

O que é um Controller?


Dentro do padrão **Model-View-Controller** um Controller é responsável por receber e processar o início da requisição gerada para uma aplicação web ou uma API

- Normalmente não possui regra de negócio
- Deve estar sempre dentro do diretório **Controllers**
- Herda da classe ***Microsoft.AspNetCore.Mvc.Controller***
- O nome da classe deve terminar com a palavra **Controller**
- Recebe, processa e retorna o resultado para o solicitante
- Não possui regras de interface, estas ficam a cargo da **View**

Actions

São os métodos implementados dentro do controller e que são expostos para receber as requisições. Para *desativar* uma Action basta incluir o atributo `[NonAction]`. Podem retornar qualquer coisa mas normalmente retorna um objeto do tipo `ActionResult`, que formata a estrutura de retorno.

```
public class HomeController : Controller
{
    [HttpGet]
    0 references
    public IActionResult Index()
    {
        return Content("Action default");
    }
}
```



Mapeando Controllers - Modo Convencional

Na estrutura de configuração do endpoint, mais utilizada para aplicações MVC, podemos direcionar a chamada para um método de um controller e uma action através de palavras chave. Também é possível definir a passagem de parâmetro opcional ou não.

```
1  var builder = WebApplication.CreateBuilder(args);  
2  var app = builder.Build();  
3  
4  app.MapControllerRoute(  
5      name: "default",  
6      pattern: "{controller=Home}/{action=Index}/{id?}");  
7  
8  app.Run();  
9
```

Mapeando Controllers – Usando Atributos

Mais utilizada na construção de APIs os atributos mapeiam a rota correspondente a implementação do action e do controller. Cada controller então possui sua configuração de rota e os detalhes de cada método, incluindo os verbos HTTP.

```
[ApiController]
[Route("Home")]
0 references
public class HomeController : ControllerBase{
    [HttpGet]
    [Route("/")]
    0 references
    public string Get()
    {
```

Action Filters

São atributos que *configuram* a action ou o controller para restringir algum cenário ou alguma execução. Os filtros de requisição, como o **HttpGet** por exemplo, permitem que apenas o verbo configurado seja aceito na chamada. Ainda é possível passar a rota como parâmetro.

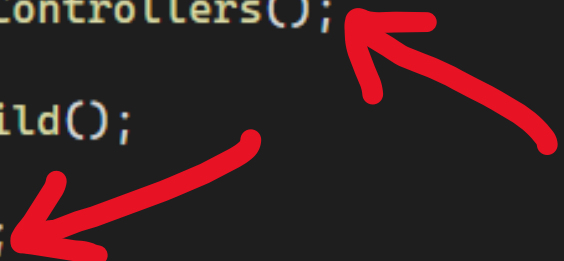
```
[HttpPost("Salvar")]  
0 references  
public string Salvar(string mensagem)  
{  
    return $"A {mensagem} foi salva com sucesso";  
}  
/[Controller]/Salvar
```

Adicionando Suporte aos Controllers

Dentro de uma aplicação ASP.NET Core podemos incluir os recursos de acordo com a necessidade do projeto. Isto facilita a modularização, a performance e o controle de configuração.

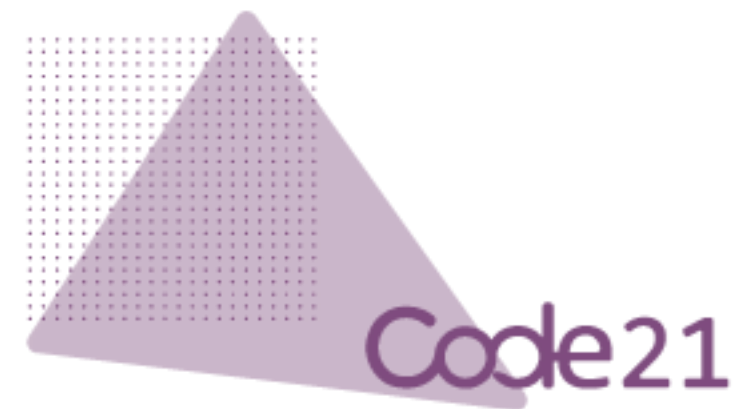
Para que o projeto *entenda* os controllers precisamos adicionar na inicialização “*Program.cs*” o serviço de *suporte* aos controllers, que será carregado e permitira o uso das funcionalidades.

```
var builder = WebApplication.CreateBuilder(args);  
builder.Services.AddControllers();  
  
var app = builder.Build();  
  
app.MapControllers();  
app.Run();
```



