Discente: Ana Beatriz Reis Alves

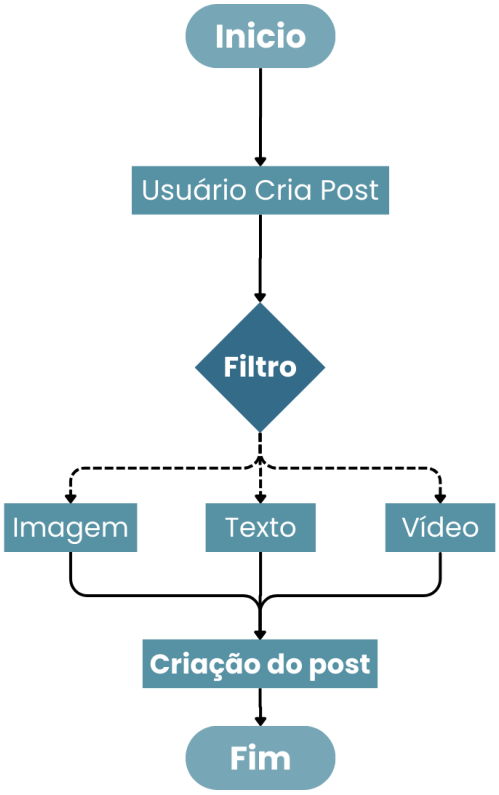
DESIGN PATTERNS

Diagrama de Fluxo:

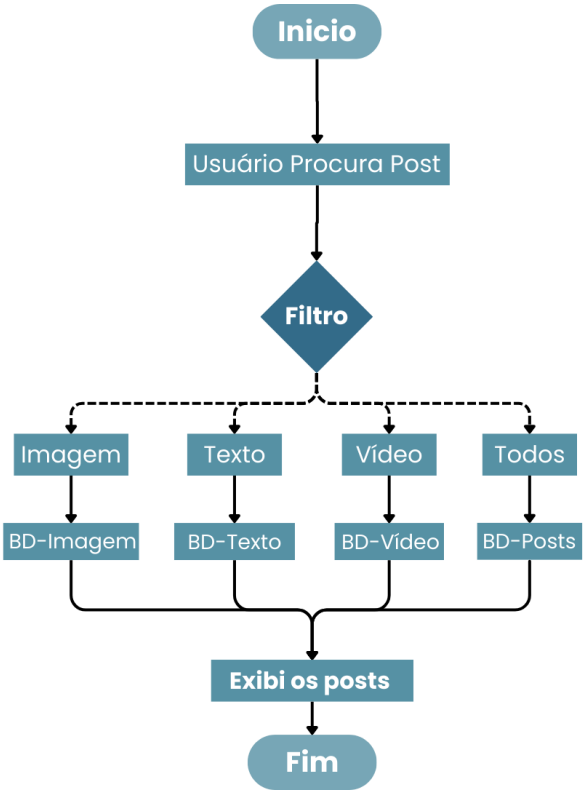
O diagrama de fluxo a seguir descreve como os usuários interagem com o sistema, desde a criação até a exibição de posts. Ele abrange os processos de criação, edição, exclusão, busca e visualização de posts na plataforma.

Fluxo de Interações:

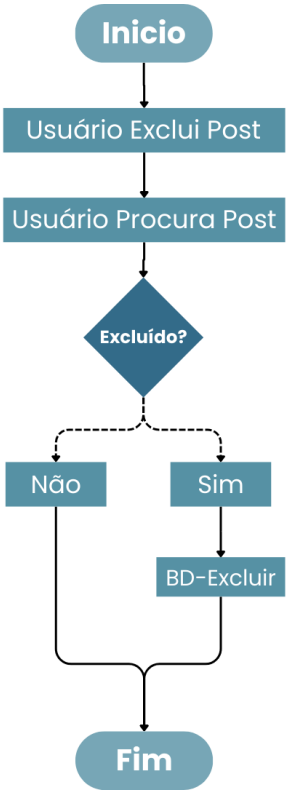
1. **Usuário Cria Post:**
    - O usuário inicia o processo clicando na opção para criar um novo post.
    - **Escolha do Tipo**: O usuário escolhe o tipo de post (Texto, Imagem, Vídeo).
    - **Criação do Post**: O sistema cria o post baseado no tipo selecionado (por exemplo, instanciando a classe correspondente via PostFactory).



