

Prof. Evandro César Freiburger

# **Estudo de Caso - Produção e Entrega de Refeições**

Brasil

3 de julho de 2024

# Sumário

1	Introdução . . . . .	2
2	Requisitos do Software Fornecido . . . . .	3
2.1	Visão Geral . . . . .	3
2.2	Backlog do Software . . . . .	4
2.3	Diagrama de Casos de Uso . . . . .	7
2.4	Refinamento das Entidades do Modelo de Domínio . . . . .	7
2.5	Entidades do Modelo de Domínio . . . . .	15
3	Modelo Arquitetural do Software Fornecido . . . . .	16
3.1	Arquivo POM do Projeto . . . . .	16
3.2	Classes DTOs . . . . .	18
3.3	Classes de Entidades . . . . .	20
3.4	Classes de Negócio . . . . .	22
3.5	Classes de Persistência (DAOs) . . . . .	24
3.6	Classes Utilitária FabricaEntityManager . . . . .	26
4	Execução do Estudo de Caso . . . . .	28
4.1	Como Desenvolver as Atividades . . . . .	28
5	Entrega e Apresentação . . . . .	28
5.1	Entrega dos Artefatos . . . . .	28
5.2	Apresentação . . . . .	28
5.3	Crterios de Avaliao . . . . .	29

# 1 Introdução

Este documento tem o objetivo descrever e especificar o Estudo de Caso prático da disciplina de Testes Automatizados de Software do curso de Pós-Graduação *Lato Sensu* em Engenharia DevOps.

O estudo de caso tem como objetivo principal a implementação de testes automatizado para um sistema hipotético proposto. Para isso, serão fornecidos os seguintes elementos do software: documentos de especificação de requisitos, diagramas em UML, especificação de estrutura de dados das entidades do software, regras de negócio e código-fonte implementado em linguagem Java.

A partir dos elementos de software fornecidos, serão produzidos testes automatizados que cobrirão os níveis: testes unitários, testes de integração, testes de sistema e testes de performance.

Este documento está organizado nas seguintes seções, além desta introdução: 2 - Requisitos do software fornecido - que apresenta uma visão geral do sistema, o Backlog do produto, as especificações de entidades de dados e modelos UML; 3 - Modelo Arquitetural do Software Fornecido - apresenta o modelo arquitetural utilizado na implementação do software fornecido, descrevendo os elementos e as informações mais relevantes para o entendimento do software; 4 - Execução do Estudo de Caso - especifica o que deve ser implementado de testes automatizados no software fornecido; 5 - Entrega e Apresentação - especifica o que deve ser entregue, prazo, formas de entrega e critérios de avaliação.

## 2 Requisitos do Software Fornecido

Nesta seção serão apresentados os requisitos do software fornecido e que serão utilizados como base para o desenvolvimento do estudo de caso.

### 2.1 Visão Geral

Esta subseção descreve a visão geral de um sistema de gestão de vendas e entregas de refeições de um restaurante fictício. Para que os pedidos de refeições possam ser realizados é necessário uma série de informações iniciais que dão suporte ao processo de pedido, produção e entrega.

Devem existir funcionalidades no sistema que permitam manutenção do cadastro de produtos, manutenção de grupos alimentares e tipos de preparos dos alimentos. Os clientes terão acesso as funcionalidades que lhes permitirão gerir os dados do seu próprio perfil de usuário do sistema, por meio de um módulo *mobile* que ele pode baixar em seu celular. Perfis de usuário internos do sistema serão geridos pelo administrador do sistema, cadastrando e configurando suas respectivas credenciais.

Os pedidos são realizados diretamente pelo cliente, por meio do aplicativo *mobile*, onde este escolhe uma refeição a partir de um cardápio diário oferecido pelo restaurante. O cardápio é criado e preenchido pelo chefe de cozinha, contemplando mais de uma opção de grupos alimentares, tais como: proteínas, cereais, legumes, verduras. As porções de cada item do cardápio sempre serão oferecidas em unidades de 50 gramas e o cliente pode escolher qualquer combinação e quantidades de frações, para compor seu pedido, contidas no cardápio do dia.

Os pedidos realizados pelos clientes devem ser inseridos em uma fila cronológica com o status pendente. Essa fila será visualizada pelos agentes de produção, que escolhem o pedido da vez e muda seu estado para produção, indicando que o pedido está em produção. Quando o agente de produção termina a produção de um pedido, este mudará seu estado para pronto, liberando o pedido para a entrega.

O gerente de produção deve ter acesso a uma fila de pedidos organizados de forma cronológica de produção e que estejam com estado pronto. Assim, o gerente de produção organizará as entregas dos pedidos prontos, mudando seu estado para entrega, quando esses forem despachados. Os pedidos são despachados para entregadores cadastrados no sistema, para que haja um controle de qualidade e pontualidade das entregas. Quando o entregador finaliza a entrega, este deve sinalizar em módulo *mobile* de entrega que o pedido foi entregue. O cliente deve possuir uma funcionalidade em seu módulo *mobile* que lhe permite avaliar o pedido recebido, em questões relacionadas ao pedido (conteúdo, sabor, temperatura, embalagem, etc) e em relação à entrega.

O chefe de cozinha deve planejar e definir os cardápios diários do restaurante. No dia que um cardápio vai ser colocado em produção, o chefe de cozinha deve fazer um procedimento de retirada de unidades de estoque dos produtos, considerando que o controle de estoque será realizado por frações de 50 gramas. Esse procedimento subtrairá essas unidades do estoque de cada produto. Isso significa que o chefe de cozinha pode criar os cardápios com antecedência, mas deve realizar um procedimento no sistema que indicará que determinado cardápio será colocado em produção naquele dia, indicando a quantidade de unidades de cada item que será produzida. Isso fará com que as respectivas unidades sejam subtraídas dos estoques dos produtos.

O gerente de estoque deve consultar o estoque dos produtos e obter uma relação de produtos que estejam com estoque abaixo da margem de segurança, para que este possa realizar pedidos aos fornecedores. Quando uma compra de produtos é recebida pelo gerente de

estoque, este deverá registrar a entradas das unidades/frações adquiridas para que o estoque seja atualizado.

O gerente de estoque também registrará baixas de estoque de produtos decorrentes de motivos que inviabilizem a utilização dos produtos, tais como: vencimento, quebra, mal estado de conservação. Esse procedimento deverá dar baixa às respectivas quantidades de estoque, e registrar a data e motivo pelo qual essas quantidades foram baixadas.

O valor cobrado em um pedido será calculado pelo valor de custo de cada item inserido no pedido, acrescido da taxa de entrega, obtida de uma planilha de valores de custo de entrega por bairro. O custo de cada item será formado pelo valor de custo do item, mais o valor de preparo, conforme o tipo de preparo estabelecido no cardápio, além de um percentual de valores fixos que representam custos como: água, energia, mão de obra, impostos, etc.

O gerente-geral do restaurante precisa de uma série de consultas para auxiliar em suas tomas de decisões, tais como:

- Quantidade de refeições produzidas por dia e com totalização mensal;
- Quantidade de produtos descartados;
- Produtos mais consumidos na produção das refeições;
- Produtos menos consumidos na produção de refeições;
- Tempo médio entre o recebimento de pedidos e o término da produção;
- Tempo médio entre a finalização da produção de um pedido e sua entrega ao cliente;

Como visão inicial do sistema, espera-se ter um módulo *mobile* para que o cliente e o entregador usem para realizar o pedido e as entregas respectivamente. Também vislumbra-se um módulo *desktop* para ser usado em um monitor *touch screen* para ser usado na produção e um módulo Web para ser usado nas configurações e consultas.

