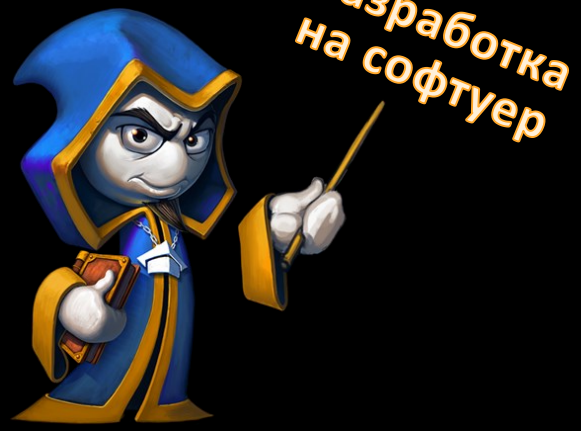
Въведение в ASP.NET Core (MVC)

ASP.NET Core, контролери и действия, маршрутизация, Razor, идентичност



Учителски екип

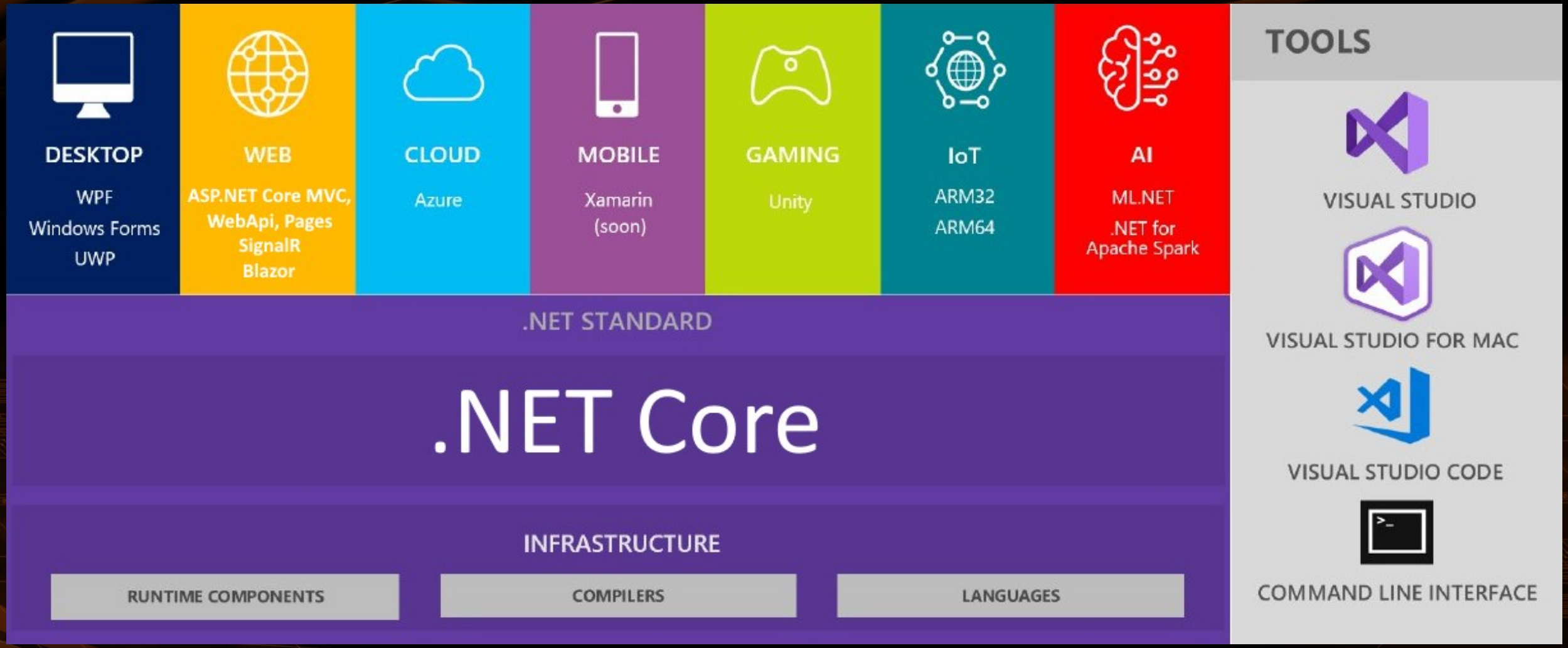
Обучение за ИТ кариера

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

ASP.NET Core

Съдържание

1. Общ преглед на ASP.NET
2. Преглед на ASP.NET Core MVC
3. Създаване на нашите първи ASP.NET Core проекти
4. Контролери и действия
5. Рутиране
6. Статични файлове
7. Razor View Engine
8. ASP.NET Core Identity System


