

Vývoj software pro nutriční péči

Bc. Petr Bálek

Diplomová práce
2013



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2012/2013

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Petr Bálek**
Osobní číslo: **A11281**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Bezpečnostní technologie, systémy a management**
Forma studia: **prezenční**

Téma práce: **Vývoj software pro nutriční péči**

Zásady pro vypracování:

1. Vypracujte literární rešerši na dané téma s důrazem na softwarovou architekturu aplikace (Model-view-controller, Model-view-presenter, Model View ViewModel) a webové služby.
 2. Popište možnosti platformy Windows Runtime a webových služeb na platformě .NET.
 3. Analyzujte daný problém a navrhnete řešení.
 4. Na základě analýzy vypracujte návrh client-server aplikace pro nutriční péči s využitím platformy Windows Runtime a .NET.
 5. Demonstrujte výsledky a formulujte závěr.
-

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. DEWEY, Ben. *Getting started with Metro style apps*. 1st ed. Sebastopol, CA: O'Reilly, c2012, xiii, 92 p. ISBN 14-493-2055-2. BRUMFIELD, Bob. *Developers guide to Microsoft Prism 4: building modular MVVM applications using Windows Presentation Foundation and Microsoft Silverlight*. 1st ed. Redmond, Wash.?: Microsoft, c2011, xviii, 261 p. ISBN 07-356-5610-X.
2. NAGEL, Christian. *Professional C 4 and .Net 4: building modular MVVM applications using Windows Presentation Foundation and Microsoft Silverlight*. 1st ed. Indianapolis, IN: Wiley Pub., c2010, 1474 p. ISBN 04-705-0225-8.
3. WATSON, Ben. *C 4.0: řešení praktických programátorských úloh*. Vyd. 1. Brno: Zoner Press, 2010, 656 s. Encyklopedie Zoner Press. ISBN 978-80-7413-094-6.
4. PETZOLD, Charles. *Mistrovství ve Windows Presentation Foundation: řešení praktických programátorských úloh*. Vyd. 1. Brno: Computer Press, 2008, 928 s. Encyklopedie Zoner Press. ISBN 978-80-251-2141-2.
5. NASH, Trey. *C 2010: rychlý průvodce novinkami a nejlepšími postupy*. Vyd. 1. Brno: Computer Press, 2010, 624 s. Encyklopedie Zoner Press. ISBN 978-80-251-3034-6.

Vedoucí diplomové práce:

Ing. Erik Král

Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce:

8. února 2013

Termín odevzdání diplomové práce:

3. června 2013

Ve Zlíně dne 8. února 2013

prof. Ing. Vladimír Vašek, CSc.
děkan



doc. RNDr. Vojtěch Křesálek, CSc.
ředitel ústavu

ABSTRAKT

Diplomová práce pojednává o vývoji software pro nutriční péči. V této práci je vytvořen, na základě analýzy, komplexní návrh klient / server systému. Technologie, jež figurují v návrhu, jsou WCF, Windows Azure a Windows Store aplikace. Dále je v rámci této práce vytvořena Windows Store aplikace, jež je připravená pro komunikaci se serverem pomocí využití WCF. V této práci je kladen důraz i na prezentační vzory MVC, MVP a MVVM. Vzor MVVM je implementován v rámci klientské Windows Store aplikace. Výsledná klientská aplikace je programována pomocí jazyků C# a XAML.

Klíčová slova: Windows Store aplikace, MVC, MVP, MVVM, architektonické vzory, Windows Azure, WCF, software pro nutriční péči, C#, XAML, XML

ABSTRACT

This thesis discusses about the development of software for nutritional care. In this thesis is created, based on the analysis, a comprehensive proposal for a client / server system. Technologies that feature in the proposal are WCF, Windows Azure and Windows Store application. Furthermore, in this work, was created Windows Store application, which is ready to communicate with the server through the use of WCF. In this work, the emphasis is on presentation patterns MVC, MVP and MVVM. The MVVM pattern is implemented in the client Windows Store application. The resulting client application is programmed using C # and XAML.