## 2.2 Backlog do Software

Nesta subseção são apresentadas as Users Stories produzidas para o software fornecido, que será usado como base do estudo de caso.

**US01** - Como Gerente de Produção, quero fazer a gestão do cadastro de produtos que são usados para produzir as refeições, para que esses possam ser inseridos como itens de futuros cardápios, produções e pedidos.

**US02** - Como Gerente de Produção, quero fazer a gestão do cadastro de grupos alimentares, para que esses possam ser usados para categorizar os produtos.

**US03** - Como Gerente de Produção, quero fazer a gestão do cadastro de tipos de preparo de alimentos, para que esses possam ser usados para caracterizar o custo de preparo de cada item de cardápio.

**US04** - Como Chefe de Cozinha, quero fazer gestão de cardápios de refeições, para que estes sejam usados na produção de refeições.

**US05** - Como Cliente, quero realizar a gestão dos dados do meu perfil no aplicativo, para que eu possa realizar pedidos de refeições.

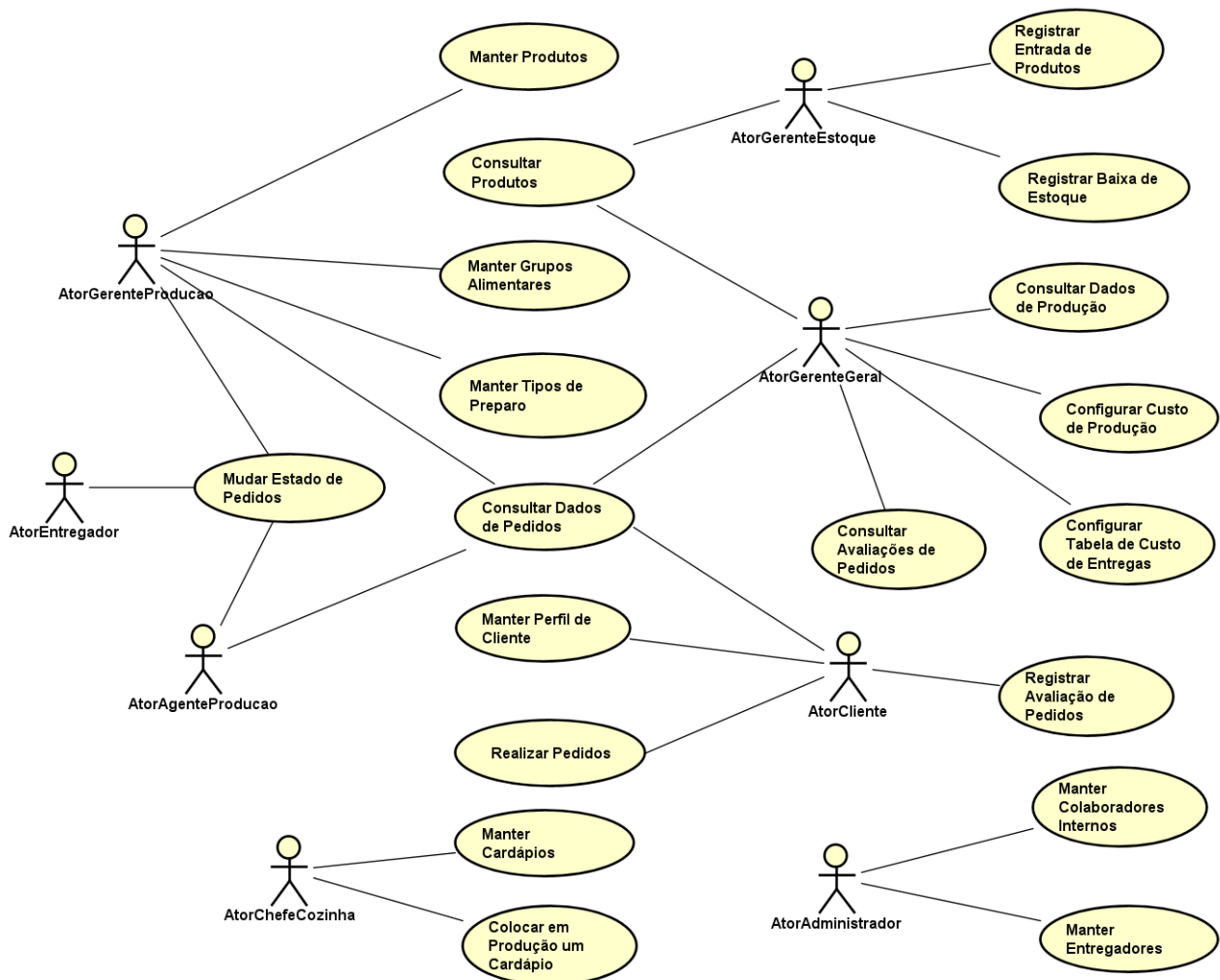
- US06** - Como Cliente, quero realizar um pedido de refeição a partir do cardápio do dia, para que eu possa receber a refeição no meu endereço de entrega.
- US07** - Como Cliente, quero acompanhar o estado do meu pedido, para saber em que situação ele está.
- US08** - Como Agente de Produção, quero consultar os pedidos do dia em ordem cronológica de solicitação e estado Pendente, para que eu possa colocá-lo em produção.
- US09** - Como Agente de Produção, quero mudar o estado de um pedido de Produção para Pronto, para que ele possa ser escalado para entrega.
- US10** - Como Gerente de Produção, quero consultar os pedidos do dia que estejam com estado Pronto, para que eu possa despachar para entrega.
- US11** - Como Chefe de Cozinha, quero escolher uma ordem de produção e colocá-la em produção no dia, para seja realizada as retiradas de unidades/frações do estoque dos produtos.
- US12** - Como Gerente de Estoque, quero consultar produtos que estejam abaixo do estoque de segurança, para que eu possa fazer pedido aos fornecedores.
- US13** - Como Gerente de Estoque, quero registrar entrada de unidades/frações de produto ao estoque.
- US14** - Como Gerente de Estoque, quero registrar a baixa de unidades/frações de produtos do estoque que serão descartados, para que eu possa manter o estoque de produtos atualizado.
- US15** - Como Gerente-Geral, quero consultar a produção de refeições por dia com totalização mensal, para que eu possa fazer estimativas e projeções futuras.
- US16** - Como Gerente-Geral, quero consultar as quantidades de produtos descartados por um período, para que eu possa tomar decisões para a diminuição dos descartes.
- US17** - Como Gerente-Geral, quero consultar quais são os produtos que mais foram produzidos em unidades, para que possa criar estratégias de compras mais eficientes.
- US18** - Como Gerente-Geral, quero consultar quais são os produtos que menos foram produzidos em unidades, para que possa analisar a possibilidade de deixar de produzir tais produtos.
- US19** - Como Gerente-Geral, quero consultar o tempo médio entre o recebimento de um pedido do cliente e o término da produção, para que eu possa criar estimativas de tempo de produção.
- US20** - Como Gerente-Geral, quero configurar um percentual de custo fixo de produção, para ser aplicado na composição dos valores dos pedidos produzidos.
- US21** - Como Gerente-Geral, quero consultar o tempo médio entre o término da produção de uma refeição e a finalização da entrega ao cliente, para que eu possa criar estimativas de tempo de entrega.
- US22** - Como Gerente-Geral, quero configurar uma tabela de custo de entrega por bairro, para que seja usado na composição do valor dos pedidos produzidos.
- US23** - Como Administrador, quero gerir o cadastro dos entregadores de pedidos, para que as entregas sejam rastreáveis por entregador.

