

Moderní technologie řešení klient-server

Solving a Client-Server using Modern Technology

Bc. David Filípek

Diplomová práce
2012



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2011/2012

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. David FILÍPEK**
Osobní číslo: **A10372**
Studijní program: **N 3902 Inženýrská informatika**
Studijní obor: **Informační technologie**

Téma práce: **Moderní technologie řešení klient-server**

Zásady pro vypracování:

1. Seznamte se s technologiemi C-Sharp pro ASP.NET MVC, webovými službami, výměnou dat mezi databázemi.
2. Provedte analýzu požadavků a uživatelských cílů.
3. Vypracujte návrh aplikace pro evidenci dárců krve.
4. Naprogramujte prototyp dané aplikace.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. HUDDLESTON, James a Vidya Vrat AGARWAL. Databáze v C-Sharp 2008. Praha: Computer Press, 2009. ISBN 978-80-251-2309-6.
2. MACDONALD, Matthew, Adam FREEMAN a Mario SZPUSZTA. ASP.NET 4 a C-Sharp 2010: tvorba dynamických stránek profesionálně, kniha 1. Brno: Zoner Press, 2011. ISBN 978-80-7413-131-8.
3. MACDONALD, Matthew, Adam FREEMAN a Mario SZPUSZTA. ASP.NET 4 a C-Sharp 2010: tvorba dynamických stránek profesionálně, kniha 2. Brno: Zoner Press, 2011. ISBN 978-80-7413-145-5.
4. Tutorials. ASP.net [online]. 2011 [cit. 2012-01-09]. Dostupné z: <http://www.asp.net/mvc/tutorials>
5. A Guide to Learning ASP.NET MVC Release Candidate 1. Stephenwalther.com [online]. 2010 [cit. 2012-01-09]. Dostupné z: <http://stephenwalther.com/blog/archive/2009/02/07/chapter-2-building-a-simple-asp.net-mvc-application.aspx>
6. Introducing ASP.NET MVC 3. ASP.net [online]. 2010 [cit. 2012-01-09]. Dostupné z: <http://weblogs.asp.net/scottgu/archive/2010/07/27/introducing-asp-net-mvc-3-preview-1.aspx>

Vedoucí diplomové práce:

Ing. Radek Šilhavý, Ph.D.

Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce:

24. února 2012

Termín odevzdání diplomové práce:

21. května 2012

Ve Zlíně dne 24. února 2012

prof. Ing. Vladimír Vašek, CSc.
děkan



doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

ABSTRAKT

V této práci jsou popsány možnosti technologie ASP.NET MVC, které lze využít pro programování dynamických webových stránek a pro jednoduché propojení s databázemi. Dále jsou rozebrány možnosti jednotlivých komponent této technologie, včetně možností grafických úprav, zabezpečení a testování. Je zde vysvětlen princip webových služeb a způsoby jejich využití. Na vytvořené aplikaci jsou vysvětleny postupy, jak lze ve vývojovém prostředí Visual Studia vytvářet databáze, způsob vytvoření webové služby jako WCF aplikace a způsoby jak využít tyto služby na straně klienta. Jsou ukázány názorné příklady tvorby controllerů a z nich generovaných pohledů i možnosti, jak lze kontrolovat uživatelem vkládaná data, zda odpovídají požadovanému formátu, včetně způsobů jak ho na špatně zadaná data upozornit.

Klíčová slova: databáze, architektura, webové služby, aplikace, komunikace

ABSTRACT

This thesis deals with facilities of technology ASP.NET MVC which can be used for programming of dynamic websites and a simple link with databases. I analyse single components of this technology including options of graphic design, security and testing. Besides the explanation of website services and their applications, this dissertation describes a specific application along with its procedures of creating databases in the developmental setting Visual Studia, the method of setting up website service as WCF application and possibilities of how the client could benefit from these services. The subsequent part introduces some examples of forming the controllers that should offer different aspects and options for checking whether the input data correspond to the format required including options for reminding the user of the data that have been wrongly entered.

Keywords: database, architecture, web services, application, communication

Zde bych chtěl uvést své poděkování vedoucímu mé práce Ing. Radku Šilhavému, Ph.D. za poskytnutí odborných rad a připomínek při tvorbě této práce a za pomoc při tvorbě návrhu i v průběhu vytváření projektu.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná od IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD	9
I TEORETICKÁ ČÁST	10
1 ARCHITEKTURA KLIENT- SERVER	11
1.1 DVOUVRSTVÉ A TŘÍVRSTVÉ ARCHITEKTURY	11
2 TECHNOLOGIE ASP.NET MVC	13
2.1 OBECNÝ PŘEHLED	13
2.2 ZALOŽENÍ NOVÉHO PROJEKTU	14
2.3 PRŮBĚH BĚHU APLIKACE	15
3 KOMPONENTY MVC	17
3.1 CONTROLLER	17
3.2 MODEL.....	19
3.3 VIEW	20
3.4 DALŠÍ PRVKY MVC.....	22
3.4.1 CSS.....	23
3.4.2 JavaScript	24
3.4.3 Bezpečnost	24
3.4.4 Testování	25
4 WEBOVÉ SLUŽBY	27
4.1 SOAP.....	27
4.2 WSDL.....	28
4.3 UDDI.....	28
II PRAKTICKÁ ČÁST	29
5 POŽADAVKY NA APLIKACI	30
5.1 POŽADAVKY DÁRCE	30
5.2 POŽADAVKY OBSLUŽNÉHO PERSONÁLU	31
5.3 NÁVRHOVÉ DIAGRAMY	31
6 TVORBA PROJEKTU	35
6.1 VÝVOJOVÉ PROSTŘEDÍ	35
6.2 ZALOŽENÍ PROJEKTU	36
6.3 VYTVOŘENÍ DATABÁZE	38
6.3.1 Databáze dárců krve	38
6.3.2 Databáze online uživatelů	40
6.3.3 Data v databázích	41
6.4 VYTVÁŘENÍ SOUBORŮ KOMPONENTY MODEL	42
6.4.1 Model pro databázi ASPNETDB.MDF	42
6.4.2 Model pro databázi Darcovství.mdf.....	44
6.5 VYTVOŘENÍ WEBOVÉ SLUŽBY	48
6.5.1 Web.config	48
6.5.2 Nastavení služby u klienta	49

6.6	CONTROLLERS A VIEWS	50
6.7	ACCOUNT	50
6.8	DÁRCE.....	51
6.8.1	Details().....	52
6.8.2	Změna údajů.....	54
6.8.3	Objednávka	55
6.8.4	Zrušení objednávky	58
6.8.5	Zobrazení odběrů	59
7	ZÁVĚREČNÉ ÚPRAVY	61
7.1	OPAČNÉ SEŘAZENÍ DAT	62
8	ZHODNOCENÍ ŘEŠENÍ	64
8.1	WEBOVÉ SLUŽBY	64
8.2	ASP.NET MVC.....	65
	ZÁVĚR V ANGLIČTINĚ.....	67
	SEZNAM POUŽITÉ LITERATURY.....	68
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	70
	SEZNAM OBRÁZKŮ	71
	SEZNAM TABULEK.....	72

ÚVOD

V této práci je cílem popsat možnosti, které nabízí technologie ASP.NET MVC, její práci s webovými službami a možnostmi výměny dat s databázemi. Jde o technologii, ve které je možné použít více programovacích jazyků, například C-Sharp, C++ a jiné.

V teoretické části je vysvětlen princip, na jakém pracují technologie klient-server a jejich rozdělení podle druhu vykonávané činnosti. Po základním popisu technologie ASP.NET MVC, jsou zde popsány jednotlivé komponenty, které tuto technologii tvoří. Tyto komponenty logicky rozdělují vytvářenou aplikaci na jednotlivé části, které se tak mohou vytvářet samostatně – nezávisle na vývoji ostatních částí. V dalších kapitolách jsou popsány možnosti využití JavaScriptu i kaskádových stylů a druhy používaných webových služeb. Důležitou součástí aplikací je i otázka jejich zabezpečení a značnou výhodu přináší i možnost vytvářet jednoduché testy, které umožňují ověření správného běhu aplikace.

K návrhu aplikace pro evidenci dárců krve bylo použito vývojové prostředí Enterprise Architect, které umožňuje přehledné grafické zpracování daného úkolu, včetně všech potřebných popisů a scénářů. Daná aplikace má sloužit dárcům krve k možnosti vytvoření online objednávky na další odběr. Nutností je být již zaregistrován na transfuzním oddělení, bez toho se nelze zaregistrovat do databáze online uživatelů. Po registraci se uživateli zobrazí jeho data a nabídka na vytvoření objednávky. Cílem aplikace je při vytváření objednávky zkontrolovat, zda je zadaný datum vhodný podle všech požadovaných kritérií. V každém kroku musí být uživatel informován o prováděné činnosti, včetně možnosti nápravy, když dojde k nějakému upozornění.

Samotná aplikace byla vytvořena ve vývojovém prostředí Visual Studio 2010. Po jednotlivých kapitolách jsou popsány kroky, jak probíhala tvorba prototypu aplikace. Jsou popsány možnosti využití prostředí Visual Studia k vytvoření databáze bez nutnosti instalace dalšího softwaru. To je výhoda i pro lepší přehled při programování ostatních částí projektu, včetně webové služby (WCF aplikace), kde jsou všechny potřebné informace dostupné v jednom vývojovém prostředí. Podmínkou je mít Visual Studio spuštěné s administrátorským oprávněním, bez toho není možné navázat komunikaci s databází SQL Server.

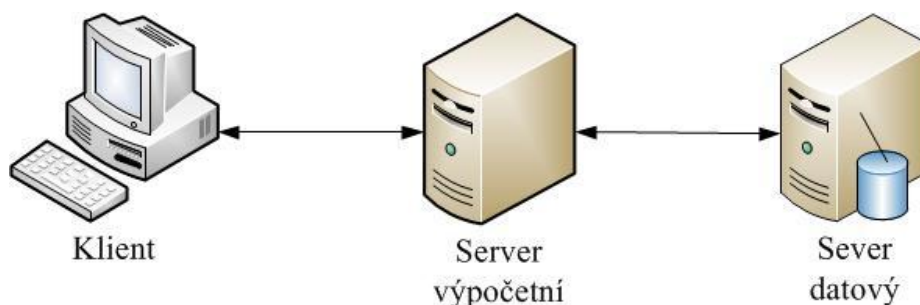
I. TEORETICKÁ ČÁST

1 ARCHITEKTURA KLIENT- SERVER

Jedná se o síťovou architekturu, kde je běh aplikace rozdělen na více částí, a to na klientskou část a serverovou část a vzájemně spolu komunikují. Existuje několik způsobů, které jsou popsány dále, jak lze rozdělit výpočetní činnost mezi klienta a server. To je důležité hlavně z hlediska možného přílišného zatížení serveru v případě, že bude provádět veškerou výpočetní činnost a klientovi se pouze zobrazí příslušné výsledky. Proto se tato činnost rozděluje podle druhu využití serveru. Z hlediska využívání serveru jsou klienti rozděleni na tenkého a tlustého. Tenký klient obsahuje pouze prezentační část, veškeré zpracování vstupů a dat se provádí na straně serveru. Výhodou tenkých klientů je, že stačí hardwarově méně vybavený počítač na straně klienta než v případě tlustých klientů. Další výhodou je centralizovanost dat, jejich snadná správa a zálohování. U tlustých klientů se část výpočetní činnosti přesouvá i na klienta. Na serveru mohou například být uložena pouze data a jejich zpracování a zobrazení je pak již na samotném klientovi. Jedná se o složitější řešení jak na návrh, tak i na vybavení klienta, ale nedochází k takové zátěži na server a je proto možné jej využít pro více klientů.

1.1 Dvouvrstvé a třívrstvé architektury

Jak v případě tenkého, tak i tlustého klienta se jedná o dvouvrstvou architekturu, kde je vše rozděleno pouze mezi dva počítače. Modernějším řešením je architektura třívrstvá. Ta kompenzuje nedostatky obou dvou předchozích způsobů zpracování. Jde o systém, kde je vše rozděleno mezi více výpočetních strojů a kde dochází k efektivnějšímu logickému rozdělení činností. Jde o samostatné uložení dat, výpočet pak probíhá na jiném místě a zobrazení je na straně klienta.



Obr. 1. Třívrstvá architektura klient – server

Zdroj: Vlastní zpracování podle [2]

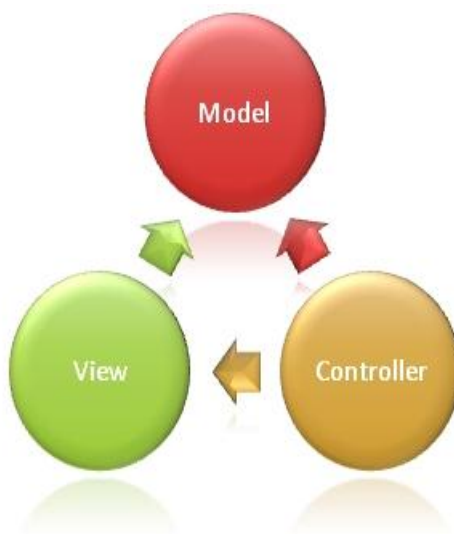
Jak je patrné z obrázku, tak veškerá činnost probíhá přes výpočetní server. Ten si načte potřebná data z datového serveru, ty poté zpracuje a výsledek odešle zpět ke klientovi. Principu činnosti třívrstvé architektury využívá i technologie MVC, kde jednotlivé komponenty lze přiřadit právě jedné vrstvě. Model tak zastupuje datový server, Controller server výpočetní a View slouží jako klient. Samozřejmě, že všechny tyto části lze provozovat i na jednom počítači, ale to se používá spíše jako způsob při vývoji aplikací. Při reálném provozu je pak vhodné tyto části oddělit na jednotlivé servery. Velkou výhodou je tak dobrá přehlednost při programování aplikací, protože se dá na řešený problém dívat vždy z požadovaného hlediska. Další výhodou je i to, že každou část může vyvíjet jiný tým bez ohledu na postup vývoje jiného týmu. V takovémto případě, ale je nutné mít dobře zpracovaný návrh projektu, kde je jasně definováno, co se od dané aplikace očekává.

2 TECHNOLOGIE ASP.NET MVC

Jedná se o jednu z možných technologií, které lze využít při vytváření dynamických webových stránek. Umožňuje jednoduché získávání dat z databází, jejich následné zpracování a zpětné uložení do databáze. Databáze je možné i vytvářet při programování dané aplikace, a to přímo ve vývojovém prostředí Visual studia od Microsoftu. Nevýhodou je, že prostředí Visual studia nenabízí tolik možností při vytváření databází jako například vývojové prostředí SQL Server Studio Management, které je přímo určené pro tvorbu databází.

2.1 Obecný přehled

Architektura ASP.NET MVC se skládá ze tří hlavních komponent. **Model** je logická část, která slouží pro komunikaci s daty v databázi. **View** je část, kde se upravuje vzhled vytvářené webové stránky a **Controller**, kde se naprogramuje činnost, která se provede jako reakce na požadavek uživatele.



Obr. 2. Model MVC [5]

Z obrázku je patrné, že komponenta Controller přijímá uživatelský požadavek a tento po zpracování posílá na obě dvě komponenty k dalšímu zpracování. Například, když chce uživatel v nějaké databázi změnit svůj telefon, tak Controller tento požadavek zpracuje a pošle zpracovanou hodnotu do Modelu a výsledek se pomocí komponenty View zobrazí uživateli na obrazovce.

Výhodou tohoto postupu je, že při vytváření takovéto aplikace se může programátor, nebo tým programátorů, starat pouze o tu komponentu, kterou programuje, ale nemusí znát podrobnosti zbylých dvou komponent.

2.2 Založení nového projektu

U zakládání nového projektu je potřeba si nejdříve vybrat z nabízených šablon. Ty je možné si vybrat v prostředí Visual Studio 2010 mezi programovacími jazyky C#, Visual Basic, C++ nebo Visual F#. Po výběru příslušného jazyka je potřeba si zvolit z nabízených možností pro tvorbu aplikací. V tomto případě se jedná o webové aplikace. Pro využití architektury MVC lze vybrat šablonu *ASP.NET MVC 2 Web Application* nebo *ASP.NET MVC 3 Web Application*. Pro možnost použít šablonu MVC 3 je potřeba ještě doinstalovat *ASP.NET MVC 3 Tools Update* a *SQL Server Compact 4.0* [14].

Šablona MVC 3 má přidáný nový engine, nazvaný Razor. Dále je možné si vybrat použití sémantického značení HTML5. Tyto rozšíření jsou používány u komponenty View. Jako praktický příklad rozdílu mezi šablonami MVC 2 a MVC 3 lze zmínit situaci, kdy uživatel při vyplňování nějakého pole ve formuláři zadá neplatnou hodnotu. U šablony MVC 2 dojde poté při potvrzení zadávaných údajů k zobrazení upozornění na neplatné zadání. Po opravě těchto údajů musí uživatel opět potvrdit zadané údaje a teprve poté dojde k jejich ověření. A to vše probíhá na straně serveru. Šablona MVC 3 využívá toho, že když uživatel zadá nějaký neplatný údaj, tak po jeho potvrzení dojde stejně jako u šablony MVC 2 k upozornění na neplatný údaj. Ale při opravě tohoto údaje již dochází k okamžité kontrole nově zadávaných údajů, protože pravidla pro kontrolu jsou již uložena na straně klienta.

Při vytváření projektů je možné si vybrat, zda se daný projekt vytvoří jako internetová aplikace, intranetová aplikace anebo je možné vytvořit prázdný projekt. Ale i projekt, který se vytvoří jako prázdný, obsahuje již spousty předdefinovaných složek. Rozdíl mezi prázdným projektem a internetovou nebo intranetovou aplikací je v tom, že tyto aplikace již mají nedefinované základní položky *HomeController* a *HomeView*. U internetové aplikace je ještě předem vytvořená položka *AccountController* a *AccountView*, které umožňují jednoduchou správu vytváření uživatelských účtů pro přihlášení k danému serveru.

