PORTFÓLIO

INSTITUIÇÃO:
Centro Universitário INSTED

CURSO:
Tecnologia em Análise e Desenvolvimento de Sistemas (TADS)

DISCIPLINA:
Desenvolvimento Orientado a Objetos e Padrões de Projeto

GRUPO:
Lucas Davi Aguilera Pache
Antonio Carlos Feliciano Terrinha da Silva Neto

DATA:
15/06/2025

Sumário

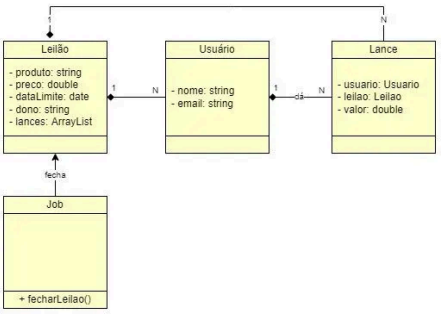
1. Principais Diagramas UML e seus Objetivos

A UML se divide em dois grandes grupos de diagramas: estruturais e comportamentais. Cada tipo de diagrama tem um papel específico na documentação e no entendimento do sistema.

1.1 Diagramas Estruturais

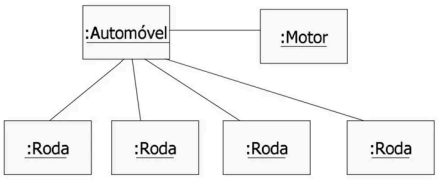
Visualizam a estrutura estática do sistema, mostrando seus componentes, relacionamentos e organização geral.

- **Diagrama de Classes:**
  Representa as classes do sistema, seus atributos, métodos e os relacionamentos entre elas. É fundamental para o design orientado a objetos, pois descreve a estrutura principal do software e serve como base para a implementação do código.

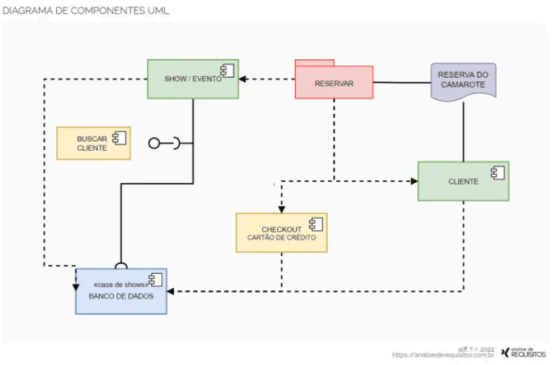


- **Diagrama de Objetos:**
  Mostra instâncias específicas de classes e seus relacionamentos em um dado momento, sendo útil para ilustrar exemplos concretos do funcionamento do sistema.

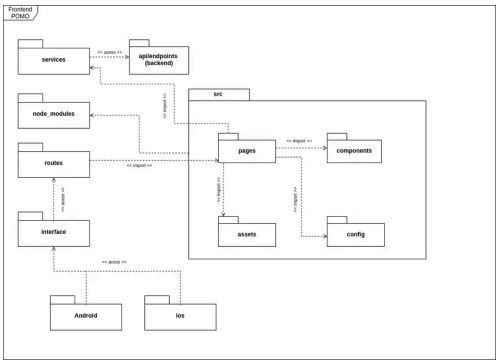
Diagrama de Objetos



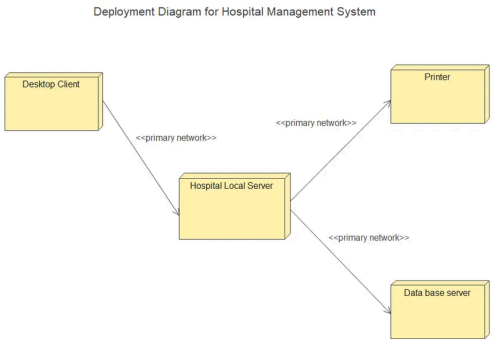
- **Diagrama de Componentes:**
  Representa os componentes físicos do sistema (módulos, bibliotecas, etc.) e suas dependências, facilitando o entendimento da arquitetura de software e a organização dos elementos reutilizáveis.



