VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2016 Veronika Foltýnová

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

WEBOVÁ APLIKACE PRO MONITOROVÁNÍ PROCESU VÝROBY
WEB APPLICATION FOR MONITORING THE PRODUCTION PROCESS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Veronika Foltýnová**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Ivo Lattenberg, Ph.D.**
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Studentka: Veronika Foltýnová ***ID:*** 164828
Ročník: 3 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Webová aplikace pro monitorování procesu výroby

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte problematiku monitoringu a logistiky ve výrobě. Definujte klíčové vlastnosti aplikace, která by měla podporou při procesech ve výrobě. Proveďte rozbor existujících frameworků, které jsou vhodné jednak pro tvorbu webové aplikace (MVC, Razor), tak pro přístup k MS-SQL databázi (jako např. Entity Framework, NHibernate apod.). Vytvořte webovou aplikaci, která bude sloužit jako podpora monitoringu a logistiky ve výrobní firmě. Aplikaci implementujte s využitím frameworků, které byly vybrány v rámci teoretického rozboru. Data budou ukládána na MS-SQL serveru.

DOPORUČENÁ LITERATURA:

[1] VIRIUS, M., C# 2010 Hotová řešení, Computer Press, 2012, 424 s., ISBN 978-80-251-3730-7

[2] WATSON, B., C# 4.0 - řešení praktických programátorských úloh, Zoner Press, 2010, 656 s., ISBN 978-8--7413-094-6

[3] STANEK, W.R., Microsoft SQL Server 2012 Kapesní rádce administrátora. Computer press, 2013, 544 s., ISBN: 9788025137970.

Termín zadání: 1.2.2016 ***Termín odevzdání:*** 1.6.2016

Vedoucí práce: doc. Ing. Ivo Lattenberg, Ph.D.
Konzultant bakalářské práce:

doc. Ing. Jiří Mišurec, CSc., *předseda oborové rady*

ANOTACE

Obsah mé práce je zaměřen dvěma hlavními směry. V prvních třech kapitolách jsou shrnuty teoretické poznatky, na jejichž základě bude vytvořena webová aplikace a v poslední kapitole bude rozebrána samotná aplikace.

Nejprve je tedy uvedena problematika monitoringu a logistiky s ukázkou několika příkladů využití webové aplikace v těchto případech. Dále jsou popsány vlastnosti a funkce, které by aplikace měla mít. Třetí kapitola uvádí možné frameworky, jak pro tvorbu webové aplikace, tak pro přístup k MS SQL serveru. U většiny z nich jsou uvedeny příklady jejich použití. Poslední kapitola je věnována popisu a rozboru vytvořené aplikace.

Součástí práce jsou přiložené projekty, které odpovídají příkladům uvedeným v textu a je zde i finální webová aplikace pro monitorování procesu výroby.

KLÍČOVÁ SLOVA

MVC, Entity Framework, NHibernate, LINQ, Razor, Monitoring, Logistika, Visual Studio, C#

ABSTRACT

Contents of this thesis is focused in two main directions. The first three chapters are summarized theoretical knowledges. Based on these knowledges will be created a web application and in the final chapter will be this application analyzed.

At first is mentioned the theme of monitoring and logistics demonstrate of several examples of the use web applications in these cases. The next chapter describes the features and functionality that the application should be. The third section presents possible frameworks, for creating web applications and access to MS SQL server. In most of them are examples of their use. The last chapter is devoted to a description and analysis of the created application.

The thesis contains projects that match the examples shown in the text of thesis and there is final a Web application for monitoring the production process.

KEYWORDS

MVC, Entity Framework, NHibernate, LINQ, Razor, Monitoing, Logistics, Visual Studio, C#

FOLTÝNOVÁ, V. *Webová aplikace pro monitorování procesu výroby*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2016. 49 s. Vedoucí bakalářské práce doc. Ing. Ivo Lattenberg, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma Webová aplikace pro monitorování procesu výroby jsem vypracovala samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autorka uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušila autorská práva třetích osob, zejména jsem nezasáhla nedovoleným způsobem do cizích autorských práv osobnostních nebo majetkových a jsem si plně vědoma následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce doc. Ing. Ivo Lattenbergovi, Ph.D. za užitečnou metodickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne ....

(podpis autora)

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Výzkum popsaný v této bakalářské práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Cílem této práce je prostudovat problematiku monitoringu a logistiky ve firmě, na základě těchto znalostí navrhnout možnosti realizace webové aplikace s využitím .NET frameworku a MS-SQL serveru pro podporu této problematiky a tuto aplikaci vytvořit.

Jelikož jde o poměrně specifické téma, je nutné, aby měl čtenář alespoň základní znalosti jazyka C#, protože příklady, i celá aplikace budou psány v tomto jazyce, ačkoli lze použít i jiné programovací jazyky podporované platformou .NET framework např.: Visual Basic .NET, F# a jiné. Také je potřebná základní znalost ASP.NET. Práce s vývojovým prostředím Visual Studio bude popsána u příkladů. Proměnné v jednotlivých ukázkách i ve výsledné aplikaci budou psány v anglickém jazyce, pouze v jednoduchých příkladech budou proměnné v jazyce českém.

První kapitola se zaměřuje na problematiku monitoringu a logistiky ve výrobě a možnostmi využití webové aplikace pro tyto účely.

Ve druhé kapitole se zaměříme na parametry webové aplikace, její vlastnosti a funkce nezbytné pro její spolehlivé fungování.

Třetí kapitola obsahuje přehled frameworků jak pro tvorbu webové aplikace (MVC, Razor, MVVM), tak pro přístup k MS-SQL databázi (Entity Framework, NHibernate a LINQ) s jednoduchými příklady.

Poslední kapitola je věnována popisu a rozboru vytvořené aplikace.

1 MONITORING A LOGISTIKA VE FIRMĚ

Tyto dva pojmy jsou velmi důležité pro chod firmy. Ať velká či menší, všechny firmy potřebují sledovat to, jak výroba jejich produktů probíhá. K tomu jim slouží řada systémů. Různé firmy nabízejí mnohá řešení. Používají různá sběrná zařízení, případně monitorovací programy. Tyto systémy jsou pro firmu často nákladné, i když většinou spolehlivé a mají technickou podporu, kterou poskytuje firma dodávající tyto systémy.

Webová aplikace může být náhradou za tyto nákladné systémy.

1.1 Monitoring

Monitorování (monitoring) výroby je důležité pro její bezproblémový průběh. Čím rozsáhlejší výroba, tím důležitější je její monitorování. Pod pojmem monitorování neboli sledování výroby si můžeme představit množství úkonů, které zpřehledňují a zjednodušují orientaci v jednotlivých částech výroby.

1.1.1 Příklad využití webové aplikace pro monitorování výroby

Každá výroba má svůj schválený postup. Zjednodušeně můžeme říci, že nejdříve přijde poptávka zákazníka, poté ji vedení schválí, a tím odstartuje výrobu. Zakázka se objeví v monitorovacím programu jako aktivní. Každá zakázka bude obsahovat určité údaje jako například: co a z čeho se má vyrobit. Program může například obsahovat funkci, která po stisku tlačítka či automaticky vyhodnotí, zda je na skladě materiál potřebný k dané výrobě. Dle toho může informovat nákupčí pro firmu nebo daný sklad, aby nakoupil nebo vychystal materiál k výrobě. Sklad po vychystání předá materiál pracovníkovi přepravy, který jej odveze k danému stroji ke zpracování. Obsluha stroje materiál přijme a zadá stroji výrobu. Po vyrobení daného množství předá výrobky pracovníkovi přepravy, který je odveze buď k dalšímu zpracování či do skladu, kde budou čekat na přepravu k zákazníkovi.

Všechny úkony při výrobě budou zaznamenávány do programu tak, jak je příslušní pracovníci zadají, zapíší či pouze odsouhlasí.

Vytvořená aplikace bude sloužit pouze k zobrazování zadaných zakázek a jejich detailů a přehledu strojů ve výrobě.

1.2 Logistika

Tímto pojmem můžeme označit soubor úkonů, které mají zajistit, aby bylo dané zboží či materiál v požadovaném množství ve správnou dobu na správném místě. Zajištění efektivity a rychlosti je zde tedy velmi důležité.

1.2.1 Příklad využití webové aplikace pro logistiku výroby

Ve výrobním procesu je důležité, aby vše bylo v danou dobu a v potřebném množství

na správném místě. Pracovník přepravy bude vybaven přenosným zařízením (např.: menším tabletem či mobilním telefonem s internetovým připojením), na kterém zobrazí webovou aplikaci. Zde uvidí odkud, co a kam má převézt, případně i v jakou dobu má být daný materiál či zboží na určeném místě. Po odvedení své práce zaznamená v aplikaci, že přepravu provedl.

Aplikace může obsahovat přihlašování jednotlivých pracovníků přepravy. V takovém případě bude dobře dohledatelné, kdo a kdy danou přepravu provedl.

Vytvořená aplikace bude obsahovat pouze zobrazování transportů.

2 PARAMETRY WEBOVÉ APLIKACE

Jedním ze základních předpokladů pro vytvoření dobré aplikace je stanovení základních vlastností a funkcí, které bude aplikace umět.

2.1 Vlastnosti

- Snadná ovladatelnost – obsluhu aplikace musí zvládnout i osoba s malou znalostí informačních technologií
- Přehlednost – jednotlivé prvky programu např.: základní menu, přihlášení uživatele nebo výpisy seznamů či událostí je třeba rozmístit tak, aby byli snadno dostupné a logicky umístěné.

2.2 Funkce

- Přihlašování uživatelů
- Výpisy a shrnutí – pro úplnost a celistvost programu je nutné, aby uměl přehledně zobrazit požadované údaje.
- Propojení s databází – jakýkoli program pracující s rozsáhlým množstvím dat potřebuje připojení k databázi.

3 FRAMEWORKY

Pro projekt webové aplikace je zapotřebí vybrat adekvátní framework, a to jak pro tvorbu webové aplikace, tak pro přístup k MS-SQL databázi.

K vývoji jakékoli aplikace je třeba vývojového prostředí (např.: Microsoft Visual Studio, SharpDevelop). Webová aplikace bude napsána v jazyce C#. Pro tento programovací jazyk je ideálním vývojovým prostředím Microsoft Visual Studio. Verzi Express je možné zdarma získat zde [5].

3.1 Frameworky pro tvorbu webové aplikace

3.1.1 Razor

Jedná se o značkovací jazyk používaný k tvorbě dynamických webových stránek či webových aplikací v programovacích jazycích .NET (C#, Visual Basic). Nejde tedy o programovací jazyk. Jde o soubor pravidel, díky kterým můžeme spojit značkovací jazyk HTML a programovací jazyk C# nebo Visual Basic.

Využívá se zde Razor engine, který slouží jako syntaktický analyzátor. Razor engine zpracovává soubory s příponou cshtml (v případě jazyka C#,) nebo .vbhtml(v případě jazyka Visual Basic).

Razor se často používá například v návrhovém vzoru MVC (Model View Controller) jako komponenta view. Příklad takového využití syntaxe Razor najdeme v kapitole 3.1.2.

Výhody syntaxe Razor:

- Podporuje IntelliSense (automatické doplňování příkazů)
- Snadné testování (jednotkový test)

Základní syntaxe kódu

- @{} označuje blok kódu

```
@{
      Int i = 23;
}
```

- @ uvozuje příkaz

```
@DateTime.Now.Year
```

- @* *@ označuje komentář strany serveru

```
@*
      Komentar.
*@
```

- Každý příkaz v bloku končí středníkem

3.1.2 MVC (Model View Controller)

Architektura Model-View-Controller (zkráceně MVC) rozděluje aplikaci do tří hlavních částí: model (datový model), view (pohled – uživatelské rozhraní) a controller (kontrolér, řadič – řídící logika). Základní myšlenkou je, že všechny tyto tři části, často označované jako komponenty, jsou na sobě v maximální možné míře nezávislé, změny provedené v jedné komponentě tedy neovlivní zbylé dvě komponenty.

Tento architektonický vzor původně vznikl pro desktopové aplikace, avšak dnes se více využívá ve webových aplikacích. Zde se zaměříme na ASP .NET MVC, tedy vzor MVC ve webových aplikacích.

Popis jednotlivých komponent

- Model

  Model obsahuje logiku a vše co do ní spadá. Může obsahovat výpočty, databázové dotazy, a podobně. Jeho funkce spočívá v přijetí parametrů zvenku a poslání dat ven. Jde o parametry metod ve třídě modelu. Model neví, odkud data v parametrech přišla a ani jak budou výstupní data zformátována a vypsána.

- View

  View (pohled) se stará o zobrazení výstupu uživateli. Jedná se o HTML šablonu, obsahující HTML stránku a tagy speciálního jazyka, který umožňuje do šablony vkládat proměnné, případně provádět iterace (cykly) a podmínky.