DESKTOP
WPF
Windows Forms
UWP
WEB
ASP.NET Core MVC,
WebApi, Pages
SignalR
Blazor
CLOUD
Azure
MOBILE
Xamarin
(soon)
GAMING
Unity
IoT
ARM32
ARM64
AI
ML.NET
.NET for
Apache Spark
.NET STANDARD
.NET Core
INFRASTRUCTURE
RUNTIME COMPONENTS
COMPILERS
LANGUAGES
TOOLS
VISUAL STUDIO
VISUAL STUDIO FOR MAC
VISUAL STUDIO CODE
COMMAND LINE INTERFACE

Общ преглед на ASP.NET

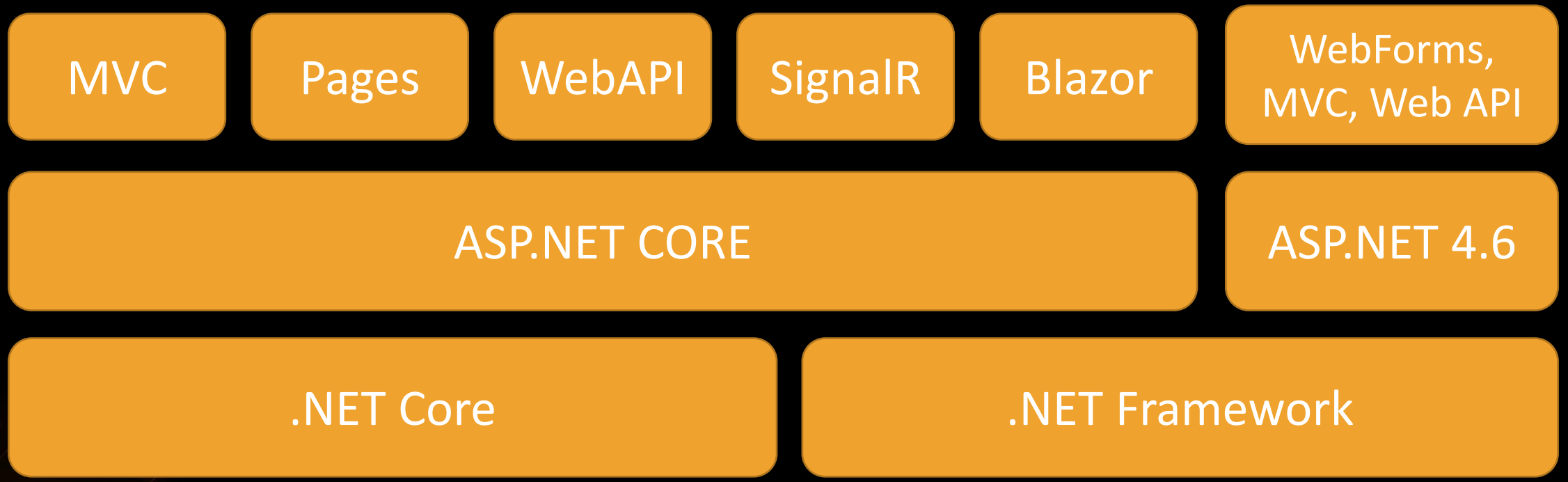
Общ преглед на ASP.NET

- ASP.NET Core е уеб платформа с отворен код
- Можете да изграждате уеб приложения и услуги, IoT приложения, мобилни пакети и всяко уеб-базирано решение с ASP.NET Core
- Има перфектна интеграция с Azure
- Страхотна документация: https://docs.microsoft.com/en-us/aspnet
- ASP.NET Core осигурява:
- Интеграция на съвременни рамки от страна на клиента (Angular, Blazor и др.) И работни процеси (MVC, WebAPI, Razor Pages, SignalR)
- Приложенията ASP.NET Core стартират както в .NET Core, така и .NET Framework

