**Documento de Arquitetura de Software**

**Versão <1.2.9>**

**Histórico da Revisão**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor** |
| 13/05 | 1.0.0 | Primeira versão do documento de arquitetura de software seus respectivos requisitos. | Mairon Azevedo  Vitor Prado |
| 17/05 | 1.0.1 | Estilos arquiteturais e modelos VCP. | Beatriz Dellatorre  Denerson Eduardo  Vitor Prado |
| 10/06 | 1.0.2 | Elaboração da seção 5 do Documento | Adriano Souza  Daniel Neves |
| 10/06 | 1.0.3 | Elaboração da seção 2 do Documento | Denerson Eduardo  Adriano Souza  Jardeson Nogueira  Vitor Prado |
| 13/06 | 1.0.4 | Edição da seção 5 do Documento | Adriano Souza  Luana Vidal Leão  Jardeson Nogueira |
| 16/06 | 1.0.5 | Edição da seção 2 do Documento | Adriano Souza |
| 16/06 | 1.1.0 | Refinamento dos modelos VCP | Luana Vidal Leão  Mairon Azevedo |
| 16/06 | 1.1.1 | Refinamento dos casos de uso | Daniel Neves |
| 16/06 | 1.1.2 | Adição do diagrama de classes | Beatriz Dellatorre  Jardeson Nogueira |
| 20/06 | 1.1.3 | Criação dos diagramas de sequência | Mairon Azevedo  Beatriz Dellatore  Vitor Prado  Denerson Eduardo |
| 20/06 | 1.1.4 | Justificativa dos modelos arquiteturais escolhidas | Jardeson Nogueira  Vitor Prado |
| 21/06 | 1.1.5 | Adição dos Padrões Builder e Strategy na seção 7 do Documento | Adriano Souza  Daniel Neves |
| 21/06 | 1.2.0 | Adição dos padrões de projeto GOF e os seus respectivos modelos de instanciação. | Luana Vidal Leão  Denerson Eduardo |
| 30/06 | 1.2.1 | Refatoração de alguns padrões de projeto, solucionando problemas na aplicação. | Beatriz Dellatorre  Adriano Souza |
| 01/07 | 1.2.2 | Adição do Padrão Factory Method na seção 7 do documento | Beatriz Dellatorre |
| 01/07 | 1.2.3 | Adição do padrão Bridge na seção 7 do documento | Mairon Azevedo |
| 02/07 | 1.2.4 | Adição dos padrões Singleton e DAO na seção 7 do documento. | Vitor Prado |
| 08/07 | 1.2.5 | Elaboração da Seção 9 do Documento, adição do link para Documento com os requisitos arquiteturalmente relevantes, ajuste do diagrama de casos de uso. | Denerson Eduardo  Luana Vidal Leão |
| 08/07 | 1.2.6 | Criação e elaboração da seção 9 do Documento | Adriano Souza |
| 12/07 | 1.2.7 | Ajuste na proposta do escopo do projeto, correção dos diagramas VCP e de Sequência do caso de uso de Login, adição do diagrama de VCP para o novo caso de uso Fazer Pedido e inclusão da seção de mecanismos arquiteturais. | Daniel Neves  Denerson Eduardo  Mairon Azevedo |
| 12/07 | 1.2.8 | Edição da seção 6 do Documento | Adriano Souza  Daniel Neves  Luana Vidal Leão |
| 13/07 | 1.2.9 | Adição do diagrama do padrão módulo, adição do diagrama de sequência do caso de uso Fazer Pedido. | Jardeson Nogueira  Beatriz Dellatorre |

**Índice Analítico**

[**Introdução**](#_gqfnzr3ygbkz) **5**

[**Finalidade**](#_ylql1lqhll6w) **5**

[**Metas e Restrições da Arquitetura**](#_cr3og8jyw8yj) **5**

[**Suposições e Dependências**](#_w98nzaueaa29) **6**

[**Requisitos Arquiteturalmente Significativos**](#_f3dp4lmvcher) **6**

[**Decisões, Restrições e justificativas**](#_lwk9rsyje1kl) **6**

[**Mecanismos Arquiteturais**](#_gmkcpxnuaa8j) **6**

[**Camadas da Arquitetura**](#_eq2ehzawsw1) **7**

[**Visões da Arquitetura**](#_a6emhxe17fqs) **13**

[**Visão dos casos de uso**](#_ei95ejwb5qx7) **14**

[**Visões das classes participantes**](#_uplgsxsrhbim) **14**

[**Caso de Uso Efetuar Venda**](#_n54s5d6tz58v) **15**

[**Caso de Uso Emitir Relatório**](#_s6oibklog1g3) **16**

[**Caso de Uso Manter Cliente**](#_z7no27g3k6oc) **17**

[**Caso de Uso Manter Produto**](#_kryhtjc3d0ib) **18**

[**Caso de Uso Realizar Login**](#_74xz59h0dt92) **19**

[**Caso de Uso Fazer Pedido por Delivery**](#_zdnuuojr0z7a) **20**

[**Visão Lógica**](#_mu9o6ensigx1) **20**

[**Diagrama de Pacotes da Arquitetura**](#_cica0yevbh5w) **21**

[**Diagrama de Classes da Arquitetura**](#_oizwooocvujn) **22**

[**Diagramas de Sequência**](#_sw26cgd62sp9) **23**

[**Caso de Uso Efetuar Venda**](#_qt8uk94njcg4) **23**

[**Caso de Uso Emitir Relatório**](#_b1bejmwmahdi) **24**

[**Caso de Uso Manter Cliente**](#_eies26l9rs9z) **25**

[**Caso de Uso Manter Produto**](#_rp9n58z3lp5r) **26**

[**Caso de Uso Realizar Login**](#_6c9x55cj6xnr) **27**

[**Qualidade**](#_r1t4qo2y7aul) **27**

**Documento de Arquitetura de Software**

# Introdução

Este trabalho tem como objetivo modelar e projetar um software de gerenciamento de um mercado local, a “Vendinha do Seu Zé”. Ao longo deste documento, pode ser encontrado as descrições das Visões de Arquitetura, classes e as decisões que nortearam este projeto.

# Finalidade

Este documento oferece uma visão geral arquitetural abrangente do sistema, usando diversas visões arquiteturais para representar diferentes aspectos do sistema. O objetivo deste documento é capturar e comunicar as decisões arquiteturais significativas que foram tomadas em relação ao sistema. De forma geral, o sistema desenvolvido buscará possibilitar que os funcionários do mercado consigam realizar vendas, gerenciar clientes e produtos, emitir alguns tipos de relatórios e monitorar o estoque de produtos. Também será possível que os clientes realizem pedidos por meio do delivery para que haja maior comodidade.

# Metas e Restrições da Arquitetura

A arquitetura de projeto utilizada neste trabalho é a Arquitetura MVC. Tal padrão de projeto organiza as classes do programa em três eixos: As de Visão, que são classes responsáveis pela apresentação da interface gráfica do sistema; as de Controle, que definem o comportamento da aplicação e que interpretam as ações do usuário e as mapeiam para chamadas das classes de Modelo, essas últimas que são responsáveis por gerenciar e controlar a forma como os dados se comportam por meio das funções, lógicas e regras de negócios estabelecidas.

A comunicação entre interfaces e regras de negócios é definida através de um controlador, que separa as camadas. Quando um evento é executado na interface gráfica, como um clique em um botão, a interface se comunicará com o controlador, que por sua vez se comunica com as regras de negócios.

Dado que a aplicação precisa se comunicar com um banco de dados MySQL para facilitar o acesso e gerenciamento dos dados, foi utilizado o estilo arquitetural Shared Data, com o objetivo de manter todas as informações residentes em um único ambiente.

Outro ponto, é que como o objetivo do sistema é realizar a maior quantidade de execuções e com a menor quantidade de erros possíveis, também foi escolhido o estilo arquitetural de microsserviços. Por isso, a parte do sistema responsável pelo backend funcionará de forma independente das demais, assim como as partes responsáveis pelo frontend e pelas manipulações do banco de dados.

E por último, a arquitetura cliente-servidor. Esta arquitetura será empregada uma vez que a sessão do sistema responsável pelo gerenciamento e armazenamento dos dados funcionará como um servidor, estando permanentemente online e que responderá às requisições das classes do frontend do sistema, essas últimas que terão o comportamento de cliente.

# Suposições e Dependências

* O projeto base da aplicação foi realizado em uma arquitetura antiga baseada em Java, será necessária uma refatoração.
* A arquitetura deverá seguir o padrão MVC para não destoar do antigo funcionamento do projeto.
* O banco de dados deverá ser desacoplado e disponibilizado por uma API externa.
* A linguagem escolhida para a refatoração será PHP devido a maior expertise dos integrantes do grupo

# Requisitos Arquiteturalmente Significativos

[Requisitos\_Arquiteturalmente\_Relevantes](https://docs.google.com/document/d/1feVACi_D9O_-kmNQ1rUduKsnZzQnAxTOVnJTk_5cZA8/edit?usp=sharing)

# Decisões, Restrições e justificativas

A abordagem presente neste trabalho tem como objetivo dividir as classes do sistema em três categorias principais, definidas na arquitetura MVC: Model, View e Controller, sendo que, as classes relacionadas a interação com o usuário estão são as Views, as relacionadas à manipulação com o banco de dados, Controllers, e por último, as Models, que são responsáveis por conectar as duas primeiras e implementar as regras de negócio. A utilização do padrão MVC traz como benefício o isolamento das regras de negócios da lógica de apresentação, que é a interface com o usuário. Isto possibilita a existência de várias interfaces com o usuário que podem ser modificadas sem a necessidade de alterar as regras de negócios, proporcionando muito mais flexibilidade e oportunidades de reuso das classes.

Dentre as razões para desenvolver a aplicação utilizando como base um padrão de projeto, é que isso pode trazer alguns benefícios, como aumento de produtividade, uniformidade na estrutura do software, redução de complexidade no código, facilita a documentação, reduz o tempo de desenvolvimento, dentre outros.

