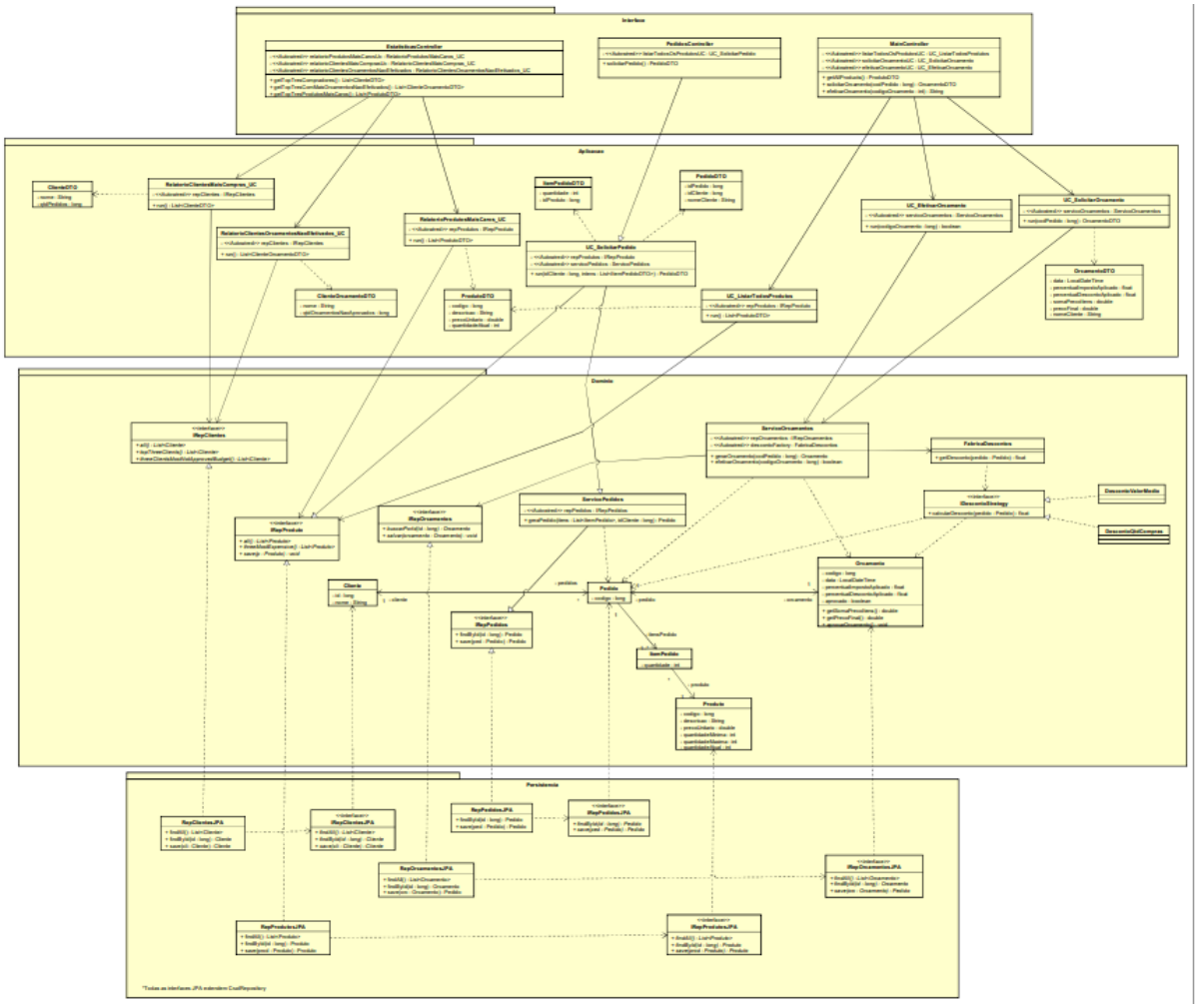
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Fundamentos de Desenvolvimento de Software - Trabalho TF
Alunos: Kristen Arguello, Ramiro Barros, Ricardo Süffert, Vinícius Turani

Relatório Trabalho Final
Fundamentos de Desenvolvimento de Software

1) Diagrama de classes da solução:



No modelo UML acima, a ordem das camadas se encontra organizada dessa forma meramente por questões de estética e legibilidade. A ordem real das camadas, conforme os conceitos da arquitetura limpa (*clean code*) é, da mais externa para a mais interna: *framework* de persistência, interface, aplicação e domínio. Além disso, está sendo entregue juntamente com o trabalho o arquivo do Astah com a modelagem apresentada acima, a fim de facilitar a leitura e a consulta ao diagrama.

