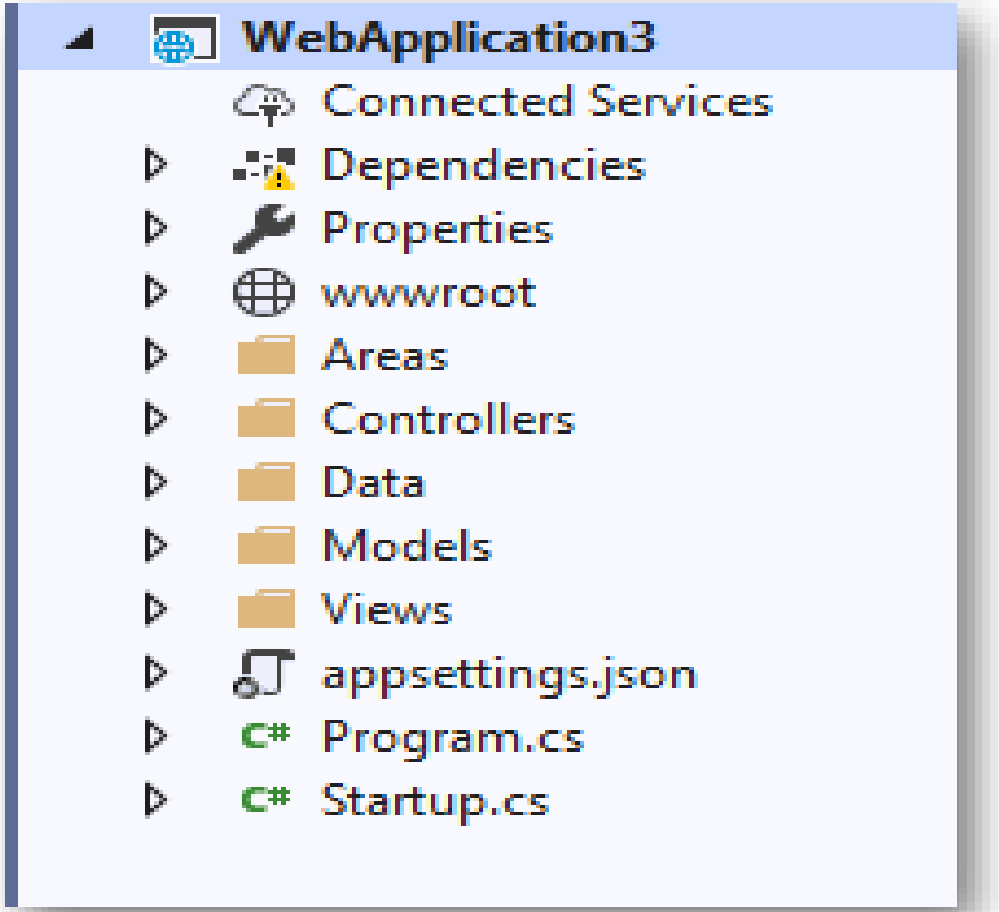
Современные платформы прикладной разработки

ПРЕДСТАВЛЕНИЯ

Представления



Представления

Представление отвечает за пользовательский интерфейс приложения

Фалы представления имеют расширение **.cshtml** и содержат HTML-разметку и блоки кода на языке **Razor**.

Представления

Файлы представлений располагают в папке:

~/Views/<имя контроллера> - если представление используется только конкретным контроллером

