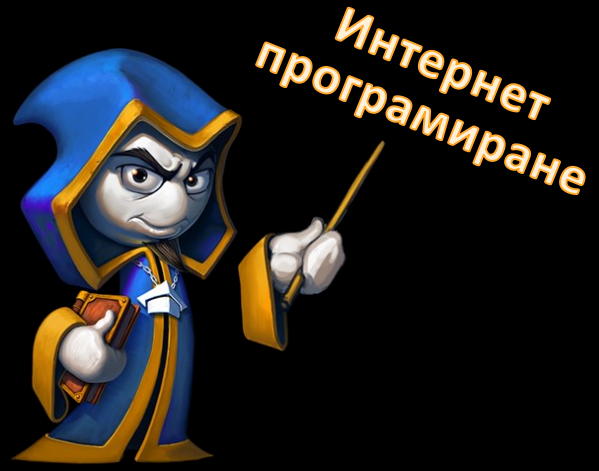
Създаване на REST API

ASP.NET Web API



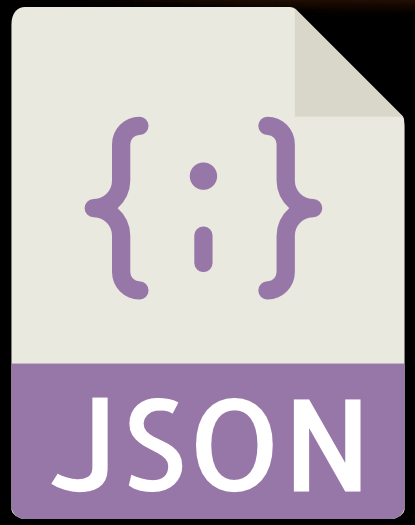
Учителски екип

Обучение за ИТ кариера

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

Съдържание

1. JSON
2. XML
3. Web API



JSON

JSON

- JavaScript Object Notation (JSON) е файлов формат с отворен стандарт
  - Използва четим от човека текст за предаване на обекти с данни
  - Обектите на данни се състоят от двойки атрибут-стойност или типове данни от масив
  - Лесно за хората да четат и пишат
  - Лесно за машините да обработват и генерират
- JSON произлиза от JavaScript
  - Независим от езика
  - Сега много езици предоставят код за генериране и обработване на JSON

JSON

- JSON е много често използван формат на данни, използван в уеб комуникацията
  - Основно в комуникация браузър-сървър или сървър-сървър
  - Официалният тип интернет медия (MIME) за JSON е application/json
  - JSON файловете имат разширение .json
- JSON обикновено се използва като заместител на XML в AJAX
  - По-кратък и лесен за разбиране
  - По-бърз за четене и писане и е по-интуитивен
  - Не поддържа схеми и пространства от имена

JSON – Пример

```
{
    "firstName": "Pesho",
    "courses": ["C#", "JS", "ASP.NET"]
    "age": 23,
    "hasDriverLicense": true
}
```

XML

XML

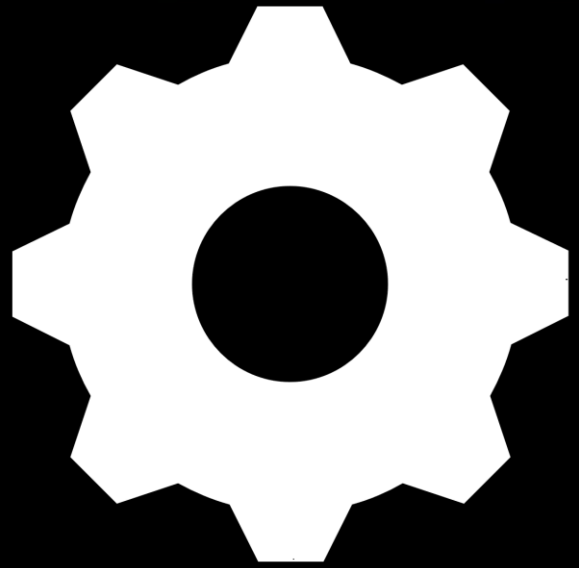
- XML дефинира набор от правила за кодиране на документи
  - Идва от Extensible Markup Language
  - Подобен като JSON
    - По отношение на читаемостта от човека и обработката от машини
    - По отношение на йерархия (стойности в стойности)
- XML е текстов формат
  - Силна поддръжка за различни човешки езици чрез Unicode
  - Дизайнът се фокусира силно върху действителните документи

XML

- Има 2 типа MIME за XML - application/xml и text/xml
- .xml разширение
- Има много приложения:
  - Широко използван в SOA
  - Конфигуриране на .NET приложения
  - Използва се във формати на Microsoft Office
  - XHTML е трябвало да бъде строг HTML формат