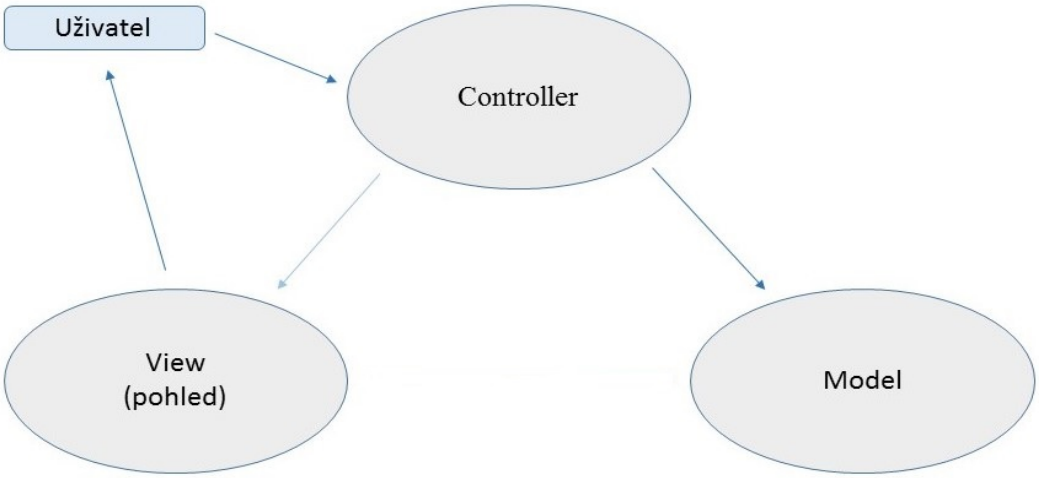
  Obsahuje minimum logiky, která je pro výpis nutná např.: kontrola vypsání všech polí v přihlašovacím formuláři.

  View stejně jako model neví, odkud data přišla, zajišťuje pouze jejich zobrazení uživateli.

- Controller

  Jedná se o prostředníka, který řídí akce mezi uživatelem, modelem a view.

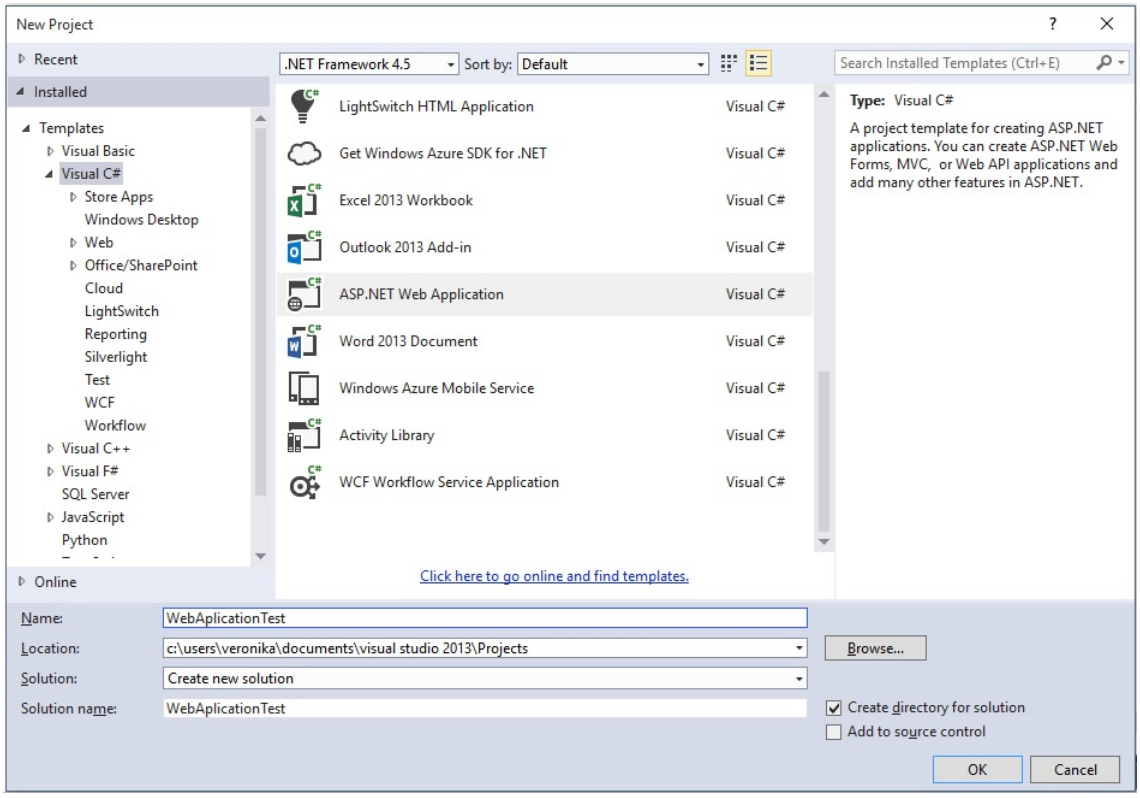
V blokovém schématu (Obr. 2.1) je zobrazený průběh událostí. Nejprve uživatel provede nějakou akci v uživatelském rozhraní. Tuto akci zachytí controller a rozhodne, jak na akci zareaguje. Nejčastěji provede změny (aktualizuje údaje) v modelu. Komponenta view použije změněný model a dle něj zobrazí data uživateli.



Obr. 2.1: Blokové schéma návrhového vzoru MVC

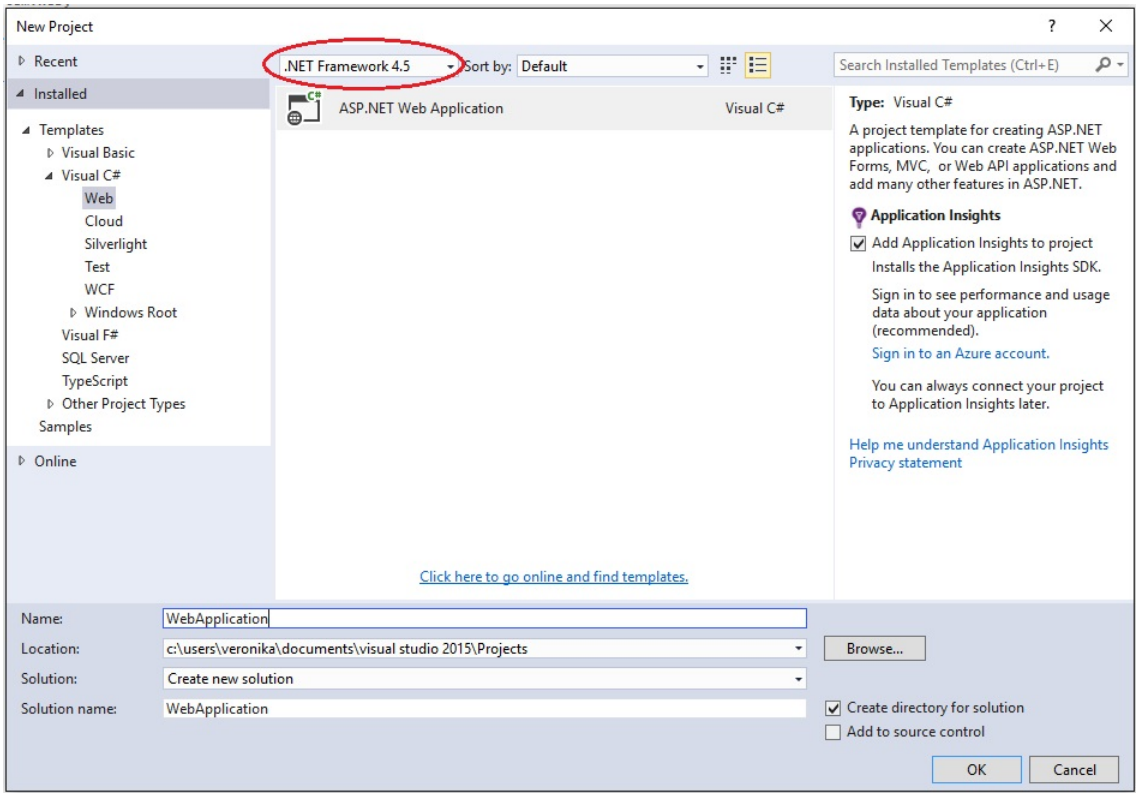
Ukázka webová aplikace využívající MVC

Ve Visual Studiu založíme nový projekt File – New – Project. Z nabídky vlevo vybereme záložku Visual C#. Typ projektu zvolíme ASP.NET Web Application. Název zvolíme dle toho, jaký projekt budeme vytvářet. Zde nám postačí název WebApplicationTestMVC. Dialogové okno pro výběr nového projektu vidíme na Obr. 2.2.



Obr. 2.2: Dialogové okno pro založení projektu MVC

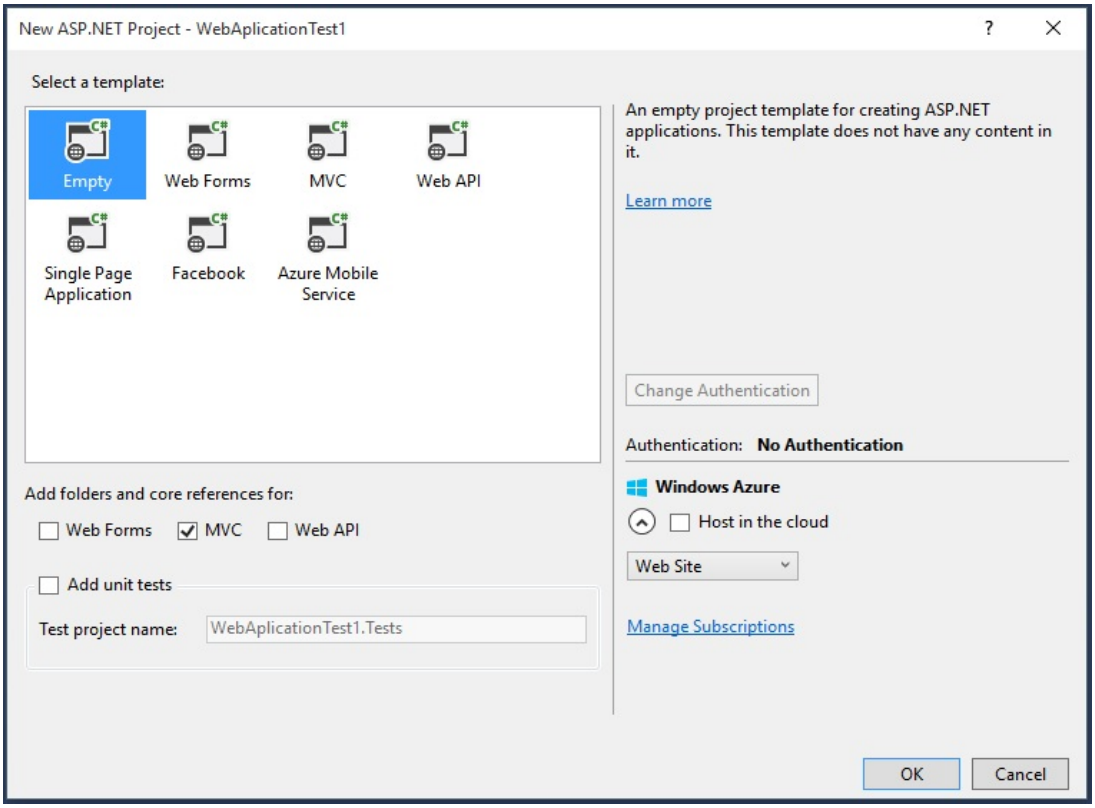
Pokud máte verzi Visual Studio Express for web a nevidíte zde typ projektu ASP.NET Web Application tak zkontrolujte, zda máte zvolený .NET framework 4.5.



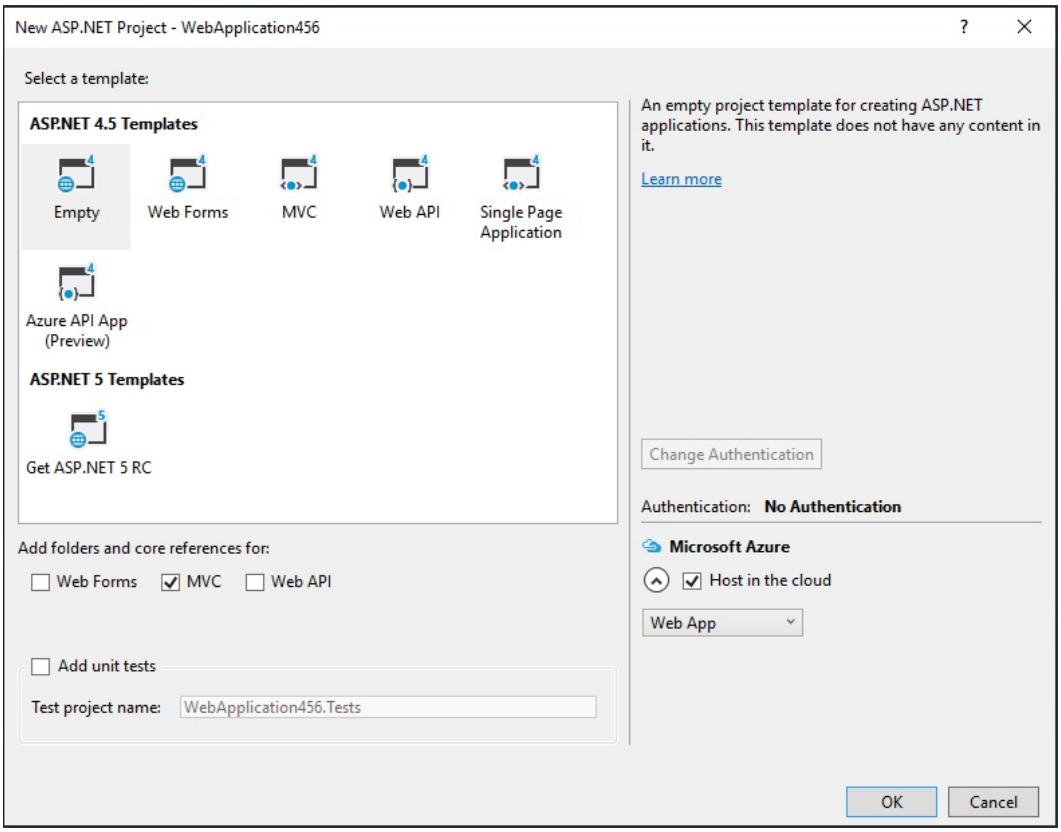
Obr. 2.3: Dialogové okno pro založení projektu MVC – Visual Studio Express for Web

Volbu potvrdíme stiskem OK.

