Relatório Sistema de Vendas

Allan Groisman Kusbick
Fundamentos de Desenvolvimento de Software
Ciência da Computação — PUCRS

10 de novembro de 2023

1. Introdução

Este trabalho tem como objetivo a elaboração de um sistema de vendas, o qual deve seguir princípios SOLID e padrões da arquitetura CLEAN. O sistema simula uma loja que possui produtos a venda em seu catálogo. As três principais funções são requeridas e obrigatórias: buscar os produtos disponíveis, requerer um orçamento e efetivar uma venda. O caminho do usuário então, é buscar os produtos à sua disposição, pedir um orçamento com os produtos selecionados e então, caso queira, efetivar a compra.

Cada produto possui limitações mínimas e máximas em estoque, dessa forma, produtos que estão abaixo de suas limitações mínimas tornam-se não disponíveis e não aparecem na busca do usuário.

O orçamento tem seu valor formado pela soma de todos os produtos que foram pedidos pelo usuário, adicionando um imposto de 10% e retirando um desconto, caso seja este o caso. Existem duas políticas iniciais de desconto, pela média das últimas três compras do cliente, que pode chegar a 30%, e por 25% por fidelidade, caso o cliente tenha feito dez compras nos últimos seis meses. Por fim, a efetivação de um orçamento acontece ao verificar se as quantidades requeridas estão à disposição para serem vendidas e se o orçamento ainda se encontra dentro do prazo (21 dias a partir da solicitação, ou 35 dias caso julho, dezembro, janeiro ou fevereiro).

O banco de dados desse sistema é composto por dois repositórios, um para os produtos e outro para os orçamentos. Existe ainda um módulo estatístico que atua sobre esses repositórios para trazer três diferentes relatórios sobre estes dados: os principais clientes, dados produtos indisponíveis e resumo de orçamentos a partir de data específica.

2. Arquiteturas e Princípios

Para organizar os diferentes agentes deste sistema foi utilizado um padrão de arquitetura em camadas juntamente com os princípios da arquitetura CLEAN e do SOLID. O objetivo é manter um ambiente de fácil manutenção, entendimento, testabilidade e escalabilidade.

Na arquitetura de camadas, cada nível, possui um conjunto de responsabilidades e interage apenas com os níveis adjacentes, de forma a depender apenas da camada imediata. A separação garante individualidade entre as partes do sistema de forma em que partes específicas possam ser modificadas sem danos colaterais nos outros níveis. As camadas neste sistema foram divididas em: Interface, aplicação, domínio e persistência.

2.1 Interface

Esta camada é responsável pela interação com o usuário, é nela que se captura as possíveis entradas e então as mapeiam de acordo com as respectivas solicitações de métodos. Para executar esta função temos dois *Controllers,* um para a parte estatística (*ControllerEstatistica*) e outro para parte dos serviços (*ControllerServicos*) que fazem ligação direta com a camada de aplicação como é possível observar na figura 1.

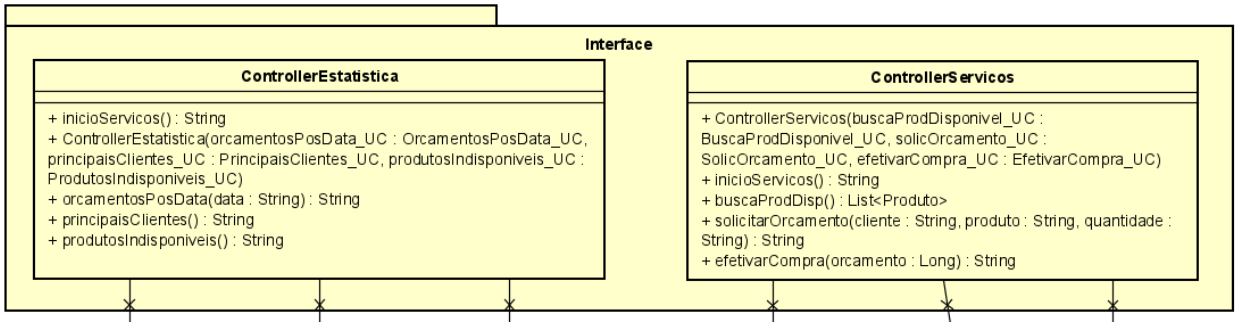

Figura 1: Camada de Interface.