ASP.NET Core главни плюсове

- Унифицирана рамка за изграждане на уеб потребителски интерфейс и уеб API, архитектирана с възможност за проверка
- Възможност за разработване и изпълнение на Windows, macOS и Linux
- Възможност за хостване на IIS, Nginx, Apache, Docker или самостоятелно хостване във вашия собствен процес
- Вградена инжекция за зависимост
- Лек, високопроизводителен и модулен HTTP тръбопровод за заявка (средни софтуерни програми)
- Razor Pages е основан на страници модел за програмиране, който улеснява изграждането на уеб интерфейс
- Blazor ви позволява да използвате C # в браузъра и да споделяте логиката на приложението от страна на сървъра и от страна на клиента
- Версия на едновременно до приложение

ASP.NET Core


MVC
Pages
WebAPI
SignalR
Blazor
WebForms, MVC, Web API
ASP.NET CORE
ASP.NET 4.6
.NET Core
.NET Framework

ASP.NET vs ASP.NET Core

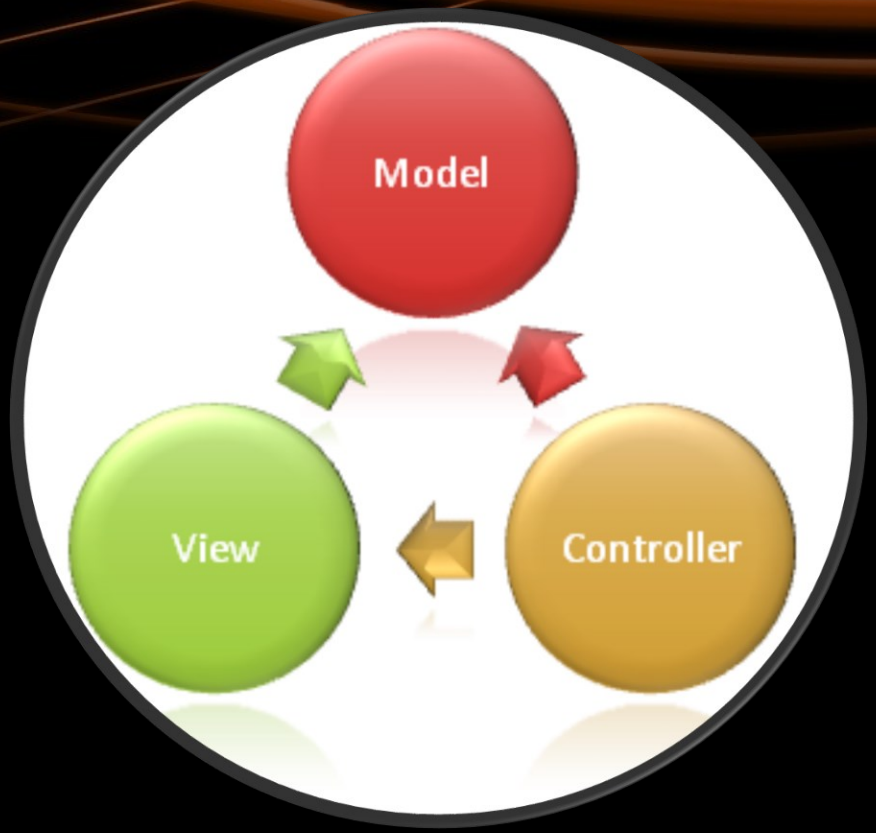
- ASP.NET е зряла рамка (v1.0 издаден януари 2002 г.)
- Предоставя услугите, необходими за изграждане на корпоративни уеб приложения, базирани на сървър, на Windows
- ASP.NET MVC работи само на Windows и само в IIS
- Последна версия: 5.2.5

- ASP.NET Core е сравнително нова рамка с отворен код (v1.0 издаден юни 2016)
- Междуплатформена рамка
- Подходящ за изграждане на модерни, облачни базирани уеб приложения на Windows, macOS или Linux
- Последна версия: 3.0
- 3.1 в развитие

ASP.NET vs ASP.NET Core (2)

- Една версия на машина
- System.Web.dll
- Всичко е включено по подразбиране
- HTTP модули, HTTP манипулатори, Global.asax
- MVC + Web API + уеб страници
- Действия с деца (Html.Render)
- Web.config

- Kestrel, Windows, Mac, Linux
- Няколко версии на машина
- Всичко е Nuget пакети
- Мидълуер. Всичко в едно. По-бързо.
- Променливи JSON и среда
- Преглед на компоненти, помощници на маркери
- Вграден DI, регистрация, услуги, доставчици на файлове, WebSockets