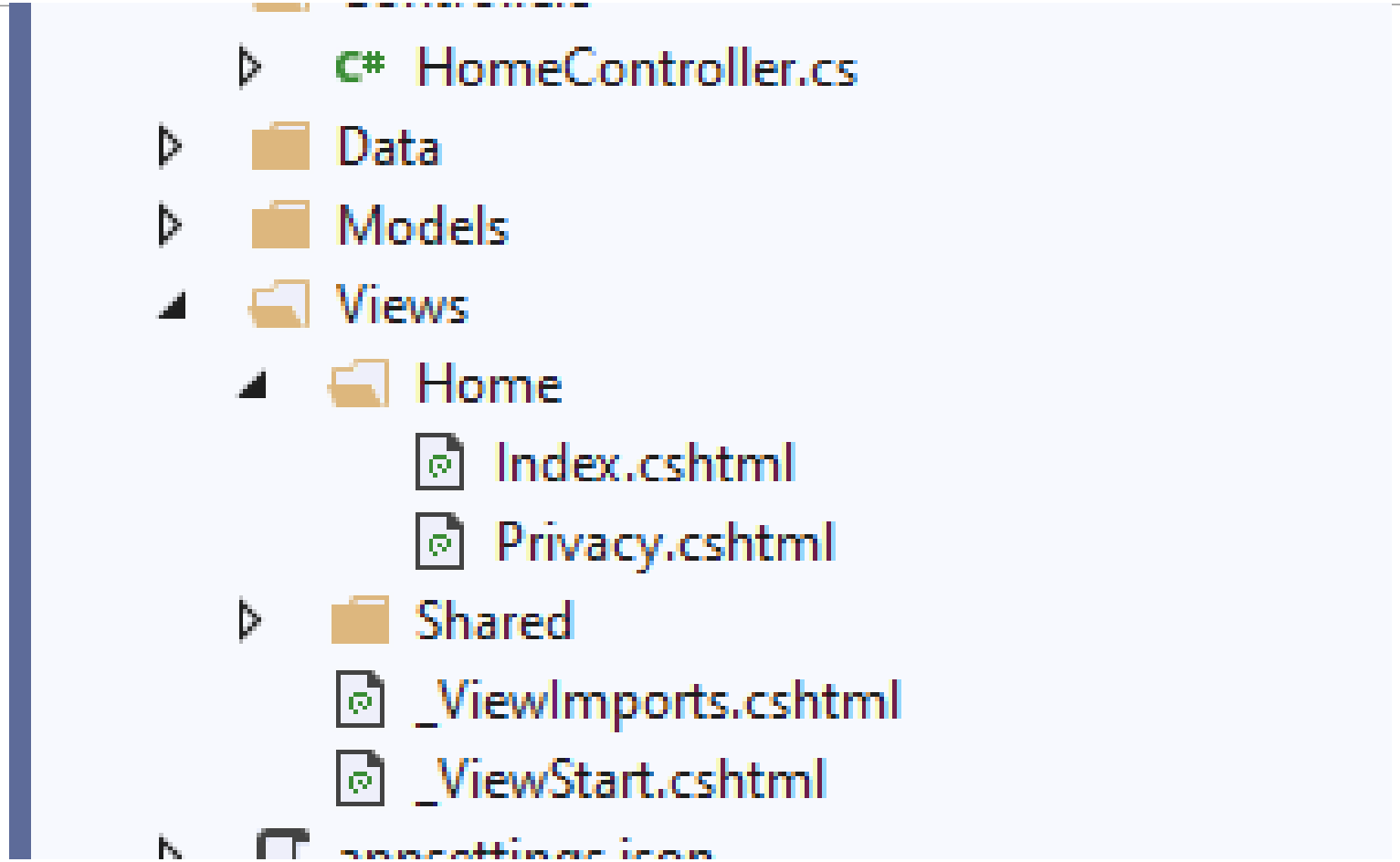
или

~/Views/Shared – если представление может использоваться несколькими контроллерами

Например, представление **Index** контроллера **Home** будет находится в папке

~/Views/Home

Представления



Язык Razor

Язык Razor

Razor - это синтаксис разметки для серверного кода в веб-страницы.

Синтаксис Razor состоит из разметки Razor, C # и HTML.

