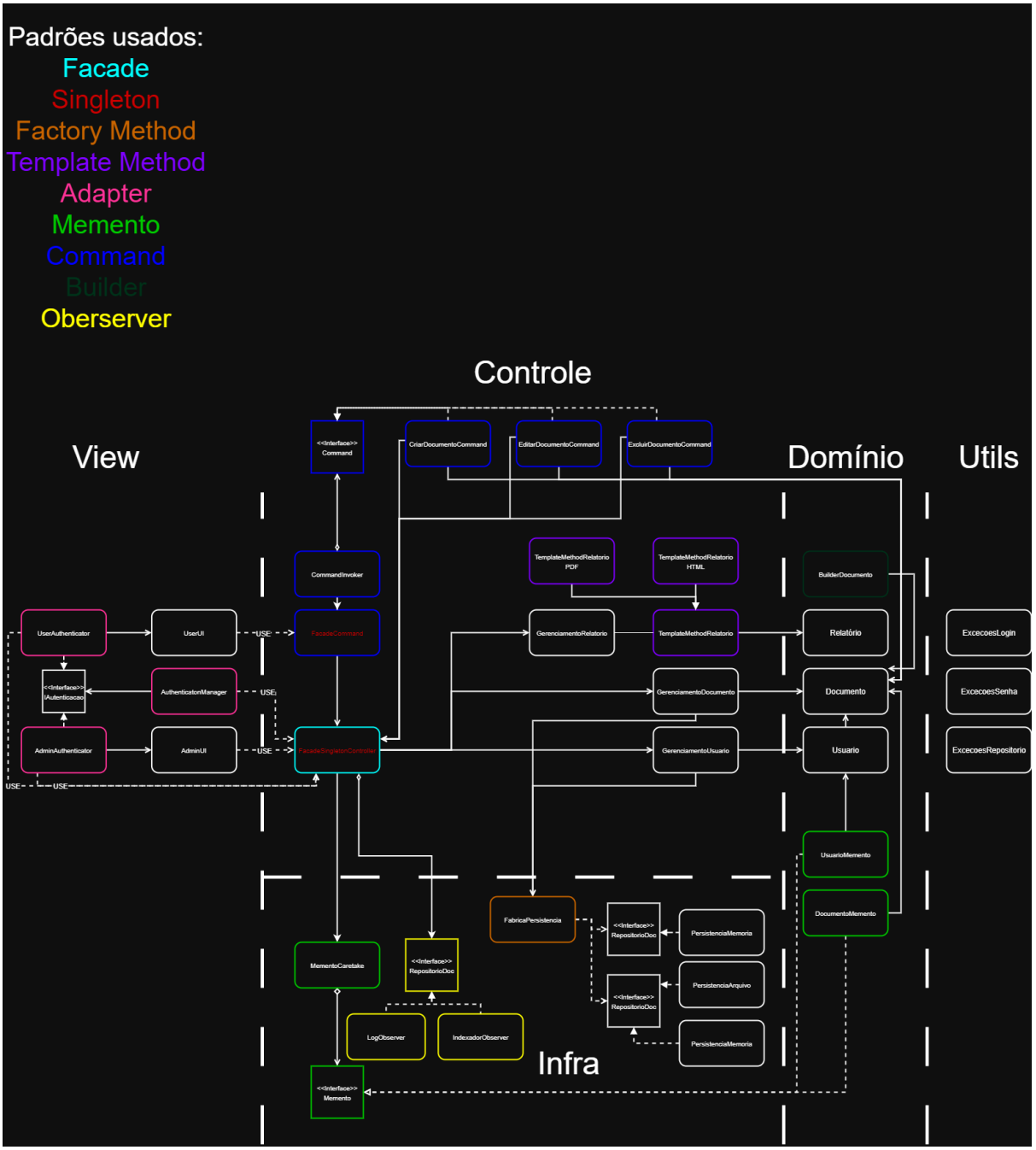
Descrição dos Padrões de Projeto Implementados



Padrão Command

Nome do Padrão: Command (Comando)
Lista de Classes que o Compõem:

- Command (interface)
- CriarDocumentoCommand (classe concreta)
- EditarDocumentoCommand (classe concreta)

- ExcluirDocumentoCommand (classe concreta)
- CommandInvoker (invocador)
- FacadeCommand (cliente)

Objetivo de Uso no Projeto:
O padrão Command é utilizado para encapsular solicitações como objetos, permitindo que operações do sistema sejam parametrizadas, enfileiradas, registradas e suportem funcionalidades de desfazer (undo) e refazer (redo). No projeto, ele é aplicado especificamente nas operações de gerenciamento de documentos realizadas pelos usuários comuns através da UserUI.
Isso permite que os usuários possam desfazer e refazer operações de criação, edição e exclusão de documentos, mantendo um histórico completo dos comandos executados. O padrão separa o objeto que solicita uma operação do objeto que sabe como executá-la, promovendo baixo acoplamento entre a interface do usuário e a lógica de negócio. O CommandInvoker atua como coordenador, gerenciando o histórico de comandos e possibilitando as operações de undo/redo.