Exercício 05

Criando e entendendo os Controllers

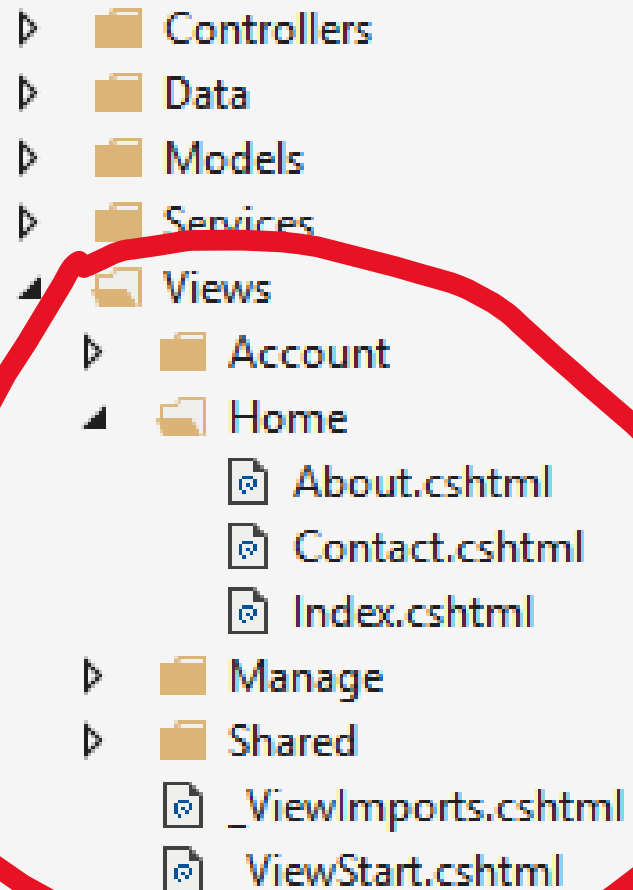


Trabalhando com Views

As Views são responsáveis por gerenciar a exibição dos dados e a interação com os usuários. É composta por um documento HTML com marcações *Razor* que são executadas no servidor.

- Arquivos com extensão ***.cshtml***
- Ficam em um diretório ***Views*** dentro do projeto
- Também é necessário registrar o uso no Services do projeto
- Seguem a estrutura dos Controllers para os subdiretórios
- Para reaproveitamento de código pode se usar:
 - *Layout pages*
 - *Partial Views*
 - *View components*

Trabalhando com Views



- ▶ Controllers
- ▶ Data
- ▶ Models
- ▶ Services
- ▶ Views
 - ▶ Account
 - ▶ Home
 - 📄 About.cshtml
 - 📄 Contact.cshtml
 - 📄 Index.cshtml
 - ▶ Manage
 - ▶ Shared
 - 📄 _ViewImports.cshtml
 - 📄 _ViewStart.cshtml

Trabalhando com Views


CSHTML

```
@{  
    ViewData["Title"] = "About";  
}  
<h2>@ViewData["Title"].</h2>  
<h3>@ViewData["Message"]</h3>  
<p>Use this area to provide additional information.</p>
```

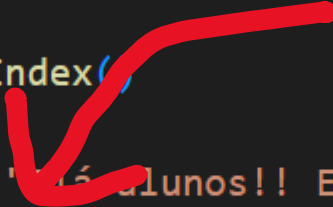
Controller -> Views

- Busca pelo nome do Método (Action)
- Especificado pelo método de retorno **View()**;
- Permite a customização da View que será exibida

```
[HttpGet]
0 references
public IActionResult Index()
{
    ViewBag.Message = "Olá alunos!! Esta é uma mensagem para a View!";
    return View();
}
```



```
[HttpGet]
0 references
public IActionResult Index()
{
    ViewBag.Message = "Olá alunos!! Esta é uma mensagem para a View!";
    return View("OutraView");
}
```



Exercício 06

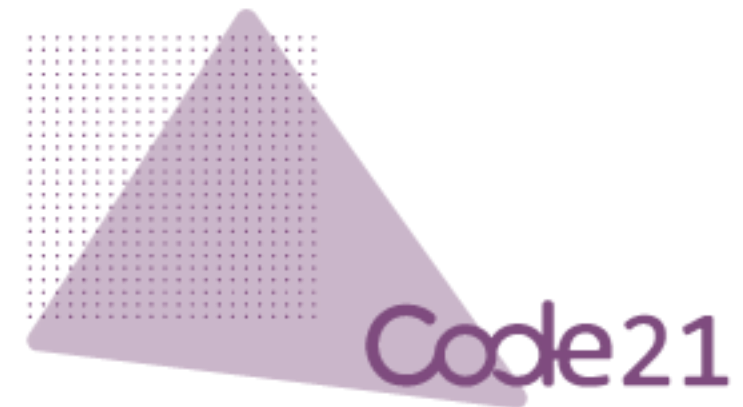
Criando as Views no Projeto



Transferindo Dados

Para exibir as informações para o usuário o Controller deve enviar os dados para uma View. Isto é possível utilizando as seguintes abordagens:

- Views tipadas: ViewModels
- ViewData (Atributos)
- ViewBag (Tipo Dinâmico)

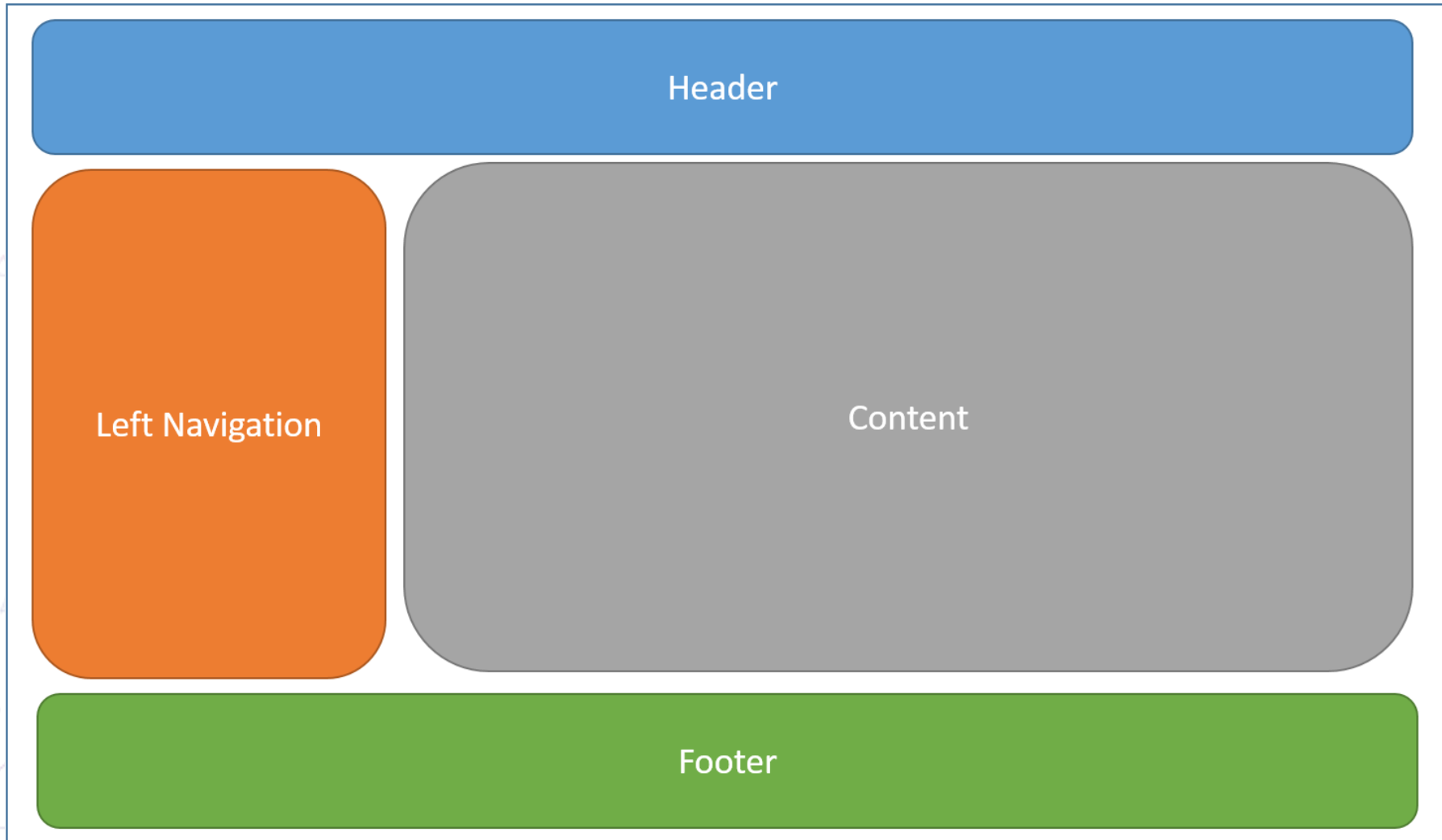


Layout Pages

Normalmente os aplicativos Web possuem trechos de layout comuns a várias páginas, estes blocos comuns são resolvidos com uso de layout pages.

- Usado para trechos de layout comuns e evitar repetição de código
- Compartilhar diretivas de inclusão de referências
- Executar códigos antes da renderização das Views
- Referenciado pela diretiva “*Layout*”
- Usos mais comuns para navegação, menu, rodapé, cabeçalho, etc...

