

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 uvedena na konci práce.

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 will also be included in the end of this document.

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](#_Toc2691057)

[1 Tvorba studijních materiálů 2](#_Toc2691058)

[1.1 Software potřebný k vývoji 2](#_Toc2691059)

[1.2 Technické specifikace .NET Core a ASP .NET Core 2](#_Toc2691060)

[1.2.1 Framework .NET Core obecně 3](#_Toc2691061)

[1.2.2 Porovnání frameworku .NET Core s frameworkem .NET 3](#_Toc2691062)

[1.2.3 Framework ASP .NET Core 3](#_Toc2691063)

[1.3 Vytvoření prvního projektu 4](#_Toc2691064)

[1.3.1 Založení projektu 4](#_Toc2691065)

[1.3.2 Struktura projektu 5](#_Toc2691066)

[1.3.3 Názorná ukázka request PIPELINE 7](#_Toc2691067)

[1.4 Principy MVC architektury 8](#_Toc2691068)

[V této kapitole si detailně popíšeme MVC architekturu a ukážeme si, jak tuto architekturu implementovat do ASP .NET Core projektu. 8](#_Toc2691069)

[1.4.1 Co je to MVC architektura? 8](#_Toc2691070)

[1.4.2 Princip MVC achitektury 8](#_Toc2691071)

[1.5 Vytvoření projektu implementující architekturu MVC 9](#_Toc2691072)

[1.5.1 Založení projektu 9](#_Toc2691073)

[1.5.2 Struktura projektu 10](#_Toc2691074)

[1.5.3 Praktická ukázka MVC 11](#_Toc2691075)

[1.6 Razor View Engine 16](#_Toc2691076)

[1.6.1 Využití Razor View Engine v MVC projektu 16](#_Toc2691077)

[1.6.2 Razor syntaxe 18](#_Toc2691078)

[1.7 HTML Tag Helpers 19](#_Toc2691079)

[1.7.1 Výhody HTML Tag Helpers 20](#_Toc2691080)

[1.7.2 Přidání nových HTML Tag Helpers. 20](#_Toc2691081)

[1.7.3 Výchozí HTML Tag Helpers 20](#_Toc2691082)

[1.8 Formuláře 21](#_Toc2691083)

[1.8.1 Vytvoření modelu 21](#_Toc2691084)

[1.8.2 Vytvoření potřebných akcí v řadiči 22](#_Toc2691085)

[1.8.3 Vytvoření formuláře pomocí HTML Tag Helpers 23](#_Toc2691086)

[1.9 Dekompozice softwarové logiky aplikace 25](#_Toc2691087)

[1.9.1 Hlavní důvody pro dekompozici aplikace 25](#_Toc2691088)

[1.9.2 Dekompozice aplikace na 3 vrstvy 26](#_Toc2691089)

[1.10 Entity Framework Core 27](#_Toc2691090)

[1.10.1 Model first 27](#_Toc2691091)

[1.10.2 Database first 28](#_Toc2691092)

[1.10.3 Code first 28](#_Toc2691093)

[1.11 Tvorba databáze pomocí Code First přístupu 29](#_Toc2691094)

[1.11.1 Další podpodkapitola 29](#_Toc2691095)

[1.11.2 Ještě další podpodkapitola 29](#_Toc2691096)

[2 Tvorba ukázkové aplikace 30](#_Toc2691097)

[2.1 Ukázka funkčnosti 30](#_Toc2691098)

[2.2 Struktura projektu 30](#_Toc2691099)

[3 Ověření výstupu na studentech 31](#_Toc2691100)

[3.1 Jak byla odezva získána? 31](#_Toc2691101)

[3.2 Odezva jednotlivých studentů 31](#_Toc2691102)

[3.3 Shrnutí a ponaučení 31](#_Toc2691103)

[Závěr 32](#_Toc2691104)

[Seznam obrázků 33](#_Toc2691105)

[Použitá literatura 34](#_Toc2691106)

[A. Seznam přiložených souborů 1](#_Toc2691107)

[B. Další příloha 2](#_Toc2691108)

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 části 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 během 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. Text těchto podkapitol cílí na 2. osobu množného čísla, tedy na studenty.

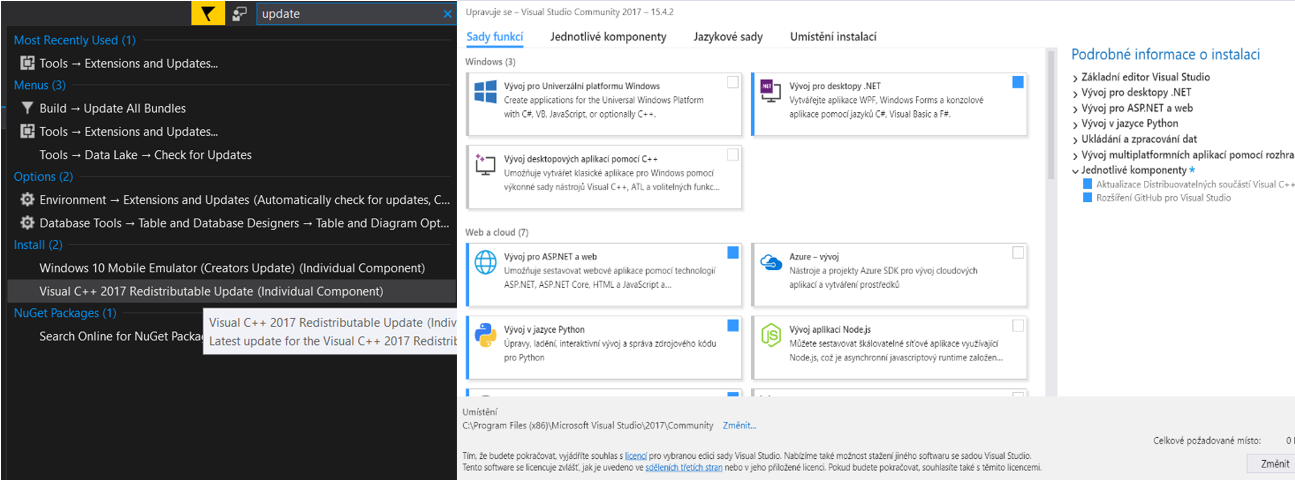
* 1. Software potřebný k vývoji

Pro bezproblémové následování tohoto návodu 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 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 a konzolové aplikace. To však neznamená, že se framework nedá využít pro vývoj jiného typu aplikací. Stačí pouze vytvořit nadstavbu na 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 kdokoliv. Microsoft si pro verzování frameworku zvolil webovou stránku github.com, kde má repositář jak pro framework .NET Core tak i pro framework 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.