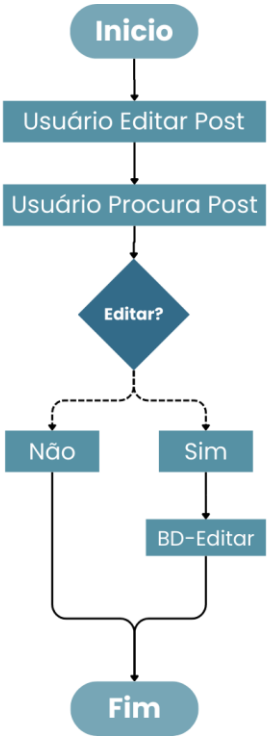
2. **Usuário Procura Post:**
    - O usuário decide buscar posts existentes na plataforma.
    - O usuário inicia o processo clicando na opção para procurar posts.
    - **Filtros**: O usuário pode filtrar a busca por tipo de post (Texto, Imagem, Vídeo ou Todos).
    - **Pesquisa no Banco de Dados**: A busca é realizada no banco de dados para encontrar posts que atendem aos critérios especificados.



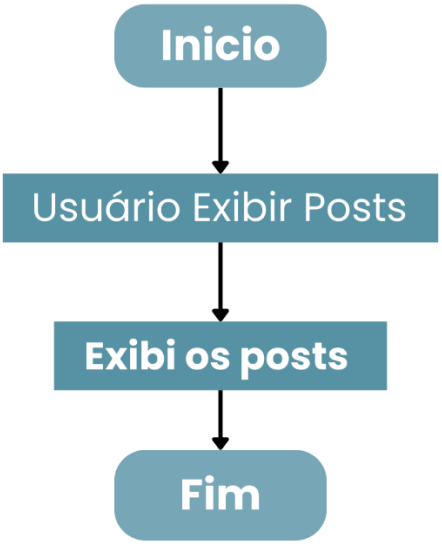
3. **Usuário Exclui Post:**
    - O usuário inicia o processo clicando na opção para procurar/editar posts.
    - O usuário procura o post (com **filtros** de tipo) e terá a opção de excluir o post.
    - **Alterações no Banco de Dados**: O post excluído é removido do banco de dados.



4. **Usuário Edita Post:**
    - O usuário inicia o processo clicando na opção para procurar/editar posts.
    - O usuário procura o post (com **filtros** de tipo) e terá a opção de editar o post.
    - **Alterações no Banco de Dados**: As modificações feitas (como alterações no conteúdo do post) são salvas no banco de dados.



5. **Exibição de Post:**
    - Os posts são exibidos na página inicial ou na página de busca de posts (com **filtros** de tipo).
    - **Leitura do Post**: O post é carregado do banco de dados e exibido na interface do usuário conforme os dados armazenados.



Análise do Diagrama de Classes(FACTORY METHOD)

O diagrama apresentado descreve um sistema para gerenciar diferentes tipos de posts (texto, imagem e vídeo). Vamos analisar cada classe e suas relações:

Post

- **Função:** Representa a classe base para todos os tipos de posts.
- **Atributos:**
  - strategy: Uma referência a uma estratégia que define como o post será exibido.
- **Métodos:**
  - Construtores e métodos para definir e obter a estratégia.
  - display(): Método abstrato que deve ser implementado pelas subclasses para exibir o conteúdo do post de acordo com a estratégia definida.
  - saveToDatabase(), editarPost(), readPost(), deletePost(): Métodos para interagir com o banco de dados, salvando, editando, lendo e deletando posts.

TextPost, ImagePost, VideoPost

- **Função:** Representam os tipos específicos de posts (texto, imagem e vídeo), herdando da classe base Post.
- **Atributos:**
  - id, texto: Atributos comuns a todos os tipos de posts.
  - imagemUrl: Atributo específico para posts de imagem.
  - videoUrl: Atributo específico para posts de vídeo.
- **Métodos:**
  - Implementações dos métodos abstratos herdados da classe base, como saveToDatabase(), editarPost(), etc., com a lógica específica para cada tipo de post.