2.3 Průběh běhu aplikace

Po spuštění aplikace vytvořené v architektuře ASP.NET MVC prohlížeč nejdříve zpracuje požadavek objektu *UrlRoutingModule*, to je modul HTTP. Tento modul zpracuje požadavek a provede výběr prvního objektu, který odpovídá danému požadavku. Jestliže není nalezena žádná shoda, objekt *UrlRoutingModule* neudělá nic a vrátí požadavek zpátky.

V následující tabulce je zobrazen podrobný postup při vykonávání projektu vytvořeném v ASP.NET MVC.

Tab. 1. Zpracování požadavků [8]

Požadavky	Detaily
Obdržení prvního požadavku aplikace	V souboru <i>Global.asax</i> , objekty <i>Route</i> jsou přidány do objektu <i>RouteTable</i> .
Provedení směrování	Modul <i>UrlRoutingModule</i> použije první shodu objektu <i>Route</i> v kolekci <i>RouteTable</i> a vytvoří objekt <i>RouteData</i> , který je použit k vytvoření objektu <i>RequestContext</i> .
Vytvoření MVC handleru požadavku	Objekt <i>MvcRouteHandler</i> vytvoří instanci třídy <i>MvcHandler</i> a předá ji instanci třídy <i>RequestContext</i> .
Vytvoření Controlleru	Objekt <i>MvcHandler</i> použije instanci <i>RequestContext</i> k identifikaci objektu <i>IControllerFactory</i> k vytvoření instance <i>Controlleru</i> .
Provedení	Instance <i>MvcHandler</i> zavolá metodu <i>Execute</i> .
Vyvolání akce	Objekt <i>ControllerActionInvoker</i> je připojen ke <i>Controlleru</i> a rozhodne, která prováděcí metoda třídy <i>Controller</i> se zavolá.
Vykonání výsledku	Typická prováděcí metoda přijme vstup od uživatele, připraví odpovídající data a provede výstup navrácením požadovaného typu.

Jak je uvedeno v posledním řádku tabulky, tak při provádění požadavku od uživatele je jako výstup použita návratová hodnota. Ta může být například typu *View*. Tento typ nemusí mít žádné parametry, ale může být použit i jako parametrický, kde se mohou zobrazit vybraná data z databáze. Dalším možným typem je *RedirectToAction*, kde dochází k přesměrování na jinou akční metodu. Tou může být třeba návrat na hlavní stránku nebo na jiný potřebný požadavek. V prvním parametru této metodu se uvádí název akční metody, na kterou se předává řízení běhu programu, popřípadě může být přidán i

druhý parametr, kde se uvádí název controlleru, ve kterém se vybraná akční metoda nachází. To umožňuje používat v různých controllerech stejné názvy metod.

3 KOMPONENTY MVC

Hlavní komponenty architektury MVC jsou Controller, Model a View. Ty slouží k rozdělení vytvářené aplikace do jednotlivých bloků. Každá z těchto komponent se zaměřuje na jiný typ problému, který je potřeba ošetřit pro správný běh aplikace. Díky tomu se mohou, při dobrém návrhu aplikace, jednotlivým komponentám věnovat různé týmy nezávisle na sobě. Tento paralelní vývoj tím pádem umožňuje rychlejší vývoj dané aplikace.

3.1 Controller

Controllers, neboli česky řečeno ovladače, slouží k tomu, že zpracovávají požadavky, které má uživatel při prohlížení webových stránek a které zadává zmáčknutím příslušného tlačítka. Konkrétní ovladač je spojen s tímto tlačítkem a po vyslání požadavku provede jemu zadaný úkol a výsledek vrátí jako zprávu, která se uživateli zobrazí na monitoru.

Jak je vidět na Obrázku Modelu MVC, tak ovladače musí komunikovat jak s komponentou *Model*, tak i s komponentou *View* [3]. Vytvoření nového ovladače je velice jednoduché. Ve vývojovém prostředí Visual Studia se pouze pravým tlačítkem klepne na složku *Controllers* a vybere se z nabídky *Add -> Controller*. Pro lepší přehlednost při zpracovávání projektu je dobré si dobře rozvrhnout, kolik takových ovladačů bude potřeba. Při založení nového projektu, který je vytvořený jako internetová aplikace, jsou automaticky vytvořeny dva ovladače. Jedná se o *HomeController* a *AccountController*. Druhý zmiňovaný ovladač slouží pro přihlašování k účtu, respektive k jeho vytvoření pro nového uživatele. *HomeController* obsahuje jednu základní metodu. Ta se nazývá *Index()*. Je to základní metoda, ve které je určeno, co se má zobrazit jako úvodní stránka.

Pro lepší přehlednost se za název nového ovladače přidává slovo *Controller*. Například při vytvoření třídy nějaké databáze, bude název ovladače *DatabaseController*. Všechny vytvořené ovladače dědí od třídy *Controller*. Metody používané v ovladačích jsou akční metody. Například již pro dříve zmiňovanou metodu *Index()*, která je veřejná, vypadá zadání metody takto:

```
public ActionResult Index() { }
```

Samozřejmě, že takováto metoda musí obsahovat i návratovou hodnotu. Tato návratová hodnota určí, co se stane po zpracování této metody. Z třídy *Controller* jsou zděděny

metody, které se používají jako návratové hodnoty. Nejčastěji se jedná o metody *View* a *RedirectToAction*. Metoda *View* slouží pro zobrazení výsledku dané akční metody na monitor. Samozřejmě, aby se něco zobrazilo na monitor, tak se musí pro danou metodu vytvořit pohled, kde se určí, co se má zobrazit. Podrobnosti jsou vysvětleny v kapitole *View*. Druhá metoda, *RedirectToAction*, slouží pro přesměrování na jinou akční metodu. Název metody, na kterou se přesune řízení běhu programu je uveden v uvozovkách jako parametr metody *RedirectToAction*. V případě, že se jedná o akční metodu, která je vytvořena v jiné třídě *Controller*, tak se jako druhý parametr použije název této třídy. Například když je ve třídě *HomeController* vytvořena metoda *Details* a ve třídě *DatabaseController* je také metoda *Details*, tak pro správné přiřazení požadované metody je lepší napsat i druhý parametr, aby bylo jasně vidět z jaké třídy je metoda použita.

```
return RedirectToAction ("Details", "Home");
```

Další možností použití návratové hodnoty je přímo zadat například hodnotu typu integer nebo třeba aktuální datum. Při takovéto definici dojde k automatickému zavolání metody *ContentResult*. Prohlížeč v tomto případě obdrží aktuální datum jako prostý text, který potom zobrazí.

```
return DateTime.Now;
```

V následující tabulce jsou uvedeny metody ze základní třídy *Controller*, které lze použít jako návratovou hodnotu pro danou akční metodu.

Tab. 2. Návratové metody třídy *Controller* [5]

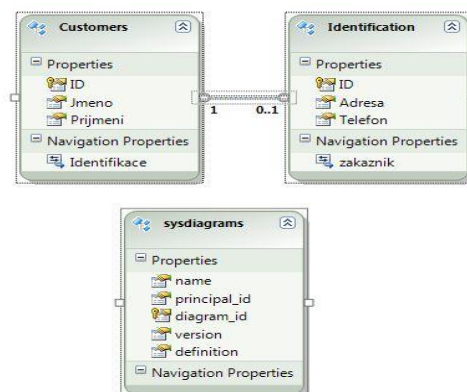
1	View	Předává řízení běhu programu na pohled, který je k dané metodě vytvořen.
2	Redirect	Přesměruje na novou URL adresu.
3	RedirectToAction	Přesměruje na jinou akční metodu.
4	RedirectToRoute	Vrací výsledek metody <i>RedirectToRouteResult</i> .
5	Json	Vrací objekt JavaScriptu použitého v AJAX aplikaci.
6	JavaScriptResult	Vrací požadovaný JavaScript.
7	Content	Vrací obsah jako text.
8	File	Podle zadaného parametru vrací zadaný stažitelný soubor.

3.2 Model

Složka *Models* slouží k vytvoření tříd, které obsahují metody a vlastnosti ve kterých jsou uložena data. Tyto data jsou pak následně zpracována ve třídách ovladačů. Data je možné uložit přímo ve třídě modelu anebo daleko lepší a efektivnější způsob je provázání dat, která jsou uložena v databázi. Nejjednodušší způsob jak zajistit, aby se data z databáze uložily do modelové třídy je implementovat je tam při založení modelu ve Visual studiu.

Vytvoření modelu, který je provázaný z daty požadované databáze se provede tak, že se klikne pravým tlačítkem myši na složku *Models* a vybere se položka *New Item*. Poté se v záložce vybere složka *Data* a z dané nabídky se vybere *ADO.NET Entity Data Model*. Dalším krokem se zvolí, aby se daná data generovala z databáze. Tady je potřeba si uvědomit, že při práci s databází je potřeba mít nastaveny administrátorská práva programu Visual studia. Jinak v nabídce dostupných databází nebude žádný výběr. V případě, že je vše v pořádku, tak lze z kontextové nabídky vybrat požadovanou databázi a dole v okně je vytvořen název entity, která slouží pro práci s danou databází. V posledním kroku při vytváření modelu je možnost vybrat si všechny tabulky z dané databáze anebo jen tabulky, které jsou potřeba pro daný projekt.

Po vytvoření modelu se objeví dva pohledy na daný model. První je grafický a obsahuje vybrané tabulky, jejich vlastnosti a vzájemné vazby. Tento model má příponu *edmx*. Druhý model obsahuje vložený název, plus má příponu *Designer.cs*. V tomto modelu jsou automaticky vygenerovány třídy a jejich vlastnosti, které odpovídají datům ve vybraných tabulkách. S takto vytvořenými daty je již možné pomocí vytvořené entity pracovat ve třídách ovladačů.



Obr. 3. Grafický model

V modelech je také možné u jednotlivých vlastností třídy určit omezení, která jsou požadována při vkládání dat uživatelem. Když například má uživatel na zobrazené stránce nějaké políčko, kam vložit požadovaný údaj, tak právě tyto omezení mohou kontrolovat, zda jsou zadané údaje v požadované formě. Může to být třeba nastavení atributu maximální možné délky zadávaného řetězce, rozsah zadávaných čísel od nejmenšího do největšího apod. Pro případ, že nějaký sloupec z databáze nemá být zobrazen uživateli v nabídce, stačí použít atribut [`ScaffoldColumn(false)`]. To se nejčastěji používá pro skrytí nějakého ID sloupce, který je v databázi nastaven na automatické vyplnění. Asi nejčastějším atributem je požadavek, že dané pole musí být vyplněno. V tomto atributu se také zadá název zprávy, která se uživateli zobrazí v případě, že dané pole nevyplní. Tyto atributy se vždy definují nad danou vlastností. Těchto atributů může být pro každou vlastnost tolik, kolik jich je potřeba.

```
[Required(ErrorMessage="Pole musí být vyplněno")]
```

V případě projektu založeného na ASP.NET MVC 2, dojde pak při zobrazení takovýchto hlášení k jejich opravě až poté, co uživatel znovu vyplní zadaný údaj a stiskne nějaké potvrzovací tlačítko. Velká výhoda architektury ASP.NET MVC 3 je v tom, že při prvním zadání dojde ke kontrole vkládaných údajů až po stisknutí potvrzovacího tlačítka, ale při opravě těchto údajů je již okamžitě kontrolována jejich platnost [7]. To je způsobeno tím, že kontrolní pravidla jsou uložena již na straně klientského počítače a ne na straně serveru.

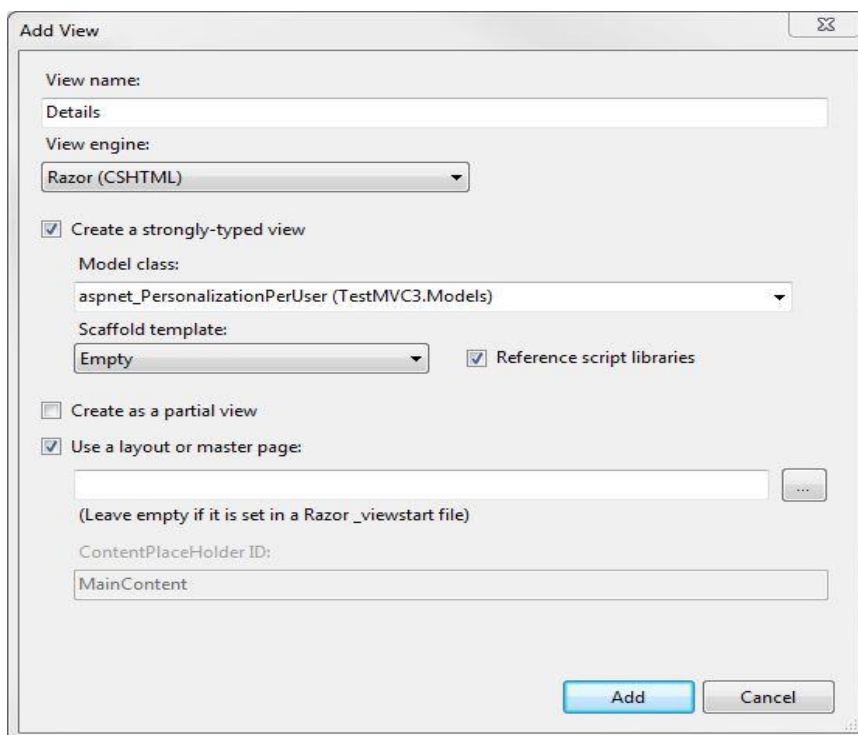
Automatické vytvoření modelu také zajistí automatickou konfiguraci nastavení spojení s databází. Takto vytvořené spojení se nachází v souboru *Web.config* v části *connectionStrings*. V modelu vytvořeném ručně se nesmí zapomenout na nastavení tohoto spojení s databází.

3.3 View

Jak již samotný název z anglického překladu napovídá, jedná se o pohled, který se naskytne uživateli při otevření dané stránky nebo po zvolení nějaké akce. Tento pohled se připojí vždy k nějaké akci, která je naprogramována v daném ovladači.

```
public ActionResult Index () {nějaký kód}
```

Aby se k dané akci připojil pohled, tak stačí kliknout pravým tlačítkem myši na danou metodu, v tomto případě *Index* a z nabídky vybrat *AddView*.



Obr. 4. Vytvoření pohledu

Na obrázku je zobrazeno okno, které se zobrazí pro nastavení pohledu View. V položce jméno je zobrazen název, který se shoduje s akční metodou, ke které daný pohled patří. V případě vytváření projektu ve starším typu architektury - ASP.NET MVC 2 se nezobrazí možnost výběru enginu. To je rozšíření, které je až součástí ASP.NET MVC 3. Další, a velice podstatnou položkou, je položka *Create a strongly-typed view*. V případě, že tato položka není zaškrtnutá, dojde k vytvoření velice jednoduchého náhledu a všechny potřebné data se musí dopsat ručně. To je vhodné pokud se pohled vytváří například jako informativní zpráva o nějaké úspěšně, případně neúspěšně provedené operaci. Ale v případě, že je potřeba v daném pohledu pracovat s daty z nějaké databáze, je vhodné tuto položku označit. Tím se zaktivuje Model tříd, kde se z dané nabídky vybere konkrétní tabulka z požadované databáze. Tím dojde k provázání daného vytvářeného pohledu s danou databází a z další položky – *Scaffold template* - je možnost vybrat jednu z předdefinovaných šablon pro vytvoření pohledu. Jedná se o šablony pro vytvoření, změnu, detaily, vymazání a seznam(List). Nebo lze vytvořit pohled jako prázdný. Každá z těchto šablon obsahuje již předem určený tvar, jak bude vytvořená stránka vypadat. Po dokončení výběru již stačí potvrdit stisknutím tlačítka *Add*.

Takto vytvořený pohled je v klasickém HTML jazyku, který lze rozšiřovat pomocí JavaScriptů a kaskádových stylů [6]. Při vytvoření projektu je již ve složce *Content*

obsažen soubor *Site.css*, kde jsou kaskádové styly definovány. Pro jejich změnu stačí již jen v tomto souboru vybrat příslušný tag a změnit požadované parametry. Například barvu pozadí celé stránky apod. Pro práci s JavaScripty je vytvořena složka *Scripts*, kde se nachází množství již vytvořených JavaScriptů a kam je také možné další potřebné JavaScripty ukládat.

Další výhodou při práci ve Visual studiu je to, že při vytvoření nového pohledu se ve složce *Views* vytvoří soubory, které odpovídají souborům ve složce *Controllers*. A v těchto jednotlivých souborech jsou již obsaženy jednotlivé metody pro konkrétní akční metody. Ovšem složka *Views* obsahuje o jednu další složku navíc. Tou je složka *Shared*. V případě projektu vytvořeného jako internetová aplikace se v této složce nachází tři soubory:

- *_Layout.cshtml*
- *_LogOnPartial.cshtml*
- *Error.cshtml*

U MVC 2 se soubor *_Layout.cshtml* jmenuje *Site.master* a soubor *_LogOnPartial.cshtml* má název *LogOnUserControl.ascx*. Je to ovšem rozdíl pouze v názvech souborů. Samotný jejich obsah se liší hlavně v nastavení zabezpečení při práci s uživatelským účtem. Hlavním principem těchto souborů je jejich univerzálnost použití. Například v souboru *_Layout.cshtml* je nedefinován základní vzhled a rozložení zobrazované stránky a je tam také umístěn název dané stránky. V případě, že pak uživatel klikne někde na nějaké tlačítko a zobrazí se mu tím příslušný jiný pohled, tak všechny tyto prvky zůstanou zobrazeny. Do tohoto souboru se proto může přidat například ikona, která se zobrazuje v adresním řádku prohlížeče, a právě proto, že se nachází v tomto souboru, tak se bude zobrazovat i po zobrazení jiné stránky z daného projektu.

3.4 Další prvky MVC

Architektura ASP.NET MVC neobsahuje pouze prvky popsané v předešlých kapitolách. Součástí této architektury je i řada dalších prvků, pomocí kterých lze daný projekt vytvářet. Jsou to prvky, které pomáhají zjednodušit vývoj aplikace a které mohou doplňovat některé věci, které jsou součástí již vytvořené komponenty. Jedná se hlavně o JavaScript a kaskádové styly. Dále lze aplikaci doplnit o potřebné obrázky, které se mají zobrazovat po načtení požadované stránky. Pro práci s obrázky je možné si vytvořit novou

složku, kde budou obrázky uloženy nebo se častěji tato složka vytvoří jako podsložka složky *Content*. V této složce jsou již uloženy nějaké obrázky ikon a také se v této složce nachází předdefinovaný kaskádový styl pojmenovaný *Site.css*.