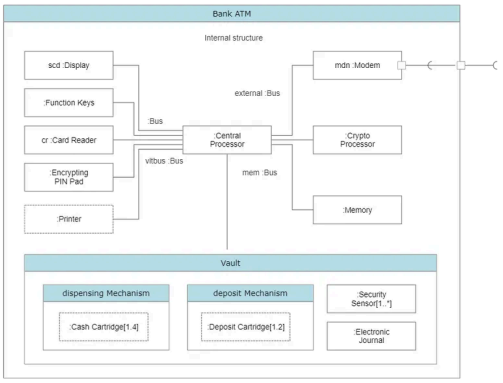
- **Diagrama de Pacotes:**
  Organiza as classes e outros elementos em pacotes, mostrando dependências e facilitando a gestão de grandes sistemas através de agrupamentos lógicos.



- **Diagrama de Implantação:**
  Visualiza a implantação física dos componentes de software em hardware, mostrando a distribuição do sistema em servidores, estações e outros dispositivos.



- **Diagrama de Estrutura Composta:**
  Detalha a estrutura interna de uma classe ou componente, mostrando suas partescolaborações internas.

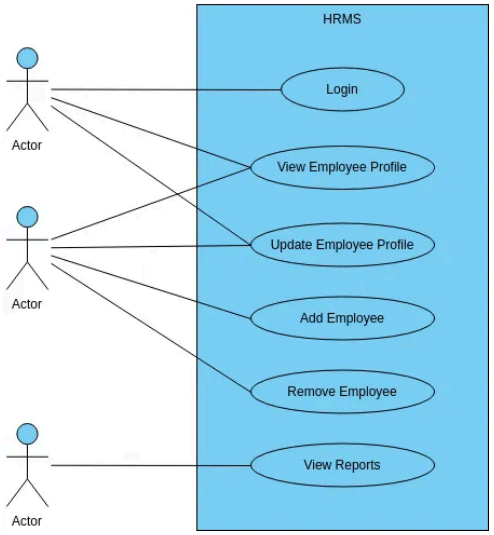


1.2 Diagramas Comportamentais

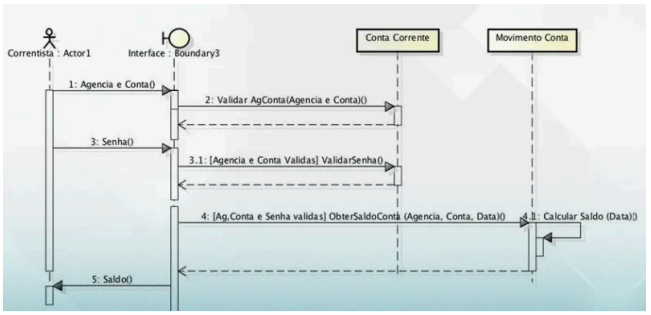
Capturam os aspectos dinâmicos do sistema, descrevendo como os componentes interagem e como o sistema responde a eventos.

- **Diagrama de Casos de Uso:**
  Mostra as funcionalidades do sistema do ponto de vista do usuário (atores) e os principais fluxos de interação, sendo essencial para levantamento de requisitos e

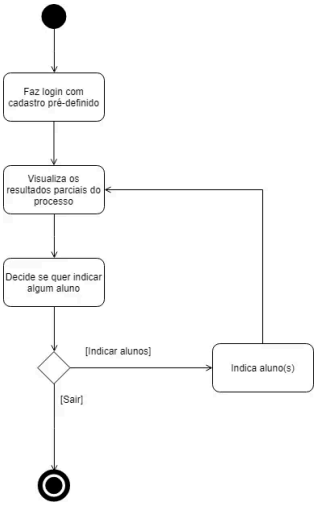
validação com o cliente.



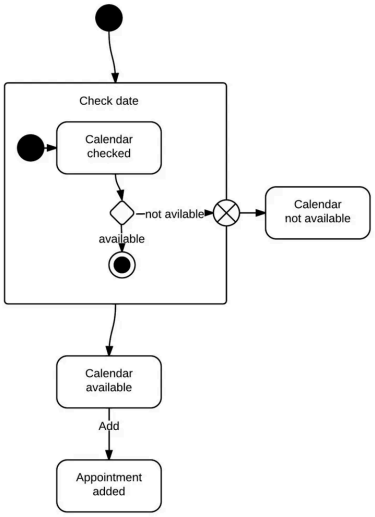
- **Diagrama de Sequência:**
  Demonstra a ordem das interações (mensagens) entre objetos ao longo do tempo, útil para detalhar cenários de uso e fluxos de processos.



