

Autor **Bruno Pfohl**

Obor **Informační technologie**

Vedoucí práce **Ing. Tomáš Kazda, DiS.**

Školní rok **2018/2019**

Střední průmyslová škola strojní   
a elektrotechnická a Vyšší odborná škola, Liberec 1, Masarykova 3

Vývoj v ASP.NET core

Maturitní/Ročníková práce

Anotace (Resumé)

Práce se zabývá vývojem webových aplikací pomocí technologie ASP.NET Core. Hlavním cílem práce je vytvoření studijních materiálů a ukázkové webové aplikace, která bude následovat postupy popsané ve studijních materiálech. Nedílnou součástí práce je také ověření výstupů na cílové skupině studentů. Zpětná odezva studentů je v této práci taktéž uvedena.

Summary

This work is about creating web applications using ASP.NET Core technology. It’s main goal is to create studying materials and sample web application. The feedback of students on studying materials is also included in this work.

Čestné prohlášení

Prohlašuji, že jsem předkládanou maturitní/ročníkovou práci vypracoval(a) sám(a) a uvedl jsem veškerou použitou literaturu a bibliografické citace.

V Liberci dne

Bruno Pfohl

Obsah

[Úvod 1](#_Toc3230856)

[1 Tvorba studijních materiálů 2](#_Toc3230857)

[1.1 Software potřebný k vývoji 2](#_Toc3230858)

[1.2 Technické specifikace .NET Core a ASP.NET Core 2](#_Toc3230859)

[1.2.1 Framework .NET Core obecně 3](#_Toc3230860)

[1.2.2 Porovnání frameworku .NET Core s frameworkem .NET 3](#_Toc3230861)

[1.2.3 Framework ASP.NET Core 3](#_Toc3230862)

[1.3 Vytvoření prvního projektu 4](#_Toc3230863)

[1.3.1 Založení projektu 4](#_Toc3230864)

[1.3.2 Struktura projektu 5](#_Toc3230865)

[1.3.3 Názorná ukázka request PIPELINE 7](#_Toc3230866)

[1.4 Principy MVC architektury 8](#_Toc3230867)

[V této kapitole si detailně popíšeme MVC architekturu a ukážeme si, jak tuto architekturu implementovat do ASP.NET Core projektu. 8](#_Toc3230868)

[1.4.1 Co je to MVC architektura? 8](#_Toc3230869)

[1.4.2 Princip MVC achitektury 8](#_Toc3230870)

[1.5 Vytvoření projektu implementující architekturu MVC 9](#_Toc3230871)

[1.5.1 Založení projektu 9](#_Toc3230872)

[1.5.2 Struktura projektu 10](#_Toc3230873)

[1.5.3 Praktická ukázka MVC 11](#_Toc3230874)

[1.6 Razor View Engine 16](#_Toc3230875)

[1.6.1 Využití Razor View Engine v MVC projektu 16](#_Toc3230876)

[1.6.2 Razor syntaxe 17](#_Toc3230877)

[1.7 HTML Tag Helpers 19](#_Toc3230878)

[1.7.1 Výhody HTML Tag Helpers 19](#_Toc3230879)

[1.7.2 Přidání nových HTML Tag Helpers. 19](#_Toc3230880)

[1.7.3 Výchozí HTML Tag Helpers 19](#_Toc3230881)

[1.8 Formuláře 20](#_Toc3230882)

[1.8.1 Vytvoření modelu 21](#_Toc3230883)

[1.8.2 Vytvoření potřebných akcí v řadiči 21](#_Toc3230884)

[1.8.3 Vytvoření formuláře pomocí HTML Tag Helpers 22](#_Toc3230885)

[1.9 Dekompozice softwarové logiky aplikace 24](#_Toc3230886)

[1.9.1 Hlavní důvody pro dekompozici aplikace 24](#_Toc3230887)

[1.9.2 Dekompozice aplikace na 3 vrstvy 25](#_Toc3230888)

[1.10 Entity Framework Core 26](#_Toc3230889)

[1.10.1 Model first 26](#_Toc3230890)

[1.10.2 Database first 27](#_Toc3230891)

[1.10.3 Code first 27](#_Toc3230892)

[1.11 Dependency Injection 28](#_Toc3230893)

[1.11.1 Vkládání závislostí v ASP.NET Core 28](#_Toc3230894)

[1.11.2 Způsoby registrace služeb 28](#_Toc3230895)

[1.11.3 Praktická ukázka 29](#_Toc3230896)

[1.12 Tvorba databáze pomocí Code First přístupu 31](#_Toc3230897)

[1.12.1 Vytvoření projektu 31](#_Toc3230898)

[1.12.2 Instalace potřebných balíčků 32](#_Toc3230899)

[1.12.3 Vytvoření entit 33](#_Toc3230900)

[1.12.4 Vytvoření DbContext 34](#_Toc3230901)

[1.12.5 Využití datové vrstvy v prezenční vrstvě 35](#_Toc3230902)

[1.12.6 Databázové migrace 37](#_Toc3230903)

[1.13 Repository Pattern a Unit Of Work 38](#_Toc3230904)

[1.13.1 Adresářová struktura 39](#_Toc3230905)

[1.13.2 Vytvoření generického repozitáře 39](#_Toc3230906)

[1.13.3 Vytvoření repozitáře pro tabulku aut 42](#_Toc3230907)

[1.13.4 Vytvoření Unit of Work 42](#_Toc3230908)

[1.13.5 Využití Unit Of Work v prezenční vrstvě 44](#_Toc3230909)

[2 Tvorba ukázkové aplikace 46](#_Toc3230910)

[2.1 Struktura projektu 46](#_Toc3230911)

[2.1.1 Prezenční vrstva 46](#_Toc3230912)

[2.1.2 Datová vrstva 46](#_Toc3230913)

[2.2 Vzhled aplikace 47](#_Toc3230914)

[3 Ověření výstupu na studentech 51](#_Toc3230915)

[3.1 Zvolená cílová skupina studentů 51](#_Toc3230916)

[3.2 Odezva jednotlivých studentů 51](#_Toc3230917)

[3.3 Shrnutí a ponaučení 56](#_Toc3230918)

[Závěr 57](#_Toc3230919)

[Seznam obrázků 58](#_Toc3230920)

[Použitá literatura 60](#_Toc3230921)

[A. Seznam přiložených souborů 1](#_Toc3230922)

[B. Další příloha 2](#_Toc3230923)

Úvod

Tato práce vznikla za účelem vytvořit stručné studijní materiály popisující validní postupy při vytváření webových aplikací pomocí technologie ASP.NET Core. Důvod, proč jsem se rozhodnul vypracovat právě tuto práci je, že jsem chtěl studentům předat své zkušenosti, které jsem nabyl v zaměstnání, kde se věnuji programování webových aplikací téměř dva roky.

Studijní materiály obsahují jak teoretické vysvětlení postupů, tak i praktické úlohy, které si studenti mohou vyzkoušet. Za zmínku také stojí, že se nejedná o kompletní návod na to, jak vytvářet webové aplikace. Materiály se zabývají jen některými částmi vývoje, které jsme po domluvě s mým vedoucím maturitní práce, Ing. Tomášem Kazdou, určili za důležité.

Součástí maturitní práce je i ukázková webová aplikace. Jedná se o internetové fórum, kam se mohou uživatelé registrovat, přidávat příspěvky a komentovat je. Webová aplikace je navržena tak, aby následovala postupy, které jsou popsané ve studijních materiálech a studentům tak sloužila jako nástroj pro lepší pochopení materiálů.

Poslední kapitola se zabývá odezvou studentů na tyto studijní materiály. Konkrétně se jedná o skupinu studentů ze třídy P4, které Ing. Tomáš Kazda za pomocí mých studijních materiálů vyučoval v průběhu výuky předmětu ASW.

Veškeré soubory jsou nahrány na CD, které je přiloženo k této práci.

Použití šablony

Nastavte název dokumentu a autora v nabídce Soubor/Vlastnosti.

Pro vkládání zdrojů použijte Reference/Spravovat prameny.

1. Tvorba studijních materiálů

Studijní materiály jsem se rozhodnul vytvořit v programu Microsoft Word. Jednotlivé kapitoly jsem rozdělil do samostatných souborů, které jsem nahrál do repozitáře na stránce github.com, ke kterému jsem studentům umožnil přístup. Následující podkapitoly této kapitoly obsahují všechny studijní materiály, které jsem vytvořil.

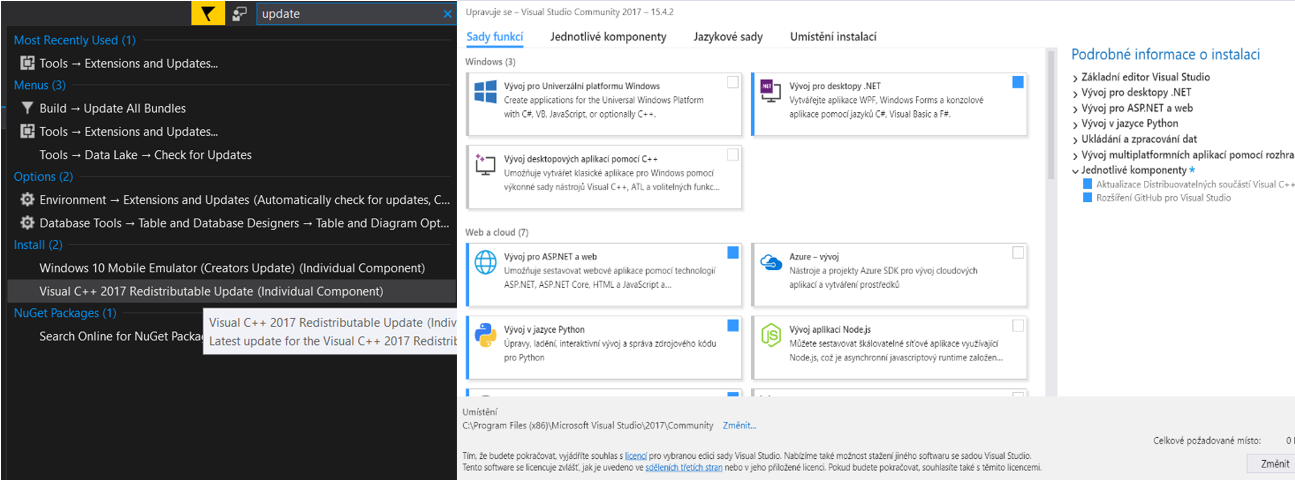
* 1. Software potřebný k vývoji

Pro bezproblémové následování těchto studijních materiálů bude z Vaší strany nutné, abyste si hned zpočátku nainstalovali veškerý potřebný software.

Potřebný software

1. VS 2017 (<https://visualstudio.microsoft.com/downloads/>)
2. .NET Core SDK a .NET Core Runtime (<https://dotnet.microsoft.com/download>)

Úprava stávající instalace vývojového prostředí Visual Studio



Obrázek 1 Úprava stávající instalace programu Visual Studio

* 1. Technické specifikace .NET Core a ASP.NET Core

Tato kapitola se zabývá technickými specifikacemi frameworků .NET Core a ASP.NET Core. Součástí kapitoly je taky porovnání těchto frameworků s jejich předchůdci, tedy frameworky .NET a ASP.NET.

* + 1. Framework .NET Core obecně

Framework .NET Core je technologie od společnosti Microsoft pro vytváření aplikací pomocí jazyku C# a F#. Navazuje na framework .NET 4.x, od kterého se však v mnohých ohledech liší. Původně se měl jmenovat .NET 5, ale společnost Microsoft chtěla na zásadní rozdíly mezi těmito technologiemi upozornit, a proto ho vydala pod jiným názvem.

Tento framework podporuje pouze tři typy aplikací, kterými jsou knihovny, testy jednotek a konzolové aplikace. To však neznamená, že se framework nedá využít pro vývoj jiného typu aplikací. Stačí pouze vytvořit nadstavbu nad konzolovou aplikaci. Tímto způsobem funguje i ASP.NET Core, což si později v tomto návodu názorně ukážeme.

* + 1. Porovnání frameworku .NET Core s frameworkem .NET

Multiplatformní

Na rozdíl od svého předchůdce je multiplatformní, což znamená, že webová aplikace vytvořená pomocí tohoto frameworku může být spuštěna nejen na operačním systému Windows, ale i na Linuxu a Mac OS.

Modulárnost a rychlost

Je více modulární než jeho předchůdce, což umožňuje jednodušší optimalizaci aplikace. Při vytváření aplikace se postupně přidávají knihovny(moduly), které potřebujete, což ve výsledku vede k lepší optimalizaci aplikace.

Open source

Díky tomu, že se jedná o plně open-source framework, k jehož kódu má kdokoliv přístup, se na vývoji může podílet každý. Microsoft si pro verzování frameworku zvolil webovou stránku github.com, kde má repozitář pro frameworky .NET Core a ASP.NET Core.

* + 1. Framework ASP.NET Core

Jedná se o kompletní webový framework, který lze provozovat jak na novém .NET Core frameworku, tak i na starším .NET frameworku.

Doporučený a také nejvíce užívaný způsob využití tohoto frameworku je za pomocí implementace MVC architektury, která rozděluje aplikaci do několika částí. Nic nám však nebrání v tom tuto architekturu neimplementovat a dotazy, posílané na náš webový server, zpracovávat jiným způsobem.

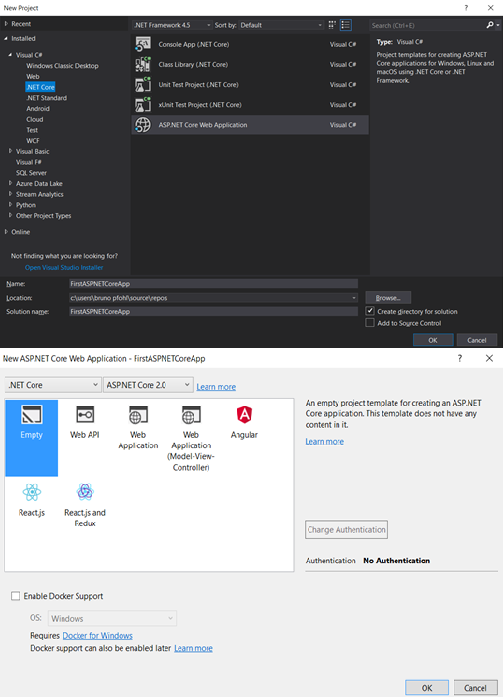
* 1. Vytvoření prvního projektu

V této kapitole je podrobný návod, jak vytvořit webovou aplikaci pomocí vývojového prostředí Visual Studio. Konkrétně jedná o velmi jednoduchou aplikaci, která neimplementuje MVC architekturu. Na této aplikaci si vysvětlíme základní strukturu ASP.NET Core projektu.

* + 1. Založení projektu

Postup

1. Klikneme na založit nový projekt
2. Zvolíme typ projektu „ASP.NET Core Web Application“
3. Zvolíme šablonu „Empty“



Obrázek 2 Založení prvního projektu

* + 1. Struktura projektu

Důležité části projektu

1. složka wwwroot
2. soubor Program.cs
3. soubor Startup.cs
4. soubor launchSettings.json

#### Obsah souboru Program.cs

Tento soubor by některým lidem, s předchozími zkušenostmi s programováním v jazyce C#, mohl připomínat konzolovou aplikaci. Je to pochopitelné, protože se o konzolovou aplikaci skutečně jedná. Tato aplikace slouží k nastavení a spuštění webového serveru.



**Metoda Main**

V této metodě dochází k sestavení a spuštění webového serveru. Pro sestavení webového serveru se využívá metoda BuildWebHost, které jsou předány parametry args. Tyto parametry jí jsou předány při spuštění aplikace. Pro spuštění serveru se volá metoda Run.

**Metoda BuildWebHost**

Uvnitř této metody se vyskytuje kód, který slouží k vytvoření, nastavení a sestavení webového serveru. Na obrázku můžeme vidět, že se volá metoda CreateDefaultBuilder. Tato metoda vrací builder s již přednastavenými vlastnostmi. Jednou z těchto vlastností je i to, že se jako webový server využívá Kestrel, který je multiplatformní a lze ho tedy spustit nejen na Windows, ale i na Mac Os a Linuxu.

Pomocí metody UseStartup je specifikováno, kterou třídu má builder využít k nastavení webového serveru. V našem případě se jedná o třídu Startup, která se nachází v souboru Startup.cs.

#### Obsah souboru Startup.cs

Třída Startup slouží k registrování služeb a sestavení request PIPELINE. Jak můžete v kódu vidět, třída nemá žádného předka. Správnou implementaci startup třídy tedy představuje jakákoliv třída, která dodržuje nutné konvence (tj. má metody ConfigureServices a Configure tak, jak je vidět na obrázku).



**Metoda ConfigureServices**

Obsahuje obslužný kód pro registraci a konfiguraci služeb, které aplikace potřebuje ke svojí funkčnosti. Právě v této metodě dochází k naplnění tzv. IoC Containeru (Inversion of Control Container), kterému dal Microsoft název Service Provider. Service Provider se stará o předávání služeb (objektů) ostatním třídám, jejichž funkčnost je na těchto službách závislá. Tuto funkci frameworku si detailněji popíšeme později.

**Metoda Configure**

Stará se o sestavení tzv. request PIPELINE. Jinými slovy, určuje, jak bude webový server zpracovávat dotazy od uživatele. Ve většině případů budeme chtít, aby dotaz procházel přes nějaký middleware, který jej zpracuje a předá dalšímu middleware zaregistrovaném do PIPELINE. Kód této metody, který vidíte na obrázku pouze registruje middleware pro zobrazení stránky s detailní chybovou hláškou pro vývojáře. Pokud tedy v aplikaci při zpracování dotazu nenastane žádná chyba, server vrátí pouze textový řetězec „Hello World!“, protože dotaz bude přes middleware předán dál.

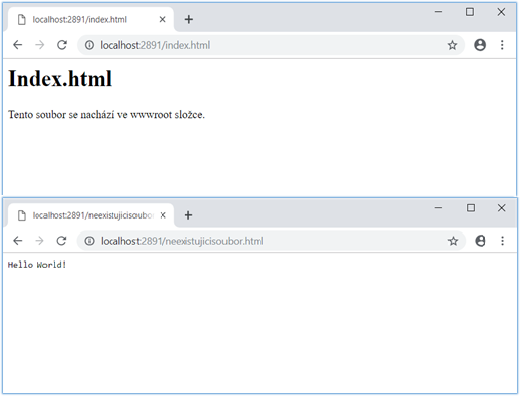
* + 1. Názorná ukázka request PIPELINE

Kód



Do Configure metody ve třídě Startup bylo přidáno volání funkce UseStaticFiles, čímž byl zaregistrován middleware, který zajišťuje uživatelovi přístup k statickým souborům ve složce wwwroot.

Volání této metody se nachází nad voláním app.Run, což v praxi znamená, že ke vrácení textového řetězce „Hello World“ dojde pouze tehdy, pokud nebude uživatelem vyžádán soubor nacházející se ve složce wwwroot, nebo pokud nedojde k chybě. Důkazem toho, že se tak PIPELINE skutečně chová, je následující obrázek.



Obrázek 3 Názorná ukázka request PIPELINE

* 1. Principy MVC architektury

V této kapitole si detailně popíšeme MVC architekturu a ukážeme si, jak tuto architekturu implementovat do ASP.NET Core projektu.

* + 1. Co je to MVC architektura?

