GEX613 – Programação II

Padrão *Model-View-Controller*

1100/1101 – CIÊNCIA DA COMPUTAÇÃO

Prof. Dr. Giancarlo Salton

Padrão MVC

MVC no Desenvolvimento Web

Problemas com MVC

DRY

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

Padrão MVC

O padrão Model-View-Controller (MVC), é um padrão de arquitetura de software construído com base na separação da apresentação dos dados dos métodos que interagem com os dados.

Um sistema MVC deve permitir que um desenvolvedor de front-end e um de back-end trabalhem no mesmo sistema em paralelo.

Originalmente, o MVC foi projetado para GUIs de desktop, mas foi adaptado e adotado por desenvolvedores web.

O padrão incentiva o desenvolvimento de sistemas modulares, permitindo que os desenvolvedores atualizem, adicionem ou até removam funcionalidades rapidamente.

Exemplo usando um drink - *Request*

Em um bar, você se aproxima do bartender e pede um Manhattan.

Você é o usuário e o seu pedido de bebida é a solicitação do usuário.

Fonte: https://blog.codeanalogies.com/2016/05/02/model-view-controller-mvc-explained-through-ordering-drinks-at-the-bar/

O bartender lhe dá um aceno rápido.

O cérebro do bartender é o controlador.

Ele imediatamente pensa na série de passos necessários para completar seu pedido de bebida:

1. Pegar um copo
2. Adicionar uísque
3. Adicionar vermute
4. Adicionar bitter
5. Mexer a bebida
6. Adicionar uma cereja
7. Pedir e processar o pagamento

Exemplo usando um drink - *Model*

O bartender só pode usar as ferramentas e recursos que estão atrás do bar. Esse conjunto limitado de ferramentas é o modelo e inclui o seguinte, que pode variar de um bar para outro:

As mãos do bartender

Equipamento de mistura/batelada

Licores

Misturas

Copos

Guarnições

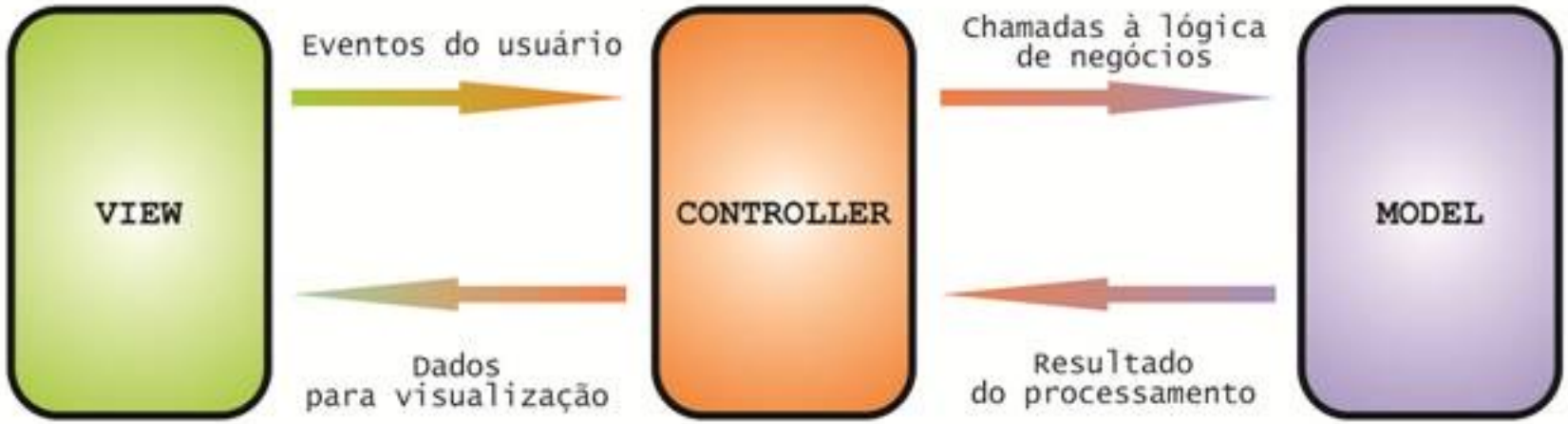
Finalmente, a bebida finalizada é a visão.

Ela é construída a partir das opções limitadas do modelo e organizada e transmitida pelo controlador (ou seja, o cérebro do bartender).

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

MVC no Desenvolvimento Web

MVC na Web


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

No MVC do lado do servidor, o Modelo é o armazenamento permanente dos dados.

Ele deve permitir acesso para que os dados possam ser visualizados, coletados e escritos.

- Tecnicamente, o Modelo é "cego" – ele não tem conhecimento do que acontece com os dados quando são passados para a Visão.

Seu único propósito é processar dados para seu armazenamento permanente ou buscar e preparar dados para serem passados adiante.

- O Modelo não faz perguntas e aceita todas as solicitações.

A Visão é onde os dados, solicitados do Modelo, são visualizados e seu resultado final é determinado.

Tradicionalmente, é a parte do sistema onde o HTML é gerado e exibido.

A Visão também provoca reações do usuário, que então interage com o Controlador.

Por exemplo, um botão gerado pela Visão, que um usuário clica e desencadeia uma ação no Controlador.

Ao aplicar o padrão MVC, muitos desenvolvedores erroneamente veem a Visão como não tendo nenhuma conexão com o Modelo e que todos os dados exibidos pela Visão são passados pelo Controlador.