2.2 Aplicação

Esta camada é responsável por receber as solicitações da camada de interface e coordenar o fluxo de trabalho. Ela atua como um intermediário entre a interface do usuário e a camada de domínio. Aqui temos classes correspondentes aos chamados *Users Cases,* ou casos de usuário, que traduzem a entrada do usuário ao sistema, ainda não lidando com a lógica de negócio, mas decidindo quais serviços da próxima camada, chamada de domínio, devem ser utilizados.

As classes *UC* tornam os casos independentes e atendem apenas aos métodos específicos para seu usuário neste caso de uso, o que respeita o princípio de SRP (*Single Responsability Principle*) do SOLID. Na figura 2, pode-se observar que no sistema existem três classes *UC* correspondentes aos três serviços requeridos: busca produtos disponíveis *(BuscaProdDisponivel_UC)*, solicitar orçamento *(SolicOrcamento_UC)* e efetivar compra *(EfetivarCompra_UC)*.

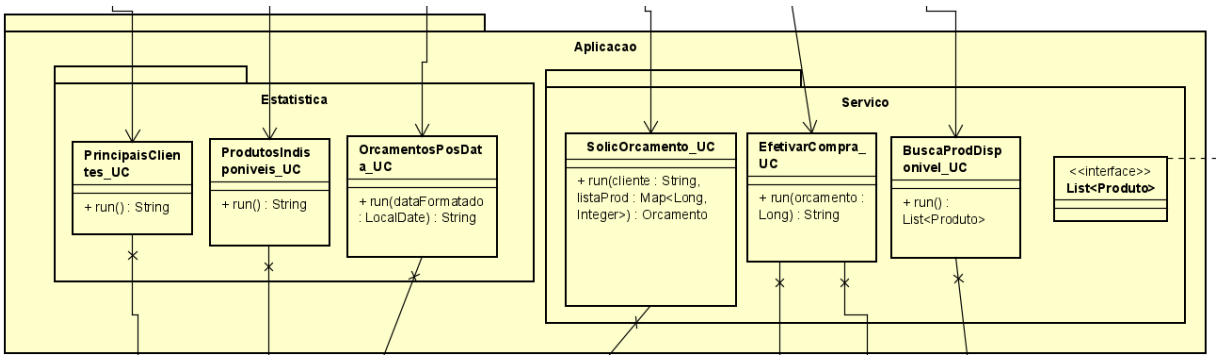

Figura 2: Camada de Aplicação.

2.3 Domínio

É nesta camada que a lógica do negócio e suas regras acontecem. Ela é o núcleo do sistema, que inclui as entidades, como produto e orçamento, serviços que neste caso gerenciam as vendas e os estoques, além de outras regras como as diferentes políticas de descontos. É a partir dos serviços, aqui presentes, que há a ligação do sistema com os repositórios.

2.3.1 Entidades (figura 3): os modelos de objetos que são significativos no negócio, que possuem identidade própria e servem de base para o sistema. No sistema tem-se o Produto, Pedido, que origina um orçamento, além do próprio orçamento.

