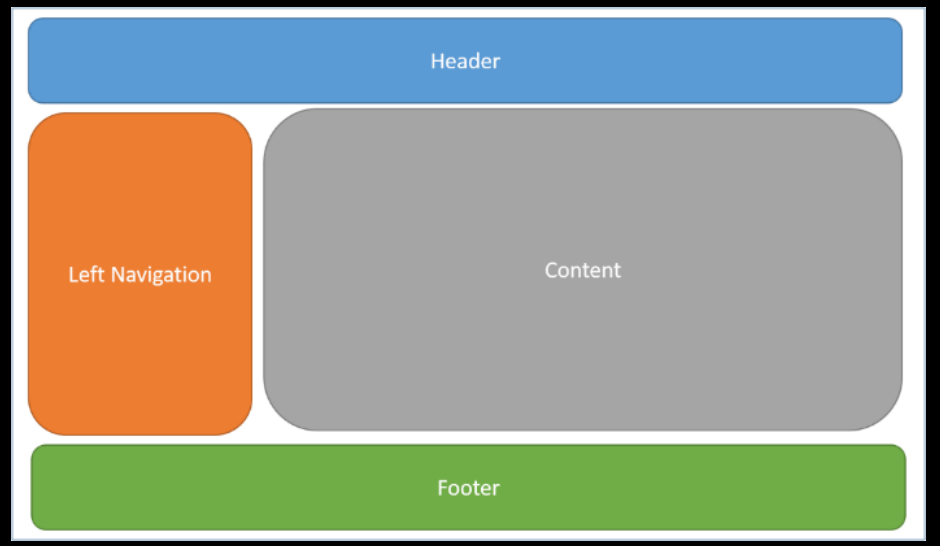
Razor изгледи

Изгледи, Razor синтаксис, Layout елементи, Tag Helpers, View компоненти

Учителски екип

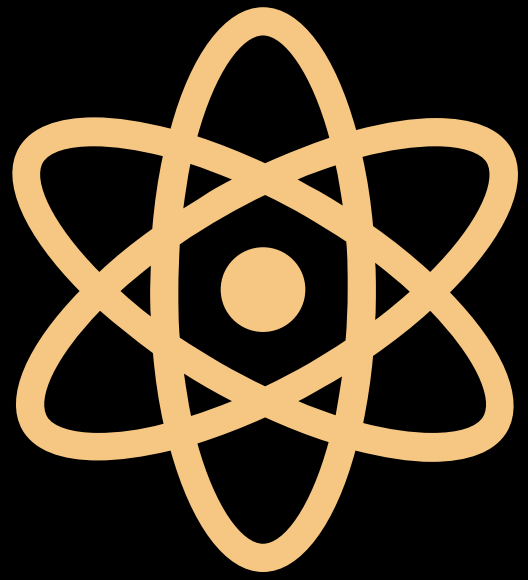
Обучение за ИТ кариера

https://it-kariera.mon.bg/e-learning/



Съдържание

- Основни изгледи
- Razor синтаксис
  - Инжектиране на зависимостта
- Файлове за оформление и специални изгледи
  - _Layout, _ViewStart, _ViewImports
- HTML Helpers & Tag Helpers
- Частични изгледи и преглед на компоненти



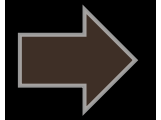
Основни елемент – View Engine

View Engine Essentials

- Изгледите в ASP.NET Core MVC използват Razor View Engine, за да вграждат .NET код в HTML маркиране.
- Обикновено те съдържат минимална логика, свързана само с представянето на данни
- Данните могат да бъдат предадени на изглед с помощта на ViewData, ViewBag или чрез ViewModel (силно типизиран изглед).

```
@{
    ViewData["Title"] = "View Engine Essentials";
}

<h1>Greetings, Developer!</h1>
<h2>Have a nice @DateTime.Now.DayOfWeek!</h2>
```



Предаване на данни към изглед

- С ViewBag (динамичен тип):
  - Действие: ViewBag.Message = "Hello World!";
  - Изглед: @ViewBag.Message
- С ViewData (dictionary)
  - Действие : ViewData["message"] = "Hello World!";
  - Изглед : @ViewData["message"]
- Със силно въведени изгледи:
  - Действие : return View(model);
  - Изглед : @model ModelDataType;

Returning Views

- Класът Base Controller осигурява много функционалност
  - Метод View () - Един от най-често използваните членове на клас Controller

```
public class HomeController :
Controller
{
    public IActionResult
Index()
    {
        return this.View();
    }
}
```

```
public IActionResult Index()
{
    return this.View
("~/Views/Other/Index.cshtml");
}
```

Как работи?