Общ преглед на ASP.NET Core MVC

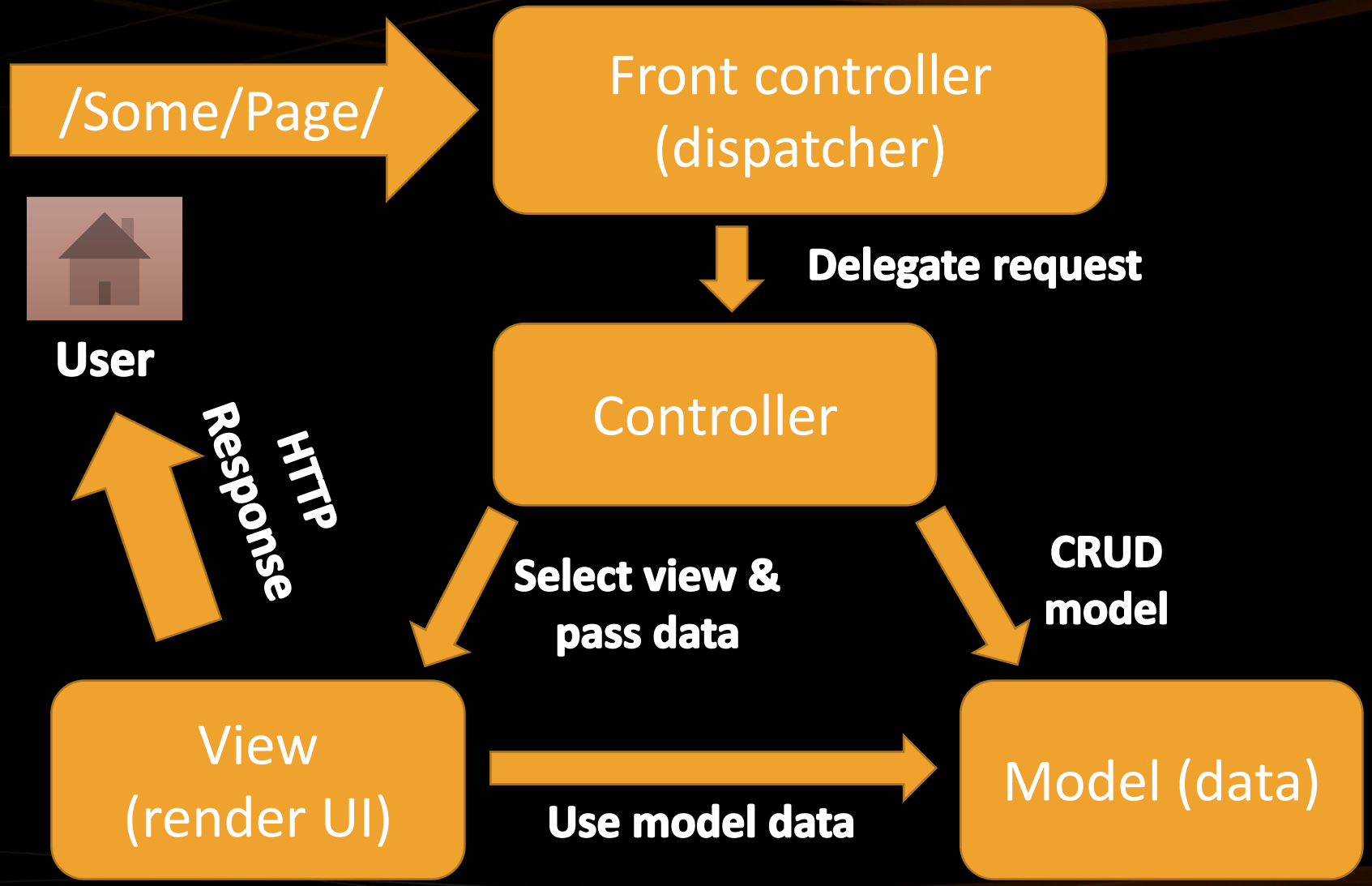
ASP.NET Core MVC

- ASP.NET Core MVC предоставя функции за изграждане на уеб API и уеб приложения
- Използва модела на дизайна на модела View-Controller (MVC)
- Лек, с отворен код, тестируем, добър инструментариум
- RESTful услуги с ASP.NET Core Web API
- Вградена поддръжка за множество формати на данни, договаряне на съдържание и CORS
- Постигнете висококачествен архитектурен дизайн, оптимизирайки работата на разработчиците
- Конвенция за конфигуриране
- Обвързването на модела автоматично картографира данни от HTTP заявки
- Проверка на модела с валидиране от страна на клиента и от страна на сървъра

