

# Jazyk C# 2

10. seminář

Radek Janošík

Univerzita Palackého v Olomouci

26. 4. 2024

# Reakce na úkoly

- Některá řešení pěkná
- Jiné pouze „nutné minimum“ (či méně)
- „Ostré vyhledávání“ – lepší použít `.Contains()` nebo alespoň `.StartsWith()`
  - ▶ Případně ještě na oba řetězce zavolat `.ToLower()`
- Naplnění daty
- Umístění akce do kontroleru

# ASP.NET – WebApi

- Web dnes:
  - ▶ Spousta funkcionality je ve frontendu
  - ▶ Backend často slouží jen jako „hloupý“ manipulátor s daty
  - ▶ Často stejný backend pro JS, Mobilní aplikace, Propojené služby,...
- ⇒ REST API
- API využívající HTTP pro GET, PUT, POST, DELETE dat
- Tedy pošleme HTTP request a backend nám vrátí výsledek
- Formát odpovědi – variabilní (XML, JSON)

# ASP.NET – WebApi (1/2)

- Vytvoření projektu podobné jako MVC (MVC se používá)
- Vybereme ASP.NET Core Web API
- Vygeneruje se podobný projekt jako minule
- Zůstaly nám však jen kontrolery
- V šabloně je funkční příklad a navíc Swagger (OpenAPI)

## ASP.NET – WebApi (2/2) – Swagger

- Při spuštění se nám vygenerují webové stránky, které ukazují definici API
- U každé akce máme popsání parametry, můžeme ji vyzkoušet
- Při změně kontroleru se přegeneruje
- Odpověď poté podle hlavičky požadavku
- .NET se postará o serializaci

# WebApi – Studenti (1/2)

- Napojíme se na DB stejně jako minule:
  - ▶ Vytvoříme BaseController.cs (nyní však dědí z ControllerBase)
  - ▶ Vytvoříme si v něm odkaz na kontext a z něj budeme vytahovat data
  - ▶ Přidáme anotace

```
1 [ApiController]
2 [Route (" [controller] / [action] " ) ]
```

- DTO (Data Transfer) Objekty
  - ▶ Nebudeme „do světa“ posílat přímo naše DB objekty
  - ▶ Utajení některých vlastností
  - ▶ „Zplacatění“ strukturovaných data
  - ▶ Externí reprezentace – bezpečnější, mezivrstva
- Filtrování musíme řešit „ručně“

## WebApi – Studenti (2/2) – Endpointy

- V novém `StudentController` vytvoříme endpointy
  - ▶ `public IEnumerable<Student> Get ()` – seznam všech studentů

# WebApi – Studenti (2/2) – Endpointy

- V novém `StudentController` vytvoříme endpointy
  - ▶ `public IEnumerable<Student> Get ()` – seznam všech studentů
  - ▶ `public IEnumerable<Student> GetByName (string nameContains)` – restrikce na jména



# WebApi – Studenti (2/2) – Endpointy

- V novém `StudentController` vytvoříme endpointy
  - ▶ `public IEnumerable<Student> Get ()` – seznam všech studentů
  - ▶ `public IEnumerable<Student> GetByName (string nameContains)` – restrikce na jména
  - ▶ `public Student Get (int id)` – student dle Id

# WebApi – Studenti (2/2) – Endpointy

- V novém `StudentController` vytvoříme endpointy
  - ▶ `public IEnumerable<Student> Get ()` – seznam všech studentů
  - ▶ `public IEnumerable<Student> GetByName (string nameContains)` – restrikce na jména
  - ▶ `public Student Get (int id)` – student dle Id
  - ▶ `public void Put (int id, [string name, string surname)` – vytvoření, editace

# WebApi – Studenti (2/2) – Endpointy

- V novém `StudentController` vytvoříme endpointy

- ▶ `public IEnumerable<Student> Get ()` – seznam všech studentů
- ▶ `public IEnumerable<Student> GetByName (string nameContains)` – restrikce na jména
- ▶ `public Student Get (int id)` – student dle Id
- ▶ `public void Put (int id, [string name, string surname)` – vytvoření, editace
- ▶ `public void Delete (int id)` – smazání

```
1 [HttpGet ("{name}")]
2 public IEnumerable<Student> GetByName (string nameContains) {
3     return students.Where (p=>p.Name.Contains (nameContains));
4 }
```

