**SUMÁRIO**

[PADRÃO MVC 2](#_Toc81675820)

[CONTROLLER 3](#_Toc81675821)

[MODEL 7](#_Toc81675822)

[VIEWS 8](#_Toc81675823)

# PADRÃO MVC

O padrão MVC **(Model, View e Controller)** juntos eles formam um padrão arquitetural, diferente de estilo arquitetural. No **estilo arquitetural** definimos uma arquitetura por completa. No **padão arquitetural** ela aplica uma aplicação em cima de uma arquitetura.

Nesse padrão temos uma separação de responsabilidades, cada parte da estrutura é responsável por um processo.

Essa arquitetura está presente desde 1976 criado pela Xerox, sendo um padrão aberto aplicando-o em diversas plataformas.

**Controller:** é responsável de fazer as interações com as outras partes do sistema, banco de dados, models e views.

**Model:** é a classe que terá todas as propriedades da classe.

**View:** é a exibição das informações na tela para o usuário.

# CONTROLLER

**ROTAS**

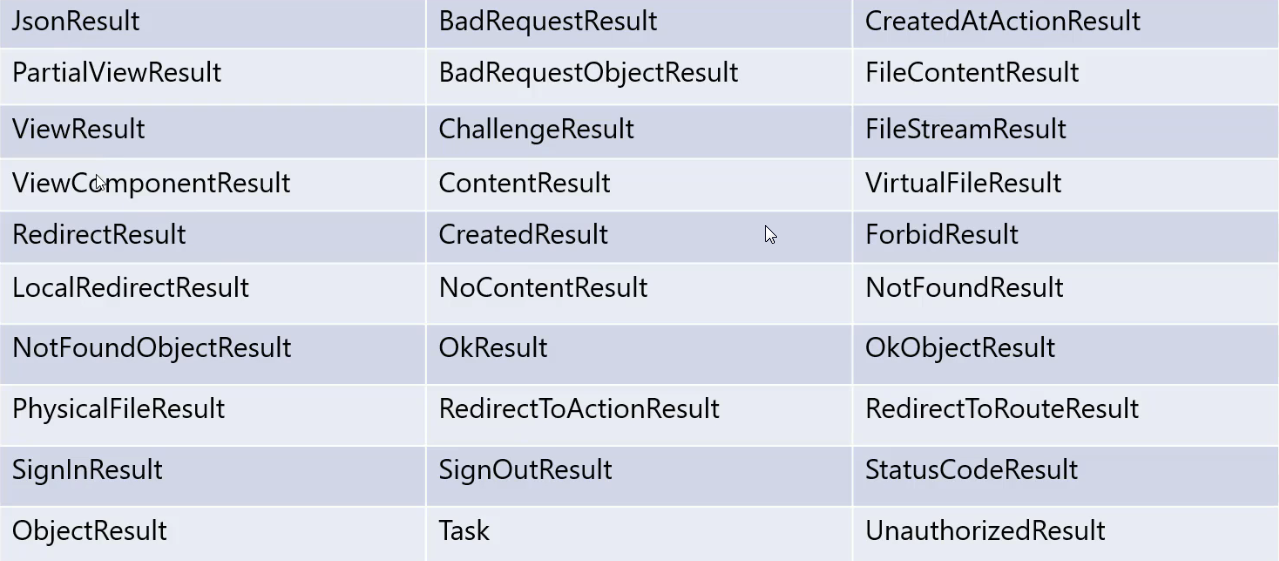
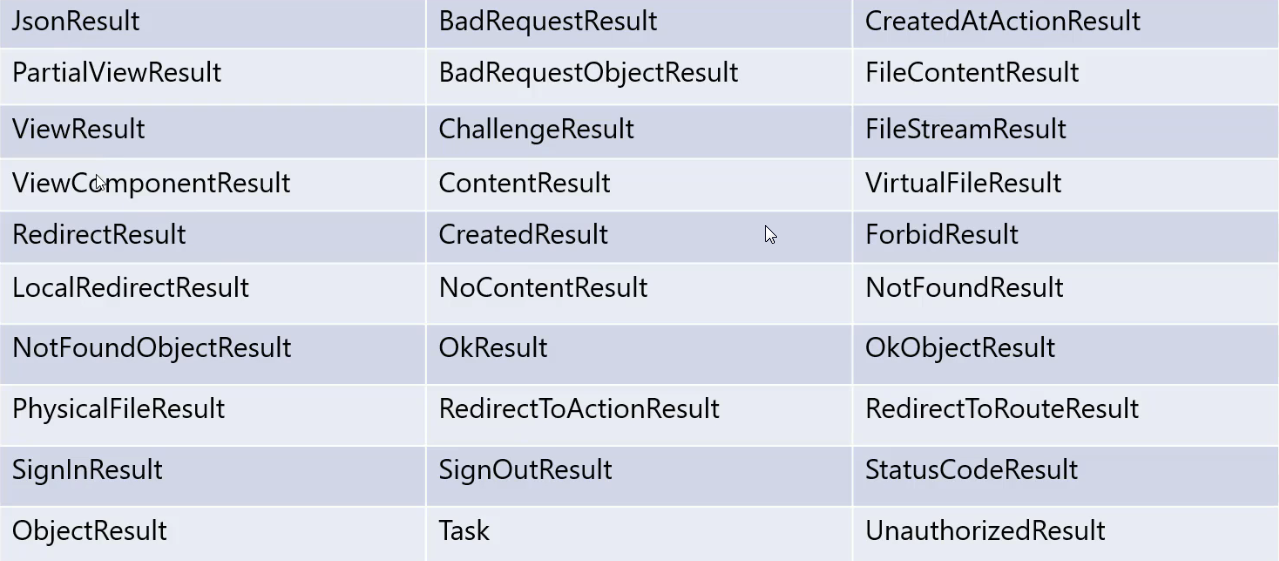
Rotas são basicamente **estruturas de navegação** personalizadas para que a URL da aplicação possua determinado padrão e atenda as necessidades de passagem de parâmetros.