XML – Пример

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```



ASP.NET Core Web API

API

- API е интерфейс за програмиране на приложения
  - Използван от Web Browser (SPA), Mobile Applications, Games, Desktop Applications, Web Server
- Състои се от публично изложени крайни точки
  - Крайните точки съответстват на дефинирана система за заявка-отговор
  - Комуникацията обикновено се изразява във формат JSON или XML
  - Комуникацията обикновено се осъществява чрез уеб протокол
    - Най-често HTTP - чрез уеб сървър, базиран на HTTP

ASP.NET Core Web API

- Няма нищо различно от уеб приложение
- Вие изграждате контролери с действия
- В този случай обаче действията са в ролята на крайни точки
- Контролерите трябва да се анотират с ApiController

```
[Route("api/[controller]")
[ApiController]
public class ProductsController
: ControllerBase
{
    ...
}
```

Път, използван за достъп до крайни точки от този ApiController

```
[assembly: ApiController]
namespace Demo.Api
{
    public class Startup
    {
        ...
    }
}
```

ASP.NET Core Web API Controller

- Наследяваме Controller
- Трябва да анотираме класа с атрибутите [ApiController] и [Route]

```
[Route("api/[controller]")]
[ApiController]
public class ProductController : Controller
{
    private readonly IProductService productService;

    public ProductController(IProductService ps)
    {
        this.productService = ps;
    }
}
```

ASP.NET Core Web API (ApiController)

- Анотацията [ApiController] предоставя удобни функции
  - Автоматични HTTP 400 отговор (за грешки в състоянието на модела)
  - Обвързване на изходния параметър на източника
  - Изисквания за Атрибутно рутиране
  - Подробни отговори за кодове за състояние на грешка

```
{
    type: "https://tools.ietf.org/html/rfc7231#section-6.5.4",
    title: "Not Found",
    status: 404,
    traceId: "0HLHLV31KRN83:00000001"
}
```

ASP.NET Core Web API (ApiController)

- Автоматични HTTP 400 отговори
  - Грешките при валидиране на модела автоматично задействат HTTP 400 отговор

```
if (!ModelState.IsValid) return BadRequest(ModelState);
```

Не е необходимо

- Обвързване на атрибути на източника
  - Атрибутите определят местоположението на стойността на параметъра

[FromBody]
[FromForm]
[FromHeader]
[FromQuery]
[FromRoute]
[FromServices]

Пример

```
[HttpPost]
public IActionResult Create(
    Product product, // [FromBody] се подразбира
    string name) // [FromQuery] се подразбира
{ … }
```

ASP.NET Core Web API (ApiController)

- Multipart / Form-data заявката се подразбира
  - Постига се чрез поставяне на атрибута [FromForm] върху параметрите на действието
  - multipart/form-data типа на съдържанието на заявката се подразбира
- Рутирането на атрибутите се превръща в изискване

```
[Route("api/[controller]")
[ApiController]
public class ProductsController : ControllerBase
```

- Крайните точки са недостъпни по пътищата, определени от :
  - UseMvc() и UseMvcWithDefaultRoute()

ASP.NET Core Web API (ApiController)

- Отговори за подробности за проблема за кодове за състояние на грешка
  - От ASP.NET Core 2.2 MVC преобразува резултатите от грешки
  - Грешките се трансформират в ProblemDetails
    - Тип, базиран на HTTP Api за представяне на грешки
    - Стандартизиран формат за машинно четими данни за грешки

```
if (product == null)
{
    return NotFound();
}
```

```
{
    type: "https://tools.ietf.org/html/rfc7231#section-6.5.4",
    title: "Not Found",
    status: 404,
    traceId: "0HLHLV31KRN83:00000001"
}
```

ASP.NET Core Web API (ApiController)

- Тези функции са вградени и активни по подразбиране
  - Поведението по подразбиране може да бъде презаписано

```
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_2)
    .ConfigureApiBehaviorOptions(o =>
    {
      o.SuppressConsumesConstraintForFormFileParameters = true;
      o.SuppressInferBindingSourcesForParameters = true;
      o.SuppressModelStateInvalidFilter = true;
      o.SuppressMapClientErrors = true;
      o.SuppressUseValidationProblemDetailsForInvalidModelStateResponses = true;
    });
```

ASP.NET Core Web API (Return Types)

- ASP.NET Core предлага няколко опции за типове връщане на API Endpoint
  - Специфичен тип
    - Най-простият тип действие
  - IActionResult тип
    - Подходящо, когато са възможни няколко типа ActionResult в съответното действие

```
[HttpGet]
public IEnumerable<Product> Get()
{
    return this.productService.GetAllProducts();
}
```

```
[HttpGet("{id}")]
[ProducesResponseType(200, Type = typeof(Product))]
[ProducesResponseType(404)]
public IActionResult GetById(int id)
{
    var product = this.productService.GetById(id);

    if (product == null) return NotFound();

    return Ok(product);
}
```