PostStrategy

- **Função:** Define a interface para as diferentes estratégias de exibição de um post.
- **Métodos:**
  - display(): Método abstrato que deve ser implementado por cada estratégia para definir como o post será exibido.

Factory

- **Função:** Classe base para a criação de objetos. Neste caso, serve como base para a criação de diferentes tipos de posts.

PostFactory

- **Função:** Fábrica concreta para criar instâncias das classes TextPost, ImagePost e VideoPost.
- **Método:**
  - createPost(): Método que recebe o tipo de post e os dados necessários para criar uma nova instância do post correspondente.

Relacionamentos entre as Classes

- **Herança:** As classes TextPost, ImagePost e VideoPost herdam da classe base Post, o que significa que elas compartilham os atributos e métodos da classe base e podem ter seus próprios atributos e métodos específicos.
- **Associação:** A classe Post tem uma associação com a interface PostStrategy, indicando que um post tem uma estratégia associada que define como ele será exibido.

- **Factory:** A classe PostFactory tem um relacionamento de herança com a classe base Factory e cria instâncias das classes concretas TextPost, ImagePost e VideoPost.

Funcionamento Geral

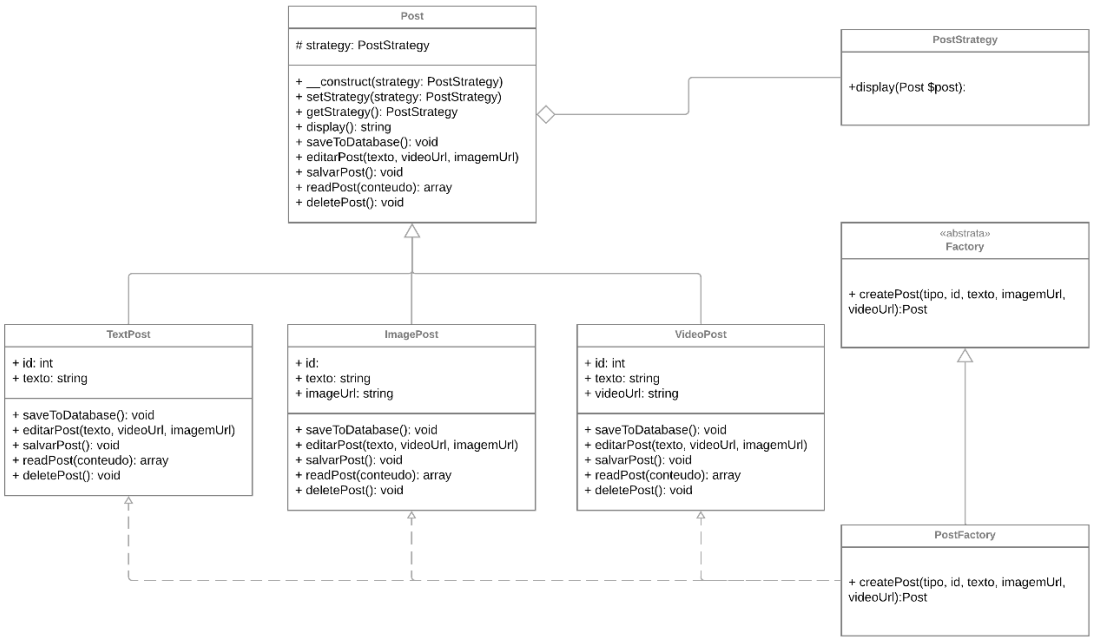
1. **Criação de um Post:**
    - É utilizado o método createPost() da classe PostFactory para criar uma nova instância de um post, especificando o tipo (texto, imagem ou vídeo) e os dados necessários.
2. **Definição da Estratégia:**
    - Ao criar um post, é associada a ele uma estratégia específica (implementação da interface PostStrategy) que define como o post será exibido.
3. **Exibição do Post:**
    - Ao chamar o método display() de um objeto Post, a estratégia associada é utilizada para formatar e exibir o conteúdo do post.
4. **Persistência:**
    - Os métodos saveToDatabase(), editarPost(), readPost() e deletePost() permitem persistir os dados dos posts em um banco de dados.