Para aplicar corretamente a arquitetura MVC, não deve haver relação direta entre a Visão e o Controlador sem o Modelo ou o Usuário entre eles.

Toda a lógica é gerenciada pelos controladores. A parte da Visão nunca recebe dados diretamente do Controlador.

Ao aplicar o padrão MVC, muitos desenvolvedores erroneamente veem a Visão como não tendo nenhuma conexão com o Modelo e que todos os dados exibidos pela Visão são passados pelo Controlador.

Para aplicar corretamente a arquitetura MVC, não deve haver relação direta entre a Visão e o Controlador sem o Modelo ou o Usuário entre eles.

Toda a lógica é gerenciada pelos controladores. A parte da Visão nunca recebe dados diretamente do Controlador.

O Controlador gerencia os dados que o usuário insere ou submete e atualiza o Modelo de acordo.

Ele serve às interações do usuário.

Cada função do Controlador é acionada pela interação do usuário com a Visão.

O Controlador coleta informações, as passa para o Modelo para serem organizadas para armazenamento, e não contém nenhuma lógica além da necessária para coletar a entrada.

O Controlador também está conectado apenas a uma única Visão e a um único Modelo, tornando-o um sistema de fluxo de dados unidirecional, com *"Hand shakes"* e confirmações em cada ponto de troca de dados.

Um erro comum dos desenvolvedores ao aplicar o padrão MVC é atribuir ao Controlador responsabilidades que deveriam ser da Visão, como o processamento e manipulação dos dados do Modelo para a Visão.

1. O usuário faz um pedido para visualizar a página de Preços.
2. O controlador recebe esse pedido e dá um conjunto específico de ordens que são pertinentes para a rota em questão (por exemplo, para a visão atualizar ou servir uma determinada página, ou para o modelo executar uma lógica específica).
3. Se o pedido tem alguma lógica associada a ele, o modelo executa a lógica, extrai dados de um banco de dados e envia de volta uma resposta consistente baseada nas instruções do controlador.
4. O controlador então passa esses dados para a visão atualizar a interface do usuário.
5. Quando os pedidos são recebidos, eles devem ir para o controlador antes de serem convertidos em instruções para a visão ou o modelo.

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

Problemas com MVC

1. Complexidade em aplicações grandes:

   Em sistemas grandes e complexos, o modelo MVC pode se tornar difícil de gerenciar.

   À medida que o número de modelos, visualizações e controladores aumenta, o sistema pode ficar mais difícil de navegar e manter.

2. Acoplamento entre *View* e *Controller*:

   Embora o MVC promova a separação de preocupações, muitas vezes as visualizações e os controladores acabam por estar mais acoplados do que o ideal.

   Isso pode tornar difícil modificar a interface do usuário sem afetar a lógica de controle e vice-versa.

3. Desempenho em aplicações de página única (*Single Page Applications* - SPAs)

   MVC tradicionalmente envia toda a lógica de controle e interações de dados do lado do servidor, o que pode resultar em um desempenho subótimo para SPAs.

   Frameworks modernos de JavaScript como Angular, React e Vue.js, que utilizam conceitos como Model-View-ViewModel (MVVM) ou componentes baseados em UI, são frequentemente mais adequados para essas aplicações.

4. Dificuldade com dados dinâmicos

   O MVC pode não ser a melhor escolha para páginas que requerem uma atualização constante de dados em tempo real.

   O padrão pode ser menos eficiente em cenários onde o modelo de dados muda frequentemente, exigindo atualizações rápidas e dinâmicas da interface do usuário.

5. Código duplicado e reutilização de lógica

   Pode haver uma tendência à duplicação de código nos controladores ou nas visualizações quando funcionalidades similares são necessárias em diferentes partes da aplicação.

   Isso pode dificultar a reutilização de código e aumentar a quantidade de esforço necessário para manutenção.

6. Testabilidade

   Embora o MVC possa melhorar a testabilidade de uma aplicação ao separar a lógica de negócios da interface do usuário, o acoplamento entre as visualizações e os controladores pode tornar os testes de unidade mais desafiadores, especialmente se não forem cuidadosamente projetados desde o início.

7. Testabilidade

   Para novos desenvolvedores, entender como trabalhar efetivamente com o MVC pode ser desafiador.

   Isso inclui não apenas aprender o padrão em si, mas também como ele é implementado em diferentes frameworks e tecnologias..

DRY

Um dos principais argumentos para usar o padrão MVC é reduzir a probabilidade de manter múltiplas instâncias do mesmo código.

A prática de manter seu código otimizado e usar componentes reutilizáveis tanto quanto possível é conhecida como filosofia DRY (Don't Repeat Yourself, ou "Não Se Repita"):

> Cada pedaço de conhecimento deve ter uma única representação autoritária dentro de um sistema.
>
> O objetivo é maximizar a dinâmica e a otimização do sistema.

Se você precisar escrever o mesmo pedaço de código em vários lugares, crie um método separado e use-o onde necessário.

Isso introduz a possibilidade de armazenamento em cache e melhora o tempo de execução geral.

Mudar um elemento do sistema não altera elementos não relacionados, tornando o DRY um princípio importante ao desenvolver com padrões MVC.

Ele promove a reutilização de código e o desenvolvimento paralelo.