V novém okně zvolíme Empty a zaškrtneme pole MVC dle Obr. 2.4. ve Visual Studiu Express for Web dle Obr. 2.5.



Obr. 2.4: Dialogové okno volby typu aplikace



Obr. 2.5: Dialogové okno pro založení projektu MVC

Ačkoli jsme zvolili prázdnou aplikaci, Visual Studio již předpřipravilo několik souborů a složek. Můžeme je vidět vpravo v okně Solution Explorer.

Nyní si ukážeme jak vytvořit postupně všechny komponenty modelu MVC. Příklady tříd jsou pouze ukázkové, nemají tedy žádný konkrétní význam a slouží jen pro názornou ukázku.

Začneme tvorbou modelu. Najdeme složku s názvem Models a přidáme do ní třídu. To provedeme stiskem pravého tlačítka na tuto složku a vybereme Add – Class. Třídu nazveme Greetings. Zobrazí se následující vygenerovaný kód:

```
namespace WebApplicationTest.Models
{
    public class Greetings
    {
    }
}
```

Do vytvořené třídy můžeme vkládat metody, které budou vykonávat požadované úkony. My zde vložíme jen jednoduchou metodu SayHello(), která vrátí text Hello World. Výsledný kód bude vypadat následovně:

```
namespace WebApplicationTest.Models
{
    public class Greetings
    {
        public string SayHello()
        {
           return "Hello";
        }
    }
    }
```

Nyní vytvoříme Controller. Najdeme složku z názvem Controllers a stiskem pravého tlačítka vybereme Add – Controller. V otevřeném okně zvolíme MVC 5 Controller – Empty. Název přepíšeme na HomeController. Vygenerovaný kód nového controlleru vypadá takto:

```
namespace WebApplicationTest.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
    }
    }
```

Nyní vepíšeme do metody Index() mimo jiné i předání hodnot z modelu do pohledu (view). Toto bude provedeno příkazem ViewBag. Metoda Index() bude vypadat následovně:

```
public ActionResult Index()
 {
     Greetings greetings = new Greetings();
     ViewBag.Znak = greetings.SayHello();    //předání pomocí ViewBag
     return View();
 }
```

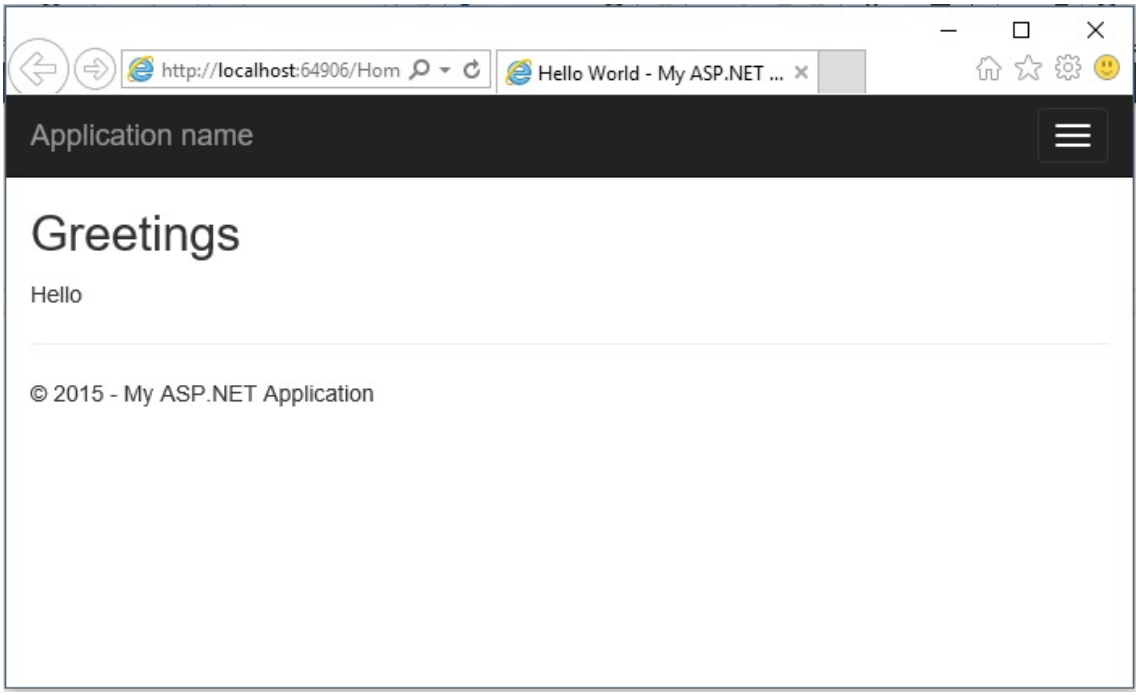
Nakonec vytvoříme metodu pro zobrazení, pohled (view). Nejjednodušším příkladem vytvoření pohledu je kliknutí pravým tlačítkem myši kamkoli do metody Index() a zvolíme Add View. V zobrazeném okně nic neměníme a pouze potvrdíme. Výsledný vygenerovaný kód vidíme níže.

```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
```

Zde můžeme vidět ukázku syntaxe Razor. Více informací o tomto značkovacím jazyce nalezneme v předchozí kapitole 3.1.1. Dále zde vidíme prvky dalšího značkovacího jazyka a tím je HTML. Provedeme úpravy tak, aby byla zobrazena data zpracovaná modelem a předaná controllerem. Výsledný kód je zde:

```
@{
    ViewBag.Title = "Hello World";
    @*comment*@                          //ukázka syntaxe Razor
}

<h2>Greetings</h2>
    <p>@ViewBag.Text</p>
```

Aplikaci nyní spustíme stiskem klávesy F5, nebo kliknutím na symbol zeleného trojúhelníku v horní liště programu. Webovou aplikaci s využitím návrhového vzoru MVC, spuštěnou v internetovém prohlížeči Internet Explorer vidíme na Obr. 2.6.



Obr. 2.6: Zobrazení aplikace spuštěné v internetovém prohlížeči Internet Explorer

3.1.3 MVVM (Model View View Model)

Architektura Model View ViewModel (MVVM) je po MVC dalším návrhovým vzorem. Tento vzor se používá spíše v desktopových aplikacích (jde o programy pro stolní počítače) a to v aplikacích typu Windows Presentation Foundation (WPF) a Silverlight.

Tato architektura rozděluje aplikaci do tří částí Model (datový model), View (pohled) a ViewModel, který představuje prostředníka mezi View a Modelem.

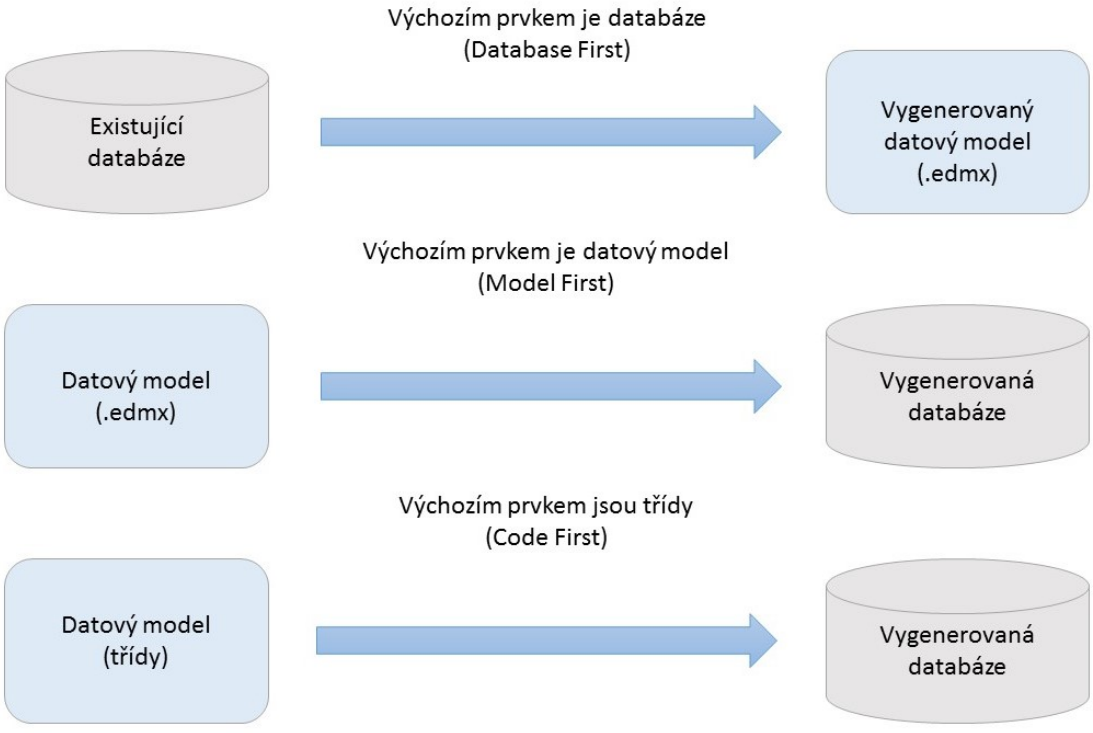
Touto problematikou se zde nebudeme podrobněji zabývat, jelikož vytvářená aplikace bude webová (internetová) a tento vzor je, jak jsem již uvedla, spíše pro desktopové aplikace. Více se o této problematice dozvíte například zde [10].

3.2 Frameworky pro přístup k MS-SQL serveru

3.2.1 Entity Framework

Jedná se o ORM (Object Relational Mapping) framework pro ADO.NET. Je součástí .NET frameworku. Umožňuje vývojářům pracovat s daty relační databáze jako s objekty a umožňuje tak vývojářům se lépe soustředit na psaní samotné aplikace, nikoli kódu na pozadí pro komunikaci s databází. Dává tedy vývojářům nástroj jak snadněji přistupovat k datům v databázi a ukládat je.

Jsou zde tři způsoby, jak pracovat s daty pomocí Entity Framework (Obr. 3.1).



Obr. 3.1: Schéma způsobů práce s daty pomocí Entity Framework

Výchozím prvkem je databáze (Database First)

Pokud máme již vytvořenou databázi v např. MS SQL Serveru, můžeme pomocí Entity Frameworku vygenerovat datový model skládající se z tříd a vlastností, které odpovídají prvkům v databázi, jako jsou například tabulky a sloupce. Informace o databázové struktuře, datovém modelu a mapováním mezi nimi jsou uloženy v souboru XML s koncovkou .edmx. Vývojové prostředí Visual Studio obsahuje možnost grafického návrhu. Můžeme zde zobrazit soubor .edmx.

Výchozím prvkem je datový model (Model First)

V případě, že nemáme vytvořenou databázi, můžeme vytvořit databázový model v grafickém editoru Visual Studia pro Entity Framework. Po dokončení tohoto modelu můžeme nechat editor vygenerovat DDL (data definition language) přehled, který slouží k definování databázové struktury pro vytvoření nové databáze.

Výchozím prvkem jsou třídy (Code First)

Zdali se nechceme zabývat tím, jestli vytvořit či nevytvořit databázi nebo datový model, můžeme začít psát své vlastní třídy a vlastnosti, které odpovídají tabulkám za použití Entity Framework, ovšem bez použití (generování) souboru .edmx. Mapování mezi datovou strukturou a datovým modelem je tvořeno vlastním kódem. Pokud po napsání vlastního kódu budeme chtít vytvořit databázi, může ji pro nás Entity Framework vytvořit.

Ukázka Entity Framework

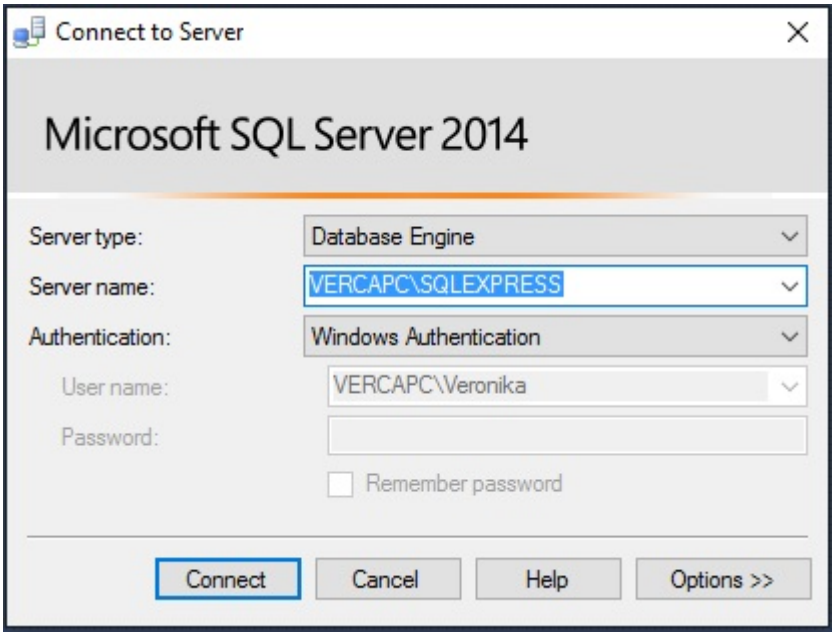
Pro ukázku Entity framework jsem vybrala model u kterého je výchozím prvkem databáze (Database First).

Vývojovým prostředím nám bude Visual Studio a pro externí databázi použijeme MS SQL Server.

Nyní si zde ukážeme jak v programu MS SQL Server vytvořit novou databázi a tabulky s údaji. MS SQL Server můžeme stáhnout zde [4].