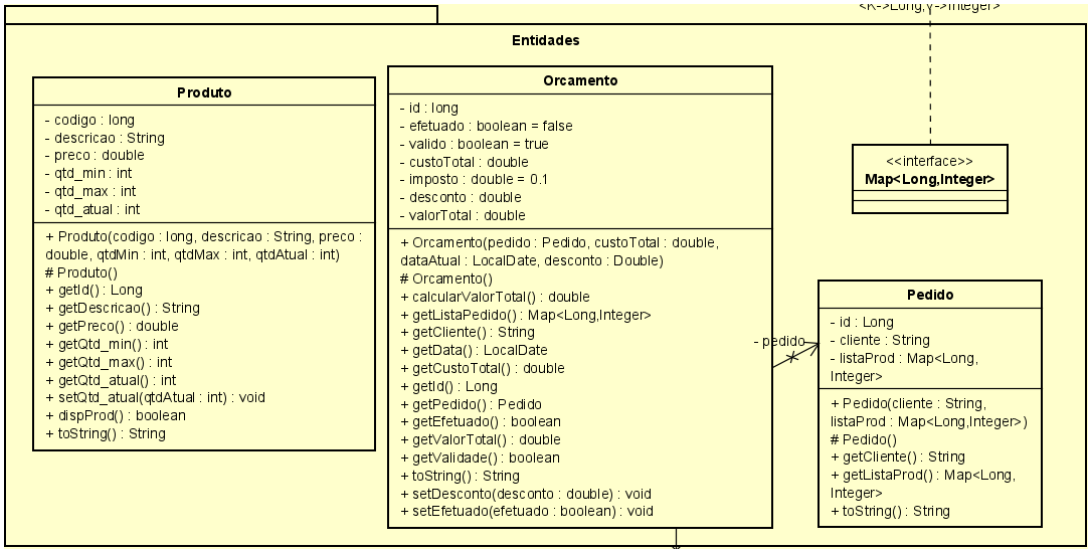

Figura 3: Pacote entidade da camada de Domínio.

2.3.2 Desconto (figura 4): especificamente neste sistema, existe a necessidade de incluir diferentes formas de avaliar o desconto aplicado sobre um orçamento. Dessa forma, esse pacote foi criado para conter uma fábrica de descontos, original do padrão *factory*, essa que manipula as diferentes formas de desconto permite a adição de futuras políticas a partir de uma interface de desconto única.

No sistema tem-se a classe *FabricaDesconto,* a própria fábrica, que concede o desconto ideal ao orçamento, além de uma interface *IpoliticaDesconto* que serve para padronizar e ser implementada pelas políticas em si, que no momento são *PoliticaMaisDez* e *PoliticaMedia.* A *politicaMaisDez* traz 25% de desconto ao cliente que efetuou compras pelo menos dez compras nos últimos seis meses. Já *PoliticaMedia,* a partir da média das últimas três compras do cliente, disponibiliza 10% caso a média ultrapasse 10 mil, além de 5% para cada 10mil excedente, até um máximo de 30%. O maior desconto individual capturado das políticas é utilizado no orçamento.

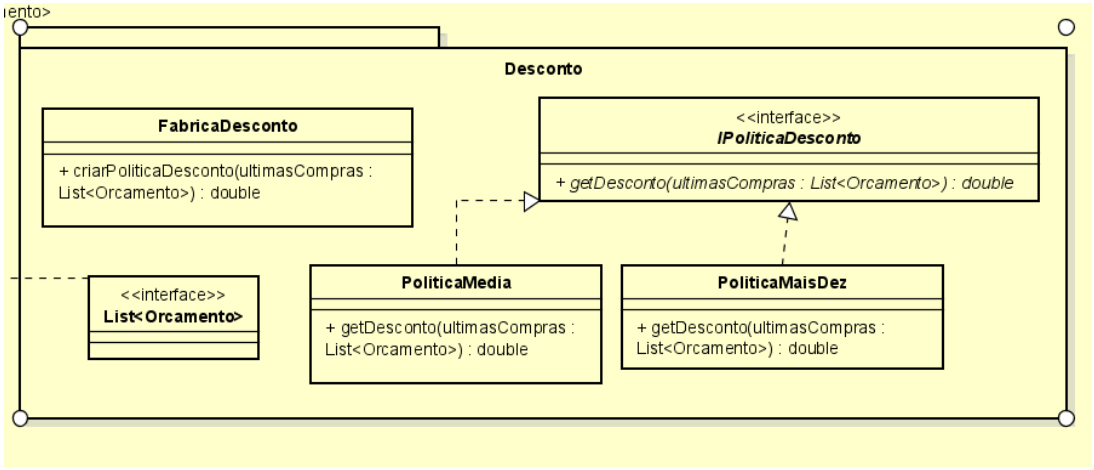

Figura 4: Pacote desconto da camada domínio.

2.3.3 Serviços (figura 5): os serviços são os núcleos pensativos que contém a lógica do negócio, que manipulam as entidades e fazem solicitações à camada de persistência. Neles estão todos os métodos necessários para a utilização e funcionamento do sistema.

No sistema temos o *ServicoEstatistica* que possui toda a lógica e os métodos para lidar com os três requerimentos do módulo de estatística vindo da camada de aplicação. Este serviço utiliza tanto do repositório de produtos quanto de orçamentos.