Keywords: Windows Store application, MVC, MVP, MVVM, architectural patterns, Windows Azure, WCF, Software for Nutrical Care, C#, XAML, XML

V první řadě patří díky panu Ing. et Ing. Eriku Královi, který se mne ujal a stal se vedoucím této práce. Díky němu jsem se totiž mohl věnovat v rámci práce informačním technologiím, byť obor, jenž studuji, není příliš zaměřen na IT. V této souvislosti patří rovněž díky i panu docentu Adámkovi, jenž schválil toto téma.

Velké díky patří i paní Pazourkové, jež mi velice pomohla s jazykovou korekturou při psaní práce, čímž snad i vylepšila mou mateřštinu do dalších dní.

Dalšími osobami, které si zaslouží poděkování, jsou všichni moji rodičové, kteří dotovali má studia a poskytovali mi morální podporu.

Poděkování také zasluhuje milá dívka Nela, jež mi byla oporou v časech psaní práce i předtím.

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 do IS/STAG jsou totožné.

Ve Zlíně

.....

podpis diplomanta

OBSAH

ÚVOD.....	11
I TEORETICKÁ ČÁST.....	12
1 NUTRIČNÍ PÉČE.....	13
1.1 PODVÝŽIVA.....	13
2 ARCHITEKTURY APLIKACÍ.....	14
2.1 MVC.....	14
2.1.1 Model.....	15
2.1.2 View.....	15
2.1.3 Controller.....	15
2.1.4 Funkcionalita a použití MVC.....	16
2.2 MVP.....	16
2.2.1 Passive View.....	17
2.3 MVVM.....	18
2.3.1 ViewModel.....	18
2.3.2 Binding.....	18
2.3.3 Command.....	18
2.3.4 Interakce v MVVM.....	19
3 WINDOWS RUNTIME.....	20
3.1 MOŽNOSTI PROGRAMOVÁNÍ PRO WINRT.....	21
3.1.1 JavaScript + HTML5 + CSS3.....	21
3.1.2 XAML + C# 5.0.....	21
3.2 PRÁCE SE SOUBOROVÝM SYSTÉMEM.....	21
3.2.1 Složka stažených souborů.....	22
3.2.2 Složka instalace.....	22
3.2.3 Složky lokální, dočasná a roaming.....	23
3.3 ŽIVOTNÍ CYKLUS.....	23
3.3.1 Zajištění persistence.....	24
3.4 NAVIGACE.....	25
3.5 NOTIFIKACE A DLAŽDICE.....	26
3.5.1 Plánované notifikace dlaždic.....	26
3.5.2 Toasty.....	27
3.5.3 Badge.....	28
3.6 KONTRAKTY.....	29
3.6.1 Vyhledávání.....	30
3.6.2 Sdílení.....	30
3.6.3 Nastavení.....	32
3.7 SPECIFICKÉ KOMPONENTY PRO WINDOWS STORE APLIKACE.....	33
3.7.1 AppBar.....	34

3.7.2	SemanticZoom.....	34
3.8	PCL.....	36
3.9	PUBLIKOVÁNÍ APLIKACÍ VE WINDOWS STORE.....	38
3.9.1	Certifikace.....	38
4	WEBOVÉ SLUŽBY.....	42
4.1	REST.....	42
4.1.1	Vytváření dat.....	42
4.1.2	Získávání dat.....	43
4.1.3	Úprava dat.....	43
4.1.4	Odstraňování dat.....	44
4.2	SOAP WEBOVÉ SLUŽBY.....	44
4.2.1	WSDL protokol.....	44
4.2.2	SOAP protokol.....	45
4.2.3	UDDI protokol.....	45
4.3	WCF.....	46
4.3.1	Principy WCF.....	46
4.3.2	Hosting WCF služeb.....	46
4.3.3	Výhody WCF.....	47
4.3.4	Podpora WCF ve Windows Store.....	48
4.3.5	Konfigurace klienta WCF ve Windows Store aplikaci.....	49
5	WINDOWS AZURE.....	50
5.1	DATOVÁ CENTRA.....	50
5.2	EXEKUČNÍ MODEL.....	50
5.2.1	Virtual Machines.....	51
5.2.2	Web Sites.....	51
5.2.3	Cloud Services.....	51
5.3	SPRÁVA DAT.....	52
5.3.1	SQL Database.....	52
5.3.2	Tables.....	52
5.3.3	Blobs.....	52
5.4	SÍŤOVÉ SLUŽBY.....	53
5.4.1	Virtual Network.....	53
5.4.2	Traffic Manager.....	53
5.5	ANALÝZY.....	53
5.5.1	SQL Reporting.....	53
5.5.2	Hadoop.....	53
5.6	MESSAGING.....	54
5.6.1	Fronty.....	54
5.6.2	Service Bus.....	54
5.7	CACHOVÁNÍ.....	54