Nainstalujeme tedy MS SQL Server. Pokud nemáme s tímto programem zkušenosti, tak při instalaci nic neměníme a řídíme se pokyny instalace. Po nainstalování nalezneme složku Microsoft SQL Server v nabídce Start v záložce všechny aplikace. V této složce nalezneme Microsoft SQL Server Management Studio a spustíme jej.

Po spuštění uvidíme podobnou obrazovku jako na Obr. 3.2. Jde o připojení k serveru pomocí jména serveru, které se vytvořilo při instalaci (jméno počítače\SQLEXRESS) a takzvané Windows Authentication, což znamená, že k ověření uživatele pro připojení k serveru využije účet v počítači (pro systém Microsoft Windows). Existuje zde také vytvoření vlastního účtu (vlastního ověření), ale my si vystačíme s tímto jednoduchým řešením.

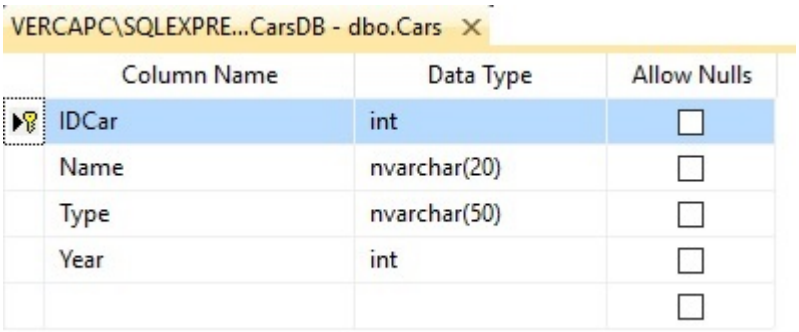


Obr. 3.2: Připojení k MS SQL Serveru

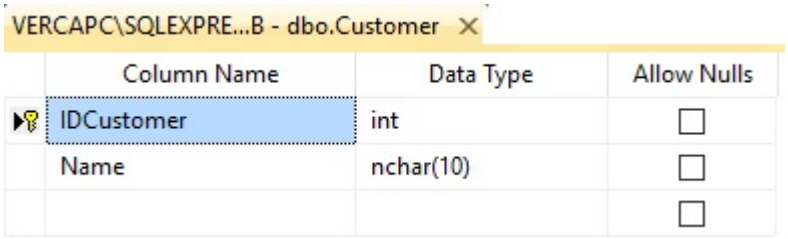
Pro pokračování stiskneme Connect. V levém panelu s názvem Object Explorer se objeví pět složek, ze kterých nás bude zajímat složka Databases. Po kliknutí pravým tlačítkem myši na tuto složku můžeme volbou New Database vytvořit novou databázi. Učiňme tak. V zobrazeném okně vypíšeme pouze položku Database name (jméno databáze). Pro naši ukázku nazveme databázi CarsDB a potvrdíme stiskem OK. Budeme tedy tvořit ukázkovou databázi aut a zákazníků, kteří si chtějí jednotlivá auta koupit. Po otevření složky Databases zde nalezneme nově vytvořenou databázi CarsDB. Otevřeme-li tuto databázi, můžeme tak učinit dvojklikem na její název nebo kliknutím na znak plus vlevo vedle jejího názvu, uvidíme mnoho složek, které za nás vygeneroval MS SQL Server. Pro vytvoření nových tabulek v databázi nás bude zajímat složka s názvem Tables. V konečné databázi budou tři tabulky Cars, Customers, CustomersCars. Ukážeme si zde vytvoření pouze jedné tabulky, ostatní se budou tvořit stejným způsobem. Ukázku vytvoření jednotlivých tabulek nalezneme na Obr. 3.3 – 3.5. Celý projekt, který jsem pro tuto ukázku vytvořila, bude i se zálohou databáze přiložen k bakalářské práci.

Tabulku v databázi vytvoříme kliknutím pravého tlačítka na složku Tables a zvolíme Table. Zobrazí se tři sloupce: Jméno sloupce (Colum Name), datový typ (Data Type) a povolení či zakázání nulové hodnoty daného sloupce (Allow Nulls). Vše vyplníme dle Obr. 3.3. Nyní potřebujeme, aby se položka IDCar stala primárním klíčem neboli jednoznačným identifikátorem. Tuto volbu nalezneme vlevo těsně vedle názvu daného sloupce, zde klikneme pravým tlačítkem a vybereme volbu s obrázkem klíče Set Primary Key. Dále potřebujeme, aby databáze přiřazovala tomuto sloupci automaticky jedinečný identifikátor, tedy nějaké konkrétní číslo, které se nebude v tomto sloupci opakovat. Příslušnou volbu nalezneme pod zadáváním tabulky v okně vlastnosti sloupce (Column Properties) skrývá se zde pod položkou Identity Specification. Dvojím kliknutím na tuto položku se otevře podseznam. Zde změníme v řádku s názvem (Is Identity) hodnotu No na hodnotu Yes. Uložit tabulku a zároveň ji vytvořit můžeme klávesovou zkratkou ctrl + s nebo kliknutím na obrázek diskety v hlavním panelu. Tabulku nazveme Cars. Obdobně vytvoříme ostatní tabulky. Pro kontrolu jsou vytvořené tabulky znázorněny na Obr. 3.3 –

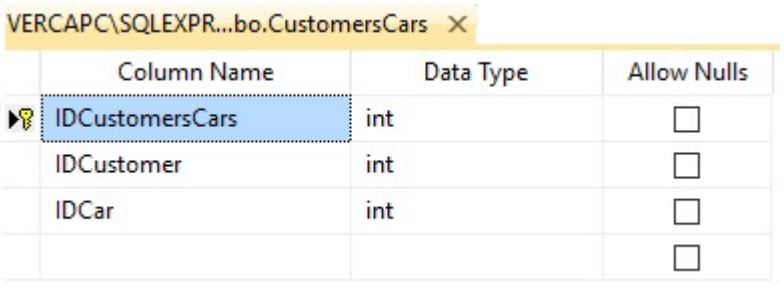
3.5.



Obr. 3.3: Vytvoření tabulky s názvem Cars



Obr. 3.4: Vytvoření tabulky s názvem Customers



Obr. 3.5: Vytvoření tabulky s názvem CustomersCars

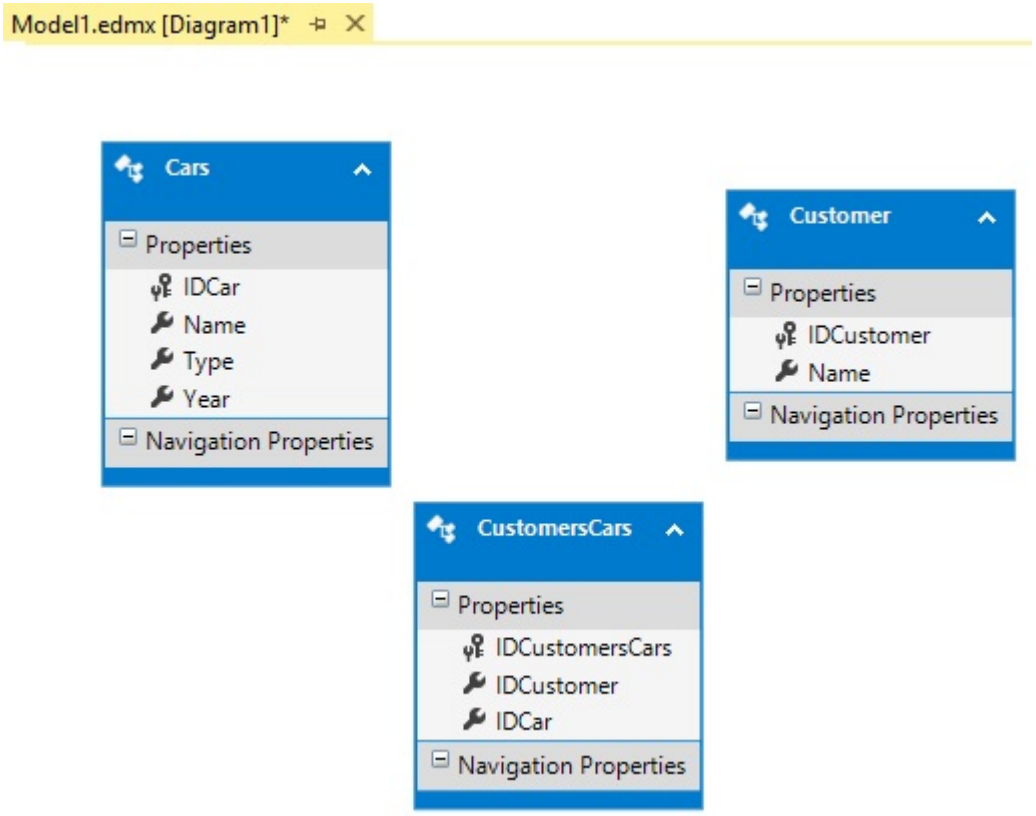
Po vytvoření databáze si založíme nový projekt ve Visual Studiu File – New Project a vybereme ASP.NET Empty web aplication.

Předtím než začneme psát samotný projekt, musíme do něj zavést Entity Framework. Toho docílíme tak, že v hlavním panelu vybereme v menu Tools – NuGet Package Manager – Package Manager Console. Do nově zobrazené konzole napíšeme příkaz Install-Package entityFramework a počkáme, až konzole oznámí jeho úspěšné provedení.

Do nového projektu vložíme nový prvek klikem pravého tlačítka na název projektu dále Add – New Item. V novém okně vybereme z levého seznamu položku data a z nabízených možností zvolíme ADO.NET Entity data model. Jméno modelu můžeme změnit, ale není to nutné.

Zobrazí se nám nabídka možností pro vytvoření modelu. Zde zvolíme EF Designer from database a potvrdíme volbou next. Pomocí tlačítka New connection vytvoříme propojení s databází. V novém okně vyplníme Server name (jméno serveru), což je jméno, které jsme viděli při připojování k MS SQL Serveru a níže vybereme databázi,

ke které se chceme připojit, v našem případě CarsDB. Potvrdíme stiskem OK a posuneme se dál tlačítkem next. Průvodce tvorbou modelu nám nabídne, která data z databáze chceme použít. Zaškrtneme jen políčko u položky tables. Tím jsme vybrali tabulky z databáze. Vše ostatní ponecháme bez změn a tvorbu modelu ukončíme tlačítkem finish. Obsah vygenerovaného souboru Model1.edmx vidíme na Obr. 3.6



Obr. 3.6: Vytvoření tabulky s názvem CustomersCars

K vytvoření tříd pro jednotlivé tabulky klikneme na kteroukoli tabulku pravým tlačítkem a zvolíme Add Code Generation Item. Z nové nabídky vybereme EF 5.x DbContext Generator. Název vypíšeme například ModelCars a potvrdíme (tlačítko Add). Nově vytvořené třídy najdeme v okně solution explorer v rozvíracím seznamu ModelCars.tt.

3.2.2 NHibernate

NHibernate je framework, který umožňuje komunikaci s relační databází objektově orientovaným způsobem. NHibernate překládá použitý objektově orientovaný programovací jazyk do jazyka srozumitelného pro databázi. To znamená, že generuje potřebné SQL příkazy pro vložení, aktualizování, vymazání a načtení dat.

Aplikace psaná v .NET je objektově orientovaná. Používá objekty, které používají data a logické operace a komunikují s jinými objekty pomocí zpráv nebo událostí. Na druhou stranu relační databáze pracuje s velkými soubory dat. Tyto databáze jsou velmi užitečné, pokud je nutné manipulovat s velkými soubory dat. Nicméně v relační databázi koncept objektů nemá smysl. Logika a data zde fungují odděleně.

Při použití tohoto frameworku nemusí programátor měnit styl psaní kódu, nadále píše kód objektově orientovaný a framework tento kód překládá dle potřeb databáze.

V obecné rovině je NHibernate nástroj či framework založený na objektově relačním mapování (ORM).

Více informací o tomto frameworku se můžeme dozvědět zde [7], [9].

NHibernate není jediným ORM frameworkem pro .NET. Některé další frameworky, které existují, jsou: Entity Framework, LLBLGen Pro, Subsonic a Genome.

Ukázka použití NHibernate

Pracovat budeme ve vývojovém prostředí Visual Studia.

Dále je nutno zvolit relační databázi, ke které bude aplikace přistupovat. NHibernate podporuje všechny známé databáze např.: Oracle, MS SQL Server, MySQL a další. Pro náš projekt vybereme MS SQL Server. Verzi Express je možné zdarma stáhnout zde [4].

Nakonec potřebujeme získat Fluent NHibernate [1], který importujeme do našeho vývojového prostředí. S jeho pomocí budeme moci používat framework NHibernate.

Nejprve vytvoříme složku s názvem například NHibernateProject. V ní vytvoříme složku lib do níž nakopírujeme obsah adresáře Fluent NHibernate a složku s názvem NHibernateTest.

Ve Visual Studiu založíme nový projekt ASP.NET Web Application. Jak na to jsme si ukázali v kapitole 3.1.2, ale tentokrát kromě názvu projektu zadáme i cestu, a to do složky NHibernateTest, kterou jsme si založili před chvílí. V dalším kroku zvolíme typ projektu Empty a necháme všechna pole ve volbě přednastavení adresářů nezaškrtnutá.

Předtím, než začneme psát samotný projekt, musíme do něj zavést NHibernate. Toho docílíme tak, že v hlavním panelu vybereme v menu Tools – NuGet Package Manager – Package Manager Console. Do nově zobrazené konzole napíšeme příkaz Install-Package FluentNHibernate a počkáme, až konzole oznámí jeho úspěšné provedení.

