**ASP NET MVC**

O padrão arquitetônico MVC (Model-View-Controller) separa um aplicativo em três componentes principais: modelo, exibição e controlador. A estrutura ASP.NET MVC oferece uma alternativa ao padrão Web Forms do ASP.NET para criar aplicativos Web. A estrutura ASP.NET MVC é uma estrutura de apresentação leve e altamente testável que (à semelhança dos aplicativos baseados em Web Forms) é integrada aos recursos ASP.NET existentes, como páginas mestras e autenticação baseada em associação. A estrutura MVC é definida no assembly **System.Web.Mvc**.

O MVC é um padrão de design padrão que muitos desenvolvedores conhecem. Alguns tipos de aplicativos Web irão se beneficiar da estrutura MVC.Outros vão continuar usando o padrão de aplicativo ASP.NET tradicional que é baseado em Web Forms e postbacks. Outros tipos de aplicativos Web irão combinar as duas abordagens; uma abordagem não exclui a outra.

O padrão MVC ajuda a criar aplicativos que separam os diferentes aspectos do aplicativo (lógica de entrada, lógica de negócio e lógica da IU), enquanto fornece um acoplamento flexível entre esses elementos. O padrão especifica onde cada tipo de lógica deve ficar localizado no aplicativo. A lógica da IU fica na exibição. A lógica de entrada fica no controlador. A lógica de negócios fica no modelo. Essa separação ajuda a administrar a complexidade quando você cria um aplicativo, porque ela permite que você se concentre em um aspecto da implementação por vez. Por exemplo, você pode se concentrar na exibição sem depender da lógica de negócios.

O acoplamento flexível entre os três componentes principais de um aplicativo MVC também promove o desenvolvimento paralelo. Por exemplo, um desenvolvedor pode trabalhar na exibição, um segundo desenvolvedor pode trabalhar na lógica do controlador e um terceiro desenvolvedor pode se concentrar na lógica de negócios do modelo.

**Por que utilizar MVC?**

Com o aumento da complexidade dos sistemas/sites desenvolvidos hoje, essa arquitetura tem como foco dividir um grande problema em vários problemas menores e de menor complexidade. Dessa forma, qualquer tipo de alterações em uma das camadas não interfere nas demais, facilitando a atualização de layouts, alteração nas regras de negócio e adição de novos recursos. Em caso de grandes projetos, o MVC facilita muito a divisão de tarefas entre a equipe.  
  
Abaixo serão listadas algumas das vantagens em utilizar MVC em seus projetos:

* Facilita o reaproveitamento de código;
* Facilidade na manutenção e adição de recursos;
* Maior integração da equipe e/ou divisão de tarefas;
* Diversas tecnologias estão adotando essa arquitetura;
* Facilidade em manter o seu código sempre limpo;

**Definição das camadas**

A estrutura MVC inclui os seguintes componentes:

* **Modelos(Models)**. Os objetos de modelo são as partes do aplicativo que implementam a lógica para o domínio de dados do aplicativo. Muitas vezes, os objetos de modelo recuperam e armazenam o estado do modelo em um banco de dados. Por exemplo, um objeto **Product** pode recuperar informações de um banco de dados, operar nele e, em seguida, gravar informações atualizadas de volta em uma tabela de Produtos em um banco de dados do SQL Server.

Em aplicativos pequenos, o modelo, muitas vezes, é uma separação conceitual em vez de física. Por exemplo, se o aplicativo apenas ler um conjunto de dados e enviá-lo para exibição, o aplicativo não terá uma camada de modelo físico nem classes associadas. Nesse caso, o conjunto de dados assume a função de um objeto de modelo.

O modelo (Model) é utilizado para manipular informações de forma mais detalhada, sendo recomendado que, sempre que possível, se utilize dos modelos para realizar consultas, cálculos e todas as regras de negócio do nosso site ou sistema.

* **Exibições(Views)**. As exibições são os componentes que exibem a interface do usuário (IU) do aplicativo, ou seja, as telas. Normalmente, esta IU é criada a partir dos dados do modelo. Um exemplo seria uma exibição de edição de uma tabela de Produtos que mostra caixas de texto, listas suspensas e caixas de seleção com base no estado atual de um objeto **Product**. A visão (View) é responsável por tudo que o usuário final visualiza, toda a interface, informação, não importando sua fonte de origem.
* **Controladores(Controllers)**. Os controladores são os componentes que lidam com a interação do usuário, trabalham com o modelo e, finalmente, selecionam uma exibição de renderização que mostra essa IU(View). Em um aplicativo MVC, a exibição só mostra informações; o controlador manipula e responde à entrada e à interação do usuário. Por exemplo, o controlador manipula valores da cadeia de consulta e passa esses valores ao modelo, que por sua vez pode usar estes valores para consultar o banco de dados.

**Exemplo do funcionamento do MVC**

Com a teoria já compreendida, vamos imaginar a seguinte situação: Você desenvolveu um site, e esse site possui uma tela de login, onde o usuário digita seu login e sua senha, após a autenticação, caso ocorra tudo certo, o usuário acessa a área restrita do site, caso contrário é redirecionado novamente para a página de login repassando uma mensagem que a combinação de usuário e senha é inválida.  
  
Conseguiu imaginar essa situação? Beleza... Agora veja como isso acontece caso você ainda não tenha adotado a arquitetura MVC em seu site: Primeiramente, o usuário preenche o formulário com seu login e sua senha e pressiona o botão “Logar”.   
  