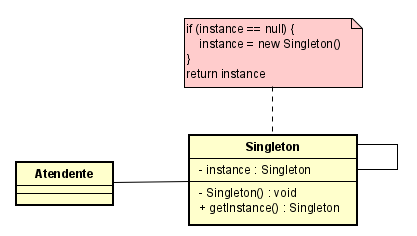
# Mecanismos Arquiteturais

|  |  |  |
| --- | --- | --- |
| *MECANISMO DE ANÁLISE* | *MECANISMO DE DESIGN* | *MECANISMO DE IMPLEMENTAÇÃO* |
| *Persistência* | *Banco de dados relacional* | *MySQL* |
| *Camada de distribuição* | *Classe de comunicação com o banco, classe de persistência.* | *PHP Data Objects(PDO)* |
| *Front-End* | *Interface de comunicação com o usuário do portal.* | *PHP, HTML, CSS, Bootstrap* |
| *Back-End* | *Estrutura que possibilita a operação do sistema.* | *PHP* |
| *Controle de Versão* | *Plataforma que possibilita o versionamento e gerenciamento dos códigos com Git.* | *GitHub* |

# Camadas da Arquitetura

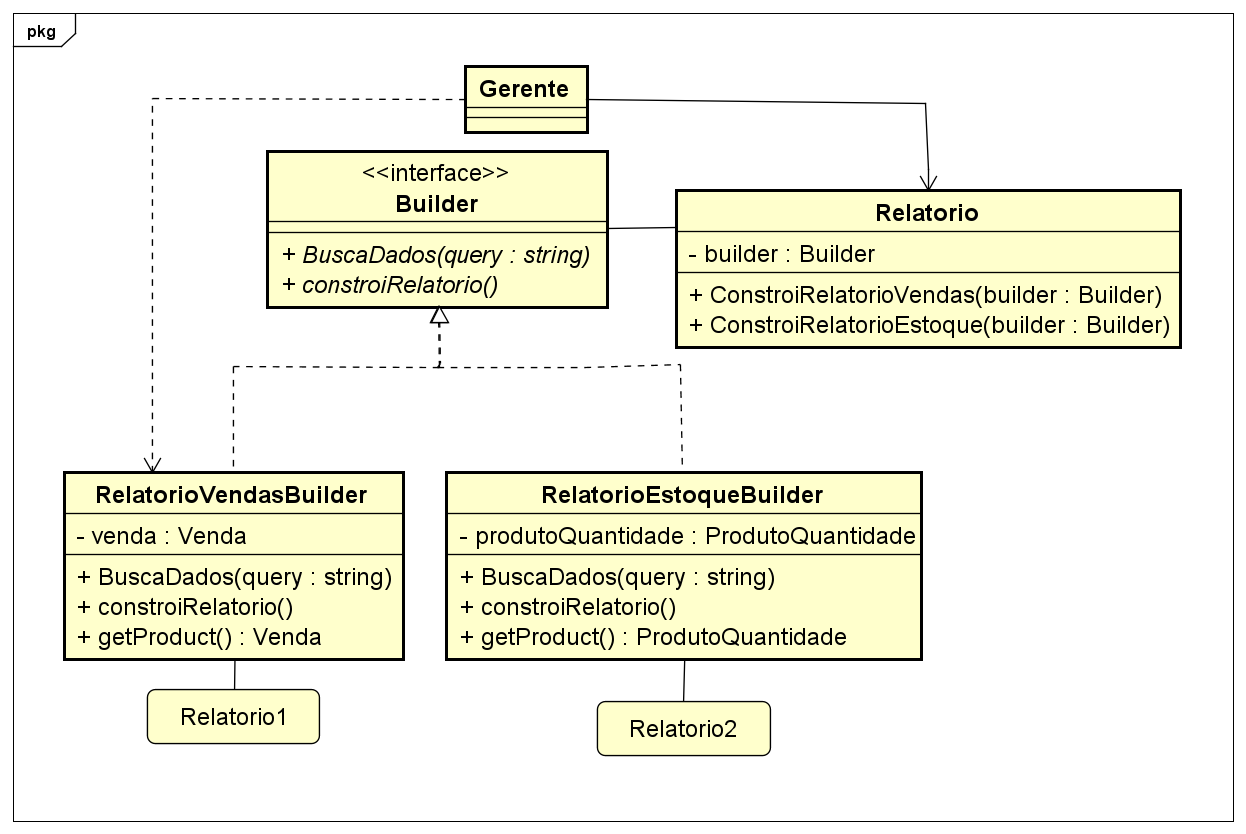
**SINGLETON**

A fim de controlar o acesso a recursos que são compartilhados, o padrão singleton foi adotado para a classe Funcionario, para garantir que somente um usuário possa interagir com o sistema por vez, vedando que múltiplos gerentes consigam acessar a base de dados simultaneamente e evitar possíveis inconsistências relacionadas ao estoque de produtos. Também foi incluído na classe de cliente para que o mesmo não seja instanciado várias vezes.



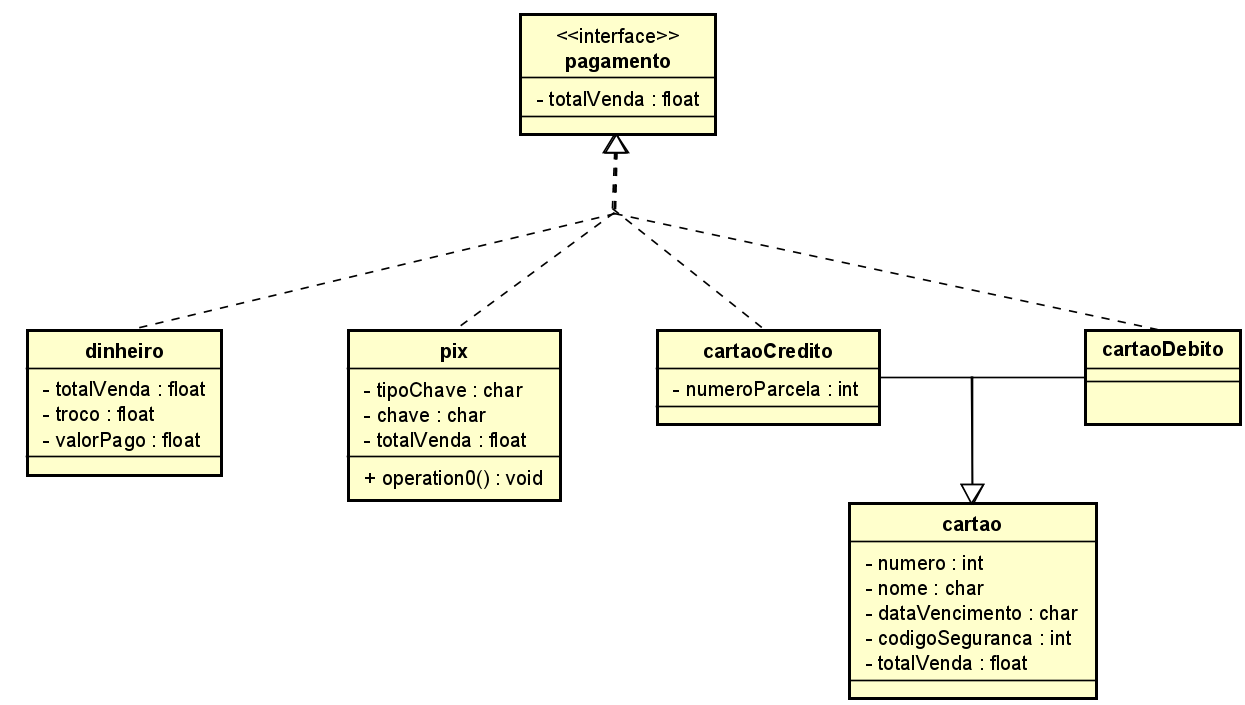
**BUILDER**

Uma das atribuições do sistema é a geração de diversos relatórios, e como a estrutura desses vários objetos é semelhante entre si, e com mudanças pontuais em pontos específicos do relatório. Assim, a utilização do padrão Builder se faz pertinente aqui, já que com ele podemos criar um mesmo objeto, nesse caso o relatório, com diversas opções de configurações, como a venda, data, produtos, quantidades, etc.