Benefícios da Arquitetura

- **Flexibilidade:** Permite adicionar novos tipos de posts no futuro sem alterar a estrutura principal do sistema.
- **Reutilização de código:** A classe base Post define a estrutura comum para todos os tipos de posts, evitando a duplicação de código.
- **Extensibilidade:** O padrão Strategy permite personalizar a exibição de cada tipo de post sem alterar a classe base.
- **Manutenibilidade:** A separação de responsabilidades entre as classes facilita a manutenção e o entendimento do código.

Em resumo:

O diagrama representa um sistema bem estruturado para gerenciar diferentes tipos de posts, utilizando conceitos como herança, polimorfismo e o padrão de projeto Strategy. Essa arquitetura oferece flexibilidade, extensibilidade e manutenibilidade, permitindo a criação de sistemas mais robustos e escaláveis.



Análise Detalhada do Diagrama de Classes(STRATEGY)

O diagrama de classes apresentado representa um sistema para gerenciar diferentes tipos de posts (texto, imagem e vídeo), utilizando o padrão de projeto Strategy para modularizar a forma como cada tipo de post é exibido.

Descrição das Classes e Relacionamentos

Classe Post:

- **Função:** Representa a classe base para todos os tipos de posts, definindo as operações comuns a todos eles.
- **Atributos:**
  - strategy: Uma referência a uma instância de PostStrategy, que define como o post será exibido.
- **Métodos:**
  - __construct(): Construtor que recebe uma estratégia como parâmetro.
  - setStrategy(): Define a estratégia para o post.
  - getStrategy(): Retorna a estratégia atual do post.
  - display(): Método abstrato que delega a exibição do post para a estratégia associada.
  - saveToDatabase(), editarPost(), readPost(), deletePost(): Métodos para interagir com o banco de dados, realizando as operações de persistência de dados.

Interface PostStrategy:

- **Função:** Define o contrato para as diferentes estratégias de exibição de um post.
- **Métodos:**
  - display(): Método abstrato que deve ser implementado por cada classe concreta para definir como o post será exibido.

Classes de Estratégia:

- **TextPostStrategy**, **ImagePostStrategy**, **VideoPostStrategy:**
  - Implementam a interface PostStrategy.
  - Cada classe possui uma implementação específica do método display() para exibir o conteúdo do post de acordo com seu tipo (texto, imagem ou vídeo).

Relacionamentos:

- **Herança:** As classes TextPost, ImagePost e VideoPost herdam da classe base Post, o que significa que elas compartilham os atributos e métodos da classe base e podem ter seus próprios atributos e métodos específicos.
- **Associação:** A classe Post tem uma associação com a interface PostStrategy, indicando que um post tem uma estratégia associada que define como ele será exibido.

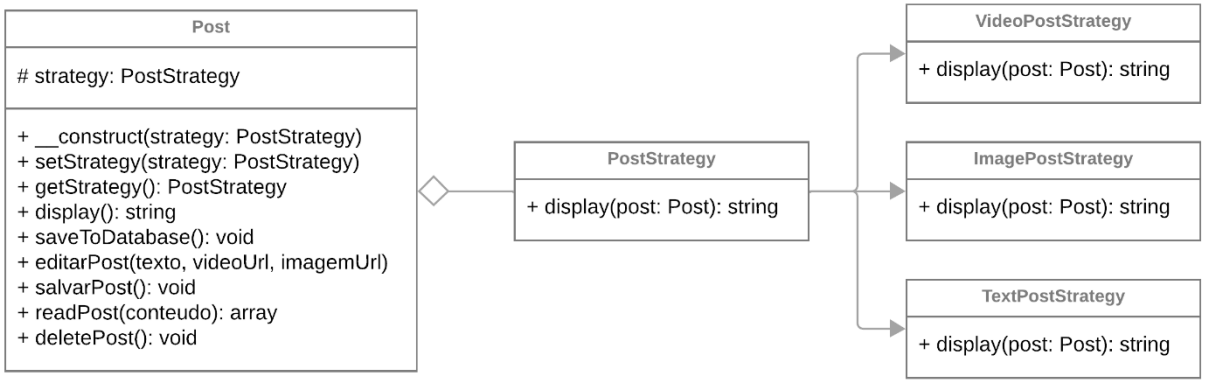
Benefícios da Arquitetura

- **Flexibilidade:** Permite adicionar novos tipos de posts no futuro sem alterar a estrutura principal do sistema.
- **Reutilização de código:** A classe base Post define a estrutura comum para todos os tipos de posts, evitando a duplicação de código.
- **Extensibilidade:** O padrão Strategy permite personalizar a exibição de cada tipo de post sem alterar a classe base.
- **Manutenibilidade:** A separação de responsabilidades entre as classes facilita a manutenção e o entendimento do código.