ASP.NET Core Web API (Return Types)

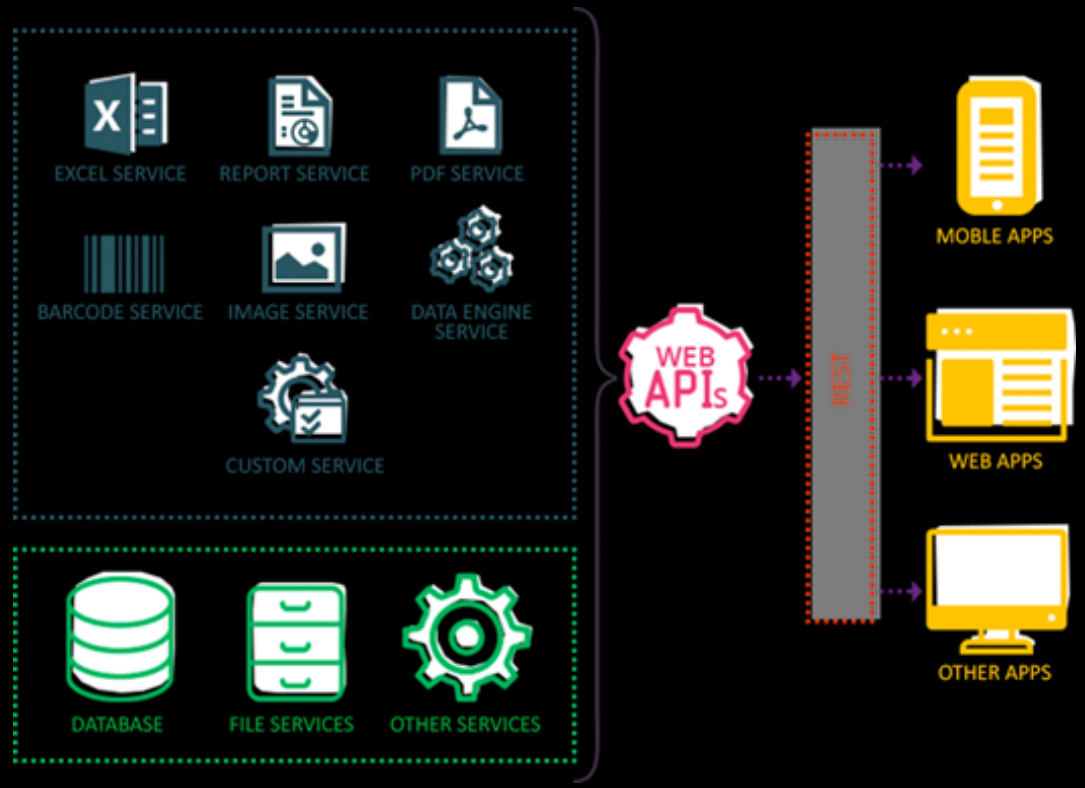
- Препоръчва се използването на ActionResult <T>

```
[HttpGet]
public ActionResult<IEnumerable<Product>> Get()
{
    return this.productService.GetAllProducts();
}
```

```
[HttpGet("{id}")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
public ActionResult<Product> GetById(int id)
{
    var product = this.productService.GetById(id);

    if (product == null) return NotFound();

    return product;
}
```



ASP.NET Core Web API (GET Методи)

- Създаване на уеб API с един контролер

```
[HttpGet]
public ActionResult<IEnumerable<Product>> GetProducts()
  => this.productService.GetAllProducts();

[HttpGet("{id}")]
public ActionResult<Product> GetProduct(long id)
{
    var product = this.productService.GetById(id);
    if (product == null) return NotFound();
    return product;
}
```

ASP.NET Core Web API (POST Методи)

- Създаване на уеб API с един контролер

```
[HttpPost]
public ActionResult<Product> PostProduct(ProductBindingModel pm)
{
    this.productService.RegisterProduct(pm);

    return CreatedAtAction("GetProduct", new { id = pm.Id }, pm);
}
```

- Методът CreatedAtAction:
  - Връща 201 (Created) отговор - стандарт за POST заявки
  - Добавя Location хедър към отговора
  - Използва път с име "GetProduct", за създаване на URL

ASP.NET Core Web API (PUT Методи)

- Създаване на уеб API с един контролер

```
[HttpPut("{id}")]
public IActionResult PutProduct(long id, ProductBindingModel pm)
{
    if (id != pm.Id) return BadRequest();
    this.productService.EditProduct(id, pm);
    return NoContent();
}
```

