Padrão MVC

Padrão MVC

- O padrão de arquitetura MVC (Model-View-Controller) separa um aplicativo em três grupos de componentes principais: Modelos, Exibições e Componentes.

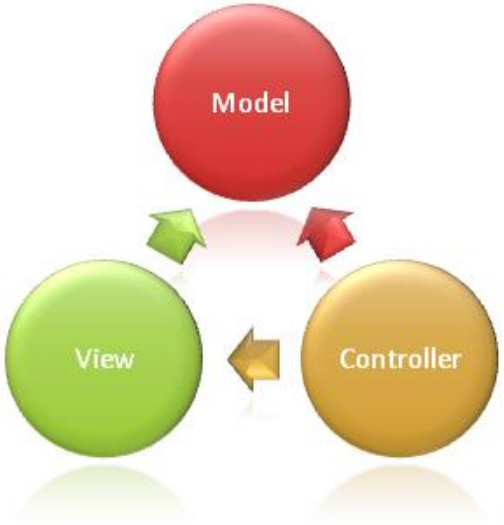
- Esse padrão ajuda a obter a separação de interesses.

- Usando esse padrão, as solicitações de usuário são encaminhadas para um Controlador, que é responsável por trabalhar com o Modelo para executar as ações do usuário e/ou recuperar os resultados de consultas.

Padrão MVC

- O Controlador escolhe a Exibição a ser exibida para o usuário e fornece-a com os dados do Modelo solicitados.

- O seguinte diagrama mostra os três componentes principais e quais deles referenciam os outros:



Padrão MVC

- Essa descrição das responsabilidades ajuda você a dimensionar o aplicativo em termos de complexidade, porque é mais fácil de codificar, depurar e testar algo (modelo, exibição ou controlador) que tem um único trabalho.

- É mais difícil atualizar, testar e depurar um código que tem dependências distribuídas em duas ou mais dessas três áreas.

Padrão MVC

- Por exemplo, a lógica da interface do usuário tende a ser alterada com mais frequência do que a lógica de negócios.
  - Se o código de apresentação e a lógica de negócios forem combinados em um único objeto, um objeto que contém a lógica de negócios precisa ser modificado sempre que a interface do usuário é alterada.
  - Isso costuma introduzir erros e exige um novo teste da lógica de negócios após cada alteração mínima da interface do usuário.

Responsabilidades do Modelo (Model)

- O Modelo em um aplicativo MVC representa o estado do aplicativo e qualquer lógica de negócios ou operação que deve ser executada por ele.
- A lógica de negócios deve ser encapsulada no modelo, juntamente com qualquer lógica de implementação, para persistir o estado do aplicativo.
- As exibições fortemente tipadas normalmente usam tipos ViewModel criados para conter os dados a serem exibidos nessa exibição.
- O controlador cria e popula essas instâncias de ViewModel com base no modelo.

Responsabilidades da Exibição (View)

- As exibições são responsáveis por apresentar o conteúdo por meio da interface do usuário.
- Eles usam o **Razor**[1] de exibição para inserir o código .NET na marcação HTML.
- Deve haver uma lógica mínima nas exibições e qualquer lógica contida nelas deve se relacionar à apresentação do conteúdo.
- Se você precisar executar uma grande quantidade de lógica em arquivos de exibição para exibir dados de um modelo complexo, considere o uso de um Componente de Exibição, ViewModel ou um modelo de exibição para simplificar a exibição.

Razor é uma linguagem de marcação de modelo compacta, expressiva e fluida para definir exibições usando código C# inserido.
Razor é usado para gerar dinamicamente o conteúdo da Web no servidor. Você pode combinar o código do servidor com o código e o conteúdo do lado cliente de maneira limpa.

Responsabilidades do Controlador (Controller)

- Os controladores são os componentes que cuidam da interação do usuário, trabalham com o modelo e, em última análise, selecionam uma exibição a ser renderizada.

- Em um aplicativo MVC, a exibição mostra apenas informações; o controlador manipula e responde à entrada e à interação do usuário.

- No padrão MVC, o controlador é o ponto de entrada inicial e é responsável por selecionar quais tipos de modelo serão usados para o trabalho e qual exibição será renderizada (daí seu nome – ele controla como o aplicativo responde a determinada solicitação).

ASP.NET Core MVC

- A estrutura do ASP.NET Core MVC é uma estrutura de apresentação leve, de software livre e altamente testável, otimizada para uso com o ASP.NET Core.
- ASP.NET Core MVC fornece uma maneira com base em padrões para criar sites dinâmicos que habilitam uma separação limpa de preocupações.
- Ele lhe dá controle total sobre a marcação, dá suporte ao desenvolvimento amigável a **TDD**[1] e usa os padrões da web mais recentes.