Изглед

```
@model WebApplication1.Models.UserModel

@{
    ViewBag.Title = "Home Page";
}

<div class="text-center">
    <h1>Hi: @Model.Username </h1>
    <h1>FullName: @Model.FullName</h1>
    <h1>Age: @Model.Username</h1>
</div>
```

Модел на изгледа

```
public class UserModel
{
    public string Username { get; set; }

    public string FullName { get; set; }

    public int Age { get; set; }
}
```

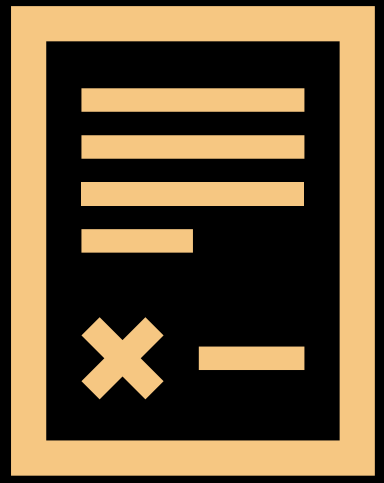
Контролер

```
public IActionResult Index()
{
    var user = new UserModel
    {
        Username = "TestUser",
        FullName = "Test User",
        Age  = 24
    };

    return View(user);
}
```

Изход

Hi: TestUser
FullName: Test User
Age: TestUser



Razor синтаксис

Razor синтаксис

- @ – За стойности (HTML кодиран)

```
<p>
     Current time is: @DateTime.Now!!!
     Not HTML encoded value: @Html.Raw(someVar)
</p>
```

- @{ … } – За код блокове (поддържайте изгледа прост!)

```
@{
     var productName = "Energy drink";
     if (Model != null) { productName = Model.ProductName; }
     else if (ViewBag.ProductName != null) { productName = ViewBag.ProductName; }
}
<p>Product "@productName" has been added in your shopping cart</p>
```

Razor синтаксис (2)

- Ако, в противен случай, for, foreach и т.н. C # изявления
  - Редовете за маркиране на HTML могат да бъдат включени във всяка част
  - @: - За да се изобрази обикновен текстов ред

```
<div class="products-list">
@if (Model.Products.Count() == 0) { <p>Sorry, no products
found!</p> }
else {
     @:List of the products found:
     foreach(var product in Model.Products) {
          <b>@product.Name, </b>
     }
} </div>
```

Razor синтаксис (3)

- Коментари

```
@* A Razor Comment
*@
@{ //A C# comment
  /* A Multi
        line C# comment */
}
```

- А как се справяме с имейл

```
<p>
     This is the sign that separates email names from domains:
@@<br />
     And this is how smart Razor is: spam_me@gmail.com
</p>
```

Razor синтаксис (4)

- @(…) – Изрично описване на код

```
<p>
     Current rating(0-10): @Model.Rating / 10.0  @* 6 / 10.0 *@
     Current rating(0-1): @(Model.Rating / 10.0) @* 0.6 *@
     spam_me@Model.Rating                        @* spam_me@Model.Rating *@
     spam_me@(Model.Rating)                      @* spam_me6 *@
</p>
```

- @using – За включване на имена в изглед
- @model – За дефиниране на модела за изгледа

```
@using MyFirstMvcApplication.Models;
@model UserModel
<p>@Model.Username</p>
```

Изгледи – Инжектиране на Зависимост

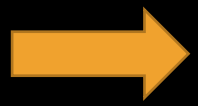
- ASP.NET Core поддържа инжектиране на зависимост в изгледи.
  - Можете да инжектирате услуга в изглед, като използвате @inject

```
public class DataService
{
    public IEnumerable<string> GetData()
    {
        return new[] { "David", "John", "Max", "George" };
    }
}
```