ASP.NET Core MVC (2)

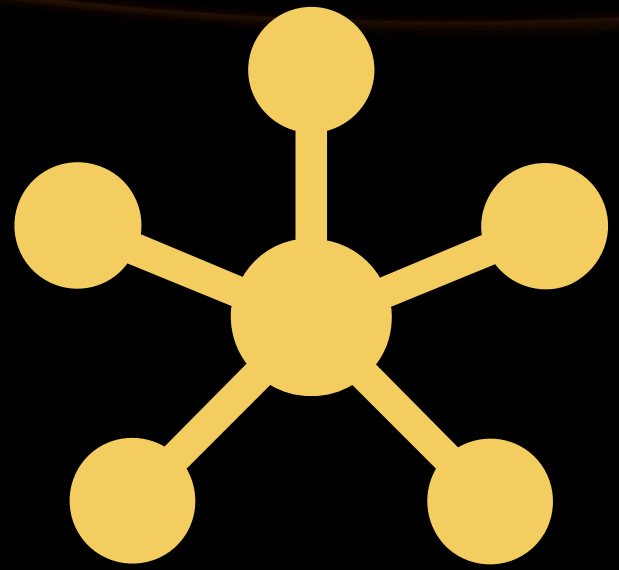
- Заедно с тези ASP.NET Core MVC предоставя функции като:
- Routing
- Инжектиране на зависимостта - DI
- Силно типизирани изгледи с двигателя Razor
- Помощниците на маркери активират код от страна на сървъра в HTML елементи
- Частични изгледи и преглед на компоненти
- Филтри, области, Middlewares
- Вградени функции за защита
- Идентичност с потребители, роли и външни доставчици
- И много други…

MVC шаблона за уеб среда



Hello, ASP.NET Core

Създайте първият си ASP.NET Core проект



Контролери и действия

Контролери

- Основният компонент на MVC модела
- Всички контролери трябва да са на разположение в името на папката Контролери
- Стандартът за именуване на контролера трябва да бъде {name} Контролер (конвенция)
- Всеки контролер трябва да наследи класа Controller
- Достъп до заявка, отговор, HttpContext, RouteData, TempData, ModelState, потребител, ViewBag / ViewData и т.н.
- Маршрутите избират Контролери при всяка заявка
- Всички заявки са картографирани към конкретно действие

Действия

- Действията са крайната дестинация за заявка
- Публични методи за контролер
- Не-статичен
- Без ограничения на връщащата стойност
- Действията обикновено връщат IActionResult

```
public IActionResult Details(int id)
{
    var viewModel = this.dataService.GetById(id).To<DetailsViewModel>();
    return this.View();
}
```

Резултат от действия

- Отговор на действие на контролер на заявка на браузър
- Представете различни кодове за състоянието на HTTP
- Наследи от базовия клас ActionResult

```
public IActionResult Index()
{
    return Json(this.dataService.GetData());
}
```

```
public IActionResult GetFile()
{
    ...

    return File(fileStream, mimeType, fileName);
}
```

```
private const string AppVersion = "v.1.0.0";

public IActionResult Version()
{
    return Content(AppVersion);
}
```

```
public IActionResult LoginConfirm(string username,
                                  string password)
{
    ...

    return Redirect("/Home/index");
}
```

Резултат от действия (2)

Име	Рамково поведение	Помощен метод
StatusCodeResult	Връща HTTP Резултат на отговор с даден статус	**StatusCode() / Ok() BadRequest() / NotFound()**
JsonResult	Връща данни във формат JSON	**Json()**
RedirectResult	Redirects the client to a new URL	**Redirect() / RedirectPermanent()**
RedirectToRouteResult	Пренасочва клиента към нов URL адрес	**RedirectToRoute() / RedirectToAction()**
ViewResult PartialViewResult	Отговорът е отговорност на двигател с оглед	**View() / PartialView()**
ContentResult	Връща низовия буквал	**Content()**
EmptyResult	Без отговор, без заглавие от тип съдържание	
FileContentResult FilePathResult FileStreamResult	Върнете съдържанието на файл	**File() / PhysicalFile()**