Funcionamento Geral

1. **Criação de um Post:**
    - É utilizado um método de fábrica (não mostrado no diagrama) para criar uma nova instância de um post, especificando o tipo (texto, imagem ou vídeo) e os dados necessários.
2. **Definição da Estratégia:**
    - Ao criar um post, é associada a ele uma estratégia específica (implementação da interface PostStrategy) que define como o post será exibido.
3. **Exibição do Post:**
    - Ao chamar o método display() de um objeto Post, a estratégia associada é utilizada para formatar e exibir o conteúdo do post.
4. **Persistência:**
    - Os métodos saveToDatabase(), editarPost(), readPost() e deletePost() permitem persistir os dados dos posts em um banco de dados.

Em resumo, o diagrama de classes apresentado representa uma solução bem estruturada e flexível para gerenciar diferentes tipos de posts, utilizando o padrão Strategy para promover a reutilização de código e a extensibilidade do sistema.



Análise do Diagrama de Classes (Façade)

Visão Geral

O diagrama apresentado descreve um sistema para gerenciamento de posts, utilizando o padrão de projeto Façade na classe PostManager. Essa classe atua como uma interface unificada para o cliente, encapsulando a complexidade do sistema e simplificando a interação com as demais classes.

Análise da Classe PostManager como Façade

A classe PostManager desempenha o papel de Façade, oferecendo uma interface simplificada para as operações relacionadas aos posts. Ao centralizar as operações de criação, busca, edição e exclusão de posts, a PostManager oculta a complexidade interna do sistema, como a interação com o banco de dados, a utilização do padrão Strategy e o mecanismo de logging.

Benefícios da utilização da Façade PostManager:

- **Simplificação da Interface:** A classe PostManager fornece uma interface única e intuitiva para o cliente, reduzindo a necessidade de conhecer os detalhes internos do sistema.

- **Isolamento de Mudanças:** Alterações na implementação interna das classes podem ser feitas sem afetar os clientes, desde que a interface da PostManager permaneça a mesma.
- **Reutilização de Código:** A PostManager pode ser reutilizada em diferentes partes do sistema, promovendo a coesão e a reutilização de código.
- **Controle de Acesso:** A PostManager pode ser utilizada para controlar o acesso às funcionalidades do sistema, implementando regras de negócio e segurança.

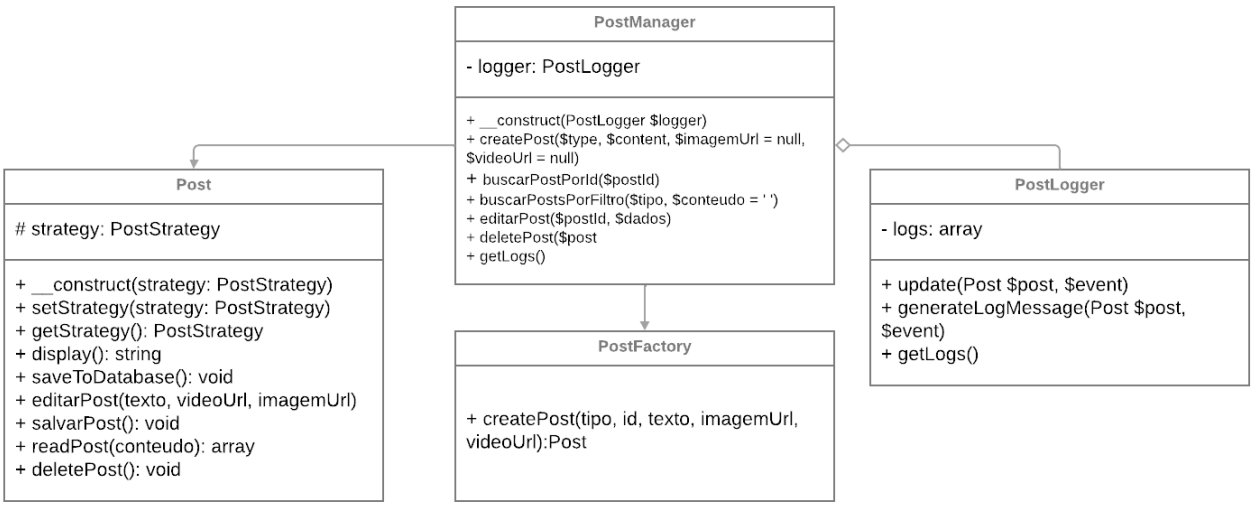
Funcionalidades da Façade PostManager:

- **Criação de Posts:** O método createPost encapsula a lógica de criação de um novo post, incluindo a definição do tipo, conteúdo e associação com a estratégia de exibição.
- **Busca de Posts:** Os métodos buscarPostPorId e buscarPostsPorFiltro permitem buscar posts por ID ou por critérios de filtro, simplificando a recuperação de dados.
- **Edição e Exclusão de Posts:** Os métodos editarPost e deletePost permitem modificar e excluir posts existentes.
- **Logging:** O atributo logger e os métodos relacionados permitem registrar as ações realizadas sobre os posts, facilitando a depuração e a análise do sistema.

Interação com Outras Classes

- **Post:** A PostManager interage com a classe Post para criar, buscar, editar e excluir instâncias de posts.
- **PostStrategy:** A PostManager utiliza as diferentes estratégias de exibição para formatar os posts de acordo com seu tipo.
- **PostLogger:** A PostManager utiliza o PostLogger para registrar as ações realizadas sobre os posts.

Em resumo, a classe PostManager desempenha um papel crucial na arquitetura do sistema, fornecendo uma interface unificada e simplificada para as operações relacionadas aos posts. Ao utilizar o padrão Façade, o sistema se torna mais fácil de entender, manter e estender.



Análise do Diagrama de Classes (OBSERVER)

O diagrama apresentado demonstra de forma clara a implementação do **padrão Observer**. Esse padrão de projeto define uma relação um-para-muitos entre objetos, onde um objeto, chamado de **sujeito** (neste caso, a classe PostManager), mantém uma lista de outros objetos dependentes, chamados de **observadores** (neste caso, a classe PostLogger). Quando o estado do sujeito muda, todos os seus observadores são notificados automaticamente.

No diagrama, podemos identificar:

- **Sujeito:** A classe PostManager é o sujeito. Ela mantém uma lista de observadores (embora não explicitamente mostrada no diagrama) e notifica esses observadores quando ocorrem mudanças nos posts, como criação, atualização ou exclusão.
- **Observador:** A classe PostLogger implementa a interface PostObserver e é um observador do PostManager. Quando o PostManager realiza uma operação que modifica um post, ele notifica o PostLogger através do método update.
- **Método update:** Esse método na interface PostObserver é o ponto central da comunicação entre o sujeito e os observadores. Quando chamado, ele informa ao observador sobre a mudança que ocorreu no sujeito.

Benefícios da Utilização do Padrão Observer neste Contexto

- **Desacoplamento:** A classe PostLogger não precisa conhecer os detalhes internos da classe PostManager para ser notificada sobre mudanças. Isso aumenta a coesão e a manutenibilidade do código.
- **Flexibilidade:** É possível adicionar novos observadores ao sistema sem afetar a classe PostManager ou outros observadores existentes. Por exemplo, poderíamos criar um observador que envia notificações por e-mail quando um novo post é criado.
- **Reutilização de código:** A interface PostObserver pode ser reutilizada em outros contextos onde se deseja notificar múltiplos objetos sobre mudanças em um objeto específico.

Em resumo, a utilização do padrão Observer neste diagrama de classes é uma excelente escolha, pois promove o desacoplamento, a flexibilidade e a reutilização de código. A implementação está bem estruturada e alinhada com os princípios do padrão.