  - Подобно на PostProduct, но използва HTTP PUT
  - Отговорът е 204 (No Content)
  - HTTP PUT изисква цяла актуализация на записа

ASP.NET Core Web API (DELETE Методи)

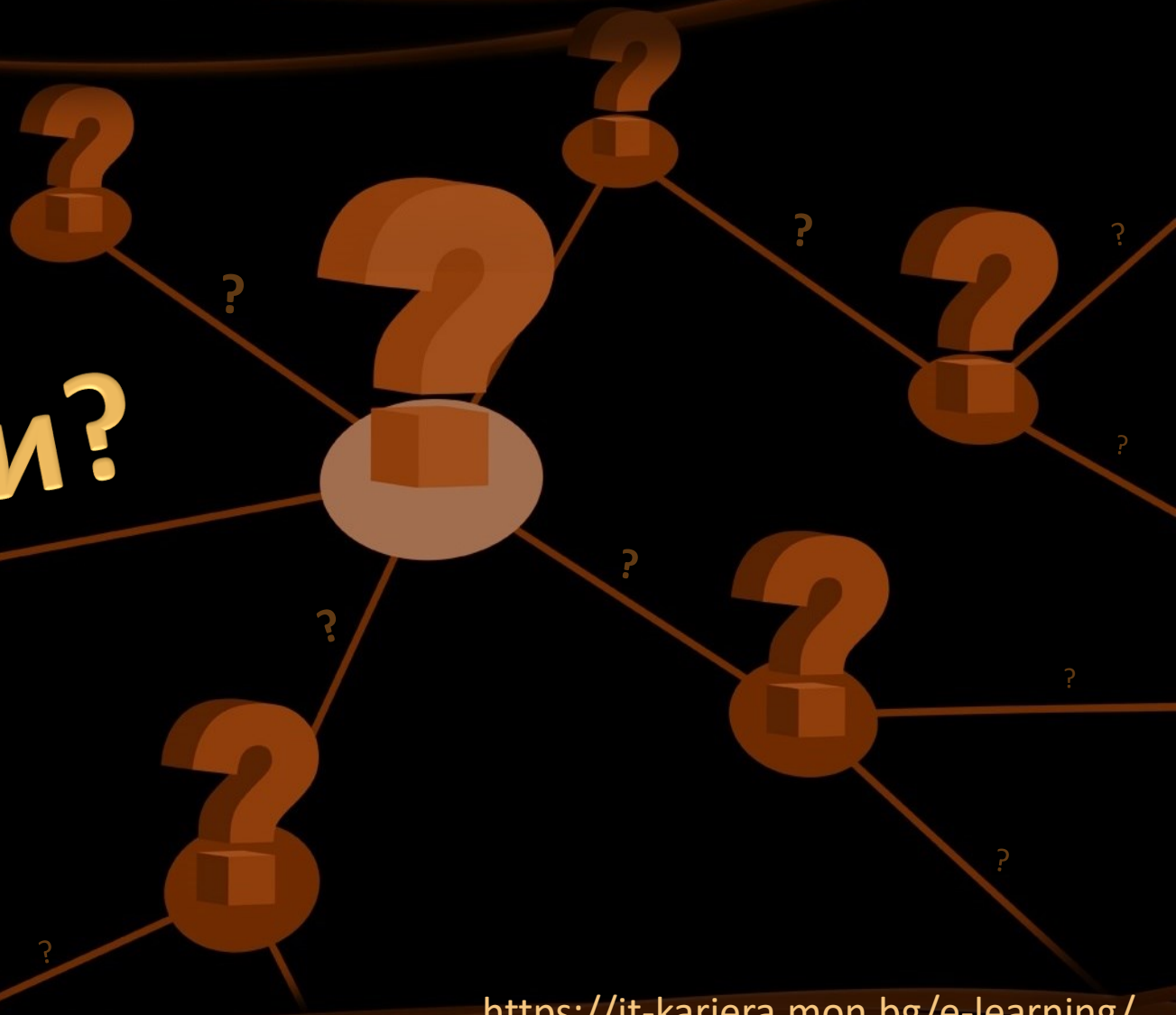
- Създаване на уеб API с един контролер

```
[HttpDelete("{id}")]
public ActionResult<Product> DeleteProduct(long id)
{
    var product = this.productService.DeleteProduct(id);
    if (product == null) return NotFound();
    return product;
}
```

  - Отговорът е 204 (No Content)
  - И с това ние имаме нашия Products Web API
  - Сега нека да тестваме крайните точки

Създаване на REST API

Въпроси?



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**