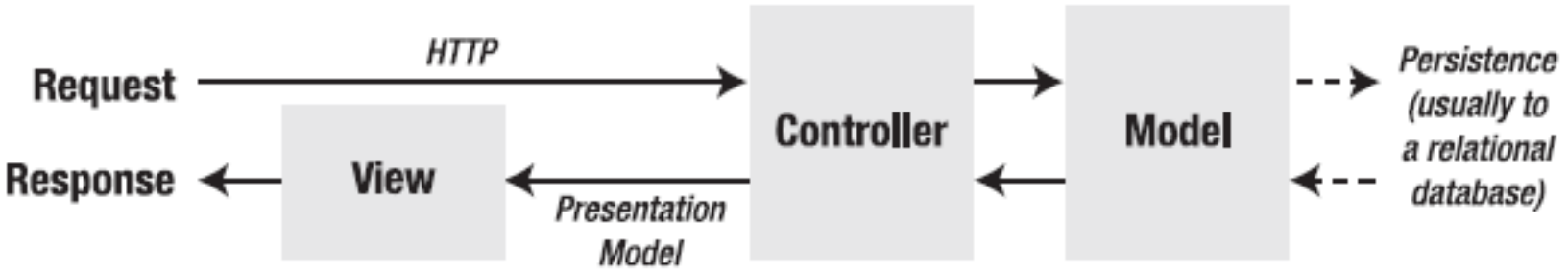
Dell IT Academy

ASP.NET CORE MVC

Recursos

- Documentação
  - https://docs.microsoft.com/en-us/aspnet/core/
  - https://docs.microsoft.com/en-us/aspnet/core/mvc/
- Fontes
  - https://github.com/aspnet/home

Ciclo de vida MVC Web



Fonte: Freeman e Sanderson

Uma aplicação ASP.NET Core MVC

- A maior parte da configuração se dá por convenções
  - Nomes de classes e pastas
  - Associações entre controladores e visões
  - Roteamento
- Todos os componentes podem ser customizados

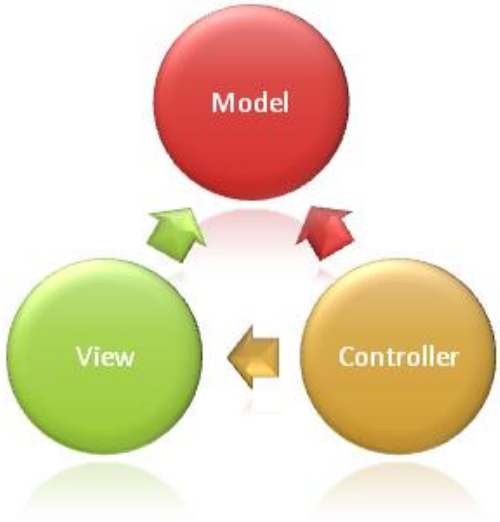
Roteamento

- O roteamento seleciona o recurso do servidor Web que é acionado por uma determinado endereço URL/URI.
- O MVC utiliza o roteamento para indicar qual controlador deve responder a uma requisição.

CONTROLLERS

Visão Geral do MVC

- Model: objetos que representam o domínio da aplicação
- View: componentes que apresentam a interface de usuário
- **Controller: componentes que tratam a interação com o usuário, operam sobre modelos e selecionam a visão adequada**



MVC *Controller*

- Um controlador é responsável por:
  - receber requisições HTTP
  - dados de entrada
  - alteração de modelos
  - retornar resposta ao usuário (dados de saída)
- Um controlador é uma abstração de nível de interface do usuário
  - Em uma separação de responsabilidades bem-feita não irá conter código de negócio ou de persistência

MVC *Controller*

- Classe do *framework* MVC
  - Microsoft.AspNetCore.Mvc.Controller
- Opera com o modelo requisição/resposta do HTTP
- Em geral, os objetos *Request* e *Response* não são alterados diretamente.
- Recebem as dependência via injeção
  - Construtor
  - Métodos de ação

```
public class HomeController : Controller
{

        public void Index()
        {
            Response.Write("<H1>Hello MVC!</H1>");
        }
 }
```

Métodos de Ação

- Método público de um objeto controlador que pode receber parâmetros e retornar um objeto (normalmente do tipo *IActionResult*).

```
public IActionResult Details(int id)
{
   var album = storeDB.Albums.Find(id);
    if (album == null)
    {
        return NotFound();
    }
    return View(album);
 }
```

Anotações para Métodos de Ação

- Para indicar um método como não sendo uma ação pública:
  - [NonAction]

Métodos de Ação GET e POST

- Normalmente, um método apresenta sobrecarga para GET e para POST
  - GET ao realizar acesso à página
  - POST para coletar o resultado de um formulário