Model-View-Controller (ve zkratce MVC) je architektonický vzor pro vytváření prezenční vrstvy aplikace. Jinými slovy, souvisí pouze s tou částí aplikace, která se stará o získání uživatelem požadovaných dat a jejich zobrazení.

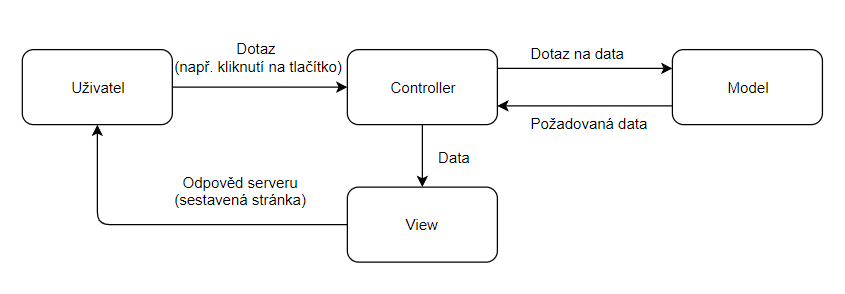
* + 1. Princip MVC achitektury

Jak už název napovídá, skládá se ze 3 vrstev, kterými jsou…

1. Model (objekt obsahující data pro view)
2. View (uživatelské rozhraní)
3. Controller (řadič)

Pro úplnou funkčnost MVC architektury je však kritický ještě směřovač(router), který podle URL a parametrů určuje, který řadič bude zpracovávat daný dotaz.

V momentě, kdy uživatel klikne např. na nějaké tlačítko umístěné v uživatelském rozhraní, se odešle dotaz na webový server, který zpracuje směrovač. Směrovačem zvolený řadič provede nutné operace, získá data a naplní jimi model. Model pak dále předá uživatelskému rozhraní, čímž vznikne vykonstruovaná stránka, která se vrátí uživatelovi.



Obrázek 4 Princip MVC architektury

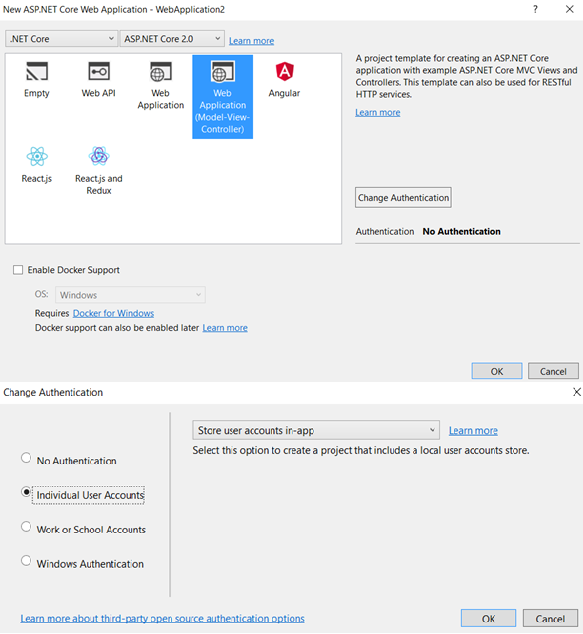
Na obrázku je vidět rozdělení prezenční vrstvy na logickou, datovou a vizuální. Toto rozdělení vede k více přehlednému, a hlavně snadněji udržitelnému kódu. Při implementaci MVC architektury by se ale však mělo myslet na to, že prezenční vrstva by měla obsahovat co nejméně logiky. V praxi to znamená, že každá rozsáhlejší aplikace by měla být rozdělena do více vrstev (projektů), které poté bude řadič využívat pro získání dat (např. samostatný projekt s databází).

* 1. Vytvoření projektu implementující architekturu MVC

Tato se zabývá vytvořením webového projektu a implementací MVC architektury. Vzhledem k tomu, že MVC architektura je mezi vývojáři ASP.NET Core webových aplikací nejoblíbenější, je její implementace velmi jednoduchá a také velmi dobře zdokumentovaná.

* + 1. Založení projektu

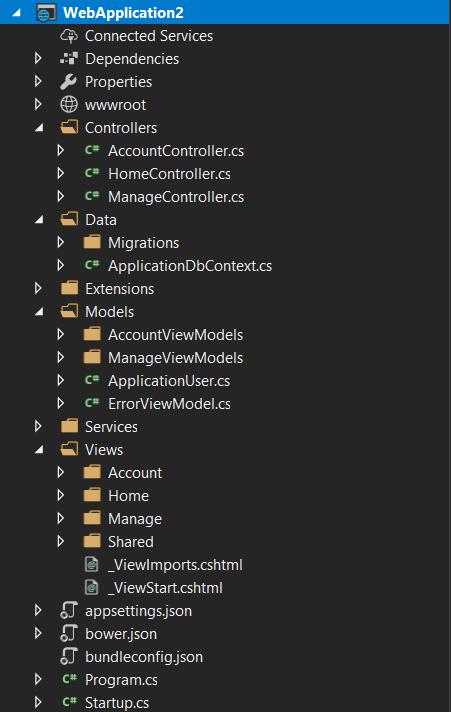
Založíme si tedy nový projekt pomocí stejného postupu jako u předchozího projektu. Tentokrát však využijeme šablonu Web Application (Model-View-Controller) a do projektu přidáme autentifikaci uživatelů.



Obrázek 5 Založení projektu implementující MVC

Po potvrzení volby se vytvoří webový projekt, který již plně implementuje architekturu MVC.

* + 1. Struktura projektu
* Složka Controllers
  + Obsahuje řadiče
* Složka Models
  + Obsahuje modely
* Složka Views
  + Obsahuje uživatelská rozhraní (Razor šablony)
  + Podložky této složky jsou pojmenovány podle řadičů
  + Šablony v jejich podsložkách jsou pojmenovány podle akcí řadiče



Obrázek 6 Struktura projektu implementující MVC architekturu

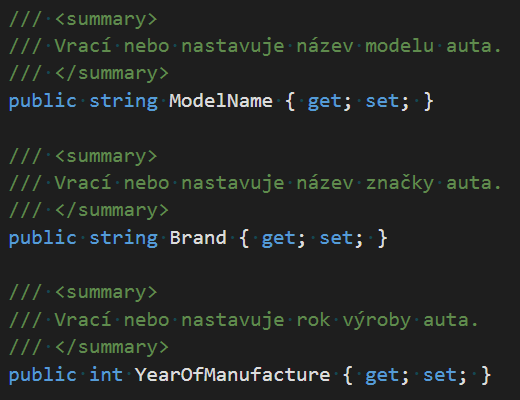
* + 1. Praktická ukázka MVC

Kroky

1. Vytvoření modelu
2. Vytvoření řadiče
3. Vytvoření uživatelského rozhraní (šablony)

#### Vytvoření modelu

Jako první je potřeba si vytvořit model. Do složky Models přidáme třídu CarViewModel, která bude uchovávat informace o autě.

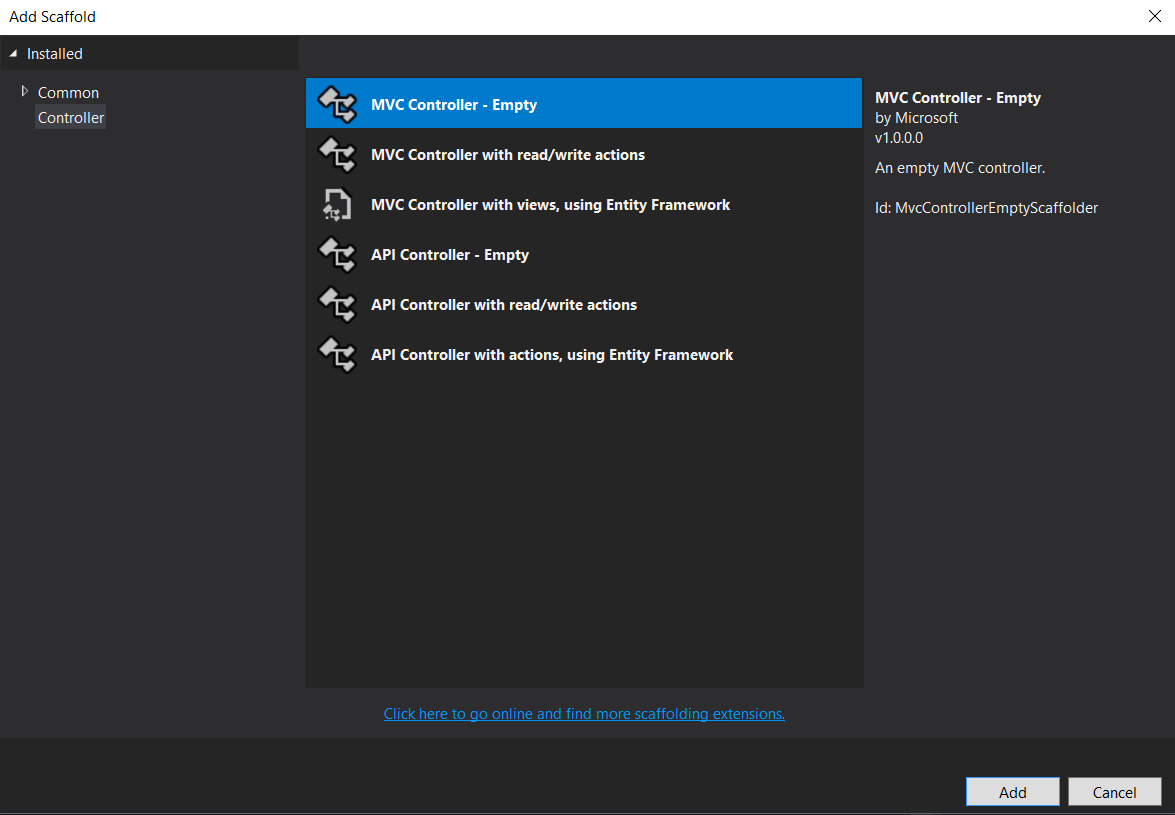


Obrázek 7 Kód třídy CarViewModel

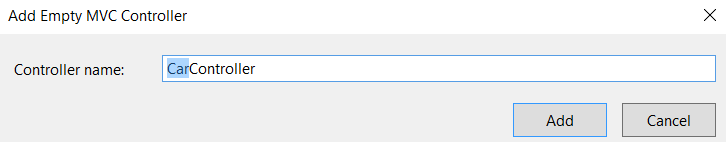
#### Vytvoření řadiče

Postup

1. Kliknout pravým tlačítkem myši na složku Controllers
2. Add
3. Controller
4. Zvolit šablonu MVC Controller – Empty
5. Pojmenovat řadič a potvrdit



Obrázek 8 Přidání řadiče

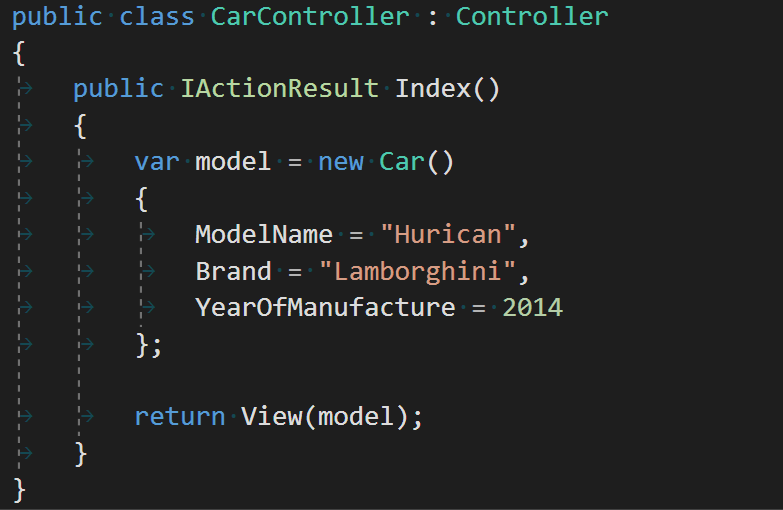


Obrázek 9 Pojmenování řadiče

#### Struktura řadiče

Každý řadič obsahuje tzv. akce. Tyto akce jsou v podstatě jen metody v třídě řadiče. Při zpracovávání dotazu se směrovač rozhoduje, které akci, v jakém řadiči, data předá. Směrovačem zvolená akce pak data zpracuje a vrátí odpověď, kterou může být například HTML stránka.

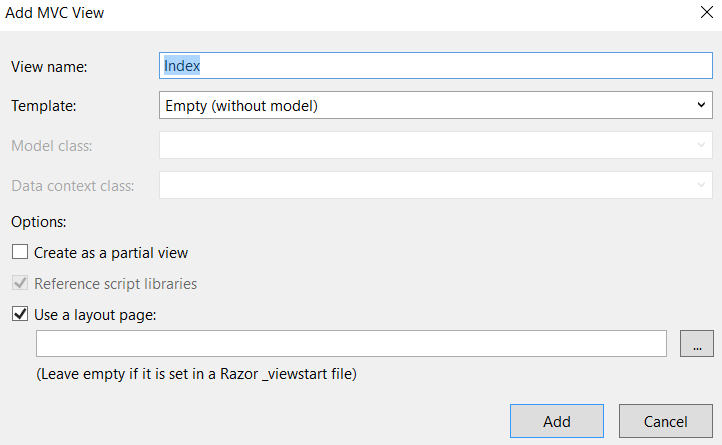
V našem případě tedy máme jednu akci jménem Index nacházející se v řadiči Car. V této akci vytváříme model, který pak předáváme metodě View. Nemáme však ještě uživatelské rozhraní (view) patřící k této akci.



Obrázek 10 Kód řadiče

#### Vytvoření šablony

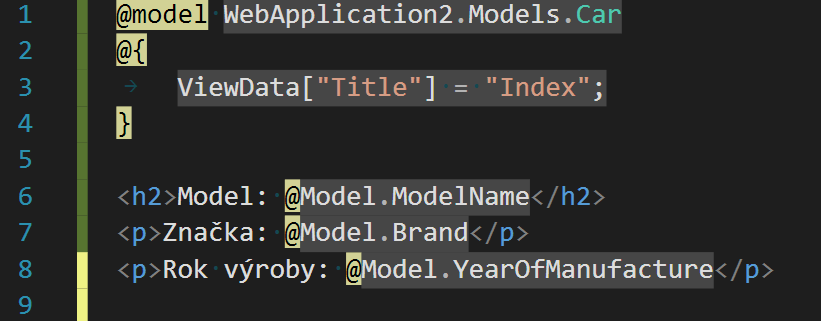
Klikneme pravým tlačítkem myši uvnitř metody Index a zvolíme „Add View“. Otevře se nám dialogové okno, které potvrdíme.



Obrázek 11 Vytvoření šablony

Po potvrzení se nám vygeneruje šablona s příponou „.cshtml“. Do šablony vepíšeme kód z obrázku. Jedná se o velmi jednoduchý pohled, který pouze vypíše data z modelu. Na 1. řádku je specifikován datový typ modelu. Tento model lze využít kdekoliv v šabloně pomocí příkazu @Model.vlastnost.

Poznámka: Akce řadiče musí uživatelskému rozhraní předat model stejného typu jako vyžaduje šablona.

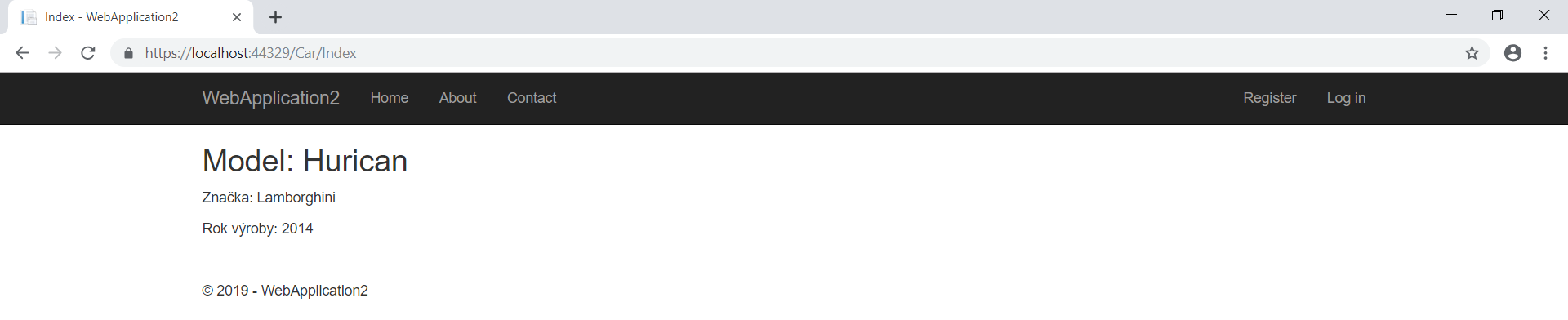


Obrázek 12 Kód šablony

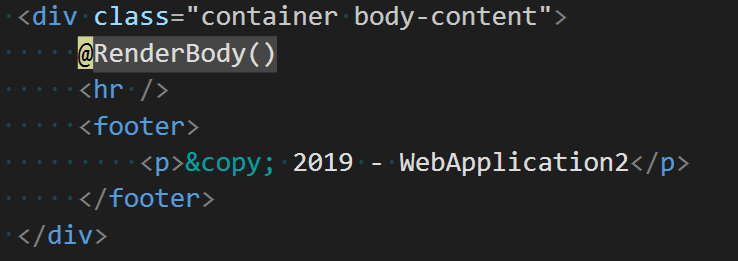
#### Ověření funkčnosti webové aplikace

Na obrázku 13 je zachycen výstup aplikace. Data z modelu se zobrazují, takže vše funguje. Kromě dat z modelu však můžeme vidět i hlavičku a patičku stránky, kterou jsme do šablony nepřidávali.

Hlavička a patička byly přidány proto, že se naše šablona vložila do tzv. layoutu, který se nachází v souboru Views/Shared/\_Layout.cshtml.



Obrázek 13 Výstup webové aplikace implementující MVC



Obrázek 14 Kód v souboru \_Layout.cshtml

V souboru \_Layouts.cshtml je spousta kódu, který je součástí výchozí šablony MVC. V tuto chvíli je pro nás však zajímavý pouze řádek, na kterém se volá příkaz @RenderBody(). Právě na místo, kde se tento příkaz volá se vkládají šablony, které tento layout využívají.

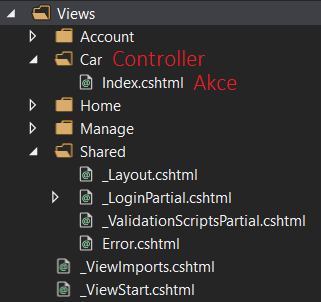
* 1. Razor View Engine

Razor View Engine je view engine vytvořený společností Microsoft. Jedná se o ASP.NET komunitou nejvíce používaný view engine. Dalšími alternativami jsou např. Spark, NDjango nebo NHaml. Umožnuje vytváření dynamických stránek pomocí kombinace HTML značkovacího jazyka a Razor syntaxe, která umožňuje vkládat C# kód do HTML. Soubor Razor šablony má koncovku „.cshtml“.

* + 1. Využití Razor View Engine v MVC projektu

#### Umístění Razor šablon

Jednotlivé šablony se nacházejí ve složce Views. Struktura této složky musí následovat konvenci určenou MVC návrhem. Každá podsložka nese stejný název jako jeden z řadičů a šablony v ní jsou určeny pro jednotlivé akce tohoto řadiče. Výjimkou je pouze složka Shared, která obsahuje sdílené šablony.



