RESUMO PADRÕES DE PROJETO MOBILE

Matheus Amaral

MVC (Model-View-Controller)

Definição: MVC é um padrão de arquitetura de software que separa uma aplicação em três componentes principais: Model, View e Controller.

Componentes:

- **Model**: Gerencia os dados e a lógica de negócios da aplicação. Responde às solicitações de dados e as manipula (geralmente interage com o banco de dados ou serviços de rede).
- **View**: Representa a interface do usuário. Ela exibe dados para o usuário e envia as interações do usuário ao Controller.
- **Controller**: Atua como intermediário entre o Model e a View. Recebe a entrada do usuário da View, processa-a (possivelmente solicitando ao Model que atualize os dados) e retorna a resposta à View.

Vantagens:

- **Separa as responsabilidades**: Facilita a manutenção e a escalabilidade, uma vez que as responsabilidades estão claramente divididas.
- **Facilita a colaboração**: Designers e desenvolvedores podem trabalhar separadamente nas Views e na lógica de negócios.

Desvantagens:

- **Complexidade em grandes aplicações**: À medida que a aplicação cresce, o Controller pode se tornar muito pesado, com muita lógica nele, o que pode dificultar a manutenção.

Exemplos de Uso:

- **Aplicações Web**: Em um site de e-commerce, o Model gerencia produtos, usuários e pedidos, a View exibe as páginas de produtos, e o Controller gerencia as interações, como adicionar um produto ao carrinho.
- **Desenvolvimento iOS**: O MVC é comumente usado no desenvolvimento de aplicativos iOS, com ViewControllers atuando como o Controller, Views como as interfaces de usuário, e Models lidando com dados.

MVP (Model-View-Presenter)

Definição: MVP é uma variação do padrão MVC, com uma separação mais clara entre a lógica de apresentação e a interface do usuário.

Componentes:

- **Model**: Similar ao MVC, gerencia os dados e a lógica de negócios.
- **View**: Exibe dados ao usuário e delega as interações do usuário ao Presenter. A View é mais passiva em MVP, apenas atualizando a UI conforme instruída pelo Presenter.
- **Presenter**: Atua como intermediário entre o Model e a View. Recebe as interações do usuário da View, processa a lógica de apresentação e atualiza a View. Diferente do Controller em MVC, o Presenter é mais responsável pela lógica de apresentação.

Vantagens:

- **Maior testabilidade**: Como a View é desacoplada do Presenter, este pode ser testado independentemente.
- **Responsabilidades bem definidas**: Clarifica o papel de cada componente, facilitando a manutenção.

Desvantagens:

- **Maior código boilerplate**: Pode haver a necessidade de escrever mais código, especialmente ao criar interfaces para a View se comunicar com o Presenter.

Exemplos de Uso:

- **Android**: MVP foi muito popular no desenvolvimento Android antes do MVVM se tornar a escolha preferida. Exemplo: Um aplicativo de agenda onde o Presenter lida com a lógica de apresentar uma lista de compromissos e a View exibe os compromissos na tela.
- **Aplicações Desktop**: MVP é utilizado em aplicações desktop com UI complexas, como sistemas de ponto de venda, onde a lógica de apresentação e manipulação de dados é crucial.

MVVM (Model-View-ViewModel)

Definição: MVVM é uma evolução dos padrões anteriores, bastante popular em desenvolvimento mobile, especialmente para frameworks que suportam data binding (como Android e WPF).

Componentes:

- **Model**: Similar ao MVC e MVP, gerencia os dados e a lógica de negócios.
- **View**: A interface do usuário que exibe dados ao usuário. A View em MVVM tem ligação direta com o ViewModel através de data binding.
- **ViewModel**: Atua como intermediário entre a View e o Model, contendo a lógica de apresentação. Ele expõe os dados do Model de forma que a View possa consumi-los diretamente por data binding. Também pode manipular eventos da UI.

Vantagens:

- **Data binding**: A View se atualiza automaticamente quando o estado do ViewModel muda, eliminando a necessidade de manualmente atualizar a UI.
- **Maior desacoplamento**: A ViewModel não precisa conhecer a View diretamente, o que permite maior testabilidade e reusabilidade de código.

Desvantagens:

- **Curva de aprendizado**: Para quem não está familiarizado com data binding, pode haver uma curva de aprendizado mais acentuada.
- **Complexidade em casos simples**: Em aplicações simples, o padrão pode ser um excesso, adicionando complexidade desnecessária.

Exemplos de Uso:

- **Aplicações Android**: MVVM é amplamente adotado no desenvolvimento Android moderno usando Android Architecture Components, como LiveData e ViewModel. Exemplo: Um aplicativo de previsão do tempo onde o ViewModel busca dados da API e os atualiza automaticamente na View.
- **Aplicações WPF**: No desenvolvimento de software empresarial em WPF, MVVM é amplamente utilizado para criar interfaces de usuário ricas com a capacidade de ligar a lógica de negócios diretamente aos componentes da interface.

Resultado da comparação Geral:

- **MVC**: Simples, mas o Controller pode ficar sobrecarregado.
- **MVP**: Melhora a testabilidade com um Presenter intermediário, mas adiciona mais código boilerplate.
- **MVVM**: Ótimo para projetos com suporte a data binding, simplificando a UI reativa, mas pode ser complexo para iniciantes.