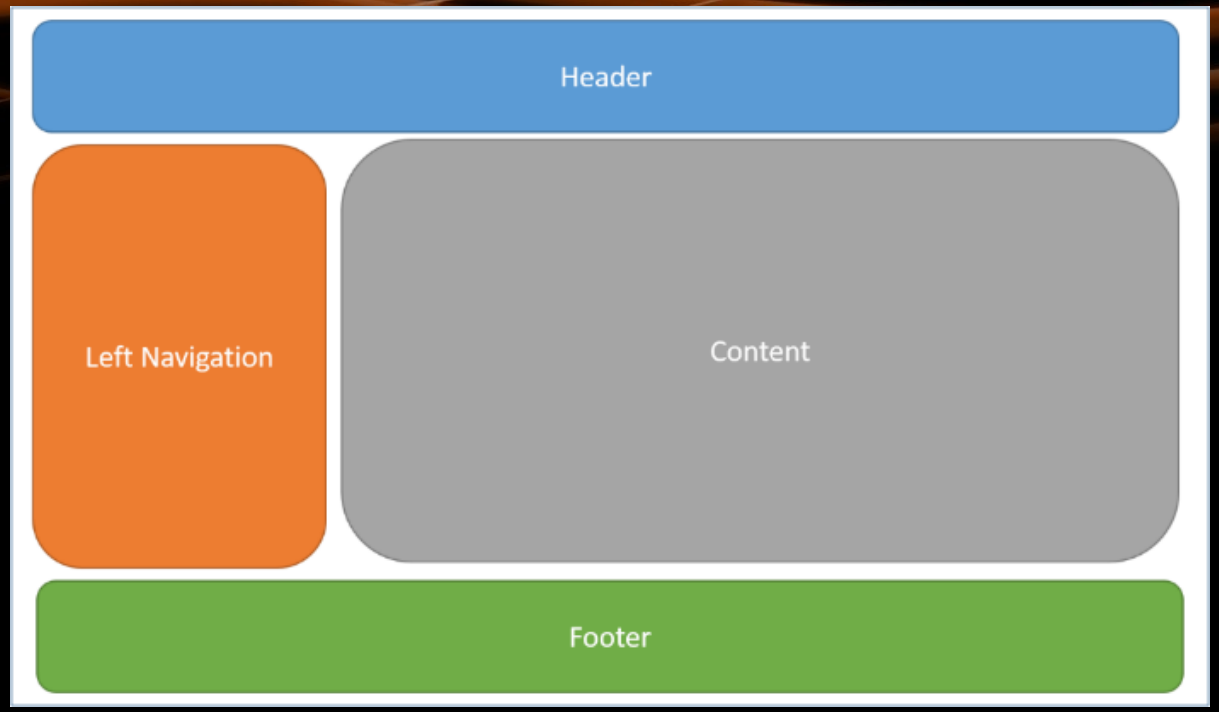
```
public void ConfigureServices(IServiceCollection services)
{
    ...
    services.AddScoped<DataService, DataService>();
    ...
}
```

```
@using WebApp.Services
@inject DataService DataService

<div class="list">
    @foreach(var item in @DataService.GetData())
    {
    <h1>@item</h1>
    }
</div>
```



David
John
Max
George



Файлове за оформление и специални изгледи

Оформление

- Определете общ шаблон на сайта (~/Views/Shared/_Layout.cshtml)
- Двигателят на Razor View визуализира съдържанието навън
- Първо е представен Изгледът, а след това – Оформлението
- @RenderBody () - посочете къде искаме изгледите въз основа на това оформление да „попълнят“ основното им съдържание на това място в HTML

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
</head>
<body>
    <nav>@* Menu *@</nav>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

_ViewStart.cshtml

- Изгледите не е необходимо да посочват оформление, тъй като тяхното оформление по подразбиране е зададено във файла им _ViewStart:
- ~ / Views / _ViewStart.cshtml (Код за всички изгледи)
- Всеки изглед може да посочи страници с персонализирано оформление

```
@{ Layout = "~/Views/Shared/_UncommonLayout.cshtml"; }
```

- Изгледи без оформление:

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

_ViewImports.cshtml

- Ако директива или зависимост е споделена между много изгледи, тя може да бъде зададена глобално в ViewImports:
- ~/Views/_ViewImports.cshtml (код за всички изгледи)

```
@using WebApplication1
@using WebApplication1.Models
@using WebApplication1.Models.AccountViewModels
@using WebApplication1.Models.ManageViewModels
@using Microsoft.AspNetCore.Identity
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

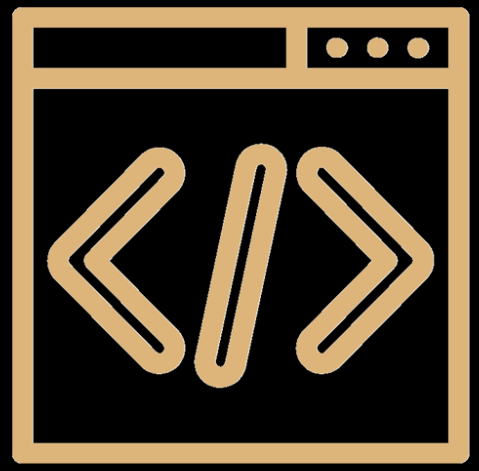
- Този файл не поддържа други функции на Razor

Секции

- Можете да имате един или повече "секции" (незадължително)
- Те са дефинирани в изгледите:

```
@section SideBar {
    <aside>
        Some side information
    </aside>
}
```

- Може да се рендерира навсякъде в страницата за оформление с помощта на метода RenderSection ()
- @RenderSection (име на низ, задължително bool)
- Ако секцията е задължителна и не е дефинирана, ще бъде изхвърлено изключение (IsSectionDefined ())



HTML Helpers и Tag Helpers

HTML Helpers

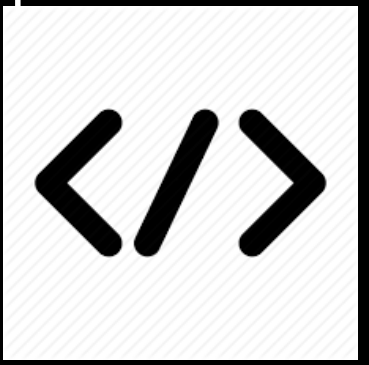
```
@using (Html.BeginForm("Search", "Users",
                       FormMethod.Post))
{
    @Html.TextBox("username")
    <input type="submit" />
}
@Html.Raw(htmlContent)
```

- Всеки изглед наследява RazorPage
  - RazorPage има свойство, наречено Html
- В свойството Html има методи, чрез които връщащият низ може да се използва за:
  - Създайте входове
  - Създайте връзки
  - Създайте формуляри

HTML Helpers	
@Html.ActionLink	@Html.TextBox
@Html.BeginForm	@Html.TextArea
@Html.CheckBox	@Html.Password
@Html.Display	@Html.Hidden
@Html.Editor	@Html.Label
@Html.DropDownList	@Html.Action

Tag Helpers

- Помощниците за маркери позволяват участието на код от страна на сървъра в създаването и изобразяването на HTML елементи в изгледи на Бръснач
  - Има вградени помощници за маркери за много общи задачи
    - Форми, връзки, активи и др.
  - В GitHub repos и NuGet има персонализирани помощници за маркери
- Помощниците за маркери предоставят
  - HTML-приятелски опит за разработка
  - Богата среда на IntelliSense за създаване на маркиране на Razor
  - По-продуктивен, надежден и поддържан код



Tag Helpers срещу HTML Helpers

- Tag Helpers се прикачват към HTML елементи в изглед на Razor
- Tag Helpers намаляват изричните преходи между HTML и C #
- Tag Helpers правят маркирането на Razor доста чисто, а гледките - доста прости

- HTML Helpers се извикват като методи, които генерират съдържание
- HTML Helpers са склонни да включват много C # код в маркирането
- HTML Helpers използват сложен и много специфичен за C # синтаксис за Razorв някои случаи

Tag Helpers срещу HTML 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>
    <div class="form-group">
        <label asp-for="ConfirmPassword" class="col-md-2 control-label"></label>
        <div class="col-md-10">
            <input asp-for="ConfirmPassword" class="form-control" />
            <span asp-validation-for="ConfirmPassword" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <button type="submit" class="btn btn-default">Register</button>
        </div>
    </div>
</form>
```

```
@using (Html.BeginForm("Register", "Account", FormMethod.Post, new { @class = "form-horizo
{
    @Html.AntiForgeryToken()
    <h4>Create a new account.</h4>
    <hr />
    @Html.ValidationSummary("", new { @class = "text-danger" })
    <div class="form-group">
        @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.ConfirmPassword, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.PasswordFor(m => m.ConfirmPassword, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" class="btn btn-default" value="Register" />
        </div>
    </div>
}
```