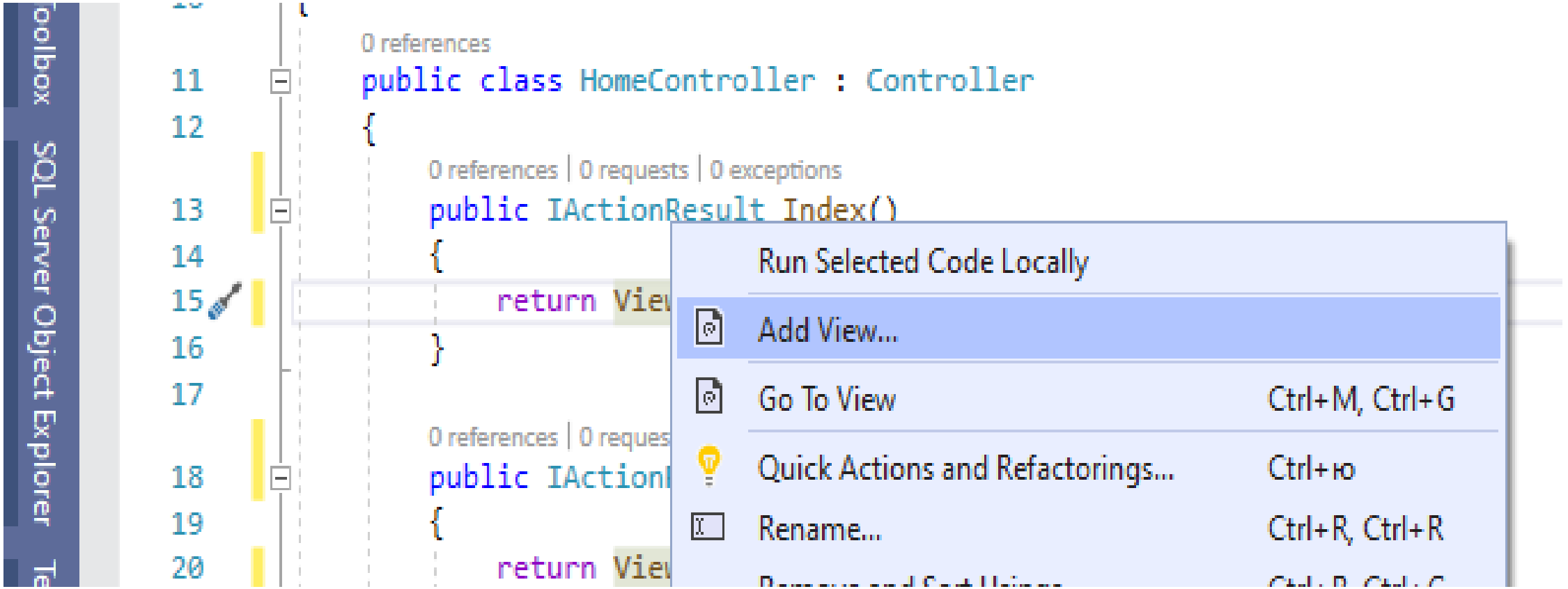
Язык Razor

Любой код Razor начинается с символа @.

Если за символом @ следует зарезервированное в Razor ключевое слово, то код транслируется в разметку HTML.

В противном случае код воспринимается просто как код C#.

Добавление представления



Добавление представления

Неявные выражения Razor

Неявные выражения Razor – это код C#, идущий после символа @ и не содержащий пробелов

Неявные выражения Razor

```
<body>
    <h2>Today is
@DateTime.Now.ToShortDateString()</h2>
</body>
```

Неявные выражения Razor

В неявных выражениях нельзя использовать обобщения, т.к. угловые скобки будут восприниматься как разметка html.

Явные выражения Razor

Явные выражения Razor следуют за символом @ и заключены в парные круглые скобки.