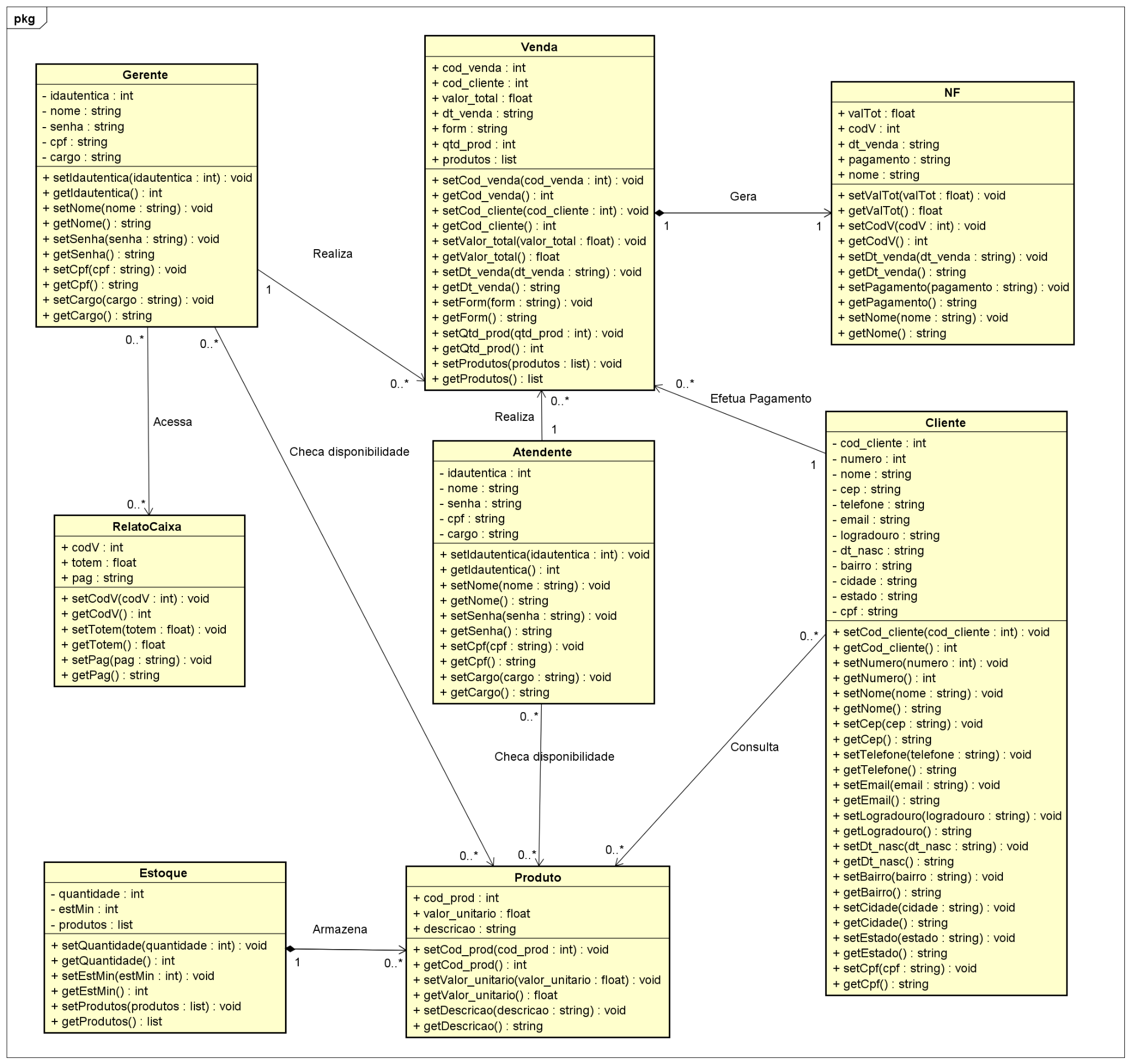
**STRATEGY**

O sistema, por lidar com meios de pagamentos, proporciona um ambiente que torna-se mais organizado e eficiente com a implementação do padrão Strategy. Como diferentes formas de pagamento são possíveis dentro do sistema, a interface Pagamento faz a assinatura o método pagar que irá ser implementado pelas classes PagtoDebito, PagtoCredito, PagtoValeRefeicao, PagtoPix e PagtoDinheiro com o intuito de atender as particularidades de cada meio de pagamento. Dessa forma, as responsabilidades e peculiaridades de cada forma de pagamento diferente são implementadas pelas classes que representam cada um desses mesmos pagamentos.



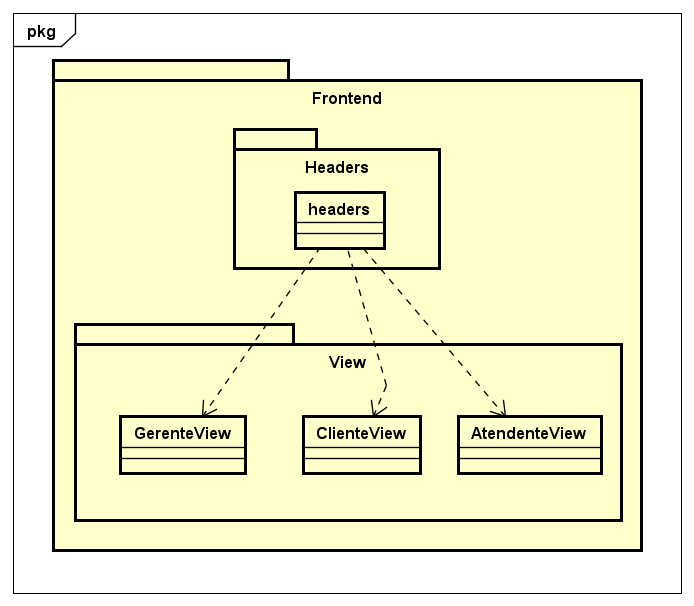
**DTO**

Para que houvesse uma diminuição de métodos chamados e para haver um objeto que, unicamente, aglutina e manipula os dados que vão ser carregados no banco de dados. Esse padrão vai auxiliar na transferência de dados entre as diferentes camadas da aplicação e também ajudará a reduzir a quantidade de acessos entre cliente e servidor.



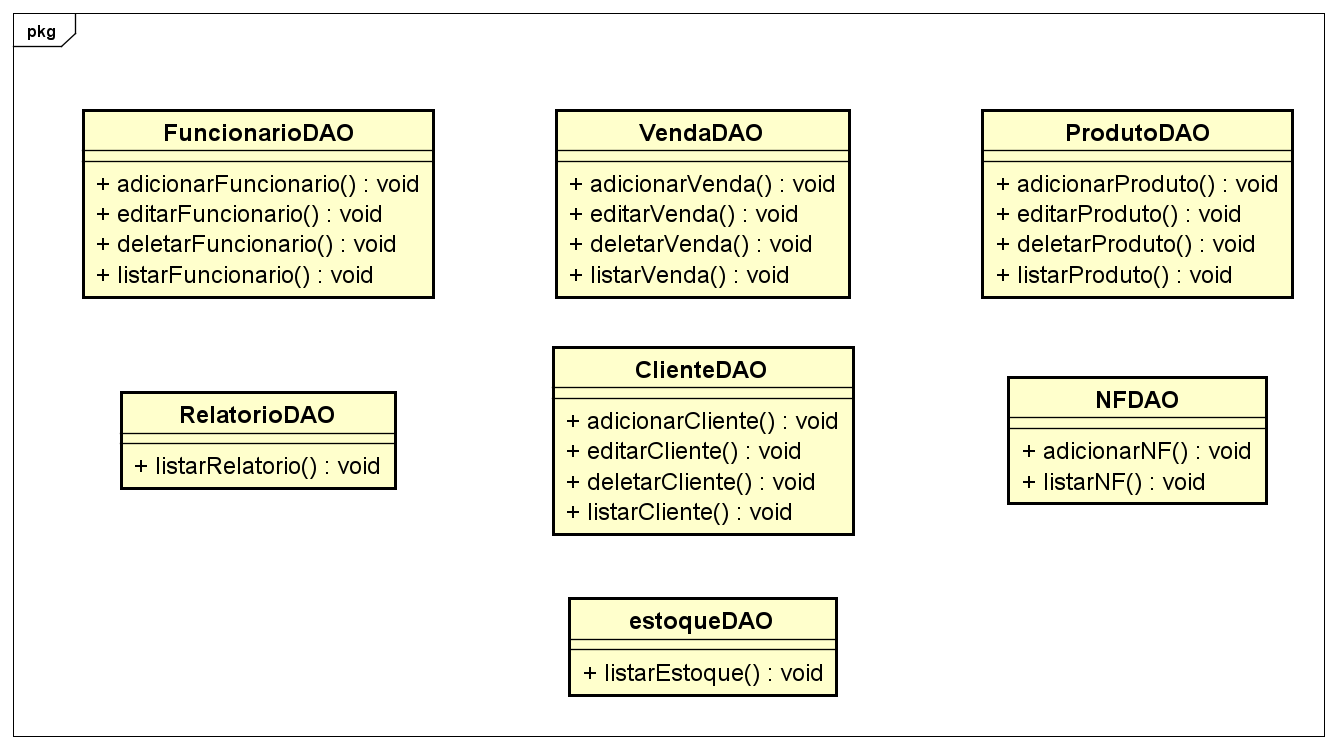
**MÓDULO**

O padrão modular foi utilizado na parte do frontend, com a implementação de headers que foram reutilizados em várias páginas. Os principais objetivos de segregar essas pequenas porções de código foram manter o código mais limpo, tornar a manutenção mais fácil e aproveitar a possibilidade de fazer o reuso desses fragmentos de código.



**DAO**

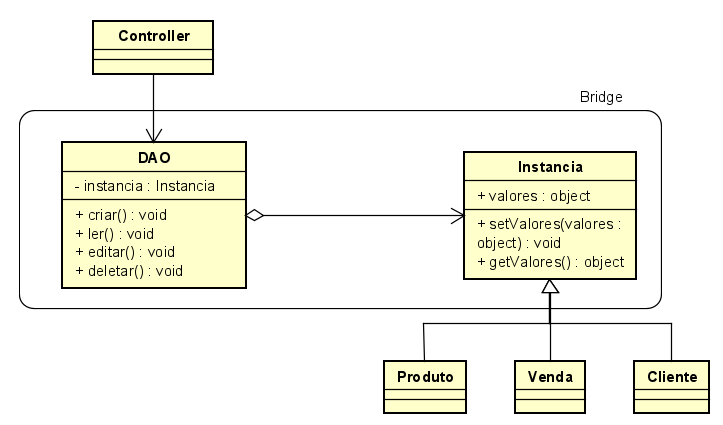
O padrão DAO (Data Access Object) é utilizado para separar a programação de manipulação de dados dos objetos da aplicação, assim facilitando na manutenção e deixando o código mais legível. Com isso, a lógica de persistência é centralizada nessas classes, fazendo com que a interação entre a base de dados e a aplicação seja feita por meio dessas implementações.



**BRIDGE**

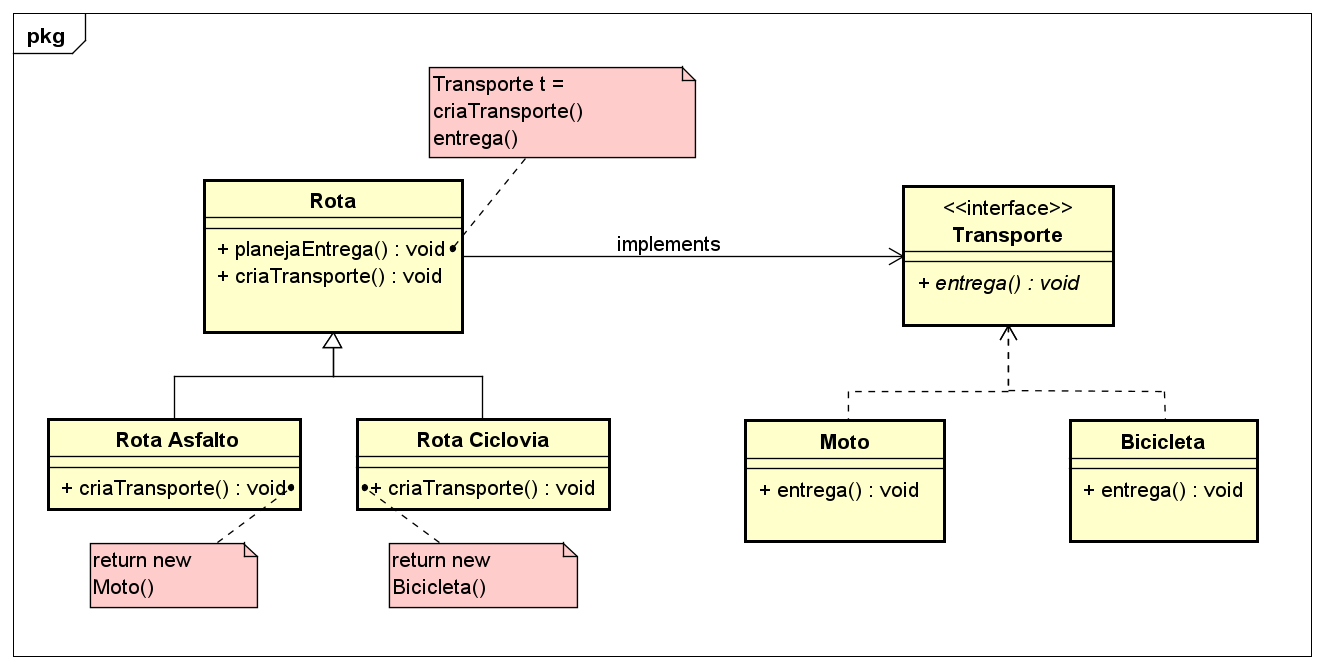
O padrão Bridge é utilizado para unificar tarefas realizadas em mais de um tipo de elemento em apenas um local. Este padrão possui uma divisão de classes ligadas em camadas de Abstração e Implementação, que são desenvolvidas de maneira independente.