- US24** - Como Administrador, quero gerir o cadastro dos colaboradores internos do restaurante, para estabelecer o perfil de usuário de cada papel a ser desempenhado.
- US25** - Como Entregador, quero registrar a finalização da entrega de um pedido, para que haja um controle de tempo de entrega de cada pedido.
- US26** - Como Cliente, quero uma funcionalidade que permita o registro de avaliação do pedido recebido, para que o restaurante tenha dados para as avaliações de qualidade da produção e entrega das refeições fornecidas.

## 2.3 Diagrama de Casos de Uso

Nesta subseção é apresentado um diagrama de Casos de Uso do software fornecido, embora não foram usadas as especificações de casos de uso, o diagrama foi produzido para complementar, de forma gráfica, a relações entre os atores do sistema e as funcionalidades descritas nas Users Stories. Veja o diagrama na Figura 1.

Figura 1 – Diagrama de Casos de Uso



## 2.4 Refinamento das Entidades do Modelo de Domínio

Nesta subseção são apresentados os detalhes das entidades do domínio de problema. Para cada entidade são especificados os atributos, tipo dos atributos, formato, validação e a informação se o atributo é chave primário. Além dessas informações também foi especificado para cada entidade as regras de negócio a ela associada. As relações entre entidades, quer sejam por atributos ou por entidades associativas não foram estabelecidas, por fazerem parte do exercício.



**Nome da Entidade:** Cardapio

**Nome da Tabela:** cardapio

Campo	Tipo	Formato	Validação	Chave
codigo	Inteiro	-	> 0	PK
nome	String (50)	-	Não pode ser nulo	
descricao	String (250)	-		
listaPreparoProduto	PreparoProduto[]	-	Não pode ser vazia	FK (Mx1)

Regras de Negócio:

- Não deve existir mais de um cardápio com o mesmo nome
- Um cardápio válido deve ter, pelo menos, um item vinculado

**Nome da Entidade:** Cliente

**Nome da Tabela:** cliente

Campo	Tipo	Formato	Validação	Chave
codigo	Inteiro	-	> 0	PK
nome	String	-	Não pode ser nulo e deve ser menor três caracteres	
RG	String	-	Não pode ser nulo e nem vazio	
CPF	String	-	Não pode ser nulo e pelo menos 11 caracteres	
telefone	String	-	Não pode ser nulo e pelo menos 8 caracteres	
logradouro	String	-	Não pode ser nulo e nem vazio	
numero	String	-	Não pode ser nulo e nem vazio	
bairro	String	-	Não pode ser nulo e nem vazio	
pontoReferencia	String	-	Não pode ser nulo	

Regras de Negócio:

- Não pode existir dois clientes com o mesmo CPF
- Não pode ser excluído, caso tenha pedidos em seu nome

**Nome da Entidade:** Entregador

**Nome da Tabela:** entregador

Campo	Tipo	Formato	Validação	Chave
codigo	Inteiro	-	> 0	PK
nome	String	-	Não pode ser nulo e deve ser menor três caracteres	
RG	String	-	Não pode ser nulo e nem vazio	
CPF	String	-	Não pode ser nulo e pelo menos 11 caracteres	
telefone	String	-	Não pode ser nulo e pelo menos 8 caracteres	

Regras de Negócio:

- Não pode existir dois entregadores com o mesmo CPF
- Não pode ser excluído, caso tenha pedidos entregues por ele

**Nome da Entidade:** GrupoAlimentar

**Nome da Tabela:** grupo\_alimentar

Campo	Tipo	Formato	Validação	Chave
codigo	Inteiro	-	> 0	PK
nome	Caractere (40)	Letras	Tamanho >= 3 caracteres	

Regras de Negócio:

- Não deve existir dois grupos alimentares com o mesmo nome.
- Não deve ser excluído um grupo referenciado por algum produto.

**Nome da Entidade:** ItemOrdemProducao

**Nome da Tabela:** item\_ordem\_producao

Campo	Tipo	Formato	Validação	Chave
codigo	Inteiro	-	> 0	PK
quantidadePorcao	Inteiro	-	> 0	
preparoProduto	PreparoProduto	-	Não pode ser nulo	FK (Mx1)

Regras de Negócio:

- Deve ser excluído juntamente a OrdemProducao

**Nome da Entidade:** ItemPedido

**Nome da Tabela:** item\_pedido

Campo	Tipo	Formato	Validação	Chave
codigo	Inteiro	-	> 0	PK
quantidadePorcao	Inteiro	-	> 0	
preparoProduto	PreparoProduto	-	Não pode ser nulo	FK (Mx1)

Regras de Negócio:

- Deve ser excluído juntamente ao Pedido

**Nome da Entidade:** OrdemProducao

**Nome da Tabela:** ordem\_producao

Campo	Tipo	Formato	Validação	Chave
codigo	Inteiro	-	> 0	PK
dataProducao	Data	-	Não pode ser nulo e deve ser menor ou igual a data corrente	
cardapio	Cardapio		Não pode ser nulo	FK(Mx1)
listaItens	ItemOrdemProducao[]	-	Não pode ser nulo	FK (1xM)
estado	EstadoOrdemProducao		Valores: REGISTRADA, PROCESSADA	

Regras de Negócio:

- Inicia no estado REGISTRADA
- Processamento muda o estado para PROCESSADA e retira as respectivas quantidades de unidades do estoque dos produtos relacionados.
- Não pode ser excluída depois de processada.

**Nome da Entidade:** Pedido

**Nome da Tabela:** pedido

Campo	Tipo	Formato	Validação	Chave
codigo	Inteiro	-	> 0	PK
dataPedido	Data	-	Não pode ser nulo e deve ser igual a data corrente	
horaPedido	Hora (time)		Não pode ser nulo e deve ser menor que a hora corrente	
cliente	Cliente	-	Não pode ser nulo	FK(Mx1)
horaProducao	Hora (time)		Se não for nulo, deve ser menor que a hora do pedido	
horaPronto	Hora (time)		Se não for nulo, deve ser menor que a hora de	

			producao	
horaEntrega	Hora (time)		Se não for nulo, deve ser menor que a hora de pronto	
horaFinalizado	Hora (time)		Se não for nulo, deve ser menor que a hora de entrega	
listaItens	ItemPedido[]	-	Não pode ser nulo	FK (1xM)
entregador	Entregador		Nos estados ENTREGA E CONCLUIDO, não pode ser nulo	FK (Mx1)
estado	EstadoOrdemProducao		Valores: REGISTRADO, PRODUCAO, PRONTO, ENTREGA, CONCLUIDO	

Regras de Negócio:

- Inicia no estado REGISTRADO
- Ir para produção muda o estado para PRODUCAO
- Término da produção muda o estado para PRONTO
- Sair para entrega muda o estado para ENTREGA
- Finalizar a entrega muda o estado para CONCLUIDO
- Não pode ser excluído depois que estiver no estado PRODUCAO

**Nome da Entidade:** PreparoProduto

**Nome da Tabela:** preparo\_produto

Campo	Tipo	Formato	Validação	Chave
codigo	Inteiro	-	> 0	PK
nome	Caractere (40)	Letras	Tamanho >= 3 caracteres <= 40	
produto	Produto	-	Não pode ser nulo	FK (Mx1)
tipoPreparo	TipoPreparo	-	Não pode ser nulo	FK (Mx1)
tempoPreparo	Inteiro	-	> 0	

valorPreparo	Float	-	> 0	
--------------	-------	---	-----	--

Regras de Negócio:

- Não deve existir mais de um PreparoProduto (Produto/TipoPreparo).

**Nome da Entidade:** Produto

**Nome da Tabela:** produto

Campo	Tipo	Formato	Validação	Chave
codigo	Inteiro		> 0	PK
nome	Caractere (50)		Tamanho > 3 caracteres	
custoUnidade	Float		> 0	
valorEnergetico	Inteiro		>= 0	
estoque	Inteiro		>= 0	
grupoAlimentar	Inteiro		Não nulo	FK (Mx1)

Regras de Negócio:

- Não deve existir dois produtos com o mesmo nome.