Явные выражения Razor

```
 <body>
      <h2>Today is
@DateTime.Now.ToShortDateString()</h2>
      <h2>Tomorrow is @(DateTime.Now +
TimeSpan.FromDays(1))</h2>
 </body>
```

Явные выражения Razor

Явные выражения можно использовать для обобщений

Кодовые блоки

Кодовые блоки заключаются в фигурные скобки. В отличие от выражений кодовые блоки не преобразуются в разметку

Кодовые блоки

```
public class Book
    {
        public int BookId { get; set; }
        public string Name { get; set; }
        public string Author { get; set; }
    }
```

Кодовые блоки

```
@using XXX.Models;
@{
    var books = new List<Book>
    {
        new Book{ BookId=1, Name="Book1",
Author="Author1"},
        new Book{ BookId=2, Name="Book2",
Author="Author2"}
    };
}
```

Кодовые блоки

```
<table class="table table-striped">
        @foreach(var book in books)
        {
            <tr>
                <td>@book.BookId</td>
                <td>@book.Name</td>
                <td>@book.Author</td>
            </tr>
        }
    </table>
```

```
@functions {
    public string MakeRow(Book book) =>
        $"<tr><td>{book.BookId}</td>"
          +$"<td>{book.Name}</td>"
          +$"<td>{book.Author}</td></tr>";
}
@foreach(var book in books)
        {
            @MakeRow(book);
        }
```

```
@using Microsoft.AspNetCore.Html

. . .

public IHtmlContent MakeRow2(Book book) =>
        Html.Raw(
          $"<tr><td>{book.BookId}</td>"
            +$"<td>{book.Name}</td>"
            +$"<td>{book.Author}</td></tr>");
```

Использование TagBuilder

```
public TagBuilder MakeRow1(Book book)
    {
        // Создать тэг <tr>
        var row = new TagBuilder("tr");
        // Создать тэг 1-й колонки
        var idCol = new TagBuilder("td");
        // добавить в колонку id объекта book
        idCol.InnerHtml.Append(book.BookId.ToString());
        // Строка двух других колонок
        var otherColumns = $"<td>{book.Name}</td>"
                           + $"<td>{book.Author}</td>";
        // добавить контент в тэг <tr>
        row.InnerHtml
            .AppendHtml(idCol)
            .AppendHtml(otherColumns);
        // Вернуть полученную строку
        return row;
    }
    }
```

Файл _ViewImports.cshtml

VIEWS

```
@using XXX.UI
@using XXX.UI.Models
@using XXX.UI.Models.MembershipViewModels
@using XXX.UI.Models.DTOModels
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Макеты (Layouts)

Макеты (Layouts)

Макет (**Layout**) – это шаблон, содержащий базовую разметку веб-страницы (скрипты, CSS, структурные элементы, например, навигацию и контейнеры содержимого) и определяющий места разметке, куда представление должно поместить содержимое веб-страницы.

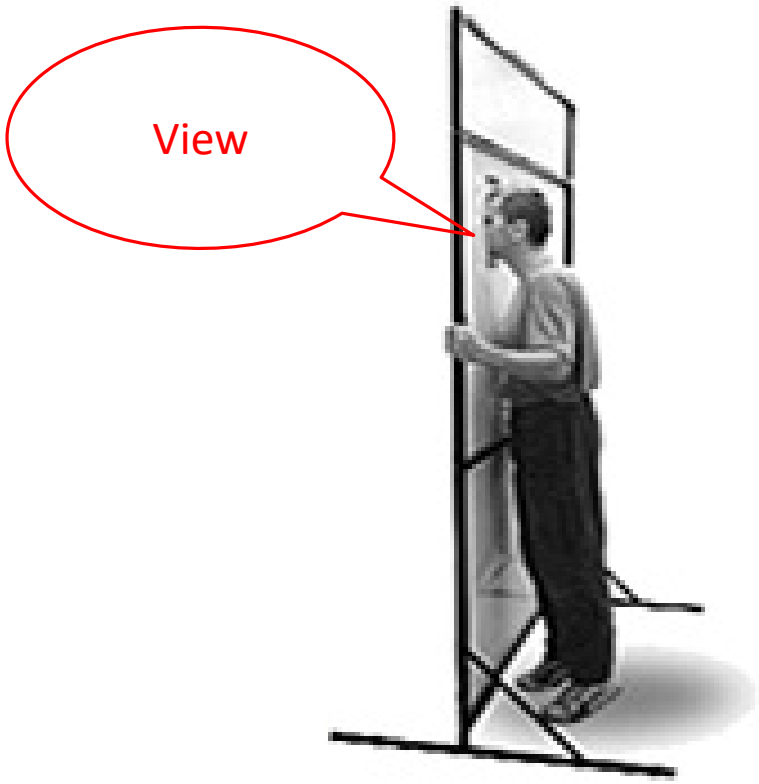
Макеты (Layouts)

@RenderSection([Имя секции])

И

@RenderBody()

Макеты (Layouts)



