

Jazyk C# 2

8. cvičení

Večeřa J., Janošík R.

Univerzita Palackého v Olomouci

11.4.2023

Reakce na úkoly

- Rozdělení kroků: Načtení - Parsování - Vložení. → Lepší hledání chyb.
- Někdy zbytečné joiny (přitom navázané entity dostupné přes .)
- Generování náhodného písmene

```
1 var rand = new Random();  
2 char grade = (char) ('A' + rand.Next(6));
```
- (Ne)provázání entit - degradace vazby 1:N na 1:1
- SaveChanges() po každém příkazu - na konci "transakce"

Webový vývoj – základy

- Jaké máte zkušenosti s vývojem webových aplikací?

Webový vývoj – základy

- Jaké máte zkušenosti s vývojem webových aplikací?
- Klient-server architektura
- Backend a frontend aplikace
- HTTP požadavky

Webový vývoj – základy

- Jaké máte zkušenosti s vývojem webových aplikací?
- Klient-server architektura
- Backend a frontend aplikace
- HTTP požadavky
- Základní typy webových aplikací
 - ▶ Statické webové stránky
 - ▶ Dynamické (generované na serveru)
 - ▶ Dynamické s AJAX
 - ▶ *Single Page Application(SPA)*
 - ▶ Kombinace

Webový vývoj – základy

- Jaké máte zkušenosti s vývojem webových aplikací?
- Klient-server architektura
- Backend a frontend aplikace
- HTTP požadavky
- Základní typy webových aplikací
 - ▶ Statické webové stránky
 - ▶ Dynamické (generované na serveru)
 - ▶ Dynamické s AJAX
 - ▶ *Single Page Application(SPA)*
 - ▶ Kombinace
- Dnes si ukážeme vývoj (klasických) dynamických webových stránek pomocí ASP.NET

ASP.NET

- Rozšíření .NET pro webové prostředí
- Využití standardních knihoven (řetězce, IO, databáze) a C# pro programování webů
- Backend: klasický C# jak jej známe
- ASP.NET přináší:
 - ▶ Framework pro zpracování webových requestů
 - ▶ Šablonovací systém pro FrontEnd (`Razor`)
 - ▶ Knihovny pro standardní webové postupy (MVC, API)
 - ▶ Zabezpečení (Auth, HTTPS, ...)
- Ukážeme si základ MVC Frameworku

MVC architektura

- Obecný programovací model
- (Poměrně) striktní oddělení vrstev aplikace
- Usnadňuje orientaci v kódu, hledání chyb
- Význam zkratky
 - ▶ M – Models
 - ▶ V – Views
 - ▶ C – Controllers

MVC architektura – Models

- Modely – (Datové) objekty implementující datovou logiku aplikace
- Velmi často získávání a ukládání dat do DB
- Pro nás budou ztotožněny s entitami EntityFrameworku
- Příklad:
 - ▶ Produkt
 - ▶ Student
 - ▶ Předmět
 - ▶ ...

MVC architektura – Views

- Views – komponenty zobrazující UI aplikace
- Pohledy, Výjevy (:
- UI se vytváří na základě dat z modelu
- Přesný popis toho, co a jak se má uživateli zobrazit
- V ASP.NET - popis, šablona webové stránky

MVC architektura – Controllers

- Kontrolery – objekty zajišťující interakci s uživatelem
- Manipulují s modely, zajišťují funkcionalitu
- Vytváří Views pro vyrenderování
- Celková aplikační/procesní logika

ASP.NET MVC Execution Process

- Aplikace neběží samostatně, ale ve webovém serveru, který odstiňuje aplikaci od síťové vrstvy
- Proces práce aplikace (zkráceně):
 - ▶ HTTP request na webový server → naše aplikace
 - ▶ Routování – zjištění, co je potřeba a která část aplikace to má dělat
 - ▶ Vytvoření MVC request handleru (přístupný v RequestContext – surovější, servisní data requestu)
 - ▶ Vytvoření Controlleru, nastavení RequestContextu
 - ▶ Zavolání konkrétní akce controlleru
 - ▶ Zobrazení výsledku (ViewResult, Redirect, JsonResult, . . .)