**Nome da Entidade:** TipoPreparo

**Nome da Tabela:** tipo\_preparo

Campo	Tipo	Formato	Validação	Chave
codigo	Inteiro	-	> 0	PK
descricao	Caractere (50)	Letras	Tamanho >= 3 caracteres	

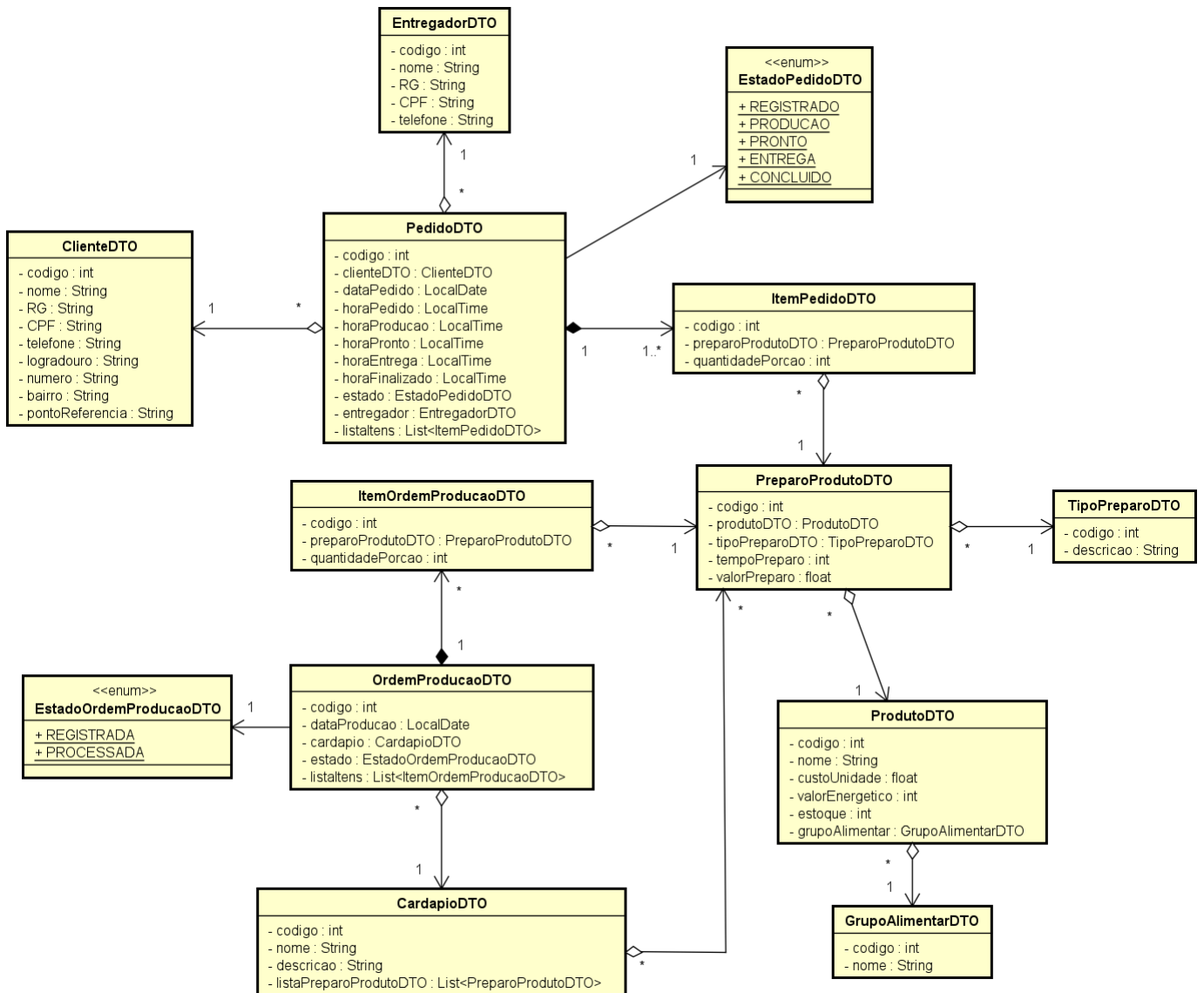
Regras de Negócio:

- Não deve existir mais de um TipoPreparo com a mesma descrição.
- Não deve ser excluído um TipoPreparo referenciado por algum PreparoProduto.

## 2.5 Entidades do Modelo de Domínio

A Figura 2 apresenta um modelo de classes que representam o modelo de domínio do software fornecido. Também é possível obter as relações entre as entidades e suas respectivas multiplicidades.

Figura 2 – Modelo Conceitual do Software

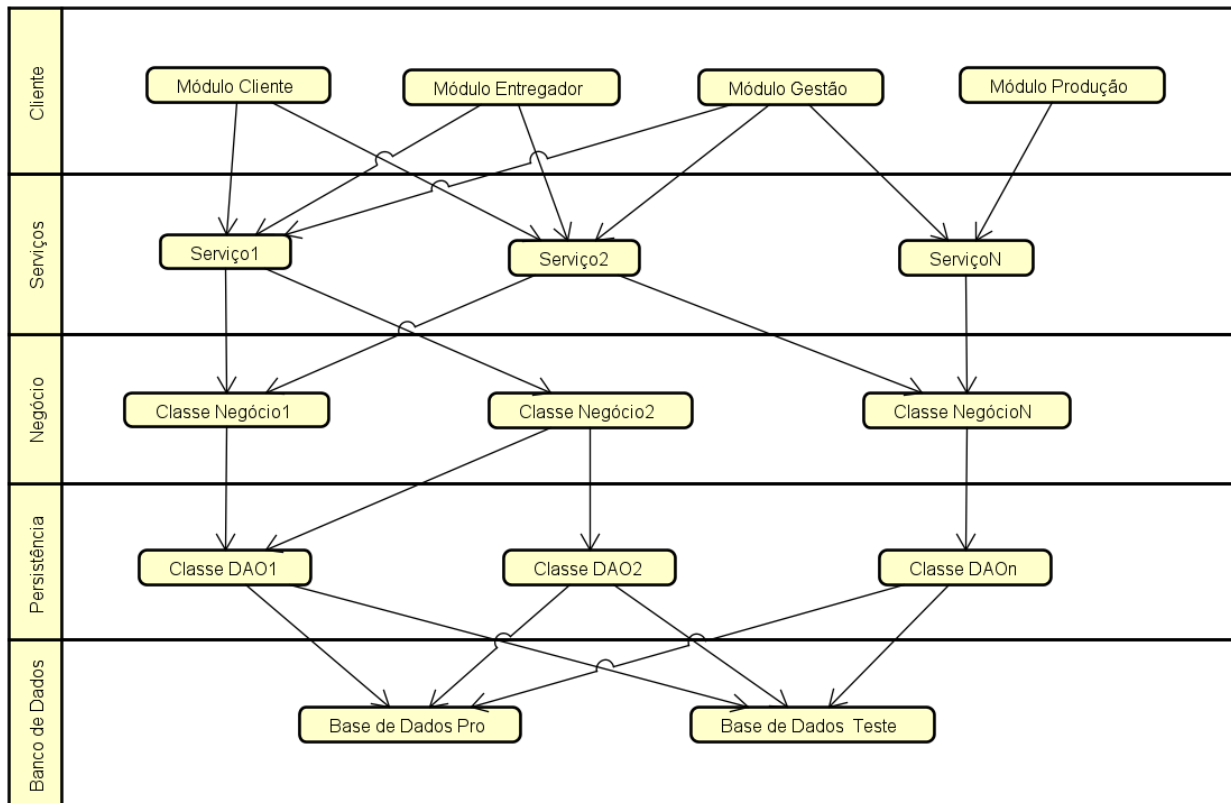




### 3 Modelo Arquitetural do Software Fornecido

Nesta seção são apresentadas informações que subsidiam o entendimento sob a arquitetura do software fornecido. A principal informação é descrever as responsabilidades dos grupo de elementos de software encontrados no projeto implementado. A Figura 3 apresenta uma representação abstrata dos elementos que compõem a arquitetura do software em termos de abstrações (camadas).