[1]Test Driven Development, que em português significa Desenvolvimento Orientado por Testes. Esse é um método de desenvolvimento muito comum atualmente. Ele se baseia na aplicação de pequenos ciclos de repetições. Em cada um deles, um teste é aplicado.

Roteamento

- O ASP.NET Core MVC baseia-se no roteamento do ASP.NET Core, um componente de mapeamento de URL avançado que permite criar aplicativos que têm URLs compreensíveis e pesquisáveis.

- Isso permite que você defina padrões de nomenclatura de URL do aplicativo que funcionam bem para SEO (otimização do mecanismo de pesquisa) e para a geração de links, sem levar em consideração como os arquivos no servidor Web estão organizados.

Roteamento

- O **roteamento baseado em convenção** permite que você defina globalmente os formatos de URL que seu aplicativo aceita e como cada um desses formatos é mapeado para um método de ação específico em um determinado controlador.

- Quando uma solicitação de entrada é recebida, o mecanismo de roteamento analisa a URL e corresponde-a a um dos formatos de URL definidos. Em seguida, ele chama o método de ação do controlador associado.

```
routes.MapRoute(name: "Default", template: "{controller=Home}/{action=Index}/{id?}");
```

Roteamento

- O **roteamento de atributo** permite que você especifique as informações de roteamento decorando os controladores e as ações com atributos que definem as rotas do aplicativo.

- Isso significa que as definições de rota são colocadas ao lado do controlador e da ação aos quais estão associadas.

```
[Route("api/[controller]")]
public class ProductsController : Controller
{
    [HttpGet("{id}")]
    public IActionResult GetProduct(int id)
    {
      ...
    }
}
```

Model binding

- ASP.NET Core MVC model binding converte dados de solicitação de cliente (valores de formulário, os dados de rota, parâmetros de cadeia de caracteres de consulta, os cabeçalhos HTTP) em objetos que o controlador pode manipular.

- Como resultado, a lógica de controlador não precisa fazer o trabalho de descobrir os dados de solicitação de entrada; ele simplesmente tem os dados como parâmetros para os métodos de ação.

```
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null) { ... }
```

Validação de modelo

- O ASP.NET Core MVC dá suporte à validação pela decoração do objeto de modelo com atributos de validação de anotação de dados.

- Os atributos de validação são verificados no lado do cliente antes que os valores sejam postados no servidor, bem como no servidor antes que a ação do controlador seja chamada.

```
using System.ComponentModel.DataAnnotations;
public class LoginViewModel
{
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    [Display(Name = "Remember me?")]
    public bool RememberMe { get; set; }
}
```

Validação de modelo

- Uma ação do controlador:

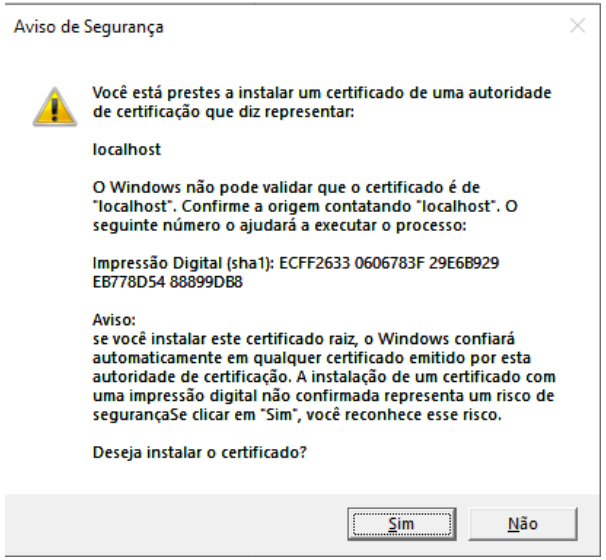
```
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)
{
    if (ModelState.IsValid)
    {
      // work with the model
    }
    // At this point, something failed, redisplay form
    return View(model);
}
```

Criando um primeiro aplicativo

- Inicie o Visual Studio e selecione Criar um projeto.
- Na caixa de diálogo criar um novo projeto , selecione ASP.NET Core aplicativo Web (Model-View-Controller) > avançar.
- Na caixa de diálogo configurar seu novo projeto , digite MvcMovie para Project nome. É importante nomear o projeto MvcMovie. A capitalização precisa corresponder a cada namespace quando o código for copiado.
- Selecione Avançar.
- Na caixa de diálogo informações adicionais , selecione .net 6,0 (suporte a longo prazo).
- Selecione Criar.