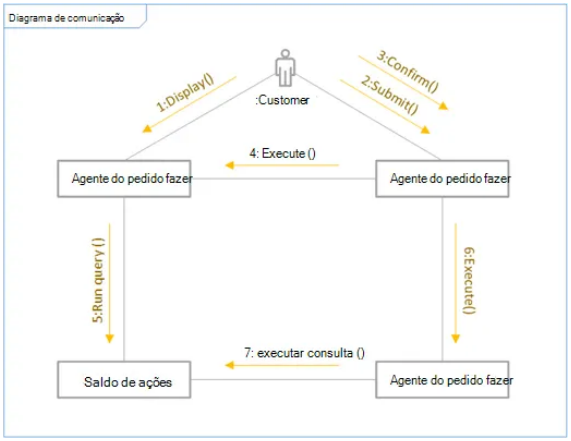
- **Diagrama de Atividades:**
  Visualiza o fluxo de trabalho ou de processos, mostrando a sequência de atividades, decisões e paralelismos. É semelhante a um fluxograma, sendo útil para descrever processos de negócio ou algoritmos.



- **Diagrama de Estados:**
  Representa os diferentes estados de um objeto durante seu ciclo de vida e as transições entre esses estados, importante para modelar comportamentos dependentes de eventos.



- **Diagrama de Comunicação:**
  Mostra como os objetos se relacionam e trocam mensagens, enfatizando as ligações e colaborações necessárias para realizar uma tarefa.



---

2. Como o Padrão MVC Organiza Sistemas

1. Model (Modelo):
O Modelo representa a lógica de negócios, os dados e as regras do sistema. É responsável por acessar, manipular e validar os dados. No contexto de um sistema de loja online, por exemplo, as classes Cliente, Produto, Venda e Carrinho de Compras fazem parte do Modelo. O Modelo não tem conhecimento sobre a interface do usuário, apenas sobre os dados e as operações possíveis sobre eles.

2. View (Visão):
A Visão é responsável pela apresentação das informações ao usuário. Ela exibe os dados que vêm do Modelo e envia as interações do usuário para o Controlador. Em um sistema

Python simples, a View pode ser representada por menus, telas ou relatórios impressos no console ou interface gráfica. A Visão não contém lógica de negócios, apenas a lógica de apresentação.

3. Controller (Controlador):
O Controlador atua como um intermediário entre a Visão e o Modelo. Ele recebe as ações do usuário (como cliques, comandos ou seleções), interpreta essas ações e chama os métodos apropriados do Modelo. Após a atualização dos dados, o Controlador pode solicitar à Visão que exiba as informações atualizadas. O Controlador é responsável por coordenar o fluxo de informações e garantir que as regras de negócio sejam respeitadas.

2.1 Benefícios do Padrão MVC

- **Separação de responsabilidades:** Cada camada tem uma função clara, facilitando a manutenção e a evolução do sistema.
- **Reutilização:** Componentes da Visão podem ser reutilizados com diferentes Modelos, e vice-versa.
- **Facilidade de testes:** É possível testar a lógica de negócios (Modelo) independentemente da interface (Visão).
- **Organização do código:** O sistema fica mais modular e organizado, facilitando o trabalho em equipe.

2.2 Exemplo Prático de Organização com MVC

Em um sistema de loja online em Python, a estrutura poderia ser:

- **Model:**
  Classes como Cliente.py, Produto.py, responsáveis por armazenar e manipular dados.
- **View:**
  Arquivos ou funções responsáveis por exibir menus, listar produtos, mostrar confirmações de compra, etc.

- **Controller:**
  Um arquivo main.py ou controladores específicos que recebem as ações do usuário, fazem validações, chamam métodos das classes do Modelo e atualizam a Visão.

2.3 Resumo

O padrão MVC organiza sistemas dividindo-os em três camadas:

- **Modelo (Model):** Gerencia os dados e regras de negócio.
- **Visão (View):** Responsável pela interface com o usuário.
- **Controlador (Controller):** Faz a mediação entre a visão e o modelo, controlando o fluxo da aplicação.

Essa separação torna o desenvolvimento mais eficiente, seguro e sustentável, sendo uma das principais práticas de arquitetura em sistemas modernos.

---

3. Princípio SOLID e sua Aplicação no Projeto

Os princípios SOLID são um conjunto de cinco diretrizes fundamentais da programação orientada a objetos, criados por Robert C. Martin e amplamente utilizados para melhorar a qualidade do código. Eles têm como objetivo tornar o software mais flexível, escalável, legível e fácil de manter, facilitando a evolução dos sistemas sem a necessidade de grandes modificações.

Os cinco princípios que compõem o SOLID são:

- **Single Responsibility Principle (SRP):** Princípio da Responsabilidade Única: Cada classe deve ter apenas uma responsabilidade, ou seja, uma única razão para mudar. Isso evita que uma classe acumule múltiplas funções, tornando o código mais organizado e fácil de manter.