O uso dele se justifica no nosso projeto para a realização do CRUD dos elementos da aplicação, por haver inúmeras classes que possuem esse mesmo comportamento, é uma estratégia interessante unificar em uma camada de Abstração uma classe que será capaz de realizar essas funções. Dessa forma, não importa quantas classes sejam criadas futuramente, seu CRUD sempre estará implementado.

~~~~

**FACTORY METHOD**

O padrão criacional Factory Method fornece uma interface para criar objetos em uma superclasse, mas permite que subclasses alterem o tipo de objetos que serão criados. No nosso projeto, ele será implementado na seção de entregas, visando criar rotas distintas tomadas por meios de transporte diferentes utilizando subclasses da classe Rota e implementando a interface Transporte.



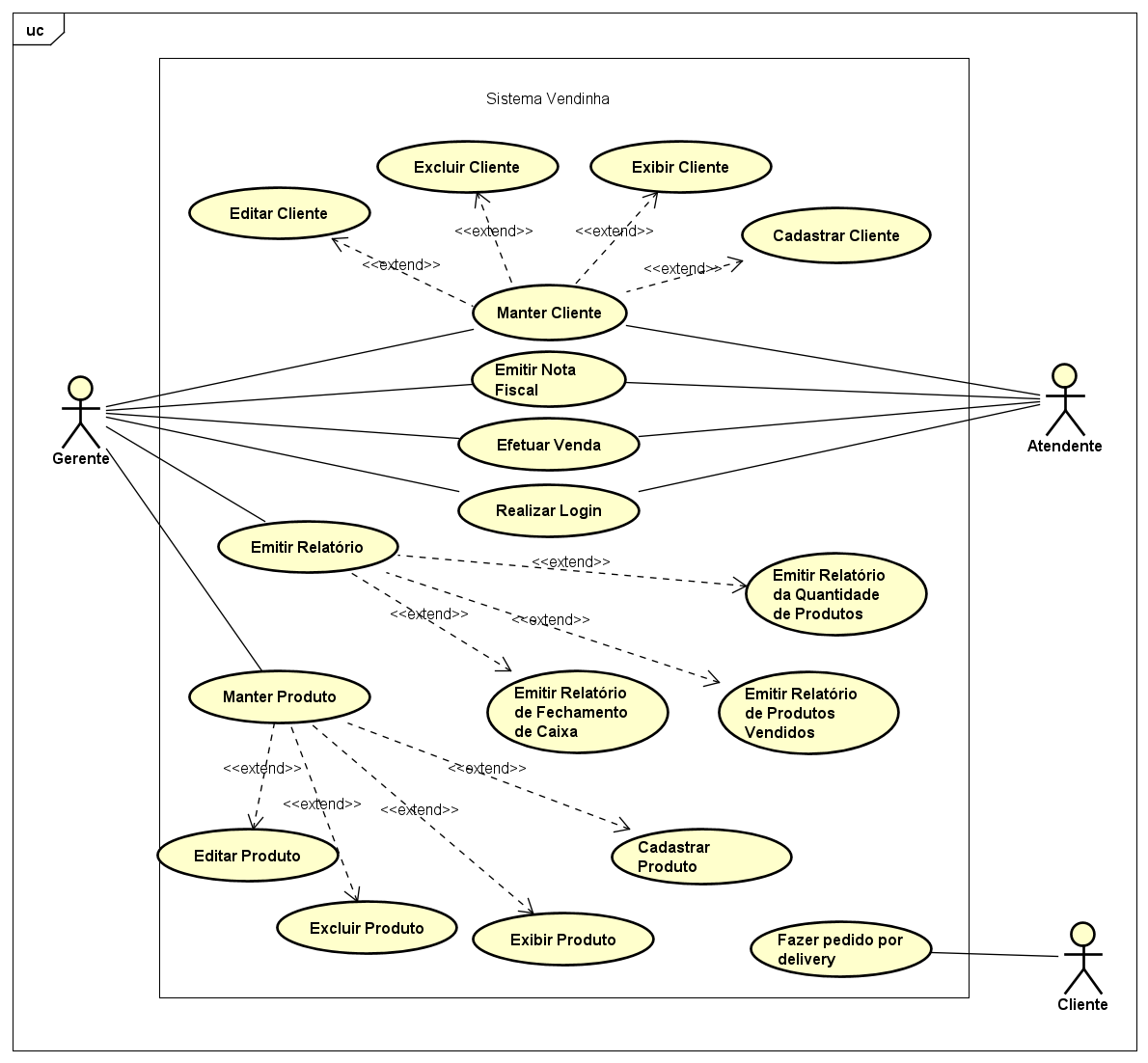
# Visões da Arquitetura

Visões Recomendadas:

* **Lógica:** Descreve a estrutura e comportamento de porções arquiteturalmente significativas do sistema. Isto deve incluir a estrutura de pacotes, interfaces críticas, importantes classes e subsistemas e as relações entre estes elementos. Isto também inclui visões físicas e lógicas dos dados persistentes.
* **Operacional:** Descreva os nós físicos do sistema e os processos, threads e componentes que rodam em cada um desses nós. Esta visão não é necessária se o sistema roda num único processo e num único thread.
* **Casos de Uso:** Uma lista ou diagrama dos casos de uso que contém requisitos arquiteturalmente relevantes.

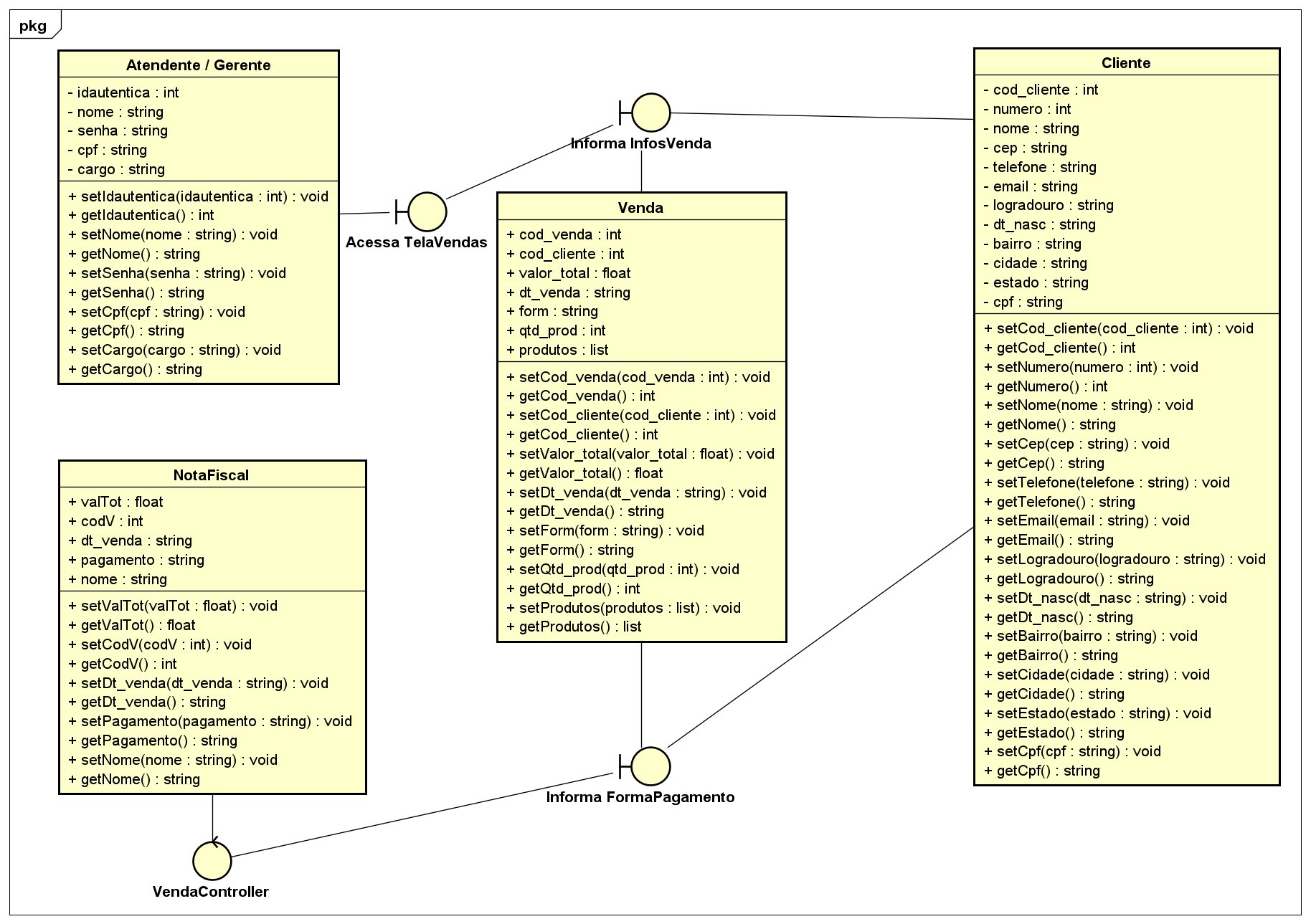
# Visão dos casos de usos

Os casos de uso do sistema Café Com Serviços serão listados abaixo.

****

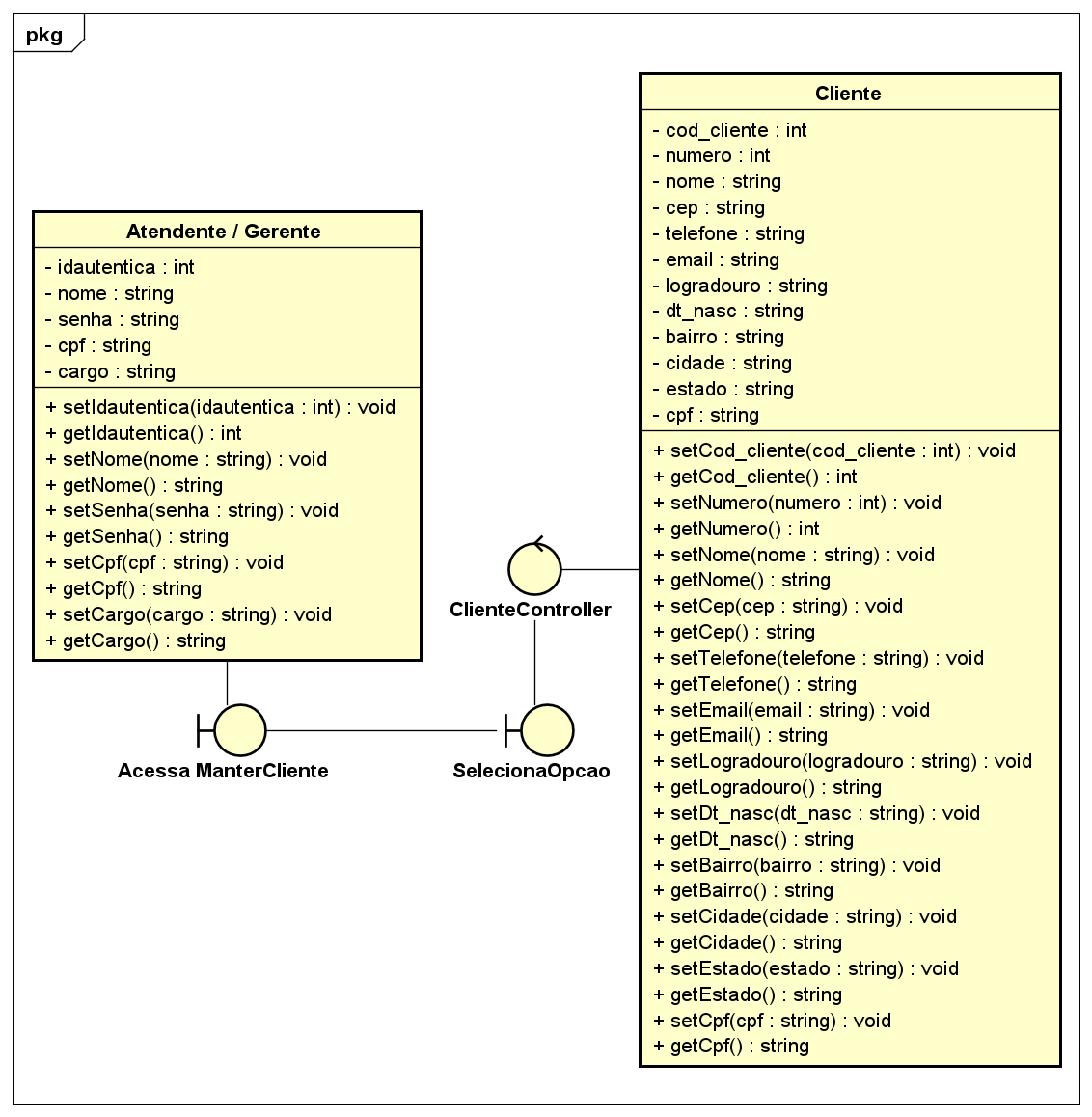
# Visões das classes participantes

# Caso de Uso Efetuar Venda

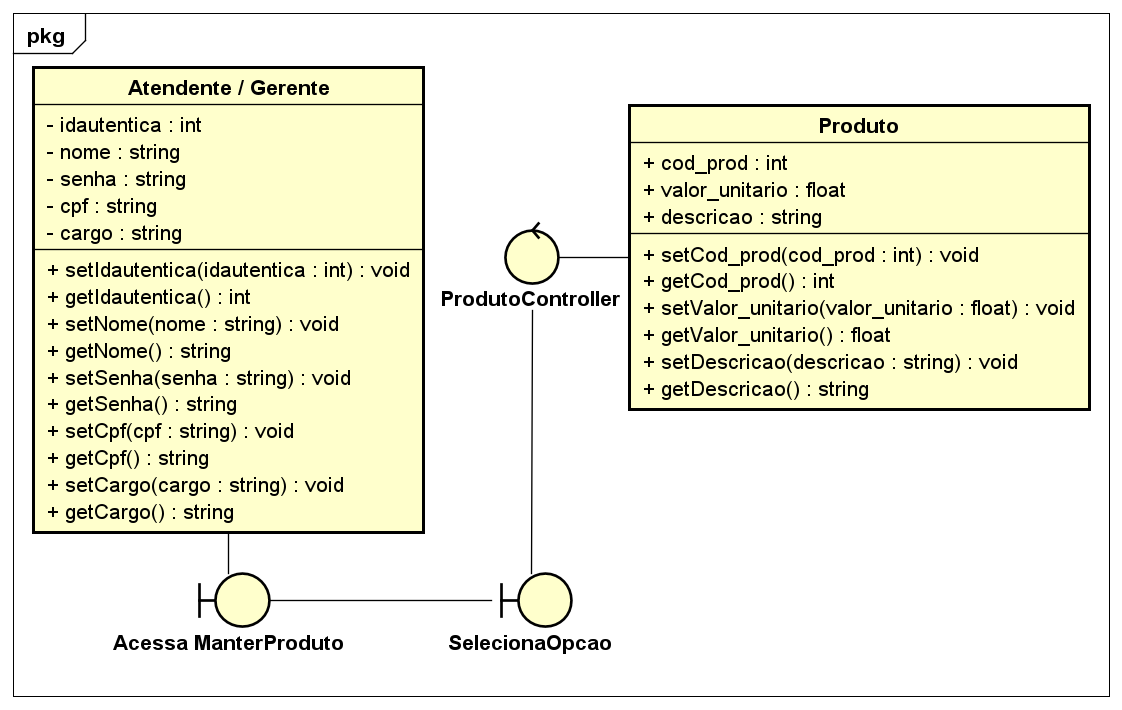