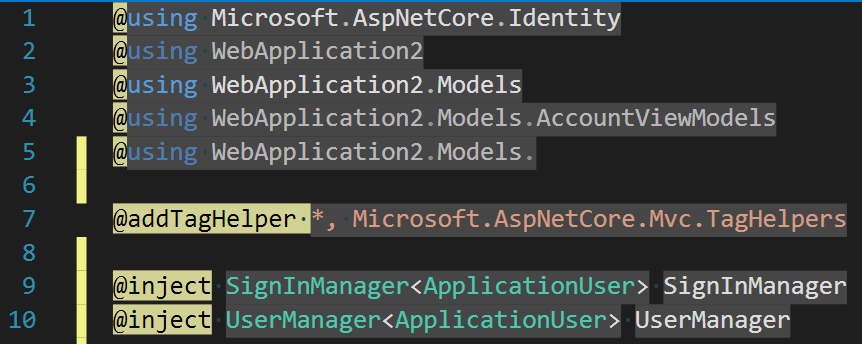
Obrázek 15 Struktura složky Views

#### Složka Shared

Obsahuje sdílené šablony. Můžeme v ní najít např. šablonu \_Layout.cshtml nebo částečná zobrazení, tzv. „partial views“. Typicky jsou v této složce převážně šablony, které jsou využívány větším množstvím jiných šablon. Kód šablony se tedy objeví pouze na jednom místě v projektu a programátor se tak může jednoduše vyhnout redundantnímu kódu.

#### Soubor \_ViewImports.cshtml

V tomto souboru se importují knihovny, vkládají závislosti a definují tzv. „helper tagy“, které jsou využívány ve větším množství ostatních šablon. Tento soubor nám tedy umožňuje vyhnout se redundantnímu kódu a mít vše přehledně na jednom místě.

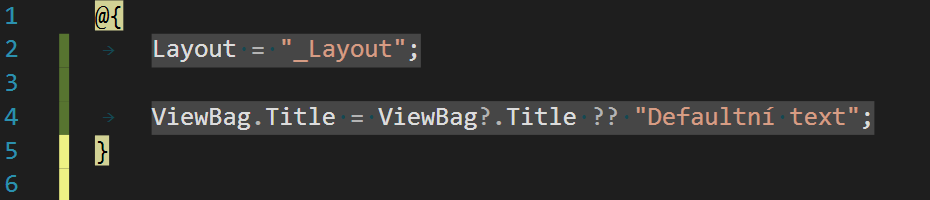


Obrázek 16 Kód v souboru \_ViewImports.cshtml

#### Soubor \_ViewStart.cshtml

Kód obsažený v tomto souboru se spustí před renderem šablony. Tento soubor se nachází přímo v kořenové složce Views. Umístění souboru právě v této složce má svůj důvod. Razor View Engine nejdříve hledá \_ViewStart.cshtml soubor v podsložce související s daným řadičem a pokud tam tento soubor nenajde, začne hledat v kořenové složce.

Typicky se v tomto souboru nachází specifikování layoutu. Můžeme ho však využít i k jiným účelům. Např. v obrázku 17 můžete vidět kód, který slouží k nastavení výchozího nadpisu stránky.



Obrázek 17 Využití souboru \_ViewStart.cshtml

* + 1. Razor syntaxe

Hlavní účel Razor šablon je dynamické vytváření obsahu. Toho můžeme docílit použitím tzv. Razor syntaxe, která nám umožňuje v šabloně spustit jakýkoliv C# kód. Razor syntaxe se vyznačuje znakem „@“.

#### Základní dělení Razor syntaxe

1. implicitní
   1. výraz se rovnou dosadí do šablony
   2. např. @DateTime.Now dosadí do šablony aktuální čas
2. explicitní
   1. výraz se vypočítá a dosadí
   2. např. výraz @(DateTime.Now – TimeSpan.FromHours()) odečte od aktuálního času 1 hodinu a výsledek dosadí do šablony
3. blok kódu
   1. jedná se o jakýkoliv blok C# kódu
   2. např. @{ var cislo = 5; } uloží do hodnotu 5 do proměnné @cislo, ke které můžeme poté kdekoliv v šabloně přistupovat.

#### Ukázka Razor syntaxe v praxi

V šabloně je definována kolekce aut, která se vypisuje jakožto nečíslovaný seznam. Přes kolekci aut iterujeme pomocí foreach cyklu a každé auto pak vypisujeme jako položku nečíslovaného seznamu.

Správně bychom však položky určené k zobrazení neměli vkládat přímo do šablony, ale raději do databáze. Z těchto záznamů by se pak tvořily modely, které by byly následně šabloně předány.



Obrázek 18 Využití Razor syntaxe v praxi

* 1. HTML Tag Helpers

HTML Tag Helpers jsou nástroje umožňující dynamické renderování HTML tagů velmi elegantním způsobem. Vznikly jako alternativa k tzv. HTML Helpers, které plní stejnou funkci, jen s rozdílem, že se jedná o metody, které vrací námi požadovaný HTML kód jako textový řetězec.

Ve webovém projektu verze ASP.NET Core 2.0 a vyšší, vytvořeném z MVC šablony, jsou již v základu přidány některé HTML Tag Helpers, sloužící např. k vytvoření odkazu, formuláře, obrázku atd.

* + 1. Výhody HTML Tag Helpers

Jsou svou syntaxí většinou identické klasickým HTML tagům. Pro každého, kdo zná základní syntaxi značkovacího jazyka HTML, se tedy jedná o velmi intuitivní a praktický nástroj. K této intuitivnosti také přispívá i to, že při vyvíjení ve vývojovém prostředí Visual Studio nám našeptávač IntelliSense napovídá, které parametry tagu lze použít a které hodnoty do nich lze dosadit.

* + 1. Přidání nových HTML Tag Helpers.

Aby byly HTML Tag Helpers přístupné v Razor šablonách, je nutné je přidat pomocí příkazu @addTagHelper. Tag lze přidat přímo v šabloně, kde ho chcete využít. Častěji se však tagy přidávají v souboru \_ViewImports.cshtml.



Obrázek 19 Příklad použití příkazu @addTagHelper

* + 1. Výchozí HTML Tag Helpers

Seznam některých tagů

1. odkaz
2. odkaz s parametry
3. formulář
4. vstupní pole
5. popis pole
6. velké vstupní pole
7. výběrové pole
8. validační zpráva

#### Odkaz



#### Odkaz s parametrem



#### Vstupní pole



#### Velké vstupní pole



#### Výběrové pole



* 1. Formuláře

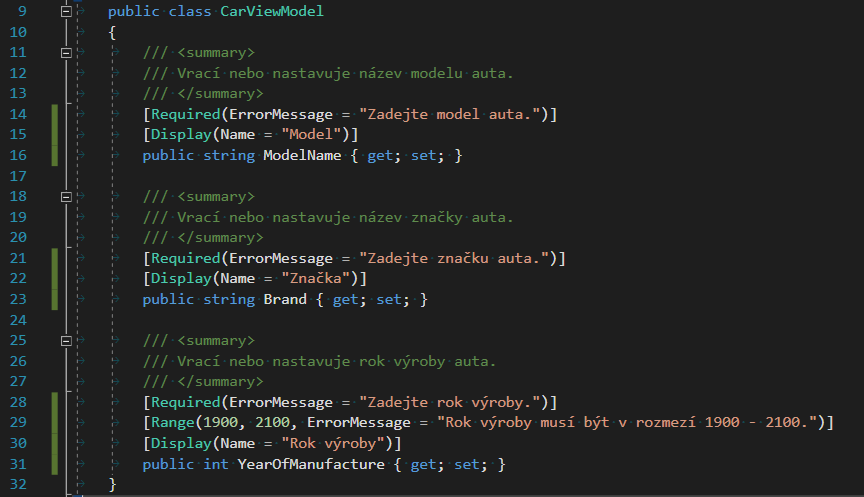
Tato kapitola se zabývá principy elektronických formulářů a jejich tvorbou pomocí technologie HTML Tag Helpers.

Formulář slouží ke shromáždění informací zadaných uživatelem. Tyto informace se pak po potvrzení formuláře odešlou na server, kde jsou zpracovány akcí v nějakém řadiči.

Základní princip formulářů není nijak složitý. Dle mého názoru bude nejlepší si vše ukázat v praxi. Vytvoříme si tedy jednoduchý formulář, který bude sloužit k vyplnění základních informací o autě. Formulář bude využívat datových anotací pro popisky vstupních polí a určení podmínek, které musí data splňovat, aby prošla validací.

* + 1. Vytvoření modelu

Pro náš formulář si nejdříve musíme vytvořit datový model. V momentě, kdy odešleme data formuláře na server, si framework sám vytvoří novou instanci modelu tohoto typu a naplní ji přijatými daty. Model se také využívá při vytváření formuláře pro dosazení výchozích hodnot.



Obrázek 20 Model formuláře

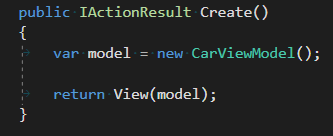
Náš model obsahuje 3 vlastnosti, kterými jsou značka auta, model auta a rok výroby. Jak můžete vidět na obrázku, nad každou vlastností jsou umístěny atributy, které specifikují text popisku vstupního pole ve formuláři (atribut Display) a podmínky, které musí hodnoty splňovat. Např. vlastnosti YearOfManufacture musíme ve formuláři vyplnit hodnotu (atribut Required), která se může pohybovat v rozmezí 1900 až 2100 (atribut Range).

* + 1. Vytvoření potřebných akcí v řadiči

Pro vytvoření formuláře potřebujeme do řadiče přidat celkově 2 akce. První akce nám vrátí formulář, který budeme moct vyplnit a druhá bude data odeslaná formulářem zpracovávat.

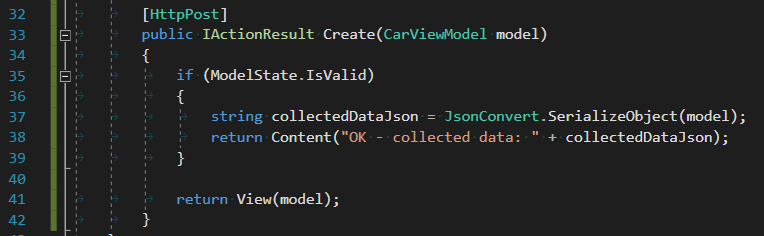
#### Vytvoření akce pro vrácení formuláře

Tato akce pouze vytváří prázdnou instanci modelu, kterou předá šabloně formuláře. Je nutno dodat, že předávat šabloně formuláře prázdný model není nutností. Například, při editaci záznamu v databázi by se šabloně předával model s předvyplněnými hodnotami.



Obrázek 21 Kód akce pro vrácení formuláře

#### Vytvoření akce pro zpracování dat

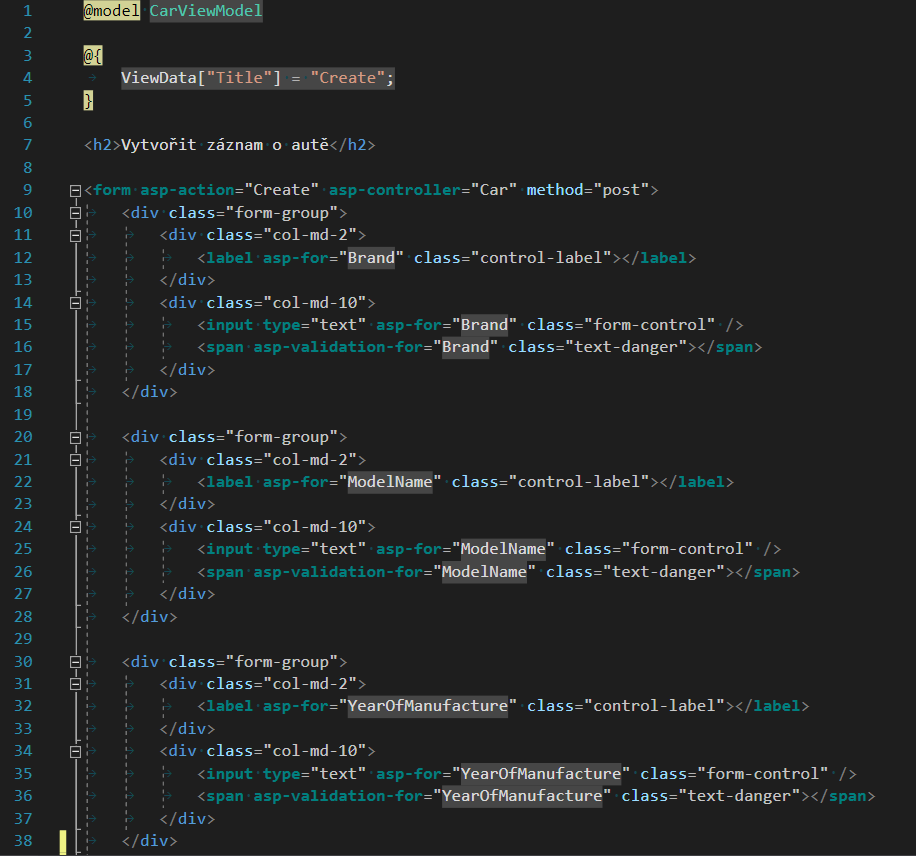


Obrázek 22 Kód akce pro zpracování dat

Tato akce přijímá jako parametr objekt typu CarViewModel. Všimněte si, že se nad metodou nachází atribut HttpPost, který specifikuje, že tuto akci si můžeme vyžádat pouze pomocí požadavku typu POST.

Na řádku 35 můžeme vidět podmíněné větvení na základě vlastnosti ModelState.IsValid. Hodnota této vlastnosti nabývá hodnot „true“ nebo „false“, kde „true“ znamená, že data jsou validní a „false“, že nejsou validní. Pokud jsou data validní, vrátí akce textový řetězec s informacemi o autě, pokud ne, tak vrátí stránku s formulářem s chybovými hláškami, které jsme specifikovali pomocí atributů v našem modelu.

* + 1. Vytvoření formuláře pomocí HTML Tag Helpers

Nyní už zbývá jen vytvořit šablonu formuláře. Kód šablony můžete vidět na následujícím obrázku. 

Obrázek 23 Kód šablony formuláře

#### Specifikování modelu

Každá šablona využívající data z nějakého modelu musí mít datový typ modelu specifikovaný pomocí příkazu @model. V našem případě se konkrétně jedná o příkaz @model CarViewModel, který se v šabloně nachází na řádku číslo 1.

#### <Form> tag

Kromě klasického atributu method, který určuje způsob předávání dat, se nastavují ještě atributy asp-action a asp-controller. Hodnoty těchto atributů určují, jaké akci, v kterém řadiči, se data odešlou.

#### Popisek vstupního pole (<label> tag)

Tento tag nám slouží jako popisek polí. Pomocí atributu asp-for se určuje, ke kterému poli se popisek váže. Popisek se pak zobrazí vedle vstupního pole jakožto text, jemuž je dosazena hodnota, kterou jsme specifikovali v modelu pomocí atributu Display*.*

#### Vstupní pole (<input> tag)

Jedná se o klasické vstupní pole, které se váže k určité vlastnosti v modelu. To, ke které vlastnosti se váže, určuje hodnota atributu asp-for.

#### Validační zpráva (<span> tag)

Tato zpráva se zobrazí, pokud data formuláře po odeslání neprojdou validací. Text validační zprávy je specifikován pomocí atributu v modelu. Stejně jako u vstupního pole a jeho popisku, musíme specifikovat, ke které vlastnosti modelu se váže. To se určuje pomocí atributu asp-validation-for.

* 1. Dekompozice softwarové logiky aplikace

Jedná se o způsob rozdělení našeho projektu do více vrstev. Tyto vrstvy si můžeme představit jako jednotlivé projekty z nichž každý zastává pouze a jen svoji specifickou funkci.

* + 1. Hlavní důvody pro dekompozici aplikace

Správná dekompozice naší aplikace je kritická, protože eliminuje zbytečné komplikace při vývoji. Zpravidla platí, že čím rozsáhlejší aplikace je, tím robustnější musí být její struktura. Avšak, není na škodu se řádně nad strukturou zamyslet i u menších projektů, protože nikdy nemůžete s jistotou vědět, do jaké míry budete svoji aplikaci v budoucnosti potřebovat rozšířit.

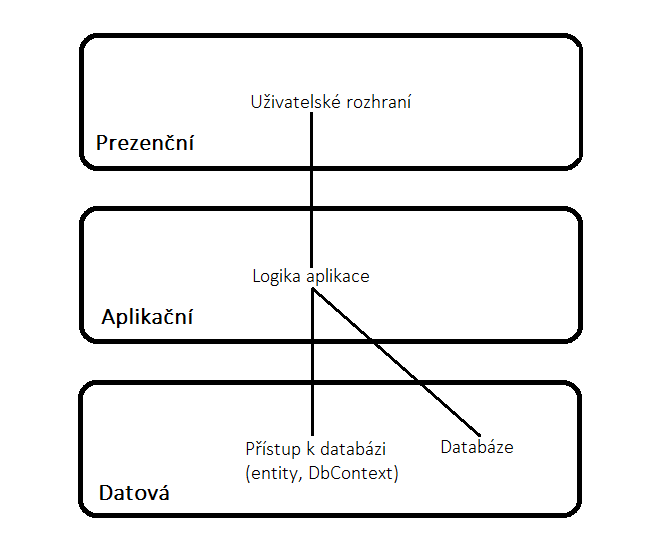
Špatná struktura aplikace může pro pozdější fázi vývoje znamenat její drastické zpomalení. Serióznost tohoto problému pak přímo úměrně stoupá s velikostí aplikace. Například, velké firmy mohou z důvodu špatné dekompozice softwarové logiky jejich projektu ztratit mnoho času a peněz, což často může vést i k bankrotu.

* + 1. Dekompozice aplikace na 3 vrstvy

Je mnoho způsobů, jak aplikaci rozdělit. Každý z těchto způsobů sebou přináší své výhody i nevýhody. My si popíšeme jeden z těchto způsobů, který spočívá v rozdělení projektu do 3 vrstev, tedy prezenční, aplikační a datové.

#### Schéma dekompozice

V následujícím obrázku můžete vidět, jak spolu jednotlivé vrstvy komunikují. Prezenční vrstva využívá aplikační vrstvu, která využívá datovou vrstvu. Tedy, prezenční vrstva ví o aplikační vrstvě, ale už neví o datové vrstvě.



Obrázek 24 Dekompozice aplikace na 3 vrstvy

#### Prezenční vrstva

Prezenční vrstva poskytuje uživatelovi uživatelské rozhraní. Toto rozhraní uživatelovi umožňuje zobrazovat záznamy z databáze, vytvářet je a upravovat. Prezenční vrstva komunikuje s aplikační vrstvou. Projekt sloužící jako prezenční vrstva (webový projekt) tak musí mít referenci na projekt aplikační vrstvy (knihovna).

#### Aplikační vrstva

Aplikační vrstva obsahuje většinu logiky naší aplikace. V podstatě se stará o komunikaci mezi prezenční a datovou vrstvou. Zpracovává požadavky prezenční vrstvy a validuje je. Pokud projdou tyto požadavky validací, tak provede požadovanou operaci. Projekt aplikační vrstvy obsahuje referenci na projekt datové vrstvy.