Figura 3 – Representação Abstrata da Arquitetura do Software



#### 3.1 Arquivo POM do Projeto

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <project xmlns="http://maven.apache.org/POM/4.0.0"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven
6     -4.0.0.xsd">
7   <modelVersion>4.0.0</modelVersion>
8
9   <groupId>ifmt.cba</groupId>
10  <artifactId>restaurante-base</artifactId>
11  <version>1.0-SNAPSHOT</version>
12
13  <name>restaurante-base</name>
14
15  <properties>
16    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17    <java.version>19</java.version>
18    <junit.version>5.10.0</junit.version>
19    <maven.compiler.source>19</maven.compiler.source>
20    <maven.compiler.target>19</maven.compiler.target>
21    <maven.compiler.plugin.version>3.11.0</maven.compiler.plugin.version>
22    <maven.surefire-plugin.version>3.2.1</maven.surefire-plugin.version>
23    <skipTests>>false</skipTests>

```

```

23 <modelmapper.version>3.1.1</modelmapper.version>
24 <commons-lang3.version>3.13.0</commons-lang3.version>
25 <h2.version>2.2.224</h2.version>
26 <postgresql.version>42.6.0</postgresql.version>
27 <hibernate-core.version>6.2.1.Final</hibernate-core.version>
28 </properties>
29
30 <dependencies>
31 <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter -->
32 <dependency>
33 <groupId>org.junit.jupiter</groupId>
34 <artifactId>junit-jupiter</artifactId>
35 <version>${junit.version}</version>
36 <scope>test</scope>
37 </dependency>
38
39 <!-- https://mvnrepository.com/artifact/org.modelmapper/modelmapper -->
40 <dependency>
41 <groupId>org.modelmapper</groupId>
42 <artifactId>modelmapper</artifactId>
43 <version>${modelmapper.version}</version>
44 </dependency>
45
46 <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->
47 <dependency>
48 <groupId>org.apache.commons</groupId>
49 <artifactId>commons-lang3</artifactId>
50 <version>${commons-lang3.version}</version>
51 </dependency>
52
53 <!-- https://mvnrepository.com/artifact/com.h2database/h2 -->
54 <dependency>
55 <groupId>com.h2database</groupId>
56 <artifactId>h2</artifactId>
57 <version>${h2.version}</version>
58 <scope>runtime</scope>
59 </dependency>
60
61 <!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
62 <dependency>
63 <groupId>org.postgresql</groupId>
64 <artifactId>postgresql</artifactId>
65 <version>${postgresql.version}</version>
66 </dependency>
67
68 <!-- https://mvnrepository.com/artifact/org.hibernate.orm/hibernate-core -->
69 <dependency>
70 <groupId>org.hibernate.orm</groupId>
71 <artifactId>hibernate-core</artifactId>
72 <version>${hibernate-core.version}</version>
73 </dependency>
74 </dependencies>
75
76 <build>
77 <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be
78 moved to
79 parent pom) -->
80 <plugins>
81 <!-- clean lifecycle, see
82 https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean_Lifecycle -->
83 <plugin>
84 <artifactId>maven-clean-plugin</artifactId>
85 <version>3.1.0</version>
86 </plugin>
87 <!-- default lifecycle, jar packaging: see
88 https://maven.apache.org/ref/current/maven-core/default-bindings.html#
89 Plugin_bindings_for_jar_packaging -->
90 <plugin>
91 <artifactId>maven-resources-plugin</artifactId>
92 <version>3.0.2</version>
93 </plugin>
94 <plugin>
95 <groupId>org.apache.maven.plugins</groupId>
96 <artifactId>maven-compiler-plugin</artifactId>
97 <configuration>
98 <source>${maven.compiler.source}</source>

```

```

97         <target>${maven.compiler.target}</target>
98     </configuration>
99     <version>${maven.compiler.plugin.version}</version>
100 </plugin>
101 <plugin>
102     <groupId>org.apache.maven.plugins</groupId>
103     <artifactId>maven-surefire-plugin</artifactId>
104     <version>${maven.surefire-plugin.version}</version>
105     <configuration>
106         <skipTests>${skipTests}</skipTests>
107     </configuration>
108 </plugin>
109 <plugin>
110     <artifactId>maven-jar-plugin</artifactId>
111     <version>3.0.2</version>
112 </plugin>
113 <plugin>
114     <artifactId>maven-install-plugin</artifactId>
115     <version>2.5.2</version>
116 </plugin>
117 <plugin>
118     <artifactId>maven-deploy-plugin</artifactId>
119     <version>2.8.2</version>
120 </plugin>
121 <!-- site lifecycle , see
122      https://maven.apache.org/ref/current/maven-core/lifecycles.html#site_Lifecycle -->
123 <plugin>
124     <artifactId>maven-site-plugin</artifactId>
125     <version>3.7.1</version>
126 </plugin>
127 <plugin>
128     <artifactId>maven-project-info-reports-plugin</artifactId>
129     <version>3.0.0</version>
130 </plugin>
131 </plugins>
132 </pluginManagement>
133 </build>
134 </project>

```

Código 1 – POM.xml

## 3.2 Classes DTOs

As classes DTOs (Data Transfer Object ) tem como responsabilidade produzir objetos que serão usados como transferência de dados entre as abstrações (camadas) do software.

A comunicação entre a camada do Cliente e a camada de serviço transfere e recebe dados no formato JSON (JavaScript Object Notation) que contemplem a estrutura de dados dos DTOs. Uma alternativa é fornecer a implementação/especificação dos DTOs aos Clientes para que possam lidar diretamente com instâncias dessas classes.

A comunicação entre a camada de Serviços e a camada de Negócio é feita por meio de instâncias de DTOs. Veja o exemplo de uma classe DTO a seguir:

```

1 package ifmt.cba.dto;
2
3 import org.apache.commons.lang3.builder.ToStringBuilder;
4 import org.apache.commons.lang3.builder.ToStringStyle;
5
6 public class ProdutoDTO {
7
8     private int codigo;
9     private String nome;
10    private float custoUnidade;
11    private int valorEnergetico;
12    private int estoque;
13    private int estoqueMinimo;
14    private GrupoAlimentarDTO grupoAlimentar;
15
16    public int getCodigo() {

```