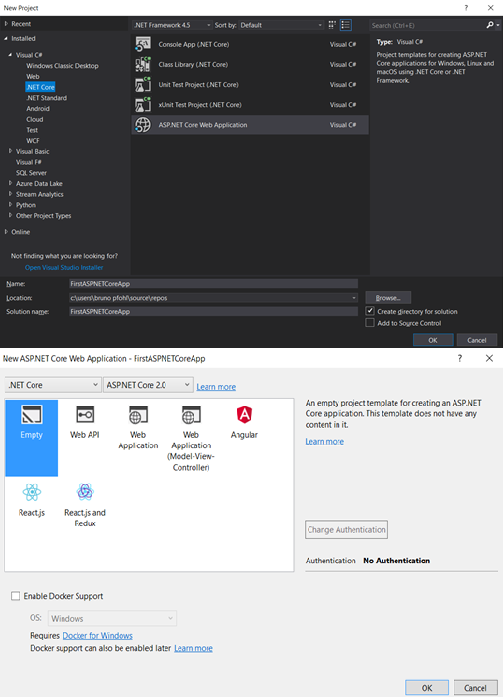
* 1. Vytvoření prvního projektu

V této kapitole je podrobný návod, jak vytvořit webovou aplikaci pomocí vývojového prostředí Visual Studio. Konkrétně se bude jednat o velmi jednoduchou aplikaci, která zatím ještě nebude implementovat MVC návrhový vzor. 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 používá metodu BuildWebHost, které předává parametry args. Tyto parametry jí byly předány při spuštění aplikace. Pro spuštění serveru volá metodu 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 specifikuje, 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 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), 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 v 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.

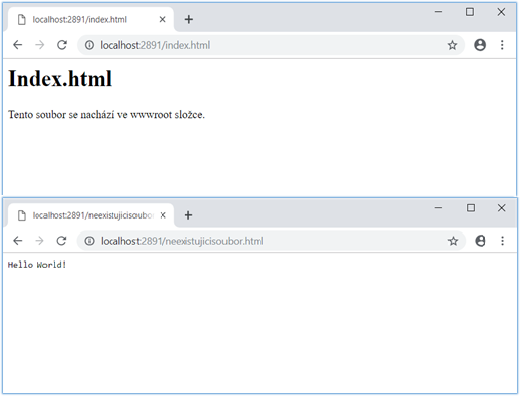
* + 1. Názorná ukázka request PIPELINE

Kód



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

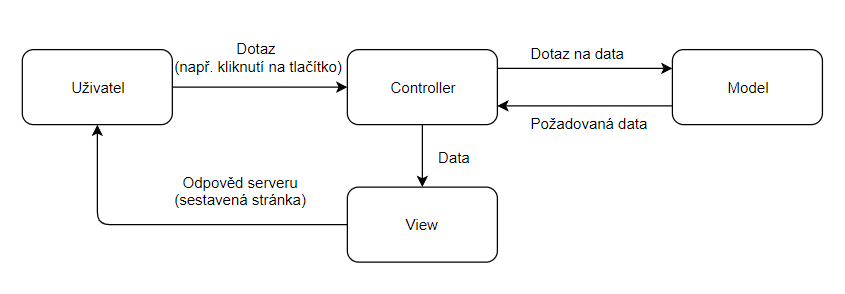
* + 1. Princip MVC achitektury

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 router. Router 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 router. Routerem 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 5 Princip MVC architektury

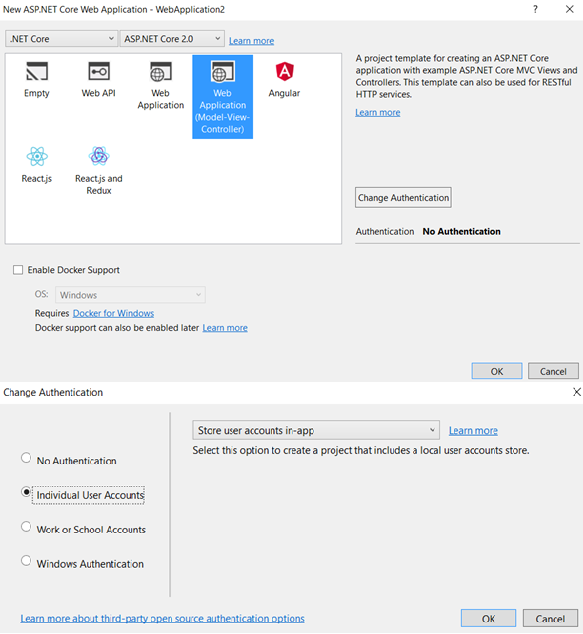
Na obrázku je vidět rozdělení prezenční vrstvy na logickou, datovou a vizuální. Toto rozdělení vede k více přehlednému, a hlavně snadněji udržitelnému kódu. Při implementaci MVC architektury by se ale však mělo myslet na to, že prezenční vrstva by měla obsahovat co nejméně logiky. V praxi to znamená, 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 kapitola se bude zabývat 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