5.7.1	Caching.....	55
5.7.2	CDN.....	55
5.8	IDENTITA UŽIVATELE.....	55
5.8.1	Windows Azure Active Directory.....	55
5.9	HPC.....	55
5.10	MEDIA SERVICES.....	56
II	PRAKTICKÁ ČÁST.....	57
6	ANALÝZA.....	57
6.1	STÁVAJÍCÍ IS.....	57
6.1.1	SWOT analýza.....	58
6.2	PLÁNOVANÝ IS.....	58
6.2.1	Subjekty využívající systém.....	59
6.2.2	Shromáždění dat.....	59
6.2.3	Screening.....	59
6.2.4	Edukace.....	60
6.2.5	Upozorňování.....	60
6.2.6	Virtuální nutriční terapeut.....	60
6.3	PŘÍPADY UŽITÍ	60
6.4	POTŘEBNÉ ZDROJE K REALIZACI.....	62
6.4.1	Software.....	62
6.4.2	Hardware.....	63
6.4.3	Vývojový tým.....	63
6.4.4	Čas.....	63
6.4.5	Finance	63
7	NÁVRH.....	65
7.1	WCF SLUŽBY.....	65
7.1.1	Generické CRUD rozhraní.....	65
7.1.2	Generická abstraktní CRUD třída.....	65
7.1.3	Konkrétní CRUD služba.....	66
7.1.4	Konfigurace WCF.....	67
7.2	ZABEZPEČENÍ DAT.....	67
7.2.1	Řízení obsahu.....	69
7.3	ENTITY JÁDRA.....	70
7.4	NASAZENÍ KLIENT / SERVER	72
7.4.1	Vlastní server.....	72
7.4.2	Windows Azure.....	73
8	KLIENŤSKÁ APLIKACE.....	75
8.1	STRUKTURA.....	75
8.1.1	Složka Assets.....	75
8.1.2	Složka Behaviors.....	75

8.1.3	Složka Commands.....	75
8.1.4	Složka ViewModels.....	75
8.1.5	Složka Views.....	75
8.1.6	Složka Controls.....	76
8.1.7	Složka Services.....	76
8.1.8	Složka Styles.....	76
8.1.9	Složka Locators.....	76
8.2	SERVISNÍ VRSTVA.....	76
8.2.1	Vzdálené služby.....	77
8.2.2	CRUD služby.....	78
8.2.3	Třída RemoteCrudService<T>	78
8.2.4	Lokální služby.....	79
8.3	VYBRANÉ ČÁSTI APLIKACE.....	80
8.3.1	Třída ViewModelLocator.....	80
8.3.2	Třída BasicViewModel.....	80
8.3.3	Metoda InitCommands třídy CRUDViewModel.....	81
8.3.4	Command.....	82
8.3.5	HeaderControl.....	83
8.3.6	Třída GroupedItem.....	84
8.3.7	Metoda NavigateByUser třídy Navigator.....	84
8.3.8	Vlastní chování.....	85
8.3.9	App.xaml.....	86
8.3.10	Nastavení datové kontextu.....	86
8.4	VYBRANÉ SNÍMKY OBRAZOVKY.....	87
8.5	IKONY POUŽITÉ V APLIKACI.....	88
8.6	ZJIŠTĚNÉ PROBLÉMY.....	89
9	DALŠÍ VÝVOJ.....	89
	ZÁVĚR.....	90
	ZÁVĚR V ANGLIČTINĚ.....	91
	SEZNAM POUŽITÝCH ZDROJŮ.....	92
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	98
	SEZNAM OBRÁZKŮ.....	101
	SEZNAM TABULEK.....	102

ÚVOD

Tato diplomová práce pojednává o vybraných možnostech vývoje klient / server aplikací pomocí technologií společnosti Microsoft. Na straně klienta to je Windows Store aplikace, na straně serveru WCF a Windows Azure. WCF v tomto případě slouží jako pojítko mezi klientem a cloudem. V práci je rovněž kladen důraz i na standardizovaný vývoj pomocí architektonických prezentačních vzorů MVC, MVP, MVVM.