```

17         return codigo;
18     }
19
20     public void setCodigo(int codigo) {
21         this.codigo = codigo;
22     }
23
24     public String getNome() {
25         return nome;
26     }
27
28     public void setNome(String nome) {
29         this.nome = nome;
30     }
31
32     public float getCustoUnidade() {
33         return custoUnidade;
34     }
35
36     public void setCustoUnidade(float custoUnidade) {
37         this.custoUnidade = custoUnidade;
38     }
39
40     public int getValorEnergetico() {
41         return valorEnergetico;
42     }
43
44     public void setValorEnergetico(int valorEnergetico) {
45         this.valorEnergetico = valorEnergetico;
46     }
47
48     public GrupoAlimentarDTO getGrupoAlimentar() {
49         return grupoAlimentar;
50     }
51
52     public void setGrupoAlimentar(GrupoAlimentarDTO grupoAlimentar) {
53         this.grupoAlimentar = grupoAlimentar;
54     }
55
56     public int getEstoque() {
57         return estoque;
58     }
59
60     public int getEstoqueMinimo() {
61         return estoqueMinimo;
62     }
63
64     public void setEstoqueMinimo(int estoqueMinimo) {
65         this.estoqueMinimo = estoqueMinimo;
66     }
67
68     public void setEstoque(int estoque) {
69         this.estoque = estoque;
70     }
71
72     @Override
73     public String toString() {
74         return ToStringBuilder.reflectionToString(this, ToStringStyle.JSON_STYLE);
75     }
76 }

```

Código 2 – ProdutoDTO.java

As classes DTOs são extremamente simples, representam apenas a estrutura de dados de entidades do modelo de domínio do software, não possuindo nenhuma lógica associada. Apenas a sobrescrita do método `toString()` é implementada nessa classe, além dos atributos e métodos de acesso aos atributos.

### 3.3 Classes de Entidades

As classes de Entidades tem como responsabilidade produzir objetos que serão usados como transferência de dados entre as camadas de negócio e persistência.

Quando as classes da camada de negócio recebem instâncias de DTOs, estes são convertidos para instâncias de Entidades. As classes de entidades possuem duas características que as diferenciam dos DTOs: 1 - possuem o MOR (mapeamento objeto-relacional); 2 - possuem lógica de validação dos dados presentes nos atributos. Veja o exemplo de uma classe de Entidade a seguir:

```
1 package ifmt.cba.entity;
2
3 import org.apache.commons.lang3.builder.ToStringBuilder;
4 import org.apache.commons.lang3.builder.ToStringStyle;
5
6 import jakarta.persistence.Column;
7 import jakarta.persistence.Entity;
8 import jakarta.persistence.FetchType;
9 import jakarta.persistence.GeneratedValue;
10 import jakarta.persistence.GenerationType;
11 import jakarta.persistence.Id;
12 import jakarta.persistence.JoinColumn;
13 import jakarta.persistence.ManyToOne;
14 import jakarta.persistence.Table;
15
16 @Entity
17 @Table(name = "produto")
18 public class Produto {
19
20     @Id
21     @GeneratedValue(strategy = GenerationType.IDENTITY)
22     private int codigo;
23
24     @Column(name = "nome")
25     private String nome;
26
27     @Column(name = "custo_unidade")
28     private float custoUnidade;
29
30     @Column(name = "valor_energetico")
31     private int valorEnergetico;
32
33     @Column(name = "estoque")
34     private int estoque;
35
36     @Column(name = "estoque_minimo")
37     private int estoqueMinimo;
38
39     @ManyToOne(fetch = FetchType.EAGER)
40     @JoinColumn(name = "id_grupo")
41     private GrupoAlimentar grupoAlimentar;
42
43     public int getCodigo() {
44         return codigo;
45     }
46
47     public void setCodigo(int codigo) {
48         this.codigo = codigo;
49     }
50
51     public String getNome() {
52         return nome;
53     }
54
55     public void setNome(String nome) {
56         this.nome = nome;
57     }
58
59     public float getCustoUnidade() {
60         return custoUnidade;
61     }
62 }
```

```

63 public void setCustoUnidade(float custoUnidade) {
64     this.custoUnidade = custoUnidade;
65 }
66
67 public int getValorEnergetico() {
68     return valorEnergetico;
69 }
70
71 public void setValorEnergetico(int valorEnergetico) {
72     this.valorEnergetico = valorEnergetico;
73 }
74
75 public GrupoAlimentar getGrupoAlimentar() {
76     return grupoAlimentar;
77 }
78
79 public void setGrupoAlimentar(GrupoAlimentar grupoAlimentar) {
80     this.grupoAlimentar = grupoAlimentar;
81 }
82
83 public int getEstoque() {
84     return estoque;
85 }
86
87 public void setEstoque(int estoque) {
88     this.estoque = estoque;
89 }
90
91 public int getEstoqueMinimo() {
92     return estoqueMinimo;
93 }
94
95 public void setEstoqueMinimo(int estoqueMinimo) {
96     this.estoqueMinimo = estoqueMinimo;
97 }
98
99 @Override
100 public String toString() {
101     return ToStringBuilder.reflectionToString(this, ToStringStyle.JSON_STYLE);
102 }
103
104 @Override
105 public int hashCode() {
106     final int prime = 31;
107     int result = 1;
108     result = prime * result + codigo;
109     return result;
110 }
111
112 @Override
113 public boolean equals(Object obj) {
114     if (this == obj)
115         return true;
116     if (obj == null)
117         return false;
118     if (getClass() != obj.getClass())
119         return false;
120     Produto other = (Produto) obj;
121     if (codigo != other.codigo)
122         return false;
123     return true;
124 }
125
126 public String validar() {
127     String retorno = "";
128
129     if (this.nome == null || this.nome.length() < 3) {
130         retorno += "Nome invalido";
131     }
132
133     if (this.custoUnidade <= 0) {
134         retorno += "Custo por unidade invalido";
135     }
136
137     if (this.valorEnergetico < 0) {
138         retorno += "Valor energetico invalido";

```

```

139     }
140
141     if (estoque < 0) {
142         retorno += "Estoque invalido";
143     }
144
145     if (this.grupoAlimentar == null || !this.grupoAlimentar.validar().isEmpty()) {
146         retorno += "Grupo alimentar invalido";
147     }
148
149     return retorno;
150 }
151 }

```

Código 3 – Produto.java

Pode-se observar que a classe entidade tem a mesma estrutura de dados da classe DTO correspondente, acrescida das anotações de MOR da JPA. Além disso sobrescreve os métodos toString(), hashCode() e equals().

Por fim, acrescenta-se um método validar(), cuja finalidade é verificar se os valores presentes nos atributos do objeto satisfazem as condições estabelecidas. Este método, cujo retorno é uma String, pode representar duas situações: 1 - se os dados estiverem corretos a String devolvida será vazia; 2 - para cada valor de atributo inválido, será concatenado uma mensagem de inconsistência, sendo devolvida ao final da validação a String contendo todas as mensagens de inconsistência concatenadas.

### 3.4 Classes de Negócio

As classes de Negócio reúnem as operações CRUD das Entidades, além de operações de processamento e consulta aos dados da Entidade relacionada. Essas recebem os DTOs vindo da camada de Serviços, convertê-los para Entidades, e verificar a validade dos dados contidos, as regras de negócio e executar as ações de persistência. Quando a resposta de operações das Negócio devolvem instâncias de objetos de Entidades, estas serão convertidas para objetos DTOs antes de serem devolvidos. A seguir, veja um exemplo de classe de Negócio:

```

1 package ifmt.cba.negocio;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import org.modelmapper.ModelMapper;
7
8 import ifmt.cba.dto.ProdutoDTO;
9 import ifmt.cba.entity.Produto;
10 import ifmt.cba.persistencia.PersistenciaException;
11 import ifmt.cba.persistencia.ProdutoDAO;
12
13 public class ProdutoNegocio {
14     private ModelMapper modelMapper;
15     private ProdutoDAO produtoDAO;
16
17     public ProdutoNegocio(ProdutoDAO produtoDAO) {
18         this.produtoDAO = produtoDAO;
19         this.modelMapper = new ModelMapper();
20     }
21
22     public void inserir(ProdutoDTO produtoDTO) throws NegocioException {
23
24         Produto produto = this.toEntity(produtoDTO);
25         String mensagemErros = produto.validar();
26
27         if (!mensagemErros.isEmpty()) {
28             throw new NegocioException(mensagemErros);
29         }
30

```