```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
    <link href="~/lib/bootstrap/dist/css/bootstrap.css" rel="stylesheet" />
</head>
<body>
    <div>
        <h4>Разметка макета 1</h4>
        @RenderSection("Header", required: false)
        <h4>Разметка макета 2</h4>
        @RenderBody()
        <h4>Разметка макета 3</h4>
        @RenderSection("Footer", required: false)
        <h4>Разметка макета 4</h4>
    </div>
</body>
</html>
```

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

Макеты (Layouts)

```
@{
    ViewData["Title"] = "LayoutDemo";
    Layout = "~/Views/Shared/_Layout1.cshtml";
}

<div class="badge badge-danger">Разметка из представления</div>

@section Footer
{
    <h2>Footer из представления</h2>
}
```

Файл _ViewStart.cshtml

```
@{
    Layout = "_Layout";
}
```

Не использовать макет

```
@{
    Layout = null;
}
```

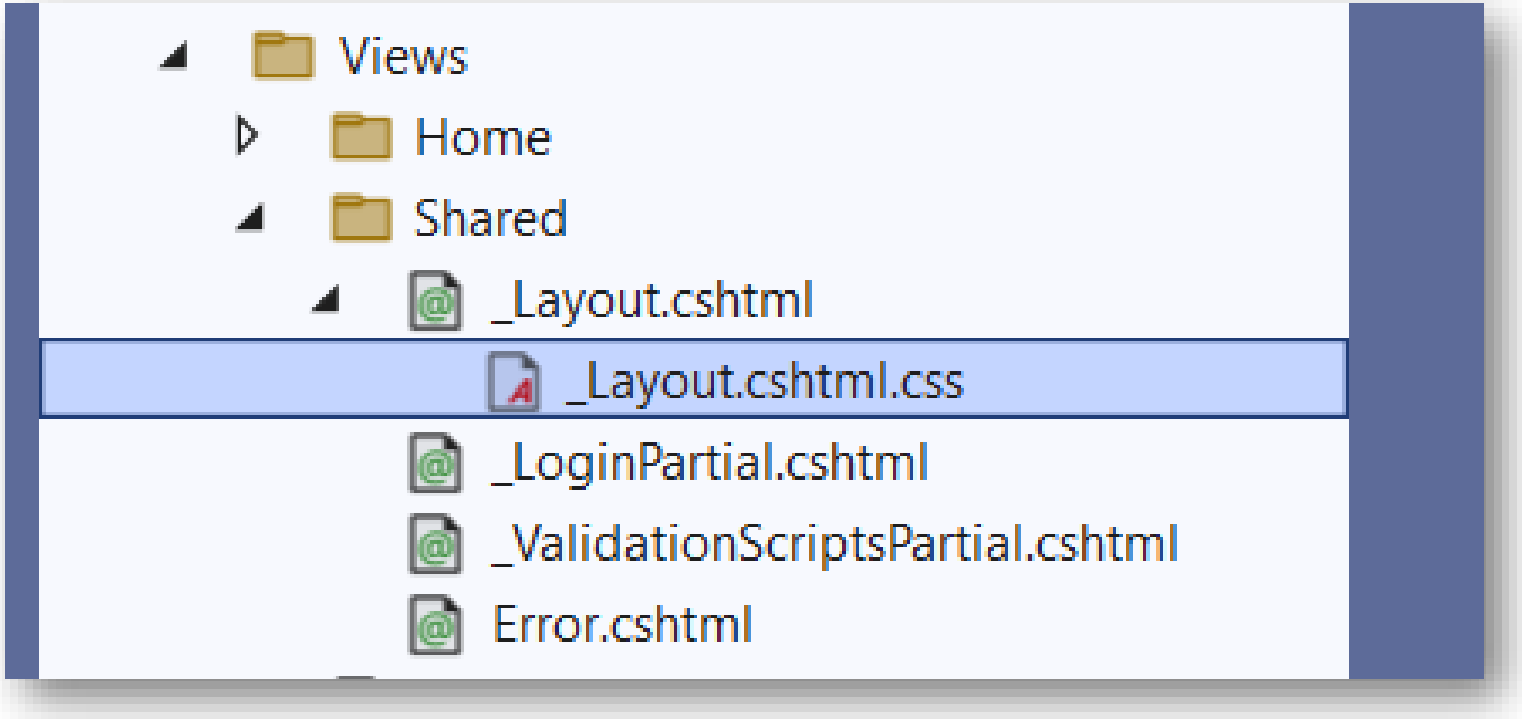
Изоляция стилей CSS

Изоляция стилей CSS

Изоляция стилей CSS для отдельных страниц, представлений и компонентов, чтобы уменьшить или избежать следующего:

- Зависимости от глобальных стилей, которые могут быть сложными в обслуживании.
- Конфликты стилей во вложенном содержимом.

Изоляция стилей CSS



Передача данных представлению

- ✓ ViewData
- ✓ ViewBag
- ✓ TempData

ViewData

ViewData представляет собой словарь, формируемый динамически, и доступный как свойство в контроллере и в представлении. ViewData наследуется от **ViewDataDictionary**, следовательно доступ к данным осуществляются с помощью пары **«ключ-значение»**.

ViewData

Запись данных:

```
ViewData["Name"]="John";
