NOMES : GUSTAVO, MATHEUS E DANIEL

Padrões utilizados no Sistema

Repository - O sistema utiliza o padrão Repository para isolar a lógica de acesso a dados e fornecer uma interface limpa e orientada ao domínio para manipulação de entidades persistentes.

As interfaces definem os contratos de repositório do domínio, enquanto as classes implementam esses contratos usando o Spring Data JPA. Essa separação permite que as camadas de serviço e de caso de uso interajam com os dados de forma abstrata, sem depender diretamente de tecnologias de persistência, promovendo baixo acoplamento, testabilidade e adesão aos princípios da Clean Architecture e do DDD.

Com isso, o sistema trata os repositórios como coleções em memória, ocultando a complexidade do banco de dados e reforçando a coesão da lógica de domínio.

Adapter - O sistema utiliza o padrão Adapter para integrar a camada de domínio com a camada de infraestrutura, permitindo que o domínio dependa apenas de contratos (interfaces) e não de implementações específicas.

Essas classes atuam como adaptadores entre a lógica de negócios e os repositórios JPA, convertendo chamadas genéricas do domínio em operações específicas do Spring Data JPA. Assim, o domínio permanece desacoplado da tecnologia de persistência utilizada, e é possível trocar ou modificar a forma de armazenamento de dados sem impactar a lógica central da aplicação. Esse uso do Adapter contribui diretamente para a aplicação dos princípios da Clean Architecture.

Singleton - O sistema utiliza o padrão Singleton de forma implícita por meio do mecanismo de gerenciamento de beans do Spring Boot, que, por padrão, instancia e mantém uma única instância (singleton) de cada componente anotado com @Service, @Repository, ou @Component durante todo o ciclo de vida da aplicação. Isso significa que classes como, bem como os repositórios e controladores, são instanciadas uma única vez e reutilizadas sempre que necessário.

Esse comportamento evita múltiplas instâncias desnecessárias, economiza recursos e garante consistência no gerenciamento de estado compartilhado (quando aplicável). Ao deixar o Spring controlar essas instâncias, o sistema obtém todos os benefícios do padrão Singleton — como controle centralizado e menor consumo de memória — sem precisar implementar manualmente esse padrão, mantendo o código mais limpo e desacoplado.

Injeção de Dependência - O sistema utiliza o padrão de Injeção de Dependência com o suporte do Spring Boot para promover o desacoplamento entre classes e facilitar a manutenção e testabilidade do código. Isso é feito principalmente através da anotação @Autowired, que permite ao Spring injetar automaticamente as dependências nas classes que precisam delas, como é o caso das classes de serviço e dos controladores.

Essas classes não criam diretamente instâncias de suas dependências — como repositórios ou casos de uso — mas recebem essas dependências prontas, gerenciadas pelo contêiner de IoC (Inversion of Control) do Spring. Com isso, o sistema ganha flexibilidade para trocar implementações, aplicar testes unitários com mocks, e manter uma arquitetura limpa e modular, alinhada aos princípios do SOLID, especialmente o princípio da Inversão de Dependência.

Inversão de Controle - O sistema utiliza o padrão de Inversão de Controle (Inversion of Control – IoC) por meio do contêiner do Spring Boot, que é responsável por criar, gerenciar e fornecer as dependências entre os componentes da aplicação, invertendo o fluxo tradicional de controle.

Em vez das classes instanciar diretamente seus objetos dependentes, elas declaram suas necessidades por meio de anotações como @Autowired e @Component, e o Spring se encarrega de resolver e injetar essas dependências automaticamente no momento da inicialização. Isso pode ser visto.

Ao adotar o IoC, o sistema ganha modularidade, flexibilidade e facilidade para testes, pois as implementações podem ser trocadas sem alterar a lógica de negócio, respeitando princípios de design como DIP (Inversão de Dependência) e promovendo a arquitetura em camadas definida pela Clean Architecture.