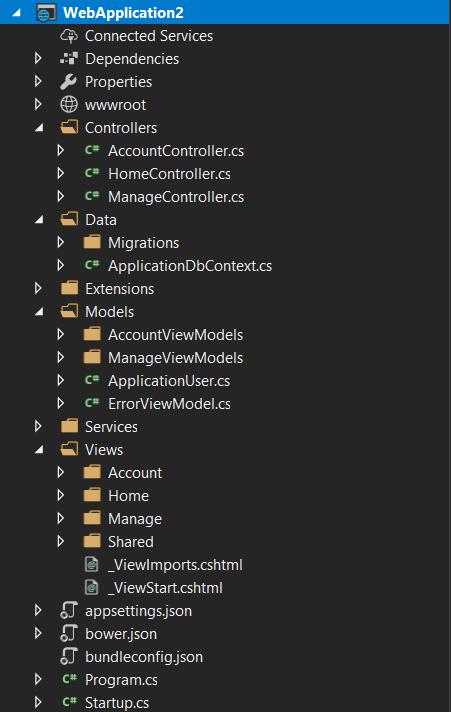
Založíme si tedy nový projekt pomocí stejného postupu jako u předchozího projektu. Tentokrát však využijeme šablonu Web Application (Model-View-Controller) a do projektu přidáme autentifikaci uživatelů.



Obrázek 6 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 7 Struktura projektu implementující MVC architekturu

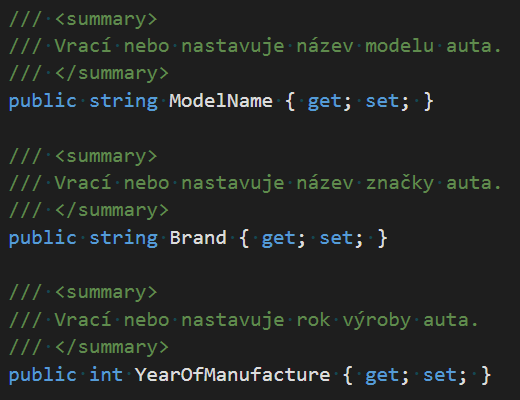
* + 1. Praktická ukázka MVC

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, který bude uchovávat informace o autě.

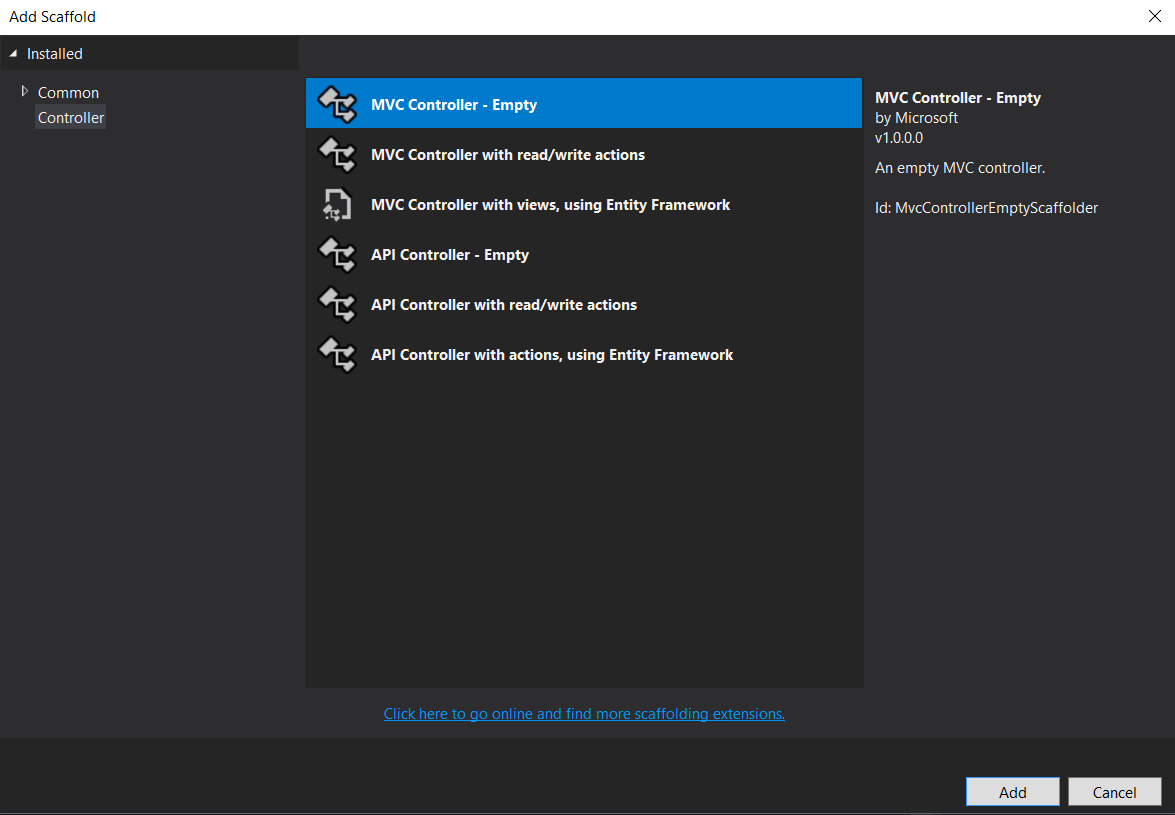


Obrázek 8 Kód třídy CarViewModel

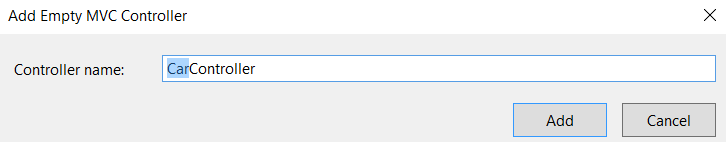
#### Vytvoření řadiče

Postup

1. Kliknout pravým tlačítkem myši na složku Controllers
2. Add
3. Controller
4. Zvolíme šablonu MVC Controller – Empty
5. Pojmenujeme řadič a potvrdíme