Důvodem výběru tohoto tématu, je můj zájem o moderní technologie ve světě softwarových systémů.

Cílem této práce je vytvoření návrhu aplikace, která umožní klient / server komunikaci. Pro demonstrativní účely návrhu, je vytvořena klientské aplikace, která je připravena na komunikaci se serverem pomocí WCF.

V teoretické části práce je kladen důraz na popsání možností vývoje klientských aplikací pro Windows Store, webové služby a Windows Azure. Vedle světa praktických technologií, je část této práce věnována prezentačním architektonickým vzorům, kde jsou popsány vzory MVC, MVP a MVVM. Popisování těchto teoretických vzorů se nese ryze v duchu teorie, jelikož tyto vzory nejsou výsadní záležitostí, které je možné přiřadit pouze společnosti Microsoft. Tyto vzory totiž mohou být v praxi aplikovány v jakékoliv technologii. Tato práce v sobě rovněž nese i kapitolu s vysvětlením, co se skrývá pod pojmem „nutriční péče“.

V praktické části je zpracována analýza požadavků a návrh, jež ukazuje, jakým způsobem je možné propojit klientské aplikace se serverem pomocí WCF. Dále v rámci této části je vytvořena demonstrativní klientská aplikace, která ilustruje vybrané možnosti software pro nutriční péči. U této aplikace byl rovněž kladen důraz na návrh podle vzoru MVVM, který byl vybrán z výběru MVC, MVP a MVVM.

Z důvodu zaměření práce na vývoj software, obsahuje práce rovněž i nezbytné úryvky kódu, které ukazují možné praktické řešení problematiky. Tyto kódy jsou psány v jazycích C#, XML, XAML či JSON.

I. TEORETICKÁ ČÁST

1 NUTRIČNÍ PÉČE

Nutriční péče je činnost, kterou zajišťují nutriční terapeuti ¹ a nutriční asistenti ². Náplní práce těchto pracovníků je zajišťování správného vyživování jim svěřených pacientů a zároveň i vyhodnocování zdravotních rizik, která vyplývají ze stavu, ve kterém se nalézá vyživovaný, jemuž je péče určena. [1]

V současném českém zdravotnictví a sociální péči není tomuto problému věnována přílišná pozornost. Počet podvyživených lidí v České republice dosahuje dle průzkumu 40 % v nemocnicích a 60 % v domácí péči. V Evropě se potýká s podvýživou přibližně 33 miliónů lidí. V zemích třetího světa je však stav ještě více alarmující, neboť podle statistik každou minutu zemře v důsledku podvýživy 5 dětí, což je 7200 dětí denně. [2] [3]

Vrátíme-li se zpět do České republiky, tak na jedné straně, při nevyužívání potenciálu nutričních specialistů, zdravotnická zařízení ušetří přímé mzdové náklady pro nutriční terapeutů a asistentů, ale na straně druhé trátí na dlouhodobých výdajích za náklady spojené s prodlouženým pobytem ve zdravotnickém zařízení. Problémy způsobené špatnou nutriční péčí stojí státní pokladnu České republiky ročně cca 66 miliard Kč. V rámci Evropy tato suma činí cca 170 miliard eur. Toto je však pouze ekonomická strana problému. Důležitější je skutečnost, že dochází ke zhoršování zdravotního stavu pacientů. Tento zhoršený stav následně komplikuje léčení původního problému a přináší nové zbytečné komplikace způsobující fyzické, ale i psychické problémy, jakými mohou být například deprese. Těmto komplikacím je možné se vyhnout vhodným nutričním režimem. [2] [4] [5]

1.1 Podvýživa

Špatný stravovací režim a nevhodné složení stravy může vést až k podvýživě organismu. Do podvyživeného stavu se organismus totiž dostává, když postrádá požadovaný přísun důležitých živin a potřebných prvků, které jsou nezbytné pro jeho správné fungování. Mezi tyto základní živiny a důležité prvky lze zařadit sacharidy, tuky, bílkoviny, ale i vitamíny, stopové prvky a jiné další nepostradatelné prvky. Požadované množství těchto živin je u každého jedince individuální. Důležité je na tomto místě poznamenat, že velice štíhlý člověk nemusí trpět podvýživou, a zároveň člověk na první pohled obézní