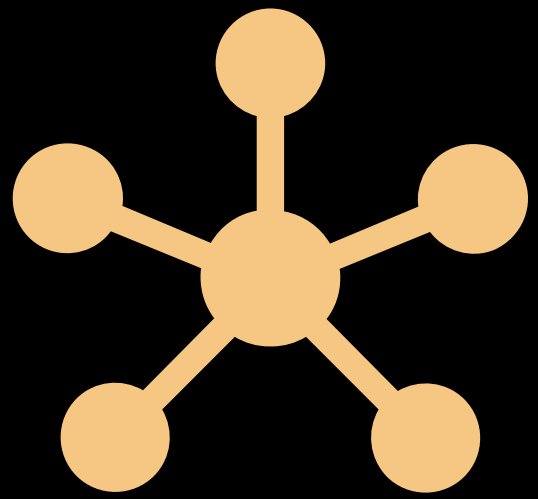
Параметри за действията

- ASP.NET Core обръща данните от HTTP заявката към параметрите на действие по няколко начина:
  - Маршрутизаторът може да предава параметри на действия
  - HTTP: // Localhost / Потребители / Niki
  - Модел на маршрутизация: Потребители / {потребителско име}
  - URL низът на заявката може да съдържа параметри
    - / Потребители / ByUsername? Потребителско име = NikolayIT
  - Данните за HTTP публикации също могат да съдържат параметри

Селектори за действията

- ActionName(string name)
- AcceptVerbs
  - HttpPost
  - HttpGet
  - HttpDelete
  - HttpOptions
  - …
- NonAction
- RequireHttps

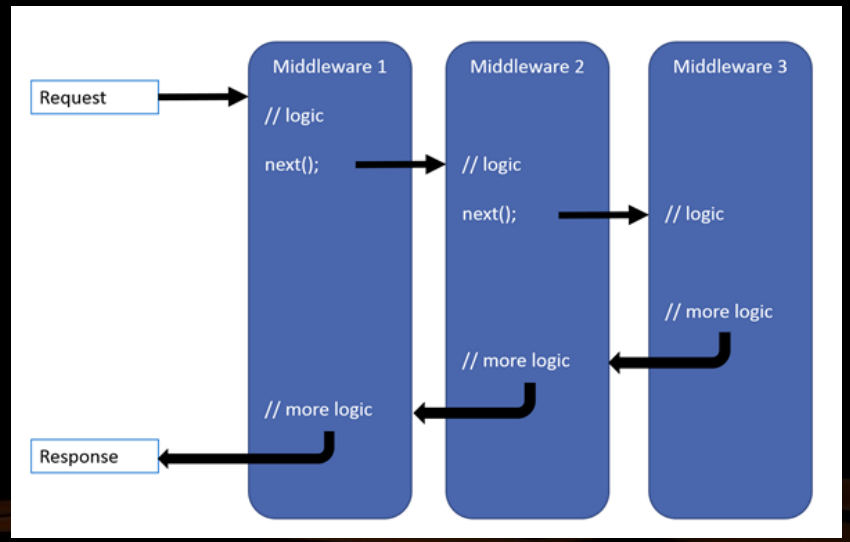
```
public class UserController : Controller
{
    [ActionName("UserLogin")]
    [HttpPost]
    [RequireHttps]
    public IActionResult Login(string username,
                               string password)
    {
        return Content("Logged In!!!");
    }
}
```



ASP.NET Core MVC рутиране

ASP.NET Core MVC Рутиране

- ASP.NET Core MVC използва middleware за маршрутизиране при клиентски заявки.
  - Маршрутите описват как пътищата на URL адреса на заявката трябва да бъдат обърнаати към действия на контролер.
  - Има 2 вида маршрутизиране на действие
    - Конвенционален
    - Атрибут



Конвенционален

- Използване на конвенционална маршрутизация с маршрута по подразбиране:
  - Оптимизира приложение, като предотвратява създаването на нов шаблон на URL адрес за всяко действие.
  - Гарантира съгласуваност на URL адресите в приложения в стил CRUD.
  - Опростява кода и прави потребителския интерфейс по-предсказуем.
- Може да се реализира така:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseMvcWithDefaultRoute();
}
```

Ограничения при Рутиране

- Ограниченията на маршрута са правила за URL сегментите
- Всички ограничения са редовен израз, съвместим с класа Regex
- Определен като един от параметрите на маршрутите.MapRoute (…)

