POO - PROGRAMAÇÃO ORIENTADA A OBJETOS

Padrão MVC - Model, View, Control

Rodrigo R Silva

Instituto Federal de Educação, Ciência e Tecnologia Sul-Rio-Grandense
Campus Bagé

TADS
Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

Nesta Aula Veremos...

TADS Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

Introdução

- O padrão MVC foi desenvolvido em 1979 por Trygve Reenskaug com a finalidade de ser utilizado como arquitetura para aplicativos desketop. Entretanto, o padrão se popularizou para uso em sistemas web, a partir da adesão de milhares de Frameworks de mercado.

MVC

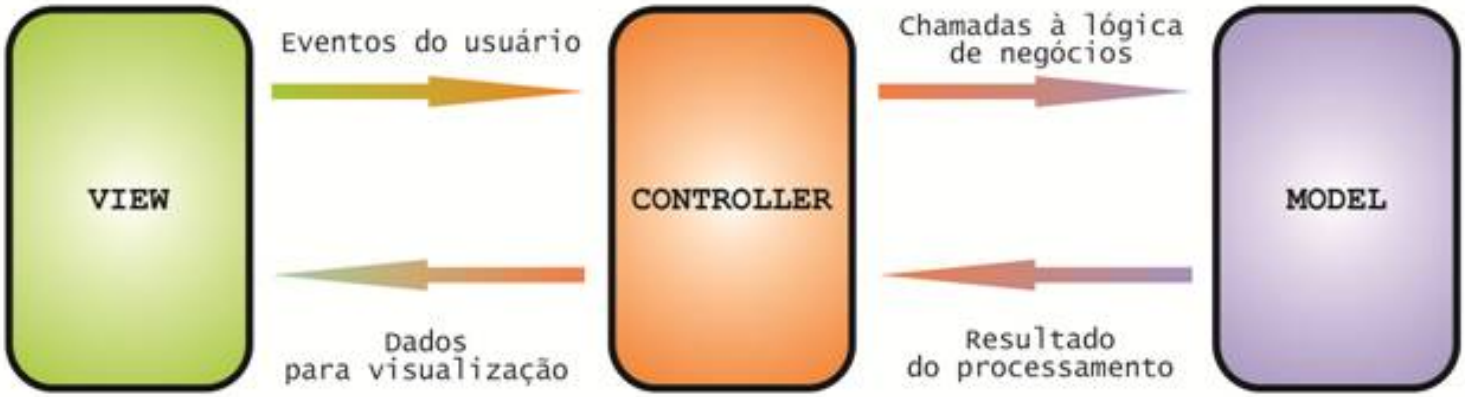
- É um padrão de arquitetura de aplicações que divide a aplicação em três camadas: a visão (view), o modelo (model), e o controlador (controller). Traduzido para o português, a expressão significa: modelo-visão-controlador.
- A utilização do padrão MVC trás como benefício isolar as regras de negócios da lógica de apresentação, a interface com o usuário.

Camadas do MVC

- O model é a camada que possui a lógica da aplicação. Ele é o responsável pelas regras de negócios, persistência com o banco de dados e as classes de entidades. O model recebe as requisições vindas do controller e gera respostas a partir destas requisições.
- Sempre que pensar em model, pense que ele terá conhecimento apenas dos dados armazenados no sistema e da lógica sobre eles. Estes dados podem estar armazenados em um banco de dados ou até mesmo em arquivos do tipo XML, TXT ou de qualquer outro tipo. É no model também que as operações de CRUD devem ser realizadas. Esta camada na verdade é responsável pelo motivo que a aplicação foi construída, ou seja, ela é o núcleo da aplicação.

- A view é a camada de visualização e representa a parte do sistema que interage com o usuário. É pela interface que haverá a entrada dos dados inseridos pelo usuário e também a saída de informações que serão exibidas para ele. Esses dados serão inseridos ou exibidos geralmente por formulários de entrada ou de saída, tabelas, grids, entre outras formas. A view não contém lógica de negócios, portanto todo o processamento é feito pela camada model e então a resposta é repassada para a view pelo controlador.
- Em aplicações de plataforma Desktop, a visão pode ser representada por classes que são construídas com base em componentes do tipo SWING, AWT, SWT, entre outros. Já as aplicações de plataforma Web seriam representadas por páginas do tipo JSP, JSF, entre outros tipos e que são exibidas a partir de um navegador Web.

- Já sabemos que as requisições são enviadas pela view e a lógica de negócios é representada pelo model. Para que haja a comunicação entre essas duas camadas de maneira organizada, é necessário construir a camada controller. Sua função é ser uma camada intermediária entre a camada de apresentação (View) e a camada de negócios (Model).
- Deste modo, toda requisição criada pelo usuário deve passar pelo controller, e este então se comunica com o model. Se o model gerar uma resposta para essas requisições, ele enviará as respostas ao controller que por sua vez repassa à camada view.
- O controlador serve como um intermediário que organiza os eventos da interface com usuário e os direciona para a camada de modelo, assim, torna-se possível um reaproveitamento da camada de modelo em outros sistemas já que não existe dependência entre a visualização e o modelo.


VIEW
Eventos do usuário
Dados
para visualização
CONTROLLER
Chamadas à lógica
de negócios
Resultado
do processamento
MODEL

Vantagens e Desvantagens

- Separação muito clara entre as camadas de visualização e regras de negócios;
- Manutenção do sistema se torna mais fácil;
- Reaproveitamento de código, principalmente da camada de modelo, que pode ser reutilizada em outros projetos;
- As alterações na camada de visualização não afetam as regras de negócios já implementadas na camada de modelo;
- Permite o desenvolvimento, testes e manutenção de forma isolada entre as camadas;
- O projeto passa a ter uma melhor organização em sua arquitetura;
- Torna o entendimento do projeto mais fácil para novos programadores que não participaram de sua criação;

- Em sistemas de baixa complexidade, o MVC pode criar uma complexidade desnecessária;
- Exige muita disciplina dos desenvolvedores em relação à separação das camadas;
- Requer um tempo maior para modelar o sistema.

OBRIGADO!