Do vytvořeného projektu vložíme postupně několik tříd a jeden soubor aspx. Nejprve napíšeme kód. Vytvoříme novou třídu stiskem pravého tlačítka na název projektu a zvolíme Add – Class. Třídu pojmenujeme Car. Rovnou vytvoříme další tři třídy s názvy Brand, CarMapper a BrandMapper. Pro názornost vytvoříme soubor aspx podobně jako třídu Add – New Item – Web z nabízených možností zvolíme Web form. Nazveme jej MainPage.aspx.

Nejprve vložíme kód do třídy Cars.

```
public class Car
{
    public virtual int Id { get; set; }
    public virtual string Name { get; set; }
    public virtual string Color { get; set; }
    public virtual Brand Brnad { get; set; }
    public virtual decimal Price { get; set; }
```

```
    }
```

Dále vložíme kód do třídy Brand.

```
  public class Brand
    {
        public virtual int Id { get; set; }
        public virtual string Name { get; set; }
    }
```

Zde jsme definovali hodnoty a jejich datové typy, ze kterých budeme tvořit položky tabulek.

Nyní je třeba vyplnit třídu CarMapper.

```
 public class CarMapper:ClassMap<Car>
    {
        public CarMapper()
        {
            Id(x => x.Id);
            Map(x => x.Name);
            Map(x => x.Color);
            Map(x => x.Price);
            References(x => x.Brnad);
        }
    }
```

A obdobně třídu BrandMapper.

```
public class BrandMapper:ClassMap<Brand>
    {
        public BrandMapper()
        {
            Id(x => x.Id);
            Map(x => x.Name);
        }
    }
```

Zde jsme vytvořili třídy, které definují to, jak mají být budoucí tabulky Car a Brand definovány v databázi. Třídy Car a Brand budou namapovány do databáze, která bude obsahovat dvě stejnojmenné tabulky.

Už zbývá jen poslat příkaz pomocí NHibernate, aby byly v dané databázi vytvořeny tabulky dle tříd. K tomu poslouží soubor MainPage.aspx. Otevřeme si jej a přidáme tlačítko. Nejjednodušším způsobem jak toto udělat je přepnout zobrazení souboru na desig, což najdeme v dolní části okna. Na levém okraji okna najdeme panel tolbox, případně jej můžeme najít na hlavním panelu v menu View – Toolbox. Najdeme položku button a přetažením ji umístíme do okna souboru MainPage.aspx. Dvojím kliknutím na toto tlačítko se přesuneme do kódu na pozadí, který se provede po stisku tlačítka.

Do zobrazené třídy MainPage.aspx.cs vložíme tento kód:

```
string conString = "Data Source=VERCAPC\\SQLEXPRESS;Initial
                   Catalog=TestNHibernate;Integrated Security=True";

        protected void Button1_Click(object sender, EventArgs e)
        {
          Fluently.Configure()
                .Database(MsSqlConfiguration
                .MsSql2012.ConnectionString(conString))
                .Mappings(m=>m.FluentMappings
                .AddFromAssemblyOf<CarMapper>())
                .ExposeConfiguration(CreateSchema)
                .BuildConfiguration();
        }

        private static void CreateSchema(Configuration cfg)
        {
            var schemaExport = new SchemaExport(cfg);
            schemaExport.Drop(false, true);
            schemaExport.Create(false, true);
        }
```

Pro náši ukázku budeme potřebovat vytvořit databázi a propojit ji s Visual Studiem. Vše je podrobně popsáno v kapitole 3.2.1.

Proměnná conString v sobě nese řetězec pro připojení databáze. Tento řetězec získáme, pokud v okně Server Explorer klikneme pravým tlačítkem na název databáze a vybereme properties. Zde najdeme Connection String (řetězec pro připojení k databázi). Je nutné v tomto řetězci zdvojit zpětné lomeno.

Zbytek kódu je syntax frameworku NHibernate. Zbává už jen projekt uložit a spustit. Po spuštění se zobrazí internetový prohlížeč a na internetové stránce bude naše tlačítko. Pokud jsme vše naprogramovali správně, jeho stiskem se vytvoří v databázi dvě nové tabulky. Pohledem do databáze zjistíme, zda tomu tak je.

3.2.3 LINQ (Language Integrated Query)

Jedná se o dotazovací jazyk integrovaný do syntaxe jazyka C# nebo Visual Basic a to od verzí C# 3.0 a Visual Basic 9. Objevuje se v .NET Framework 3.5. Umožňuje dotazování nad různými daty a usnadňuje v nich vyhledávání, třídění a celkově práci s nimi.

Jeho cílem je poskytnout řadu nástrojů pro zlepšení implementace kódu při používání různých architektur.

Pomoci s použitím LINQ při programování nám může buď nepřeberné množství návodů na internetu, nebo ucelená knižní publikace [8].

Typy LINQ

- LINQ to Objects – vytvořeno pro práci s kolekcemi objektů. Pro použití stačí zahrnout jmenný prostor System.Linq.
- LINQ to Entities – využívá ORM (Object Relational Mapping) a EDM (Entity Data Model). ORM zajišťuje konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem. EDM je model popisující entity a vztahy mezi nimi. Abstraktní vrstva je zde nezávislá na fyzické datové vrstvě.
- LINQ to SQL – slouží k dotazování nad databází využívající SQL Server. Zde je datový model relační databáze namapován na objektový model v programovacím jazyce vývojáře.
- LINQ to DataSets – umožňuje dotazování na DataSet s využitím LINQ
- LINQ to XML (XLinq) – je rozšířením LINQ to Objects umožňující práci s XML dokumenty i jejich tvorbu.

Klíčová slova LINQ

- From – definujeme zdroj dat, se kterými budeme dále pracovat. Ukázka použití:

  ```
  from c in zvirata
  ```

  Malé c zde představuje položku v souboru dat nazvaných zvirata. Zde se může jednat o databázi, či pouhé pole hodnot.

- Where – tímto klíčovým slovem můžeme více specifikovat výběr dat. Ukázka použití:

  ```
  where c.Length <= 4
  ```

  V tomto případě vybíráme taková data, která mají počet znaků menší nebo roven čtyřem.

- Select – upřesňuje dotaz ovlivňující výstup.

  ```
  Select c;
  ```

  ```
  Select new { c.Zvirata, o.DruhyZvirat};
  ```

  V prvním případě jde o nejjednodušší variantu selectu. V druhém případě sbíráme data do dvou proměnných c a o. Pokud z nich chceme udělat výběr, použijeme tento zápis (platí pouze pro programovací jazyk C#).

- Group
- Into
- Orderby
- Join
- Let

Ukázka kódu v LINQ

Jednoduchý příklad zápisu kódu LINQ .

```
string[] nazvyZvirat = {"pes", "kocka", "zaba", "krokodyl", "lev",
"koza", "veverka" };
            var kratkeNazvyZvirat = from c in nazvyZvirat
                                    where c.Length <= 4
                                    orderby c.Length
                                    select c;
            foreach (string s in kratkeNazvyZvirat)
            {
                Console.WriteLine(s);
            }
Console.ReadLine();
```

Nejprve vytvoříme pole řetězců. Poté z nich pomocí LINQ vybereme ty, které mají počet písmen menší nebo roven čtyřem a setřídíme je dle jejich délky. Jednoduchým foreach cyklem je vypíšeme na obrazovku. Stejně tak je můžeme uložit do proměnné a předat dále.

Tento příklad můžeme napsat i zkráceným zápisem, který v LINQ umožňují takzvané Lambda výrazy.

```
string[] nazvyZvirat = {"pes", "kocka", "zaba", "krokodyl", "lev",
"koza", "veverka" };
       var kratkeNazvyZvirat = nazvyZvirat.Where(c => c.Length <= 4).OrderBy
                               (c => c.Length);

            foreach (string s in kratkeNazvyZvirat)
            {
                Console.WriteLine(s);
            }
Console.ReadLine();
```

Ukázka „propojení“ databáze s aplikací pomocí LINQ to Classes ve Visual Studiu

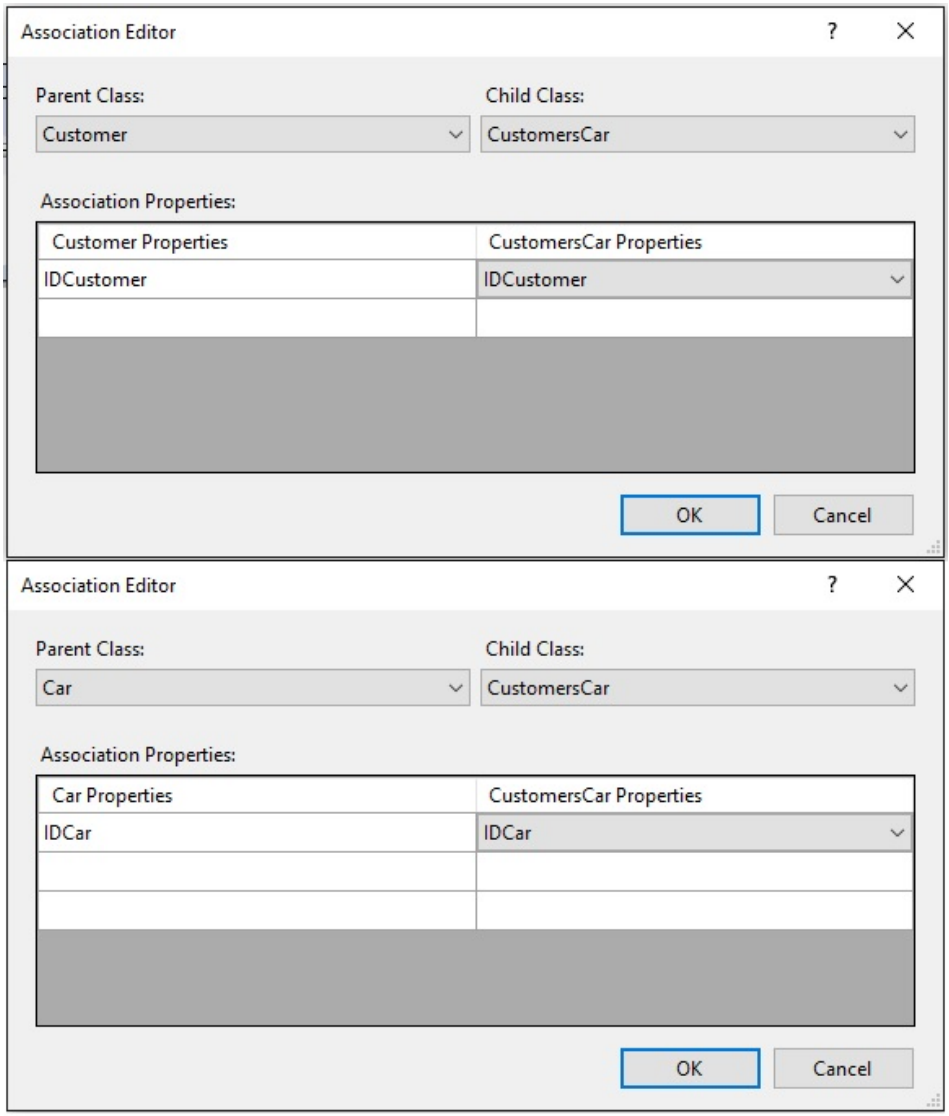
Pro naši ukázku použijeme databázi z kapitoly 3.2.1. Spustíme Visual Studio a založíme nový projekt ASP.NET Web Application. Jak na to jsme si ukázali v kapitole 3.1.2, avšak tentokrát necháme všechna pole ve volbě přednastavení adresářů nezaškrtnutá. Nazveme ji například WebApplicationTestLinq. Vytvoří se čistá webová aplikace, která pro naše účely postačí.

Nyní vložíme do projektu nový prvek, který později převezme tabulky z databáze a vytvoří z nich objekty, se kterými bude možné později pracovat. To provedeme stiskem pravého tlačítka na název projektu (v okně Solution Explorer) a vybereme Add – New Item. V nově otevřeném okně vybereme z levého sloupce položku data a ze zobrazených možností vybereme LINQ to SQL Classes. Název ponecháme a potvrdíme. Visual Studio nám zobrazí prázdnou plochu, kam vložíme tabulky z naší databáze.

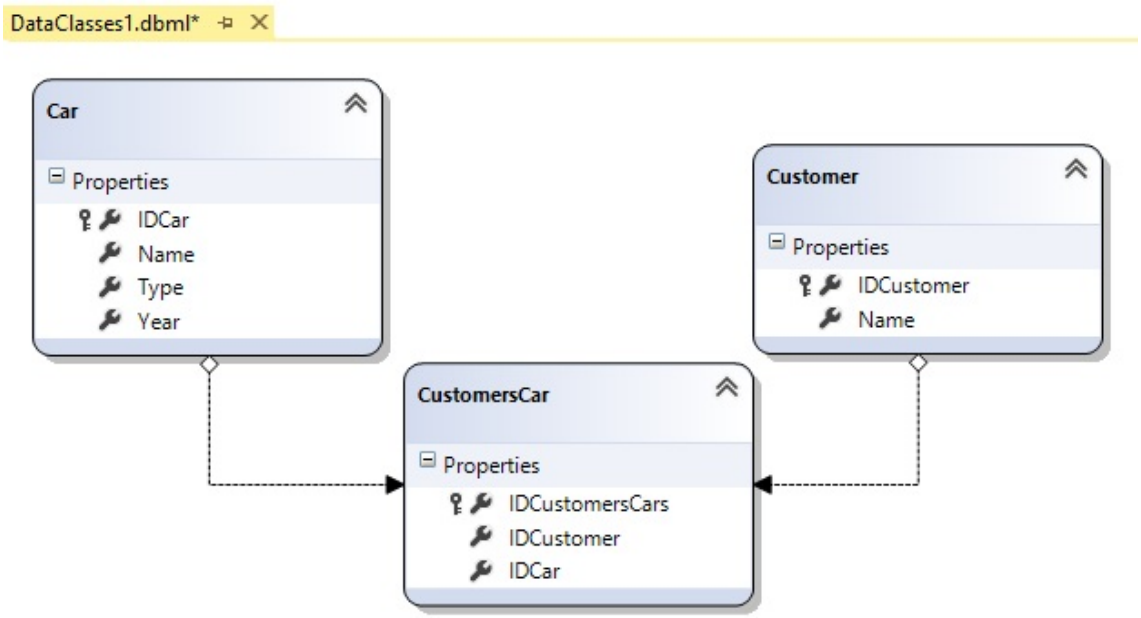
Pro práci s databází ve Visual Studiu ji potřebujeme připojit. K tomu slouží okno Server Explorer. Pokud jej nevidíme, můžeme si jej zobrazit klávesovou zkratkou ctrl+alt+s nebo kliknutím v horní liště na View – Server Explorer. Připojení databáze provedeme kliknutím pravého tlačítka na položku Data Conection – Add Conection.