Създаване на собствен Tag Helper

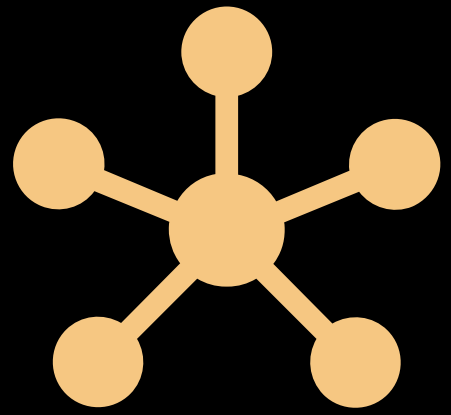
```
[HtmlTargetElement("h1")]
public class HelloTagHelper : TagHelper
{
    private const string MessageFormat = "Hello, {0}";
    public string TargetName { get; set; }

    public override void Process(TagHelperContext context, TagHelperOutput output)
    {
        string formattedMessage = string.Format(MessageFormat, this.TargetName);
        output.Content.SetContent(formattedMessage);
    }
}
```

```
@using WebApplication;
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelper
@addTagHelper WebApplication.TagHelpersHelloTagHelper, WebApplication

<div class="tag-helper-content">
    <h1 target-name="John"></h1>
</div>
```

Hello, John



Частични изгледи и преглед на компоненти

Общи изгледи

- Частични изгледи правят части от страницата
  - Разбийте големи файлове за маркиране на по-малки компоненти
  - Намалете дублирането на общ код на изглед
- Razor Частичните изгледи са нормални изгледи (.cshtml файлове)
  - Обикновено се поставя в / споделено / или в същата директория, където се използва
- Може да се посочи с HTML Helper или Tag Helper
  - Html помощници: Partial, PartialAsync, RenderPartial и т.н.
  - Помощник за етикети: <частично име = "" модел = "" view-data = "" for = "" />

Използване на общи изгледи

- HTML Helper за общи изгледи

```
@using WebApplication.Models;
@model ProductsListViewModel

@foreach (var product in Model.Products)
{
    @await Html.PartialAsync("_ProductPartial", product);
}
```