Existe também o *ServicEstoque* que atua somente sobre o repositório de produtos e que possui as inteligências e funcionalidades para lidar e gerenciar sobre quais produtos estão disponíveis.

Por último tem-se o *ServicoVenda,* este que é maior e atua manipulando tanto produtos como orçamentos e seus repositórios. É nele que se tem a lógica para a criação dos orçamentos além da efetivação dos mesmos.

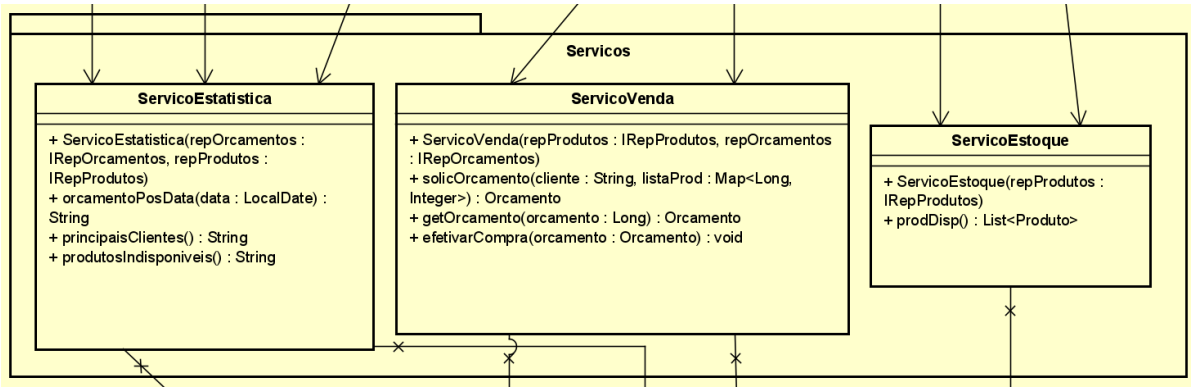

Figura 5: Pacote interfaces em camada domínio.

2.3.4 Interfaces (Figura 6): assim como a camada de interface, que serve como ponte entre o usuário e o sistema, as classes de interface dentro do domínio fazem parte da ligação entre os serviços e a camada de persistência.

A ideia da arquitetura em camadas é isolar e tornar genérica a integração de cada nível quando conversa com o seu adjacente. É necessário garantir que a lógica de negócio não dependa diretamente de detalhes de implementação de persistência e para isso estas classes interface servem como mapa das necessidades do domínio para com os dados.

Aqui entra o princípio DIP *(Dependency Inversion Principle),* ou princípio de inversão de dependência, que inverte uma possível dependência a uma classe concreta de persistência, por uma interface que se molda e que atende apenas suas necessidades (ISP, outro princípio SOLID que prega que o módulo deve depender apenas seus métodos necessários). No sistema as camadas interfaces presentes são *IrepOrcamento* e *IrepProdutos.*

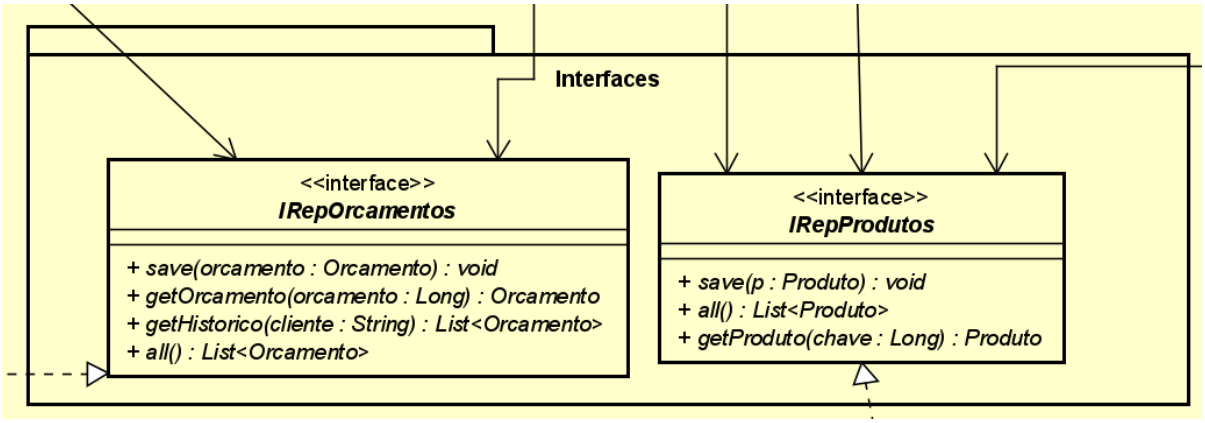

Figura 6: Pacote interfaces em camada domínio.