1 Nutriční terapeut, do roku 2004 nazývaný dietní sestra, je absolvent vysokoškolského bakalářského oboru zaměřeného na nutriční péči. Jako nutriční terapeut může pracovat i absolvent vyšší odborné školy s obdobným zaměřením. Důležité u terapeutů je, že mohou pracovat bez odborného dohledu. [1]

2 Jako nutriční asistent může pracovat absolvent střední zdravotnické školy. Nad prací asistentů musí dohlížet nutriční terapeut. [1]

může být podvyživený. [6]

2 ARCHITEKTURY APLIKACÍ

V následujících podkapitolách jsou popsány často využívané prezentační vzory, které se používají ve světě navrhování aplikací. Všeobecnou snahou těchto vzorů je oddělit data od chování a vzhledu.

2.1 MVC

Architektonický vzor MVC je znám světu již od roku 1979, kdy jej publikoval norský emeritní profesor University of Oslo Trygve Mikkel Heyerdahl Reenskaug, jenž je zachycen na obrázku 1. [7]



Obrázek 1. Trygve Reenskaug – přejato z [8]

Původně se tomuto vzoru říkalo Model-View-Editor. Později byl však přejmenován na Models-Views-Controllers. Roku 1987 byl tento teoretický vzor prvně implementován za použití programovacího jazyka Smalltalk firmou Xerox PARC. [9]

Tento vzor se skládá ze 3 částí:

- **Model** popsáný v podkapitole 2.1.1.
- **View** o kterém pojednává podkapitola 2.1.2.
- **Controller**, který je popsán v podkapitole 2.1.3.

Důležitým faktem je skutečnost, že tento vzor významně ovlivnil softwarový návrh a vzešlo z něj několik podobných návrhových vzorů. O některých je zmínka v následujících kapitolách.

2.1.1 Model

Modelem v níže popisovaných architektonických vzorech se rozumí data a operace nad těmito daty, se kterými se pracuje v rámci aplikace. Tato data mohou být reprezentována jako třídy, struktury nebo pole prvků. Všeobecně je jedno, kde a jak jsou data uložena. Pro lepší pochopení necht' poslouží příklad, kde se pracuje s naměřenými daty (hmotnost pacienta + datum měření) z vážení pacienta. Z těchto dat je možné v rámci modelu vypočítat pochopitelně i jiné hodnoty. Kupříkladu z hodnot se bude moci získávat průměrná hmotnost pacienta v úseku nějakého období, nejvyšší či nejnižší naměřená hmotnost, apod. Modelem se totiž rozumí data (např. výška a hmotnost), ale i operace související s daty. Model v kontextu MVC, který používá pouze data a ne operace, se nazývá „anemický doménový model“ a je považován za tzv. antivzor¹. Do modelu totiž patří i tzv. validační pravidla, která například kontrolují, zda je zadávaná výška osoby vyšší než např. 5 cm nebo zda datum narození rodiče předchází datu narození dítěte apod. [10]

2.1.2 View

Překladem anglického „view“ do češtiny se získá „pohled“. „Pohledem“ je myšleno vykreslení určitých dat modelu v nějakém úhlu uživatelského pohledu. Pro přesnější příkladné vysvětlení je dále uvedeno podrobnější vysvětlení. Různé pohledy mají různé vypovídací hodnoty. Z tohoto důvodu necht' pro lepší představu poslouží model z kapitoly 2.1.1. Jak je možné se dívat na tato data? Tato data mohou být interpretována jako tabulka. Pokud je potřeba zobrazit tatáž naměřená či spočtená data ilustrativněji, je možné použít k reprezentaci hodnot graf. Tento graf může být 2D či 3D. Dále tento graf může být sloupcový, koláčový, spojitý, apod. Jak je vidět, stejná data mohou být prezentována rozmanitými způsoby. [10]