Obrázek 8 Přidání řadiče

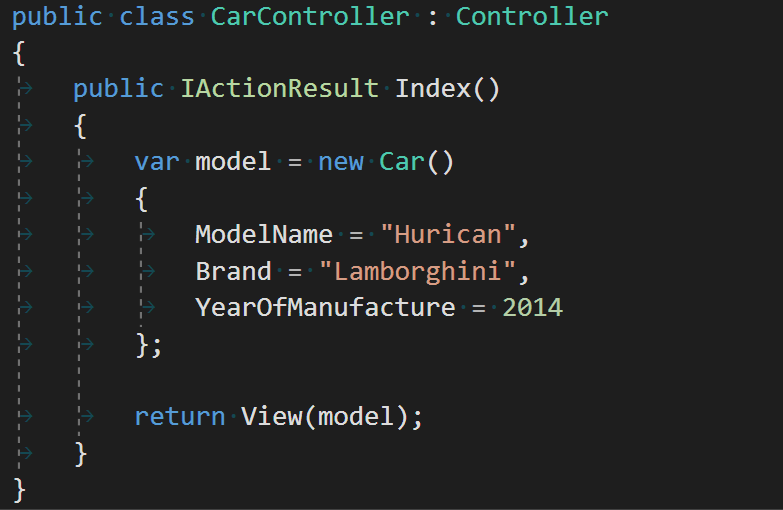


Obrázek 9 Pojmenování řadiče

#### Struktura řadiče

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

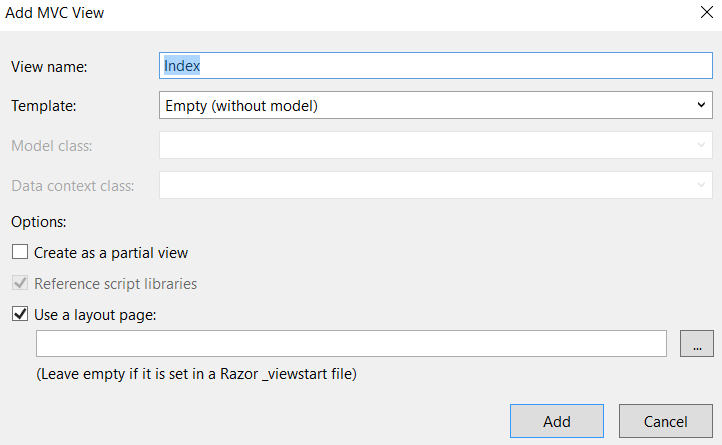
V našem případě tedy máme jednu akci jménem Index nacházející se v řadiči Car. V této akci vytváříme model, který pak předáváme metodě View. Nemáme však ještě uživatelské rozhraní (view) patřící k této akci.



Obrázek 10 Kód řadiče

#### Vytvoření šablony

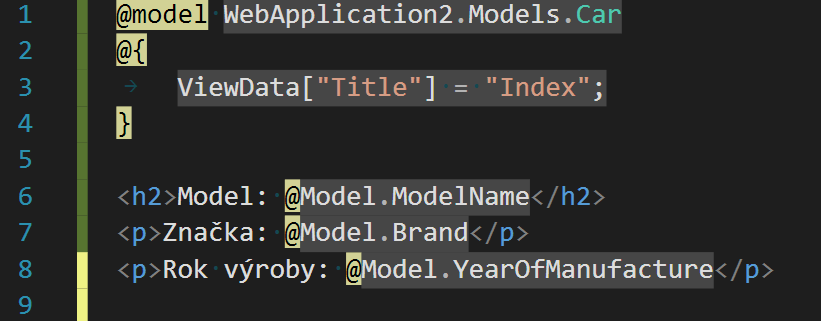
Klikneme pravým tlačítkem myši uvnitř metody Index a zvolíme „Add View“. Otevře se nám dialogové okno, které potvrdíme.



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.

Pozn.: Akce řadiče musí uživatelskému rozhraní předat model stejného typu, jaký vyžaduje šablona.

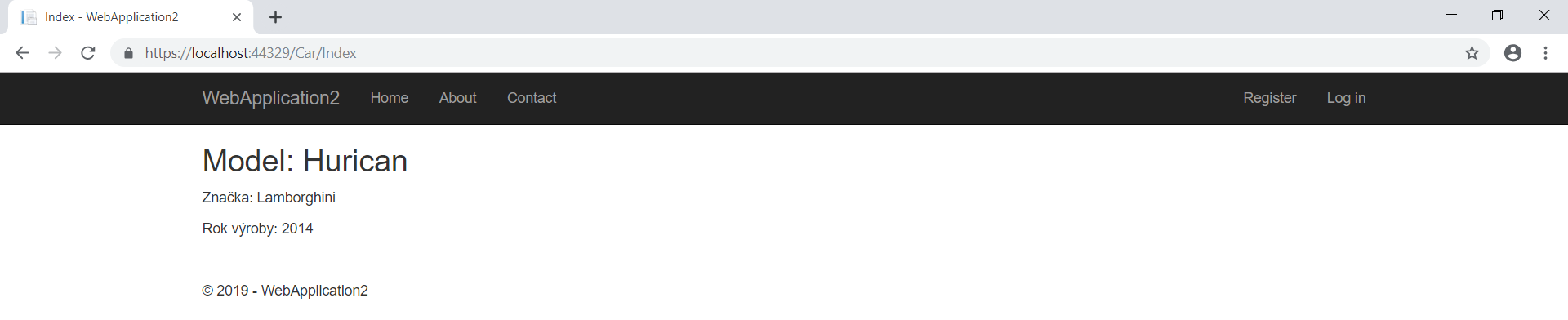


Obrázek 12 Kód šablony

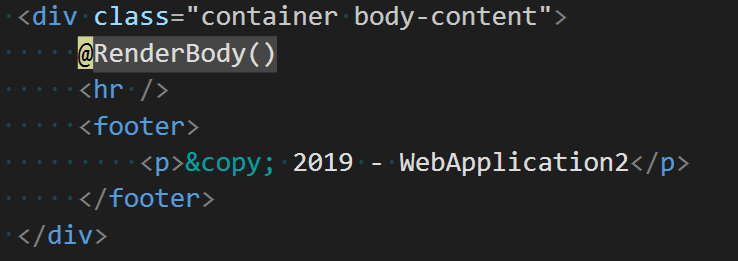
#### Ověření funkčnosti webové aplikace

Na obrázku 13 je zachycen výstup naší 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 ve souboru Views/Shared/\_Layout.cshtml.



