SENAI-SP

MVC

Arquitetura de Software

Model – View - Controller

SENAI
SÃO PAULO

Definição

- O MVC é um padrão de **arquitetura de software**. O MVC sugere uma maneira para você pensar na divisão de responsabilidades, principalmente dentro de um software web.

- O princípio básico do MVC é a divisão da aplicação em três camadas: a camada de interação do usuário (**view**), a camada de manipulação dos dados (**model**) e a camada de controle (**controller**).

- Com o MVC, é possível separar o código relativo à interface do usuário das regras de negócio, o que sem dúvida traz muitas vantagens que veremos mais à frente.

Model

- A responsabilidade dos models é representar o negócio. Também é responsável pelo acesso e manipulação dos dados na sua aplicação.

- O que colocamos em nossos Models?

  - - Atributos
  - - Métodos referentes ao CRUD de nossa base de dados

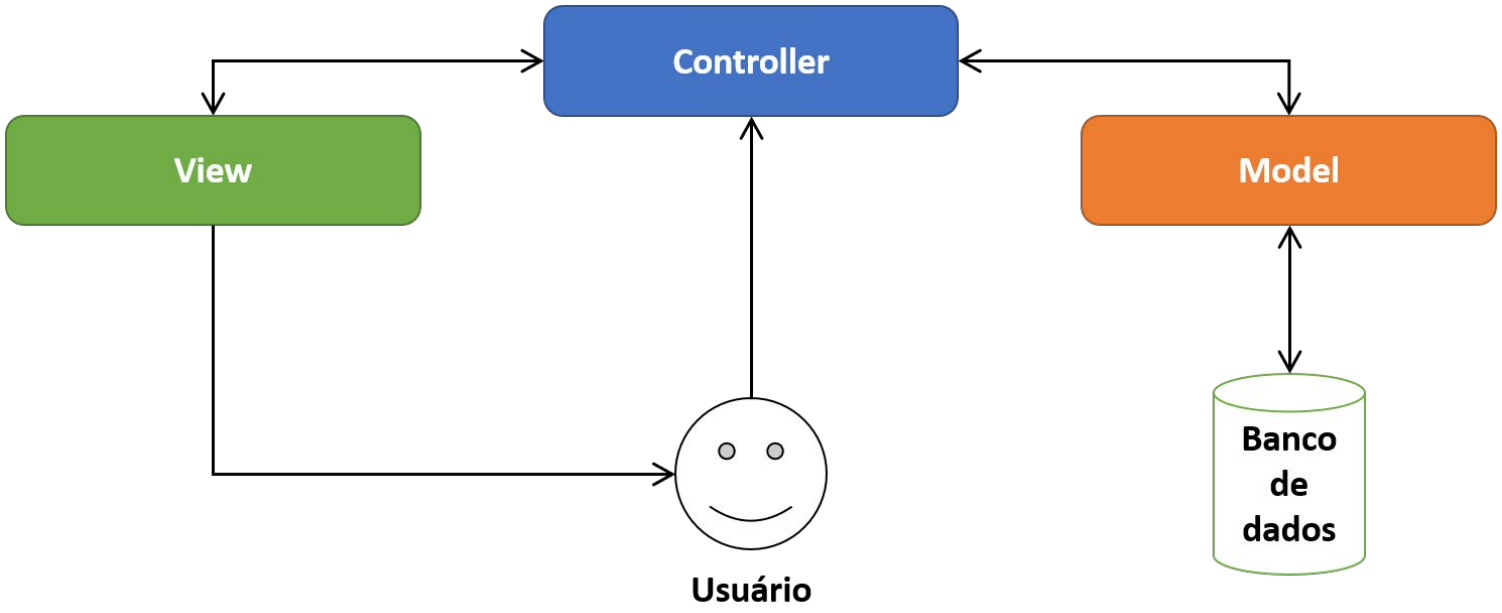
View

- A view é responsável pela interface que será apresentada, mostrando as informações do model para o usuário.

- O que colocamos em nossas Views?

  - - Métodos que retornam da forma que o usuário vê a aplicação. Ex: html+css+javascript ou até o próprio Console.WriteLine() do terminal;

FLUXO



Vantagens

- Não dá para falar do MVC sem citar a importância que ele traz em meio ao desenvolvimento de software.
- Uma dessas vantagens é que ele nos ajuda a deixar o código mais manutenível, ou seja, mais fácil de fazer manutenção, já que temos as responsabilidades devidamente separadas. Isso também traz uma facilidade na compreensão do código, além da sua reutilização.
- Além disso, você tem um código mais testável, pois ele é mais granular: se você tem uma aplicação onde, por exemplo, na página de listagem de usuários, o nome do usuário está sendo cortado ou não está sendo exibido da maneira correta, é muito mais fácil você fazer um teste que atinja somente as estruturas de views.
- Aqui, podemos ver claramente que você tem um problema de apresentação: os models não são responsáveis por aspectos de apresentação, assim como os controllers também não são… Veja que é até mais fácil de identificar que o problema está na view. Por isso, você consegue corrigir somente a view e testá-la de maneira isolada.
- Um segundo exemplo seria se você tivesse um problema de validação ou uma informação de um campo que o usuário está preenchendo na view e não está chegando no banco de dados: não é a view que envia coisas para o banco de dados, assim como também não é o model que é responsável por esse papel (aliás, o model pode até enviar coisas para o banco de dados, mas essas informações são repassadas por outras estruturas anteriores), como por exemplo validar um CPF ou CNPJ.
- Então, podemos chegar à conclusão de que o problema é no controller. Sendo assim, você consegue trabalhar somente no controller, sabendo que as alterações provavelmente não irão impactar nas views e nos models. Além disso, você conseguirá realizar testes de uma maneira muito mais rápida e eficiente.

Desvantagens

- A estrutura pode parecer confusa nos primeiros contatos com seu uso;
- Demanda-se um pouco mais de tempo para se planejar corretamente sua utilização;