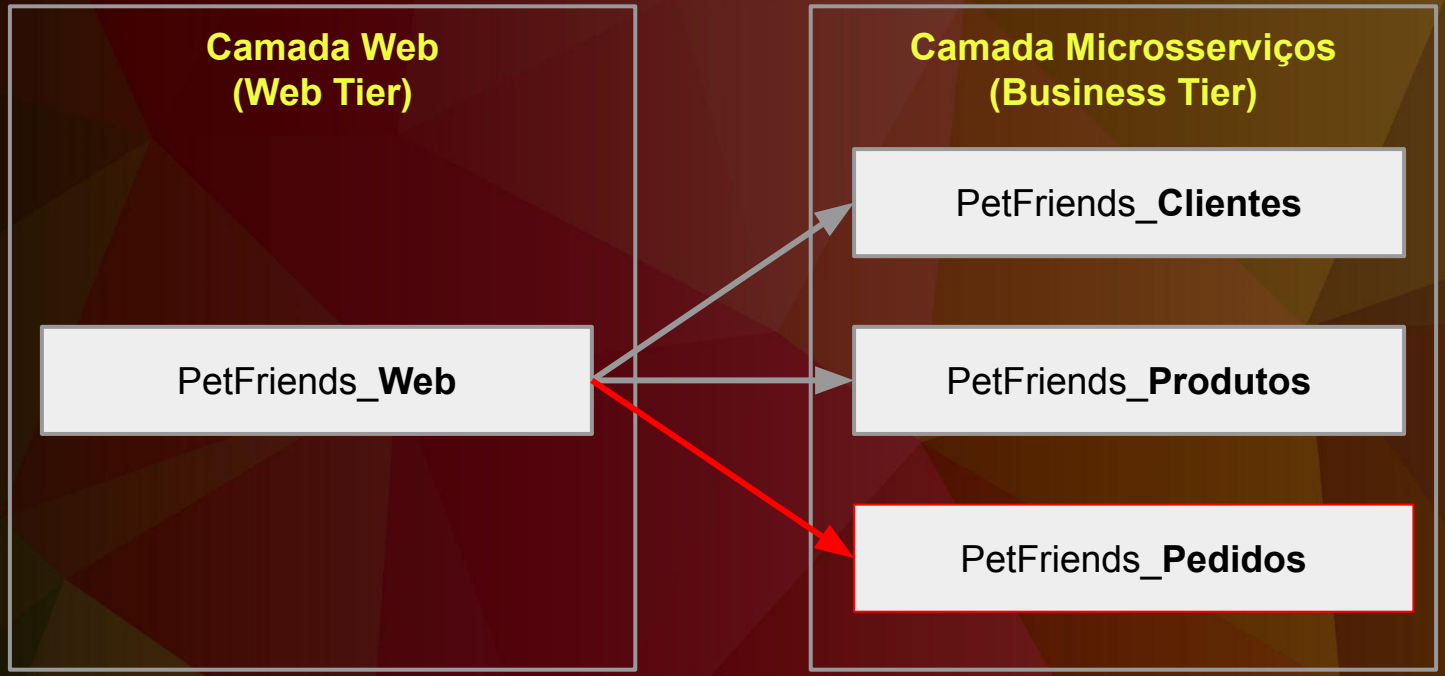
Podemos ter tipos de produtos categorizados como higiene, diversão e farmácia.