Součástí vytvořeného projektu jsou i další složky. Pro práci s databázemi se používá složka *App_Data*. V této složce jsou umístěny jednotlivé databáze, které jsou používány v projektu. Další složkou, která je součástí projektu je *Scripts*. Do té jsou ukládány požadované JavaScripty. Důležitou složkou je *Global.asax*. V této složce se nachází soubor *Global.asax.cs* ve kterém se nachází hlavní třída celé aplikace. Součástí této třídy je metoda, které určuje pravidla pro routování adres. Dále je zde metoda určující start aplikace. A jako poslední důležitá součást projektu je vytvořen soubor nazvaný *Web.config*. V tomto souboru jsou nastaveny parametry pro spojení s databázemi. V případě, že je databáze k projektu připojena při vytváření komponenty *Model*, tak je toto spojení do tohoto souboru doplněno automaticky.

Všechny tyto složky a soubory se nachází ve Visual studiu v okně nazvaném Solution Explorer. Pro práci s databázemi je dobré si aktivovat i okno Server Explorer. V tomto okně se databáze zobrazí stejným způsobem jako v klasických prostředích pro vývoj databází. Dají se tak vybrat jednotlivé tabulky, diagramy a další položky a zobrazit jejich data nebo údaje o jejich nastavení. Také je možné přímo ve Visual studiu vytvořit zcela novou databázi, včetně všech tabulek apod. Jen je potřeba si uvědomit, že je potřeba mít Visual studio spuštěné s administrátorskými právy. S běžným uživatelským oprávněním nebude přístup k danému serveru umožněn.

3.4.1 CSS

CSS – Cascading style sheets jsou v překladu kaskádové styly. Jedná se o metody, které se starají o grafickou podobu vytvářených webových stránek. Samozřejmě, že se dá grafická podoba vytvořit i přímo pomocí jazyku HTML, ale v případě, že je více stránek, které mají mít stejný vzhled, barvu písmen a například velikost nadpisů, tak je lepší použít kaskádové styly. V případě, že je tvůrce nepoužije, tak při tvorbě každé stránky musí vše řešit znovu, a pokud v budoucnu někdy mění nějaký styl, tak opět musí vše procházet znovu. Pokud ale využije možnosti, jaké nabízí kaskádové styly, tak si v nich nadefinuje potřebné parametry a ty se pak projeví na každé vytvářené stránce [4]. V případě změny potom stačí změnit nějaký atribut v souboru css a změna se projeví všude. Například příkaz `table{color: blue;}` způsobí, že barva písma při použití tohoto atributu

bude modrá. Pokud se ovšem uvnitř tohoto atributu použije další atribut, který bude mít nastavenou barvu jinou, tak bude ve výsledku přiřazena ta barva, která je definována pro vnitřní příkaz.

Samozřejmě, aby bylo možné daný styl využít, musí se definovat pomocí příkazu `link` na dané stránce. Při vytváření aplikace pomocí architektury MVC je to vyřešeno elegantním způsobem tak, že je daný styl definovaný v souboru `_Layout.csthml`. Jak již bylo dříve napsáno, tak tento soubor je nastaven jako hlavní, a proto stačí odkaz na styl definovat pouze v tomto souboru.

3.4.2 JavaScript

Při vytváření aplikací v ASP.NET MVC je také možné využít JavaScript. Na rozdíl od kaskádových stylů neslouží k práci s grafikou, ale hlavní výhodou je programování funkcí. Možnost psát JavaScriptové funkce lze přímo v HTML souborech, mezi tagy `<script></script>` anebo vytvořit samostatný soubor, který pak lze pouze do potřebného HTML souboru vložit. V MVC je na JavaScripty vytvořena složka `Scripts`, kde jsou již automaticky nějaké JavaScriptové soubory vytvořeny.

Jedná se o interpretovaný jazyk, který umožňuje provádění výpočtů, obsluhu událostí, ale neumožňuje zápis do souborů a čtení dat ze souborů na straně klienta. Pomocí JavaScriptu lze vytvářet například kalendáře, hodiny nebo třeba i jednoduché hry. JavaScriptové soubory není nutné vždy vytvářet. Na internetu existuje veliké množství volně dostupných skriptů, které lze použít pro vlastní aplikaci.

Při vytváření aplikací, kde se používá JavaScriptů, je potřeba si uvědomit, že uživatel webových stránek nemusí mít JavaScript ve svém prohlížeči povolený a zobrazované stránky tak nebudou nabízet možnosti, které byly cílem vytvářené aplikace.

3.4.3 Bezpečnost

Důležitým požadavkem v dnešní době je také otázka zabezpečení webových stránek proti neoprávněnému přístupu k vybraným datům. Zatímco některé informace mohou být přístupné komukoliv, tak data z databází a podobné informace již slouží pro konkrétní přihlášené uživatele.

V technologii ASP.NET MVC je možné využít nabízené možnosti vytvoření zabezpečeného přihlašování uživatelů. Existují tam při vytváření projektu jako internetové aplikace složky `Account`, které obsahují již naprogramovaný a zabezpečený způsob

přihlašování. Automaticky je také vytvořena databáze, která se nazývá *ASPNETDB.MDF*, kde se ukládají informace o jednotlivých uživateli a kde je heslo uloženo v zašifrované podobě. Součástí tohoto modelu je i vytvoření grafické podoby, kde si může uživatel zvolit, aby si jeho počítač pamatoval přihlašovací údaje, stejně tak je tam i možnost změny hesla.

Při vytváření aplikace v MVC 2 lze vidět ve vytvořeném kódu v souboru *AccountController.cs*, že není požadováno žádné ověření při přesměrování URL parametru. Toho může zneužít hacker metodou útoku phishing. Uživatel klikne na falešný odkaz, který je ale směřován na správnou stránku, kde je ale nastavena jako návratová hodnota URL stránka podvodná. Uživatel se tak na správné stránce pokusí o přihlášení, ale v tomto okamžiku dojde k přesměrování na podvodné stránky, kde je nějaká nepatrná změna v názvu stránky, a tam dojde k zobrazení chybové zprávy o neúspěšném přihlášení s výzvou o opakování přihlášení. Uživatel, tentokrát již na podvodné stránce, zadá znovu své údaje, v tuto chvíli je získává útočník, a poté dojde k přesměrování na správné stránky, kde již ale přihlášení proběhlo hned na začátku a tudíž se uživateli zobrazí již stránka kde je přihlášený. Uživatel tak nemá tušení, že poskytl své údaje někomu dalšímu.

Tento problém řeší technologie MVC 3, kde je použita nová metoda *IsLocalUrl*. Tato metoda ověří, zda se jedná o URL adresu daného serveru a v případě pokusu útočníka o přesměrování na jiný server dojde k nesplnění podmínky a dojde k přesměrování na požadovanou stránku, kterou zadal programátor dané aplikace. Tímto způsobem se tedy uživatel vůbec nedostane na podvodné stránky a útočník tak nemůže získat jeho data.

3.4.4 Testování

Při zakládání nového projektu existuje také možnost vytvořit testovací soubor, tzv. unit test. V podstatě se jedná o automatizované testy, které slouží pro testování vytvářeného projektu [9]. Jde o to, aby se při programování nemuseli zkoušet všechny možnosti tím způsobem, že se daná aplikace spustí, a poté se zkouší klikat na všechny možnosti. To samozřejmě zabere spoustu času a při rozsáhlejších projektech ani není možné všechny tyto možnosti pokrýt.

V ASP.NET MVC je možnost přidávat k projektům unit testy, ale je potřeba mít Visual studio v edici professional nebo vyšší. V jiném případě se dá využít aplikací od jiných výrobců. Testy se vytváří pro jednotlivé třídy a při jejich tvorbě se vlastně simulují možnosti zadání uživatelů. Například lze natestovat situaci, kdy má uživatel zadat svůj

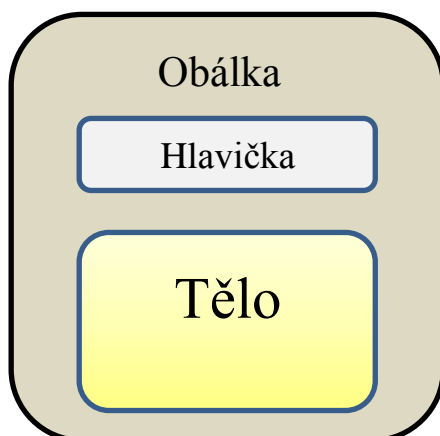
email a v testu se provede test jak na dobré zadání, tak i na špatně vyplněné údaje. V případě vytváření více testů je při jejich spuštění možnost vybrat, zda testovat všechny vytvořené testy nebo jen vybrat nějaký konkrétní pro danou situaci. To je výhodné hlavně v situacích, kdy je vytvořené velké množství testů a jejich zpracování pak zabere dlouhý čas. Proto je lepší způsob vytvářet testy pro každou funkci programu zvlášť. Výsledek testu je tak daleko dříve a zároveň se i lépe odhaluje případná chyba nebo chyby.

4 WEBOVÉ SLUŽBY

Díky webovým službám je možné programovat dynamické webové stránky. Umožňují komunikaci s databázemi, tzn. zadávat dotazy do databáze a zpět získat požadovanou odpověď. Vzhledem k tomu, že komunikace probíhá v různých prostředích, využívá se nezávislých standardů, hlavně protokolu HTTP a jazyku XML (3. Webové služby, 2007). Při vytváření aplikace v ASP.NET MVC jsou již tyto služby zabudovány přímo v daném programovacím jazyku a pouze stačí si vybranou službu v požadovaném bodu aplikace zavolat. Často se používá jako atribut u metod *ActionResult*, kde se tím vlastně omezuje činnost dané metody jen na vybraný požadavek konkrétní služby. Může se jednat o atributy *HttpPost*, *HttpGet* a další. Při rozhodování, který atribut je vhodnější, tak podle oficiálního MSDN fóra Microsoftu [12] je pro případ, že se vytváří operace, která nemá vedlejší efekty a vrací data, která nejsou nikdy nebo málo měněná, lepší využít atribut *HttpGet*. Je to z důvodu, když je stejná operace volána mnohokrát, tak existuje pouze jeden požadavek na službu.

4.1 SOAP

Simple Object Access Protocol, jedná se o protokol, pomocí kterého se posílají zprávy ve formátu XML. Zprávy se posílají mezi jednotlivými aplikacemi navzájem a jedná se tak o model peer-to-peer. Každá zpráva je vlastně jednoduchý XML dokument. Ten obsahuje kořenový element *Envelope* – obálka. Do této obálky jsou vloženy další dva elementy. Jedná se o hlavičku – *Header* a tělo – *Body*. Element hlavička se nemusí použít, do tohoto elementu se ukládají pouze doplňující informace, například jméno uživatele apod.



Obr. 5. Elementy SOAP [10]

4.2 WSDL

WSDL je popisující jazyk ve formátu XML, který popisuje operace nebo zprávy. Tyto operace a zprávy jsou popisovány abstraktně a poté jsou připojeny ke konkrétnímu síťovému protokolu nebo formátu zpráv (Understanding WSDL, 2003). Tímto určí koncový bod. Jedná se tak o strukturovaný popis komunikace.

Dokument WSDL definuje služby jako kolekci síťových koncových bodů nebo portů. Pro abstraktní popis vyměňovaných dat slouží zprávy, zatímco typy portů jsou abstraktní kolekce operací [13]. Port je definován přiřazením síťové adresy a kolekce portů definuje službu. WSDL dokument používá následující prvky k definici síťových služeb:

- typy – kontejner pro definici typu dat
- zpráva – abstraktní, definuje typ dat použitých ke komunikaci
- operace – abstraktní popis akce podporované službou
- typ portu – abstraktní množina operací podporovaná jedním nebo více koncovými body
- omezení – konkrétní protokol a formát dat pro daný typ portu
- port – koncový bod definovaný jako kombinace omezení a síťové adresy
- služba – kolekce souvisejících koncových bodů

4.3 UDDI

Tato služba je ve své podstatě vlastně rejstřík obsahující informace o konkrétních firmách, jako je například IBM, Microsoft apod., a obsahující seznam služeb, které tyto firmy nabízejí [11]. Tyto informace jsou rozděleny do kategorií podle jednotlivých firem i podle jednotlivých služeb. Samotné vyhledávání pak funguje na základě těchto kategorií.

Provoz této služby probíhá na uzlech. Tyto uzly musí splňovat specifikaci, a proto není podstatné, který uzel se použije pro vyhledání konkrétní služby. Uzel ale neobsahuje službu jako takovou, pouze adresu webové služby.

Práce s UDDI funguje na principu prohledání rejstříku vývojářem projektu, kde si vybere potřebné služby. Tam si pro ně získá popis v libovolném formátu, ale nejčastěji se jedná o WSDL.

II. PRAKTICKÁ ČÁST

5 POŽADAVKY NA APLIKACI

Cílem vytvářené aplikace je vytvořit systém, pomocí kterého se budou dárci krve moci přes webový prohlížeč objednat na odběr krve. Aby se dárci mohli k odběru objednat, musí již být zaregistrovaní na příslušném hematologicko-transfuzním oddělení. Tam je zaregistrován při svém prvním odběru krve, na který se neobjednává. Při tomto prvním odběru uloží pracovník odběrové stanice veškeré potřebné informace o dárci do databáze, kde jsou později i ukládány informace o jednotlivých odběrech, respektive datum odběru.

Takto zaregistrovaný dárci se již může objednat online. Ještě před provedením první online objednávky si musí založit účet na daném serveru. Vytvoření takového účtu je podmíněno ověřením pomocí rodného čísla, zda je již daný uživatel zaregistrován v databázi dárců. V případě, že rodné číslo v databázi dárců není, tak není možné vytvořit online účet. Ale pokud je rodné číslo již v databázi obsaženo, tak dojde k vytvoření nového účtu uživatele, kde se do databáze uloží uživatelské jméno a heslo. Samozřejmostí je, že uživatelské jméno se v databázi nesmí opakovat. V tomto případě je pak uživatel upozorněn, že jméno již nelze použít a musí si zvolit jiné. Pro ukládání jména a hesla uživatele je využita jiná databáze, než ve které jsou uložena dárcova osobní data, a to hlavně z důvodu bezpečnosti dat. Důvod je ten, že v této databázi je heslo uloženo v zašifrované podobě. To je jedna z mnoha výhod architektury ASP.NET MVC, kde při zakládání nové internetové aplikace je takováto databáze vytvořena automaticky, včetně způsobu šifrování hesla.

Po úspěšně provedeném založení účtu a při každém dalším následujícím přihlášení jsou zobrazena dárcova data spolu s nabídkou možností, které může uživatel využít. Kromě přihlášení se na odběr, si může zjistit i další informace, například kolikrát a kdy již daroval krev, může také měnit své údaje v databázi dárci. Samozřejmostí je, že může měnit pouze některé údaje, jako je adresa, telefon nebo email a nemůže měnit data jako krevní skupina nebo rodné číslo.

5.1 Požadavky dárci

Dárci, který se již zaregistroval a je přihlášený ke svému účtu, tak již může přistoupit k samotnému objednání. Zde jsou ale jisté omezující podmínky, které musí být splněny, aby došlo k zaevidování požadavku do databáze. Podmínky jsou tyto:

- na vybrané datum nesmí být již zaregistrován maximální počet dárců, limit je nastaven na 60 dárců denně
- od posledního odběru krve musí uplynout k vybranému datu minimálně 3 měsíce
- dárci nesmí být již objednan na jiné datum
- objednávka nesmí být na datum v minulosti

V případě, že nejsou tyto podmínky splněny, je dárci na toto upozorněn a může si zvolit jiné datum. Jakmile jsou všechny podmínky splněny, tak se datum plánovaného odběru spolu s identifikačním číslem dárci uloží do tabulky objednávek v databázi.

Další možností dárci je zrušení již provedené objednávky. To lze uskutečnit pouze v limitu do 24 hodin před plánovaným odběrem. Po této době již není možné odběr zrušit online a dárci musí zrušit objednávku například telefonicky.

5.2 Požadavky obslužného personálu

Zaměstnanci transfuzního oddělení mají na pracovním počítači nainstalovaný program, který je propojen z databází dárců. V této databázi jsou uloženy data objednávek, tak i veškeré informace o dárcích. Do této databáze také příslušný zaměstnanec ukládá data v případě registrace nového uživatele.

Důležitou funkcí tohoto programu je výběr data, kdy se budou provádět odběry. Zaměstnanec si vybere potřebné datum a zobrazí se mu seznam objednaných dárců. Pak mu již jen stačí zadat příkaz na tisk karet a karty jednotlivých dárců se vytisknou. Na tyto karty se pak budou vyplňovat údaje o provedených vyšetřeních před odběrem krve. Jedná se o údaje jako krevní tlak, puls a výsledky z rozboru krve provedeného ještě před plánovaným odběrem. Z těchto informací pak již příslušný doktor posoudí, zda dárci může jít darovat krev anebo ne.