Obrázek 13 Výstup webové aplikace implementující MVC



Obrázek 13 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 vyrenderované šablony, které tento layout využívají.

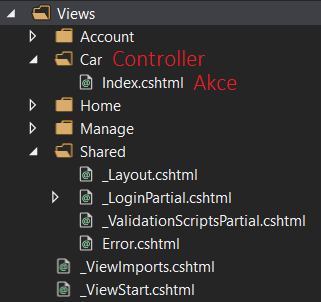
* 1. Razor View Engine

Razor View Engine je renderovací engine vytvořený společností Microsoft. Jedná se o ASP .NET komunitou nejvíce používaný renderovací engine. Umožnuje vytváření dynamických stránek pomocí kombinace HTML značkovacího jazyka a Razor syntaxe, která nám 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.



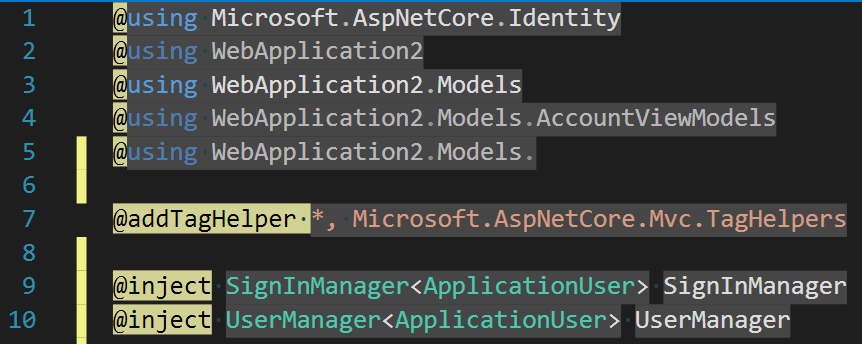
Obrázek 14 Struktura složky Views

#### 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 jednou 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.

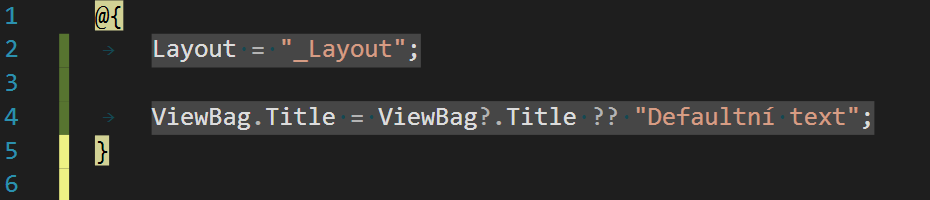


Obrázek 15 Kód v souboru \_ViewImports.cshtml

#### 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 16 můžete vidět kód, který slouží k nastavení výchozího nadpisu stránky.



Obrázek 16 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 renderuje jakožto nečíslovaný seznam. Přes kolekci aut iterujeme pomocí foreach cyklu a každé auto pak renderujeme 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 tvořili modely, které by byly následně šabloně předány.



Obrázek 17 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 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 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 18 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

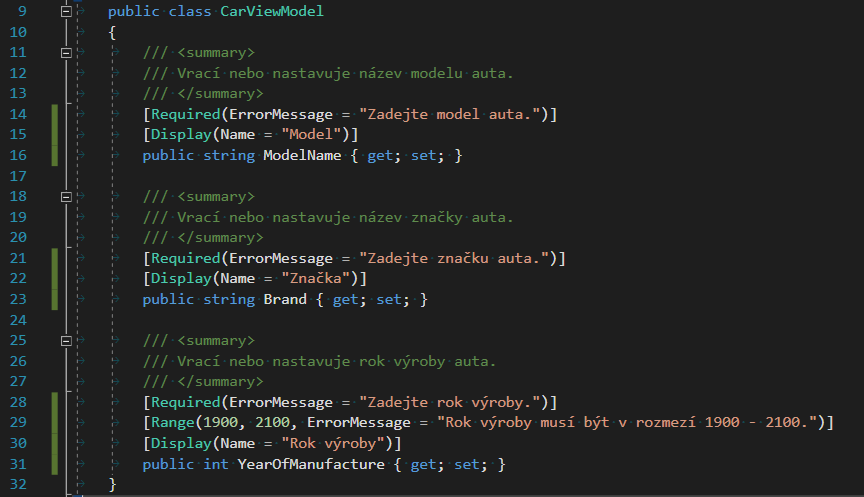
Tato kapitola se zabývá principy elektronických formulářů a jejich tvorbou pomocí technologie HTML Tag Helpers.

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 jen přijatými daty. Model se také využívá při vytváření formuláře pro dosazení výchozích hodnot.



Obrázek 19 Model formuláře

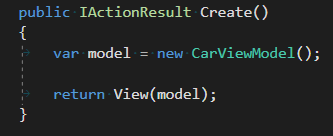
Náš model obsahuje 3 vlastnosti, kterými jsou značka auta, model auta a rok výroby. Jak můžete vidět na obrázku, nad každou vlastností jsou umístěny atributy, které specifikují text popisku vstupního pole ve formuláři (atribut Display) a podmínky, které musí hodnoty splňovat. Např. vlastnost YearOfManufacture 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.