V novém okně vyplníme Server name (jméno serveru), což je jméno, které jsme viděli při připojování k MS SQL Serveru a níže vybereme databázi, ke které se chceme připojit, v našem případě CarsDB. Otestovat spojení s databází můžeme volbou Test connection. Nakonec naše nastavení potvrdíme.

Nyní jsme připojili databázi k Visual Studiu a můžeme ji vidět v Server Exploreru. Po dvojitém kliknutí na název databáze uvidíme známý seznam adresářů a mezi nimi i složku Tables. V ní nalezneme tři tabulky, které jsme si na začátku vytvořili. Tyto tabulky přetažením umístíme do prázdného místa souboru DataClasses1.dbml. Nakonec je třeba propojit tyto tabulky tak, aby mezi nimi vznikly požadované vazby. Zde je klíčové pochopit, proč máme tři tabulky. Dvě tabulky jsou seznamem aut, zákazníků a údajů o nich. Třetí tabulka propojí dvě předchozí tak, aby mezi nimi vznikly vazby. Například Zákazník A si chce koupit auto B a podobně. To zajistíme pravým klikem na již přetaženou tabulku Customers a výběrem Add – Association. V novém okně vše vyplníme dle Obr. 3.7. Podobně to provedeme i s tabulkou Cars. Tabulky s vazbami vidíme na Obr. 3.8.



Obr. 3.7: Propojení tabulek vazbami – Association Editor



Obr. 3.8: Tabulky s vazbami v souboru DataClasses1.dbml

Úspěšně jsme ukončili ukázku. Nyní po uložení souboru dbml se můžeme podívat do souboru Dataclasses1.designer.cs v němž uvidíme objekty, které pro nás vytvořilo Visual Studio.

4 WEBOVÁ APLIKACE

Na základě poznatků z předchozích kapitol byla vytvořena webová aplikace. Vizí bylo vytvořit aplikaci, která data z databáze výrobní firmy přehledně zobrazí zaměstnanci a ten v nich bude moci vyhledat informace potřebné pro svou práci.

V této kapitole bude shrnuto, co je potřeba k jejímu vzniku, popsána struktura a funkce jednotlivých komponent (částí architektury MVC, jednotlivých tříd a metod).

4.1 Tvorba webové aplikace

K vytvoření webové aplikace je zapotřebí vývojové Studio, databáze a znalost programovacího jazyka. Na základě poznatků z kapitoly tři byl zvolen framework pro tvorbu webové aplikace a to architektonický vzor MVC s použitím syntaxe Razor i framework pro přístup k databázi LINQ. Pro vývoj aplikace bylo vybráno vývojové prostředí Microsoft Visual Studio, pro databázi MS SQL Server. Programovacím jazykem zde bude jazyk C#.

Jak již bylo uvedeno MS SQL Server lze stáhnout zde [4] a verzi Express vývojového prostředí Visual Studio je možné zdarma získat zde [5].

Teorií, jak vytvořit projekt a základní práce ve vývojovém prostředí Visual Studia se zde již zabývat nebudeme. Tato teorie byla rozebrána v předchozích kapitolách a nebude pro spuštění výsledné aplikace potřebná. Vytvořená aplikace bude obsažena na přiloženém DVD a spustit ji bude možné ve vývojovém prostředí Visual Studia. Vytvoření databáze potřebné ke správnému fungování aplikace nebude nutné. Pro tyto účely bude k bakalářské práci přiložena záloha testovací databáze s koncovkou .bak a v případě nekompatibility s MS SQL Serverem i skript pro vygenerování databáze a její naplnění daty. Jak nahrát zálohu do MS SQL Serveru a jak spustit generovací skript bude naznačeno v následujících podkapitolách.

Projekt bude tedy spustitelný s využitím MS SQL Serveru a Microsoft Visual Studia.

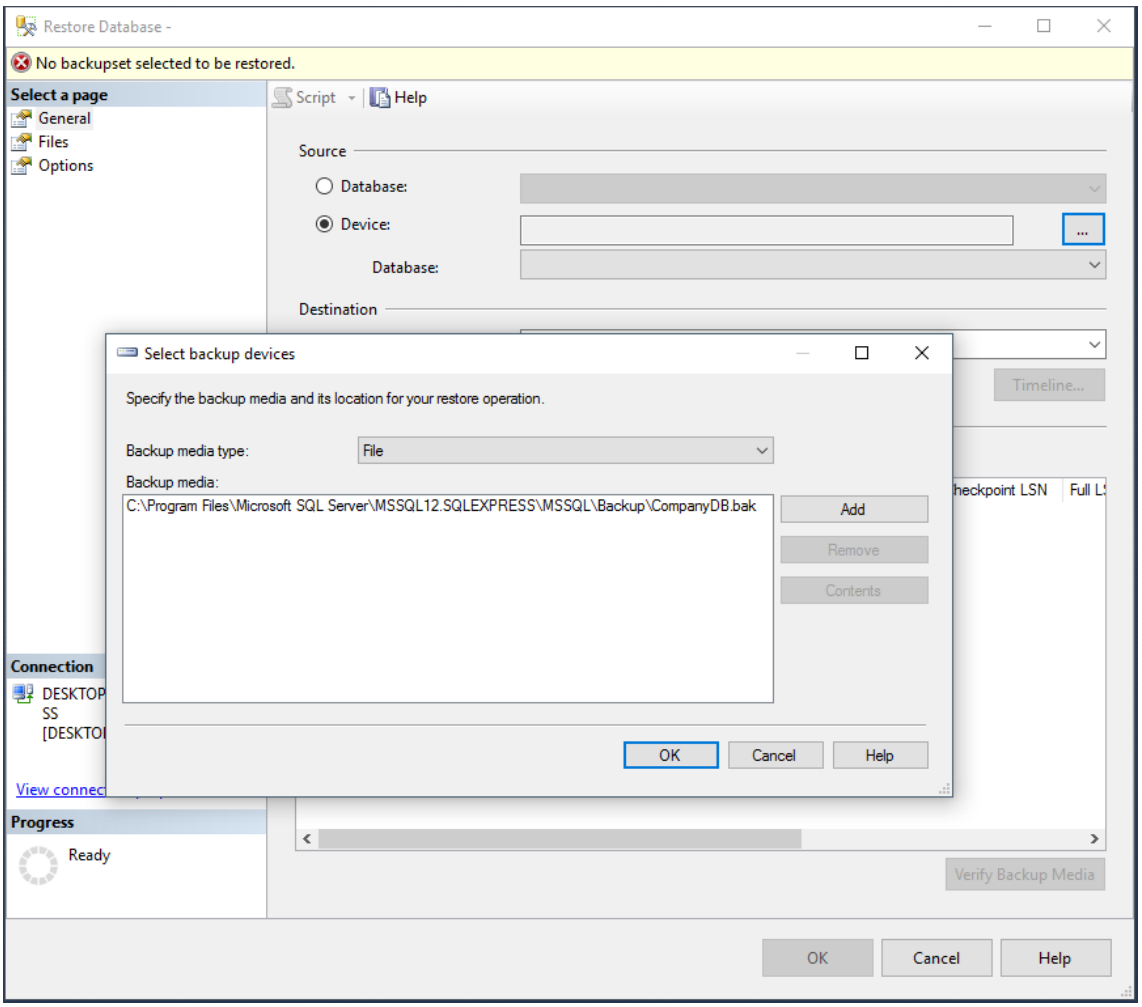
4.1.1 Obnovení zálohy MS SQL Serveru

MS SQL Server ukládá zálohy programů do složky backup. Pokud je nainstalován na systémovém disku nalezneme jej zde C:\Program Files\Microsoft SQL Server\MSSQL12.SQLEXPRESS\MSSQL\Backup. Na DVD přiloženém k této bakalářské práci se nachází soubor CompanyDB.bak. Tento soubor umístíme do již zmiňované složky backup.

Nyní spustíme SQL Server Management Studio. Po přihlášení nalezneme v levém sloupci složku Databases. Klikneme na ni pravým tlačítkem myši a vybereme položku Restore Database. Ve zobrazeném okně Pod nápisem Source klikneme na tečku vlevo vedle řádku Device. Dále vpravo klikneme na tři tečky. Zobrazí se nám podokno jako na obrázku obr. 4.1. Zde klikneme na Add a vybereme soubor CompanyDB.bak. Volbu potvrdíme OK. Pokud vše proběhlo v pořádku, tak se ve spodní části okna pod nápisem back up sets to restore objeví údaje o databázi, kterou chceme obnovit. Pokud není vedle

názvu databáze zatrženo restore učiníme tak. Potvrdíme stiskem OK. Při úspěšném provedení se zobrazí okno Database ´CompadyDB´ restore successfully. Stiskem OK potvrdíme a vlevo ve složce Databases nalezneme obnovenou databázi.

Pokud v průběhu tohoto obnovování došlo k chybě, v kterékoli z jejích částí je třeba vytvořit databázi ze skriptu.



Obr. 4.1:Okno Restore Database a podokno pro vyplnění cesty ke zdroji

4.1.2 Obnovení databáze MS SQL Serveru ze skriptu

Databázi lze kromě obnovení ze souboru obnovit také ze skriptu. Zde nejde přímo o obnovení, ale o vytvoření nové databáze ze skriptu, který byl vygenerován z fungující databáze zde konkrétně z testovací databáze CompanyDB.

Potřebný soubor s generovacím skriptem nalezneme na DVD přiloženém k této bakalářské práci. Jde o soubor CompanyDB.sql. Jelikož máme nainstalován MS SQL Server a s ním i SQL Server Management Studio stačí na soubor dvakrát kliknout. Spustí se Management Studio a po přihlášení se ve středové části aplikace zobrazí skript sql. Klikneme na začátek prvního řádku ve skriptu. Tímto se nám aktivuje tlačítko !Execute v liště nad zobrazeným skriptem nebo stiskem klávesy F5. Stiskem toho tlačítka spustíme provádění skriptu. Pokud vše proběhlo správně zobrazí se v dolní části okna skriptu

zpráva: Command(s) completed successfully a níže si můžeme všimnout žlutého řádku se zeleným kolečkem. Všechna tato hlášení nám oznamují správné provedení skriptu a vlevo ve složce Databases nalezneme novou databázi CompanyDB. Pokud by se v této složce nezobrazila ihned, je třeba kliknout pravým tlačítkem na složku Databases a vybrat položku refresh. Dojde tak k opětovnému načtení dat v této složce.

Takto jsme vytvořili prázdnou databázi. Pro její naplnění musíme spustit skript InsertDataToCompanyDB.sql, který nalezneme na přiloženém DVD. Spustíme jej stejně jako předchozí skript.