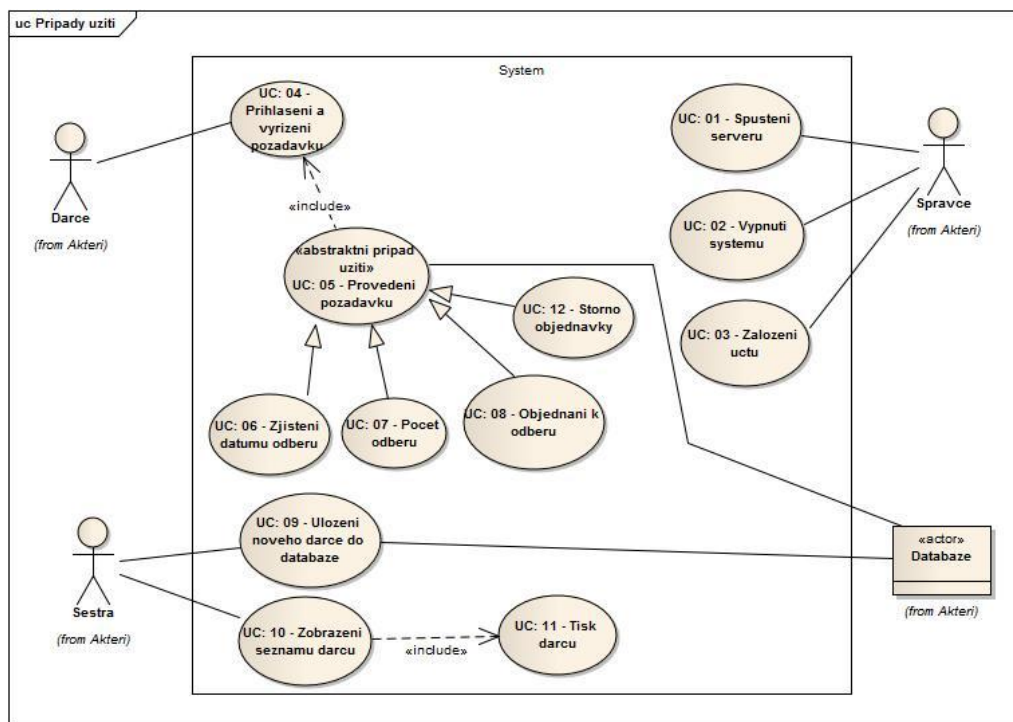
Po potvrzení doktorem, že dárci může absolvovat odběr krve, se do databáze uloží datum provedeného odběru a celý cyklus od provedení objednávky až po provedení odběru končí, včetně vymazání data objednávky a dárci tak může vytvořit objednávku novou.

5.3 Návrhové diagramy

Pro vytvoření návrhu aplikace bylo použito vývojové prostředí Enterprise Architect verze 8.0. V tomto prostředí lze vytvářet návrhové diagramy, pomocí kterých se poté

programuje samotná aplikace. Při vytváření těchto diagramů jsou objasňovány požadavky, které se od aplikace očekávají.

Základním diagram pro rozdělení činností jednotlivých aktérů je diagram *případů užití*. V tomto diagramu jsou zakresleny jednotlivé činnosti, které mohou aktéři s aplikací provádět.



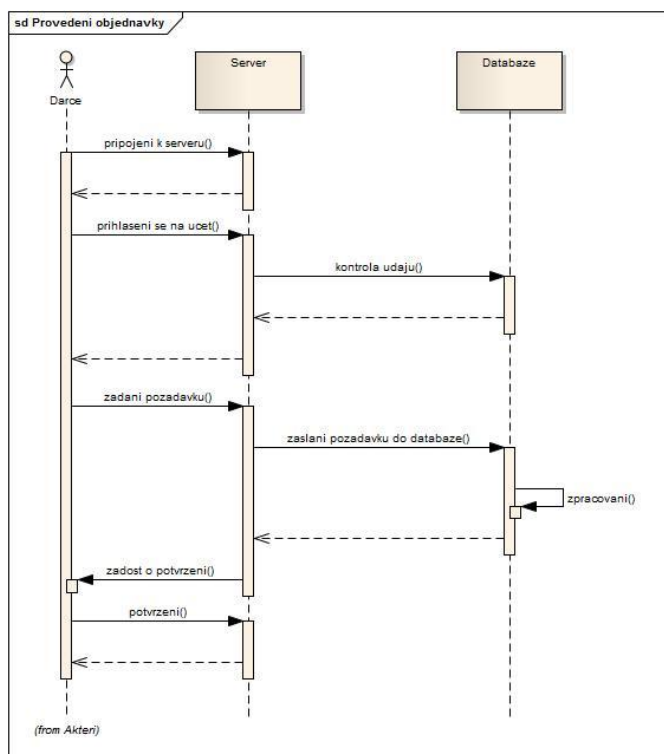
Obr. 6. Diagram případů užití

K jednotlivým případům užití je vždy vytvořen popis, který uvádí co je náplní činnosti daného případu užití. Spolu s popisem je také vytvořen scénář, kde je v jednotlivých krocích tato činnost popsána. Například pro případ užití č. 8 - Objednání k odběru vypadá scénář takto:

1. Uživatel si vybere datum odběru.
2. Systém překontroluje, zda vybrané datum odpovídá požadavkům.
3. Systém poté zobrazí datum na obrazovce se žádostí o potvrzení data.
4. Uživatel si datum překontroluje a potvrdí.
5. Systém uloží toto datum do databáze.
6. Systém zobrazí zprávu o úspěšně provedeném uložení.
7. Případ užití končí.

K základnímu scénáři je zpravidla vytvořen i alternativní scénář, který popisuje situace, kdy není splněn nějaký požadavek v základním scénáři. Těchto alternativních scénářů může být pro každý základní scénář několik.

V návaznosti na případy užití a jejich scénáře se vytváří i grafický pohled na řešení problému. Jedná se o *sekvenční diagramy*. Zde jsou důležitou součástí *lifelines* (čáry života), které představují vždy konkrétního účastníka. Mezi těmito účastníky pak existuje komunikace, která se popisuje pomocí zpráv, které si účastníci předávají. Tyto zprávy jsou řazeny sekvenčně tak, jak postupují v čase.



Obr. 7. Sekvenční diagram

Dalším důležitým diagramem je diagram *funkčních požadavků*. V tomto diagramu jsou popsány jednotlivé funkce systému, které může uživatel využít, a jejich návaznosti na sebe. I v tomto diagramu pro každý požadavek existuje popisek, který doplňuje informace o tom, co daný požadavek má na starost. Požadavky, které nesouvisí s funkcemi z pohledu uživatele, popisuje diagram *nefunkčních požadavků*. V tomto diagramu jsou zobrazeny požadavky, které se od systému očekávají. Jedná se hlavně o zabezpečení, spolehlivost, dostupnost apod.

Pro třídy nebo pro případy užití se vytváří *aktivitní diagramy*. V těchto diagramech jsou *tokens* (objekty), které prochází přes jednotlivé akce, kde jsou zpracovány a poslány

na další akci. Důležitou součástí v těchto diagramech jsou akce rozhodnutí, kde dochází k více možnostem, kam bude token směřován. Například pro přihlášení uživatele je vyžadována akce zadání jména a hesla. Dalším krokem je rozhodnutí, které popisuje úspěšné nebo neúspěšné přihlášení. Když je přihlášení úspěšné, tak token pokračuje k další akci, jinak dojde k návratu na začátek přihlášení.

Posledním vytvořeným diagramem je *model tříd*. V tomto modelu se pomocí již vytvořených diagramů vytváří třídy, které pak slouží jak vzor pro třídy vytvářené přímo v aplikaci. Tyto třídy již obsahují jednotlivé atributy a operace, které se od dané třídy očekávají. Zároveň jsou vytvářena spojení, která existují mezi třídami.

6 TVORBA PROJEKTU

Projekt má sloužit jako prototyp aplikace pro online objednávání dárců na odběr krve. Před vytvořením samotné aplikace je zapotřebí vytvořit databázi, ve které jsou v tabulkách uložena data, se kterými aplikace poté pracuje. Tímto způsobem je poté možné využít principu třívrstvé architektury klient-server, kde jsou data uložena na jiném serveru, než na kterém běží samotná aplikace. Na vytvoření prototypu aplikace je využit framework ASP.NET MVC 3. Jedná se v současné době o nejnovější framework dodávaný firmou Microsoft, ovšem v beta verzi lze již použít framework ASP.NET MVC 4.

6.1 Vývojové prostředí

Pro vytvoření aplikace bylo použito prostředí Microsoft Visual Studio 2010. Aby bylo možné vytvářet aplikace použitím frameworku MVC 3 je zapotřebí k tomuto studiu doinstalovat ještě další komponenty [14].

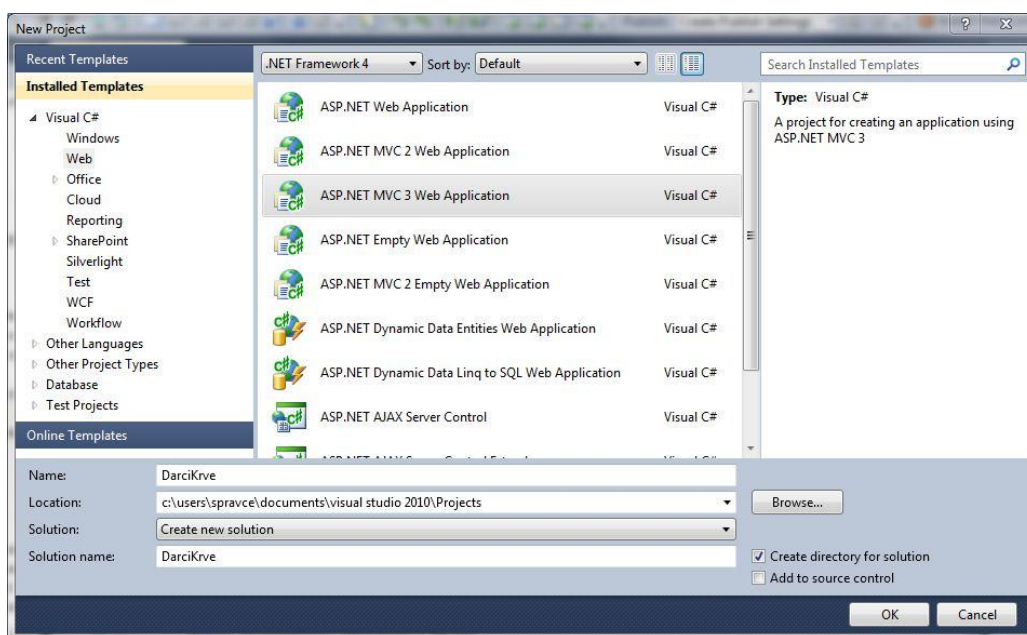
- Servis pack 1 pro Visual Studio 2010
- Web Platform Installer 3.0
- ASP.NET MVC 3 Tools Update
- SQL Server Compact 4.0

Nejdříve je potřeba nainstalovat Web Platform Installer 3.0. Tento instalátor pak spravuje instalaci všech ostatních komponent do Visual studia. Po nainstalování těchto doplňkových komponent je již možné vytvořit nový projekt jako MVC 3 aplikaci. Po vytvoření projektu, a stejně tak kdykoliv v průběhu jeho vytváření, je možné přímo ve Visual studiu nastavit úroveň validace webových stránek, který zkontroluje chyby v HTML kódu a vypíše jejich seznam. Tento seznam je rozdělen na varování a zprávy. Je to stejný princip, kterým můžeme zkontrolovat jakoukoliv webovou stránku pomocí online validátoru W3C.

Přímo ve Visual studiu je možnost vytvořit i databázi. Visual studio ovšem v tomto případě slouží pouze jako grafický nástroj na vytvoření databáze. Aby se mohla databáze vytvořit, musí být v počítači nainstalovaný program SQL Server 2005 nebo 2008. Pak již Visual studio pracuje na podobném principu při vytváření databází, jako se pracuje v SQL Server Management Studiu, které je jinak přímo určeno pro vytváření databází v SQL Serveru.

6.2 Založení projektu

Pro založení projektu, ve kterém se bude využívat propojení s databází, je potřeba spustit Visual studio s administrátorskými právy. Jinak by nedošlo k navázání komunikace s databází. Pro založení nového projektu, který využívá frameworku MVC 3, se vybere webová šablona, ještě je potřeba zkontrolovat, pro který programovací jazyk je šablona určena, v tomto případě C#. Tím se zobrazí nabídka dostupných možností pro vytvoření webového projektu.



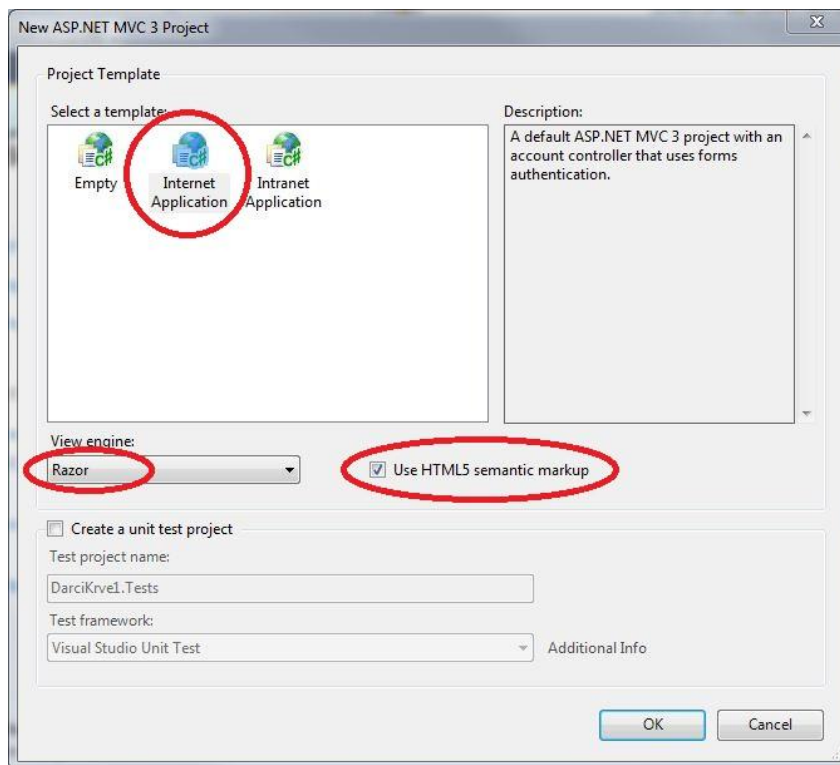
Obr. 8. Nový projekt

Po zobrazení dostupných možností šablony se vybere *ASP.NET MVC 3 Web Application*. V případě, že není zobrazena v nabídce, musí se zkontrolovat nastavená verze .NET Frameworku. Změna se provádí přímo v okně nového projektu. Nahoře v tomto okně je rozbalovací nabídka, kde se potřebný framework vybere. Pro MVC 3 to musí být Framework 4 anebo novější.

Po nastavení všech těchto parametrů již pak stačí zadat jméno projektu a vybrat umístění pro uložení nového projektu. Ještě je dobré zkontrolovat, zda je zaškrtnutá možnost *Create directory for solution*. Tím se vytvoří jediná složka, ve které jsou uloženy všechny vytvářené soubory nového projektu. V tuto chvíli již stačí zmáčknout tlačítko OK a přesunout se na další nastavení vytvářenému projektu.

V druhém, a posledním, okně při zakládání nového projektu se vybírá, jaký typ aplikace se vytvoří. V dostupné nabídce je vytvoření prázdné aplikace, internetové

aplikace nebo intranetové aplikace. Jak už z názvů plyne, pro vytvoření webových stránek dostupných odkudkoliv, je vhodné vybrat internetovou aplikaci. V takto vytvořené aplikaci se již automaticky vygenerují soubory pro přihlašování uživatelů.



Obr. 9. Nastavení parametrů nové aplikace

Z obrázku je vidět, které parametry se dají nastavit pro nový projekt. Kromě internetové aplikace je také důležité, jaký se zvolí *engine*. V takto nakonfigurovaném Visual studiu je možnost pouze mezi enginey *Razor* a *ASPX*. Samozřejmě je možné nainstalovat i další enginey. Pro využití možností architektury MVC 3 je přímo určen engine *Razor*, na rozdíl od engineu *ASPX*, který byl určen pro dřívější typy struktur MVC.

Další možností je využití jazyka HTML5. Zaškrtnutím této položky, bude automaticky generovaný HTML kód v souboru *_Layout.csthml* obsahovat novější způsoby značení některých prvků. Jedná se tak pouze o vygenerování těchto prvků, které se mohou později upravit podle konkrétních potřeb.

Před konečným potvrzením vytvoření nového projektu je možnost vytvořit *unit test project*. Ten umožňuje vytvářet jednotlivé testy na kontrolu různých možností zadání od uživatele. Takto se mohou testovat jednotlivé části právě vytvářeného projektu a není nutné spouštět projekt a testovat ho přímo ve webovém prohlížeči zkoušením všech

možností zadání. To je výhodné hlavně u rozsáhlých projektů, kde se vyplatí vytvářet testovací kód pro různé možnosti zadání.

Po nastavení všech těchto parametrů pak stisknutím tlačítka OK dojde k vytvoření již všech přednastavených souborů a založení nové aplikace je u konce.

6.3 Vytvoření databáze

Databáze byla vytvořena přímo ve Visual studiu. Založení databáze ve Visual studiu je velice jednoduché. Stačí pouze pravým tlačítkem kliknout na složku *App_Data* a z nabídky vybrat *Add->NewItem*. V zobrazeném okně se z nabízených šablon vybere šablona *Data*. Tím se zobrazí možnosti pro tuto šablonu. Vybere se *SQL Server Database* a pak již stačí zadat název pro novou databázi a její vytvoření je hotovo.

Aby bylo možné s takto nově vytvořenou databází pracovat, musí být ve Visual studiu zobrazené okno *Server Explorer*, ve kterém jsou vytvořené databáze zobrazené. V případě, že se databáze nezobrazí, stačí na ní ve složce *App_Data* dvakrát klepnout myší a databáze se v *Server Exploreru* zobrazí. V tuto chvíli je již možné s databází pracovat stejně jako v *SQL Serveru Management Studiu*.

6.3.1 Databáze dárců krve

V databázi dárců krve je potřeba ukládat a zpětně získávat data o dárcích, jejich odběrech a objednávkách. Podle těchto požadavků se vytvoří tabulky:

- tabulka dárců
- tabulka objednávek
- tabulka odběrů

Nejdůležitější tabulkou je tabulka dárců krve, kde jsou uloženy veškeré informace o dárcích. V této tabulce je primárním klíčem *ID-dárce*. Pro *ID-dárce* je zvolen datový typ *integer* a dále je nastaven *Identity Specification*, kde jako výchozí hodnota je nastavena 1 a přičítat se bude po jedné. Tím je zajištěno jedinečné číslo pro každého dárce. Dále jsou vytvořeny položky: Jméno, Příjmení, Rodné Číslo, Ulice, PSČ, Město, Telefon, E-mail, Krevní Skupina a Rh-faktor. Kromě položek Telefon a E-mail jsou všechny ostatní položky povinné. Pro položky Rodné Číslo, Krevní skupina a Rh-faktor jsou vytvořeny ještě omezující podmínky pro vkládaná data.