# WebApi – OData

- Ruční filtrování je zdlouhavé a (dnes už) zbytečné
- Knihovna OData <https://www.odata.org/>
- Poskytuje pokročilé API včetně filtrace a expanze „on demand“
  - ▶ Vytvoříme nový ASP.NET Core Web Api projekt
  - ▶ Doinstalujeme balíček `Install-Package Microsoft.AspNetCore.Odata`
  - ▶ Můžeme nyní vytvořit nový kontroler napojíme na EF podobně jako předtím
  - ▶ Do Program.cs přidáme:

```
1 static IEdmModel GetEdmModel () {
2     ODataConventionModelBuilder builder = new ();
3     builder.EntitySet<Student> ("Student");
4     return builder.GetEdmModel ();
5 }
6 builder.Services.AddControllers ().AddOData (options => options
7     .AddRouteComponents ("api", GetEdmModel ())
8     .Select ().Filter ().OrderBy ().Count ().Expand ());
```

# OData – dotazování

- Endpoint umožní dotazování

```
1 [EnableQuery]
2 public IQueryable<Student> Get () {
3     return ctx.Students;
4 }
5 [EnableQuery]
6 public SingleResult<Student> Get ([FromODataUri] int key) {
7     IQueryable<Student> result = ctx.Students.Where(p => p.Id ==
8         key);
9     return SingleResult.Create(result);
10 }
```

- Pomocí URL a OData syntaxe nemusíme řešit filtraci my

# OData – Tvar dotazu

- Metadata naleznete na URL/`$metadata` např.:  
`http://localhost:54828/$metadata`, případně dle routy
- XML s popisem dostupných entit
- Filtrování pomocí URL dotazů:
  - ▶ `/Student(2)` – nalezení studenta s ID 2
  - ▶ `/Student?$Filter=Name eq 'Lukáš'` – Najde všechny Lukáše
  - ▶ `/Student?$Select=Name` – Vybere pouze jména
  - ▶ `/Student?$Select=Name&$Top=3` – První tři jména
  - ▶ `Count, Skip, Expand(napojené entity),...`

# Napojení z druhé aplikace

- Právý klik na projekt → Add connected service
- Neuvidíme-li Odata, musím nainstalovat rozšíření
  - ▶ Extensions → Manage Extensions
  - ▶ Odata Connected Service
  - ▶ Je potřeba restart Visual Studia
- Add more services, nalezneme OData Connected Service
- Můžeme přidat naši service, zadáme název a URL endpointu. Např.:  
`http://localhost:54828/$metadata`
- V SolutionExploreru se nám objevily nové položky – ConnectedServices s nagenеровaným kódy

## Druhá aplikace

- Vygenerované zdrojáky již máme, teď je jen použít

```
1 Container c = new Container(new Uri("http://localhost:54828/"));
2 var students = c.Student.Where(p => p.Name == "Ivo");
3 foreach (Student s in students) {
4     Console.WriteLine(s.Surname);
5 }
```

- Vygenerovaný kód se nám postará o překlad na OData dotaz, my používáme LINQ jakoby data byly u nás
- Dotaz se vyhodnotí, až jsou data potřeba



# Úkol

- Přidejte Web API pro aplikaci Útulku s jedním endpoitem pro vyhledávání
- Vytvořte View `/dog/search`, s textovým polem
- Pomocí AJAX při změně textu zavolejte API a zobrazte (ihned) výsledky
- Vytvořte OData Web API (klidně nová aplikace), která poskytne seznam psů
- Napojte konzolovou aplikaci na Odata API, konzolová aplikace umožní uživateli vyhledávat a filtrovat
- Bonusový bod: zprovoznění obou API v jednom (původním) projektu