Criando um primeiro aplicativo

- Compile (CTRL + F5 – Iniciar sem depuração) e o Visual Studio exibirá a caixa de diálogo a seguir quando um projeto ainda não estiver configurado para usar SSL.
  - Clique em sim e após solicitará a instalação do certificado raiz e clique em sim.



Criando um primeiro aplicativo

- Visual Studio executa o aplicativo e abre o navegador padrão.
- A barra de endereços mostra localhost:port# e não algo como example.com. O nome do host padrão para seu computador local é localhost.
  - Uma porta aleatória é usada para o servidor Web quando o Visual Studio cria um projeto Web.
- Iniciar o aplicativo sem depuração selecionando CTRL + F5 permite que você:
- Realize alterações de código.
- Salve o arquivo.
- Atualize rapidamente o navegador e veja as alterações no código.

Criando um primeiro aplicativo

- Os aplicativos baseados no MVC contêm:
  - Models: classes que representam os dados do aplicativo. As classes de modelo usam a lógica de validação para impor regras de negócio aos dados. Normalmente, os objetos de modelo recuperam e armazenam o estado do modelo em um banco de dados
  - Views: exibições são os componentes que exibem a interface do usuário do aplicativo. Em geral, essa interface do usuário exibe os dados de modelo.
  - Controllers: classes que:
    - Manipular solicitações do navegador.
    - Recuperar dados do modelo.
    - Modelos de exibição de chamada que retornam uma resposta.

Criando um primeiro aplicativo

- Em um aplicativo MVC, a exibição exibe apenas informações.
- O controlador trata e responde à entrada e à interação do usuário.
- Por exemplo, o controlador lida com segmentos de URL e valores de cadeia de caracteres de consulta e passa esses valores para o modelo.

Criando um primeiro aplicativo

- O modelo pode usar esses valores para consultar o banco de dados. Por exemplo:
  - https://localhost:5001/Home/Privacy: especifica o controlador Home e a Privacy.
  - https://localhost:5001/Movies/Edit/5: é uma solicitação para editar o filme com ID=5 usando o controlador e a ação.

Adicionando um Controller:

- No Gerenciador de Soluções, clique com o botão direito do mouse em Controladores > Adicionar > Controlador.
- Na caixa de diálogo Adicionar Novo Item Scaffolded, selecione Controlador MVC – Adicionar > Vazio.
- Na caixa de diálogo Adicionar Novo Item – MvcMovie, insira HelloWorldController.cs e selecione Adicionar.

Adicionando um Controller

- Adicionando um Controller:

```
public class HelloWorldController : Controller
{
    //
    // GET: /HelloWorld/

    public string Index()
    {
        return "Minha ação padrão";
    }

    //
    // GET: /HelloWorld/Welcome/

    public string Welcome()
    {
        return "Bem vindo ao meu método de boas vindas do meu controller";
    }
}
```

Adicionando um Controller

- Cada método public em um controlador pode ser chamado como um ponto de extremidade HTTP. Na amostra acima, ambos os métodos retornam uma cadeia de caracteres. Observe os comentários que precedem cada método.

- Um ponto de extremidade HTTP:

  - É uma URL que pode ser direcionada no aplicativo Web, como https://localhost:5001/HelloWorld .

  - Combina:

    - O protocolo usado: HTTPS .

    - O local de rede do servidor Web, incluindo a porta TCP: localhost:5001 .

    - O URI de destino: HelloWorld .

Adicionando um Controller

- O primeiro comentário indica que este é um método HTTP GET invocado por meio do acréscimo de /HelloWorld/ à URL base.

- O primeiro comentário especifica um método HTTP GET invocado por meio do acréscimo de /HelloWorld/Welcome/ à URL base. Posteriormente no tutorial, o mecanismo de scaffolding é usado para gerar HTTP POST métodos, que atualizam dados.

- Execute o aplicativo sem o depurador.

- Anexar "HelloWorld" ao caminho na barra de endereços. O método Index retorna uma cadeia de caracteres.

Adicionando um Controller

- O MVC invoca classes de controlador e os métodos de ação dentro delas, dependendo da URL de entrada.
- A lógica de roteamento de URL padrão usada pelo MVC usa um formato como este para determinar qual código invocar:
  - /[Controller]/[ActionName]/[Parameters]

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

Adicionando um Controller

- Quando você navega até o aplicativo e não fornece nenhum segmento de URL, ele assume como padrão o controlador e o método "Index" especificados na linha de modelo realçada Home acima.