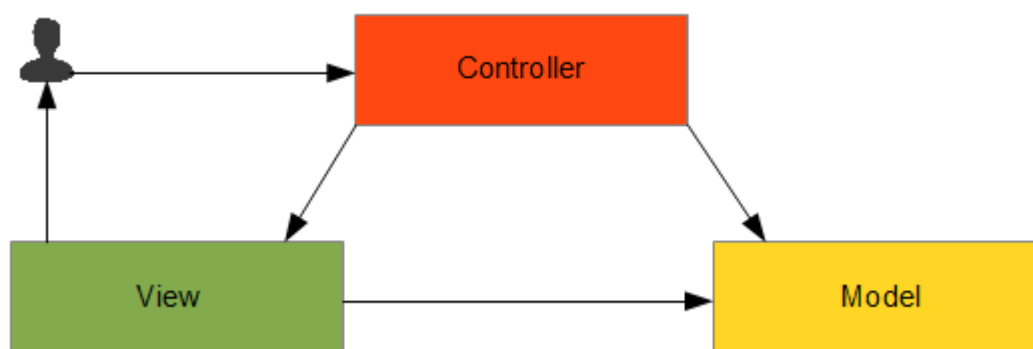
2.1.3 Controller

Tento poslední prvek vzoru MVC slouží jako ústřední. Controller primárně přijímá požadavek a vykonává akce nad Modelem dle vstupních parametrů. Dále také pověřuje View, aby vykreslilo výstup. [10]

¹ Antivzor je špatný návrhový vzor.

2.1.4 Funkcionalita a použití MVC

Vzor MVC je dnes hojně používaný ve webových systémech, kde vstup (HTTP požadavek) obvykle přijímá Controller, který obvykle vykoná akce (načtení / upravení) nad Modelem a pověří příslušné View vykreslením výsledku a odesláním výstupu (HTTP odpověď). Ovšem může nastat i scénář, kdy Controller neupravuje Model, ale určí konkrétní View pro vykreslení. V klientských desktopových aplikacích se využívá spíše vzor MVP popisovaný v kapitole 2.2, u kterého požadavek přijímá View, ne Controller. Celou interakci uživatele s MVC systémem charakterizuje obrázek 3, který ukazuje, že Controller ví o View i Modelu, View ví pouze o Modelu, ale Model nemá vazbu na žádný prvek. [11]

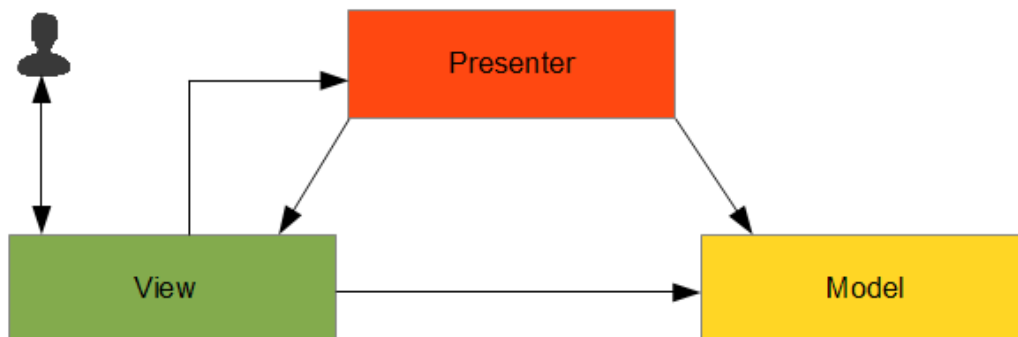


Obrázek 2: Interakce uživatele s MVC systémem (překresleno z [11])

2.2 MVP

Jak již bylo psáno v kapitole 2.1, ze vzoru MVC vzešly další architektonické vzory a vzor Model-view-presenter je právě jeden z nich. Tento vzor je používaný v systémech, kdy prvky (textová pole, tlačítka, zaškrtačací pole, apod.) View přijímají požadavky a zároveň je i vykreslují, což je případ mnohých desktopových aplikací. Implementace tohoto vzoru není však omezena pouze na desktopové použití, jelikož existují i technologie (např. ASP. NET Web Forms), které umožňují lehce implementovat tento vzor. Obrázek 4 demonstruje způsob, jakým probíhá komunikace při interakci s uživatelem. Na tomto obrázku je vidět, že uživatel vykoná akci nad View, které tuto událost zachytí. View může ihned odpovědět nebo může požádat (což je vhodnější) Presenter, aby vykonal sofistikovanější akci aplikační logiky (např. ukládání Modelu do databáze). Praktičtější je však scénář, kdy View bude využívat aplikační logiku Presenteru a součástí View budou pouze záležitosti, které přímo souvisí s uživatelským rozhraním. Presenter může rovněž vybrat