```

31 try {
32     if (!produtoDAO.buscarPorParteNome(produto.getNome()).isEmpty()) {
33         throw new NegocioException("Ja existe esse produto");
34     }
35     produtoDAO.beginTransaction();
36     produtoDAO.incluir(produto);
37     produtoDAO.commitTransaction();
38 } catch (PersistenciaException ex) {
39     produtoDAO.rollbackTransaction();
40     throw new NegocioException("Erro ao incluir o produto - " + ex.getMessage());
41 }
42 }
43
44 public void alterar(ProdutoDTO produtoDTO) throws NegocioException {
45
46     Produto produto = this.toEntity(produtoDTO);
47     String mensagemErros = produto.validar();
48     if (!mensagemErros.isEmpty()) {
49         throw new NegocioException(mensagemErros);
50     }
51     try {
52         if (produtoDAO.buscarPorCodigo(produto.getCodigo()) == null) {
53             throw new NegocioException("Nao existe esse produto");
54         }
55         produtoDAO.beginTransaction();
56         produtoDAO.alterar(produto);
57         produtoDAO.commitTransaction();
58     } catch (PersistenciaException ex) {
59         produtoDAO.rollbackTransaction();
60         throw new NegocioException("Erro ao alterar o produto - " + ex.getMessage());
61     }
62 }
63
64 public void excluir(int codigo) throws NegocioException {
65     try {
66         Produto produto = produtoDAO.buscarPorCodigo(codigo);
67         if (produto == null) {
68             throw new NegocioException("Nao existe esse produto");
69         }
70         produtoDAO.beginTransaction();
71         produtoDAO.excluir(produto);
72         produtoDAO.commitTransaction();
73     } catch (PersistenciaException ex) {
74         produtoDAO.rollbackTransaction();
75         throw new NegocioException("Erro ao excluir o produto - " + ex.getMessage());
76     }
77 }
78
79
80 public List<ProdutoDTO> pesquisaParteNome(String parteNome) throws NegocioException {
81     try {
82         return this.toDTOAll(produtoDAO.buscarPorParteNome(parteNome));
83     } catch (PersistenciaException ex) {
84         throw new NegocioException("Erro ao pesquisar produto pelo nome - " + ex.getMessage());
85     }
86 }
87
88 public List<ProdutoDTO> pesquisaProdutoAbaixoEstoqueMinimo() throws NegocioException {
89     try {
90         return this.toDTOAll(produtoDAO.buscarProdutoAbaixoEstoqueMinimo());
91     } catch (PersistenciaException ex) {
92         throw new NegocioException("Erro ao pesquisar produto abaixo do estoque minimo - " + ex.
93             getMessage());
94     }
95 }
96
97 public ProdutoDTO pesquisaCodigo(int codigo) throws NegocioException {
98     try {
99         return this.toDTO(produtoDAO.buscarPorCodigo(codigo));
100     } catch (PersistenciaException ex) {
101         throw new NegocioException("Erro ao pesquisar produto pelo codigo - " + ex.getMessage());
102     }
103 }
104
105 public List<ProdutoDTO> toDTOAll(List<Produto> listaProduto) {

```



```

106 List<ProdutoDTO> listaDTO = new ArrayList<ProdutoDTO>();
107
108 for (Produto produto : listaProduto) {
109     listaDTO.add(this.toDTO(produto));
110 }
111 return listaDTO;
112 }
113
114 public ProdutoDTO toDTO(Produto produto) {
115     return this.modelMapper.map(produto, ProdutoDTO.class);
116 }
117
118 public Produto toEntity(ProdutoDTO produtoDTO) {
119     return this.modelMapper.map(produtoDTO, Produto.class);
120 }
121 }

```

Código 4 – ProdutoNegocio.java

As operações de persistência invocadas pelas classes de Negócio são executadas pelos objetos de classes DAO, injetados pelo construtor dessas classes.

### 3.5 Classes de Persistência (DAOs)

Essas classe tem o objetivo de executarem as operações de persistência e recuperação de informações das entidades do domínio do software. As classes DAOs foram implementadas por meio da API JPA do Java.

Como forma de reutilizar um código comum entre os DAOs, foi codificada uma superclasse DAO, que reúne as operações de inclusão, alteração e exclusão, além do controle de trasação com o banco de dados. Veja a seguir a codificação da superclasse DAO:

```

1 package ifmt.cba.persistencia;
2
3 import jakarta.persistence.EntityManager;
4
5 public class DAO<VO> {
6
7     protected EntityManager entityManager;
8
9     public DAO(EntityManager entityManager) throws PersistenciaException {
10         this.entityManager = entityManager;
11     }
12
13     public void incluir(VO vo) throws PersistenciaException {
14         try {
15             this.entityManager.persist(vo);
16         } catch (Exception e) {
17             throw new PersistenciaException("Erro ao incluir " + vo.getClass() + " - " + e.getMessage());
18         }
19     }
20
21     public void alterar(VO vo) throws PersistenciaException {
22         try {
23             this.entityManager.merge(vo);
24         } catch (Exception e) {
25             throw new PersistenciaException("Erro ao alterar " + vo.getClass() + " - " + e.getMessage());
26         }
27     }
28
29     public void excluir(VO vo) throws PersistenciaException {
30         try {
31             this.entityManager.remove(vo);
32         } catch (Exception e) {
33             throw new PersistenciaException("Erro ao excluir " + vo.getClass() + " - " + e.getMessage());
34         }
35     }

```

```

36
37 public EntityManager getEntityManager() {
38     return entityManager;
39 }
40
41 public void setEntityManager(EntityManager entityManager) {
42     this.entityManager = entityManager;
43 }
44
45 public void beginTransaction() {
46     this.entityManager.getTransaction().begin();
47 }
48
49 public void commitTransaction() {
50     this.entityManager.getTransaction().commit();
51 }
52
53 public void rollbackTransaction() {
54     this.entityManager.getTransaction().rollback();
55 }
56
57 }

```

Código 5 – DAO.java

A classe DAO executa as operações de persistência por meio de um objeto EntityManager, injetado pelo construtor dessa classe. Veja a seguir a codificação de uma subclasse DAO:

```

1 package ifmt.cba.persistencia;
2
3 import java.util.List;
4 import ifmt.cba.entity.Produto;
5 import jakarta.persistence.EntityManager;
6 import jakarta.persistence.Query;
7
8 public class ProdutoDAO extends DAO<Produto>{
9
10     public ProdutoDAO(EntityManager entityManager) throws PersistenciaException {
11         super(entityManager);
12     }
13
14     public Produto buscarPorCodigo(int codigo) throws PersistenciaException {
15
16         Produto produto = null;
17
18         try {
19             produto = this.entityManager.find(Produto.class, codigo);
20         } catch (Exception ex) {
21             throw new PersistenciaException("Erro na selecao por codigo - " + ex.getMessage());
22         }
23         return produto;
24     }
25
26     @SuppressWarnings("unchecked")
27     public List<Produto> buscarPorGrupoAlimentar(int codigoGrupo) throws PersistenciaException {
28         List<Produto> listaProduto;
29         try {
30             Query query = this.entityManager
31                 .createQuery("SELECT p FROM Produto p WHERE p.grupoAlimentar.codigo = :codgrupo ORDER BY
32                     p.nome");
33             query.setParameter("codgrupo", codigoGrupo);
34             listaProduto = query.getResultList();
35         } catch (Exception ex) {
36             throw new PersistenciaException("Erro na selecao por grupo alimentar - " + ex.getMessage());
37         }
38         return listaProduto;
39     }
40
41     @SuppressWarnings("unchecked")
42     public List<Produto> buscarPorParteNome(String nome) throws PersistenciaException {
43         List<Produto> listaProduto;

```