```
routes.MapRoute(
    name: "blog",
    template: "{year}/{month}/{day}",
    defaults: new { controller = "Blog", action = "ByDate" },
    constraints: new { year = @"\d{4}", month = @"\d{1,2}", day =
@"\d{1,2}", });
```

Атрибут

- Рутирането на атрибутите използва набор от атрибути, за да обръщат действията директно към шаблоните на маршрута.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseMvc();
}
```

```
public class HomeController : Controller
{
    [Route("/")]
    public IActionResult Index()
    {
        return View();
    }
}
```

Атрибут (2)

- Рутирането на атрибутите също може директно да дефинира метода на заявка.

```
public class HomeController :
Controller
{
    //...

    [HttpGet("/")]
    public IActionResult Index()
    {
        return View();
    }
}
```

```
public class UsersController :
Controller
{
    //...

    [HttpPost("Login")]
    public IActionResult Login()
    {
        return View();
    }
}
```

- Атрибутите Http {action} често се използват в REST API.

Атрибут (2)

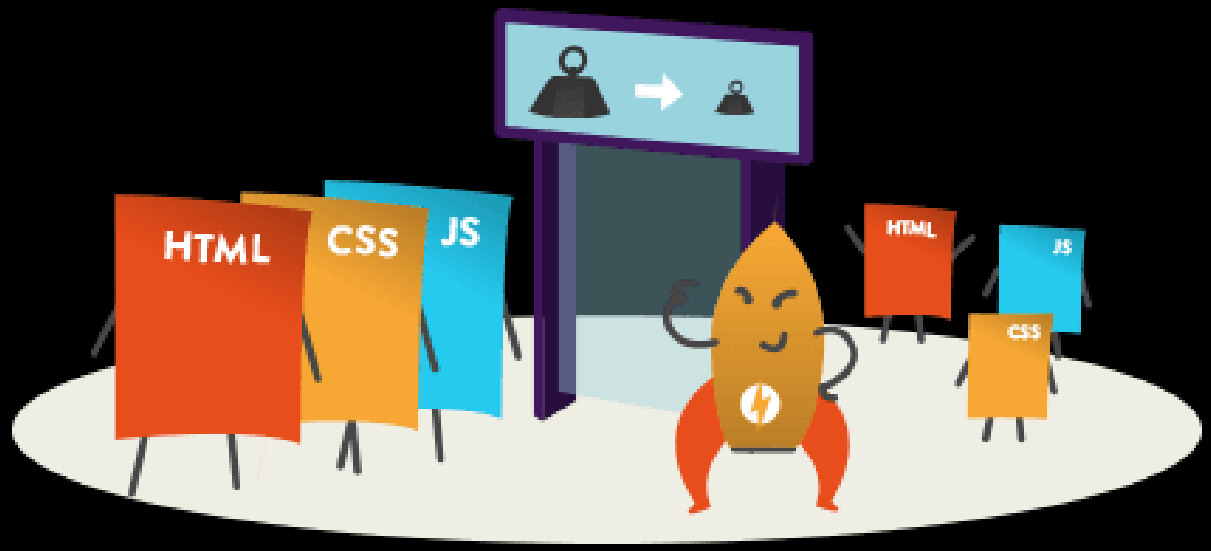
- Рутирането на атрибути ви позволява да създавате няколко маршрута за едно действие.
- Той също така ви позволява да комбинирате маршрут за контролер и маршрут за действие.

```
public class HomeController :
Controller
{
    //...

    [Route("/")]
    [Route("Index")]
    public IActionResult Index()
    {
        return View();
    }
}
```

```
[Route("Home")]
public class HomeController : Controller
{
    //...

    [Route("/")] // Does not combine, Route – /
    [Route("Index")] // Route - /Home/Index
    [Route("")] // Route - /Home
    public IActionResult Index()
    {
        return View();
    }
}
```



Рутиране на статични файлове

Статични файлове

- Статичните файлове са необходимост за работа на уеб приложение.
- Файлове като HTML, CSS, JS и различни активи могат да се обслужват директно на клиенти с ASP.NET Core