Padrão Memento

Nome do Padrão: Memento (Mementó)
Lista de Classes que o Compõem:

- Memento (interface)
- UsuarioMemento (classe concreta)
- DocumentoMemento (classe concreta)
- MementoCaretaker (zelador/cuidador)
- FacadeSingletonController (originadora)

Objetivo de Uso no Projeto:
O padrão Memento é utilizado para capturar e externalizar o estado interno de um objeto sem violar o encapsulamento, de modo que o objeto possa ser restaurado para esse estado mais tarde. No projeto, ele é aplicado para permitir que administradores salvem e restaurem o estado completo do sistema (usuários e documentos) em pontos específicos.
Isso funciona como um sistema de backup/restore que permite reverter o sistema para estados anteriores, seja para recuperação de erros ou para fins de auditoria. O padrão garante que apenas a própria entidade (FacadeSingletonController) tenha acesso aos detalhes internos do estado salvo. Os mementos concretos (UsuarioMemento e DocumentoMemento) armazenam cópias seguras dos dados, enquanto o MementoCaretaker gerencia o histórico de estados salvos.

Padrão Singleton

Nome do Padrão: Singleton (Única Instância)
Lista de Classes que o Compõem:

- FacadeSingletonController (implementação principal)
- FacadeCommand (implementação secundária)

Objetivo de Uso no Projeto:
O padrão Singleton é utilizado para garantir que uma classe tenha apenas uma instância e

fornecer um ponto global de acesso a essa instância. No projeto, ele é aplicado nas fachadas principais do sistema para garantir que haja apenas uma instância gerenciando o estado da aplicação.
Isso é crucial para manter a consistência dos dados, evitar condições de corrida e garantir que todas as partes do sistema trabalhem com a mesma instância dos controladores. O padrão é implementado com sincronização para ambiente multi-thread, assegurando que mesmo em cenários concorrentes apenas uma instância seja criada.

Padrão Factory Method

Nome do Padrão: Factory Method (Método de Fábrica)
Lista de Classes que o Compõem:

- RepositorioFactory (interface fabrica)
- MemoriaRepositorioFactory (implementação concreta)
- ArquivoRepositorioFactory (implementação concreta)

Objetivo de Uso no Projeto:
O padrão Factory Method é utilizado para definir uma interface para criar um objeto, mas permitir que as subclasses decidam qual classe instanciar. No projeto, ele é aplicado na criação de repositórios de persistência, permitindo que o sistema alterne entre persistência em memória e persistência em arquivo sem alterar o código cliente.
Isso proporciona flexibilidade para configurar o tipo de persistência em tempo de execução e facilita a adição de novos mecanismos de persistência no futuro. O sistema pode facilmente mudar entre diferentes estratégias de armazenamento sem impactar o código que utiliza os repositórios.

Padrão Template Method

Nome do Padrão: Template Method (Método Modelo)
Lista de Classes que o Compõem:

- RelatorioTemplate (classe abstrata)
- RelatorioPDF (implementação concreta)
- RelatorioHTML (implementação concreta)

Objetivo de Uso no Projeto:
O padrão Template Method é utilizado para definir o esqueleto de um algoritmo em uma operação, postergando alguns passos para subclasses. No projeto, ele é aplicado na geração de relatórios de acesso de usuários, onde a estrutura geral do relatório (cabeçalho, corpo, rodapé) é definida na classe base.
Enquanto a formatação específica para cada tipo (PDF texto ou HTML) é implementada nas subclasses. Isso elimina duplicação de código e facilita a criação de novos formatos de relatório. O template method garante que a sequência de geração do relatório seja consistente, enquanto permite variações na formatação de cada seção.