Omezení se provede tak, že se pravým tlačítkem klikne na řádek, pro který se podmínka vytváří a vybere se položka *Check Constraints*. Pro Rodné Číslo je vytvořeno omezení, aby bylo devítimístné nebo desetimístné, jako druhá podmínka pro Rodné Číslo je ověření, zda je dělitelné 11.

```
( [RodneCislo] % (11) = (0) )
```

Jde o dělení modulo, kde se pouze zkontroluje, zda po dělení nezůstává žádný zbytek. Další omezení je pro Krevní Skupinu, kde jsou zadány všechny druhy krevních skupin.

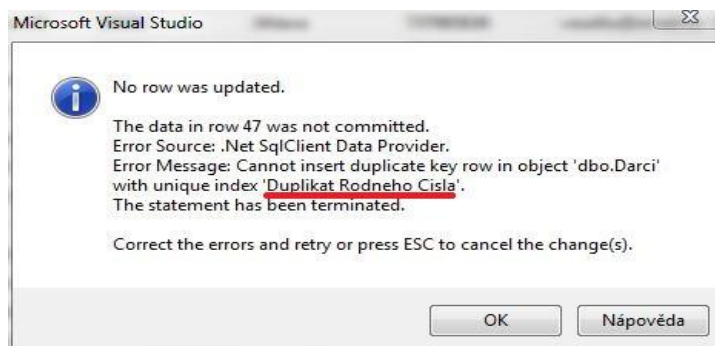
```
( [KrevniSkupina]='A' OR [KrevniSkupina]='B' OR  
[KrevniSkupina]='AB' OR [KrevniSkupina]='0' )
```

Tím je zaručeno, že při zadávání krevní skupiny nedojde k nechtěnému zadání jiného údaje. Stejným způsobem je nastaveno omezení i pro Rh-faktor.

```
( [RH-faktor]='P' OR [RH-faktor]='N' )
```

Takto nastavené omezení zajistí, že vkládané hodnoty budou pouze P – jako pozitivní a N – jako negativní.

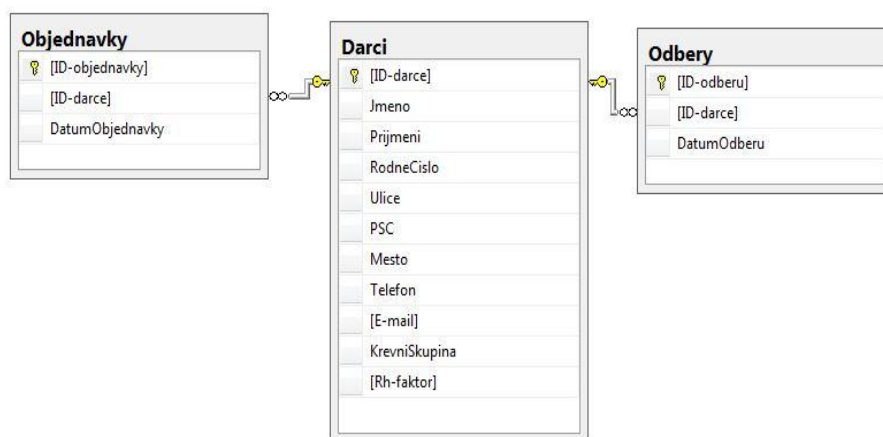
Další způsob jak zajistit správnost zadávaných dat do databáze je nastavení jedinečné hodnoty pro položku Rodné Číslo. Postup provedení je jednoduchý. Stačí pouze pravým tlačítkem kliknout na řádek Rodné Číslo a zde vybrat položku *Indexes/Keys*. V zobrazeném okně pak stačí překontrolovat řádek *Columns*, zda je vybrán správný sloupec a poté již stačí změnit hodnotu v řádku *Is Unique* na *YES*. Tím je zaručeno, že kdyby při zadávání rodného číslo došlo náhodou k zadání již dříve zadaného číslo, došlo by k upozornění na tuto chybu.



Obr. 10. Upozornění na stejné rodné číslo

Jak je z obrázku patrné, dojde k upozornění na vkládání duplikátního klíče. Při jeho vytvoření je důležité zvolit správný název pro tento index, aby bylo co nejjednodušší zjištění příčiny, která způsobila zobrazení této chybové zprávy.

V databázi je vytvořen i diagram, kde je zajištěno provázání mezi jednotlivými tabulkami v databázi. Provázání je provedeno pomocí primárního klíče v tabulce dárců a cizími klíči, jak v tabulce odběrů, tak i v tabulce objednávek.



Obr. 11. Diagram databáze dárců

6.3.2 Databáze online uživatelů

Druhá databáze, která je vytvořená v aplikaci, slouží pro uložení přihlašovacích údajů uživatelů. Tato databáze je vygenerována automaticky při zakládání nového projektu jako internetové aplikace. Ovšem v okně *Solution Explorer* se nezobrazuje. Pro její zobrazení je zapotřebí v tomto okně kliknout na ikonu *Show All Files*, pak se název databáze v tomto okně zobrazí. Pro zobrazení i v okně *Server Explorer* pak stačí pouze dvakrát kliknout na název této databáze. Název je ASPNETDB.MDF.

V této databázi již existuje spousta tabulek, ale pro potřeby této aplikace jsou potřebné pouze 2 tabulky:

- tabulka aspnet_Users
- tabulka aspnet_Membership

Do tabulky aspnet_Users se ukládá uživatelské jméno a je tam také uložen datum posledního přístupu uživatele ke svému účtu. V druhé tabulce jsou důležitými položkami heslo uživatele a také email uživatele. Ovšem pro potřeby této aplikace je položka email změněna na Rodné Číslo. To je velice důležité v době, kdy se uživatel registruje do této

databáze. Rodné číslo pak slouží při registraci uživatele jako kontrolní prvek, který zkontroluje, zda je již uživatel zavedený v databázi dárců. Při dalším přihlašování již uživatel rodné číslo zadávat nemusí, ovšem rodné číslo je klíčový prvek, pomocí kterého dojde k zjištění, o jakého dárce v databázi dárců se jedná.

Heslo uživatele se do databáze ukládá v zašifrované podobě. Jedná se o hash kód, který je uložen v databázi a tudíž ani při přímém přístupu k datům v databázi není možné toto heslo zjistit. Příklad uloženého hesla vypadá takto:

```
k54HkLg4z8MGeSpOSTj+kSOPnWc=
```

V případě, že by se tak někdo neoprávněný dostal k těmto údajům, budou mu tyto údaje k ničemu, protože při přihlašování k účtu se heslo v takovéto podobě nemůže použít. Nejjednodušším způsobem, v případě že uživatel heslo zapomene, je zrušit jeho uživatelský účet. To se provede pouhým vymazáním jednoho řádku v databázi ASPNETDB.MDF. Uživatel si pak musí vytvořit účet nový. Vzhledem k tomu, že veškeré údaje o dárci jsou uloženy v jiné databázi, nedojde tak k poškození žádných jiných dat, než těch, které slouží k přihlášení.

6.3.3 Data v databázích

Do vytvořených databází je potřeba zadat zkušební data, pomocí kterých se kontroluje správná činnost aplikace. Je potřeba zadat data do tabulky *Dárci* a do tabulky *Odběry*. Do tabulky dárců je uloženo kolem padesáti fiktivních dárců. Všechny vkládané údaje musí splňovat dříve popsané požadavky, které je potřeba dodržet při jejich vkládání. Do tabulky *Odběrů* bylo vloženo přibližně 300 fiktivních odběrů těchto dárců. Spolu s *ID-dárce* je v této tabulce uloženo i datum odběru. V reálném provozu, by se tato data zadávala pomocí aplikace nainstalované na počítači pověřeného zaměstnance transfuzní stanice.

Do databáze ASPNETDB.MDF se data vkládají již požadovaným způsobem, a to při registraci uživatele přímo prostřednictvím webových stránek. Pro lepší kontrolu zadávaného hesla, je po uživateli požadováno při zakládání účtu potvrzení hesla. Tím se zajistí situace, že při prvním zadání se uživatel překlepne. Samozřejmostí je i to, že heslo se při zadávání zobrazuje pouze tečkami.

6.4 Vytváření souborů komponenty Model

Soubory vytvořené jako model úzce souvisí s databázemi v projektu. Vzhledem k tomu, že v tomto projektu existují 2 databáze, tak jsou vytvořeny 2 soubory ve složce *Models*. Existují 2 základní způsoby jak vytvořit tento model. Prvním a složitějším, je vytvořit daný model ruční vypsáním všech tříd a jejich vlastností, které požadujeme v projektu zpracovávat. Jedná se o položky – sloupce – v databázi, se kterými chceme v aplikaci pracovat. Druhý způsob je vytvořit daný model pomocí průvodce ve Visual studiu.

6.4.1 Model pro databázi ASPNETDB.MDF

Tento projekt je vytvořen jako internetová aplikace, a tak již při založení projektu byl vytvořen i model pro účet online uživatelů. Jedná se o soubor *AccountModels.cs* ve složce *Models*. Zde jsou již vytvořeny všechny třídy i jejich vlastnosti, které jsou potřeba pro ovládání účtu uživatele. Třídy jsou:

- `ChangePasswordModel`
- `LogOnModel`
- `RegisterModel`

Ve třídě *LogOnModel* jsou vytvořeny vlastnosti jméno, heslo a vlastnost pro pamatování si přihlášení. Z toho vyplývá, že vlastnosti této třídy slouží pro přihlašování uživatelů ke svému účtu. Ke každé vlastnosti, nejen v této třídě, lze vytvořit kontrolu vkládaných údajů, tzv. validaci dat. Tato validace se provádí, když uživatel zadává nějaká data do připraveného formuláře. V případě, že data nejsou v požadovaném formátu, je uživatel upozorněn na chybné zadání dat.

```
[Required(ErrorMessage = "Jméno musí být zadáno.")]
```

Příkaz *Required* kontroluje, zda uživatel zadal data. Když uživatel žádná data nezadá, tak dojde k zobrazení chybové zprávy. Znění této zprávy se zadává pomocí parametru *ErrorMessage*. Aby se uživatel lépe orientoval ve formuláři, který má vyplnit, je nad každé políčko umístěn název dané položky. To se provádí pomocí příkazu

```
[Display(Name = "Jméno")]
```

Aby bylo možné tento příkaz použít, musí se ještě na začátek daného souboru doplnit potřebný *namespace*: `using System.ComponentModel.DataAnnotations;`

Stejným způsobem se nastavují tyto atributy i u ostatních vlastností. Zajímavým atributem, je možnost nastavení zadávání hesla. Při zadávání hesla se nezobrazí zadávané znaky, ale pouze tečky místo těchto znaků. To se provede jednoduše pomocí příkazu

```
[DataType(DataType.Password)]
```

Další třídou v modelu *AccountModels.cs* je třída *ChangePasswordModel*. V této třídě jsou vytvořené vlastnosti, které umožní uživateli změnit své heslo. Zde jsou vytvořeny vlastnosti heslo, nové heslo a potvrzení nového hesla. I zde jsou použity atributy na kontrolu zadání od uživatele, ale je zde i jeden zajímavý atribut, který umožňuje porovnání dvou vlastností mezi sebou. Pro názorný příklad je uveden celý kód, který je vytvořen pro vlastnost potvrzení hesla.

```
[Required]
[DataType(DataType.Password)]
[Display(Name = "Potvrzení nového hesla")]
[Compare("NewPassword", ErrorMessage = "Potvrzení
hesla nebylo úspěšné.")]
public string ConfirmPassword { get; set; }
```

V tomto kódu pouze poslední řádek je vlastnost dané třídy, všechny předchozí řádky jsou atributy, které slouží pro kontrolu vkládaných dat. U atributu *Compare* je první parametr – *NewPassword* – pouze název jiné vlastnosti dané třídy. Jedná se o tu vlastnost, se kterou dochází k porovnání hodnot. Přímo v atributu *Compare* se zadá i zpráva, která se uživateli zobrazí v případě, že hesla nejsou shodná.

Poslední třídou je třída *RegisterModel*. Vlastnosti této třídy jsou využity při zakládání nového účtu uživatele. Zde jsou vytvořeny vlastnosti uživatelské jméno, rodné číslo, heslo a potvrzení hesla. Atributy na kontrolu dat jsou zde stejné jako u předchozích tříd, jedinou výjimkou je zde atribut, který má za úkol zkontrolovat délku zadávaného hesla.

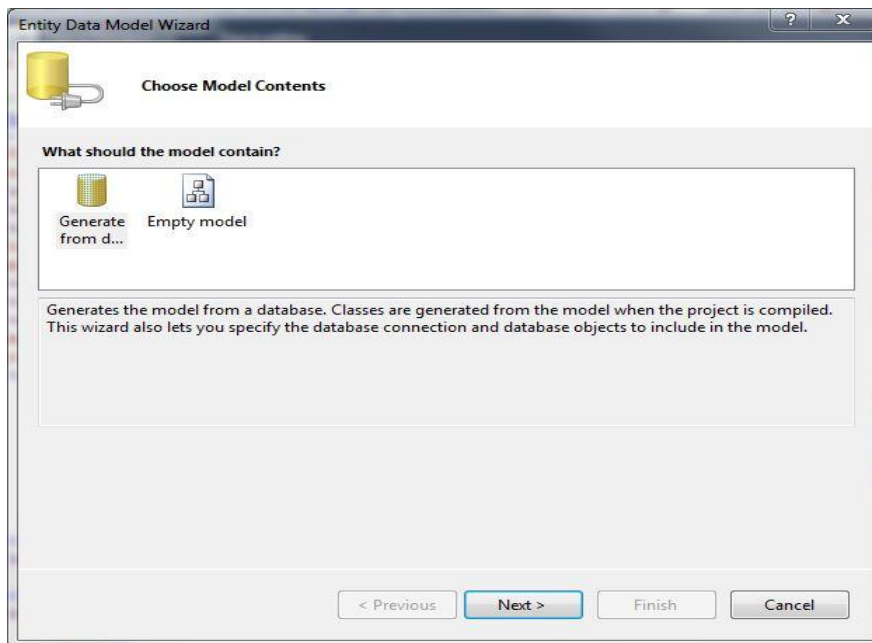
```
[StringLength(100, ErrorMessage = "{0} Heslo musí být
nejméně {2} znaků dlouhé.", MinimumLength = 6)]
```

Tento atribut udává, že maximální délka zadávaného hesla je 100 znaků a minimálně musí heslo obsahovat 6 znaků. V případě, že je heslo kratší, zobrazí se uživateli zpráva, ve které je uvedena minimální požadovaná délka hesla.

6.4.2 Model pro databázi Darcovství.mdf

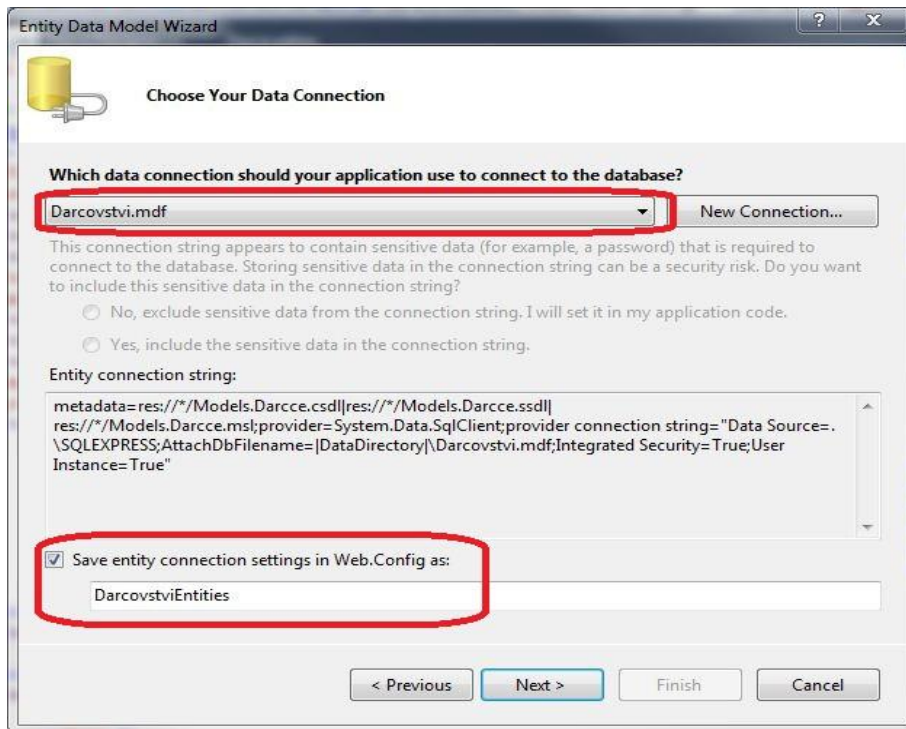
Na rozdíl od předchozího modelu, který byl vytvořen automaticky při založení projektu a kde došlo jen k úpravě atributů, je potřeba pro databázi dárců již model vytvořit. Před vytvořením modelu je potřeba si rozmyslet, které tabulky s databáze bude potřeba využít.

Model se vytvoří tak, že se pravým tlačítkem klikne na složku *Models* a vybere se položka *Add ->New Item*. V zobrazeném okně se vybere šablona *Data* a v ní se vybere položka *ADO.NET Entity Data Model*. Další krokem je zobrazení okna, kde dojde k nabídce, zda vytvořit tento model jako prázdný, nebo ho vygenerovat z databáze.