2.4 Persistência

Este nível trata, como o próprio nome sugere, da persistência e armazenamento dos dados. Além de ser responsável por garantir a integridade e consistência dos dados, a camada é responsável também por traduzir as operações de leitura e gravação de dados do sistema para operações no banco de dados, incluindo consultas, inserções, atualizações e exclusões.

No sistema, dois pacotes respectivos ao repositório de orçamentos e outro ao de produtos (Figura 7). Dentro de cada um temos uma classe que implementa a respectiva classe interface presente na camada domínio e que trata de complementar os métodos requeridos e que para isso utiliza de uma última classe interface, esta que por fim herda o *CrudRepository.*

O *CrudRepository* é uma interface fornecida pelo Spring que abstrai as operações básicas de persistência em um banco de dados. É a partir daí que vem seu nome *CRUD (Create, Read, Update, Delete),* ou seja, criar, ler, atualizar e deletar.

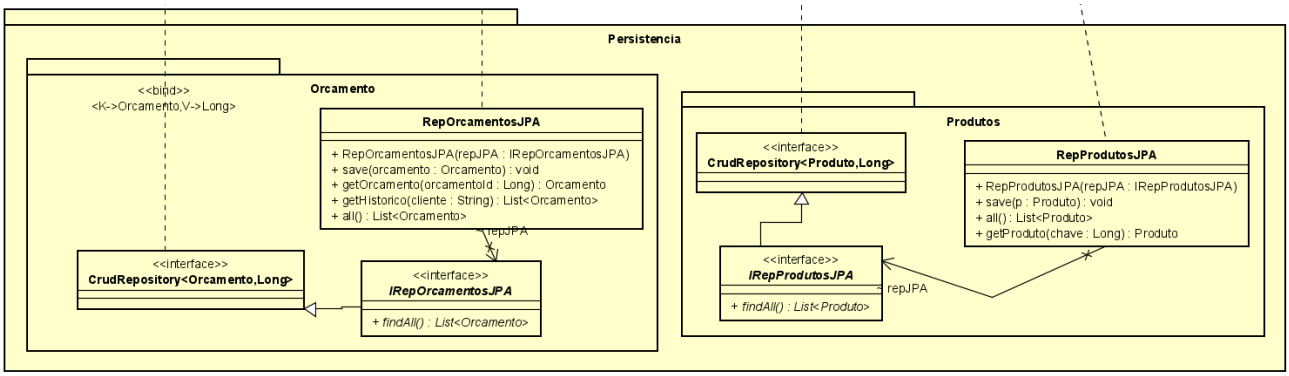

Figura 7: Camada de Persistência.

3. Conclusão

A estruturação e o detalhamento apresentados no relatório demonstram a criação de um sistema de vendas bem-organizado. A utilização de padrões de projeto, como o SOLID e a arquitetura CLEAN, evidenciam um cuidado com a manutenção, testabilidade e escalabilidade do sistema. A implementação em camadas (Interface, Aplicação, Domínio e Persistência) mostra uma clara separação de responsabilidades, possibilitando uma fácil compreensão e alteração de partes específicas do sistema sem afetar outras áreas.

A aplicação dos princípios SOLID em várias camadas do sistema como o Princípio da Responsabilidade Única (SRP), que foi aplicado nas classes dos Casos de Usuário (UC) mantendo-as focadas em tarefas específicas e tornando-as independentes, o que facilita a manutenção e reutilização do código. Além disso, o Princípio de Inversão de Dependência (DIP) é evidente entre a camada domínio e a de persistência, onde as interfaces são utilizadas para desacoplar a lógica de negócios da implementação concreta dos dados.

Em síntese, o sistema de vendas foi projetado com uma arquitetura sólida, seguindo boas práticas de desenvolvimento de software e padrões de projeto reconhecidos, o que o torna capaz de lidar com as demandas do ambiente de vendas de forma eficiente e escalável.