Padrão Facade

Nome do Padrão: Facade (Fachada)
Lista de Classes que o Compõem:

- FacadeSingletonController (fachada principal)
- FacadeCommand (fachada especializada)

Objetivo de Uso no Projeto:
O padrão Facade é utilizado para fornecer uma interface unificada para um conjunto de interfaces em um subsistema. No projeto, ele é aplicado para simplificar a complexidade do sistema, fornecendo interfaces simples e de alto nível para as operações do sistema.
A FacadeSingletonController oferece uma interface simplificada para todas as operações administrativas, enquanto a FacadeCommand fornece uma interface especializada para operações com suporte a undo/redo. Isso reduz o acoplamento entre as camadas e facilita o uso do sistema, escondendo a complexidade dos subsistemas subjacentes.

Padrão Adapter (Authenticator como Adaptador)

Nome do Padrão: Adapter (Adaptador) - Implementado implicitamente
Lista de Classes que o Compõem:

- Authenticator (interface alvo)
- AdminAuthenticator (adaptador concreto)
- UserAuthenticator (adaptador concreto)

Objetivo de Uso no Projeto:
O padrão Adapter é utilizado para converter a interface de uma classe em outra interface esperada pelos clientes. No projeto, ele é aplicado implicitamente no sistema de autenticação, onde os diferentes métodos de autenticação (fixo para admin, repositório para usuários) são adaptados para uma interface comum (Authenticator).
Isso permite que sistemas de autenticação heterogêneos trabalhem juntos de forma transparente. O AuthenticationManager pode tratar todos os tipos de autenticação de maneira uniforme, sem precisar conhecer os detalhes específicos de cada implementação.

Padrão Builder

Nome do Padrão: Builder (Construtor)
Lista de Classes que o Compõem:

- DocumentoBuilder (construtor)

Objetivo de Uso no Projeto:
O padrão Builder é utilizado para separar a construção de um objeto complexo de sua representação, de modo que o mesmo processo de construção possa criar diferentes representações. No projeto, ele é aplicado na criação de objetos Documento, permitindo uma construção mais fluente e legível.
Isso é particularmente útil quando a criação do objeto envolve múltiplos parâmetros e quando se deseja garantir que o objeto seja construído em um estado consistente. O DocumentoBuilder oferece métodos with para cada atributo e um método build para finalizar a construção, melhorando a legibilidade e a segurança do código.

Padrão Observer

Nome do Padrão: Observer (Observador)
Lista de Classes que o Compõem:

- Observer (interface)
- LogObserver (observador concreto)
- IndexadorObserver (observador concreto)
- FacadeSingletonController (sujeito)

Objetivo de Uso no Projeto:
O padrão Observer é utilizado para definir uma dependência um-para-muitos entre objetos, de modo que quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente. No projeto, ele é aplicado para permitir que diferentes componentes do sistema (como logging e indexação) sejam notificados sobre eventos importantes.
Isso promove um baixo acoplamento entre o núcleo do sistema e funcionalidades auxiliares, permitindo que novas funcionalidades sejam adicionadas sem modificar o código existente.
O sistema pode notificar múltiplos observadores sobre eventos como criação de usuários e documentos, permitindo reações em cadeia sem acoplamento direto.

📊 Resumo da Aplicação dos Padrões

Padrão	Objetivo Principal	Benefícios no Projeto
Command	Encapsular operações como objetos	Undo/Redo, histórico de comandos
Memento	Capturar e restaurar estados	Backup/Restore do sistema
Singleton	Instância única global	Consistência de estado
Factory Method	Criação flexível de objetos	Troca dinâmica de persistência
Template Method	Esqueleto de algoritmo	Relatórios formatados consistentes
Facade	Interface simplificada	Redução de complexidade
Adapter	Interface comum	Integração de sistemas heterogêneos
Builder	Construção segura de objetos	Código mais legível e seguro
Observer	Notificações de eventos	Sistema reativo e extensível

Arquitetura Geral dos Padrões

Cada padrão foi cuidadosamente selecionado e implementado para resolver problemas específicos de design no sistema:

- Padrões Criacionais (Singleton, Factory Method, Builder): Controlam a criação de objetos e garantem flexibilidade
- Padrões Estruturais (Facade, Adapter): Organizam a estrutura do sistema e simplificam interfaces
- Padrões Comportamentais (Command, Memento, Template Method, Strategy, Observer): Gerenciam algoritmos, responsabilidades e comunicação entre objetos

Esta combinação de padrões resulta em um sistema altamente coeso, com baixo acoplamento, fácil manutenção e grande capacidade de extensão, seguindo os princípios SOLID e as melhores práticas de orientação a objetos.