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseStaticFiles();
}
```

- Това ще каже на ASP.NET Core App да обслужва статичните файлове в директорията wwwroot.

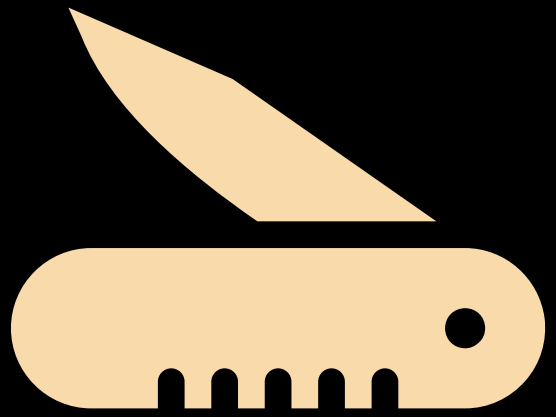
Статични файлове (2)

- Може да се модифицира, за да служи на други папки.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseStaticFiles(); // For the wwwroot folder

    app.UseStaticFiles(new StaticFileOptions()
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(), "OtherFiles")),
        RequestPath = new PathString("/files")
    });
}
```

- Това ще обслужва файла „style.css“ при поискване „http: // {app} /files/style.css“ от „OtherFiles“ вместо „wwwroot“

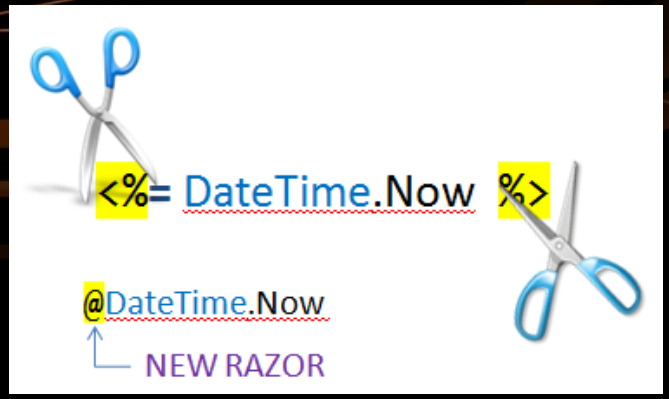


Razor View

Изгледи (Views)

- HTML шаблони на приложението
- На разположение много двигатели за оглед
  - Прегледът на двигателите изпълнява код и предоставя HTML
  - Осигурете много помощници за лесно генериране на HTML
  - Най-популярният е Razor View Engine
- Можем да предаваме данни на изгледи чрез:
  - ViewBag, ViewData и Model (силно типизирани изгледи)

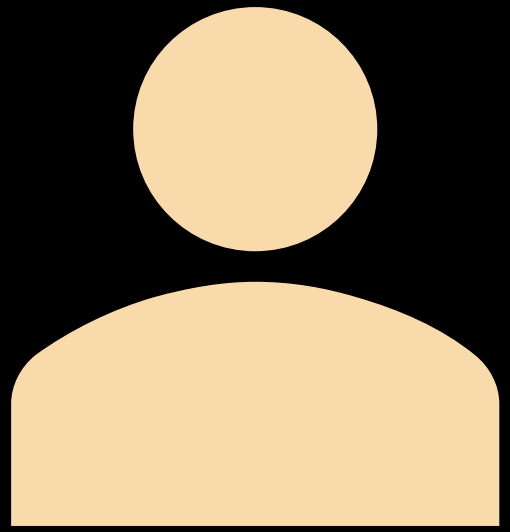
Razor



- Синтаксис за маркиране на шаблона
- Проста синтаксична машина
- Въз основа на езика за програмиране на C #
- Разрешава на програмиста да използва работен поток за изграждане на HTML
- Подход за шаблониране, фокусиран върху кода, с минимален преход между HTML и код
- Синтаксисът на Razor стартира кодови блокове с символ @ и не изисква изрично затваряне на кодовия блок

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;



ASP.NET Core Identity System

ASP.NET Identity

- Системата ASP.NET Core Identity
  - Система за удостоверяване и упълномощаване за ASP.NET Core Web приложения
  - Поддържа ASP.NET Core MVC, Web API, SignalR, Blazor
  - Работи с потребители, потребителски профили, влизане / излизане, роли и т.н.
  - Поддържа потребителските акаунти в локалната БД или във външния магазин за данни
  - Включено чрез междинен софтуер:

```
app.UseAuthentication();
```

Обобщение

- ASP.NET Core е чудесна платформа за разработване на уеб приложения
- Добре проектиран, лесно разтегаем, високо тестван
- Има голяма (и нарастваща) общност
- Построен от земята нагоре
- Но прави развитието по-лесно за наследените разработчици на ASP.NET

Въведение в ASP.NET Core (MVC)

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**