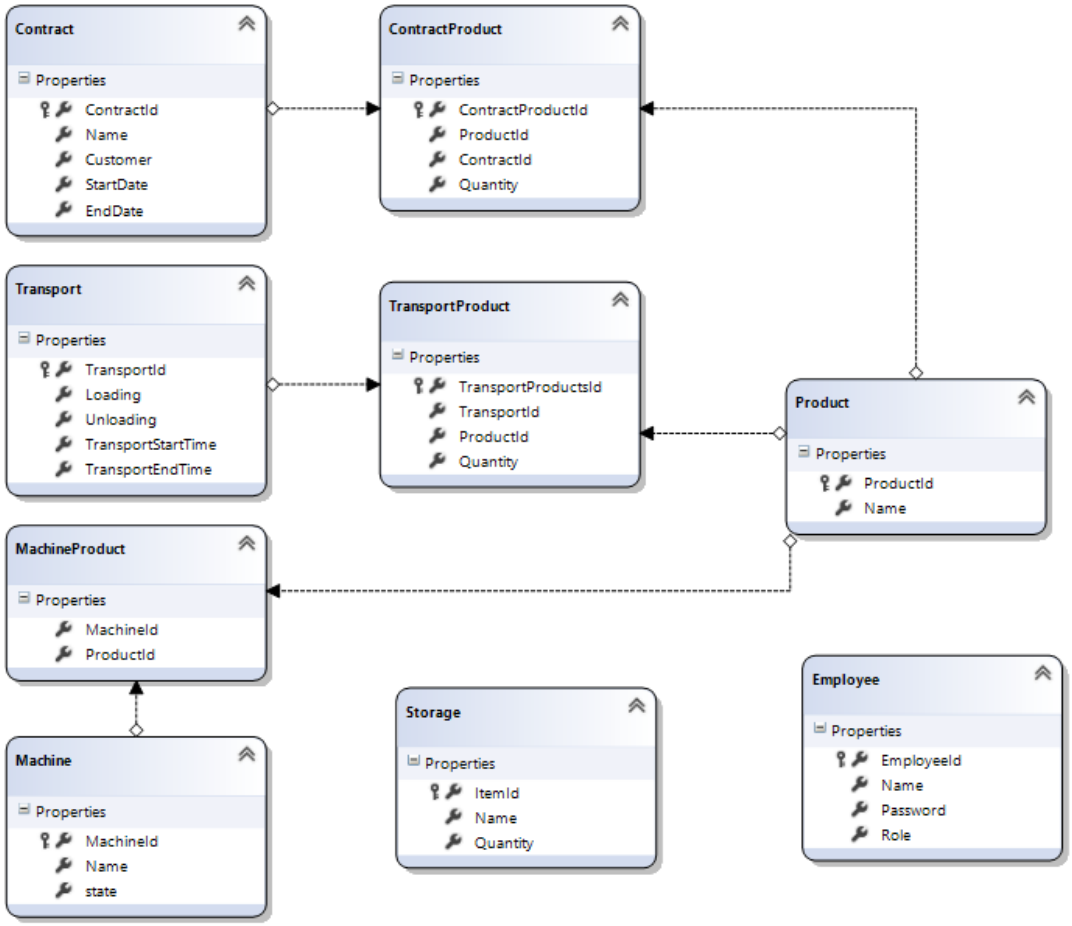
4.2 Struktura testovací databáze

Pro vytváření a testování jakéhokoli programu, který využívá databázi, je nutné tuto databázi mít. Pokud pracujeme na projektu pro konkrétního zákazníka a chceme tedy program „ušít na míru“ přímo jemu, budeme nejspíše pracovat s konkrétní, již existující databází. Případně danou databázi vytvoříme dle jeho požadavků. Nemáme-li pevně danou strukturu databáze (konkrétního zákazníka), musíme si ji vytvořit a naplnit testovacími daty. V předchozí kapitole 4.1, konkrétně v jejích podkapitolách (4.1.1, 4.1.2), jsme si ukázali jak připojit zálohu testovací databáze, případně jak použít skripty pro její vygenerování a naplnění daty. Zde si ukážeme strukturu databáze a vysvětlíme si, co představují jednotlivé její části.

Pro ukázku struktury databáze využijeme schéma ze souboru DataClasses1.dbml, které bylo vygenerováno po propojení databáze s aplikací pomocí LINQ to Classes. Ukázku vzniku tohoto propojení nalezneme v kapitole 3.2.3. Toto schéma pro názornost postačí, ale mohli bychom jej vytvořit také pomocí jazyka UML (Unified Modeling Language).

Na schématu vyobrazeném na Obr. 4.2 vidíme několik prvků propojených přerušovanými čarami s šipkami. Tyto obdélníkové či čtvercové prvky představují tabulky v databázi. Pod názvem jednotlivých prvků jsou vlastnosti představující sloupce v tabulkách. Jelikož jsou v tomto projektu všechny prvky pojmenovávány v anglickém jazyce, bude nyní uveden v závorkách vedle názvu prvku i jeho překlad. Prvky Contract (Zakázka), Transport (Převoz), Product (Výrobek), Machine (Stroj), Storage (Sklad), Employee (Zaměstnanec) představují základní tabulky obsahující důležité údaje, které bude webová aplikace zobrazovat. Aby vše fungovalo správně, je nutné vytvořit mezi základními tabulkami vazby. Toho docílíme dvěma kroky. Nejprve vytvoříme spojovací tabulky (prvky). Každá z těchto tabulek v sobě spojuje údaje ze dvou základních tabulek a případně přidá nějakou další informaci. Spojovacími tabulkami zde jsou ContractProduct, TransportProduct, MachineProduct. Pojmenování těchto tabulek nám napoví, které dvě základní tabulky jsou zde spojovány. Například ContractProduct je tabulka spojující tabulku Contract (Zakázka) s tabulkou Product (Výrobek) a obsahuje navíc vlastnost Quantity (množství). Spojení je zabezpečeno pomocí jednoznačných identifikátorů daných tabulek. V tabulce ContractProduct jde o identifikátory ProductId a ContractId. Každá tabulka obsahuje svůj jedinečný identifikátor. Druhým krokem je vytvoření spojení (přerušované čáry s šipkami). Jde zde o propojení Jeden k mnoha (One To Many). Toto propojení spojuje jeden záznam v základní tabulce s libovolným počtem záznamů v sekundární tabulce. Toto spojení můžeme provést při tvoření tabulek v aplikaci pro tvorbu databáze, což je v našem případě MS SQL Server nebo přímo

ve Visual Studiu po přetažení tabulek do souboru DataClasses1.dbml. Propojení ve Visual Studiu je ukázáno v kapitole 3.2.3. O spojení tabulek v MS SQL Serveru se dočtete například zde [6]. Tabulky Storage (Sklad) a Employee (Zaměstnanec) nemají žádná spojení s jinými tabulkami, protože nejsou potřebná.



Obr. 4.2: Struktura (diagram) testovací databáze

4.3 Struktura webové aplikace z pohledu uživatele

Při tvorbě této webové aplikace byl kladen důraz na jednoduché a intuitivní ovládání. Aplikace může fungovat na monitoru počítače, kde bude ovládána myší, nebo na dotykovém zařízení.

Při prvním spuštění aplikace ve webovém prohlížeči se v její horní části objeví menu. Zde nalezneme čtyři tlačítka vlevo a jedno vpravo, jak můžeme vidět na obrázku Obr. 4.3. Jejich funkci popisují následující podkapitoly.

Obr. 4.3: Základní lišta webové aplikace

4.3.1 Úvod

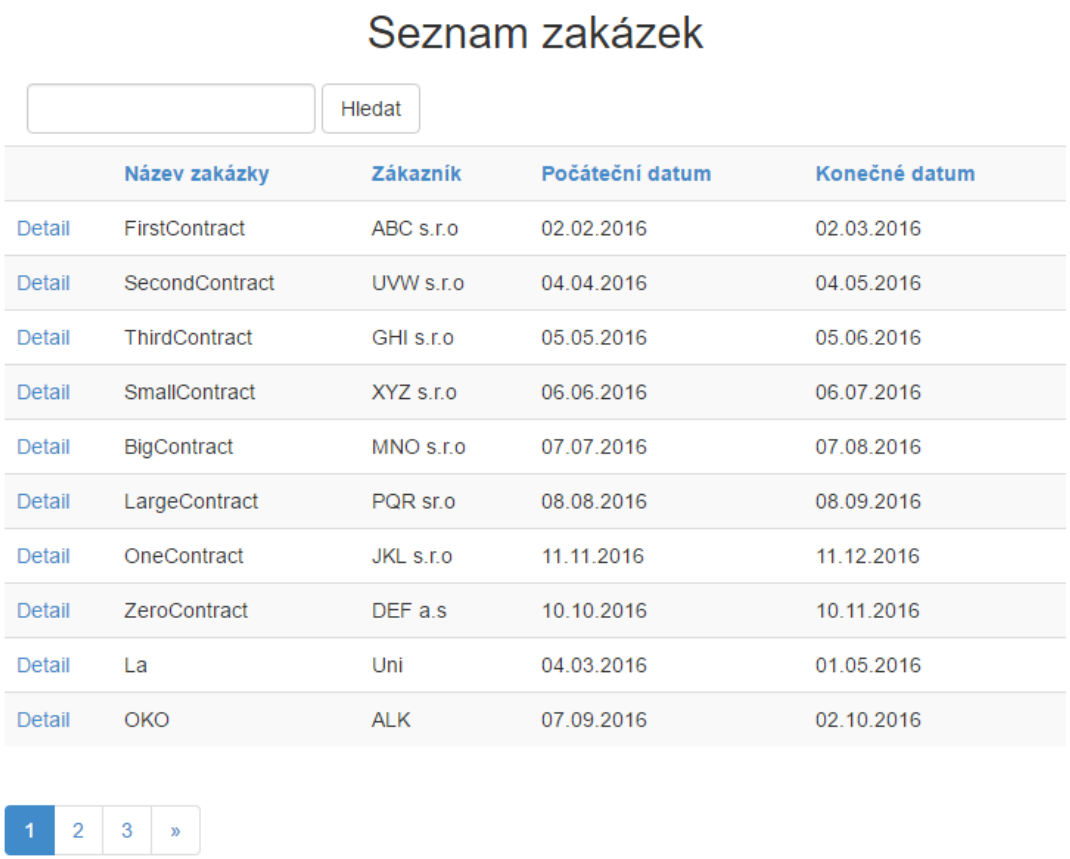
Toto tlačítko, jak již název napovídá, vrátí uživatele na základní, tedy úvodní stránku.

4.3.2 Zakázky

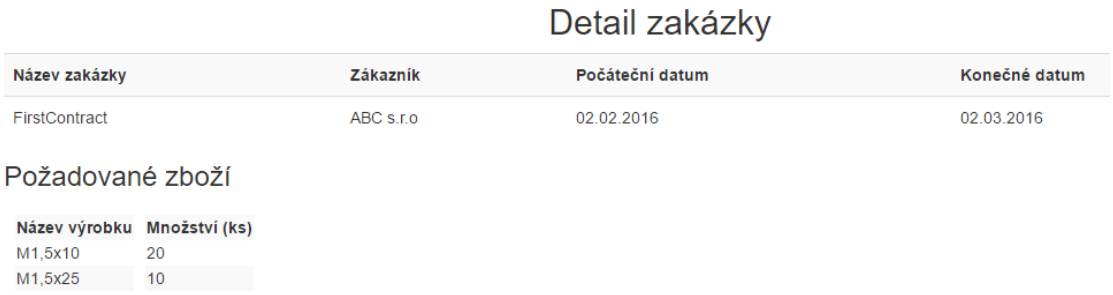
Zde nalezneme tabulku se seznamem zadaných zakázek. V této tabulce je možné pomocí políčka nad ní a následném stisku vyhledávacího tlačítka vyhledávat informace. Dále je zde stránkování, řazení dle nadpisu jednotlivého sloupce a u každé zakázky můžeme zobrazit její detail.

V detailu zakázky se zobrazí spolu se základními parametry i zboží požadované zákazníkem.

Základní zobrazení webové stránky se seznamem kontraktů můžeme vidět na obrázku Obr. 4.4 a detail první zakázky vidíme na obrázku Obr. 4.5.



Obr. 4.4: Zobrazení seznamu zakázek



Obr. 4.5: Zobrazení detailu zakázky

4.3.3 Výroba

Pod tímto tlačítkem nalezneme seznam strojů a jejich momentální stav. Vlevo vedle názvu každého stroje nalezneme také tlačítko detail. Po jeho stisku se zobrazí detail daného stroje a součástky, které vyrábí.

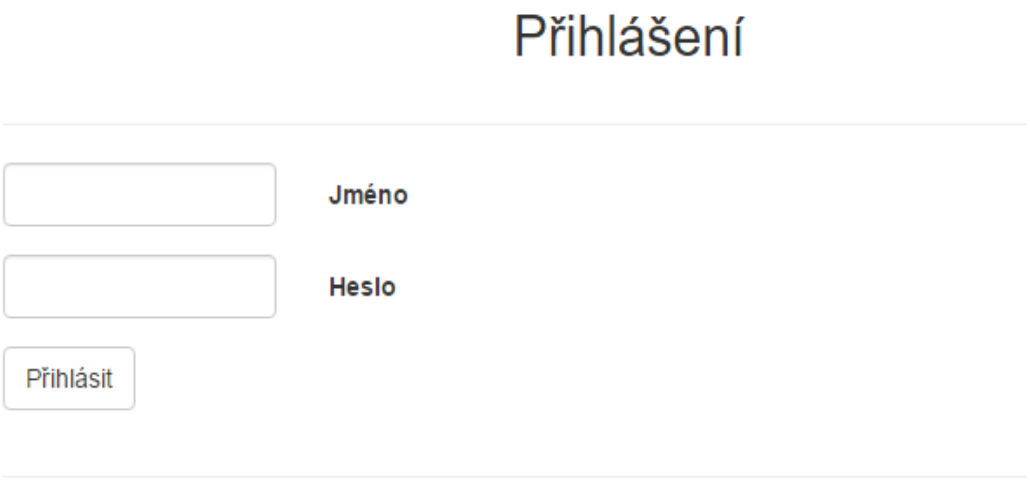
4.3.4 Transport

Po stisku posledního levého tlačítka uvidíme seznam zadaných převozů. V jednotlivých řádcích jsou informace co se má odkud a kam převézt. I zde je vedle každého řádku tlačítko detail. Po jeho stisku uvidíme detail daného převozu i s materiálem, který má být převezen.

4.3.5 Přihlášení

Toto jediné tlačítko umístěné vpravo slouží pro přihlášení uživatele. V této aplikaci jde pouze o symbolické přihlášení uživatelů, jelikož zde nejsou žádná další práva či omezení pro dané uživatele. Jde jen o ukázku dalších možností webové aplikace.

Na zobrazené stránce po stisku tohoto tlačítka uvidíme přihlašovací formulář, který je třeba pro přihlášení vyplnit (Obr. 4.6). Program počítá s tím, že oprávnění uživatelé jsou zaznamenáni v databázi, se kterou údaje porovnává. V případě shody je uživatel přihlášen a přesměrován na úvodní stránku. Pokud dojde k neshodě, program ohlásí chybu.



Obr. 4.6: Přihlašovací formulář

4.4 Struktura webové aplikace z pohledu programátora

Zde je podrobněji rozebrána struktura aplikace z pohledu programátora. Slouží k lepší orientaci v programu, pokud jej někdo bude chtít pochopit, použít některé metody pro své účely, případně aplikaci rozšířit. Zkrácený popis jednotlivých metod nalezneme přímo v programu ve formě komentářů.