```

Чтение данных в представлении:

```
@ViewData["Name"]
```

```
public class HomeController : Controller
{
    [ViewData]
    public string UserName { get; set; }

    . . .

    public IActionResult Index()
    {
        UserName = "Bob";
        return View();
    }
. . .
```

ViewData

Строковые данные можно хранить и использовать напрямую без приведения.

При получении других значений объектов ViewData они должны быть приведены к соответствующим типам.

Пример использования ViewData

Передача данных между...	**Пример**
Контроллером и представлением	Заполнение данными раскрывающегося списка.
Представлением и представлением макета	Задание содержимого <title> элемента в режиме макета из файла представления.
Частичным представлением и представлением	Виджет, в котором выводятся данные получаемые из запрошенной пользователем веб-страницы.

Вывод данных из ViewData на макете

```
<body class="container">
    <strong>Содержимое ViewData</strong>
    @if (ViewData.Count == 0)
    {<h4>ViewData has no data</h4>}
    else
    {
        <table class="table table-bordered">
            <thead>
                <tr>
                    <th>Key</th> <th>Value</th>
                </tr>
            </thead>
            <tbody>
                @foreach(var key in ViewData.Keys)
                {
                    <tr>
                        <td>@key</td><td>@ViewData[key]</td>
                    </tr>
                }
            </tbody>
        </table>
    }
    <div>
        <strong>View markup</strong>
        @RenderBody()
    </div>
</body>
```

Вывод списка из ViewData