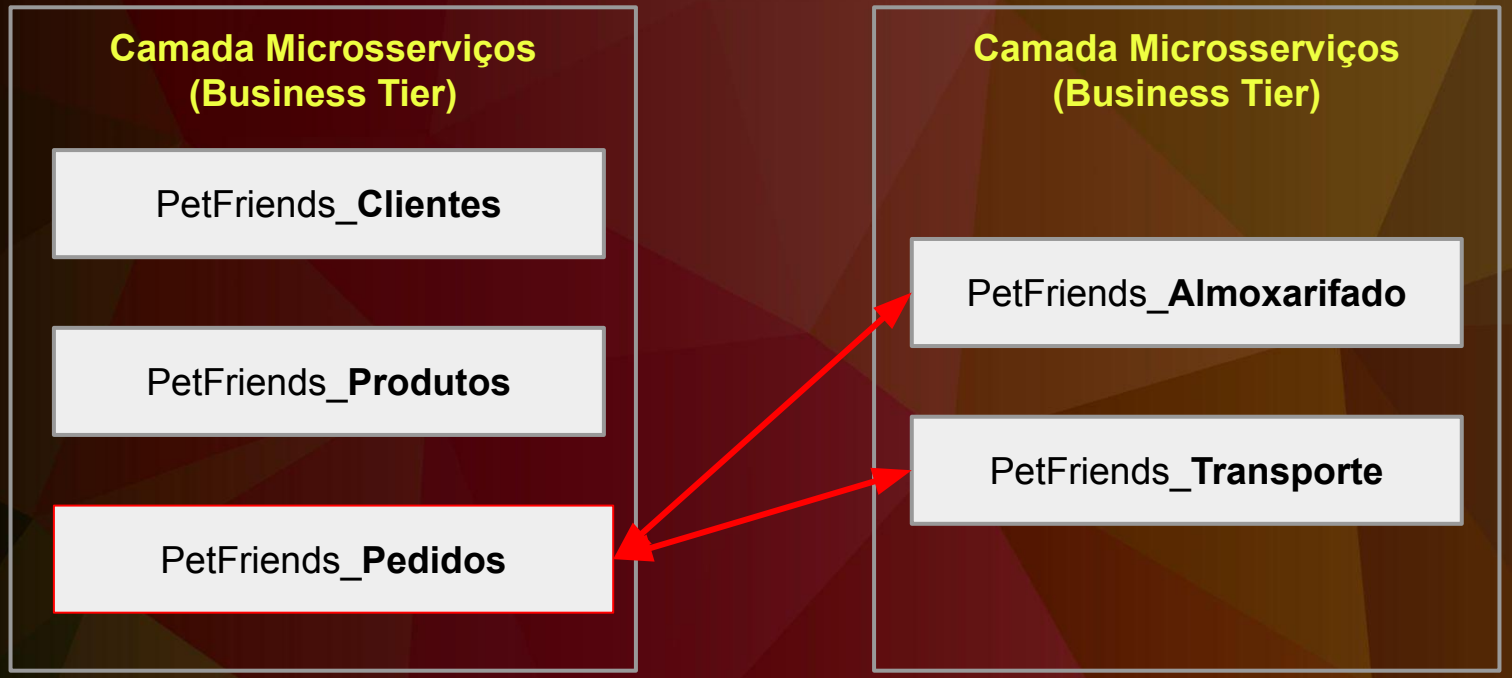
O usuário final poderá manter uma bolsa de produtos e ditar a frequência com que o mesmo deverá ser entregue em sua residência.

A loja da região do cliente é quem faz a entrega ou permite a retirada.


E-Commerce
Almoxarifado
Transporte
Pesquisar Produtos
Ver Detalhe do Produto
Colocar Produto no Carrinho
<<entity>> pedido : Pedido [Novo]
Fechar Compra
Realizar Pagamento
<<entity>> pedido : Pedido [Fechado]
Fechar Pedido
Preparar pedido para entrega
<<entity>> pedido : Pedido [Em Preparação]
Despachar pedido
<<entity>> pedido : Pedido [Em Trânsito]
Entregar pedido
<<entity>> pedido : Pedido [Entregue]


E-Commerce
Novo
[pagamento confirmado]
Fechado
Cancelado
Cancelar Pedido
Enviar Pedido
Almoxarifado
Em Preparação
Cancelar Pedido
Despachar Pedido
Transporte
Estravidado
Em Trânsito
Devoldido
[decorridos 30 dias]
[rejeitado]
[recebido]
Entregue


Camada Web
(Web Tier)
PetFriends_Web
Camada Microsserviços
(Business Tier)
PetFriends_Clientes
PetFriends_Produtos
PetFriends_Pedidos


Camada Microsserviços
(Business Tier)
PetFriends_Clientes
PetFriends_Produtos
PetFriends_Pedidos
Camada Microsserviços
(Business Tier)
PetFriends_Almoxarifado
PetFriends_Transporte