# Caso de Uso Emitir Relatório

# Caso de Uso Manter Cliente



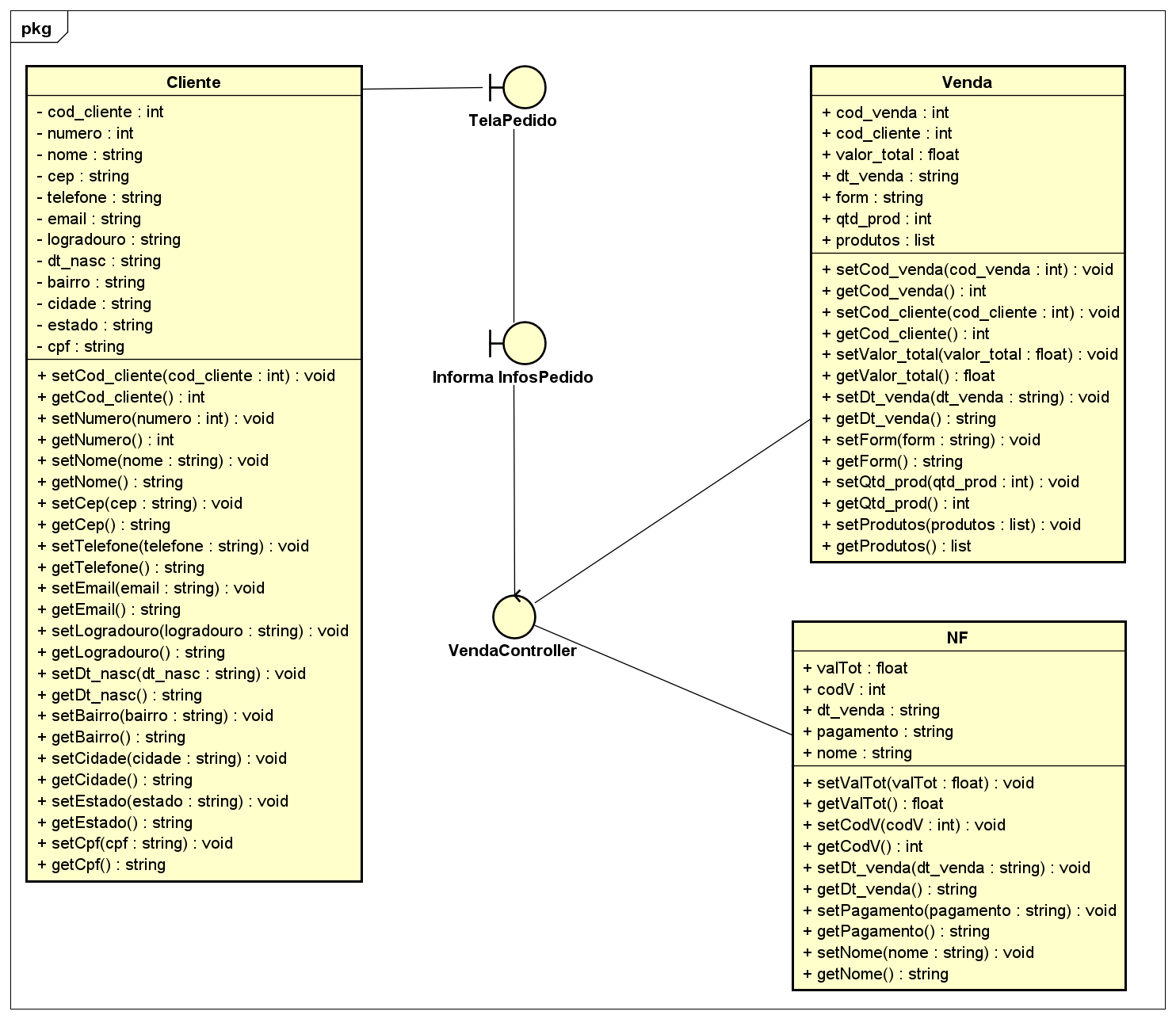
# Caso de Uso Manter Produto



# Caso de Uso Realizar Login

# 

# Caso de Uso Fazer Pedido por Delivery

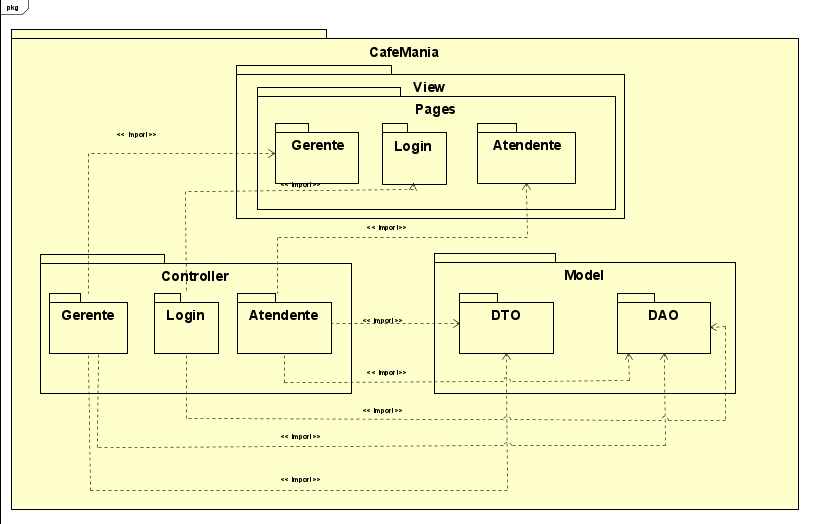


# Visão Lógica

A visão lógica define a estrutura da arquitetura. Abaixo será especificado o padrão utilizado para o desenvolvimento do sistema, no caso: MVC, Microsserviços, Shared-data e Cliente-Servidor.

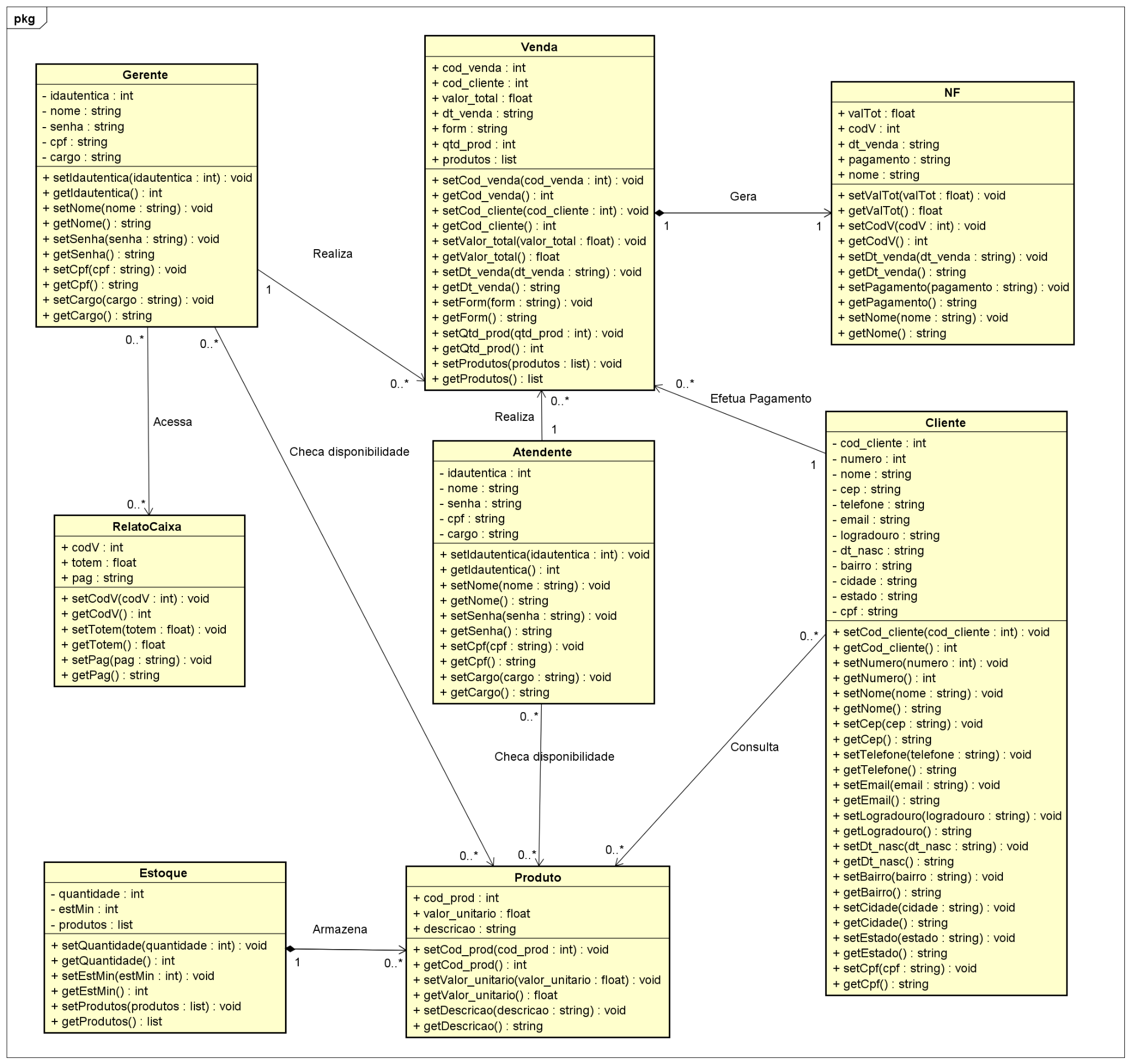
# 

# Diagrama de Pacotes da Arquitetura



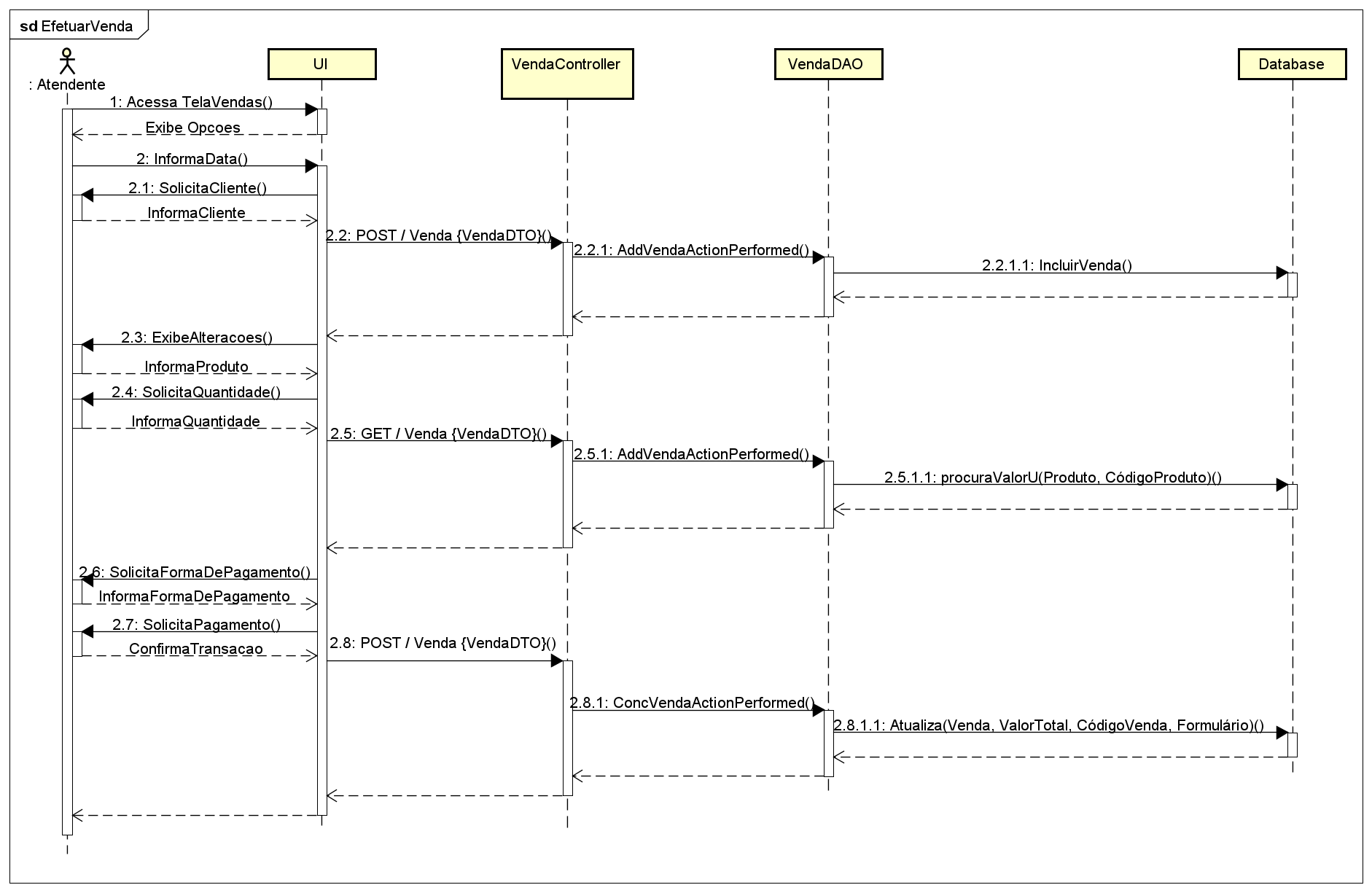
# 

# Diagrama de Classes da Arquitetura

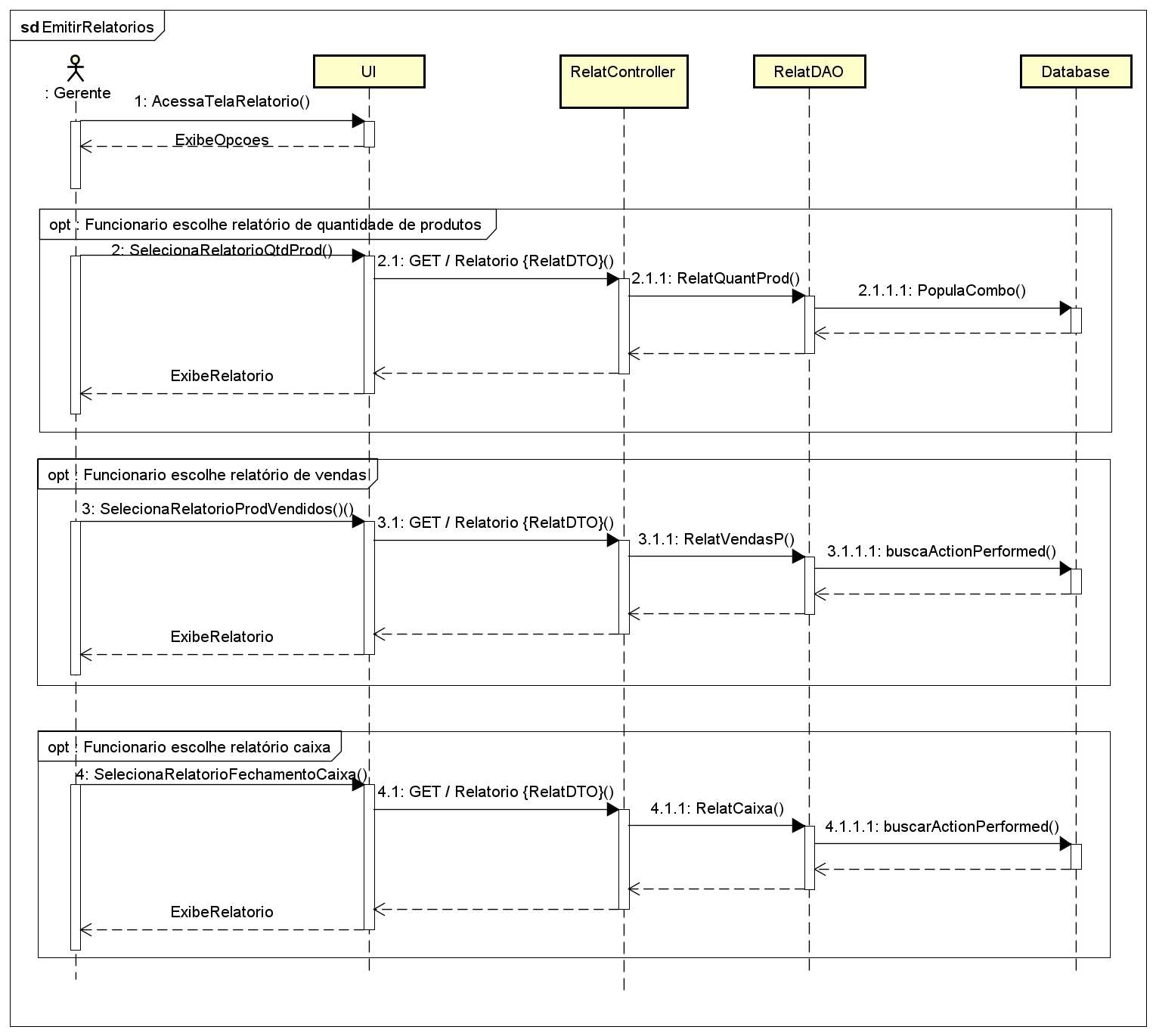


# Diagramas de Sequência

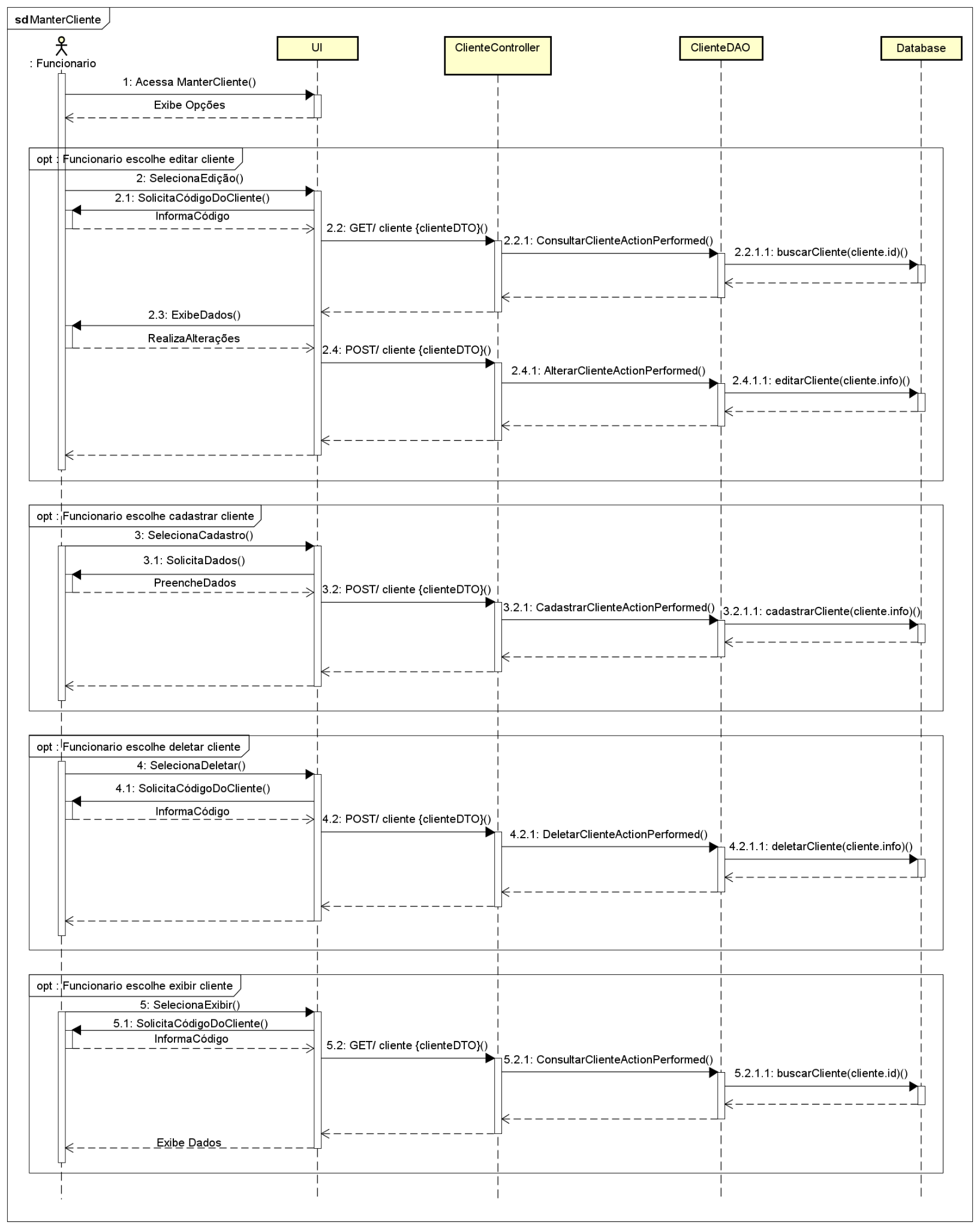
# Caso de Uso Efetuar Venda



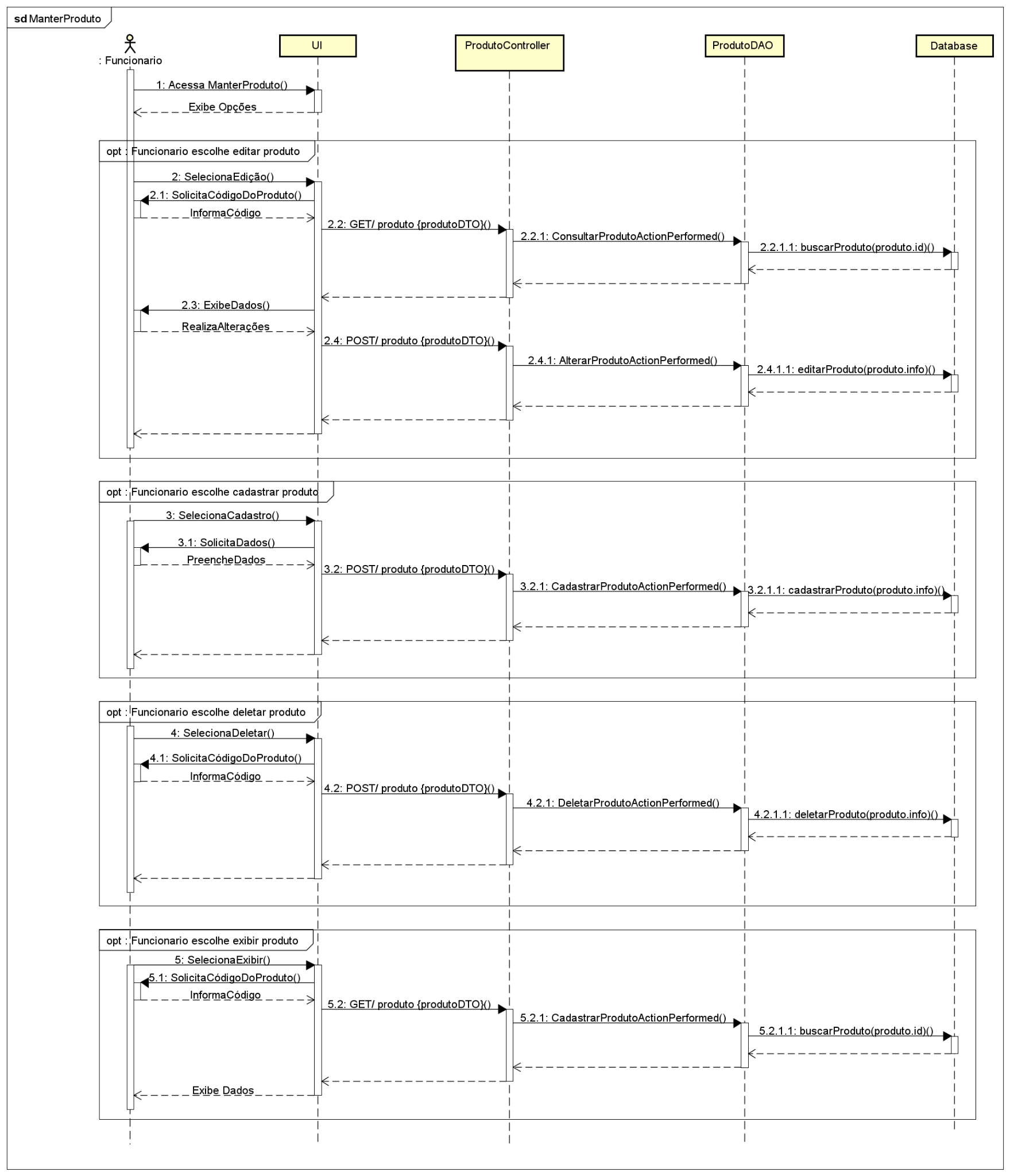