```
@{
    var books = ViewData["Books"] as List<Book>;

}
@if (books != null)
{
    <table class="table table-striped">
        @foreach (var book in books)
        {
            @MakeRow(book)
        }
    </table>}
else
{
    <p>No books</p>
}
```

ViewBag

ViewBag – это «обертка» ViewData, позволяющая создавать динамические свойства.

Использование ViewBag аналогично ViewData, за исключением того, что доступ к данным осуществляется через свойства:

```
ViewBag.SomeData = "Данные во ViewBag";
```

Отличия ViewBag от ViewData

ViewData

- Является производным от ViewDataDictionary, поэтому имеет свойства словаря, которые могут быть полезны, например ContainsKey, Add, Remove и Clear.
- Ключи в словаре представляют собой строки, поэтому пробел допустим. Пример: ViewData["Ключ с пробелами"]
- Любой тип, кроме string, при чтении должен быть выполнено приведение типа.

ViewBag

- Наследуется от Microsoft.AspNetCore.Mvc.ViewFeatures.Internal.DynamicViewData, поэтому позволяет создавать динамические свойства с помощью точечной нотации (@ViewBag.SomeKey = <value or object>), и приведение не требуется. Синтаксис свойства ViewBag позволяет быстрее добавлять его в контроллеры и представления.
- Проще проверять значение NULL. Пример: @ViewBag.Person?.Name

TempData

TempData - такая же коллекция, что и ViewData. Отличие заключается во времени жизни коллекции.

ViewData *хранит данные только* ***для текущего запроса*** *и уничтожается при переадресации (например, если контроллер вернет Redirect)*

TempData *не уничтожается при переадресации, поэтому данные можно передавать между запросами.*

Model

ПЕРЕДАЧА ДАННЫХ ПРЕДСТАВЛЕНИЮ

Model

Наиболее надежный подход при передаче данных представлению — указание типа модели в представлении. Такая модель называется *viewmodel*. Экземпляр типа viewmodel передается в представление из действия контроллера.

Model

В представлениях Razor для использования свойства Model необходимо указать, к какому классу относится модель. Для этого используется ключевое слово **@model,** например:

```
@model IEnumerable<Book>
```

Внедрение зависимостей

Внедрение зависимостей

```
@using Microsoft.AspNetCore.Identity
@inject SignInManager<IdentityUser> SignInManager
@inject UserManager<IdentityUser> UserManager
```

```
@if (SignInManager.IsSignedIn(User))
{
    . . .
}
```

Частичные представления

VIEWS

Частичные представления

Частичное представление – это файл на языке Razor, который генерирует разметку HTML внутри другой разметки.

Частичные представления

Частичные представления позволяют эффективно выполнять следующие задачи:

- Разбить большие файлы разметки на мелкие компоненты.
- Сократить дублирование элементов разметки в разных файлах разметки.

Частичные представления

Имена файлов частичных представлений принято начинать с нижнего подчеркивания и заканчивать суффиксом Partial, например, **_ListPartial.cshtml.**

Частичные представления

Как правило, частичное представление – это отдельный файл.

Однако, действие контроллера также может вернуть частичное представление. Для этого используется синтаксис:

```
return PartialView();
```

Частичные представления

Для получения частичного представления Microsoft предлагает два подхода:

- ✓ - использование тэга <partial>
- ✓ - использование асинхронных Html-хелперов (HTML-helpers) Html.PartialAsync и Html.RenderPartialAsync.

Тэг <partial>

```
<label class="badge-dark">тэг &lt;partial&gt;
</label>
<table class="table table-striped">
    @foreach (var book in Model)
    {
        <partial name="_RowPartial«
                 model="@book"/>
    }
</table>
```

Html.PartialAsync

```
 <label             class="badge-dark">использование
Html.PartialAsync</label>
 <table class="table table-striped">
      @foreach (var book in Model)
      {
          @await
Html.PartialAsync("_RowPartial", book)
      }
 </table>
```

Html.RenderPartialAsync