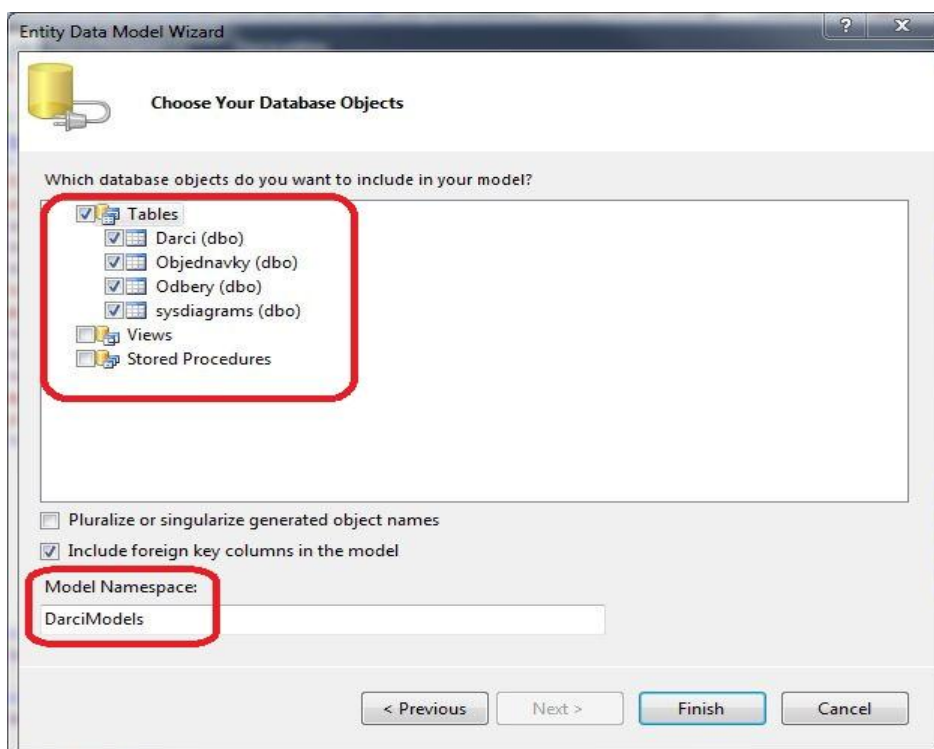
Obr. 12. Výběr obsahu modelu

Výběrem položky generovat, dojde v daném modelu k vytvoření tříd pro jednotlivé tabulky a vlastností pro jednotlivé sloupce v tabulce. Při zvolení této možnosti, dojde po stisknutí tlačítka *Next* k zobrazení dalšího okna, kde se volí možnost, z jaké databáze se budou data importovat. Dále je v daném okně zobrazeno zaškrťovací políčko, které určuje, zda se uloží nastavení spojení pro danou databázi v souboru *Web.config* a zadá se název pro takto vytvořenou entitu. Tato entita je později využita pro vytvoření objektu, pomocí kterého lze později získat data z databáze.



Obr. 13. Výběr databáze a vytvoření entity

Po vybrání databáze se zobrazí již poslední okno, kde je možnost vybrat konkrétní tabulky, které je potřeba zahrnout do modelu.



Obr. 14. Výběr tabulek do modelu

Po vybrání potřebných tabulek, se zadá již jen název vytvářeného modelu a stisknutím tlačítka *Finish* je vytvoření modelu ukončeno. Tím ale není práce s daným modelem ukončena.

Dalším krokem je vytvoření atributů pro kontrolu zadávání dat od uživatelů. Uživatel bude zadávat data, když bude chtít změnit své údaje a když bude zadávat datum, na který se bude objednávat. Ze všech údajů, které jsou o uživateli evidovány, bude moci změnit jen některé:

- ulici a číslo popisné
- město
- PSČ
- telefon
- email

Všechny ostatní údaje jsou neměnné a v případě změny příjmení, bude změnu provádět pracovník na transfuzní stanici. Požadavky u ulice a města jsou, že musí být zadány a je také nastavena maximální délka možného řetězce zadávaných dat.

Pro PSČ, které musí být zadáno, a pro telefon, který zadaný být nemusí, je nastaven atribut *regulární výraz*.

```
[RegularExpression("[0-9]{5}", ErrorMessage="Poštovní  
směrovací číslo obsahuje 5 číslic.")]
```

V tomto atributu je v hranaté závorce určeno, že vložená data jsou číslice a ve složené závorce je nastaven počet číslic, které musí být zadány. Pro lepší přehlednost, je dána i informativní zpráva pro uživatele v případě špatného zadání dat. Podobným způsobem je nastaven i atribut u telefonu, kde je ovšem povolen jiný počet číslic – konkrétně devět.

Pro ověření platnosti zadávaných dat u emailu je kontrola provedena také pomocí regulárního výrazu. Ovšem v tomto případě je tento výraz složitější, podrobněji například na stránkách firmy Microsoft [15]. Dochází ke kontrole po částech. První část je před zavináčem, druhá část je název poskytovatele emailu a třetí část je doména. Jinak je princip funkce stejný jako u předchozích ověření. V případě nesprávných dat, je uživatel na toto upozorněn a musí data upravit, pokud chce, aby se data uložila do databáze.

Při zadávání data objednávky, dochází již k více kontrolním mechanismům pro ověření platnosti tohoto data. Jako základní ověření je formát, v jakém je datum napsán. Požadovaný formát je *dd.mm.rrrr*. Pro kontrolu tohoto formátu je již vytvořen atribut přímo v jazyce C#.

```
[DataType (DateTime.Date) ]
```

Tímto je ale ověřen pouze formát vloženého data, ale je potřeba při vkládání data ověřit ještě další záležitosti. Jedná se o kontrolu, zda již uplynula dostatečně dlouhá doba od posledního odběru a jestli nemá dárce evidovanou již jinou objednávku. K ověření těchto dat je ale zapotřebí získání dat z databáze pro daného dárce. To je ovšem záležitost, která se ověří v *controlleru*. V modelu jde ovšem vyřešit požadavek, zda vkládané datum není v minulosti.

Pro vyřešení tohoto požadavku je zapotřebí implementovat do třídy *Objednavky* rozhraní *IValidatableObject*. Toto rozhraní implementuje metodu *Validate* spolu s generickým rozhraním *IEnumerable<>*, pro jehož použití je zapotřebí do projektu přidat ještě namespace *using System.Collections.Generic*. Celý kód, který ověří, že zadané datum není v minulosti, vypadá takto:

```
public IEnumerable<ValidationResult>
Validate(ValidationContext validationContext)
{
    if (DatumObjednavky <= DateTime.Now)
    {
        yield return new ValidationResult
            ("Nelze vytvořit objednávku v minulosti");
    }
}
```

Kromě podmínky *if*, se jedná o syntaxi rozhraní *IValidatableObject*. V podmínce dojde pouze k porovnání, zda zadané datum není starší nebo stejné než aktuální datum. Vzhledem k tomu, že se jedná o generickou kolekci, je vhodné použít klíčové dvouslovo *yield return*, které vrací novou instanci třídy *ValidationResult*, kde je v parametru jako řetězec zapsána chybová zpráva, která se zobrazí v případě nesplnění podmínky.

6.5 Vytvoření webové služby

K vytvoření webové služby je opět použito Visual Studio 2010. Pro webovou službu je založen nový projekt, ve kterém je použita šablona WCF Service Application. Tento projekt je nazván *DarciKrveService* a této služby je využito v projektu *DarciKrve*, který je klientem této služby.

V tomto projektu je soubor *IDarci.cs*, který slouží jako rozhraní. Aby se mohlo využít tohoto rozhraní jako webové služby, musí být před názvem tohoto rozhraní deklarován atribut *[ServiceContract]*. V takto označeném rozhraní jsou pak jednotlivé metody označeny atributem *[OperationContract]* a klient tak ví, které metody může používat. Tyto metody jsou klienty využity pro získání potřebných údajů z databáze dárců, i když klient nemá přímý přístup k této databázi.

Pro práci s daty v tomto projektu je přidán model z předchozí kapitoly, ve kterém jsou vytvořeny třídy pro jednotlivé tabulky databáze. Před tyto třídy musí být přidán atribut *[DataContract]*, který definuje, které datové typy se budou ve službě používat. V každé této třídě musí být k jednotlivým vlastnostem přidán ještě atribut *[DataMember]*.

Oba vytvořené soubory jsou využity v souboru *Darci.svc.cs*, ve kterém je ve třídě *Darci* implementováno rozhraní *IDarci*. V implementovaných metodách jsou vytvořeny příkazy, které jsou popsány dále, které vyhledávají požadovaná data v databázi nebo do databáze ukládají data nová. Důležitou částí je určení návratového typu jednotlivých metod, pomocí kterých klient získává nazpět informace a stejně tak jsou důležité parametry těchto metod, do kterých jsou zase potřebné informace klientem zasílány do služby.

6.5.1 Web.config

V souboru *Web.config* jsou uloženy konfigurační nastavení služby. Důležitými částmi tohoto souboru je nastavení služby – systém, ve kterém jsou poskytovány *endpointy*, které slouží pro příjem a odesílání SOAP zpráv. Tyto *Endpointy* tvoří 3 části:

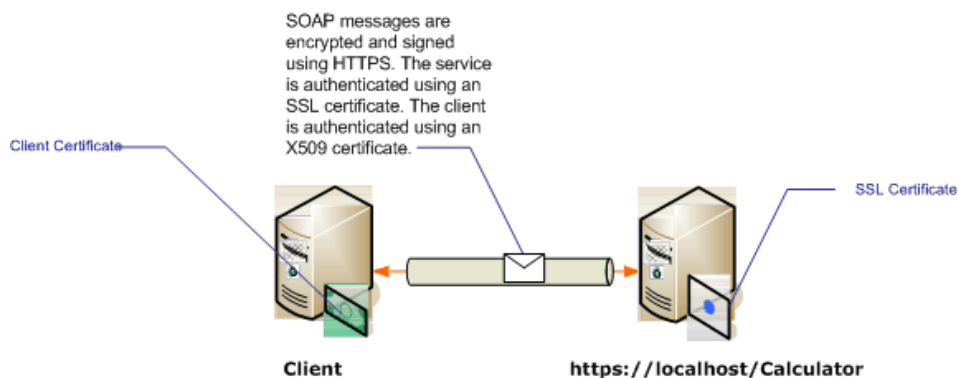
- *address* – určuje, kde služba poběží
- *binding* – určuje použitý komunikační protokol, kódování, zabezpečení atd.
- *contract* – specifikuje rozhraní, které služba poskytuje

Všechny tyto nastavení lze psát přímo v souboru nebo je možné využít zabudovaného *Edit WCF Configuration*, který lze získat v nabídce po klepnutí pravého tlačítka myši na soubor *Web.config*. V zobrazeném okně je na výběr velké množství

možností, jak nastavit jednotlivé parametry služby. Jednou ze základních možností je nastavení *binding* na požadovaný komunikační protokol. Mohou to být:

- BasicHttpBinding
- WsHttpBinding
- WsDualHttpBinding
- NetTcpBinding
- NetNamedPipeBinding

Takovýchto *bindingů* existuje daleko více, a každý poskytuje jiné možnosti zabezpečení a kódování. Při vybrání požadovaného endpointu je možnost nastavení jeho zabezpečení v záložce *Security*. Pro zabezpečení neoprávněného přístupu ke službě je možnost vybrat v záložce *TransportClientCredentialType* položku *Certificate*. Používá se certifikát X.509. Každý klient má certifikát, který je ověřován službou [16].



Obr. 15. Způsob ověření klienta [16]

Po ukončení všech nastavení v editoru, se tyto změny projeví i v souboru *Web.config*.

6.5.2 Nastavení služby u klienta

Aby klient mohl danou službu využívat, je zapotřebí přidat referenci na službu. To se provede kliknutím pravým tlačítkem na složku *References* a přidáním nové reference. Dalším krokem je vytvoření složky *Service References*. Po kliknutí pravým tlačítkem na *References* se vybere *Add Service Reference*. Zde se do políčka adresy zadá adresa, kde je vytvořena služba a v okně *Services* je možné zobrazit dostupné operace. Na závěr je ještě nutné zadat název *namespace* pro tuto servisní referenci. Když je takto vytvořena konfigurace, je již možné vytvořit instanci proxy třídy.

```
private DarceReference.DarciClient darce = new
DarceReference.DarciClient();
```

Takto vytvořená instance slouží jako prostředník mezi klientem a službou. Pomocí této instance se poté volají požadované metody ze služby.

6.6 Controllers a Views

Po vytvoření modelů je již možné začít vytvářet soubory *controllers* a pohledy, které s nimi souvisí. Při vytvoření nového projektu je již vytvořen *HomeController*, který obsahuje pouze jednu akční metodu *Index*. Pro tuto metodu je vytvořen pohled, který je zobrazen jako první po načtení webové stránky, a který se také zobrazí po stisknutí tlačítka *Home*.

Po načtení úvodní stránky je zobrazeno základní rozhraní, které slouží uživateli k přesměrování na jím vybranou akci. Důležitou částí je možnost přihlášení se ke svému účtu a možnost zobrazit informace o dárci. Tyto možnosti jsou naprogramovány v souboru *_Layout.csthml*. Pro přihlášení jsou již vytvořeny pohledy ve složce *Views->Account*. Pro zobrazení informací o dárci, je vytvořen nový controller - *DarceController*. Zde jsou naprogramovány akční metody, které zajišťují veškeré požadavky, které má uživatel k dispozici, a pro tyto akční metody jsou vytvořeny pohledy, sloužící pro zobrazení pohledů, které jsou určeny těmito metodám.

6.7 Account

V souboru *AccountController* jsou nadefinovány akční metody, které mají na starost přihlášení uživatele, odhlášení, registraci nového účtu a změnu hesla. Všechny tyto metody jsou již vytvořeny při založení projektu, ale je potřeba je upravit. Kromě toho, že jsou vytvořeny v angličtině a je potřeba přeložit hlášení, která generují, tak je také potřeba určit, kam dojde k přesměrování. To se v akční metodě *LogOn* a stejně tak v metodě *Register* zajistí pomocí návratové metody:

```
return RedirectToAction("Details","Darce");
```

V obou těchto metodách dojde při úspěšném přihlášení nebo provedené registraci k přesměrování na *DarceController* a jeho akční metodu *Details*. V metodě *Register* je ale ještě zapotřebí zajistit porovnání zadávaného rodného čísla, zda odpovídá rodnému číslu v databázi dárců. To je zajištěno pomocí inicializace entity z modelu dárců.

```
private DarcovstviEntities _db=new DarcovstviEntities();
```

Tato entita slouží při vyhledávání v databázi dárců pomocí jazyka LINQ. Pomocí tohoto jazyka, je tedy porovnáním rodných čísel nalezen odpovídající dárci v databázi dárců. Pokud není rodné číslo v databázi dárců nalezeno, tak dojde k zachycení výjimky a je zobrazena zpráva o neplatném rodném čísle. V případě, že rodné číslo je platné, tak dojde k vytvoření nového uživatele.

```
Membership.CreateUser(model.UserName, model.Password,  
model.RodneCislo, null, null, true, null,  
out createStatus);
```

Po zadání tohoto příkazu na vytvoření nového uživatele, dojde ještě ke kontrole, zda vytvoření bylo úspěšné. K tomu slouží poslední parametr tohoto příkazu *createStatus*. Pokud vše proběhlo v pořádku, nastaví se na klientském počítači autorizační *cookie*. Stejný princip je i u akční metody *LogOn*, kde je také po úspěšném přihlášení tato *cookie* vytvořena.

Pro všechny akční metody jsou vytvořeny i pohledy, kromě akční metody *LogOff*, která slouží pro odhlášení uživatele. Zde je příkaz:

```
FormsAuthentication.SignOut();
```

Tento příkaz zajistí odhlášení uživatele a poté dojde k přesměrování na úvodní stránku.

Vytvořené pohledy jsou formuláře pro zadávání požadovaných údajů. Jenom pohled *ChangePasswordSuccess* slouží pouze jako informativní zpráva o úspěšně provedené změně hesla. U všech těchto pohledů tak stačí pouze přeložit do češtiny zprávy, které se zobrazují uživateli.

6.8 Dárce

Pro správu všech požadavků dárci je vytvořen *DarceController* a ve složce *Views* je vytvořena složka *Darce*, která obsahuje soubory s jednotlivými pohledy pro dané akční metody. *DarceController* je potřeba nejdříve vytvořit. Pravým kliknutím tlačítka na složku *Controllers* se objeví nabídka, kde se vybere *Add->Controller*. V zobrazeném okně se zadá jméno controlleru a vybere se nabídka vytvořit prázdný controller. Tím je controller vytvořen a dojde k vytvoření třídy *DarceController*, která dědí od třídy *Controller*.

Pro získávání dat z databází, jak z databáze dárců, tak z databáze uživatelů, je zapotřebí inicializovat objekty entit těchto databází. Vzhledem k tomu, že jsou využívány pouze ve třídě *DarceController*, jsou vytvořeny jako soukromé. K takovýmto členům lze přistupovat pouze v metodách dané třídy.

```
private DarcovstviEntities _db=new DarcovstviEntities();
```

Po vytvoření objektu této entity je již možné začít vytvářet akční metody. Po přihlášení nebo registraci uživatele, dojde k zobrazení jeho údajů. K tomu je vytvořena akční metoda *Details()*.