- Nos segmentos de URL anteriores:

  - O primeiro segmento de URL determina a classe do controlador a ser executada. Portanto, localhost:5001/HelloWorld mapeia para a classe HelloWorld Controller.

  - A segunda parte do segmento de URL determina o método de ação na classe. Portanto, localhost:5001/HelloWorld/Index faz com que o método da classe seja Index HelloWorldController executado. Observe que você precisou apenas navegar para localhost:5001/HelloWorld e o método Index foi chamado por padrão. Index é o método padrão que será chamado em um controlador se um nome de método não for especificado explicitamente.

  - A terceira parte do segmento de URL (id) refere-se aos dados de rota. Os dados de rota são explicados posteriormente no tutorial.

Adicionando um Controller

- Navegue até: https://localhost:{PORT}/HelloWorld/Welcome . Substitua {PORT} pelo número da porta.
- O método Welcome é executado e retorna a cadeia de caracteres “Bem vindo ao meu método de boas vindas do meu controller”
- Para essa URL, o controlador é HelloWorld e Welcome é o método de ação. Você ainda não usou a parte [Parameters] da URL.

Adicionando um Controller

- Modifique o código para passar algumas informações de parâmetro da URL para o controlador. Por exemplo:
  - https://localhost:{PORT}/HelloWorld/Welcome?nome=Ricardo&Vezes=4

```
public string Welcome(string nome, int vezes = 1)
{
    return HtmlEncoder.Default.Encode($"Oi {nome}, vezes: {vezes}");
}
```

  - HtmlEncoder.Default.Encode para proteger o aplicativo contra entradas mal-intencionadas, como por meio de JavaScript.

Adicionando um Controller

- Execute o aplicativo e insira a seguinte URL:
- https://localhost:{PORT}/HelloWorld/Welcome/3?nome=Ricardo

```
public string Welcome(string nome, int ID = 1)
{
    return HtmlEncoder.Default.Encode($"Oi {nome}, ID: {ID}");
}
```

- O terceiro segmento de URL corresponderam ao parâmetro de rota id .
- O método Welcome contém um parâmetro id que correspondeu ao modelo de URL no método MapControllerRoute.

Adicionando uma View

- Modificar HelloWorldController classe para usar os Razor arquivos de exibição. Isso encapsula corretamente o processo de geração de respostas HTML para um cliente.
- Os modelos de exibição são criados usando o Razor . Razormodelos de exibição com base em:
- Ter uma extensão de arquivo . cshtml .
- Forneça uma maneira elegante de criar saída HTML com C#.
- Atualmente, o Index método retorna uma cadeia de caracteres com uma mensagem na classe Controller. Na classe HelloWorldController, substitua o método Index pelo seguinte código:

Adicionando uma View

- Os modelos de exibição são criados usando o Razor . Razormodelos de exibição com base em:
  - Ter uma extensão de arquivo . cshtml .
  - Forneça uma maneira elegante de criar saída HTML com C#.

Adicionando uma View

- Atualmente, o Index método retorna uma cadeia de caracteres com uma mensagem na classe Controller. Na classe HelloWorldController, substitua o método Index pelo seguinte código:

```
public IActionResult Index()
{
    return View();
}
```

Adicionando uma View

- O código anterior:
  - Chama o método do controlador View .
  - Usa um modelo de exibição para gerar uma resposta HTML.

Adicionando uma View

- Clique com o botão direito do mouse na pasta views e adicione > nova pasta e nomeie a pasta HelloWorld.
- Clique com o botão direito do mouse na pasta views/HelloWorld e adicione > novo item.
- Na caixa de diálogo Adicionar novo item
  - Na caixa de pesquisa no canto superior direito, insira exibição
  - Selecione Razor Exibir-vazio
  - Mantenha o valor da caixa Nome, Index.cshtml.
  - Selecione Adicionar

Adicionando uma View

- Substitua o conteúdo do arquivo de exibição exibições/HelloWorld/index.cshtml Razor pelo seguinte:

```
@{
    ViewData["Title"] = "Index";
}

<h2>Index</h2>

<p>Hello from our View Template!</p>
```

Adicionando uma View

- Acesse o diretório https://localhost:{PORT}/HelloWorld:

  - O Index método na HelloWorldController instrução executou return View(); , que especificou que o método deve usar um arquivo de modelo de exibição para renderizar uma resposta para o navegador.

  - Um nome de arquivo de modelo de exibição não foi especificado; portanto, o MVC assume o uso do arquivo de exibição padrão. Quando o nome do arquivo de exibição não é especificado, o modo de exibição padrão é retornado. O modo de exibição padrão tem o mesmo nome que o método de ação, Index neste exemplo. O modelo de exibição /views/HelloWorld/index.cshtml é usado.

