2 – conceito de Arquitetura de software e design patterns

2.1 Arquitetura de Software

“O objetivo da arquitetura de software é minimizar os recursos humanos necessários para construir e manter um determinado sistema “ - Clean Architecture: A Craftsman's Guide to Software Structure and Design (2017), escrito por Robert C. Martin.

Arquitetura de software é a definição da estrutura organizacional de um sistema, englobando seus componentes principais, suas responsabilidades e as formas de comunicação entre eles. O conceito foi consolidado na engenharia de software nas décadas de 1990 e 2000, com forte influência de autores como Bass, Clements e Kazman (1998), além de Sommerville (2019), que destacam a importância da arquitetura para garantir atributos de qualidade como escalabilidade, segurança e manutenibilidade.

Alguns exemplos de arquitetura são a em camadas, cliente-servidor, microservices é um exemplo prático, mais usado e conhecido por programadores é o modelo Model-View-Controller (MVC), amplamente adotado em sistemas web, que separa responsabilidades em três camadas distintas: a lógica de domínio (Model), a interface de usuário (View) e o controle de fluxo (Controller), promovendo modularidade e manutenção facilitada.

Além disso, na camada de persistência, sistemas modernos podem adotar bancos de dados relacionais (como Oracle) ou não relacionais (como MongoDB), dependendo da natureza dos dados. Enquanto bancos relacionais garantem consistência em estruturas com forte relacionamento entre entidades, bancos NoSQL são indicados para dados mais flexíveis, como documentos, mídias e registros que não exigem relacionamentos complexos.

2.1.1 A Camada de Persistência na Arquitetura

Um componente crítico em qualquer arquitetura de software é a camada de persistência, responsável por armazenar e recuperar os dados da aplicação. A escolha da tecnologia de banco de dados (SQL como MySQL/Oracle ou NoSQL como o MongoDB) é uma das decisões arquiteturais mais importantes, pois define como a lógica de domínio irá interagir com os dados, a natureza dos dados da aplicação.

No contexto do Sistema de Gerenciamento de Projetos Integradores, a utilização de uma arquitetura híbrida de persistência mostra-se uma alternativa eficiente. Nesse modelo, o Oracle é responsável pelo armazenamento das entidades fortes do sistema e altamente relacionais, como usuários, grupos, tarefas e avaliações, garantindo integridade referencial e consistência transacional dos dados. Em contrapartida, o MongoDB é empregado para dados de natureza mais flexível, como documentos de entregas, apresentações em slides e vídeos, que não demandam relacionamentos rígidos e podem crescer em volume de forma mais acelerada. Essa divisão de responsabilidades entre bancos relacionais e não relacionais permite que o sistema equilibre confiabilidade e flexibilidade, além de otimizar o desempenho na manipulação de diferentes tipos de dados e melhorar a escalabilidade.

2.2 Design Patterns

Design Patterns, ou padrões de projeto, são soluções reutilizáveis para problemas recorrentes no desenvolvimento de software. Eles foram introduzidos formalmente no livro “Design Patterns: Elements of Reusable Object-Oriented Software” (1994), escrito por Erich Gamma,

Richard Helm, Ralph Johnson e John Vlissides, conhecidos como o Gang of Four (GoF). O objetivo principal desses padrões é oferecer soluções testadas para problemas frequentes no design de sistemas orientados a objetos.

Ao contrário dos frameworks, que fornecem uma estrutura completa na qual o código é construído, os Design Patterns não são pedaços de código prontos, mas sim descrições abstratas que orientam como o código pode ser organizado. Enquanto frameworks impõem uma arquitetura pré-definida, os padrões de projeto oferecem maior flexibilidade e podem ser combinados de diversas maneiras em diferentes cenários para melhorar o desenvolvimento do sistema e a sua manutenção futura ajudando na otimização do ciclo de vida de um projeto de software.

2.3 categorias de design patterns

Os Design Patterns podem ser divididos em três categorias principais, onde entram os seus 23 padrões de design como subclasses. Com exemplos onde cada padrão se encaixa nas categorias principais.

2.3.1 Padrões Estruturais

Este grupo de padrões de projeto facilita o projeto de software ao identificar uma maneira simples de realizar relacionamentos entre entidades. Esses padrões são todos sobre composição de Classes e Objetos. Os padrões estruturais de criação de classes usam herança para compor interfaces, enquanto os padrões estruturais de objetos definem maneiras de compor objetos para obter novas funcionalidades. Exemplos de padrões de projeto estrutural incluem adapter, decorator, bridge, composite, flyweight, façade e proxy.

2.3.2 Padrões Criacionais

Esses padrões de projeto se preocupam basicamente com a instanciação de classes e são compostos por duas ideias dominantes. Uma encapsula o conhecimento sobre quais classes concretas o sistema utiliza e a outra oculta como as instâncias dessas classes concretas são criadas e combinadas. Os padrões de projeto criacionais são ainda categorizados em padrões de criação de objetos e padrões de criação de classes, onde os padrões de criação de objetos lidam com a criação de objetos e os padrões de criação de classes lidam com a instanciação de classes. Em maiores detalhes, os padrões de criação de objetos adiam parte da criação de objetos para outro objeto, enquanto os padrões de criação de classes adiam a criação de objetos para as subclasses. Os exemplos de padrões de projeto criacionais são: abstract factory, builder, factory method, singleton e prototype.

2.3.3 Padrões Comportamentais

Esses padrões identificam padrões comuns de comunicação entre objetos e realizam a atribuição de responsabilidades entre eles. Ao fazer isso, esses padrões aumentam a flexibilidade na execução dessa comunicação, pois desviam o foco do fluxo de controle e se concentram apenas na maneira como os objetos estão interconectados. Os exemplos de padrões de projeto comportamentais são: State, Observer, chain of responsibility, iterator, interpreter, template method, memento, command, strategy, visitor e mediator.

2.4 O que é uma aplicação web?

Um aplicativo web é um programa de computador que utiliza navegadores e tecnologias da web para realizar uma variedade de operações pela internet. Um aplicativo web utiliza uma

combinação de 8 scripts do lado do servidor (PHP, Python e ASP) para lidar com o armazenamento e a recuperação das informações, e scripts do lado do cliente (JavaScript e HTML) para apresentar informações aos usuários. Isso permite que os usuários interajam com a empresa usando formulários online, sistemas de gerenciamento de conteúdo, carrinhos de compras e muito mais. Além disso, os aplicativos permitem que os funcionários criem documentos, compartilhem informações, colaborem em projetos e trabalhem em documentos comuns, independentemente da comunicação (Henzel, 2018).

2.4 Arquitetura de aplicações web

Toda ideia recém-nascida precisa encarar a realidade na qual precisa se desenvolver. No caso de aplicações web, o seguinte deve ser levado em consideração:

- O protocolo HTTP, a base tecnológica das aplicações web, não possui estado. Cada comando é executado de forma independente, sem qualquer conhecimento dos comandos que o antecederam. A responsabilidade da aplicação web é manter o estado entre as solicitações.
- A conversa entre o usuário e o aplicativo só pode ser iniciada pelo usuário.
- O fluxo de controle de aplicativos web é orientado por solicitações do usuário. Geralmente consiste em uma sequência complexa de interações entre o usuário e o servidor.