2) Detalhamento dos padrões de projeto utilizados e o objetivo de cada um:

Para implementar os diferentes tipos de descontos, e possibilitar alterações futuras com um determinado nível de abstração, além de possibilitar a adição de políticas de descontos novas no futuro, foi utilizado o **padrão Strategy**. Foi implementada uma interface

com um método que gera o desconto, e foram criadas classes para gerar os descontos seguindo as diferentes regras para cada tipo de desconto. Assim, dependendo do desconto que será selecionado para ser aplicado, poderá ser chamado da mesma maneira, utilizando a interface. Essas classes e interfaces podem ser vistas na camada de domínio. Porém, como foi definido no enunciado, o maior desconto é o que será aplicado e estes não são acumuláveis, então foi necessário criar uma classe que elege qual desconto é o selecionado e o retorna. Assim, foi criada a classe *FábricaDeDescontos*. Essa implementa um método *getDesconto* que analisa qual implementação do *strategy* de descontos retorna o maior desconto e retorna para o serviço de orçamento, que é onde o desconto é utilizado.

Além disso, a estratégia adotada para o desenvolvimento deste sistema se baseia na prática de **injeção de dependência**. Essa abordagem é fundamental para garantir que as classes principais do sistema recebam as instâncias necessárias de serviços e componentes externos de forma automática. Isso promove uma estrutura mais flexível e coesa, reduzindo a interdependência entre os diferentes elementos do código, e, ao implementar a injeção de dependência, buscamos criar um sistema onde as classes não dependam diretamente umas das outras para funcionar. Em vez disso, elas recebem as dependências de que precisam, permitindo uma melhor modularidade e facilitando a substituição ou atualização de componentes sem impactar todo o sistema.

Essa prática foi aplicada de forma ampla nas principais classes de “UC”, como *BuscaFornecedorUC*, *ProdutosDisponiveisUC*, *SolicitarPedidoUC* e *ProdutoFornecedorUC*, garantindo que elas recebam automaticamente os serviços necessários para operar. Isso não apenas simplifica o código, mas também facilita a manutenção e a escalabilidade do sistema.

Mapeamento dos DTOs:

Foram desenvolvidos alguns Objetos de Transferência de Dados (DTOs) para facilitar a comunicação entre a aplicação e o front-end. Esses objetos foram criados com o intuito de fornecer informações específicas e relevantes sem a necessidade de retornar todos os atributos dos objetos originais, otimizando a eficiência da aplicação e evitando exibir informações adicionais e desnecessárias para o usuário final. Os DTOs criados foram: **ClienteDTO** (representa nome e quantidade de orçamentos totais do cliente), **ClienteOrcamentosDTO** (nome e quantidade de orçamentos não efetivados totais do cliente), **OrcamentoDTO** (guarda o desconto aplicado, impostos e valor total) , **ProdutoDTO** (representa a quantidade de produtos e o seu preço unitário), **ItemPedidoDTO**(representa a quantidade de um produto específico) e **PedidoDTO**(Representa um Pedido criado com a *id* do pedido e informações do cliente).

Mapeamento das Entidades:

Na modelagem do banco de dados, foram definidas diversas entidades para organizar as informações. Foram estruturadas para facilitar a organização e a gestão dos dados, garantindo a integridade das informações e permitindo a interconexão entre os diferentes elementos do sistema.

Clientes: Representa um cliente e possui os atributos "id" (identificador único do cliente) e "nome". Além disso, mantém uma lista de pedidos associados a cada cliente, criando um relacionamento de um cliente para vários pedidos.