```

43 try {
44     Query query = this.entityManager
45         .createQuery("SELECT p FROM Produto p WHERE UPPER(p.nome) LIKE :pNome ORDER BY p.nome");
46     query.setParameter("pNome", "%" + nome.toUpperCase().trim() + "%");
47     listaProduto = query.getResultList();
48 } catch (Exception ex) {
49     throw new PersistenciaException("Erro na selecao por parte do nome - " + ex.getMessage());
50 }
51 return listaProduto;
52 }
53
54 @SuppressWarnings("unchecked")
55 public List<Produto> buscarProdutoAbaixoEstoqueMinimo() throws PersistenciaException {
56     List<Produto> listaProduto;
57     try {
58         Query query = this.entityManager
59             .createQuery("SELECT p FROM Produto p WHERE p.estoque < p.estoqueMinimo ORDER BY p.nome")
60             ;
61         listaProduto = query.getResultList();
62     } catch (Exception ex) {
63         throw new PersistenciaException("Erro na selecao por estoque minimo - " + ex.getMessage());
64     }
65     return listaProduto;
66 }

```

Código 6 – ProdutoDAO.java

As subclasses de DAO reúnem operações particulares de cada Entidade, principalmente operações de recuperação de dados.

Os objetos recebidos e devolvidos pelas classes DAO são de classes do tipo Entidade, que serão tratados e convertidos para DTOS nas classes de negócio.

### 3.6 Classes Utilitária FabricaEntityManager

A classe utilitária FabricaEntityManager tem como objetivo gerar instâncias de objetos EntityManager que são usados pelas classes DAOs para executarem as operações de persistência via JPA.

```

1 package ifmt.cba.persistencia;
2
3 import jakarta.persistence.EntityManager;
4 import jakarta.persistence.EntityManagerFactory;
5 import jakarta.persistence.Persistence;
6
7 public class FabricaEntityManager {
8
9     private static EntityManagerFactory emf_producao;
10    private static EntityManagerFactory emf_teste;
11    public static final String UNIDADE_PRODUCAO = "restaurante_producao";
12    public static final String UNIDADE_TESTE = "restaurante_teste";
13
14    private FabricaEntityManager() {
15
16    }
17
18    public static EntityManager getEntityManagerProducao() {
19        if (emf_producao == null) {
20            emf_producao = Persistence.createEntityManagerFactory(UNIDADE_PRODUCAO);
21        }
22        return emf_producao.createEntityManager();
23    }
24
25    public static EntityManager getEntityManagerTeste() {
26        if (emf_teste == null) {
27            emf_teste = Persistence.createEntityManagerFactory(UNIDADE_TESTE);
28        }
29        return emf_teste.createEntityManager();
30    }

```

## Código 7 – FabricaEntityManager.java

Pode-se observar que em sua codificação existem dois métodos que produzem objetos EntityManager: `getEntityManagerProducao()` e `getEntityManagerTeste()`. Essa implementação permite que o código de classes de negócio e persistência trabalhem com uma base de dados diferente da base de dados dos testes automatizados. No arquivo de configurações da JPA do projeto, foram especificadas duas unidades de persistência, uma para ser usada pelo código que irá entrar em produção e outra pelos códigos dos testes automatizados, permitindo o isolamento dessas duas ações. Veja o código do arquivo `persistence.xml`:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/
   persistence/persistence_2_1.xsd"
5   version="2.1">
6
7   <persistence-unit name="restaurante_producao" transaction-type="RESOURCE_LOCAL">
8     <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
9     <exclude-unlisted-classes>>false</exclude-unlisted-classes>
10    <properties>
11
12      <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost:5432/
        restaurante_producao" />
13      <property name="javax.persistence.jdbc.user" value="postgres" />
14      <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" />
15      <property name="javax.persistence.jdbc.password" value="postgres" />
16      <property name="javax.persistence.schema-generation.database.action" value="none" /> <!--
        none / create / create-drop / update -->
17      <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />
18      <property name="hibernate.show_sql" value="true" /> <!-- Show SQL in console -->
19      <property name="hibernate.format_sql" value="true" /> <!-- Show SQL formatted -->
20    </properties>
21  </persistence-unit>
22
23  <persistence-unit name="restaurante_teste" transaction-type="RESOURCE_LOCAL">
24    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
25    <exclude-unlisted-classes>>false</exclude-unlisted-classes>
26    <properties>
27
28      <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />
29      <property name="javax.persistence.jdbc.url" value="jdbc:h2:mem:restaurante_teste" />
30      <property name="javax.persistence.jdbc.user" value="sa" />
31      <property name="javax.persistence.jdbc.password" value="" />
32
33      <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect" />
34
35      <property name="javax.persistence.schema-generation.database.action" value="create" /> <!--
        none / create / create-drop / update -->
36      <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />
37      <property name="hibernate.show_sql" value="true" /> <!-- Show SQL in console -->
38      <property name="hibernate.format_sql" value="true" /> <!-- Show SQL formatted -->
39    </properties>
40  </persistence-unit>
41 </persistence>

```

## Código 8 – Persistence.xml

A unidade de persistência **restaurante\_producao** está configurada para acessar uma base de dados com o nome de `restaurante_producao` em um SGBD PostgreSQL.

A unidade de persistência **restaurante\_teste** está configurada para acessar uma base de dados com o nome de `restaurante_teste` em um SGBD H2 em memória.

## 4 Execução do Estudo de Caso

Considerando as informações que foram apresentadas nas seções anteriores, esta seção descreve as atividades que deverão ser desenvolvidas, como devem ser desenvolvidas e como devem ser entregues.

### 4.1 Como Desenvolver as Atividades

As atividades do estudo de caso descrito nesse documento devem ser desenvolvidas por grupos de alunos formados por 2 a 3 integrantes. As atividades serão compostas por ações de desenvolvimento do modelo conceitual por meio de classes DTO, classes de entidades, classes de negócio, classes de persistência, classes utilitárias e classes de serviços. As execuções dos serviços deverão ser realizadas em uma aplicação cliente (projeto separado) ou via classes de testes de API (projeto separado);

O projeto deverá ser desenvolvido seguindo as definições da seção Modelo Arquitetural. Deverá ser entregue a pasta do projeto compactada em formato zip. **Obs: antes de compactar executar a ação Clean do maven para limpar o projeto de arquivos de compilação e execução.**

## 5 Entrega e Apresentação

Nesta seção serão apresentadas as informações referente as entregas, prazos, formas de entrega e critérios de avaliação.

### 5.1 Entrega dos Artefatos

Os resultados do desenvolvimento do estudo de caso serão entregues com os seguintes arquivos:

1. Projeto Servidor Implementado - formato ZIP (nome: Grupox-ProjetoServidorRestaurante.zip);
2. Projeto Cliente Implementado - formato ZIP (nome: Grupox-ProjetoClienteRestaurante.zip);
3. Apresentação - formato PDF (nome: Grupox-ApresentacaoPOS.pdf);

As entregas desses arquivos serão realizadas no ambiente AVA/Moodle em tarefas específicas para cada arquivo.

Datas de Entregas: a serem definidas.

### 5.2 Apresentação

Cada grupo deve preparar uma apresentação por meio de um software de apresentação de livre escolha, contendo os resultados do estudo de caso, baseados nas entregas descritas.

É interessante que o grupo prepare a apresentação, e gere um PDF da apresentação, para evitar possíveis problemas de compatibilidade de software e/ou versões.

### 5.3 Critérios de Avaliação

A nota do estudo de caso será calculada seguindo as seguintes distribuições:

1. Projeto Servidor Implementado - 5,0 pontos;
2. Projeto Cliente Implementado - 3,0 pontos;
3. Apresentação - 2 pontos;