MVC aplikace – vytvoření

- Nový projekt → C# → Web → ASP.NET Core Web Application (Model-View-Controller)

- Struktura:

- ▶ Adresáře `Controllers`, `Models` a `Views` pro příslušné objekty
- ▶ Adresář `wwwroot` pro statická data (css, js). Výchozí *Bootstrap*, *jquery*
- ▶ `appsettings.json` – nastavení aplikace
- ▶ `Program.cs` – spouštěcí soubor dříve s metodou `Main`
 - ★ runtime konfigurace projektu (dříve v metodách)
 - ★ více

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/startup>

Jednoduchá evidence studentů

- Pomocí NuGet package manageru doinstalujeme EntityFramework
- Vytvoříme příslušné modely (Student, Address, Subject)
- Vytvoříme předka pro kontrolery, který bude obsahovat Context
- Vytvoříme nový kontroler `StudentController`
- `return View();` vrátí defaultně View pojmenovaný jako akce (dle složky)
- Při spuštění aplikace se nám otevře webový prohlížeč
- Volání akcí: `URL/JmenoKontroleru/JmenoAkce`
- např.: `http://localhost:61317/Home/Index` Spustí HomeController s akcí Index (default)

Zobrazení všech studentů

- Do `StudentController` přidáme akci `All`
- Ta z databáze vybere všechny studenty a předá je `View`

```
1 public IActionResult All() {  
2     List<Student> students = Ctx.Students.ToList();  
3     return View(students);  
4 }
```

- Ve složce `Views/Student` vytvoříme `All.cshtml` – náš `View`, nastavíme model

```
1 @model List<WebApplication1.Models.Student>
```

- `ViewBag` Dynamický kontejner pro „další data“

Views – Razor

- Šablonovací jazyk pro renderování dynamických Views – mix HTML a C#
- Důležitý znak: @ – způsobuje vyhodnocení

```
1 @{ var title = "Titulek stránky";  
2   var date = DateTime.Now;  
3   var message = " Today is: " + date; }
```

- Inline vyhodnocení

```
1 <p>Name: @Model.Name</p> <a href=".../@Model.Id">DELETE</a>
```

- Výrazy:

```
1 @if (ViewBag.Neco) {  
2   <a href="odkaz vedouci nekam">Odkaz</a>  
3 } else {  
4 <p>Nic!</p>  
5 }
```

- Tedy vhodnou kombinací C# a HTML se dají vytvářet stránky reagující na obsah

Razor – Automatická kompilace Views

- Ve starém MVC se rekompilace dělá automaticky bez nutnosti restartu aplikace
- Nyní potřeba doinstalovat
`Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation`
- V `Program.cs` **změnit** `builder.Services.AddControllersWithViews()` ;
na:
 - 1 `builder.Services.AddControllersWithViews()`
 - 2 `.AddRazorRuntimeCompilation();`
- Views by se měly dynamicky rekompilovat po změně bez restartu aplikace

Razor – cykly

- for, while, do ... while **jak je známe**

```
@for(var i = 0; i < Model.Kolekce.Length; i++)  
{  
    <p>Item @i: @Model.Kolekce[i]</p>  
}
```

- Procházení kolekcí pomocí foreach

```
@foreach (Student s in Model.Students)  
{  
    <tr><td>@s.Name</td><td>@s.Surname</td></tr>  
}
```

- Občas trochu zmatek, kdy použít @ a kdy ne

Views – Razor – Helpery

- Helpery – pomocné funkce v Razor pro práci s HTML, URL, ...
- Dokumentace: <https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers>
- Mohou vás (téměř) zcela odstínit od HTML
- Hůře se kombinují s Frontend frameworky (jQuery, KnockoutJS, ...)
- Např.: `asp-action`, `asp-route-id`

```
1 <td><a asp-action="Edit" asp-route-id="@s.Id">Editovat</a></td>
```