Tipos de resultado
Subclasses de *ActionResult*

- **ViewResult** – marcação HTML
- **EmptyResult** – sem resultado (página HTML sem conteúdo)
- **RedirectResult** – redireciona para uma nova ULR
- ***PartialViewResult*** – gera uma seção de HTML
- **JsonResult** – objeto JavaScript Object Notation, normalmente utilizado em aplicações AJAX
- **JavaScriptResult** – script JavaScript
- **ContentResult** – retorna texto
- **FileContentResult** – arquivo para download (com conteúdo binário)
- **FilePathResult** – arquivo para donwload (com caminho)
- **FileStreamResult** – arquivo para download (com um *filestream*)

Gerar tipos de retorno
Métodos da classe Controller

- **View** – retorna um elemento do tipo *ViewResult*
- **Redirect** – retorna um elemento do tipo *RedirectResult*
- **RedirectToAction** – retorna um elemento *RedirectResult*
- **RedirectToRoute** – retorna um elemento *RedirectResult*
- **Json** – retorna um elemento *JsonResult*
- **JavaScriptResult** – retorna um elemento *JavaScriptResult*
- **Content** – retorna um elemento *ContentResult*
- **File** – retorna um elemento FileContentResult, FilePathResult, or FileStreamResult

Compartilhamento de dados

- ***ViewBag****:* permite passar informações do método de ação para a *view* por meio de propriedades (objetos dinâmicos de qualquer tipo)
- ***ViewData***: idem, utilizando um dicionário
- ***ViewModel:*** permite passar objetos fortemente tipados para as views

Exemplos de uso

Controller

```
ViewBag.Data= DateTime.Now;
```

View

```
<h4>Hora: @ViewBag.Data.ToShortDateString()</h4>
```

Controller

```
ViewData["Hora"] = DateTime.Now;
```

View

```
<h4>Data: @(((DateTime)ViewData["Hora"]).
                ToShortTimeString())</h4>
```

Parâmetros

- Requisições podem conter parâmetros
- A identificação de parâmetros ocorre por:
  - Variáveis
  - Regras de roteamento
- Geralmente, os parâmetros são fornecidos por uma URL, ou pelo envio de um formulário ou um *link* em uma página HTML.

Model binding

- Determina como os parâmetros devem ser passados para o controlador
- Por padrão é utilizado um *model binder* que busca valores para parâmetros (mapeados por nome) das ações da seguinte forma (nesta ordem):
  - *Valores de formulário (coleção Request.Form)*
  - *Valores definidos nas regras de roteamento*
  - *Valores de query-strings*
- O *binder* não gera uma exceção se o processo falhar
  - Cada ação deve verificar a propriedade *ModelState.IsValid*

Model binding

- Customização via atributos/anotações
  - [BindRequired] – adiciona um estado de erro se o *binding* não ocorrer
  - [BindNever] – a propriedade não participa do *binding*
  - [FromHeader], [FromQuery], [FromRoute], [FromForm] – especifica a fonte a ser utilizada para o *binding*
  - [FromServices] – define a fonte como o serviço configurado na injeção de dependências
  - [FromBody] – define a fonte como o corpo da requisição
    - JsonInputFormatter é o padrão
  - [ModelBinder] – utilizado para sobrescrever o *binder*

Validação

- Etapa que ocorre após *model binding*
- Questões de validação:
  - Erros causados por incompatibilidade de tipos de dados
    - A estrutura do banco de dados foi alterada
    - O banco de dados rejeita um operação
  - Erros causados por incompatibilidade com regras de negócio
    - Representados via anotações
    - Utilizam classes parciais e classes auxiliares

Validação

- Diferentes anotações em *System.ComponentModel.DataAnnotations*:
  - Required
  - RegularExpression
  - StringLength
  - Range
  - etc
- Atributos de validação customizados
- Validação:
  - Propriedade *ModelState.IsValid*
  - Executar explicitamente pelo método *TryValidateModel()*

Exemplo: atributos de validação

```
[Required(ErrorMessage= "Informe nome")]
[StringLength(25,
              MinimumLength = 3,
              ErrorMessage = "Nome deve ter entre 3 e 25 caracteres!")]
public String Nome { get; set; }


[Required]
[EmailAddress(ErrorMessage ="EMail inválido")]
public int EMail { get; set; }


[Required]
[RegularExpression(@"((\d{2}))?\d{8}",
                   ErrorMessage = “Telefone invalido!")]
public string Telefone { get; set; }


[Range(0, 100, ErrorMessage = "Valor deve estar entre 0 e 100.")]
public int Percentual { get; set; }
```