Jako příklad může sloužit třeba vyhledání záznamu v databázi podle hodnoty, kterou zadal uživatel. Pokud hodnota splní dané požadavky, tak aplikační vrstva vytáhne data z databáze, entity (záznamy v podobě objektů) převede na objekty jiného typu a vrátí je prezenční vrstvě. Objekty tohoto typu pak využije prezenční vrstva k naplnění modelů.

#### Datová vrstva

Datová vrstva slouží pro práci s databází. Její funkcí je převážně načítat, vkládat a upravovat databázové záznamy. Tato vrstva jako jediná nemá referenci na žádný jiný projekt, protože se nachází v nejspodnější části hierarchie.

* 1. Entity Framework Core

Entity Framework Core je framework zajišťující automatickou konverzi dat mezi relační databází a naším projektem. V podstatě se jedná o nástroj, který nám umožňuje pracovat s databází pomocí objektů v jazyce C#. Objekty, do kterých se databázové záznamy převádějí, nazýváme entity.

Ke komunikaci s databází nám slouží 1 objekt, tzv. DbContext, který obsahuje tzv. DbSets. Každý DbSet pak vlastně představuje 1 tabulku v databázi.

Existuje několik přístupů při využívání technologie Entity Framework Core. Hlavní tři jsou Model First, Database First a Code First.

* + 1. Model first

Při vytváření databáze tímto způsobem se nejdříve vytvoří model, ze kterého se pak automaticky vygenerují SQL scripty. Pomocí těchto skriptů se pak vytvoří nebo upraví databáze a Entity Framework Core vygeneruje entity odpovídající struktuře databáze. Model (diagram) databáze se dá navrhnout pomocí mnoha různých nástrojů. Visual Studio obsahuje nástroje na vytváření a úpravu databázových diagramů XML-based DataSet Schema (XSD) a Entity Designer Model XML visual interface (EDMX).

Výhody

1. návrh databáze pomocí intuitivního grafického rozhraní
2. není nutné psát žádný kód

Nevýhody

1. omezená kontrola nad kódem souvisejícím s databází
   * 1. Database first

Database first přístup je částečně podobný přístupu Model first. Na rozdíl od Model first přístupu však vše začíná u vytvoření databáze. Z databáze se pak automaticky vytvoří diagram, ze kterého se vygenerují entity. Tento přístup se nejvíce vyplatí, když potřebujeme pracovat s již existující databází.

Výhody

1. není nutné psát žádný kód, pokud chceme pracovat s již existující databází
2. plná kontrola nad databází

Nevýhody

1. ještě nižší kontrola nad kódem souvisejícím s databází než u Model first přístupu
2. úprava databáze vyžaduje pokročilé znalosti jazyka SQL
   * 1. Code first

Jedná se o opak Model first a Database first přístupů. Vše začíná vytvořením tříd (entit) ze kterých pak Entity Framework Core vygeneruje databázi. Vzhledem k tomu, že si při využití tohoto přístupu píšeme třídy entit sami, není nutné vytvářet ani vygenerovat databázový diagram.

Pokročilejší nastavení databáze se pak dá nastavit také přímo v C# kódu, pomocí tzv. Fluent Api.

Výhody

1. nevyužívá se databázový diagram
2. pro pokročilejší nastavení databáze je k dispozici Fluent Api

Nevýhody

1. nutnost pokročilých znalostí o technologii Entity Framework Core a Fluent Api
   1. Dependency Injection

Při psaní kódu je důležité dbát na to, aby byl kód robustní, přehledný a lehce upravitelný. Je tedy důležité si důkladně promyslet strukturu projektu a předejít tak nežádoucím komplikacím. Nad programem bychom neměli přemýšlet pouze jako nad celkem, ale spíše jako nad propojením mnoha malých celků (komponent).

Co ale dělat, když se rozhodneme jednu z těchto komponent nahradit? Obvykle to pro programátora znamená najít všechna místa, kde se daná komponenta využívá a ručně kód upravit. Takto zdánlivě jednoduchá úprava je pak u velkých projektů často velmi časově náročná.

Jako řešení tohoto problému přichází tzv. vkládaní závislostí, což je český překlad pro anglický termín „dependency injection“. V podstatě se jedná o technologii, která zjednodušuje záměnu jednotlivých komponent.

* + 1. Vkládání závislostí v ASP.NET Core

Komunita vývojářů ASP.NET Core aplikací se shodla na tom, že chce, aby bylo vkládaní závislostí implementováno v každé aplikaci již od základu. Jedná se tedy o kritickou část každé ASP.NET Core aplikace.

O vkládání závislostí se stará tzv. IoC Container (Inversion Of Control Container), který spravuje kolekci služeb. Každá služba je vlastně objekt, který lze vyžádat v jakémkoliv místě aplikace.

Za zmínku také stojí, že společnost Microsoft se IoC Container rozhodla pojmenovat jakožto „Service Provider“. Pod tímto názvem k této technologii na internetu referuje většina článků a dokumentací.

* + 1. Způsoby registrace služeb

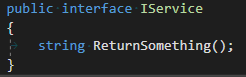
Služby se dají zaregistrovat třemi způsoby

1. Singleton
   1. pouze jedna instance služby vytvořená při spuštění aplikace
   2. nejméně náročné na výkon
   3. pro případy, kdy uživatel nepotřebuje vlastní instanci služby
2. Scoped
   1. rozdílná instance služby pro každého uživatele
   2. středně náročné na výkon
3. Transient
   1. každý nový dotaz na server vytvoří novou instanci služby
   2. nejvíce flexibilní
   3. nejvíce náročné na výkon
      1. Praktická ukázka

Postup

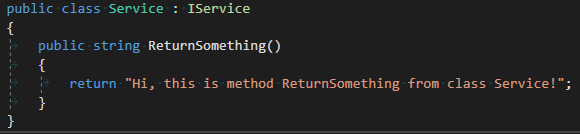
1. vytvoření rozhraní služby (interface)
2. vytvoření služby implementující rozhraní
3. registrace služby do Service Provider
4. vyžádání služby

#### Vytvoření rozhraní služby



Obrázek 25 Vytvoření rozhraní služby

#### Vytvoření implementace rozhraní

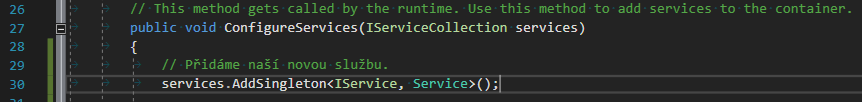


Obrázek 26 Vytvoření implementace rozhraní

#### Registrace služby

Službu zaregistrujeme jako Singleton. Jak můžete na následujícím obrázku vidět, funkce pro přidání služby vyžaduje 2 generické parametry. První z nich je rozhraní služby a druhý je implementace tohoto rozhraní, tedy konkrétní třída.

Důvod, proč se služby přidávají jakožto rozhraní, je prostý. Všude v kódu, kde budeme danou službu vyžadovat, vlastně budeme vyžadovat pouze její rozhraní, což znamená, že můžeme implementaci zaměnit, kdykoliv budeme chtít a všechno bude fungovat tak, jak má.



Obrázek 27 Registrace služby do Service Provideru

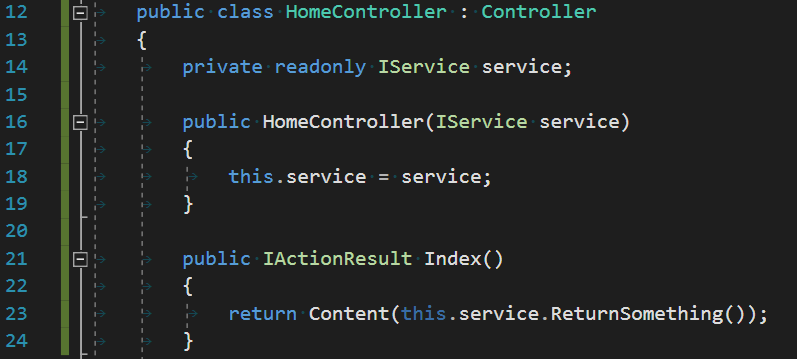
#### Vyžádání služby

Způsobů, jak službu vyžádat, je hned několik. Ani o jednom z nich se nedá říct, že by byl tím nejlepším. Každý z nich se hodí pro jiné využití.

Způsoby vyžádání služby

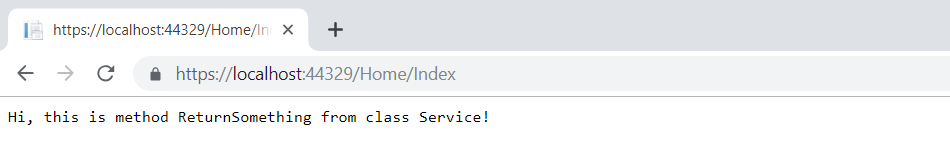
1. předání služby do konstruktoru třídy
2. předání ServiceProvider objektu do konstruktoru třídy
3. vyžádání služby v Razor šabloně pomocí příkazu @inject

Nejvyužívanějším způsobem vyžádání služby je její vyžádání přímo v konstruktoru třídy. Toho docílíme tím, že ji budeme vyžadovat jakožto parametr. Na následujícím obrázku můžete vidět, jak to vypadá v praxi.



Obrázek 28 Vyžádání služby v konstruktoru třídy

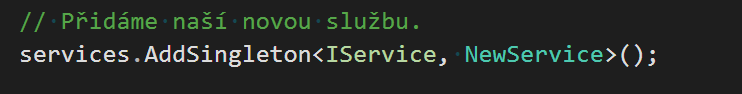
Akce Index v řadiči Home by měla vracet textový řetězec „Hi, this is method ReturnSomething from class Service!“.



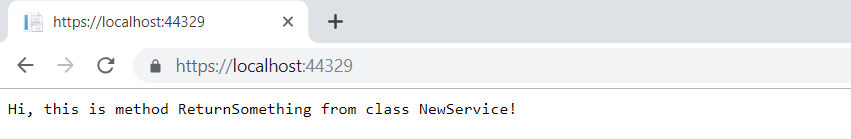
Obrázek 29 Otestování první služby

#### Záměna implementace služby

Záměna implementace služby je opravdu velmi jednoduchá. Stačí pouze nahradit implementaci třídy ve Startup třídě. V našem případě tedy nahradíme třídu Service třídou NewService.



Obrázek 30 Záměna služby



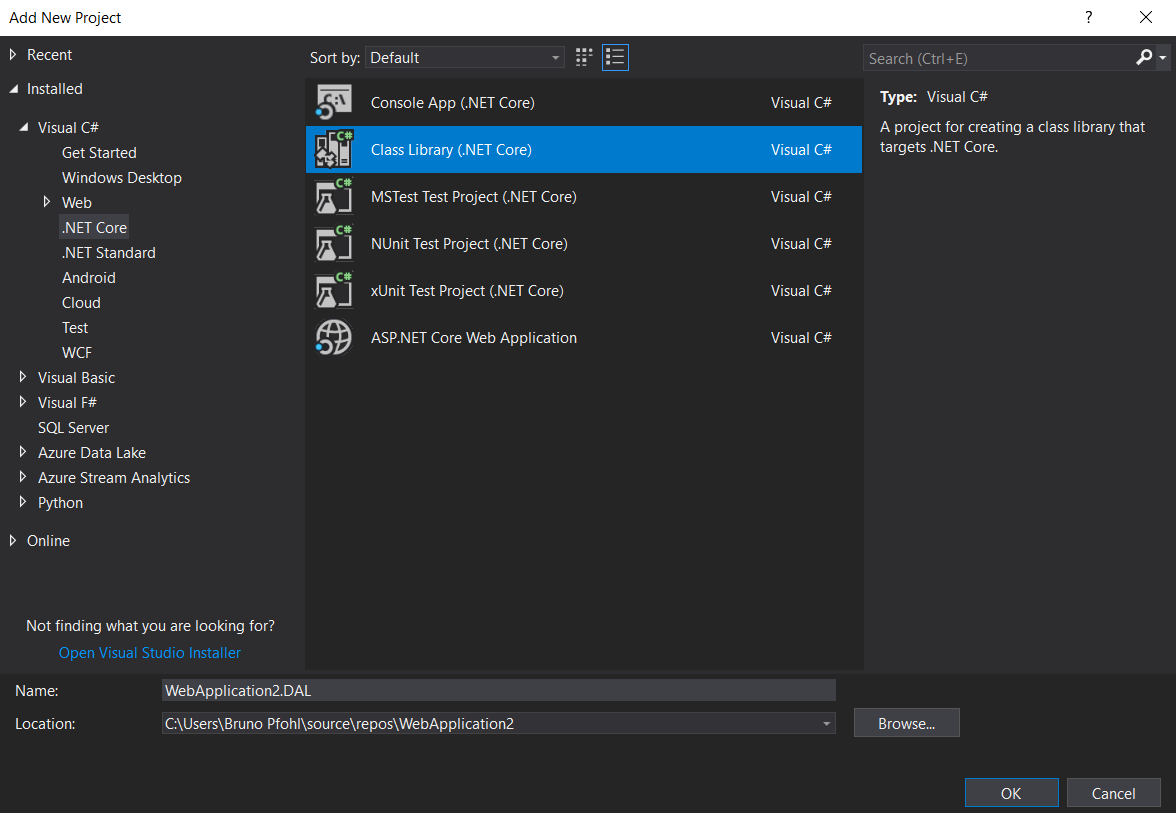
Obrázek 31 Otestování záměny služby

* 1. Tvorba databáze pomocí Code First přístupu

Tato kapitola se zabývá tvorbou jednoduché databáze. Databáze bude obsahovat tabulky pro uložení informací o autech a automobilkách. Kromě těchto tabulek ještě bude databáze obsahovat tabulky potřebné pro registraci a přihlašování uživatelů, k čemuž bude vyžita technologie Identity.

* + 1. Vytvoření projektu

Pro kód související s databází si vytvoříme nový projekt typu Class Library (.NET Core). Toto oddělení vyplývá ze základních principů dekompozice softwarové logiky, kterou jsme si teoreticky popsali v 9. podkapitole 1. kapitoly. Pokud jste ji nečetli, doporučuji se k ní vrátit a až poté pokračovat ve studiu této kapitoly.



Obrázek 32 Založení projektu datové vrstvy

* + 1. Instalace potřebných balíčků

Dalším krokem je nainstalovat potřebné balíčky pomocí Nuget Package manageru.

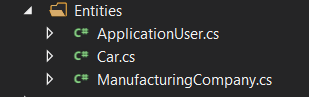
Postup při instalaci

1. kliknout pravým tlačítkem na projekt
2. zvolit „Manage Nuget packages“
3. vyhledat potřebný balíček
4. zvolit požadovanou verzi
5. kliknout na tlačítko „install“

Seznam potřebných balíčků

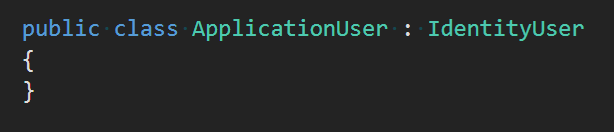
* Microsoft.AspNetCore.Identity.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.Design
* Microsoft.EntityFrameworkCore.Proxies
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.NETCore.App
  + 1. Vytvoření entit

Vytvoříme si v kořenovém adresáři složku „Entities“. Dle konvence by se všechny entity měly nacházet právě v této složce.



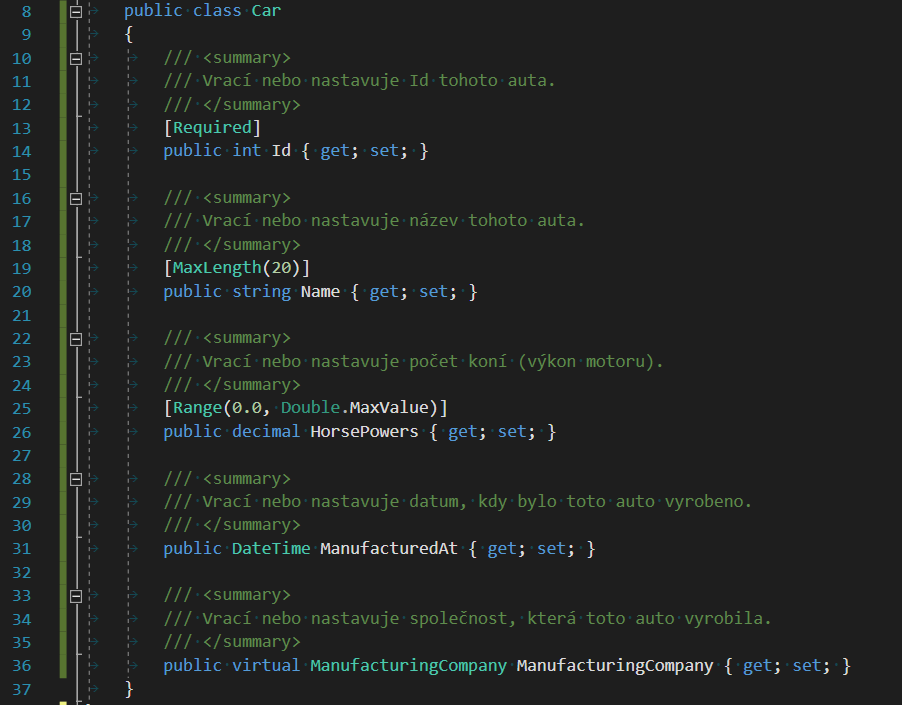
Obrázek 33 Adresářová struktura datové vrstvy

#### Entita uživatele



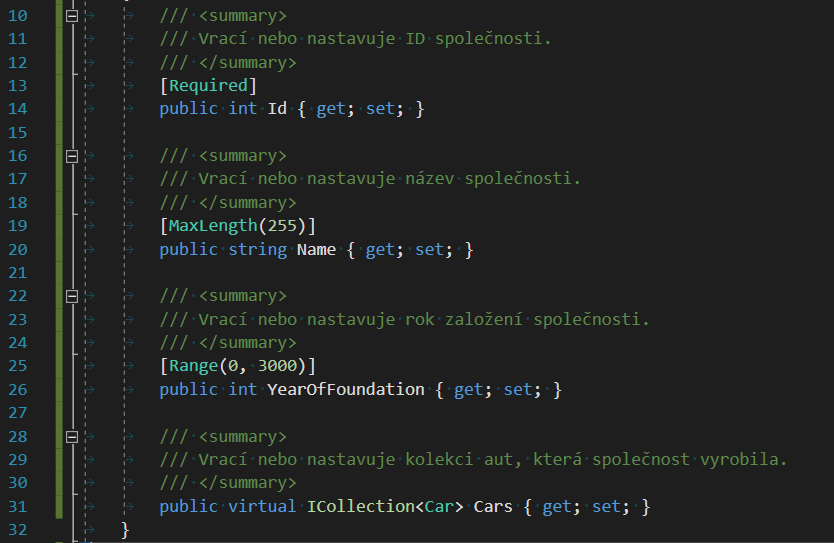
Obrázek 34 Entita uživatele

#### Entita auta



Obrázek 35 Entita auta

#### Entita společnosti vyrábějící auta



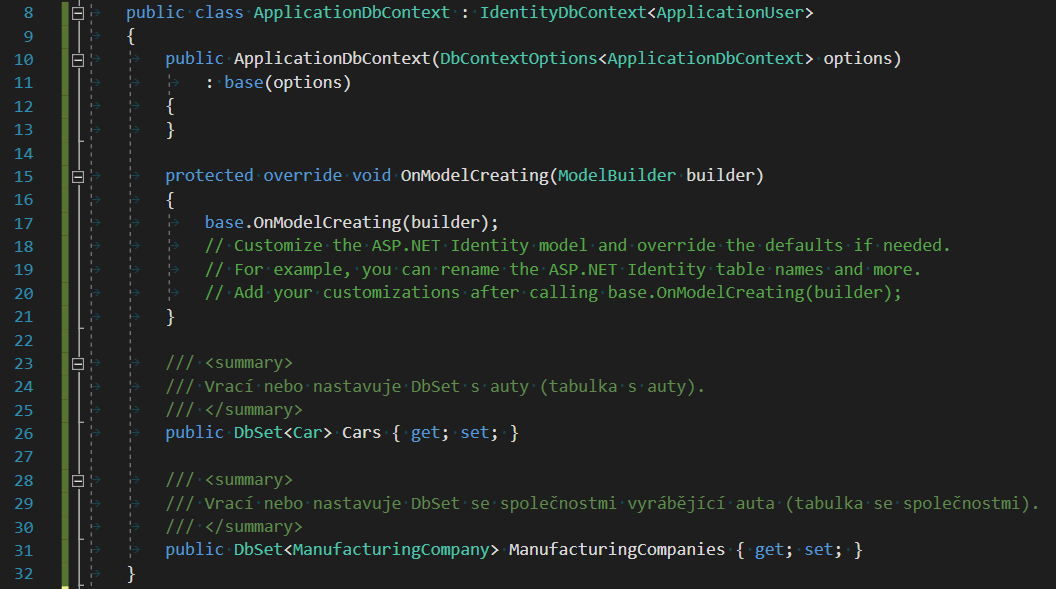
Obrázek 36 Entita společnosti vyrábějící auta

* + 1. Vytvoření DbContext

Abychom mohli bez problému komunikovat s databází musíme si vytvořit tzv. „DbContext“ (Microsoft.EntityFrameworkCore.DbContext). Tento objekt slouží ke komunikaci s databází. Můžeme tak díky němu odesílat dotazy databázi k načtení, smazání, vytvoření a upravení záznamů. Záznamy, které nám databáze vrátí, jsou pak pomocí Entity Framework Core převedeny na entity, se kterými můžeme dále pracovat.