Editace studenta (1/2)

- Views/Student → Add View
- Pojmenujeme `Edit`, jako Template zvolíme `Edit` a zvolíme třídu `Student`
- Visual studio vygeneruje `Razor` šablonu dle vlastností třídy
- Přídáme akci `Edit` kontroleru `Student` s parametrem `int id` a anotací `HttpGet`
- Najdeme studenta a vrátíme jej jako model do View

```
1 [HttpGet]
2 public ActionResult Edit(int id) {
3     Student s = Ctx.Students.FirstOrDefault(p => p.Id == id);
4     if (s!=null) {
5         return View(s);
6     } else {
7         return NotFound("Student not found");
8     }
9 }
```

Editace studenta (2/2)

- Vznikl formulář, který se namapoval na hodnoty vlastností třídy `Student`
- Cílem formuláře je opět akce `Edit` – vytvoříme další s anotací `HttpPost`

```
1 [HttpPost]
2 public ActionResult Edit(Student studentEdited) {
3     Student s = Ctx.Students.FirstOrDefault(p => p.Id ==
4         studentEdited.Id);
5     if (s != null) {
6         s.Name = studentEdited.Name;
7         s.Surname = studentEdited.Surname;
8         s.Rocnik = studentEdited.Rocnik;
9         Ctx.SaveChanges();
10        return RedirectToAction("All");
11    } else {
12        return NotFound("Student not found");
13    }
```

Disclaimer

- Předchozí slidy nemají být rozsáhlým průvodcem do ASP.NET MVC
- Spíše slouží jako úvod do problematiky a „co vlastně existuje“
- Byly vynechány nějaké základní principy, jako:
 - ▶ Odchytávání výjimek
 - ▶ DTO
 - ▶ Bezpečnost
 - ▶ Anotace u modelů (restrikce pro generování View)
 - ▶ Deploy v IIS
 - ▶ Autentizace
 - ▶ ...
- Téma je rozsáhlé, nejde stihnout vše – je lepší zažít

Úkol 1/2

- Webová aplikace pro útulky:
 - ▶ Úvodní strana budou různé útulky
 - ▶ Každý útulek po rozkliknutí zobrazí svůj detail - seznam psů
 - ▶ Každý útulek na detailové stránce bude mít možnost přidat nového psa (tlačítkem zobrazí formulář k přidání) - po přidání se zobrazí znovu seznam i s novým psem
 - ▶ V Layoutu bude okénko hledání psa podle jména
 - ▶ Pes po rozkliknutí bude mít tlačítko "adoptovat", které smaže záznam z databáze (BONUS pro ty, kteří budou mít tlačítko u každého psa přímo ve výpisu v seznamu a záznam pouze zmizí)
 - ▶ V databázi budou persistentně ukládány záznamy o útulcích a o psech. Útulek musí vědět, jaké má psy. Pes nemusí po vyhledání vědět, v jakém je útulku (BONUS pro ty, kteří po vyhledání budou schopni proklikem zobrazit detaily o útulku, ve kterém pes je).
 - ▶ Zkuste se zamyslet, jaké informace chcete uchovávat o daných entitách (Název, popis, umístění, kapacita, ...)(Jméno, věk, rasa, popis, ...)

Úkol 2/2

- MVC architektura tedy bude mít:
 - ▶ Minimálně 4 jednoduché Views + Layout: Landing Page s útulky, Seznam psů v útulku, detail psa pro adopci (smazání), frm pro přidání psa.
 - ▶ (Doporučení) Alespoň 3 controllery: HomeController pro úvodní fce, ShelterController, DogController
 - ▶ Alespoň 2 DB tabulky: Shelter, Dog; a odpovídající modely
- Úkol za 4 body + 2 bonusové
- Na úkol máte „dva týdny“ – odevzdání tedy 25.4.
- Příští seminář - nepovinné konzultace