

Introdução ao desenvolvimento em camadas e arquitetura de aplicações Web

1. Introdução

À medida que uma aplicação web cresce em complexidade, torna-se impraticável concentrar toda a lógica de negócio, acesso a dados e apresentação em um único bloco de código. Para lidar com esse crescimento de forma organizada, os desenvolvedores adotam o conceito de arquitetura em camadas (layered architecture), que divide a aplicação em partes bem definidas, cada uma com uma responsabilidade específica. Essa abordagem facilita a manutenção, os testes e a evolução do sistema ao longo do tempo.

2. Front-end e back-end

No contexto do desenvolvimento web, a primeira grande divisão ocorre entre front-end e back-end. O front-end corresponde à parte da aplicação executada no navegador do usuário, responsável pela interface visual e pela interação direta com a pessoa que utiliza o sistema, sendo construído com HTML, CSS e JavaScript. Já o back-end corresponde à parte executada no servidor, responsável por processar regras de negócio, acessar bancos de dados e responder às requisições enviadas pelo front-end, geralmente desenvolvido em linguagens como Node.js, Python, Java, PHP ou C#.

3. As três camadas clássicas

Uma das formas mais tradicionais de organizar uma aplicação é por meio da arquitetura em três camadas:

- **Camada de apresentação (presentation layer)** — responsável pela interface com o usuário, exibindo informações e capturando suas interações. Corresponde, em uma aplicação web, ao front-end.
- **Camada de negócio (business logic layer)** — responsável por processar as regras específicas da aplicação, como cálculos, validações complexas e tomadas de decisão baseadas nos dados recebidos.
- **Camada de dados (data layer)** — responsável por armazenar e recuperar informações, geralmente por meio de um sistema de gerenciamento de banco de dados.

Essa separação permite, por exemplo, que a camada de apresentação seja completamente substituída — passando de uma página web para um aplicativo móvel — sem que seja necessário alterar as regras de negócio ou a forma como os dados são armazenados, desde que a comunicação entre as camadas permaneça consistente.

4. O papel das APIs na comunicação entre camadas

A comunicação entre o front-end e o back-end normalmente ocorre por meio de uma API (Application Programming Interface), que define um conjunto de regras e endpoints (endereços) que o front-end pode utilizar para solicitar ou enviar dados. O padrão arquitetural REST (Representational State Transfer), baseado no protocolo HTTP, é um dos mais utilizados atualmente para a construção dessas APIs, associando métodos HTTP (GET, POST, PUT, DELETE) a operações sobre recursos específicos do sistema:

```
GET /api/produtos      // lista todos os produtos
GET /api/produtos/5    // obtém o produto de ID 5
POST /api/produtos     // cria um novo produto
DELETE /api/produtos/5 // remove o produto de ID 5
```

5. A camada de dados e os bancos de dados

A camada de dados é responsável por armazenar as informações de forma persistente, ou seja, que não se perdem quando a aplicação é reiniciada. Essa camada geralmente utiliza um sistema de gerenciamento de banco de dados, que pode ser de dois tipos principais: bancos de dados relacionais (como MySQL, PostgreSQL e SQL Server), que organizam os dados em tabelas com relações bem definidas entre si, e bancos de dados não relacionais, ou NoSQL (como MongoDB e Firebase), que armazenam os dados em formatos mais flexíveis, como documentos ou pares de chave-valor.

6. Vantagens da arquitetura em camadas

- **Organização e clareza** — cada camada possui uma responsabilidade bem definida, facilitando a compreensão do sistema como um todo.
- **Facilidade de manutenção** — alterações em uma camada tendem a ter impacto reduzido nas demais, desde que as interfaces entre elas sejam respeitadas.
- **Reutilização de código** — a camada de negócio pode ser reutilizada por diferentes interfaces, como uma versão web e um aplicativo móvel do mesmo sistema.
- **Facilidade de testes** — cada camada pode ser testada de forma isolada, sem a necessidade de simular o sistema inteiro.

7. Exemplos de aplicação

A arquitetura em camadas está presente na grande maioria das aplicações web utilizadas atualmente:

- **Aplicativos de e-commerce** — o front-end exibe produtos e processa o carrinho de compras visualmente, enquanto o back-end valida estoques, calcula preços finais e se comunica com o sistema de pagamento; os dados dos produtos e pedidos ficam armazenados em um banco de dados.

- **Redes sociais** — a camada de apresentação exibe publicações e comentários, a camada de negócio processa regras como limites de caracteres e filtros de conteúdo, e a camada de dados armazena perfis, publicações e interações dos usuários.
- **Aplicativos bancários** — a separação em camadas é particularmente importante nesse contexto, já que a camada de negócio precisa validar rigorosamente transações financeiras antes que qualquer alteração seja refletida na camada de dados.
- **Sistemas de gestão empresarial (ERPs)** — diferentes módulos (financeiro, estoque, recursos humanos) compartilham a mesma camada de dados, mas possuem regras de negócio específicas implementadas separadamente para cada área.
- **Plataformas de streaming** — o front-end (aplicativo ou site) solicita à API informações sobre catálogos e progresso de reprodução, enquanto o back-end gerencia licenciamento de conteúdo, recomendações e histórico de visualização armazenado no banco de dados.
- **Aplicações com múltiplas interfaces (multiplataforma)** — empresas que oferecem tanto um site quanto um aplicativo móvel frequentemente reutilizam a mesma API e camada de negócio para ambas as interfaces, evitando duplicar regras de negócio.

8. Considerações finais

A arquitetura em camadas é um princípio fundamental no desenvolvimento de aplicações web modernas, permitindo que sistemas complexos sejam construídos de forma organizada, sustentável e escalável. Ao separar claramente as responsabilidades entre apresentação, lógica de negócio e armazenamento de dados, equipes de desenvolvimento conseguem trabalhar de forma mais colaborativa, testar componentes de forma isolada e adaptar a aplicação a novas necessidades ao longo do tempo, sem comprometer a estrutura geral do sistema.