Vzhledem k tomu, že požadujeme, aby aplikace obsahovala i tabulky potřebné pro správu uživatelů, nevyužijeme přímo třídu „DbContext“, ale třídu „IdentityDbContext“, která se nachází v jmenném prostoru Microsoft.AspNetCore.Identity.EntityFrameworkCore.

Vytvoříme si tedy třídu ApplicationDbContext v kořenové složce projektu obsahující následující kód, který je zachycen na následujícím obrázku.



Obrázek 37 Třída ApplicationDbContext

Konstruktor třídy přejímá parametr typu DbContextOptions, který obsahuje nastavení. Toto nastavení může například obsahovat tzv. „connection string“, což je textový řetězec sloužící ke specifikaci připojení k databázi.

Metoda OnModelCreating slouží také ke specifikaci nastavení. Právě v této metodě můžete využít tzv. Fluent Api například k explicitnímu přejmenování tabulek, určení vztahů mezi tabulkami atd.

Na předchozím obrázku můžete ještě vidět, že třída ApplicationDbContext má 2 vlastnosti typu DbSet. Tyto vlastnosti pro nás vlastně představují jednotlivé tabulky v naší databázi.

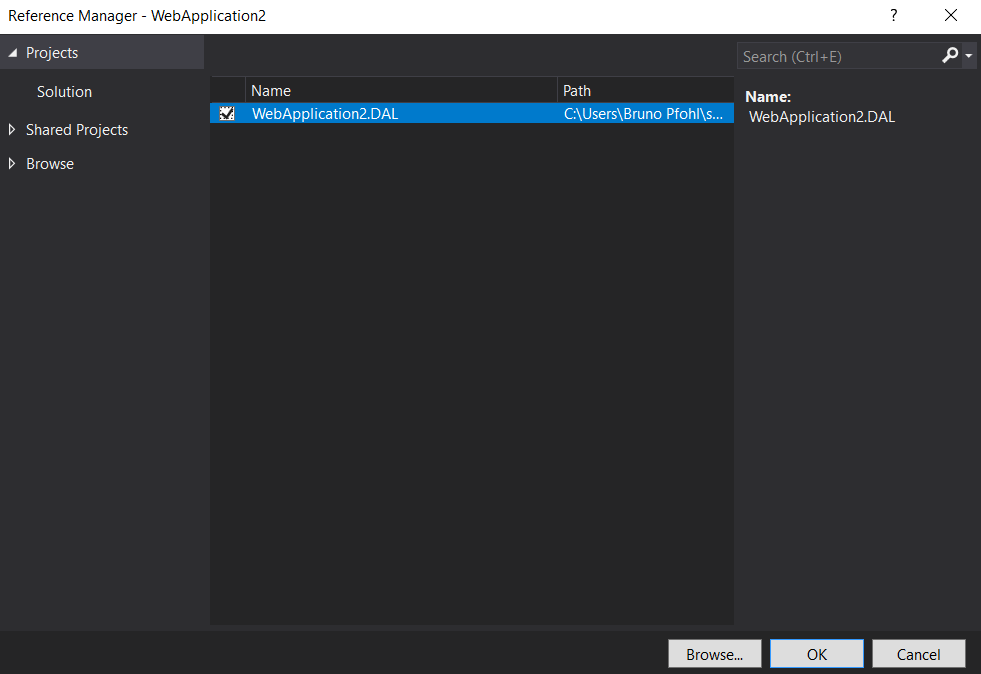
* + 1. Využití datové vrstvy v prezenční vrstvě

Abychom mohli naší datové vrstvy využít, musíme na ní v projektu prezenční vrstvy přidat referenci. Dále musíme ve třídě Startup přidat ApplicationDbContext jako službu, k čemuž je nutné do projektu prezenční vrstvy přidat potřebné balíčky pomocí nástroje Nuget Package Manager.

#### *Přidání reference na datovou vrstvu*

Postup

1. kliknout pravým tlačítkem na projekt prezenční vrstvy v okně „Solution Explorer“
2. kliknout na Add
3. New reference
4. Zaškrtnout v dialogovém okně projekt datové vrstvy
5. Potvrdit kliknutím na tlačítko OK



Obrázek 38 Přidání reference na datovou vrstvu

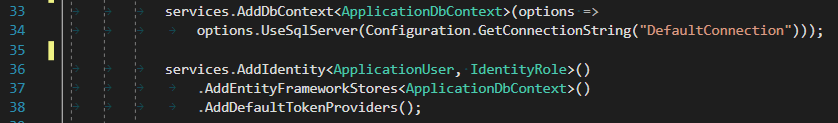
#### *Přidání potřebných balíčků*

Potřebné balíčky

* Microsoft.AspNetCore.All
* Microsoft.EntityFrameworkCore.Tools
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.NETCore.App

#### *Registrace a konfigurace DbContext ve Startup třídě*

Abychom mohli datovou vrstvu využívat, musíme přidat DbContext do ServiceProvideru. Příkaz, pomocí kterého toho lze docílit, můžete vidět na následujícím obrázku (řádek 33). Dále je ještě nutné přidat službu pro registraci a přihlašování uživatelů (řádek 36).



Obrázek 39 Registrace a konfigurace DbContext ve Startup třídě

Na obrázku můžeme vidět, že metodě UseSqlServer předáváme jako parametr hodnotu z konfiguračního souboru appsettings.json. Konkrétně se jedná o connection string.

Connection string je u všech začátečníků v technologi ASP zdrojem mnohé kontroverze. Pokud si nejste jisti, jak connection string sestavit, doporučuji se podívat na oficiální dokumentaci od společnosti Microsoft, kterou naleznete na následující adrese.

**Odkaz na oficiální dokumentaci**

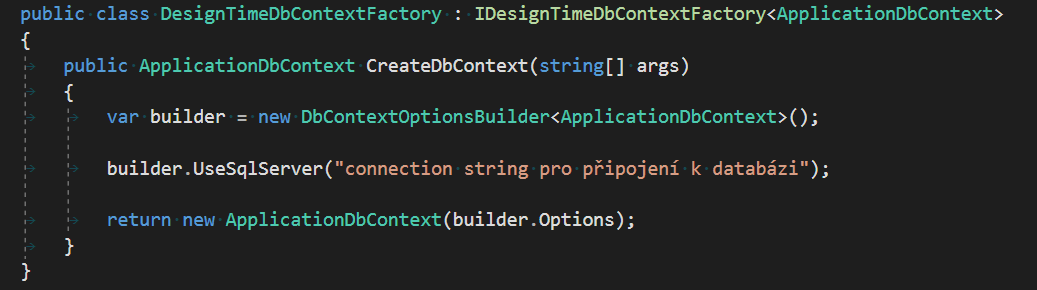
https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/connection-string-syntax

* + 1. Databázové migrace

Entity Framework nám umožňuje verzování struktury databáze. Každá úprava databáze je tedy uložena jako databázová migrace a v případě potřeby lze převést databázi na požadovanou verzi.

#### Třída DesignTimeDbContextFactory

Tato třída slouží pouze pro zprovoznění databázových migrací. Jejím účelem je poskytnout instanci třídy ApplicationDbContext, aby při uplatnění migrací mohlo vzniknout spojení s databází.

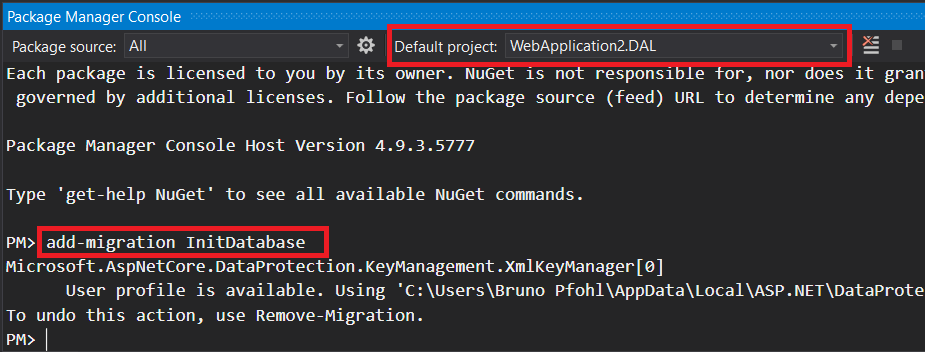


Obrázek 40 Třída DesignTimeDbContextFactory

#### Přidání nové migrace

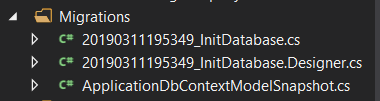
Postup

1. otevřít nástroj Package Manager Console
2. zvolit projekt datové vrstvy
3. přidat migraci pomocí příkazu add-migration



Obrázek 41 Přidání nové migrace

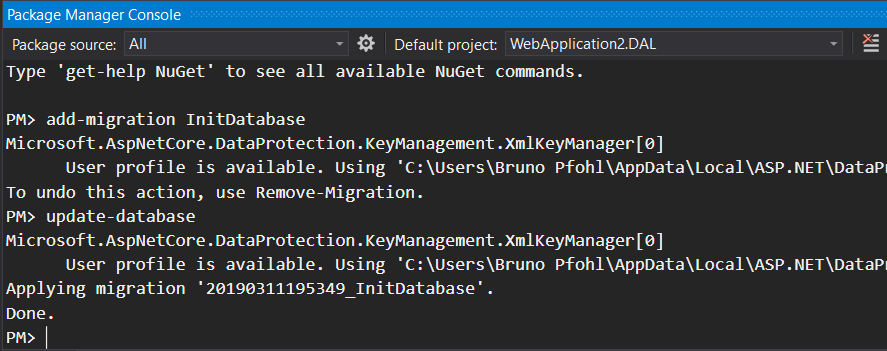
Po spuštění příkazu se do kořenové složky projektu datové vrstvy přidá nová složka Migrations obsahující nově vytvořenou migraci.



Obrázek 42 Složka Migrations

#### Provedení migrace

Pro přidání nové migrace už stačí pouze danou migraci provést. Postup je stejný jako při přidávání migrací, jen s rozdílem, že příkaz add-migration musíme nahradit příkazem update-database.



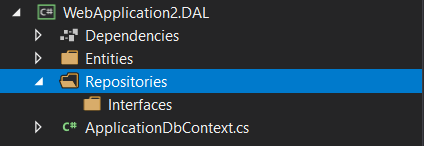
Obrázek 43 Provedení migrace

* 1. Repository Pattern a Unit Of Work

Jedná se o abstrakci mezi aplikační a datovou vrstvou aplikace. V podstatě je to nadstavba nad objektem DbContext, která nám usnadňuje používání databáze. Repozitář (anglicky repository) je vlastně objekt, který slouží k práci s určitou tabulkou v databázi. Unit Of Work pak slouží jako jednotka obsahující všechny repozitáře.

* + 1. Adresářová struktura

Hlavní výhodou, kterou nám Repository Pattern přináší, je právě možnost psát robustnější a přehlednější kód. Abychom toho docílili, je nutné dodržovat předem stanovené konvence. Začneme tedy tím, že si v kořenovém adresáři projektu vytvoříme složku „Repositories“ a v ní vytvoříme podsložku „Interfaces“. Tuto strukturu projektu můžete vidět na následujícím obrázku.



Obrázek 44 Adresářová struktura u Repository Pattern

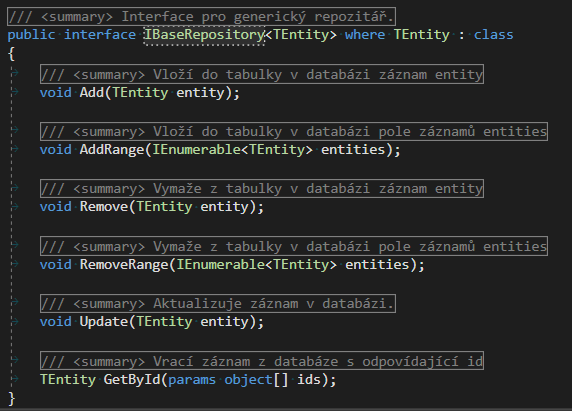
* + 1. Vytvoření generického repozitáře

Všechny repozitáře dědí z jednoho bázového repozitáře. Tento bázový repozitář obsahuje metody, které jsou potřeba u všech ostatních repozitářů. Množství těchto společných metod závisí na konkrétním projektu. V každém případě se však jedná přinejmenším o metody pro přidávání a odebírání dat.

#### Vytvoření rozhraní generického repozitáře

Dle konvence musíme při vytváření každého repositáře začít nejdříve vytvořením jeho rozhraní. V tomto rozhraní specifikujeme, jaké metody musí náš repositář mít. Navíc, tento přístup koresponduje s ideologií ASP.NET Core o vkládání závislostí. Tj., pro využití vkládání závislostí se nám hodí, abychom měli rozhraní.

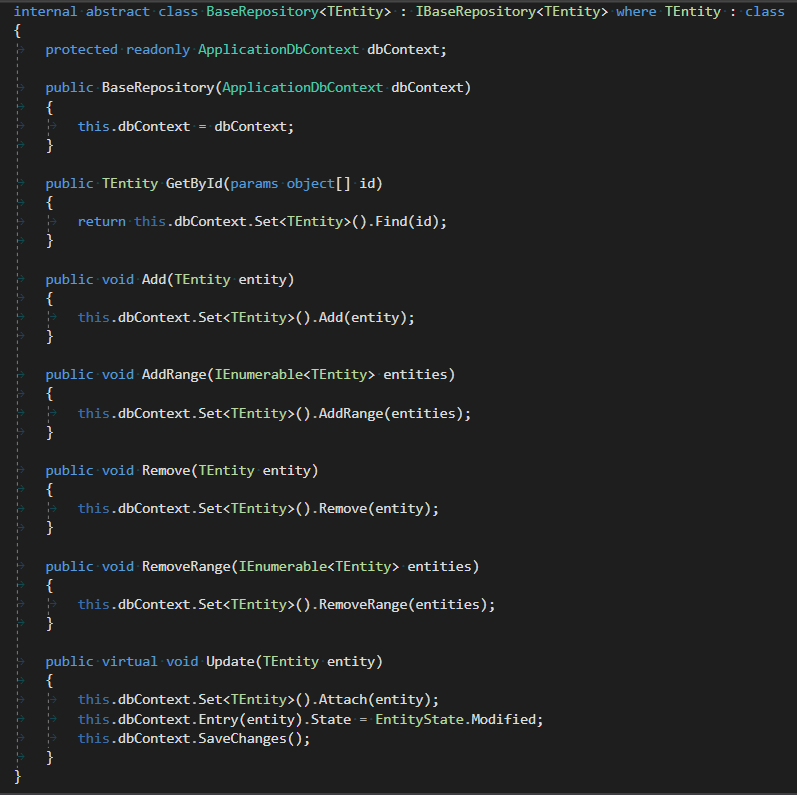
Bázový repozitář přijímá generický parametr TEntity. Tento generický parametr pak určuje, s jakou tabulkou v databázi bude repozitář pracovat.



Obrázek 45 Rozhraní bázového repozitáře

#### Implementace rozhraní generického repozitáře

Na následujícím obrázku můžete vidět kód třídy bázového repozitáře. Všechny metody využívají generického parametru TEntity a přistupují tak k příslušné tabulce v databázi. Např., pokud předáme repozitáři jako generický parametr entitu Car, bude repozitář pracovat s databázovou tabulkou aut.



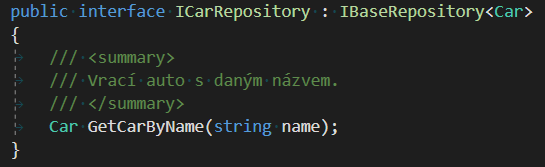
Obrázek 46 Implementace rozhraní bázového repozitáře

* + 1. Vytvoření repozitáře pro tabulku aut

Tato podkapitole se zabývá vytvořením repozitáře pro práci s tabulkou aut. K vytvoření využívá bázového repozitáře z předchozí kapitoly.

#### Vytvoření rozhraní repozitáře aut

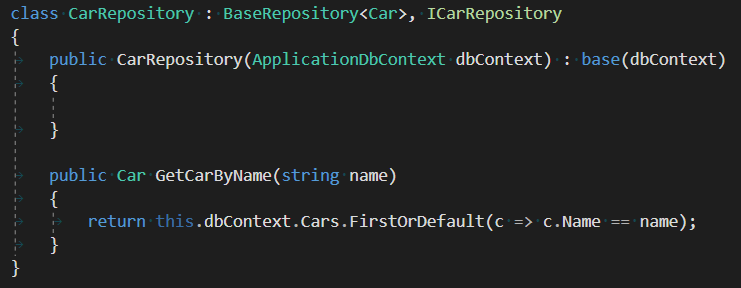
Rozhraní repozitáře pro tabulku aut dědí z rozhraní bázového repozitáře a přidává k jeho metodám ještě metodu GetCarByName.



Obrázek 47 Rozhraní repozitáře aut

#### Implementace rozhraní repozitáře aut

Třída repozitáře dědí z bázového repozitáře, kterému jako generický parametr předává entitu Car. Dále ještě obsahuje metodu GetCarByName, která vrací první auto z databáze, jehož název se shoduje s hodnotou parametru name. Na rozdíl od metod bázového repozitáře k tomu však ale nevyužívá generickou metodu Set<TEntity>.