Program je tvořen dle architektonického vzoru MVC. Pro View je použit syntax Razor. Skládá se ze dvou kontrolerů, dvanácti pohledů (View) a šesti modelů. Tyto komponenty budou popsány níže. Nacházejí se zde také použité kaskádové styly umístěné ve složce content a soubory s javascriptem ve složce scripts.

4.4.1 Controllers (kontrolery)

Metody v těchto souborech slouží jako spojující článek mezi Modelem a View a k obsluze stejnojmenných View. Jednotlivé vnitřní metody budou popsány blíže v popisu modelu.

AccountController.cs

Tento soubor obsahuje metody LogIn() a LogOut().

- LogIn () – tuto metodu zde nalezneme s parametry i bez parametrů. Metoda Login() bez parametru slouží k prvnímu načtení stejnojmenného View, zatímco metoda s parametry přebírá informace z formuláře ve stejnojmenném pohledu a dle metody NamePassValidation() v podmínce je ověřuje. Pokud jsou data ověřena, dojde k vytvoření nového záznamu o uživateli v paměti programu a přesměrování na hlavní stránku.
- LogOut() – zajišťuje, aby došlo k odhlášení uživatele a přesměrování na hlavní stránku programu.

HomeController.cs

Tento soubor obsahuje metody Index(), Contracts(), Transport(), Production(), DetailC(), DetailT(), DetailP().

- Index() – zajišťuje zobrazení hlavní stránky.
- Contracts() – sbírá hodnoty z View a zasílá je patřičnému modelu ke zpracování. Tato metoda používá metodu ContractList() z modelu Connect.cs.
- Transport() – využívá metodu TransportLists() z modelu TransportModel.cs.
- Production() – využívá metodu ProductionMachines() z modelu Production.cs.
- DetailC(), DetailT(), DetailP() – přebírají hodnoty z pohledů Contracts, Transport a Production a zpracovávají je pomocí dílčích metod. Výsledek posílají do svých stejnojmenných pohledů.

Ukázka kódu kontroleru - AccountController.cs, metoda LogIn()

```
[HttpPost]
public ActionResult Login(Models.Account user)
            {
                if (ModelState.IsValid)
                {
                    if (Models.MyIdentity.NamePassValidation(user.Name,
                        user.Password))
                    {
                        ViewBag.ErrorMessage = "";
                        Session["userName"] = user.Name;
                        System.Web.HttpContext.Current.User = new
                        System.Security.Principal.GenericPrincipal(new
                        Models.MyIdentity(user.Name, "Forms", true,
                        null, user.Password), new string[] { });

                        System.Web.Security.FormsAuthentication.SetAuth
                        Cookie(user.Name, false);
                        return RedirectToAction("Index", "Home");
                    }
                    else
                        ModelState.AddModelError("ErrMessage",
                        "Nesprávné jméno nebo heslo.");
                    ViewBag.log = "err";
                }
                return View();
            }
```

4.4.2 Views (pohledy)

Account

- Login.cshtml – úkolem tohoto View je zobrazit formulář pro přihlášení uživatele.

- Logout.cshtml – tento View nemá žádnou funkci. Je zde kvůli funkčnosti provedených příkazů v metodě Logout(), nacházející se v kontroleru AccountController.

Home

- Contracts.cshtml – zde se zobrazují požadovaná data z databáze, konkrétně seznam zakázek. Zobrazená data lze stránkovat, řadit a vyhledávat v nich. Také lze zobrazit detaily každé zakázky.
- Index.cshtml – slouží k zobrazení hlavní stránky programu.
- Transport.cshtml – v tomto pohledu jsou zobrazeny záznamy převozů z databáze. Také lze zobrazit detaily každého převozu.
- Production.cshtml – tento pohled slouží k vykreslení seznamu strojů a jejich momentálního stavu z databáze. Také lze zobrazit detaily každého stroje.
- DetailC.cshtml – přebírá hodnoty z pohledu Contracts.cshtml a zobrazuje patřičná data, konkrétně detaily zvolené zakázky.
- DetailT.cshtml – přebírá hodnoty z pohledu Transport.cshtml. Vykresluje detaily zvoleného převozu.
- DetailP.cshtml – přebírá hodnoty z pohledu Production.cshtml. Vykresluje detaily zvoleného stroje.

Shared

- _Layout.cshtml – zde je šablona, jejíž obsah je aplikován na všechna View.
- _Login.cshtml – zajišťuje v šabloně _Layout.cshtml vzhled její přihlašovací části.

_ViewStart.cshtml - Tento soubor nám říká, kde je umístěn _Layout.cshtml.

Ukázka kódu View - složka Account, soubor Logout.cshtml

```
<ul class="nav navbar-nav navbar-right login">
@if (Request.IsAuthenticated)
{
      <li class="dropdown">
          <a href="#" class="dropdown-toggle" data-toggle="dropdown"
            role="button" aria-haspopup="true" aria
            expanded="false">Uživatel: @User.Identity.Name
            <span class="caret"></span></a>
            <ul class="dropdown-menu">
                <li>@Html.ActionLink("Odhlásit", "Logout",
                  "Account")</li>
            </ul>
        </li>
}
else
{
<li>@Html.ActionLink("Přihlášení","Login","Account",routeValues:  null,
```

```
htmlAttributes: new { id = "loginLink" })
</li>
}
</ul>
```

4.4.3 Models (modely)

Account.cs

Tento model představuje přihlašovací formulář, který je umístěn ve View LogIn.cshtml. Deklaruje hodnoty, které budou zadány, a nutnost jejich vyplnění.

Connect.cs

V tomto modelu jsou metody ContractsListSimple(), ContractsList(), OneContracProducts(), QuantityProducts(), ContractList() a ContractsSort().

- ContractsListSimple() – tato metoda vrací seznam zakázek.
- ContractsList() – výstupem je seřazený, vyfiltrovaný a stránkovaný seznam kontraktů.
- OneContracProducts() – dle čísla Id výrobku vrátí data o konkrétním výrobku.
- QuantityProducts() - dle čísla Id výrobku vrátí množství konkrétního typu výrobku.
- ContractList() – vrátí zakázku, ve které se nachází produkt s daným číslem Id.
- ContractsSort() – tato metoda slouží k řazení zakázek. Vrací list kontraktů seřazený dle požadovaného kritéria (předané hodnoty).

MyIdentity.cs

V tomto modelu jsou metody MyIdentity() a NamePassValidation().

- MyIdentity() – zde jsou vlastnostem přiřazeny hodnoty předané v parametru metody. Tyto vlastnosti jsou definovány níže, mimo tuto metodu. Slouží k uložení přihlášeného uživatele do paměti.
- NamePassValidation() – tato metoda ověřuje, zda předané hodnoty (uživatelské jméno a heslo) souhlasí se záznamem v databázi.

MyPrincipal.cs

Tento model je nutný pro správnou funkci přihlašovací procedury.

Production.cs

V tomto modelu jsou metody ProductionMachines(), ProductionList() a OneMachineProducts().

- ProductionMachines() – tato metoda vrací seznam strojů.
- ProductionList() – dle čísla Id stroje vrátí data o konkrétním stroji.
- OneMachineProducts() – dle čísla Id stroje vrátí seznam výrobků, které se na něm vyrábí.

TransportModel.cs

V tomto modelu jsou metody TransportLists(), TransportList(), OneTransportProducts(), QuantityProductsT() a TransportsSort().

- TransportLists() – výstupem je seznam převozů.
- TransportList() – tato metoda dle čísla Id převozu vrátí data o konkrétním převozu.
- OneTransportProducts() – dle čísla Id převozu vrátí seznam výrobků, které se mají převézt.
- QuantityProductsT() – dle čísla Id výrobku vrátí množství konkrétního typu výrobku pro daný převoz.
- TransportsSort() - tato metoda slouží k řazení převozů. Výstupem je list převozů seřazený dle požadovaného kritéria (předané hodnoty).

Ukázka kódu modelu - MyIdentity.cs, metoda MyIdentity() i s definicí vlastností

```
public class MyIdentity:IIdentity
{
    public MyIdentity(string name, string authenticationType, bool
                    isAuthenticated, Guid? userId, string password)
    {

        Name = name;
        Password = password;
        AuthenticationType = authenticationType;
        IsAuthenticated = isAuthenticated;
    }

    #region IIdentity
    [Display(Name = "Jméno")]
    public string Name { get; set; }

    public string AuthenticationType { get; private set; }
    public bool IsAuthenticated { get; private set; }

    [DataType(DataType.Password)]
    [Display(Name = "Heslo")]
    public string Password { get; set; }
```

ZÁVĚR

V práci byly nejprve shrnuty teoretické poznatky, které vedly k vytvoření webové aplikace. První kapitola osvětluje problematiku monitoringu a logistiky ve výrobní firmě. Ve druhé kapitole jsou sepsány hlavní vlastnosti a funkce, které by měla budoucí aplikace mít. Třetí kapitola prozkoumává frameworky, dle kterých může být tvořena budoucí aplikace.

Na základě poznatků z prvních tří kapitol vznikla webová aplikace pro monitorování procesu výroby. Popis a rozbor této aplikace se nachází v poslední kapitole.

Vytvořená aplikace předpokládá externí databázi, ze které získává data a zobrazuje je uživateli. Pro svůj chod potřebuje databázi, server, na kterém poběží a zobrazovací zařízení s připojením k síti. Nevyžaduje žádné speciální technické vybavení.

Webová aplikace vytvořená pro účely tohoto projektu slouží pouze pro monitoring, avšak má mnohem větší potenciál. Funkčnost se zde omezuje pouze na zobrazení dat, ale bylo by možné doplnit i funkce pro jejich vytváření a úpravu. Lze tedy doplnit tuto aplikaci o řídící část („managing“), která by umožňovala například vkládání zakázek, správu skladu, správu uživatelských účtů nebo potvrzování převozů pro logistiku. Tato část je již nad rámec zadání projektu a proto ji výsledná aplikace neobsahuje.

Aplikace pro monitorování vytvořená pro webové prostředí kontrastuje s monitorovacími systémy. Jde o systémy, které potřebují ke svému fungování speciální hardware i software. V dnešní době jsou na trhu nabízeny obě možnosti řešení a záleží na výrobních firmách, které je pro ně výhodnější. Dle mého názoru je webová aplikace pro monitorování výhodnou alternativou monitorovacích systémů.

LITERATURA

[1] GREGORY, J. Fluent NHibernate [online]. 2008 – 2012 [cit. 4. 12. 2015]. Dostupné z URL: <http://www.fluentnhibernate.org/>.

[2] JENNINGS, R. Profesional ADO.NET 3.5 with LINQ and the Entity Framework, Wiley publishing, Inc., 2009, 672 s., ISBN 978-0-470-18261-1.

[3] JOHNESS, J. Entity Framework tutorial [online]. 2015, poslední aktualizace 2015 [cit. 4. 12. 2015]. Dostupné z URL: <http://www.entityframeworktutorial.net/>.

[4] Microsoft Edice systému SQL Express [online]. 2015 [cit. 4. 12. 2015]. Dostupné z URL: <http://www.microsoft.com/cs-cz/server-cloud/products/sql-server-editions/sql-server-express.aspx>.

[5] Microsoft Visual Studio Express [online]. 2015 [cit. 4. 12. 2015]. Dostupné z URL: <https://www.visualstudio.com/cs-cz/products/visual-studio-express-vs.aspx>.

[6] MISTRY, R., MISNER, S. Introducing Microsoft SQL Server 2014 Technical Overview, Microsoft Press, A Division of Microsoft Corporation, 2014, 144 s., ISBN 978-0-7356-8475-1

[7] PERKINS, B. Working with NHibernate 3.0, Wiley Publishing, Inc., 2011, 230 s., ISBN 978-1-118-10460-6

[8] PIALORSI, P. Programing Miscrosoft LINQ in Microsoft .NET Framework 4, O'Reilly Media, 2010, 708 s., ISBN 978-0-735-64057-3.

[9] SCHENKER, G. N., CURE, A. NHibernate 3 Beginner's Guide, Packt Publishing , 2011, 368 s., ISBN 978-1-849516-02-0.

[10] SMITH, J. Advanced MVVM, 2013, 50 s., ISBN 0985784547.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

MVC	Model View Controller (návrhový vzor)
ORM	Object Relational Mapping (objektově relační mapování)
LINQ	Language Integrated Query (dotazovací jazyk)
SQL	Structured Query Language (strukturovaný dotazovací jazyk)
XML	Extensible Markup Language (rozšiřitelný značkovací jazyk)
DDL	Data definition language (jazyk definující datovou strukturu)
EDM	Entity Data Model (model popisující entity)
HTML	Hypertext Markup Language (značkovací jazyk)
WPF	Windows Presentation Foundation (framwork pro formulářové aplikace)
MVVM	Model View View Model (návrhový vzor)
UML	Unified Modeling Language

SEZNAM PŘÍLOH

A PŘÍLOHA - DVD

A.1 Obsah přiloženého DVD

Ukázky projektů

- NHibernate (NHibernateProject)
- Entity Framework (WebAppEntFram)
- LINQ (WebApplicationTestLinq)
- návrhový vzor MVC (WebApplicationTestMVC)

Webová aplikace pro monitorování procesu výroby

- WebApp2 (spouštěcí soubor pro Visual Studio)
- WebApp2 (složka souborů se zdrojovými kódy a soubory nutnými pro fungování aplikace)
- Packages (složka s použitými balíčky např.: pro správné fungování syntaxe Razor)

Databáze

- CompanyDB.bak
- CompanyDB.sql
- InsertDataToCompanyDB.sql