Depois disso, o formulário envia essas informações para um arquivo onde, no mesmo arquivo, você executa as seguintes etapas:  
  
1. Armazena em variáveis os dados digitados pelo usuário;  
2. Montam um comando SQL para selecionar o usuário;  
3. Verifica se retornou alguma informação;

* Se retornar alguma informação, armazena o usuário em uma sessão e redireciona para a área restrita;
* Se não retornar nenhuma informação, redireciona para a página de login com uma mensagem notificando que a combinação digitada é inválida;

Aparentemente está tudo ok, tudo funcionado. Mas veremos agora como funcionaria se o seu site estivesse utilizando a arquitetura MVC: Os passos seguem os mesmos, primeiramente, o usuário preenche o formulário com seu login e sua senha e pressiona o botão “Logar”. Agora veremos algumas mudanças.   
  
**Depois disso, o formulário envia essas informações para uma controladora, e essa controladora realizará as seguintes etapas:**  
  
1. A controladora (controller) carrega um modelo (model), e executa um método que realiza a validação;

2. No modelo (model) são executadas as seguintes tarefas:

* Armazena as informações digitadas pelo usuário;
* Realiza a consulta no banco retornando verdadeiro (true) em caso de sucesso(usuário existe e a senha bate), ou falso (false) no caso de a combinação das informações digitadas serem inválidas;

3. A controladora (controller) verifica o que o modelo retornou;

* Se retornar verdadeiro (true) armazena as informações em uma sessão e redireciona o usuário para visão (view) da área restrita;
* Se retornar falso (false) redireciona o usuário de volta para a tela (view) de login repassando a mensagem que a combinação digitada é inválida;

Agora você pode ficar se perguntando, mas do modo que eu faço também funciona? Pode ser que sim, mas imagine ter que alterar a regra de negócio. Antes de utilizar MVC você precisaria abrir o arquivo que realiza todas as tarefas e localizar a sua regra, para depois alterar. No caso do MVC, você já sabe onde se encontra as suas regras de negócio, então você vai direto ao arquivo. Por isso que a maioria dos frameworks já vem com sua estrutura de diretórios pronta, facilitando a localização dos arquivos.

**ViewData, ViewBag e TempData**

* Similaridades entre ViewData e ViewBag

ViewData e ViewBag são similares nas seguintes características:

* São utilizadas para persistir dados entre a Controller e a View correspondente.
* A duração “tempo de vida” é apenas entre o envio através da Controller e a exibição na View, depois disso tornam-se nulas novamente.
* No caso de um redirect se tornam nulas.

## Diferenças entre ViewData e ViewBag

|  |  |
| --- | --- |
| **ViewData** | **ViewBag** |
| É um dicionário de objetos derivado de ViewDataDictionary e é acessível utilizando strings como chaves. | É uma propriedade dinâmica baseada na funcionalidade “dynamic” do C# 4.0 |
| Requer typecasting (conversão) quando associada a tipos complexos. | Não necessita de conversão para tipos complexos. |

**Exemplos de aplicação**

Controller

Uma imagem contendo texto

Descrição gerada com alta confiança

View

Uma imagem contendo interior

Descrição gerada com alta confiança

Resumindo, ViewData e ViewBag possuem a mesma proposta, porém o ViewBag está disponível a partir do ASP.Net MVC 3, enquanto o ViewData existe desde a primeira versão.

O ViewData é um wrapper, uma implementação do ViewBag, pois utiliza o ViewBag internamente, portanto:

Por este motivo ViewData é mais rápido que o ViewBag, porém essa diferença de velocidade é mínima, não é necessário deixar de usar o ViewBag por este motivo.

**TempData**

* TempData assemelha-se mais a uma sessão de servidor, porém de curta duração.
* Possui um tempo de vida maior que o ViewBag e ViewData, o TempData perdura desde sua criação até que seja chamado, ou seja, quando houver um request da informação do TempData, ele se tornará nulo novamente.
* Uma informação em TempData criada em um Controller persiste após um redirect entre actions (apenas um) e pode ser exibido em sequência em uma View (muito usado em tratamento de erros).
* Caso não seja chamado o TempData pode manter o estado de seus dados até que a sessão do usuário se encerre.
* É utilizado para compartilhar informações entre Controllers.
* O TempData salva suas informações no SessionState do servidor.
* Após a leitura os dados do TempData são marcados para deleção, ou seja, no final do request todos os dados marcados serão deletados.
* É um benefício quando necessário transmitir um volume de informações entre as Controllers sem se preocupar em zerar os valores, pois o TempData automaticamente faz isso.

**Exemplo de aplicação**

* **Controller**

**Uma imagem contendo captura de tela, texto

Descrição gerada com alta confiança**

Neste exemplo pudermos entender que o propósito do TempData é compartilhar dados entre Controllers, portanto sua duração persiste até que a informação seja lida.  
Outro detalhe é sempre checar se o TempData não está nulo.

Caso você queira manter o dado de um TempData mesmo após a leitura, basta chamar o método Keep(), assim o dado será persistido novamente até a próxima requisição.

* **View**

Uma imagem contendo parede, interior, monitor, tela

Descrição gerada com alta confiança

Recomenda-se utilizar sempre ViewBag e ViewData para transferência de dados entre Controller e View. O TempData em Views é recomendado no caso de um dado necessitar ser redirecionado entre Actions e posteriormente ser exibido numa View (ViewBag e ViewData são anulados em redirects).