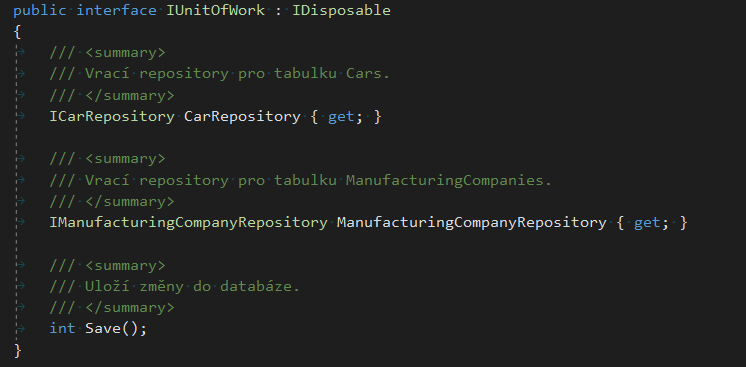
Obrázek 48 Implementace rozhraní repozitáře aut

* + 1. Vytvoření Unit of Work

Unit of Work je objekt, který obsahuje všechny repozitáře. V praxi se pak využívá pro komunikaci s databází právě tento objekt, což usnadňuje práci s databází a umožňuje psát přehlednější kód.

#### Vytvoření rozhraní pro Unit of Work

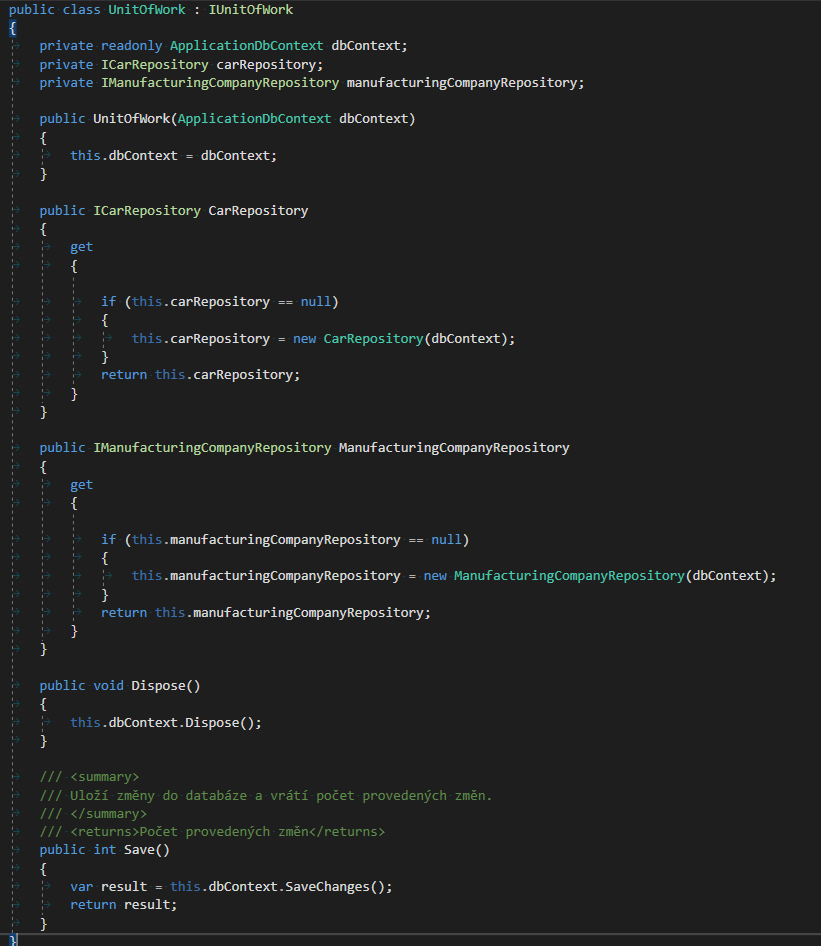
Rozhraní Unit of Work dědí z rozhraní IDisposable. To znamená, že třída implementující rozhraní IUnitOfWork musí obsahovat metodu Dispose. Tato metoda se volá například po skončení using bloku. Kód rozhraní můžete vidět na následujícím obrázku.



Obrázek 49 Vytvoření rozhraní pro Unit of Work

#### Implementace rozhraní Unit of Work

Konstruktor třídy implementující rozhraní IUnitOfWork přijímá jako parametr objekt typu ApplicationDbContext, který si pak ukládá do privátní proměnné. S touto instancí třídy ApplicationDbContext pak pracují všechny repozitáře. Parametr je konstruktoru předán pomocí vkládání závislostí.



Obrázek 50 Implementace rozhraní Unit Of Work

* + 1. Využití Unit Of Work v prezenční vrstvě

Abychom mohli Unit Of Work využít v prezenční vrstvě, musíme ho zaregistrovat jako službu ve třídě Startup. Službu zaregistrujeme jako Scoped, díky čemuž se vytvoří nová instance objektu pro každého uživatele.



Obrázek 51 Využití Unit Of Work v prezenční vrstvě

1. Tvorba ukázkové aplikace

Jako ukázkovou aplikaci jsem se rozhodnul vytvořit jednoduché internetové fórum, kam se uživatelé mohou zaregistrovat a přidávat příspěvky pod různé kategorie. Přidané příspěvky mohou uživatelé také komentovat.

Při tvorbě webové aplikace jsem kladl důraz převážně na jednoduchost a přehlednost kódu. Webová aplikace tedy nedisponuje mnoha funkcemi, což studentům umožňuje rychlejší orientaci v celém projektu. Tento přístup k věci se nakonec ukázal být správným a studenti tvrdili, že se v kódu aplikace dokáží vcelku jednoduše zorientovat.

* 1. Struktura projektu

Webová aplikace prošla v průběhu vývoje mnoha drastickými úpravami její struktury. Zpočátku jsem plánoval aplikaci rozdělit do 4 menších projektů, avšak po konzultaci s vedoucím mé maturitní práce se rozhodlo, že se projekt zásadně zjednoduší.

Finální verze webové aplikace se tedy skládá pouze ze 2 projektů, z nichž první zastupuje prezenční vrstvu a druhý zastupuje vrstvu datovou. Co se aplikační vrstvy týče, tu v aplikaci částečně zastává implementace Repository Pattern a UnitOfWork, jejichž princip je podrobně popsán ve studijních materiálech.

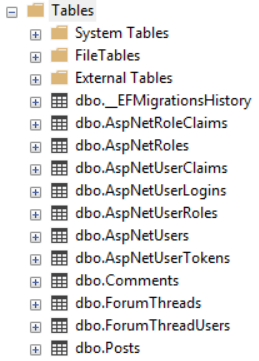
* + 1. Prezenční vrstva

Prezenční vrstvu jsem vytvořil pomocí kombinace Razor View šablon a implementace architektury MVC. Šablony jsem vytvářel s důrazem na to, aby co nejvíce odpovídaly studijním materiálům. Například, všechny formuláře jsou vytvořeny pomocí HTML Tag Helpers, jejichž vysvětlení je součástí studijních materiálů.

* + 1. Datová vrstva

Datovou vrstvu jsem vytvořil jakožto třídu (class library). Pro práci s databází jsem využil technologii Entity Framework Core, nad kterou jsem udělal nadstavbu v podobě implementace Repository Pattern a UnitOfWork.

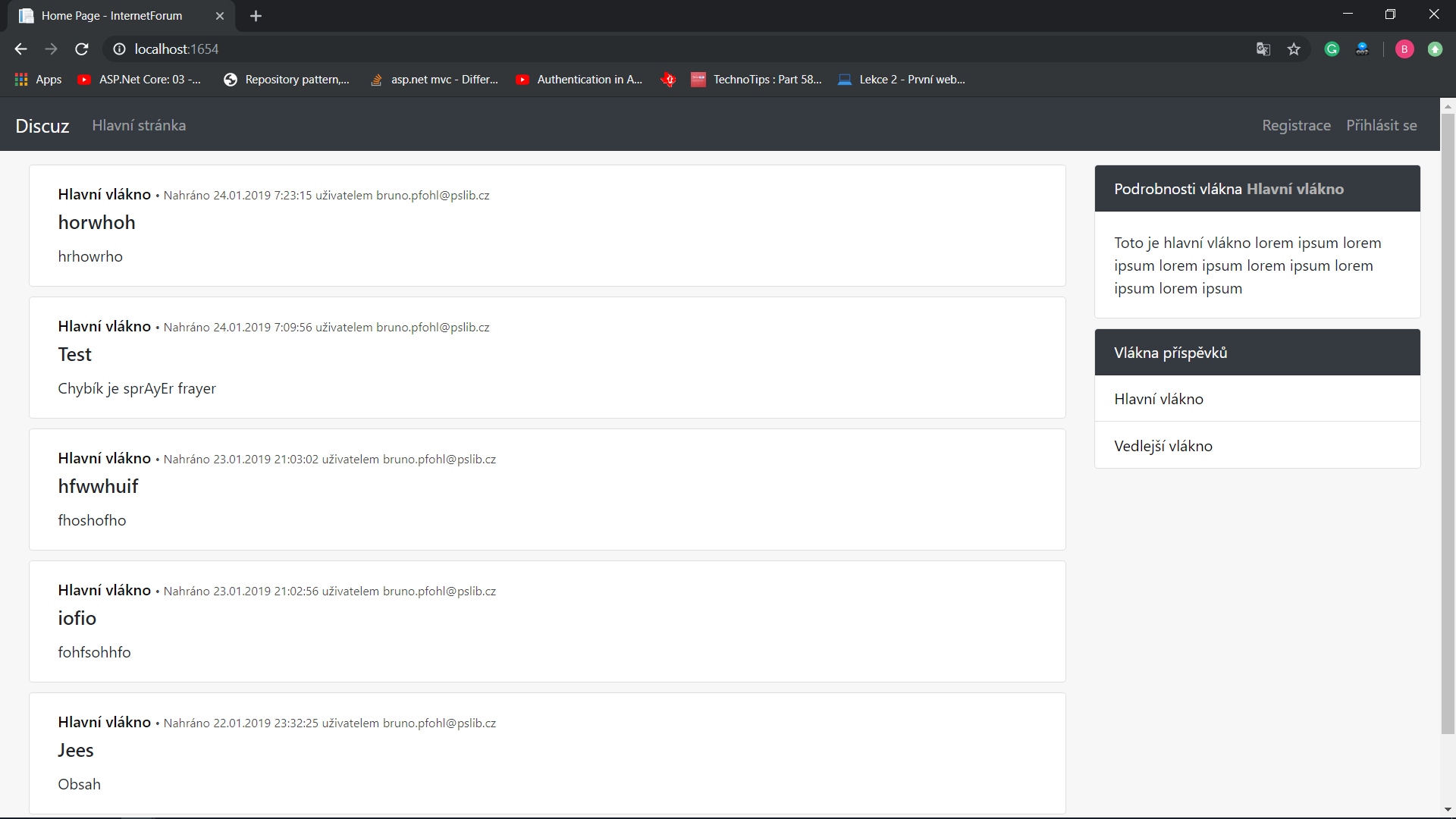
Databáze aplikace se skládá dohromady z 11 tabulek, z nichž 8 slouží pro správu uživatelů a zaznamenání historie databázových migrací. Zbylé 3 tabulky slouží pro uložení kategorií příspěvků, příspěvků a komentářů.



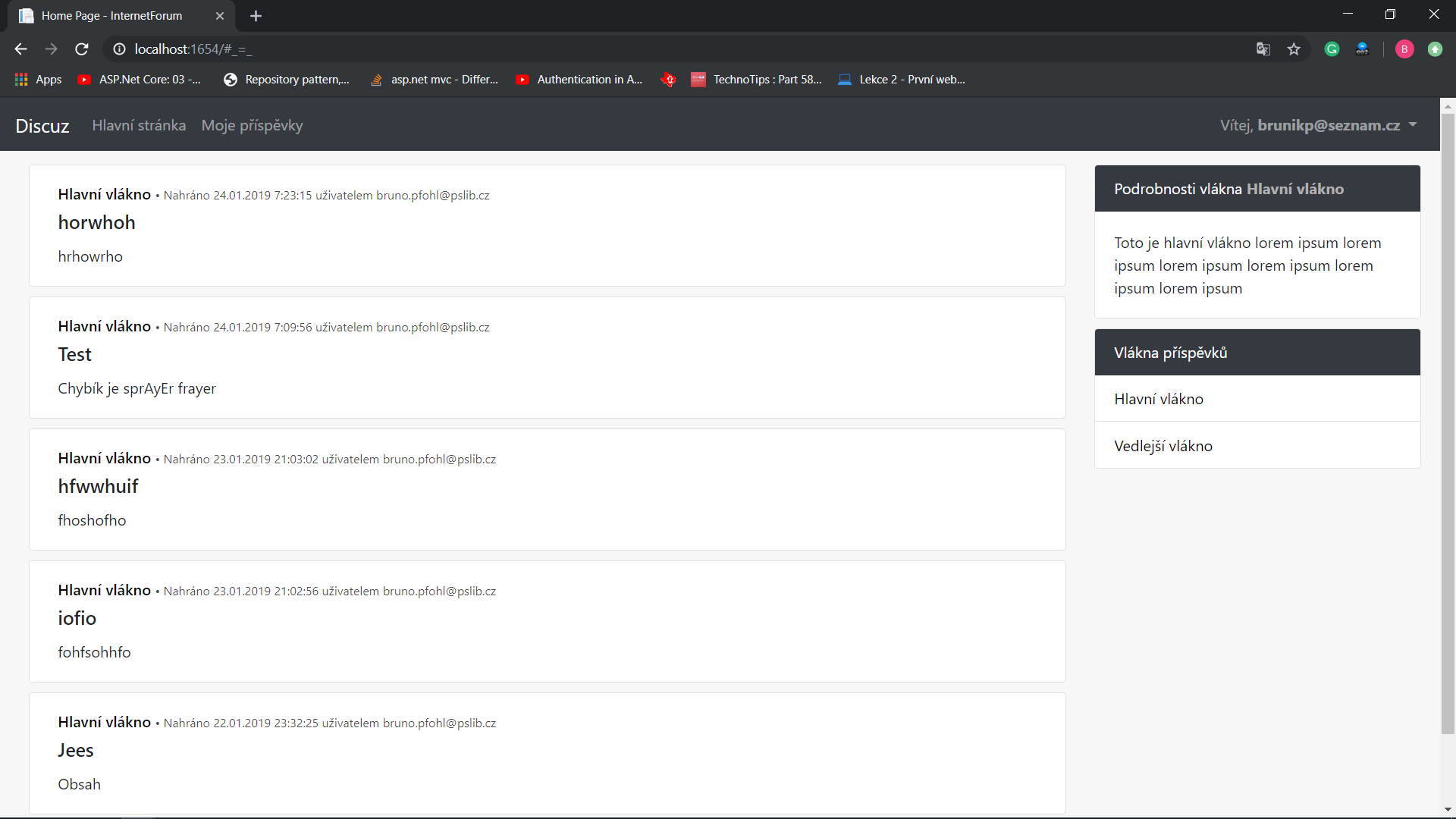
Obrázek 52 Struktura databáze

* 1. Vzhled aplikace

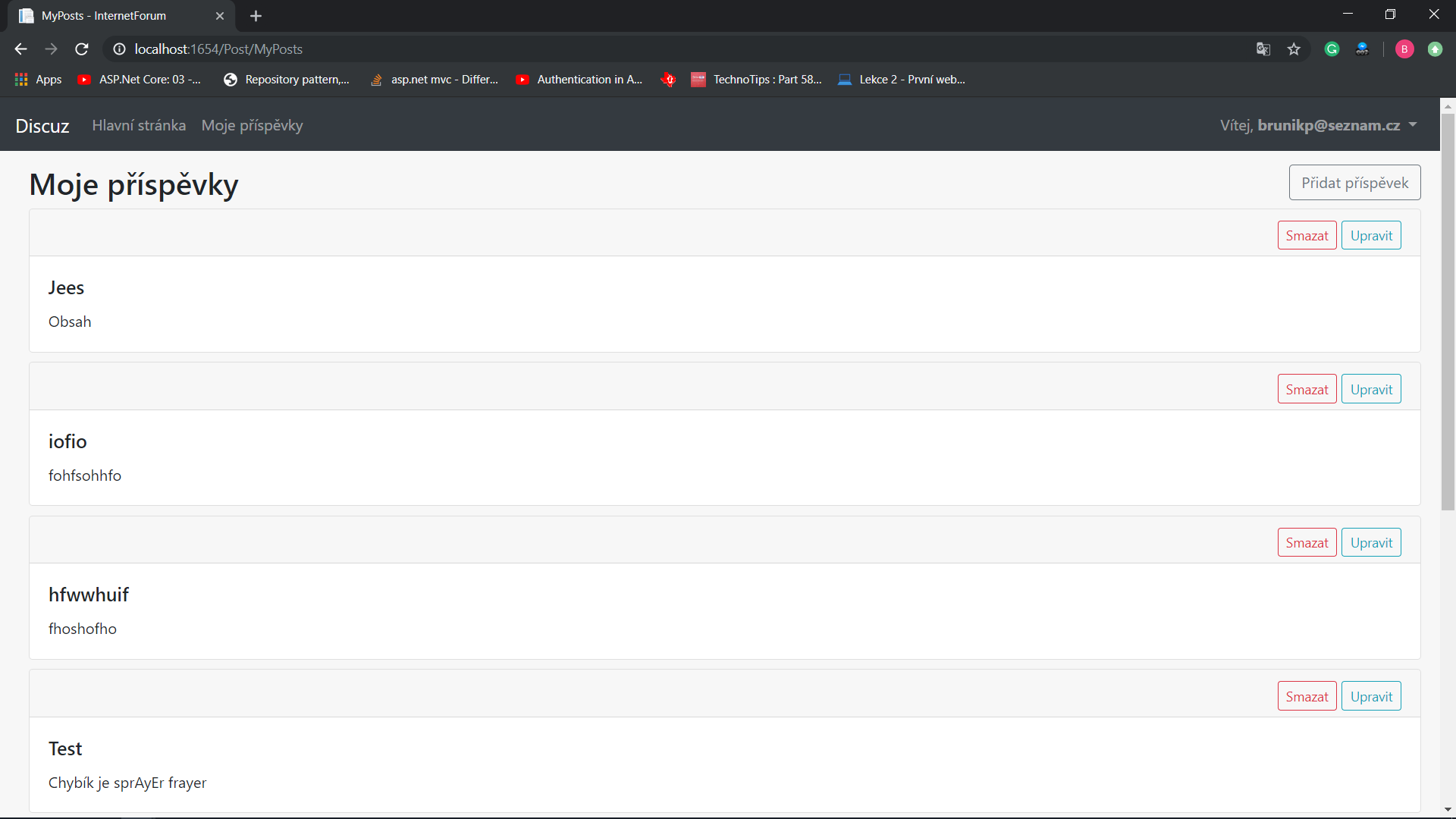
V této podkapitole se vyskytují pouze obrázky spuštěné aplikace. Obrázky zachycují pouze některé podstránky, které se dají považovat za relevantní.