As rotas possuem um **name**, possui um nome de rota, podendo ter outros. **Template**: mostra o caminho padrão que deve ser seguido na aplicação.

**Attribute Routes**, é uma maneira diferente de trabalhar com ortas, tornando-as mais flexíveis e fáceis de personalizar. Elas são personalizadas diretamente nas controllers. É uma boa recomendação, pois, com ela trabalhamos com personalização isoladas, não para o projeto todo.

**ACTION RESULTS**

No ASP.NET Core, uma **Action Result** é o tipo de retorno da action da Controller, é utilizada a interface IActionResult. que pode retornar alguns tipos de resultados.



**PROTOCOLOS HTTP – VERBOS BÁSICOS**

* **Get:** Um request (pedido) de uma informação ao server. É feito através da URL.
* **Post:** Envia informações ao servidor (formulários).
* **Put:** Similar ao Post, envia informações ao servidor. Sua diferença é por ser utilizado para atualização de informações existentes.
* **Delete:** Solicita a exclusão de uma informação no servidor através da URL indicada.
* **LEITURA DAS DEFINIÇÕES DOS METÓDOS HTTPS.**

**CONVENÇÕES DO ASP.NET MVC:**

* Se ao chamarmos uma View, e no nome da nossa Action não possuir o mesmo nome da nossa View, podemos especificar dentro do chamado: (“**return View(nome da view)**”.
* **A rota padrão sempre deve ser a última especificada**, pois, quando desenvolvermos nossa aplicação, o ASP ele criará “pontos”. Nas rotas definidas, ele irá testando uma por uma, a fim de garantir que a rota passada na URL seja compatível. Sendo assim, é lógico deixarmos a nossa rota padrão por último.
* Os nomes passados como parâmetros em um método, **ele deve ser igual** **ao definido na rota**, pois, caso diferentes, ele será reconhecido como um parametro diferente.
* Para passar parâmetros que não estão especificados na rota, colocamos **?parametro=”Valor**”.
* É possível passar as rotas também pelo controlador, fazendo assim, toda IActionResult (view) ter um padrão ou caminho definido.
* É possivel também fazer uma sobrecarga de rotas. (CUIDADO: Cuidado com os conflitos de rotas).
  + Nota: A última sobrecarga sempre será reconhecida como padrão, porém, ela aceita as outras sobrecargas caso ela for chamada.
* Podemos passar parâmetros para a rota também [**Rota(“consumo”/{id}/{preco?}**)]
  + Podemos definir o tipo de parâmetro que serão passadas na url. Para isso, basta colocarmos nosso parâmetro e dois pontos o tipo. Exemplo: **{id:int}**

**ACTION RESULTS**

As Actions sempre retornam um resultado. As actions elas podem retornar diversos tipos de arquivos. A sua grande maioria de forma mais explicita dentro da Controller, que possui todas as heranças desses arquivos.

# MODEL

No MVC um **Model** é a representação de um objeto do mundo real. Na maioria das vezes, este objeto pode representar uma tabela de um banco de dados.

**Annotations** é um importante recurso utilizado principalmente para especificar que tipo de dado a propriedade deve receber. Elas podem definir tamanho, padrões, obrigatoriedade etc.

**VALIDAÇÃO DA MODEL**

Para validarmos a nossa própria model, acessamos o “dicionário” de erros que a Model e o ASP fazem para nós. Com isso, conseguimos saber todos os erros.

Uma das formas de validar a model é o **ModelState.IsValid**

Uma outra forma é acessando o **ModelState.Values.SelectMany.**  foreach (var error in ModelState.Values.SelectMany(m => errors))

{

Console.WriteLine(error.ErrorMessage);

}

# VIEWS

* **Razor Views:** as Razor Views é o motor de renderização das views no MVC. Ele transforma as views em HTML puro, que serão interpretadas pelo browser. Elas também podem ser tralhadas com models, **CADA VIEW PODE TRABALHAR APENAS COM UMA MODEL.**
* **Tag Helpers:** as tag helpers são um auxílio que o asp.net dá as páginas HTML. Ela nos ajuda com diversas coisas, desde uma validação de um input (**asp-validation)**, trabalhar com conexões com models (**asp-for**). Assim, com as tag helpers, as nossas páginas HTML ficam muito mais inteligentes.
* **\_ViewStart Page:** é dizer qual página que será o layout, ou, a página de padronização das views.
* **\_ViewImports Page:** Importar componentes que serão utilizadas de forma global.
* **Shared:** Aqui é onde ficam as views que serão compartilhadas em outras views. Por exemplo: página de erro, página de avisos etc.
* **Home:** Por convenção do ASP.NET MVC, todas as views elas ficaram em pastas de acordo com sua Controller. Assim, cada pasta corresponde as views de determinadas Controllers.
* **\_Layout.cshtml:** nessa página é onde determinamos como será o nosso layout, formato, padrão e estilos.
* **Partial Views:** As partial views são pedaços de views que podem ser reaproveitadas em qualquer outra view. **Exemplo:** quando tivermos um pedaço de html que sempre se repete, podemos colocar em **Partial Views**, que poderão ser implementadas dentro de outras views. As partial views **dependem** de um model, limitando assim o seu uso.
* **View Components:** Parecido com uma Partial View, as Views Components são componentes independentes que auxiliam no desenvolvimento das views. Elas possuem processamento **server-side** e são **exibidos como um pedaço de HTML**, assim como as partial view. Além disso, elas são **independentes de models**, podendo trabalhar de forma mais eficiente. **Exemplo**: São excelentes para componentizar recursos da página, como o carrinho de compras.
* Os underlines na frente dos nomes das **Views,** por convenção,representam que aquela view trabalha em conjunto com outras views.