  - A imagem a seguir mostra a cadeia de caracteres "Olá de nosso modelo de exibição!" embutido em código na exibição:

Adicionando uma View

- Alterar exibições e páginas de layout:
  - Selecione o menu links MvcMovie, Home e Privacy . Cada página mostra o mesmo layout de menu. O layout de menu é implementado no arquivo Views/Shared/_Layout.cshtml.
  - Abra o arquivo Views/Shared/_Layout.cshtml.
  - Os modelos de layout permitem:
    - Especificando o layout de contêiner HTML de um site em um único lugar.
    - Aplicar o layout de contêiner HTML em várias páginas no site.
    - Localize a linha @RenderBody(). RenderBody é um espaço reservado em que todas as páginas específicas à exibição criadas são mostradas, encapsuladas na página de layout. Por exemplo, se você selecionar o Privacy link, a exibição views / Home / Privacy . cshtml será renderizada dentro do RenderBody método.

Adicionando uma View

- Alterar o título, o rodapé e o link de menu no arquivo de layout
  - Substitua o conteúdo do arquivo views/Shared/_Layout. cshtml pela marcação a seguir. As alterações são realçadas:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Movie App</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shad
            <div class="container-fluid">
                <a class="navbar-brand" asp-area="" asp-controller="Movies" asp-action="Index">Movie App</a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navba
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Priva
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div class="container">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2021 - Movie App - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
        </div>
    </footer>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

Adicionando uma View

- ## Passando dados do controlador para a exibição

- As ações do controlador são invocadas em resposta a uma solicitação de URL de entrada. Uma classe de controlador é o local em que o código é escrito e que manipula as solicitações recebidas do navegador. O controlador recupera dados de uma fonte de dados e decide qual tipo de resposta será enviada novamente para o navegador. Modelos de exibição podem ser usados em um controlador para gerar e formatar uma resposta HTML para o navegador.

- Os controladores são responsáveis por fornecer os dados necessários para que um modelo de exibição renderize uma resposta.

- Os modelos de exibição não devem:

  - Fazer lógica de negócios

  - Interagir diretamente com um banco de dados.

Adicionando uma View

- Um modelo de exibição deve funcionar apenas com os dados fornecidos pelo controlador. Manter essa "separação de preocupações" ajuda a manter o código:
  - Limpo.
  - Testável.
  - Sustentável.
- Atualmente, o método Welcome na classe HelloWorldController usa um parâmetro name e um ID e, em seguida, gera os valores diretamente no navegador.

Adicionando uma View

- Em HelloWorldController.cs, altere o método Welcome para adicionar um valor Message e NumTimes ao dicionário ViewData.
- O ViewData dicionário é um objeto dinâmico, o que significa que qualquer tipo pode ser usado.
- O ViewData objeto não tem propriedades definidas até que algo seja adicionado.
- O sistema de model binding MVC mapeia automaticamente os parâmetros nomeados e da cadeia de caracteres de consulta para name numTimes parâmetros no método.

Adicionando uma View

- O método Welcome do HelloWorldController:

```
public IActionResult Welcome(string name, int numTimes = 1)
{
    ViewData["Message"] = "Hello " + name;
    ViewData["NumTimes"] = numTimes;

    return View();
}
```

- O objeto de dicionário ViewData contém dados que serão passados para a exibição.

- Crie um modelo de exibição Boas-vindas chamado Views/HelloWorld/Welcome.cshtml.

Adicionando uma View

- Você criará um loop no modelo de exibição Welcome.cshtml que exibe “Olá” NumTimes.

- Substitua o conteúdo de Views/HelloWorld/Welcome.cshtml pelo seguinte:

```
@{
    ViewData["Title"] = "Welcome";
}

<h2>Welcome</h2>

<ul>
    @for (int i = 0; i < (int)ViewData["NumTimes"]!; i++)
    {
        <li>@ViewData["Message"]</li>
    }
</ul>
```

Adicionando uma View

- Salve as alterações e navegue para a seguinte URL:
- https://localhost:{PORT}/HelloWorld/Welcome?name=Ricardo&numtimes=4
- Os dados são retirados da URL e passados para o controlador usando o model binder do MVC.
- O controlador empacota os dados em um dicionário ViewData e passa esse objeto para a exibição.
- Em seguida, a exibição renderiza os dados como HTML para o navegador.

Referências

- Visão sobre ASP.NET Core 6. https://docs.microsoft.com/pt-br/aspnet/core/mvc/overview?view=aspnetcore-6.0