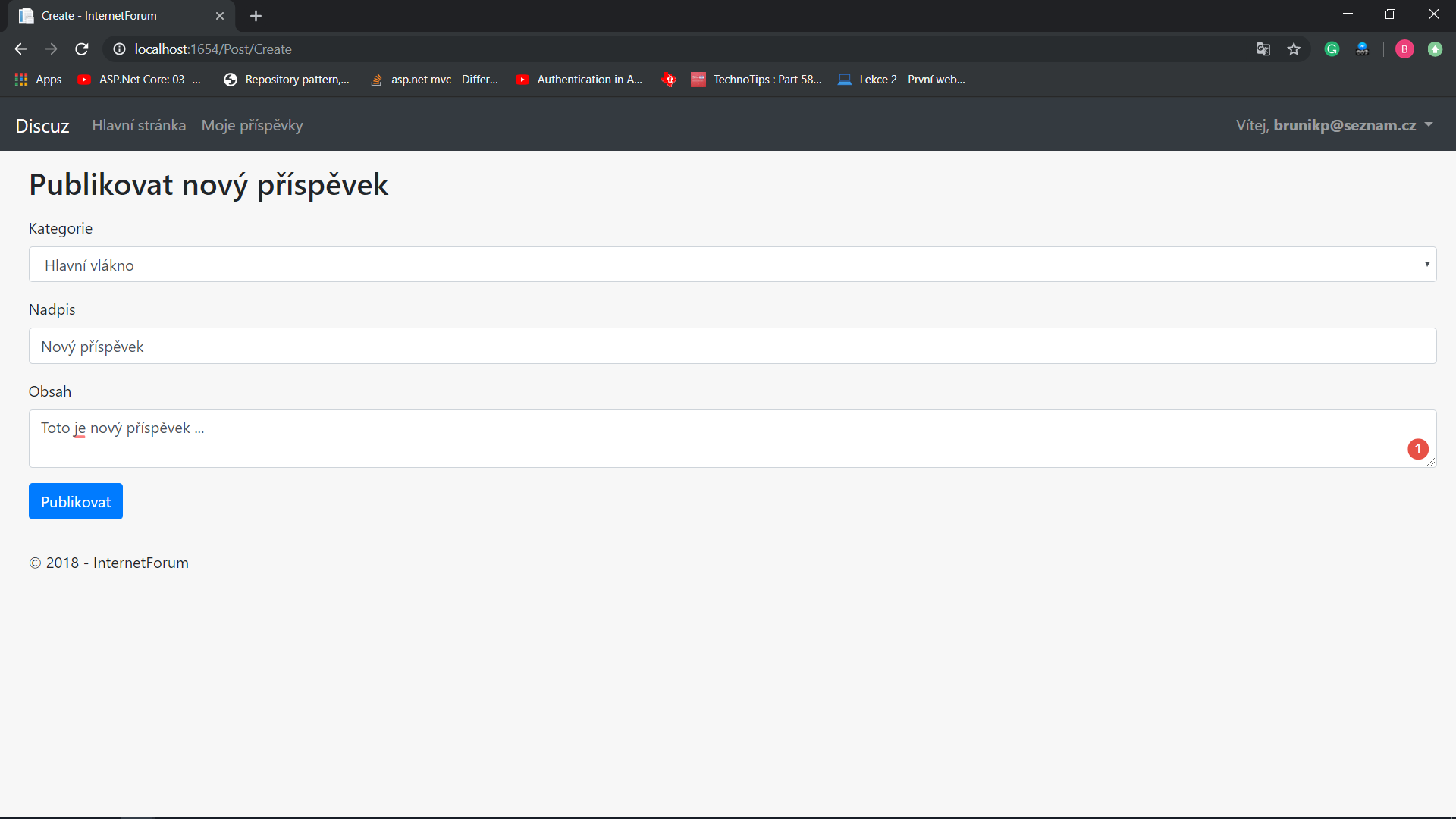
Obrázek 53 Pohled pro nepřihlášeného uživatele



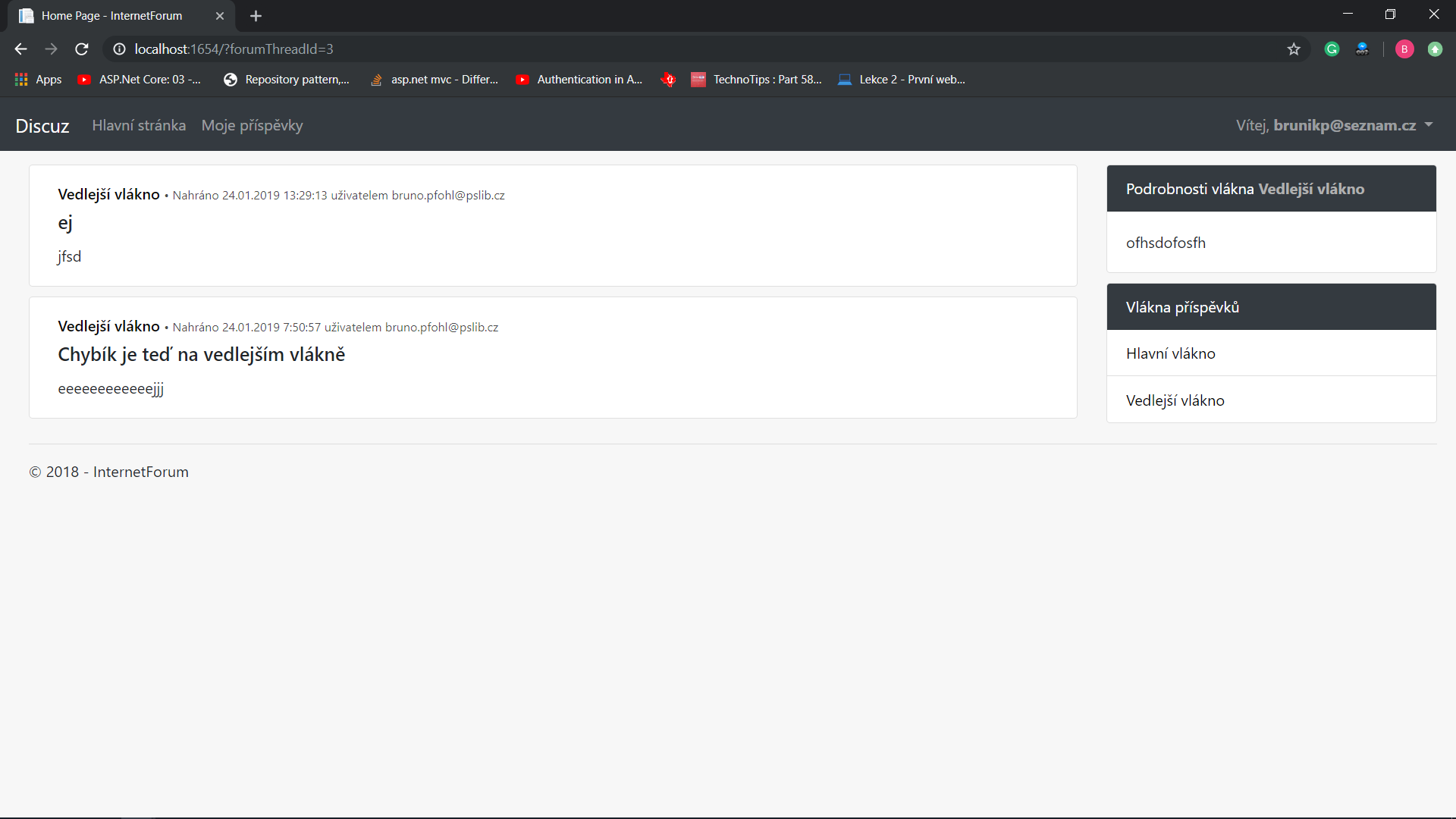
Obrázek 54 Pohled přihlášeného uživatele



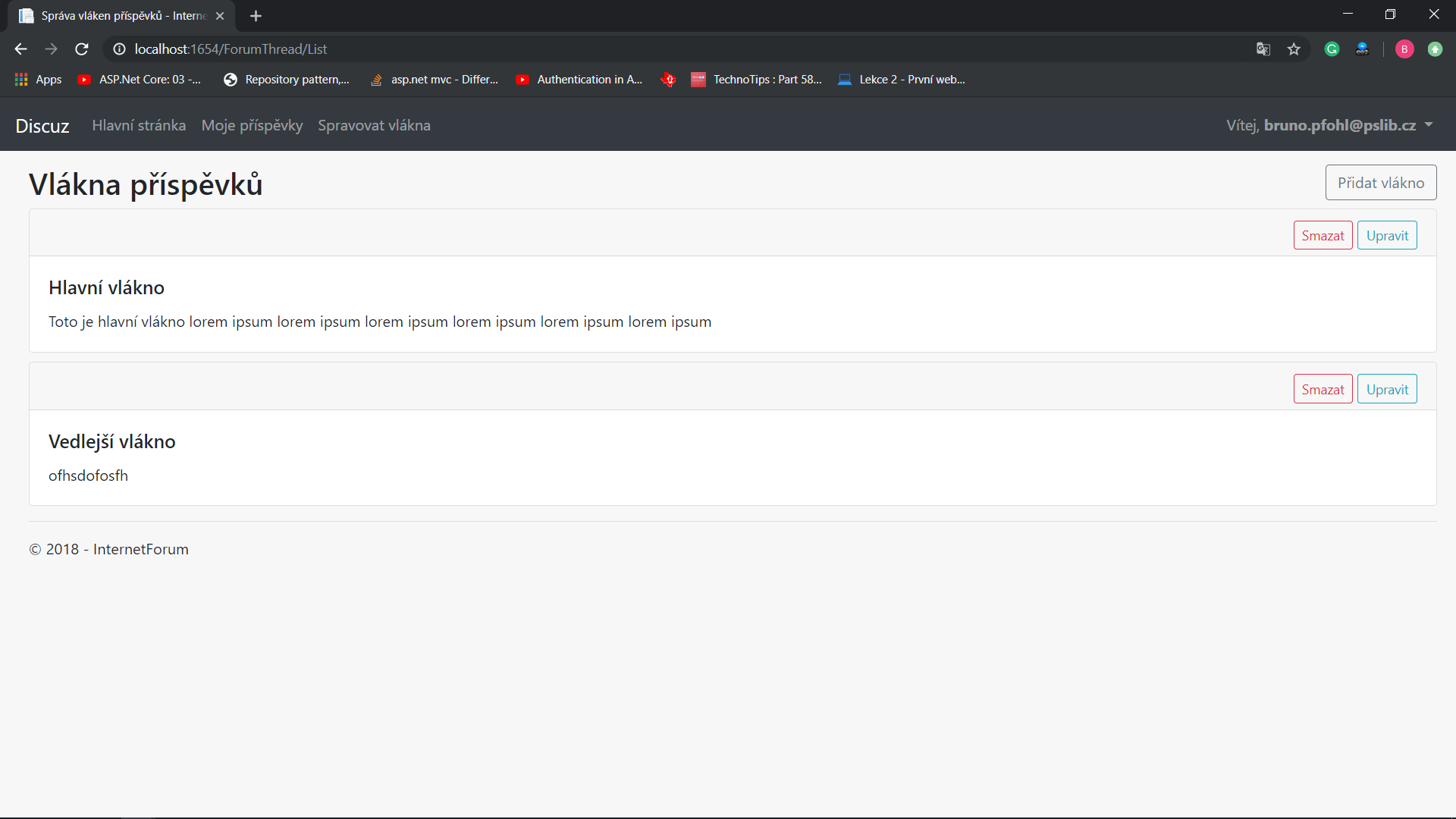
Obrázek 55 Správa příspěvků



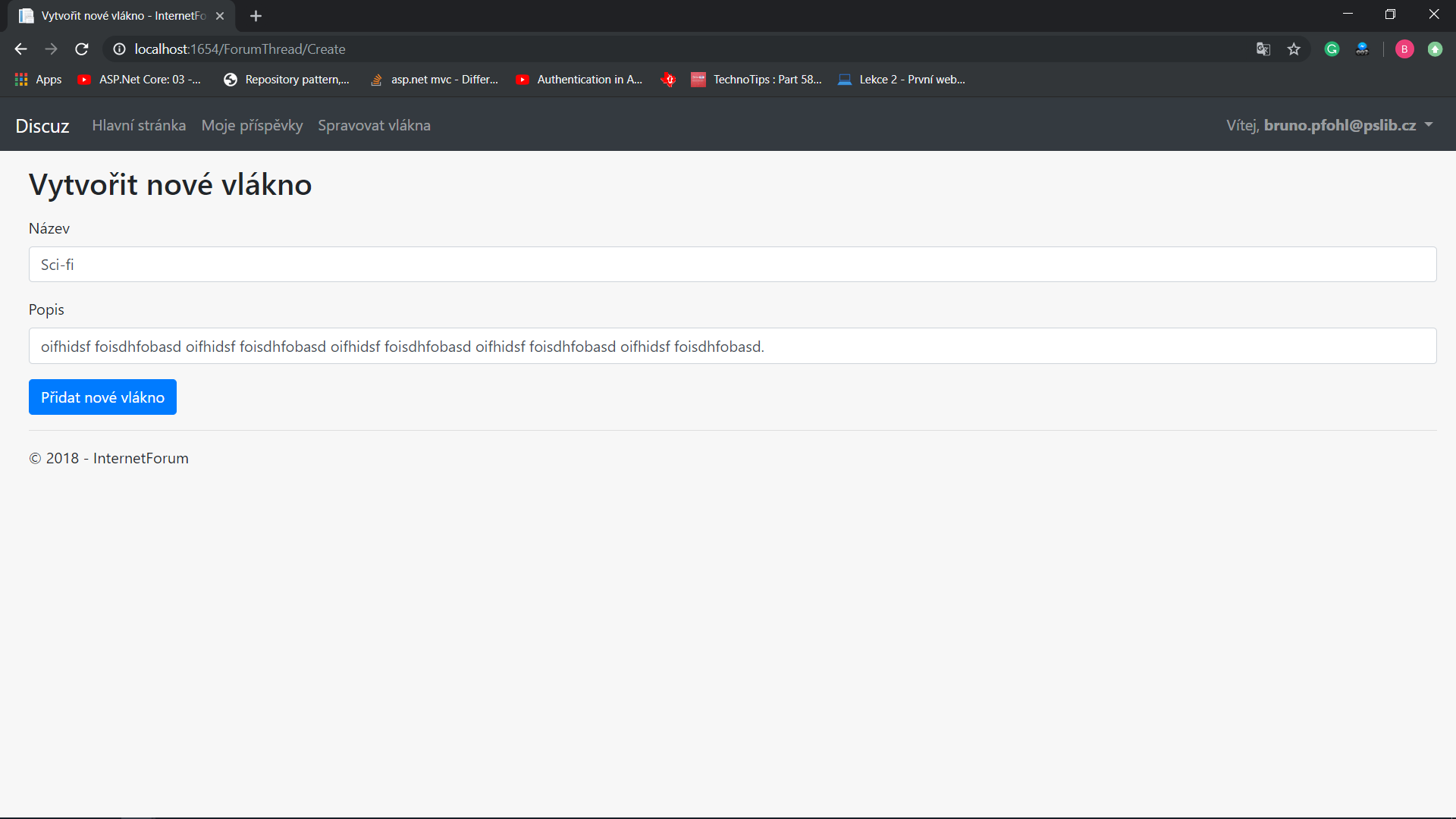
Obrázek 56 Přidání nového příspěvku



Obrázek 57 Zobrazení příspěvků z jiné kategorie



Obrázek 58 Správa kategorií příspěvku



Obrázek 59 Přidání nové kategorie

1. Ověření výstupu na studentech

Tato kapitola se zabývá integrací mnou vytvořených studijních materiálů do výukových hodin předmětu ASW.

* 1. Zvolená cílová skupina studentů

Na základě domluvy, kterou jsme uskutečnili s Ing. Tomášem Kazdou, bylo rozhodnuto, že studijní materiály zařadíme do výuky studentů maturitního ročníků v oboru informačních technologií (třída P4).

Ing. Tomáš Kazda tedy do výukových hodin předmětu ASW zakomponoval částečně mnou vytvořené studijní materiály. Studijní materiály studentům nesloužily jako plnohodnotný zdroj všech informací, ale spíše jen pro inspiraci a samostudium.

* 1. Odezva jednotlivých studentů

Každý ze studentů byl požádán, aby vypracoval recenzi studijních materiálů o minimálním počtu 150 slov. Všichni studenti recenzi vypracovali, ale jen některé recenze se dají považovat za přínosné. Všechny přínosné a smysluplné recenze jsou v této podkapitole obsaženy. Texty prošly lehkou úpravou, nutnou pro odstranění gramatických chyb. Původní myšlenka textů nebyla nijak změněna.

**Martin Košek**

Dokumenty jsou zpracované přehledným způsobem a dobře se v nich orientuje. Není problém najít to, co zrovna potřebuji a spojit to s jinými informacemi, neboť je stručně napsané, k čemu se daná věc používá a které jiné služby s ní spolupracují. Líbí se mi zpracování a popis praktické ukázky služeb, kdy je kód napsán čitelným a jednoduše pochopitelným způsobem. Naopak na druhou stranu bych na konec kapitoly dal například část trochu komplexnějšího kódu, kde by bylo vidět, jaký kód se využívá v praxi. Některé části jsou popsány trochu méně, než bych očekával, ale to asi bude způsobeno tím, že se o té dané věci nejspíše prostě nedá napsat o moc více informací. Ještě bych se podíval na styl psaní textu. U některých vět by se dal vymyslet trochu smysluplnější slovosled, aby byla věta lépe pochopitelná, ale to bych spíš počítal do finální úpravy a není to nijak extra zásadní. Myslím si, že pro začátečníka jsou dokumenty nápomocné a prospěšné.

**Jaroslav Urban**

Brunova maturitní práce je vskutku obsáhlá, ale i přesto jednoduše čitelná. Je to vskutku skvělý úvod do ASP.NET Core. Velmi mi pomohla s pochopením základních principů této technologie. Vytknul bych možná jen detaily jako je střídání českých a anglických výrazů jako například řadič, kontroler nebo testy jednotek a unit testy. Zároveň nevím, co je cílem maturitní práce, ale přišlo mi, že se v prvních několika kapitolách soustředila na popis tvorby internetové aplikace webového fóra, a v druhé části se zabývala spíše teoretickou stránkou věci, než pokračováním v popisu toho, jak podobnou aplikaci vytvořit.

Pokud bych začal číst tuto dokumentaci s tím, že bych se chtěl naučit, jak krok po kroku vytvořit podobnou aplikaci, skončil bych v polovině. Jinak je to výborná práce a hodně jsem se díky ní naučil.

**Miloš Pánek**

Učební materiály jsou velice dobře zpracované. Návody navazují na dosavadní znalosti studentů adekvátně a téma projednávají od dostatečných základů. Velice oceňuji jejich přehlednost a stručnost. Časté ukázky kódu nebo diagramy velice pomáhají s názorností a lépe čtenáři přibližují daný postup. Líbí se mi struktura materiálů do podoby tutoriálu, čtenář si tak dané téma může přímo na místě vyzkoušet. Vytknout mohu pouze nejednotnou strukturu dokumentů, nejvýraznější v kapitole o HTML Tag Helpers, jejíž struktura se od ostatních úplně liší.

Další výtkou jsou občasné chyby ve stylistice textu a typografii, ty ale nejsou nikdy tak špatné, aby narušily srozumitelnost textu. Dokumentace není sice příliš podrobná, což ovšem nebylo účelem práce a návod je tak přehlednější a srozumitelnější. Dokázal bych si představit použití takovýchto materiálů při výuce.