# Caso de Uso Emitir Relatório



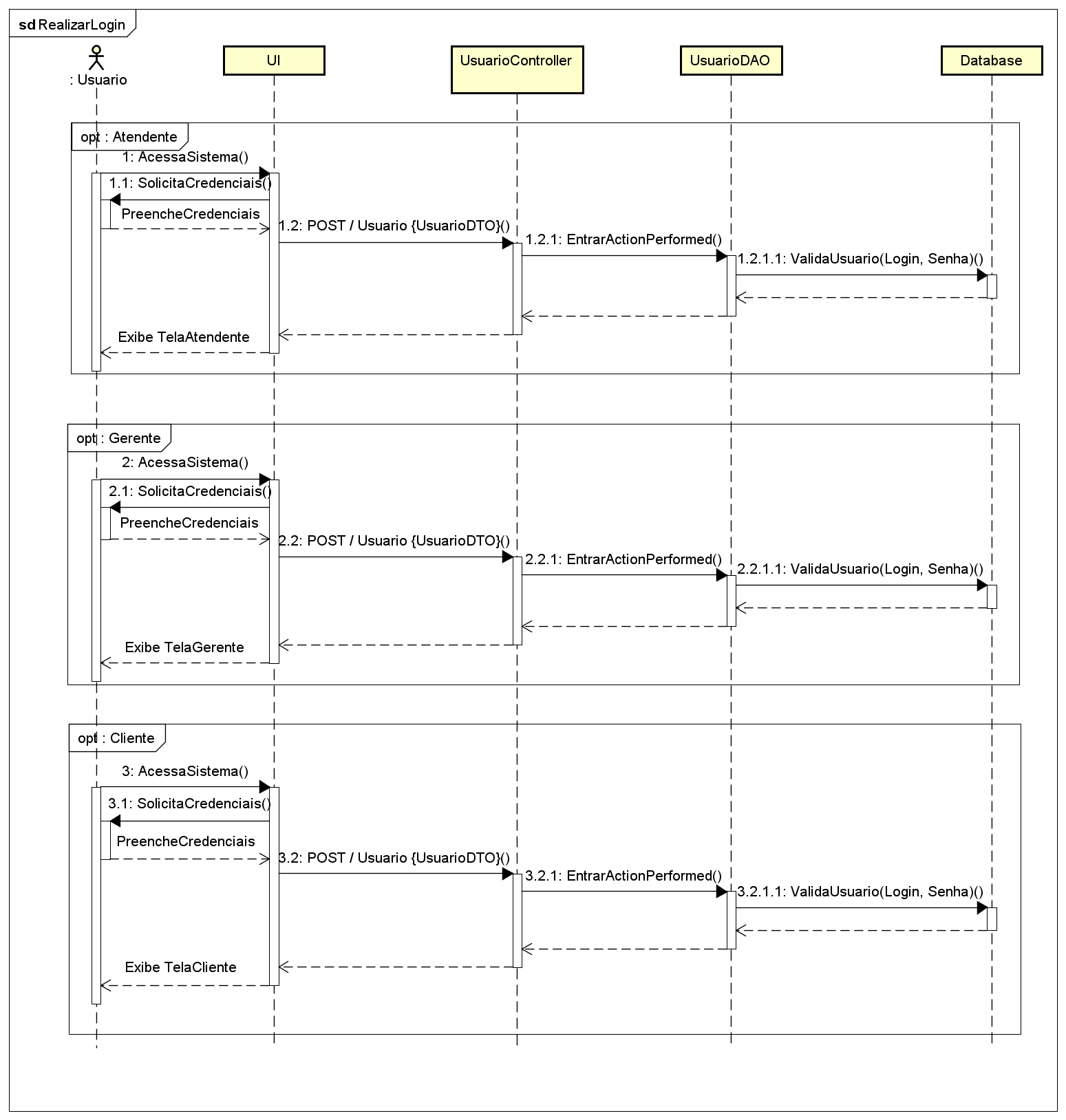
# Caso de Uso Manter Cliente



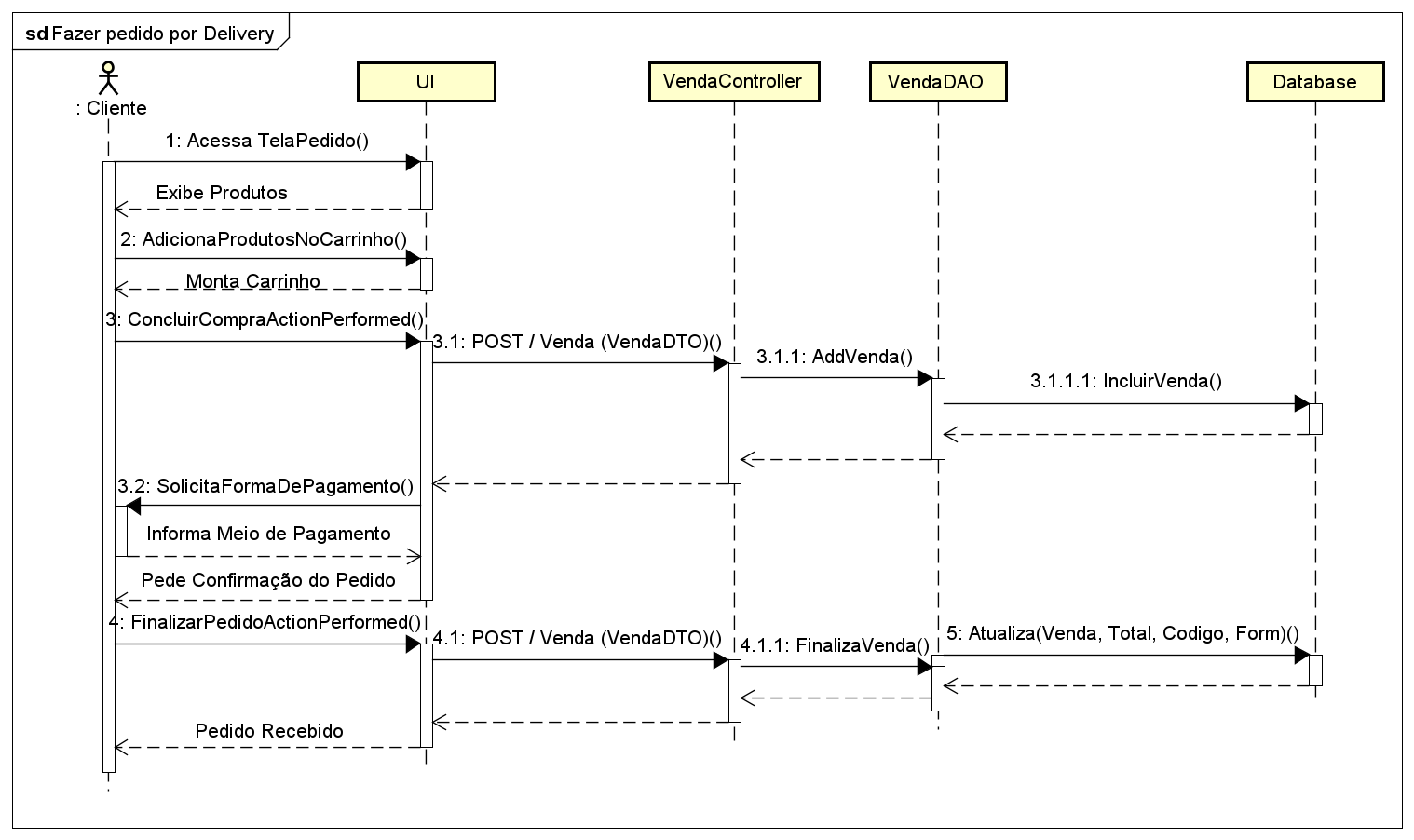
# Caso de Uso Manter Produto



# Caso de Uso Realizar Login



# Caso de Uso Fazer Pedido por Delivery



# Qualidade

A implementação de arquiteturas de software pré definidas na elaboração do projeto é de extrema importância para a construção de um código eficiente e organizado. É com a implementação das arquiteturas e padrões de projetos que conseguimos elaborar um ambiente que reduz consideravelmente a repetição do mesmo código escritos em várias classes diferentes, e que cria uma estrutura que consegue ser estendida com novas funcionalidades com mais facilidade, ao contrário de um código rígido e único. Um exemplo, foi a utilização do padrão Strategy para segregar o algoritmo de pagamento da classe responsável pelo pagamento, uma vez que, futuramente, um novo modo de pagamento pode ser inventado ou a cafeteria pode passar a aceitar outras formas de pagamento, e como o código responsável por essa parte não está implementado diretamente na classe, isso torna a aplicação flexível para atualizações e implementações futuras, contribuindo para a extensibilidade do sistema.

Além disso, a organização do projeto seguindo estes princípios é muito útil para a resolução de problemas. Dessa forma, caso algum erro ou comportamento indesejado seja identificado na aplicação, é mais fácil identificar em qual classe aquele problema está acontecendo, uma vez que um código flexível e distribuído, divide as suas responsabilidades em várias partes diferentes, facilitando a identificação do problema. Tal abordagem seria bem mais complicada de ser realizada em um projeto rígido e em desacordo com os padrões de software.