- Tag Helper за общи изгледи

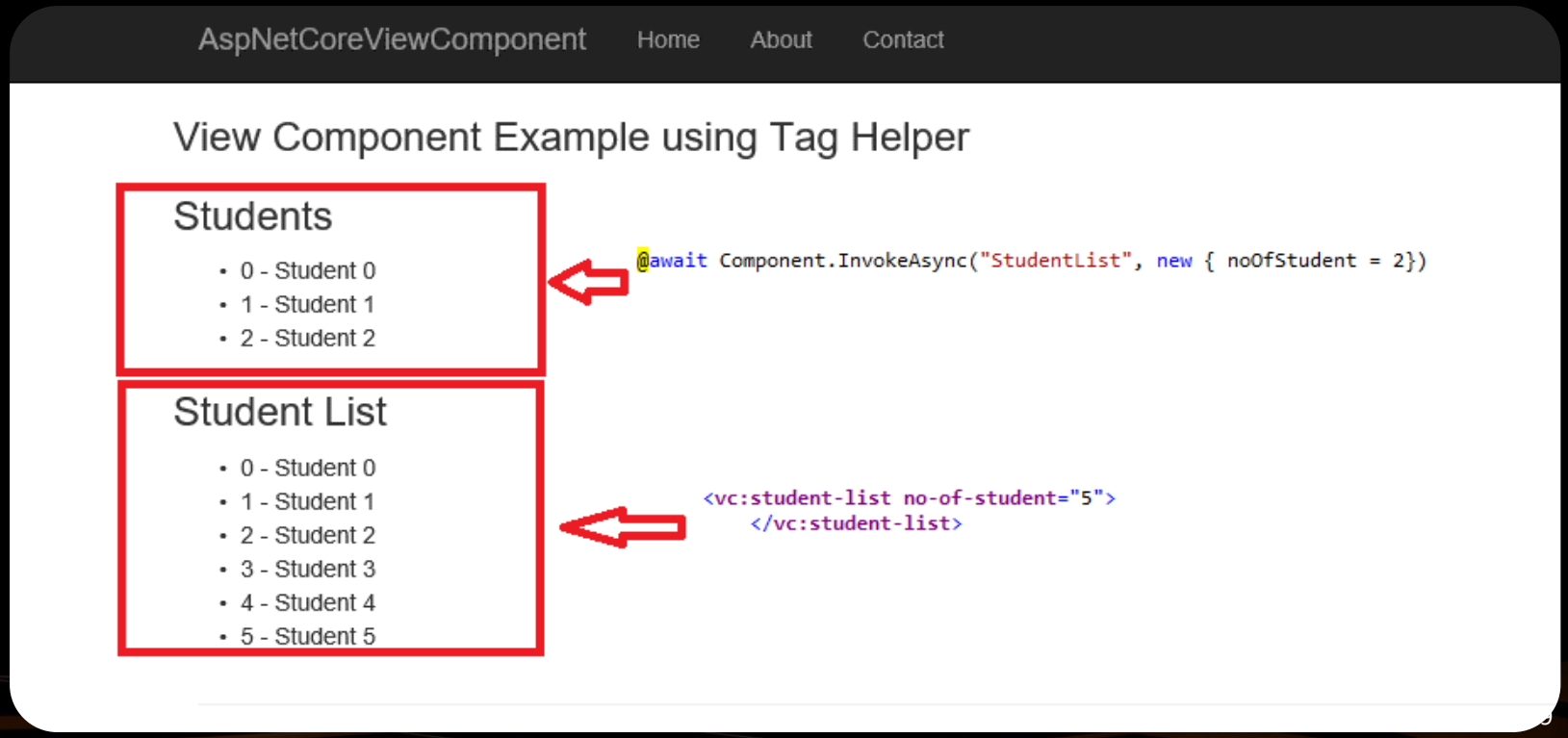
```
@foreach (var product in Model.Products)
{
    <partial name="_ProductPartial" model="product" />
}
```

Изгледи – компоненти

- Компонентите на View са подобни на Partial Views, но много по-мощни
  - Без обвързване на модела
  - Зависи само от данните, предоставени му
- Преглед на компоненти:
  - Представете парче, а не цял отговор (както в Html.Action ())
  - Може да има параметри и бизнес логика
  - Обикновено се извиква от страница за оформление
  - Включва едни и същи предимства на S-o-C и проверка между контролер / изглед

Изгледи – компоненти

- Компонентите на View са предназначени навсякъде, където имате логика за изобразяване за многократна употреба, която е твърде сложна за частичен изглед
  - Dynamic navigation menus
  - Login panels
  - Shopping carts
  - Sidebar content
  - Recently published articles
  - Tag cloud



Изгледи – компоненти

- Преглед на компоненти се състои от 2 части:
  - Клас - обикновено произлиза от ViewComponent
  - Резултат - обикновено изглед
- Преглед на компоненти
  - Определете тяхната логика в метод, наречен InvokeAsync ()
  - Никога не обработвайте директно заявка
  - Обикновено инициализирайте модел, който се предава на изгледа

Създаване на собствен ViewComponent

```
[ViewComponent(Name = "HelloWorld")]
public class HelloWorldViewComponent : ViewComponent
{
    private readonly DataService dataService;
    public HelloWorldViewComponent(DataService dataService)
    {
        this.dataService = dataService;
    }

    public async Task<IViewComponentResult> InvokeAsync(string name)
    {
        string helloMessage =
            await this.dataService.GetHelloAsync();

        this.ViewData["Message"] = helloMessage;
        this.ViewData["Name"] = name;

        return View();
    }
}
```

Създаване на собствен ViewComponent

```
@* In Default.cshtml *@
<h1>@ViewData["Message"]!!! I am @ViewData["Name"]</h1>
```

```
<div class="view-component-content">
    @await Component.InvokeAsync("HelloWorld", new { name = "David" });
    <vc:HelloWorld name="John"></vc:HelloWorld>
</div>
```