**Vojtěch Hrůza**

Pokud bych měl práci hodnotit od začátku, pak musím, ač nerad, začít jedním záporem. První kapitoly jsou na úvod velmi krátké. V zásadě to ničemu nevadí a je spíše dobře, že tu není snaha o nacpání co nejvíce zbytečností do čtenáře, ale ty kapitoly by se daly například sloučit, ať to je trochu lépe pohromadě. Není to sice nic vysloveně špatného, ale trochu mi to během čtení vadilo.

Co se úvodu týče, hodnotím ho kladně. Je v něm hezky řečeno, co je vlastně .NET Core zač, něco o něm atd. Kdo nechce, může ho asi přeskočit, ale i tak to jsou informace, které by mohli být pro člověka zajímavé.

Už od začátku je vše zpracováváno stylem "jak pro naprostého blba," což jménem všech blbů náramně oceňuji. Vysvětlení je pomalé, jednoduché a přehledné. Vše, již od naprosto základního zakládání projektu je navíc doprovázeno obrázky a barevným zvýrazněním. Jednotlivé kusy kódu pak přidávají na přehlednosti a na názorných ukázkách se člověk vždy nejvíce naučí. Velmi dobrým faktorem pro učení je také možnost zpracovávat si aplikaci za chodu, souběžně s čtením. Velmi se mi líbilo také lehčí prokládání teorií, kdy je přesně vysvětleno, co vlastně v tu chvíli čtenář dělá a čím se bude zabývat. Rozhodně to člověku více vysvětlí a může to i prohloubit jeho nadšení pro věc.

Jedinou další vadou, kterou bych tu asi viděl, je nedostatek materiálu. Respektive, není zde spousta z toho, co jsme se učili ve škole. Na druhou stranu však chápu, že se nedá obsáhnout všechno, aniž by pak výsledek vypadal jako kniha od S. Kinga.

Celkově tedy můžu říct, že je to pěkně zpracovaný, i blbům přístupný návod, který zaručeně může přijít nejednomu člověku vhod.

**Václav Šír**

Úvod je stručný a jasný. Krátký text o historii považuji za část osnovy výukových materiálů, proto je super, že je zde uveden. Lekce „Založení prvního projektu“ je příjemně rozdělena na sekce. Každý krok je následován obrázkem, což dodává na přehlednosti jednotlivých akcí. Vysvětlení základních metod je sice krátké, ale dobře pochopitelné. Příliš informací ze začátku je spíše ke škodě než k užitku. Stručná a jednoduchá názorná ukázka s popisem je dobrá.

Popis MVC frameworku je dobře pochopitelný a není zbytečně zdlouhavý. Implementace MVC je dobře popsaná. Ukázka implementace v praxi je lehce pochopitelná i pro začátečníky.

U „RazorViews“ by neuškodilo napsat i jiné používané „renderovací enginy“, aby se čtenář seznámil s alternativami. Obeznámení s konvencemi ve složkách hned ze začátku je dobrý přístup k věci. Text o HTML Tag Helpers má osnovu na začátku, čímž se liší od ostatních dokumentů, bylo by lepší ji buď smazat nebo osnovu doplnit k ostatním textům.

U Dekompozice softwarové logiky je z nějakého důvodu osnova na konci s odkazy. Odkazy na konci jsou dobré, ale osnova se dává na začátek.

**Jan Pecka**

Jedná se o krátké a jednoduché seznámení s frameworkem ASP.NET Core. Celý návod se drží jedné předlohy a tím umožňuje snadnější porozumění textu. Jednoduše jsem se zorientoval. Je vidět, že Pfohl tématu rozumí a po srovnání myšlenek dokáže předat cenné informace. Po přečtení všech kapitol jsem si dokázal více představit funkčnost MVC a celého frameworku.

Některé věty mi přišly celkem krkolomné a občasné stylistické chyby na kráse též nepřidávají. Pokud dojde k opravě těchto nedokonalostí, tak se bude jednat o materiál na vysoké úrovni, který poslouží výtečně k výuce frameworku ASP.NET Core. Dokázal bych si představit, že by mohl být nápomocen mnoha lidem.

Obrázky oživují celý text a ulehčují orientaci v daném tématu. Jsou vhodně zvolené, jen u některých není jednoznačně vidět o čem se hovoří (mohly by se použít podržení v obrázku atd.).

V kapitole o dekompozici softwarové logiky je zvláštně veliká mezera a chtěla by poupravit.

Celkově hodnotím Brunovu práci velmi kladně a jsem rád, že jsem mohl pomoci mým hodnocením.

**Pavel Rudolf**

Na autorově práci se mi líbí její stručnost. Dobrý je také způsob vyjadřování, díky němuž se čtenář se neztratí ve složitých zkratkách a přehnaně odborném tvarosloví, vše je hezky vysvětleno.

Dalším bonusem je také barevné zvýrazňování důležitých věcí, trochu mi ale vadí, že nesouhlasí s barvami v obrázcích, u kterých tyto věci popisuje. Líbí se mi i to, jak šel autor postupně, nejdříve popisuje samotný ASP.NET Core projekt a až potom k němu přidává MVC architekturu, takže čtenář není přetížen věcmi, kterým nerozumí a vše jde logicky pěkně za sebou.

Dalším plusem byly ukázkové projekty, na kterých si čtenář může ověřit své teoretické znalosti. Postupy jsou vyjádřeny do detailu, takže se čtenář neztrácí. Dobré je také to, že autor i vysvětluje, co k čemu slouží, např. u Dependency Injection si klade otázku na co nám to je a co to umožňuje.

Autora bych tímto chtěl upozornit, že v kapitole o MVC je malými písmeny zkratka „URL“, což je špatně. Čistě z češtinářského pohledu bych autorovi doporučil dohledat si význam a využití pevných mezer, aby se komise nezastavila nad těmito věcmi a řešila více obsah práce.

**Vojtěch Boura**

Práce jako taková je dobře napsaná, tudíž se v ní snadno orientuje i člověk, který danému tématu částečně nerozumí. Zakomponování obrázku se mi zdá jako dobrý nápad, protože to zjednodušuje vysvětlení. Zároveň člověk vidí přímo obrázek z prostředí, ve kterém pracuje a umožnuje mu to lehčí orientaci.

Položka „RazorViews“ byla zpočátku těžce pochopitelná, ale po chvíli strávené u dokumentu a opakovanému přečtení se to pochopit dalo. Zbytek práce už byl podobný, jen samozřejmě s jiným obsahem. Text na sebe dobře navazoval, takže nebyla nutnost dohledávat si informace z jiných zdrojů, což se u hodně těchto materiálu stává. K programu asi také nemohu říct nic jiného, než že je napsaný dobře a v celku přehledně.

Jediné, co bych vytknul je nahrávání dokumentů na GitHub. Uživatel je nucen dané soubory stahovat. V dnešní době je mnoho úložišť, které povolují živé zobrazení.

**Petr Svoboda**

Práce je velmi přehledná, dobře zpracovaná a dobře rozdělená. Díky této práci jsem se naučil a doučil základy a různorodost APS.NET Core. V práci je vše vysvětleno podrobně, a tak aby to pochopil i mírně pokročilý začátečník. Jediné, co bych vytknul je střídání českých a anglických názvů, občas jsem z toho byl mírně zmaten. I přesto, že je tato práce velmi rozsáhlá, čte se snadno a rychle. Vše je napsáno srozumitelně. Velmi užitečné jsou ukázky kódů a detailně vysvětlené kroky pro vytvoření jednotlivých částí.

Jediné, co mě trochu překvapilo, že ve škole používáme React-Redux, ale Pfohl se rozhodl popsat technologii Razor Views. Nepovažuji to za chybu, ale chtěl bych upozornit autora na to, že by bylo lepší se s vyučujícím lépe domluvit, aby studijní materiály lépe navazovaly na výuku.

Možná by bylo lepší, kdyby autor popsal všechny šablony (Empty, Angular, MVC, React-Redux) stejně tak, jako popsal všechny přístupy při vytváření databáze pomocí technologie EntityFramework Core. Tento přístup se mi líbil mnohem více.

* 1. Shrnutí a ponaučení

Odezva na studijní materiály je vesměs kladná. Podle studentů se jedná o použitelné materiály, které se dle nich dají využít při výuce či samostudiu. Líbí se jim především grafická znázornění v podobě obrázků, jednoduchost a návaznost materiálů.

Téměř všichni studenti upozornili na gramatické a stylistické chyby. Na základě toho jsem se stylistiku a gramatické chyby rozhodnul opravit, což se mi dle mého názoru do větší míry povedlo.

Mezi studenty se objevily i konfliktní názory. Dle většiny studentů je jednoduchost a stručnost materiálů jejich hlavní výhodou. Podle některých studentů by však materiály měli být komplexnější. Tyto rozdílné názory studentů dle mého souvisí s množstvím jejich předchozích zkušeností.

Velmi přínosnou mi byla zpětná odezva od studenta Vojtěcha Boury, který upozornil na to, že bych měl materiály umístit na úložiště umožňující živý náhled dokumentu. Zvolit si GitHub jako úložiště pro studijní materiály nebylo dobré rozhodnutí, protože studenti nevěděli, jestli mají aktuální verzi dokumentů nebo ne.

Dále ještě děkuji studentu Marku Honcovi, který vypracoval velmi obsáhlou recenzi. Upozornil mě na spoustu chyb v mých studijních materiálech, které jsem tak měl možnost opravit. Jeho recenze však byla natolik obsáhlá, že mi přišlo nevhodné ji sem přidat.

Závěr

Dle zadání mé maturitní práce jsem měl za úkol vytvořit studijní materiály, ukázkovou webovou aplikaci a zpracovat reakci studentů na zmíněné studijní materiály do finální dokumentace.

V průběhu vypracovávání mé maturitní práce jsem narazil na několik problémů, které jsem zkonzultoval s vedoucím mé maturitní práce Ing. Tomášem Kazdou a dle naší domluvy je pak adekvátním způsobem vyřešil.

Jako největší problém se ukázala moje nerozvážnost. Zpočátku jsem si řádně nepromyslel, čím přesně se bude moje maturitní práce zabývat. Začal jsem vývojem ukázkové webové aplikace, kterou jsem vybavil mnoha přebytečnými funkcemi. Po konzultaci s Ing. Tomášem Kazdou jsme pak došli k závěru, že tyto dispozice ukázkové aplikace by byly pro studenta při jeho snaze o pochopení studijních materiálů pouze přítěží. Webovou aplikaci jsem se pak rozhodl značně zjednodušit, což mi umožnilo studijní materiály napsat stručněji a přehledněji.

Dalším problémem byl nedostatek komunikace mezi mnou a studenty třídy P4. Přecenil jsem jejich zájem o mé studijní materiály, což vedlo k tomu, že spousta studentů se zpočátku ke studiu materiálů vůbec nedostala. Nakonec se však informaci o tom, že by se studenti třídy P4 měli zabývat mými studijními materiály, podařilo rozšířit. Názory všech studentů na mé studijní materiály se mi podařilo úspěšně shromáždit a zpracovat do této maturitní práce.

Všichni studenti mi buď písemně nebo ústně sdělili, že jim studijní materiály byly nápomocné. Většině z nich se líbila převážně stručnost a jednoduchost materiálů. Samozřejmě se mi z jejich strany dostalo i negativní kritiky, ze které jsem se snažil poučit a studijní materiály jsem podle ní upravil.

Celkově tedy hodnotím výsledek mé maturitní práce kladně. Vypracovávání maturitní práce pro mě byla velmi zajímavá a cenná zkušenost. Díky této maturitní práci se mi podařilo vzdělat nejen studenty třídy P4, ale taky i mě samotného.

Seznam obrázků

[Obrázek 1 Úprava stávající instalace programu Visual Studio 2](#_Toc3285095)

[Obrázek 2 Založení prvního projektu 4](#_Toc3285096)

[Obrázek 3 Názorná ukázka request PIPELINE 7](#_Toc3285097)

[Obrázek 4 Princip MVC architektury 8](#_Toc3285098)

[Obrázek 5 Založení projektu implementující MVC 9](#_Toc3285099)

[Obrázek 6 Struktura projektu implementující MVC architekturu 10](#_Toc3285100)

[Obrázek 7 Kód třídy CarViewModel 11](#_Toc3285101)

[Obrázek 8 Přidání řadiče 12](#_Toc3285102)

[Obrázek 9 Pojmenování řadiče 12](#_Toc3285103)

[Obrázek 10 Kód řadiče 13](#_Toc3285104)

[Obrázek 11 Vytvoření šablony 13](#_Toc3285105)

[Obrázek 12 Kód šablony 14](#_Toc3285106)

[Obrázek 13 Výstup webové aplikace implementující MVC 14](#_Toc3285107)

[Obrázek 14 Kód v souboru \_Layout.cshtml 15](#_Toc3285108)

[Obrázek 15 Struktura složky Views 16](#_Toc3285109)

[Obrázek 16 Kód v souboru \_ViewImports.cshtml 16](#_Toc3285110)

[Obrázek 17 Využití souboru \_ViewStart.cshtml 17](#_Toc3285111)

[Obrázek 18 Využití Razor syntaxe v praxi 18](#_Toc3285112)

[Obrázek 19 Příklad použití příkazu @addTagHelper 19](#_Toc3285113)

[Obrázek 20 Model formuláře 21](#_Toc3285114)

[Obrázek 21 Kód akce pro vrácení formuláře 22](#_Toc3285115)

[Obrázek 22 Kód akce pro zpracování dat 22](#_Toc3285116)

[Obrázek 23 Kód šablony formuláře 23](#_Toc3285117)

[Obrázek 24 Dekompozice aplikace na 3 vrstvy 25](#_Toc3285118)

[Obrázek 25 Vytvoření rozhraní služby 29](#_Toc3285119)

[Obrázek 26 Vytvoření implementace rozhraní 29](#_Toc3285120)

[Obrázek 27 Registrace služby do Service Provideru 30](#_Toc3285121)

[Obrázek 28 Vyžádání služby v konstruktoru třídy 30](#_Toc3285122)

[Obrázek 29 Otestování první služby 31](#_Toc3285123)

[Obrázek 30 Záměna služby 31](#_Toc3285124)

[Obrázek 31 Otestování záměny služby 31](#_Toc3285125)

[Obrázek 32 Založení projektu datové vrstvy 32](#_Toc3285126)

[Obrázek 33 Adresářová struktura datové vrstvy 33](#_Toc3285127)

[Obrázek 34 Entita uživatele 33](#_Toc3285128)

[Obrázek 35 Entita auta 33](#_Toc3285129)

[Obrázek 36 Entita společnosti vyrábějící auta 34](#_Toc3285130)

[Obrázek 37 Třída ApplicationDbContext 34](#_Toc3285131)

[Obrázek 38 Přidání reference na datovou vrstvu 36](#_Toc3285132)

[Obrázek 39 Registrace a konfigurace DbContext ve Startup třídě 36](#_Toc3285133)

[Obrázek 40 Třída DesignTimeDbContextFactory 37](#_Toc3285134)

[Obrázek 41 Přidání nové migrace 38](#_Toc3285135)

[Obrázek 42 Složka Migrations 38](#_Toc3285136)

[Obrázek 43 Provedení migrace 38](#_Toc3285137)

[Obrázek 44 Adresářová struktura u Repository Pattern 39](#_Toc3285138)

[Obrázek 45 Rozhraní bázového repozitáře 40](#_Toc3285139)

[Obrázek 46 Implementace rozhraní bázového repozitáře 40](#_Toc3285140)

[Obrázek 47 Rozhraní repozitáře aut 41](#_Toc3285141)

[Obrázek 48 Implementace rozhraní repozitáře aut 41](#_Toc3285142)

[Obrázek 49 Vytvoření rozhraní pro Unit of Work 42](#_Toc3285143)

[Obrázek 50 Implementace rozhraní Unit Of Work 43](#_Toc3285144)

[Obrázek 51 Využití Unit Of Work v prezenční vrstvě 43](#_Toc3285145)

[Obrázek 52 Struktura databáze 45](#_Toc3285146)

[Obrázek 53 Pohled pro nepřihlášeného uživatele 45](#_Toc3285147)

[Obrázek 54 Pohled přihlášeného uživatele 46](#_Toc3285148)

[Obrázek 55 Správa příspěvků 46](#_Toc3285149)

[Obrázek 56 Přidání nového příspěvku 47](#_Toc3285150)

[Obrázek 57 Zobrazení příspěvků z jiné kategorie 47](#_Toc3285151)

[Obrázek 58 Správa kategorií příspěvku 48](#_Toc3285152)

[Obrázek 59 Přidání nové kategorie 48](#_Toc3285153)

Použitá literatura

1. **ČÁPKA, David.** Lekce 1 - Úvod do MVC architektury v ASP.NET. itnetwork.cz [online]. Praha: David Čápka, 2015 [cit. 2019-03-10]. Dostupné z: https://www.itnetwork.cz/csharp/asp-net/mvc/zaklady/asp-dot-net-uvod-do-mvc-architektury

2**. ČÁPKA, David.** Lekce 2 - První webová aplikace v ASP.NET MVC. *Itnetwork.cz* [online]. Praha: David Čápka, 2016 [cit. 2019-03-10]. Dostupné z: https://www.itnetwork.cz/csharp/asp-net/mvc/zaklady/asp-dot-net-mvc-tutorial-prvni-webova-aplikace

3. **41. (ASP.NET Core 1.0 & MVC) Tag Helpers. In**: Youtube [online]. 5.12.2016 [cit.

2019-03-10]. Dostupné z: https://youtu.be/S3\_XKjtZh7E. Kanál uživatele Programming.

4. **32. (ASP.NET Core 1.0) MVC ViewStart Files. In**: Youtube [online]. 11.10.2016 [cit. 2019-03-10]. Dostupné z: https://youtu.be/issFzHzk0io. Kanál uživatele Programming.

5. **45. (ASP.NET Core 1.0 & MVC) Introduction To Dependency Injection. In**: Youtube [online]. 15.1.2017 [cit. 2019-03-10]. Dostupné z: https://youtu.be/oKyzza01rzA. Kanál uživatele Programming.

6. **33. (ASP.NET Core 1.0 & MVC) ViewImports Files. In**: Youtube [online]. 17.10.2016 [cit. 2019-03-10]. Dostupné z: https://youtu.be/cmNUCbSan0o. Kanál uživatele Programming.

7. BAGERI, Saineshwar. Create and Implement 3-Tier Architecture in ASP.Net. *C# Corner* [online]. Wyoming (USA): Saineshwar Bageri, 2018 [cit. 2019-03-10]. Dostupné z: <https://www.c-sharpcorner.com/UploadFile/4d9083/create-and-implement-3-tier-architecture-in-Asp-Net/>

8. **ROTH, Daniel, Rick ANDERSON a Shaun LUTTIN**. Introduction to ASP.NET Core. *Microsoft* [online]. Redmond (USA): Microsoft, 2019 [cit. 2019-03-10]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2

9. **DYKSTRA, Tom**. Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application (9 of 10). *Microsoft* [online]. Redmond (USA): Microsoft, 2013 [cit. 2019-03-10]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

10. **Ryan**. Code-First vs Model-First vs Database-First: Pros and Cons. *Ryadel* [online]. Toronto (Kanada): Ryadel, 2018 [cit. 2019-03-10]. Dostupné z: https://www.ryadel.com/en/code-first-model-first-database-first-vs-comparison-orm-asp-net-core-entity-framework-ef-data/

1. **učitelé SPŠSE.** Úvod. *SPŠSE a VOŠ Liberec.* [Online] 01. 09 2016. [Citace: 01. 09 2016.] https://www.pslib.cz.

1. Seznam přiložených souborů

Text

1. Další příloha