ItemPedido: Esta entidade representa um item específico de um pedido. Inclui os atributos "id" (identificador único do item) e "quantidade", além de estar relacionada com a entidade Produto e com o Pedido ao qual o item pertence.

Orcamentos: Destinada a armazenar informações sobre orçamentos. Possui atributos como "id" (identificador único do orçamento), "pedido" (referência ao pedido relacionado), "data de criação", "percentual de imposto", "percentual de desconto" e "status". Também conta com métodos para calcular o preço final do orçamento considerando impostos e descontos. O orçamento possui relacionamento com pedido sendo vários para um pedido. Além disso, possui um relacionamento bidirecional com seus pedidos.

Pedidos: Responsável por guardar informações sobre pedidos. Inclui atributos como "id" (identificador único do pedido), uma lista de itens associados, referência ao cliente que fez o pedido e uma lista de orçamentos vinculados ao pedido. Pedidos possuem um relacionamento com orçamento de 1 para vários, sendo esse bidirecional.

Produtos: Destinada a armazenar informações sobre os produtos disponíveis. Possui atributos como "descrição", "preço unitário", "quantidade máxima e mínima suportada no estoque" e "quantidade atual em estoque"

Relacionamentos entre as Entidades no Banco de Dados:

Clientes e Pedidos:

Um cliente está associado a uma lista de pedidos (OneToMany).

Cada pedido mantém uma referência ao cliente correspondente (ManyToOne).

Pedido, ItemPedido e Produto:

Cada item pedido está vinculado a um produto específico (ManyToOne).

Os produtos possuem uma lista de itens pedidos correspondentes (OneToMany).

Cada item pedido possui uma referência ao pedido ao qual pertence (ManyToOne).

Os pedidos mantêm uma lista de itens pedidos associados (OneToMany).

Pedido e Orçamento:

Cada orçamento está diretamente ligado a um único pedido (ManyToOne).

Pedidos mantêm uma lista dos orçamentos relacionados (OneToMany).

Pedido e Cliente:

Cada pedido possui uma referência direta ao cliente que o realizou (ManyToOne).

Os clientes mantêm uma lista de pedidos realizados por eles em ordem realizada (OneToMany).

Mapeamento dos Endpoints da API e as estatísticas criadas:

Estes endpoints foram feitos para permitir a obtenção das **três estatísticas obrigatórias**, a **manipulação de orçamentos** e a realização de ações como a **efetivação de orçamentos** e a **listagem de produtos** e coisas relevantes para os usuários da API.

Endpoint: (“/statistics”)

(“/três-clientes-mais-compras”)

Método: GET

Descrição: Retorna uma lista com os três clientes que mais realizaram pedidos, exibindo a quantidade de pedidos feitos por cada um.

(“/três-produtos-mais-caros”)

Método: GET

Descrição: Retorna uma lista com os três produtos mais caros, incluindo seu ID, nome, quantidade em estoque e preço unitário.

(“/três-clientes-mais-chatos”)

Método: GET
Descrição: Retorna uma lista com os três clientes que mais solicitaram orçamentos não aprovados, exibindo a quantidade de orçamentos de cada um.

Endpoint: (“/api”)

(“ /efetivar-orçamento”)

Método: PATCH
Descrição: Recebe o ID de um orçamento, verifica sua validade, a disponibilidade de produtos para o pedido e outros critérios. Posteriormente, efetiva o orçamento e atualiza o estoque dos produtos. Retorna um status indicando se o orçamento foi efetivado com sucesso ou não.

(“ /solicitar-orçamento”)

Método: POST
Descrição: Recebe o ID de um pedido e cria um orçamento correspondente com os dados fornecidos na requisição. Retorna o orçamento recém-criado.

(“/listar-todos-orçamentos”)

Método: GET
Descrição: Retorna uma lista contendo todos os orçamentos, com informações detalhadas sobre o pedido e os dados específicos do orçamento.

Endpoint: (“/pedidos”)

(“/gerar”)

Método: POST
Descrição: Dada uma lista de itens do e um id de cliente, um pedido será criado. Assim, se os itens existirem, eles serão adicionados ao pedido, e, ao final, o pedido será transformado em um DTO para que seja retornado para o usuário final.