```
<table class="table table-striped">
    @foreach (var book in Model)
    {
        await
Html.RenderPartialAsync("_RowPartial", book);
    }
</table>
```

Поиск частичного представления

Если в имени частичного представления присутствует расширение файла, то частичное представление, необходимо разместить в той же папке, что и вызывающий его файл разметки

Поиск частичного представления

Если ссылка на частичное представление указывает только имя файла, без расширения, просматриваются следующие расположения в указанном порядке:

страницы Razor;

1.папка текущей выполняемой страницы;

2.Папка на уровень выше папки страницы.

3./Shared

4./Pages/Shared

5./Views/Shared

MVC

1./Areas/<Area-Name>/Views/<Controller-Name>

2./Areas/<Area-Name>/Views/Shared

3./Views/Shared

4./Pages/Shared

ViewData

Когда создается экземпляр частичного представления, он получает ***копию*** словаря **ViewData** из родительского представления.

Изменения, вносимые в данные в частичном представлении, ***не сохраняются*** в родительском представлении.

Разметка частичного представления

```
@model IEnumerable<Book>
 <tr>
        <td>@Model.BookId</td>
        <td>@Model.Name</td>
        <td>@Model.Author</td>
    </tr>
```

Компоненты представлений (ViewComponent)

VIEWS

Компоненты представлений

Компоненты представлений - это классы, которые обеспечивают логику приложения для поддержки частичных представлений или для внедрения небольших фрагментов данных HTML или JSON в родительское представление.

Компоненты представлений

Компоненты заменяют @Html.Action предыдущих версий ASP.NET.

По сравнению с частичными представлениями, компоненты позволяют реализовать сложную логику при формировании представления.

Компоненты представлений

- ✓ Работает с частью, а не со всем ответом
- ✓ Использует преимущества разделения ответственности и тестируемости, какие существуют между контроллером и представлением
- ✓ Может содержать параметры и бизнес логику
- ✓ Чаще всего используется на страницах-макетах

https://learn.microsoft.com/ru-ru/aspnet/core/mvc/views/view-components?view=aspnetcore-7.0#view-components

Применение компонентов представлений

- ✓ Динамические навигационные меню
- ✓ Облако тегов (где оно запрашивает базу данных)
- ✓ Панель входа
- ✓ Корзина
- ✓ Недавно опубликованные статьи
- ✓ Содержание боковой панели в типичном блоге
- ✓ Панель входа в систему, которая будет отображаться на каждой странице и отображать либо ссылки для выхода, либо для входа, в зависимости от состояния входа пользователя

https://learn.microsoft.com/ru-ru/aspnet/core/mvc/views/view-components?view=aspnetcore-7.0#view-components

Класс компонента

Компонент представления состоит из двух частей:

- Класс, как правило, производный от ViewComponent
- Результат, который он возвращает, как правило, представление.

Класс компонента

```
public class CartSummary : ViewComponent
    {
        private IRepository repository;
        public CartSummary(IRepository repo)
        {
            repository = repo;
        }
        public IViewComponentResult Invoke()
        {
            return View(repository.GetAll());
        }
    }
```

Использование компонента на странице

```
@await Component
            .InvokeAsync("Имя компонента")
```

Размещение компонентов

Среда выполнения ищет представление по следующим путям:

- /Views/{Имя контроллера}/Components/{Имя компонента представления}/{Имя представления}
- /Views/Shared/Components/{Имя компонента представления}/{Имя представления}
- /Pages/Shared/Components/{Имя компонента представления}/{Имя представления}
- /Areas/{Имя области}/Views/Shared/Components/{Имя компонента представления}/{Имя представления}

Размещение компонентов

Имя представления по умолчанию для компонента представления — Default, что означает, что файлы представления обычно будут называться **Default.cshtml**.