- **Open/Closed Principle (OCP):** Princípio Aberto/Fechado: As entidades de software devem estar abertas para extensão, mas fechadas para modificação. Assim, é possível adicionar novas funcionalidades sem alterar o código existente, garantindo maior estabilidade ao sistema.

- **Liskov Substitution Principle (LSP):** Princípio da Substituição de Liskov: Objetos de uma classe derivada devem poder substituir objetos da classe base sem afetar o

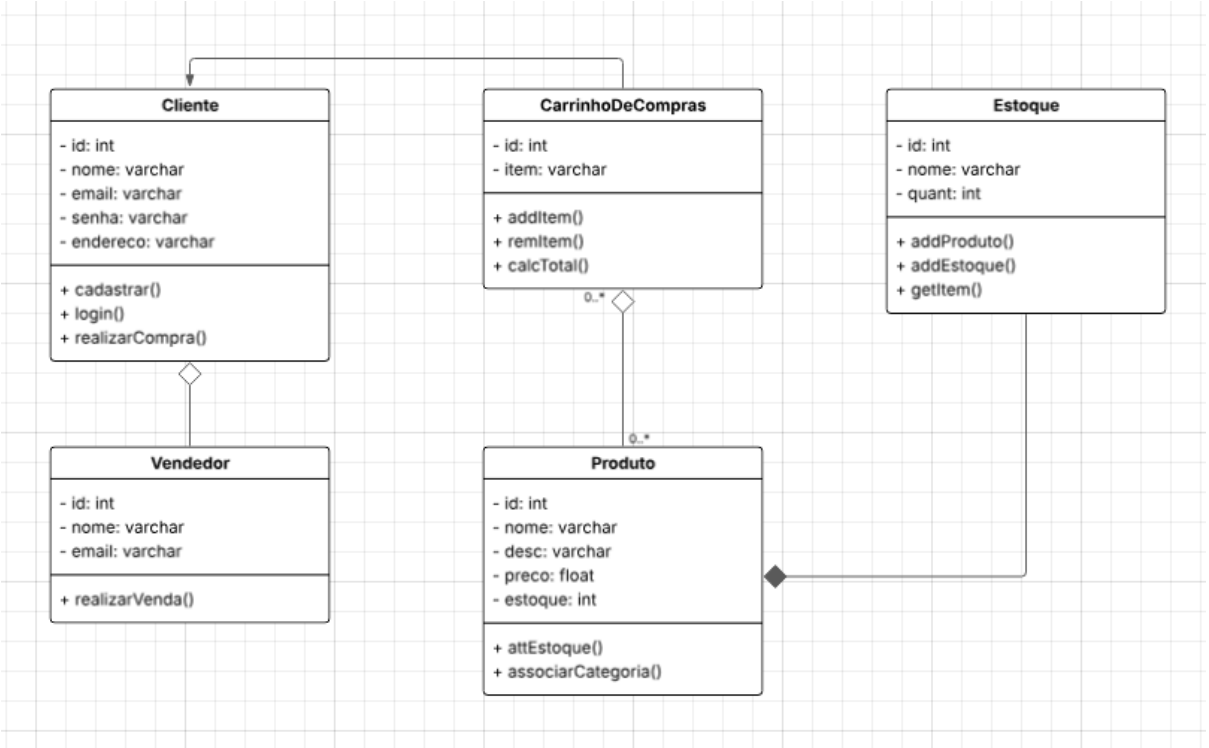
funcionamento correto do programa. Isso assegura a compatibilidade entre classes e promove a reutilização.

- **Interface Segregation Principle (ISP):** Princípio da Segregação de Interface: As interfaces devem ser específicas e conter apenas os métodos necessários para as classes que as implementam, evitando que uma classe seja obrigada a implementar métodos que não utiliza.

- **Dependency Inversion Principle (DIP):** Princípio da Inversão de Dependência: Módulos de alto nível não devem depender de módulos de baixo nível, mas ambos devem depender de abstrações. Isso reduz o acoplamento e aumenta a flexibilidade do sistema.

---

4. Diagrama de classe do nosso projeto:



---

5. Referencias

1. https://www.fecap.br/wp-content/uploads/2021/04/Manual-ABNT-2021-1.pdf
2. https://acailandia.ifma.edu.br/wp-content/uploads/sites/20/2023/06/TUTORIAL-DE-ACESSO-AS-NORMAS-DA-ABNT.pdf
3. https://abnt.org.br
4. https://www.mybib.com/pt/ferramentas/gerador-referencias-abnt