Layout Pages



ViewImports

- As Views também possuem recursos e funções da framework
- Ou seja, é necessário importar as referências que serão usadas
- O arquivo ViewImports referencia as dependências globalmente
- Além de configurar alguns elementos como as Tag Helpers
- Normalmente incluído no diretório Views
- Porém pode ser incluído em qualquer parte da hierarquia

```
_ViewImports.cshtml  ▢ X
1  @using Microsoft.AspNetCore.Identity
2  @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
3
4
```


ViewStart

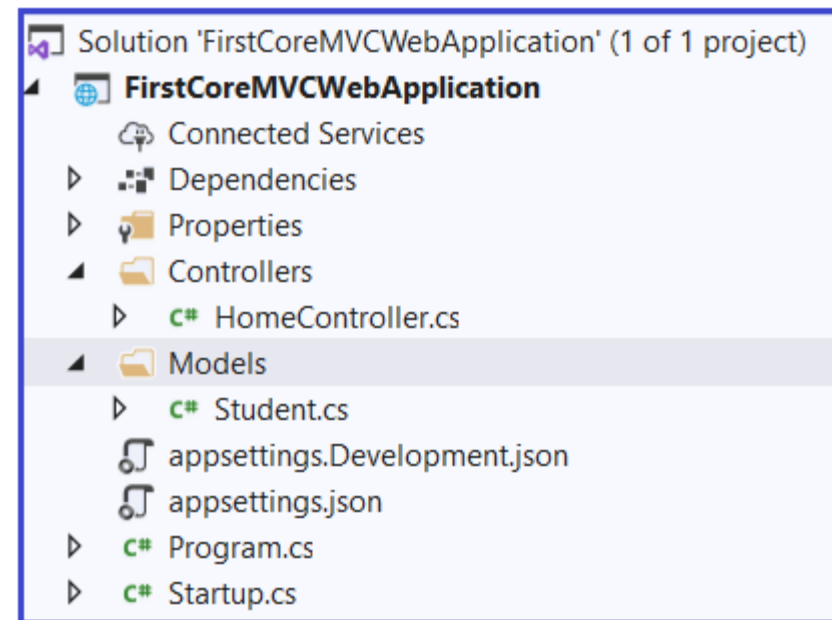
- Rotina que será executada antes de cada View
- Normalmente incluído no diretório Views
- Porém pode ser incluído em qualquer parte da hierarquia
- Não aplicado as LayoutPages ou PartialViews

```
_ViewStart.cshtml  + X
1  @{
2      Layout = "_Layout";
3  }
```

Entendendo as Models

Responsável pela estrutura da representação dos dados no sistema, o modelo é criado dentro do mesmo projeto ou em uma estrutura de bibliotecas independente, melhorando assim a organização.

- Normalmente responsável também pela persistência dos dados
- Não existe uma obrigação de implementação na arquitetura
- Pode ser criado com estruturas e complexidades diferentes



Typed Views

As Views Tipadas permite *vincular* uma view a uma classe. Essas classes são normalmente chamadas de ViewModel, pois fazem a *comunicação* entre o modelo de dados e a estrutura visual

- A View recebe uma *instância* do objeto vinculado
- As propriedades ficam disponíveis na View permitindo o *intellisense*
- Facilita e ajuda na produtividade na construção dos elementos
- Passagem das informações para o Controller fica mais simples
- Permite incluir validações do Modelo dentro da View

Trabalhando com Tag Helpers

Ajudam o servidor a criar e renderizar as Tags HTML dentro de uma View Razor antes que ela seja encaminhada para o cliente.

- Semelhantes a Tags HTML
- Suportam Intellisense durante o desenvolvimento
- Permitem ao design trabalhar na View sem conhecer C#
- Evolução das HTML Helpers (ASP.NET MVC 4.x)
- Mais simples e fáceis de ler
- Não incluem novos elementos apenas modificam



Tag Helpers

```
<form asp-controller="Account" asp-action="Register" method="post" class="form-hori
  <h4>Create a new account.</h4>
  <hr />
  <div asp-validation-summary="ValidationSummary.All" class="text-danger"></div>
  <div class="form-group">
    <label asp-for="Email" class="col-md-2 control-label"></label>
    <div class="col-md-10">
      <input asp-for="Email" class="form-control" />
      <span asp-validation-for="Email" class="text-danger"></span>
    </div>
  </div>
  <div class="form-group">
    <label asp-for="Password" class="col-md-2 control-label"></label>
    <div class="col-md-10">
      <input asp-for="Password" class="form-control" />
      <span asp-validation-for="Password" class="text-danger"></span>
    </div>
  </div>
```

Exercício 07

Desenvolvendo com Model, View e Controller