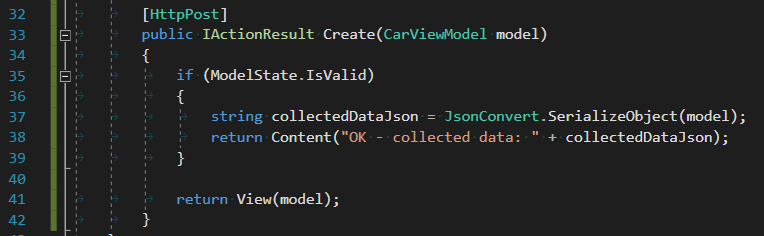
#### Vytvoření akce pro vrácení formuláře

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 20 Kód akce pro vrácení formuláře

#### Vytvoření akce pro zpracování dat



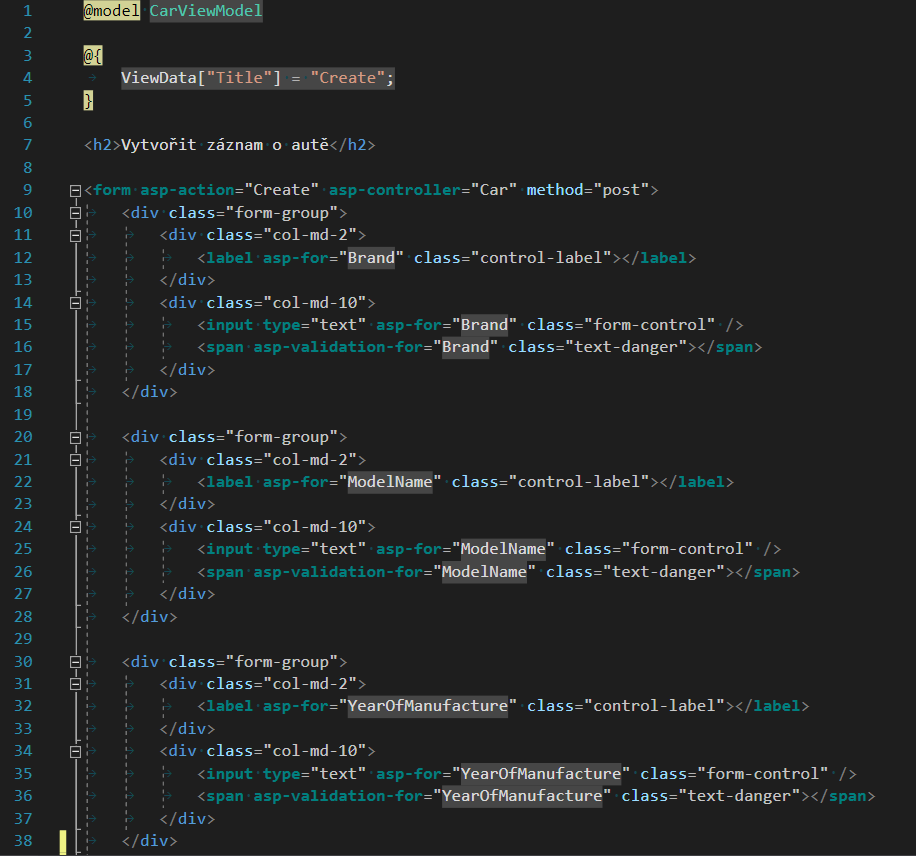
Obrázek 21 Kód akce pro zpracování dat

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 22 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 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.

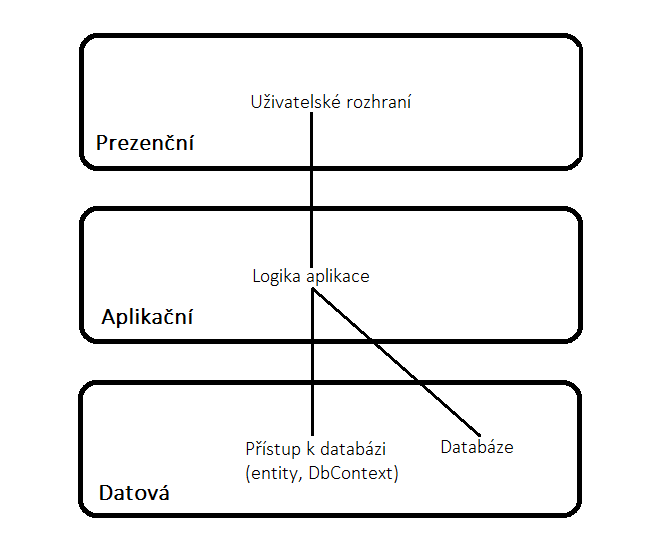
Špatná struktura naší aplikace může pro pozdější fázi vývoje znamenat její drastické zpomalení. Serióznost tohoto problému pak přímo úměrně stoupá s velikostí aplikace. Například, velké firmy mohou z důvodu špatné dekompozice softwarové logiky jejich projektu ztratit mnoho času a peněz, což často může 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 23 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 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 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 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 o jazyku 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