6.8.1 Details()

V této akční metodě se nejdříve zjistí, který uživatel je přihlášený a pomocí tohoto údaje se zjistí, o jakého dárce se jedná. Pak již dojde k zobrazení jeho údajů na obrazovku. K zobrazení informací o dárci dojde po stisknutí tlačítka *Dárce*. V případě, že není ještě přihlášený, dojde k přesměrování na přihlašovací obrazovku, stejně jako kdyby uživatel stisknul tlačítko *Přihlásit*. Po úspěšném přihlášení dojde v této metodě ke zjištění, který uživatel je přihlášený.

```
MembershipUser uzivatel = Membership.GetUser(  
User.Identity.Name);  
string rodneCislo = uzivatel.RodneCislo;
```

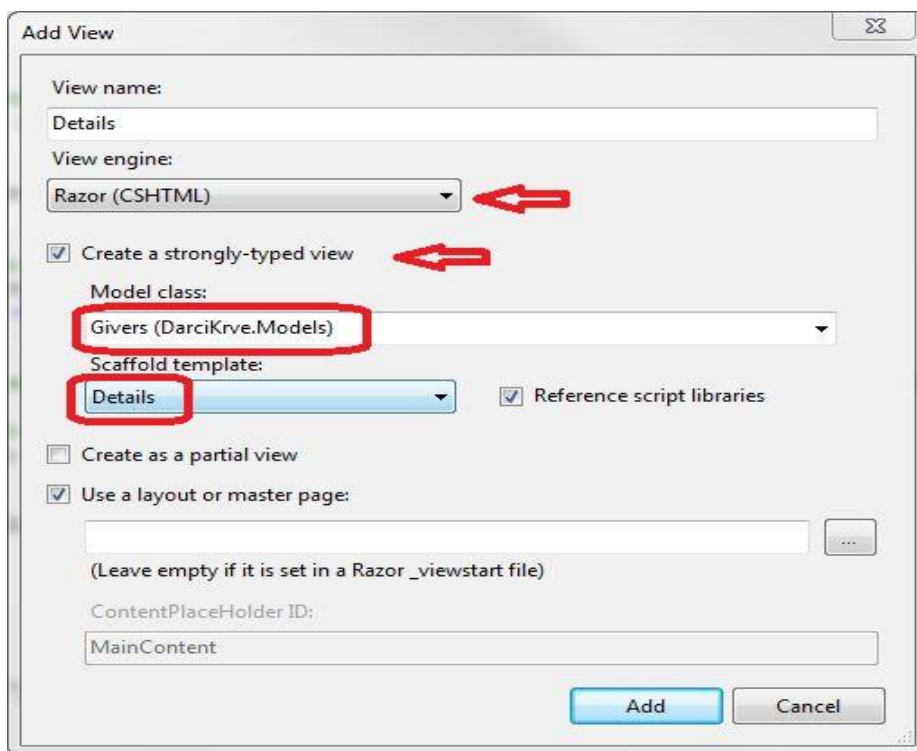
Aby bylo možné zjistit v databázi dárců, o kterého dárce se jedná, je zapotřebí zjistit z databáze uživatelů rodné číslo přihlášeného uživatele a podle něj pak zjistit dotyčného dárce.

```
var darce = (from m in _db.Darci  
            where m.RodneCislo == rodneCislo  
            select m).First();
```

V tomto dotazu se do proměnné *darce* uloží již požadovaný dárce a může dojít k zobrazení jeho dat z databáze dárců na obrazovku. Na to se použije příkaz:

```
return View(darce);
```

Když je takto akční metoda vytvořena, je zapotřebí program zkompileovat a pak je již možné k dané metodě vytvořit pohled. To se provede stisknutím pravého tlačítka myši přímo na dané metodě a vybráním položky *AddView*.



Obr. 16. Okno View

V tomto okně dojde k nastavení požadovaných parametrů. Jméno je výhodné nechat tak je automaticky vygenerováno, protože se shoduje se jménem akční metody, ke které se vztahuje. Další možností je zvolení engineu pro daný pohled. V MVC 3 je nový engine *Razor*. Jeho možnosti jsou popsány v kapitole *Komponenty MVC*, podkapitola *View*. Důležitou částí při vytváření pohledu je možnost využití silně typového pohledu, který umožní do vytvořeného pohledu implementovat položky z vybrané databáze. Zaškrtnutím této možnosti dojde k možnosti vybrat si požadovanou databázi v položce *Model class*. Poslední možností je volba šablony, která vytvoří předdefinovaný návrh pohledu. V tomto případě se jednoznačně nabízí šablona *Details*. Při použití této šablony dojde v pohledu k zobrazení všech informací o dárci. Když jsou takto nadefinovány všechny potřebné parametry, stisknutím tlačítka *Add* dojde k vytvoření souboru *Details.cshtml*.

Zde jsou jednoduchým způsobem určeny data, která se vypíší na obrazovku. Jedná se o jméno, příjmení, adresa apod. Jednotlivé informace se zobrazují pod sebou. Všechny tyto informace jsou v bloku `<fieldset>`. V dalším bloku `<p>` jsou vytvořeny možnosti, které má uživatel k dispozici. Jedná se o možnosti:

- změnit údaje
- objednávka
- zrušit objednávku
- zobrazit provedené odběry

Aby byl daný odkaz aktivní, je možné použít takovýto příkaz:

```
@Html.ActionLink("Změnit údaje", "Edit",  
                new {id=Model.ID_darce})
```

První parametr je zobrazený název, druhý je název akční metody, na kterou se běh programu přesměruje a třetí parametr uloží id dárce, kterého se daná akce týká a které je načteno v akční metodě jako její parametr.

6.8.2 Změna údajů

Akční metoda *Edit* je určena pro uživatele, aby mohl změnit některá svá data, která jsou uložena v databázi dárců. Po zadání nových dat a jejich validaci, jsou poté data uložena do databáze. Jsou vytvořeny 2 metody *Edit*, které jsou přetíženy. Rozdíl mezi nimi je v typu parametru. První metoda má jako parametr datový typ *integer*.

```
public ActionResult Edit(int id) {}
```

Tato metoda v parametru načte ID dárce, které získá při stisknutí tlačítka, jak bylo popsáno v předchozí kapitole. Získané ID dárce je využito pro zjištění dárce z databáze pomocí jazyka LINQ. Když je získán dárce, tak se zobrazí na obrazovce formulář s daty, které je možné změnit. K tomu slouží příkaz:

```
return View(zmenit);
```

Parametr *zmenit* je proměnná, která je výsledkem předchozího dotazu v jazyce LINQ. Pro metodu *Edit(int id)* je poté vytvořen pohled. Stejně jako pro metodu *Details*, lze vytvořit tento pohled využitím šablony. Využitím této šablony dojde k zobrazení všech jednotlivých položek, které jsou o dárci uloženy v databázi. Vzhledem k tomu, že některá data nemůže dárce měnit, je potřeba mu v tom zabránit. Nejjednodušším způsobem je, aby se dané položky vůbec nezobrazily na obrazovce. To lze provést jejich smazáním v kódu nebo jejich potlačením pomocí příkazu. Druhý způsob je výhodný v tom, že je vidět, které položky jsou v databázi obsaženy a v případě pozdějších změn není nutné je znovu hledat.

```
@Html.HiddenFor(model => model.ID_darce)
```

Tento příkaz způsobí, že takovéto položky nejsou zobrazeny uživateli na monitoru. Důležitou částí v pohledu *Edit.cshtml* je příkaz pro uložení změněných dat.

```
<input type="submit" value="Uložit"/>
```

Tento příkaz vyvolá druhou akční metodu *Edit*. Tato metoda má atribut [*HttpPost*]. Tento atribut zajistí, že klient může odeslat na server data jakéhokoliv datového typu. Tím je zajištěno, že když uživatel změní některá data, jsou odeslána na server k uložení. V případě, že se žádný atribut nezadá, tak přednastavený atribut akční metody je [*HttpGet*]. Tento atribut zajistí pouze odeslání URL a hlavičky na server.

Parametrem druhé metody je objekt třídy *Givers*. V tomto objektu jsou uloženy nově zadaná data. Aby se tato data mohla uložit do databáze je zapotřebí ještě pomocí ID dárce zjistit dárce v databázi, u kterého změna proběhne. Po načtení všech těchto dat, dojde ještě ke kontrole, zda je daný model dat platný.

```
if (!ModelState.IsValid)
    return View(puvodni);
_db.ApplyCurrentValues(puvodni.EntityKey.EntitySetName,
    darce);
_db.SaveChanges();
return RedirectToAction("Details");
```

Příkaz *ApplyCurrentValues* má první parametr původní data a druhý parametr data nová. Tento příkaz zajistí, že se v objektu vytvořené entity změní data z původních za nová. V tuto chvíli je již možné uložit data do databáze příkazem *SaveChanges()*. Když jsou data uložena, použije se příkaz k přesměrování na zobrazení informací o dárci v pohledu *Details*. V tomto pohledu jsou již zobrazena nová data a dárce si tak může změnu zkontrolovat.

6.8.3 Objednávka

Vytvoření akční metody *Order*, která má za úkol zajistit uložení do databáze datum objednávky, které uživatel zadá, je vázáno mnoha podmínkami, které je potřeba dodržet. Stejně jako předchozí metoda *Edit*, je i metoda *Order* vytvořena jako přetížená. Ve chvíli, kdy uživatel stiskne tlačítko na objednání je zavolána první metoda *Order*, která ve svém parametru přijímá ID dárce. Hlavním úkolem této metody je ověření, zda již není daný uživatel objednan, protože lze vytvořit vždy jen jednu objednávku. Dotazem v jazyku

LINQ se zjistí, zda již existuje k danému ID dárce objednávka. Pokud objednávka již existuje, tak dojde k přesměrování na akční metodu *Reserved*.

V akční metodě *Reserved* se pro daného uživatele vybere z databáze datum, na které je objednan a ve vytvořeném pohledu dojde k informování uživatele, že je již objednan a na který den je objednávka vytvořena.

Pokud uživatel ještě objednaný není, tak je k této metodě vytvořen pohled, který obsahuje políčko pro zadání data na odběr. Datum je kontrolován jednak v dříve vytvořeném modelu, kde je pomocí atributů zajištěn správný formát vložených dat a je tam vyřešeno i to, že vložené datum musí být nejdříve na další den. Ostatní kontroly, jestli je datum vhodný, jsou řešeny v druhé metodě *Order*, která přetíží první metodu tím, že má jako parametr objekt třídy *Objednavky*. Pro tento parametr je přidán atribut *Bind*, který má ve svém parametru nastaveno omezení *Exclude*, že ID-objednávky a ID-dárce nelze měnit. To způsobí, že při generování pohledu pro danou metodu, bude uživatelům k dispozici pouze pole pro zadání data.

V této metodě je nejdříve potřeba zjistit ID dárce, aby bylo možné zjistit, kdy naposledy daroval krev. To je kvůli podmínce, že od posledního odběru krve musí uplynout minimálně tři měsíce.

```
var datumPoslednihoOdberu = (from m in _db.Odbery
                             where m.ID_darce == darce
                             orderby m.DatumOdberu descending
                             select m.DatumOdberu).First();
```

Tímto způsobem se vytvoří proměnná, ve které je uložen datum posledního odběru dárce. V dotazu je důležitá klauzule *orderby descending*. Tím je zajištěn výběr data posledního provedeného odběru [1].

Dále je v této metodě vytvořena proměnná, ve které je uložen celkový počet objednávek, které už jsou evidovány pro zadané datum.

```
var pocetObjednavek = (from m in _db.Objednavky
                       where m.DatumObjednavky == objednavka.DatumObjednavky
                       select m.ID_darce);
int celkemObjednavek = pocetObjednavek.Count();
```

Do proměnné *pocetObjednavek* jsou uloženi všichni dárce, kteří jsou na daný den již objednáni a v proměnné *celkemObjednavek* je pak uložen jejich počet. Pro přehlednost je ještě vytvořena proměnná *denniLimitOdberu*, kde je uložen maximální počet dárců, které je možné v jeden den objednat.

Když jsou všechny proměnné vytvořené, může již dojít k jejich využití, zda vyhovují zadaným podmínkám. Pro kontrolu maximálního počtu dárců v jeden den je vytvořena podmínka:

```
if (celkemObjednavek > denniLimitOdberu)
{
    ModelState.AddModelError("", "Na vybraný den je
        již objednán dostatečný počet dárců.");
}
```

Jestliže je podmínka splněna, dojde k vypsání chybové zprávy uživateli na obrazovce a uživatel má možnost si vybrat jiný datum. Podobným způsobem je vytvořena i podmínka pro kontrolu, zda od posledního odběru uplynula již požadovaná doba.

```
if (datumPoslednihoOdberu.ToOADate()+90 >
    objednavka.DatumObjednavky.ToOADate())
{
    ModelState.AddModelError("", "Od Vašeho posledního
        odběru neuplynula ještě požadovaná doba");
}
```

V této podmínce je nutné datum přetypovat na číslo, aby se mohla data navzájem porovnat. K tomu je využita metoda *ToOADate*, která převede hodnotu data na číslo, které znamená počet dní od 30. prosince 1899. Tímto způsobem je pak možné přičíst k datu posledního odběru 90 dní a v případě, že je toto číslo větší než číslo, které je určeno pro zadaný datum, tak není splněna podmínka dostatečně dlouhé doby od posledního odběru a dojde k vypsání informativní zprávy. Uživatel, pak stejně jako v přechozím případě, má možnost zadat nové datum.

Pokud jsou všechny podmínky splněny, může již být zadané datum uložené do databáze. Ale vzhledem k tomu, že ID-dárce nebylo nikde zadáno, ale v tabulce uložené být musí, tak je nutné ho přidat k objednávce příkazem:

```
objednavka.ID_darce = darce;
```

V tuto chvíli již opravdu nic nebrání uložení ID-dárce a data do databáze.

```
_db.AddToObjednavky(objednavka);  
_db.SaveChanges();
```

Prvním příkazem je objekt *objednavka* přidán do databáze a druhým příkazem dojde k uložení dat v databázi. Nakonec je v této metodě ještě příkaz k přesměrování na potvrzovací metodu *Confirm*. Pro tuto akční metodu je vytvořen pohled, ve kterém je zadána potvrzovací zpráva o úspěšně provedené objednávce.

6.8.4 Zrušení objednávky

Pro zrušení objednávky slouží akční metoda *Delete*. Opět jsou vytvořeny dvě akční metody, které jsou přetíženy. Úkolem první metody je pomocí ID dárce zjistit, zda je pro daného dárce nějaká objednávka evidována. V případě, že není žádná objednávka v databázi evidována, dojde k přesměrování na akční metodu *NoOrder*. Pro tuto metodu je vytvořen pohled, kde dojde pouze k informaci uživatele, že žádná objednávka nebyla dříve vytvořena.

Pokud se dotazem v jazyce LINQ najde odpovídající objednávka, je pro akční metodu *Delete* vytvořen pohled, kde dojde k zobrazení zprávy, na který den je objednávka vytvořena a zda ji chce uživatel opravdu zrušit.

```
<input type="submit" value="Vymazat"/>
```

Pro potvrzení zrušení je vytvořeno tlačítko *Vymazat*, po jehož stisknutí dojde k vyvolání druhé akční metody *Delete*. Tato metoda je bezparametrická a je v ní zajištěno vymazání dané objednávky z databáze. Nejdříve je pomocí dotazu uloženo do proměnné *darce* ID dárce. To je využito v dalším dotazu, kde se nalezne pro daného dárce odpovídající objednávka.

```
var zrusit = (from m in _db.Objednavky  
             where m.ID_darce == darce  
             select m).First();
```

Když je tímto způsobem nalezena odpovídající objednávka k vymazání, provede se před konečným zrušením objednávky ještě kontrola platnosti modelu.

```
if (!ModelState.IsValid)  
    return View();
```

Při úspěšném splnění této podmínky je již možné objednávku vymazat z databáze.

```
_db.DeleteObject(zrusit);  
_db.SaveChanges();  
return RedirectToAction("Cancel");
```

V prvním příkazu se označí objekt pro vymazání a ve druhém příkazu je tato změna uložena do databáze. Poté dojde k přesměrování na akční metodu *Cancel*, která obsahuje pouze příkaz na vyvolání pohledu. V tomto pohledu je poté uživatel informován, že jeho objednávka byla úspěšně zrušena. Jakmile není pro daného dárce vedena v databázi žádná objednávka, teprve v této chvíli může dárce zadat jinou objednávku. To je z důvodu, že nelze pro jednoho dárce evidovat v databázi více jak jednu objednávku.

6.8.5 Zobrazení odběrů

Poslední možností, kterou má uživatel k dispozici, je možnost zobrazit všechny své předchozí odběry, včetně data, kdy byly provedeny. Na konci tohoto výpisu je ještě informace o jejich celkovém počtu.

K výpisu všech odběrů je vytvořena akční metoda *Show*, která má parametr ID dárce. Při stisknutí daného požadavku uživatele ve vytvořeném pohledu *Details*, je ID dárce uloženo v parametru a může se použít v dotazu. Celá metoda *Show* pak vypadá takto:

```
public ActionResult Show (int id)  
{  
    var zobraz = (from m in _db.Odbery  
                  where m.ID_darce == id  
                  select m);  
    return View(zobraz);  
}
```

Jak je z dotazu patrné, dojde k načtení všech řádků v tabulce *Odbery*, kde se shoduje ID dárce s ID, které je v parametru této metody. Tato získaná data jsou poté použita ve vytvořeném pohledu.

Aby bylo možné zobrazit všechny provedené odběry, jednoduchým způsobem je využít možnosti cyklu *foreach*. Před použitím příkazu *foreach* je zapotřebí implementovat rozhraní *IEnumerable*. Implementace tohoto rozhraní je velice jednoduchá. Stačí pouze na prvním řádku, před název vloženého modelu, napsat název tohoto rozhraní.

```
@model IEnumerable<DarciKrve.Models.Odbery>
```

V tuto chvíli je již možné použít konstrukce *foreach* pro výběr všech dat, kdy daný dárce absolvoval odběr krve.

```
@foreach (var odbery in Model)
{
    <p>
        @odbery.DatumOdberu.Day.
        @odbery.DatumOdberu.Month.
        @odbery.Datumodberu.Year
    </p>
}
```

Postupným procházením této kolekce, dochází k vypsání všech odběrů, které dárce absolvoval. Pro lepší přehlednost je ještě doplněn příkaz, který zajistí vypsání celkového počtu všech odběrů:

```
<p>
    Celkem odběrů: @Model.Count()
</p>
```

Tento pohled tak slouží dárci pro informace o odběrech a podle těchto informací si může zvolit i vhodné datum další odběru, na který se objedná.

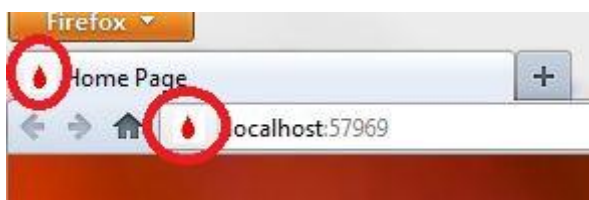
7 ZÁVĚREČNÉ ÚPRAVY

Po provedení všech kroků, které jsou od aplikace požadovány, jsou na závěr vytvořeny ještě grafické úpravy a také možnost pro uživatele zobrazit data v obráceném pořadí než je výchozí nastavení. V grafických úpravách se jedná hlavně o přidání ikony do adresního řádku webového prohlížeče a doplnění obrázků a odkazů na zobrazované stránky.

