Padrão MVC

O padrão MVC (Model-View-Controller) é amplamente utilizado para estruturar aplicações, distinguindo a lógica de apresentação da lógica de negócios. A seguir, é apresentado cada uma das principais partes da estrutura:

- **Model**: é o responsável por estabelecer a lógica de negócios e manipular os dados. Sendo uma representação do estado da aplicação e das regras que regem os dados. O Model notifica a View sobre alterações nos dados, permitindo a atualização da interface do usuário.

- **View**: é a interface que exibe os dados do Model. Ela é responsável por apresentar as informações ao usuário e capturar as ações do usuário, tais como cliques e entradas de dados. Nesta parte da aplicação, não deve conter lógica de negócios, mas pode realizar a formatação dos dados para exibição.

- **Controller**: é como um intermediário entre o Model e a View. Recebe as entradas do usuário pela View, analisa essas informações (geralmente interagindo com o Model) e atualiza a View conforme necessário. Além de ser o encarregado por gerenciar o fluxo da aplicação.

Vantagens:

- **Separação de preocupações**: esse padrão estabelece uma clara separação entre a lógica de apresentação, a lógica de negócios e a interface do usuário, o que torna mais fácil a manutenção e a escalabilidade do código.
- **Facilidade de teste**: A separação das responsabilidades torna possível que cada componente seja testado de forma independente, melhorando a testabilidade da aplicação.
- **Reutilização de código**: permite que os componentes possam ser reutilizados em diferentes partes da aplicação ou em diferentes projetos.

Curiosidades:

- **Adaptabilidade:** esse padrão pode ser adaptado para diferentes tipos de aplicações, como web, desktop, dentre outros. Em frameworks web, como o ASP.NET MVC, a estrutura pode ser integrada com outros padrões e técnicas de desenvolvimento.
- **Escalabilidade:** em grandes aplicações, esse padrão pode ser estendido com outros padrões, como o Repository Pattern para o Model, ou técnicas como Dependency Injection para melhorar a modularidade e a testabilidade.

Padrão MVP

O padrão MVP (Model-View-Presenter) é uma variação do padrão MVC, ao qual se concentra na separação de responsabilidades entre a lógica de apresentação e a interface do usuário. Neste padrão, a responsabilidade pela lógica de apresentação é atribuída a um componente chamado Presenter, que atua como intermediário entre o Model e a View. A seguir, é apresentado cada uma das partes da estrutura:

- **View**: é responsável apenas pela apresentação dos dados ao usuário e pela captura das ações do mesmo. Deve ser o mais passiva possível, deixando toda a lógica de apresentação ao Presenter.

- **Presenter**: contém a lógica de apresentação e interage com o Model para obter os dados necessários. Ele também atualiza a View com esses dados e responde às ações do usuário.

- **Model**: representa a lógica de negócios e os dados da aplicação. É totalmente independente da View e do Presenter.

Vantagens:

- Por ser separada, a lógica de apresentação pode ser facilmente testada independentemente da interface do usuário.
- Estabelece uma separação clara de responsabilidades, o que pode resultar em um código mais organizado e de fácil manutenção.

Curiosidades:

- **Interação com o Model:** O Presenter, geralmente, invoca métodos do Model e manipula seus dados. A comunicação entre os dois pode ser assíncrona, sendo útil para operações de I/O ou redes.
- **Variantes do MVP:** Pode haver variações em que o Presenter também é responsável por manipular eventos e comportamentos específicos da View.

Padrão MVVM

O padrão MVVM (Model-View-ViewModel) é uma arquitetura de software que facilita a separação de tarefas em aplicações, especialmente em interfaces ricas e interativas, como as desenvolvidas para plataformas de desktop e dispositivos móveis. É uma extensão dos padrões MVC e MVP, que introduziu o conceito de ViewModel para gerenciar a lógica de apresentação. A seguir, é apresentado cada uma das partes da estrutura:

- **Model**: representa a lógica de negócios e os dados da aplicação. Sendo o responsável por fornecer os dados necessários para a View e o ViewModel.

- **View**: é a interface do usuário que apresenta os dados e possibilita a interação do usuário. No MVVM, é frequentemente implementada usando bindings, que permitem a conexão direta da View ao ViewModel.

- **ViewModel**: atua como um intermediário entre a View e o Model, apresentando dados e comandos que a View pode utilizar por meio de bindings. Contém a lógica de apresentação e é responsável por preparar os dados do Model para a exibição na View, além de responder a interações do usuário.

Vantagens:

- **Data Binding**: permite que a View seja atualizada automaticamente quando os dados no ViewModel mudam, o que reduz a necessidade de código de atualização manual.
- **Testabilidade**: o ViewModel pode ser testado independentemente da View, o que torna a criação de testes automatizados mais fácil.
- **Separação de preocupações**: a separação entre a lógica de apresentação e a interface do usuário, proporciona um código mais organizado e de fácil manutenção.

Curiosidades:

- **Binding Bidirecional:** a maioria dos frameworks que utilizam o MVVM (como WPF e Xamarin) suportam binding bidirecional, o que torna possível que as alterações na View também atualizem o ViewModel.
- **Comandos:** O ViewModel expõe comandos que a View pode executar, o que torna mais fácil o encapsulamento de ações e a lógica associada.