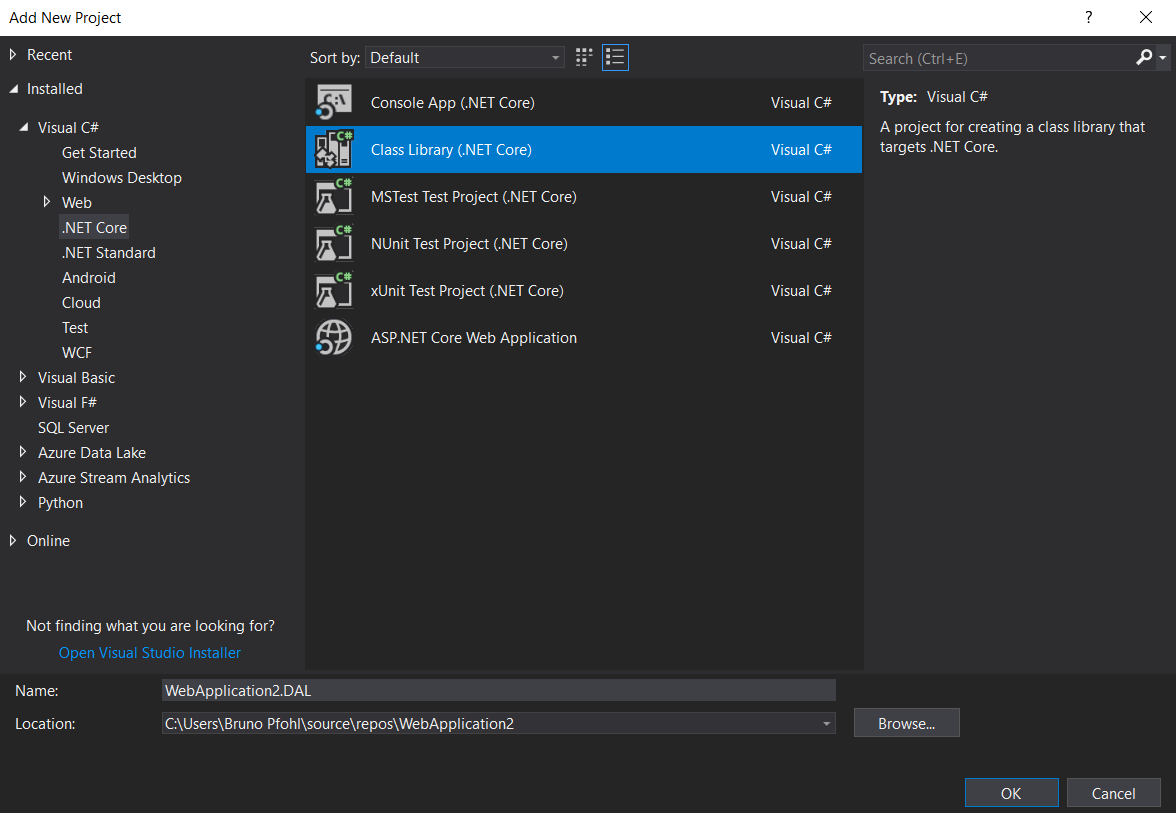
Nevýhody

1. nutnost pokročilých znalostí o technologii Entity Framework Core a Fluent Api
   1. 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 24 Založení projektu datové vrstvy

* + 1. Instalace potřebných balíčků

Dalším krokem je nainstalovat potřebné balíčky pomocí Nuget Package manageru.

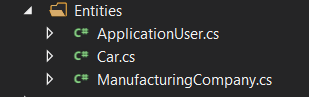
Postup při instalaci

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

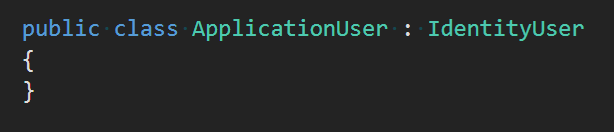
* Microsoft.AspNetCore.Identity.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.Design
* Microsoft.EntityFrameworkCore.Proxies
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.NETCore.App
  + 1. Vytvoření entit

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



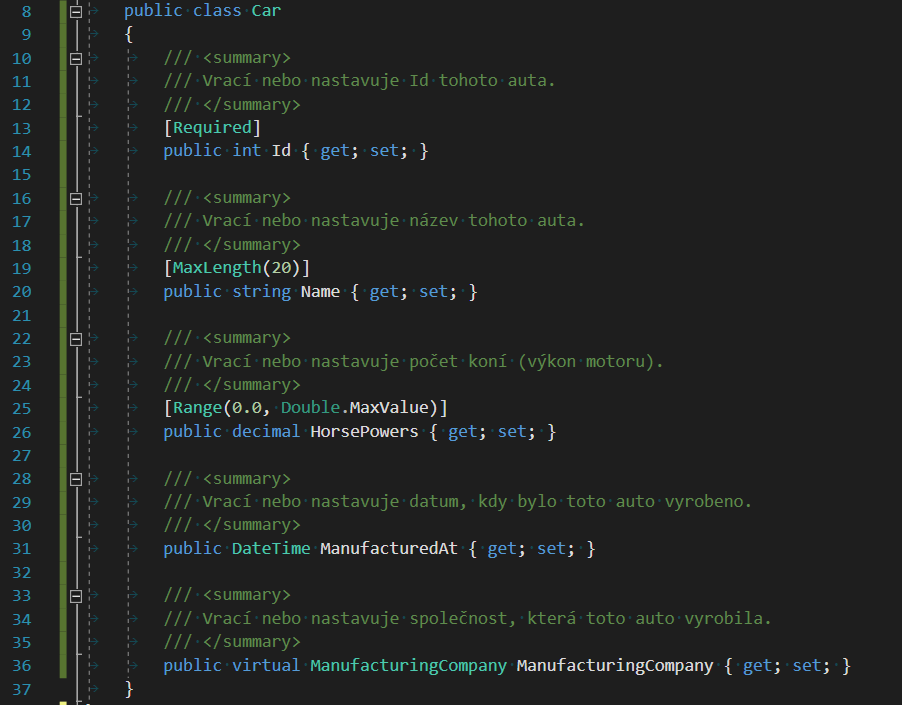
Obrázek 25 Adresářová struktura datové vrstvy