Hello World!!! I am David

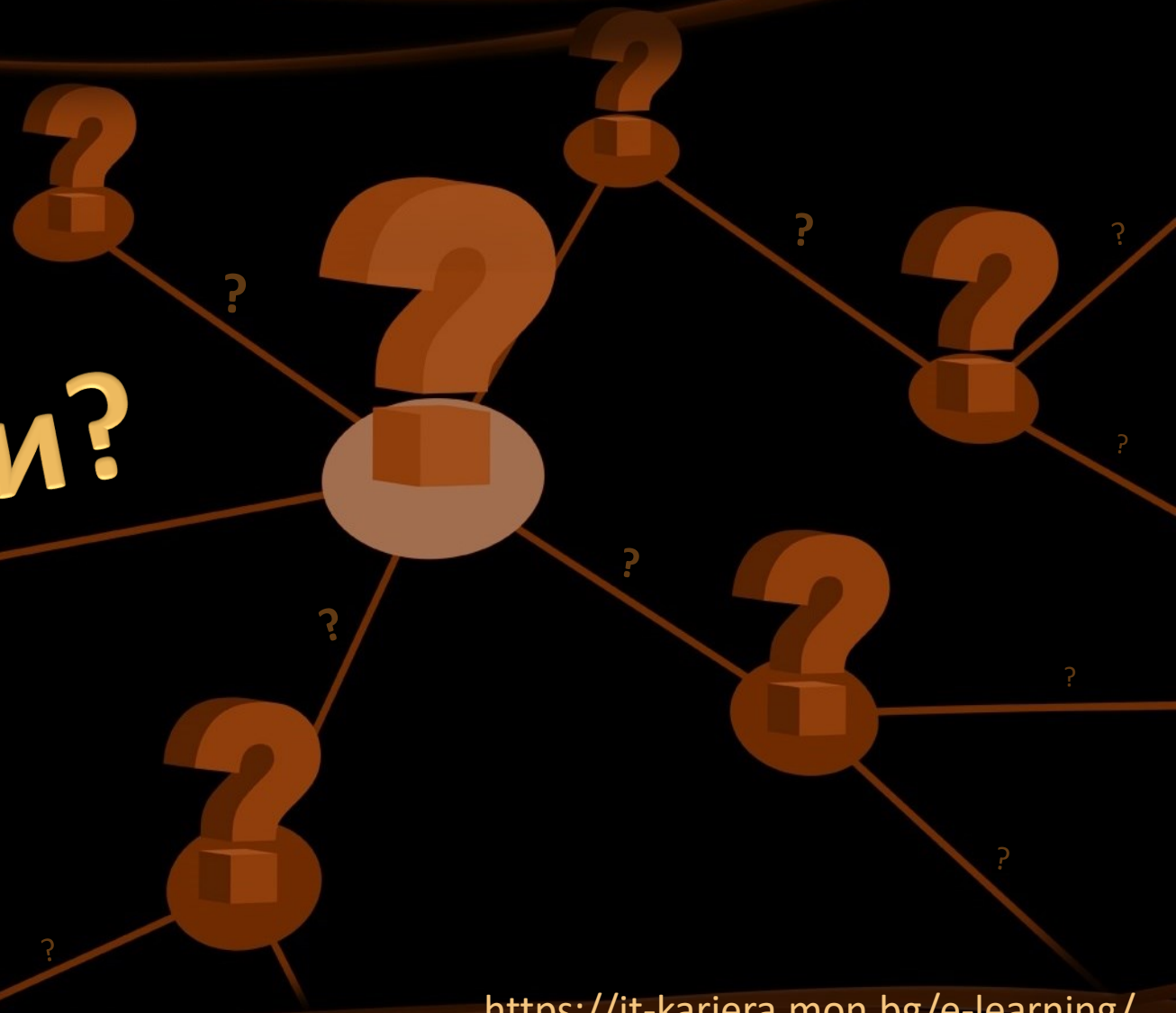
Hello World!!! I am John

Summary

- Основни изгледи
- Razor синтаксис
  - Инжектиране на зависимостта
- Файлове за оформление и специални изгледи
  - _Layout, _ViewStart, _ViewImports
- HTML Helpers & Tag Helpers
- Частични изгледи и преглед на компоненти

Razor изгледи

Въпроси?



https://it-kariera.mon.bg/e-learning/

Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист"

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под свободен лиценз **CC-BY-NC-SA**