Pro lepší přehlednost při práci s obrázky, které jsou součástí vytvářených stránek, je vytvořena složka *Images* ve složce *Content*. Díky tomu jsou všechny obrázky uloženy na jednom místě a při jejich použití v kódu stačí poté zadávat vždy stejné parametry, pouze s rozdílným názvem pro konkrétní obrázek. Jako příklad lze uvést způsob, jakým je vytvořena ikona webové stránky. Do složky *Images* je vložen obrázek *headicon.ico* a do souboru *_Layout.cshtml* je do hlavičky (head) doplněn následující příkaz:

```
<link rel="shortcut icon" href="/content/images/  
headicon.ico"/>
```

Vzhledem k tomu, že je tento příkaz vložen do souboru *_Layout.cshtml*, je tato ikona zobrazena vždy, bez ohledu na to, který pohled aplikace je zrovna zobrazen.



Obr. 17. Zobrazení ikony

Jak je z obrázku patrné, je ikona zobrazena nejen v adresním řádku, ale i v titulku spolu s názvem stránky.

Jako pozadí stránky je doplněn obrázek, který je nastaven v souboru *Site.css*. Pomocí příkazu *background-image* do prvku *body* je zajištěno, že toto pozadí bude stejně jako ikona, zobrazeno při použití jakéhokoliv pohledu.

Pohledy, které jsou vytvořené pro jednotlivé controllery, mají v souboru *_Layout.cshtml* příkaz *@RenderBody()*. Pohledy jsou tak umístěny do stránky tam, kde dojde k volání tohoto příkazu. Vedle těchto pohledů je vždy zobrazen obrázek. Obrázek je umístěn vždy úplně na pravé straně. To je zajištěno tím, že příkaz *@RenderBody()* je roztažen na 100% a na obrázek zůstane pouze místo, které je dáno velikostí obrázku.

Do projektu je doplněn i obrázek, který slouží jako aktivní odkaz na stránky projektu *Ted' je čas darovat...vlastní krev a naději*. Tento obrázek se načítá přímo z daných stránek příkazem:

```
<a href="http://www.darujkrev.cz">  
</a>
```

Po kliknutí na tento obrázek jsou uživateli zobrazeny stránky *www.darujkrev.cz*.

7.1 Opačné seřazení dat

Dalším způsobem jak zpříjemnit uživatelům zobrazení požadovaných informací je možnost zobrazit data provedených odběrů v obráceném pořadí, než je výchozí nastavení. To je praktické hlavně pro uživatele, kteří již mají více odběrů. Ve výchozím zobrazení jsou data seřazena tak, jak jsou vložena do databáze, tzn. od nejstarších po nejnovější. K zajištění těchto požadavků jsou zapotřebí upravit soubory *Show.chtml* a *DarceController.cs*.

V souboru *Show.chtml* je pouze vytvořen nový link:

```
@Html.ActionLink("Seřadit data obráceně", "Show",  
new { sortOrder=ViewBag.DateSortParm})
```

V tomto příkazu je první parametr text, který je uživatelům zobrazen, ve druhém parametru je určen pohled, který je po klepnutí na tento link zobrazen (v tomto případě jde o znovunačtení stejného pohledu) a třetí parametr určí proměnnou, která je načtena v akční metodě, která je spojená s pohledem *Show*. Více úprav již v tomto pohledu není zapotřebí, všechny další úpravy jsou provedeny v dané akční metodě.

V akční metodě *Show* v souboru *DarceController.cs* je jako druhý parametr přidán řetězec *sortOrder*. Tento parametr je v dané metodě využit ve dvou případech.

V prvním případě určuje, v jakém pořadí jsou data zobrazena. K tomu je použit ternární operátor, který nahrazuje zdlouhavější způsob pomocí podmínky *if*.

```
ViewBag.DateSortParm = String.IsNullOrEmpty(sortOrder)  
? "Date desc" : "";
```

Tímto jednoduchým způsobem dojde vždy k vybrání druhé možnosti seřazení dat, než té, která je zrovna zobrazena. Je to zařízeno podmínkovým příkazem *IsNullOrEmpty*,

který z pohledu načte řetězec *sortOrder*. V případě, že je řetězec prázdný (výchozí nastavení), je podmínka splněna a za otazníkem je tak vybrána první možnost. V případě, že podmínka splněna není, dojde k vybrání druhé možnosti. Tyto možnosti jsou mezi sebou rozděleny dvojtečkou. Tím je do proměnné *sortOrder* uložen požadovaný řetězec a jeho zpracování je provedeno v příkazu *switch*.

Příkaz *switch* je druhý případ, kdy je využit parametr *sortOrder*.

```
switch (sortOrder)
{
    case "Date desc":
        zobraz=zobraz.OrderByDescending(m=>m.DatumOdberu);
        break;
    default:
        zobraz=zobraz.OrderBy(m=>m.DatumOdberu);
        break;
}
```

Příkaz *switch* slouží jako přepínač mezi jednotlivými možnostmi. Pokud tedy byla do řetězce uložena hodnota *Date desc* dojde k seřazení dat v obráceném pořadí, tzn. od nejnovějšího po nejstarší. V druhém případě je řetězec prázdný a dojde tak k zavolání příkazu *default*, který seřadí data od nejstaršího po nejnovější – tak jak jsou data uložena v databázi. V proměnné *zobraz* je uložen přihlášený dárci a dojde tak k zobrazení dat odběrů, která se týkají jen jeho a ne všech odběrů, které jsou uloženy v databázi.

8 ZHODNOCENÍ ŘEŠENÍ

8.1 Webové služby

Využití technologie klient-server je možné více způsoby. Nejjednodušším způsobem je základní dvouvrstvá architektura, kde je činnost rozdělena pouze na dvou místech. Výhodou tohoto řešení je skutečnost, že všechna data jsou na jednom místě a na stejném místě dochází i k jejich zpracování. Na druhém místě (klient) pak dochází pouze k zobrazení výsledků a zaslání požadavků ke zpracování dalších informací.

Další možností je rozšíření dvouvrstvé architektury na třívrstvou, kde dochází k tomu, že data jsou uložena na jednom místě, na jiném místě jsou zpracována a odeslána klientovi. Výhoda tkví v tom, že je logické rozdělení úloh, které jednotlivé části provádí. Nevýhodou ovšem je, pokud je veškerá výpočetní logika na jednom místě, tak při jakékoliv úpravě nebo rozšíření na jiné systémy je potřeba vytvořit novou aplikaci, která tuto činnost zpracuje.

Aby se mohlo využít již dané aplikace například na mobilní telefony je zapotřebí vytvořit nějaké rozhraní, které je pak využito v aplikaci, která se poté stará pouze o to jak získaná data zobrazit. Tohoto lze dosáhnout použitím webových služeb. Webová služba tak slouží jako výpočetní rozhraní, kde je implementována veškerá potřebná logika v konkrétních metodách a která tyto metody poté nabízí klientům. Klienti použitím těchto metod získávají požadované informace. Takovou to webovou službu může využít například aplikace vytvořená v prostředí .NET, tak aplikace, která je vyvinutá pro mobilní telefon.

Ve vytvořené aplikaci je webová služba naprogramovaná ve frameworku WCF, který nabízí firma Microsoft. Výhodou použití tohoto frameworku je skutečnost, že lze projekt vytvořit ve Visual Studiu, stejně jako projekt pro klienta, který využívá architekturu ASP.NET MVC. Tím, že je vytvořena webová služba, je možné vytvořit i dalšího klienta, který ji využívá, například desktopovou aplikaci. Ta pak může využívat také všech operací poskytovaných službou a získávat tak potřebná data. Aby se zabránilo zneužití webové služby a získání dat neoprávněným způsobem je možnost zabezpečit webovou službu několika různými způsoby. Může se jednat o zabezpečení přenosu dat pomocí přenosových protokolů a dále kontrolu přístupu například pomocí certifikátů.

Takto zabezpečenou službu pak může využívat pouze autorizovaný klient, pro kterého jsou ve službě nastavena potřebná oprávnění.

Použití webových služeb je v dnešní době stále častější hlavně ve firmách. Díky otevřenému standardu XML, který webové služby využívají, umožňuje firmám vytvářet aplikace, které jsou provozovány na různých platformách. Tím dochází u firem ke snížení nákladu spojených s vývojem nových aplikací, a to je pro firmy zajímavé řešení. To, že webové služby jsou moderním řešením je vidět i na tom, že nástroje pro vývoj webových služeb vytváří spousta firem, jako například Microsoft, IBM apod.

8.2 ASP.NET MVC

Pokud je již tedy webová služba vytvořená, je zapotřebí ještě vytvořit klientskou část. Pro tvorbu webových stránek je vhodné použít architekturu ASP.NET MVC. V této architektuře jsou jednoduchým způsobem zabudovány možnosti pro tvorbu webových stránek. Jednotlivé komponenty slouží pro logické zpracování daného úkolu. Vzhledem k tomu, že je vytvořena webová služba, tak komponenta model není využita, protože ta je určena pro zpracování dat a místo toho je k projektu připojena reference na požadovanou službu.

Pro práci jsou tak využity komponenty *Controller* a *View*. V komponentě *Controller* jsou volány operace z webové služby a pohledy slouží k zobrazení získaných informací. Díky této jednoduché implementaci je možné se věnovat spíše grafické úpravě, která slouží jako uživatelské rozhraní. Vytváření pohledů, které jsou určeny pro jednotlivé akční metody, je také velice jednoduché. Stačí pouze kliknout pravým tlačítkem myši na danou metodu a vybrat položku *Add View*. Tímto způsobem jsou vytvořeny pohledy, ve kterých se v jazyku HTML vyřeší potřebný vzhled.

Kombinací vytvořením webové služby jako logického rozhraní daného problému a implementace této služby do architektury ASP.NET MVC lze dosáhnout přehledného komplexního řešení, kde další výhodou je i možnost vytvoření dalšího klienta, který bude využívat operací webové služby, a to je důležitým přínosem daného řešení.

ZÁVĚR

V této práci byla popsána technologie klient-server a různé způsoby jejího použití. Byly popsány možnosti, kterých je možné využít při práci s technologií ASP.NET MVC, včetně popisu jednotlivých komponent této technologie a vysvětleny možnosti propojení aplikace s databází a databázemi navzájem. Byly zde popsány principy webových služeb, spolu s možnostmi jejich využití.

Pomocí technologie ASP.NET MVC je možné vytvářet dynamické webové stránky jednoduchým a přehledným způsobem, kde se tvorba aplikace rozkládá do 3 hlavních částí. V každé této části se řeší jiná část úkolu a je tak možné pracovat na každé části samostatně, bez většího ohledu na vývoj ostatních částí. Je tak možné vytvářet aplikace, které si uživatel nemusí instalovat na vlastní počítač, pouze mu stačí se připojit k požadovanému serveru.

Součástí této práce je vytvoření prototypu aplikace pro objednávání dárců na odběr krve. V praktické části jsou postupně vysvětleny jednotlivé kroky, jak se aplikace vytvářela. Od vytvoření databáze dárců krve, webové služby, která slouží jako rozhraní mezi databází a klientskou aplikací, tak souborů v jednotlivých komponentách architektury MVC. V dané aplikaci je možné registrovat nového uživatele, vytvářet pro něj objednávky na požadované datum, měnit své osobní údaje, zobrazovat již dříve provedené odběry a také stornovat vytvořenou objednávku. Pro vyřizování těchto požadavků je využit jazyk LINQ, který je součástí frameworku .NET, a kterého je v aplikaci použito pro vyhledávání potřebných dat v databázi. Pomocí této aplikace jsou tak popsány možnosti praktického využití technologie ASP.NET MVC, které využívají metod nabízených webovou službou.

ZÁVĚR V ANGLIČTINĚ

This thesis describes the technology client-server and various methods of its application. There are options which can be used for the technology ASP.NET MVC, including the description of single components and the analysis of various links of the application with the database and the interlink within different databases. Furthermore, I present some definitions of web services and their applications.

Using the technology ASP.NET MVC it is possible to create dynamic websites in a simple and well-arranged manner, and in which the formation of application consists of three main stage. In each of these stages a different part of a task is dealt with, therefore it is possible to work on each stage separately without special regard to the development of the others. As a result, there are applications which do not need to be installed in the user's computer but can only be linked to a specific server.

The aim of the practical part is to form the prototype application used for an appointment system of blood collection, including the description of single steps of forming the application – developing the database of blood donors, the web service providing the interface between the database and the client application, last but not least, developing the files in each components of MVC.

The application allows to register new users, to make their appointments for a specific date, to change their personal data, to show the history of blood taking and to cancel their appointments. For execution of these commands we use the language LINQ which is a part of framework .NET and which is used for searching the specific records in the database. In addition, the description of this prototype application illustrates the practical functions of the technology ASP.NET MVC which use the methods offered by a web service.

SEZNAM POUŽITÉ LITERATURY

- [1] HUDDLESTON, James a Vidya Vrat AGARWAL. Databáze v C# 2008. Praha: Computer Press, 2009. ISBN 978-80-251-2309-6.
- [2] MACDONALD, Matthew, Adam FREEMAN a Mario SZPUSZTA. ASP.NET 4 a C# 2010 – tvorba dynamických stránek profesionálně, kniha 1. Brno: Zoner Press, 2011. ISBN 978-80-7413-131-8.
- [3] MACDONALD, Matthew, Adam FREEMAN a Mario SZPUSZTA. ASP.NET 4 a C# 2010 – tvorba dynamických stránek profesionálně, kniha 2. Brno: Zoner Press, 2011. ISBN 978-80-7413-145-5.
- [4] PROCHÁZKA, David. CSS a XHTML: tvorba dokonalých WWW stránek krok za krokem. Druhé, aktualizované vydání. Praha: Grada Publishing, a.s., 2011. ISBN 978-80-247-3897-0.
- [5] Tutorials. ASP.net[online].2011[cit. 2012-01-09]. Dostupné z: <http://www.asp.net/mvc/tutorials>
- [6] A Guide to Learning ASP.NET MVC Release Candidate 1. Stephenwalter.com [online]. 2010 [cit. 2012-01-09]. Dostupné z: <http://stephenwalther.com/blog/archive/2009/02/07/chapter-2-building-a-simple-asp.net-mvc-application.aspx>
- [7] Introducing ASP.NET MVC 3. Wwww.ASP.net [online]. 2010 [cit. 2012-01-09]. Dostupné z: <http://weblogs.asp.net/scottgu/archive/2010/07/27/introducing-asp-net-mvc-3-preview-1.aspx>
- [8] Understanding the ASP.NET MVC Execution Process. Wwww.asp.net [online]. 2009 [cit. 2012-02-22]. Dostupné z: <http://www.asp.net/mvc/tutorials/overview/understanding-the-asp-net-mvc-execution-process>
- [9] ASP.NET MVC v praxi od A do Z, 14. díl – Unit testy I. část. Wwww.programujte.com [online]. 2009 [cit. 2012-03-09]. Dostupné z: <http://programujte.com/clanek/2009082700-asp-net-mvc-v-praxi-od-a-do-z-14-dil-unit-testy-i-cast/>
- [10] Využití webových služeb v .NET aplikacích. Wwww.interval.cz [online]. 2002 [cit. 2012-03-12]. Dostupné z: <http://interval.cz/clanky/vyuziti-webovych-sluzeb-v-net-aplikacich/>

- [11] 3. Webové služby. Wwww.vsb.cz [online]. 2007 [cit. 2012-03-09]. Dostupné z: <http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text/ch06s03.html>
- [12] How to: Choose between HTTP POST and HTTP GET requests for ASP.NET AJAX Endpoints. Wwww.microsoft.com [online]. 2012 [cit. 2012-03-09]. Dostupné z: <http://msdn.microsoft.com/en-us/library/bb628610.aspx>
- [13] Understanding WSDL. Wwww.microsoft.com [online]. 2003 [cit. 2012-03-11]. Dostupné z: <http://msdn.microsoft.com/en-us/library/ms996486.aspx>
- [14] Intro to ASP.NET MVC 3. Wwww.asp.net [online]. 2011 [cit. 2012-04-07]. Dostupné z: <http://www.asp.net/mvc/tutorials/getting-started-with-aspnet-mvc3/getting-started-with-mvc3-part1-cs>
- [15] How to: Verify That Strings Are in Valid E-Mail Format. Wwww.microsoft.com [online]. 2011 [cit. 2012-04-13]. Dostupné z: <http://msdn.microsoft.com/en-us/library/01escwtf.aspx>
- [16] Transport Security with Certificate Authentication. Wwww.microsoft.com [online]. 2012 [cit. 2012-05-09]. Dostupné z: <http://msdn.microsoft.com/en-us/library/ms731074%28v=vs.90%29.aspx>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ASP	Active Server Pages
CSS	Cascading Style Sheets
LINQ	Language Integrated Query
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery, and Integration
WCF	Windows Communication Foundation
WSDL	Web Services Description Language

SEZNAM OBRÁZKŮ

Obr. 1. Třívrstvá architektura klient – server.....	11
Obr. 2. Model MVC [5].....	13
Obr. 3. Grafický model.....	19
Obr. 4. Vytvoření pohledu.....	21
Obr. 5. Elementy SOAP [10].....	27
Obr. 6. Diagram případů užití.....	32
Obr. 7. Sekvenční diagram.....	33
Obr. 8. Nový projekt.....	36
Obr. 9. Nastavení parametrů nové aplikace.....	37
Obr. 10. Upozornění na stejné rodné číslo.....	39
Obr. 11. Diagram databáze dárců.....	40
Obr. 12. Výběr obsahu modelu.....	44
Obr. 13. Výběr databáze a vytvoření entity.....	45
Obr. 14. Výběr tabulek do modelu.....	45
Obr. 15. Způsob ověření klienta [16].....	49
Obr. 16. Okno View.....	53
Obr. 17. Zobrazení ikony.....	61

SEZNAM TABULEK

Tab. 1. Zpracování požadavků [8]	15
Tab. 2. Návrátové metody třídy Controller [5]	18