#### Entita uživatele



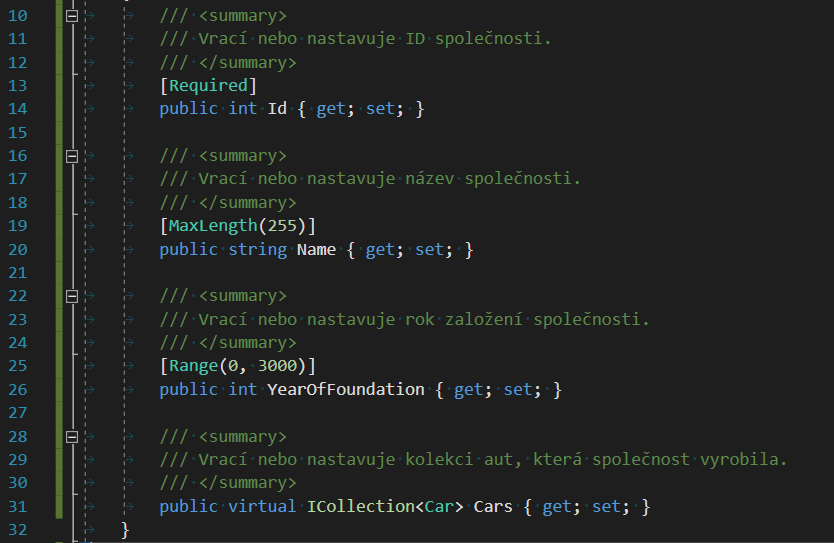
Obrázek 26 Entita uživatele

#### Entita auta



Obrázek 27 Entita auta

#### Entita společnosti vyrábějící auta



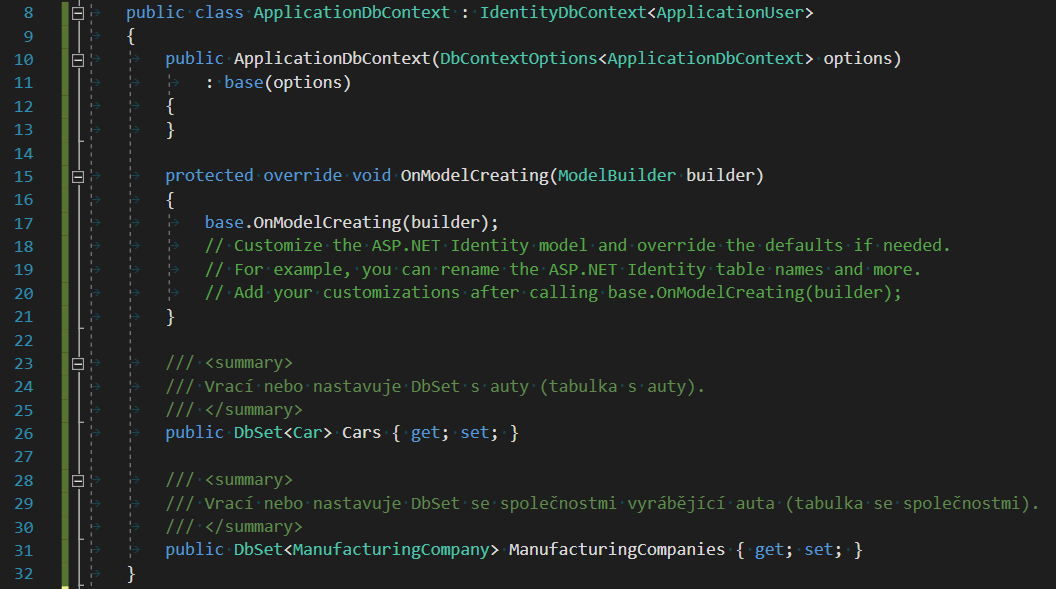
Obrázek 28 Entita společnosti vyrábějící auta

* + 1. Vytvoření DbContext

Abychom mohli 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 „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 obrázku 29.



Obrázek 29 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 obrázku 29 můžete ještě vidět, že třída ApplicationDbContext má 2 vlastnosti typu DbSet. Tyto vlastnosti pro nás vlastně představují jednotlivé tabulky v naší databázi.

#### Využití datové vrstvy v prezenční vrstvě

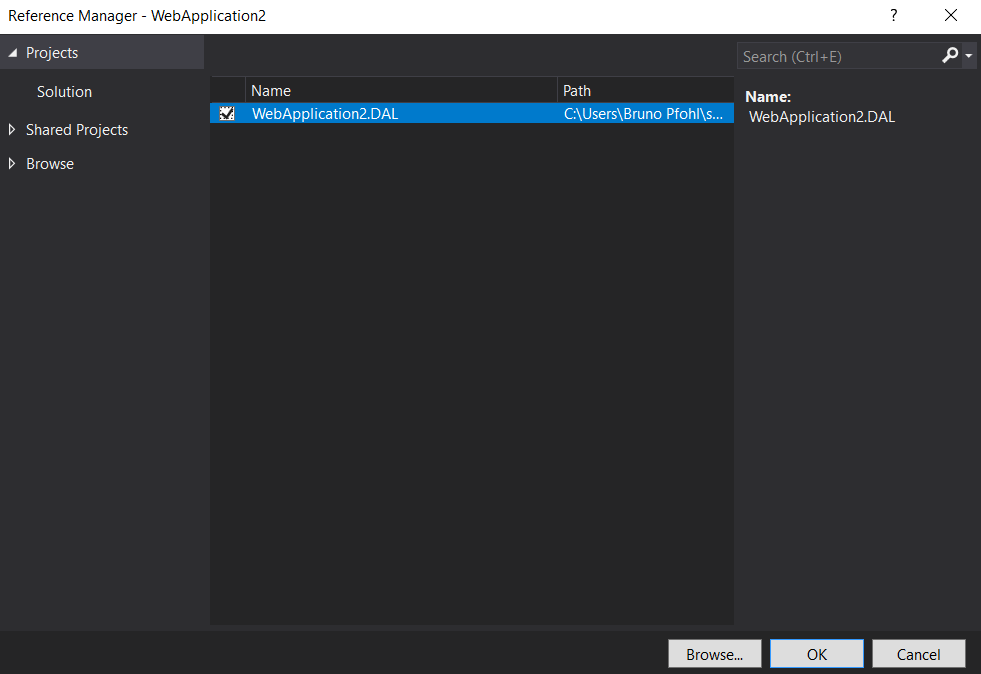
Abychom mohli naší datové vrstvy využít, musíme na ní v projektu prezenční vrstvy přidat referenci. Dále musíme ve třídě Startup přidat ApplicationDbContext jako službu, k čemuž je nutné do projektu prezenční vrstvy přidat potřebné balíčky pomocí Nuget Package Manageru.

Postup

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

Potřebné balíčky

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



Obrázek 30 Přidání reference na datovou vrstvu

1. Tvorba ukázkové aplikace

Cíl ukázkové aplikace …

* 1. Ukázka funkčnosti
  2. Struktura projektu

1. Ověření výstupu na studentech

Úvod o hodinách s kazdou, P4, atd…

* 1. Jak byla odezva získána?
  2. Odezva jednotlivých studentů
  3. Shrnutí a ponaučení

Závěr

Tak jsem to dokázal! A teď ještě, co jsem mohl udělat jinak a jak by se dalo v práci pokračovat.

Seznam obrázků

**Nenalezena položka seznamu obrázků.**

Použitá literatura

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