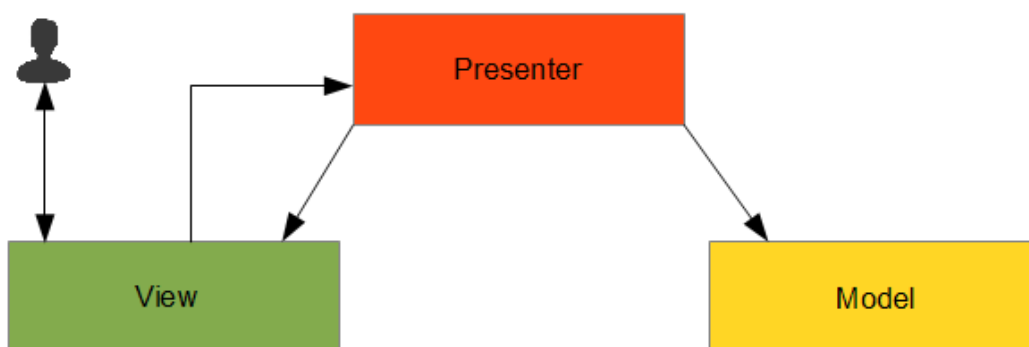
View, které se vykreslí. Pověřované View může být stejné jako to, co přijímalo vstup, nebo jiné, což je např. případ otevření jiného okna po kliknutí na tlačítko. [11]



Obrázek 3: Interakce uživatele s MVP systémem (překresleno z [11])

2.2.1 Passive View

Varianta vzoru MVP, které se říká Passive View, vypouští úzkou vazbu z View na Model, což může být výhoda, jelikož všechny vykonávané operace nad Modelem musí procházet přes Presenter. Zároveň se může toto řešení jevit jako nevýhoda, jelikož deleguje větší zodpovědnost na Presenter (synchronizace Model ↔ View), jak ilustruje obrázek 4.



Obrázek 4: MVP variace Passive View (překresleno z [11])

2.3 MVVM

Tento prezentační vzor je podobný MVP. V tomto vzoru však nevystupuje Presenter, ale ViewModel. Pomocí tohoto vzoru se minimalizuje code-behind jednotlivých View.

2.3.1 ViewModel

ViewModel je část MVVM, která drží Model ve stejném stavu, v jakém je zobrazen ve View. K tomu, aby bylo synchronizováno View a ViewModel, se používá Binding, o kterém je zmínka v kapitole 2.3.1. ViewModel musí být rovněž informován o událostech nad View (např. stisknutí tlačítka, odškrtnutí zatržítka, apod.). Tyto informace jsou zpracovány v prvcích Command (kapitola 2.3.3), které jsou rovněž uloženy ve ViewModel.

2.3.2 Binding

Binding nabízí způsob¹, jak synchronizovat data z ViewModelu s View. Toto vázání dat může být obousměrné či jednosměrné ve směru ViewModel → View. Při jednosměrném bindingu se automaticky předávají data z ViewModelu do View. Oproti tomu při obousměrném bindingu je umožněno informovat i ViewModel z View například o změně textu, který je svázan s datovým kontextem ViewModelu, nebo o událostech (kliknutí, změna textu, přejetí kurzoru myši, apod.) vyvolaných nad uživatelským rozhraním.

2.3.3 Command

Prvky Command suplují code-behind událostí View. V důsledku to vypadá tak, že code-behind jednotlivých View je prázdný, jelikož většina operací se provádí v prvcích Command, které jsou součástí ViewModel. Tohoto navázání událostí na Command se může prakticky využít při nasazení kódu na různé technologie, kde jednotlivé technologie se obvykle liší pouze možnostmi View (např. mají jiné události), neboť mají podporu pro Model i ViewModel stejnou. V tomto případě je možné sdílet části Model a ViewModel, včetně jednotlivých prvků Command. K zajištění přenositelnosti kódu mezi technologiemi společnosti Microsoft se používá tzv. PCL knihoven, kterým se věnuje kapitola 3.8.

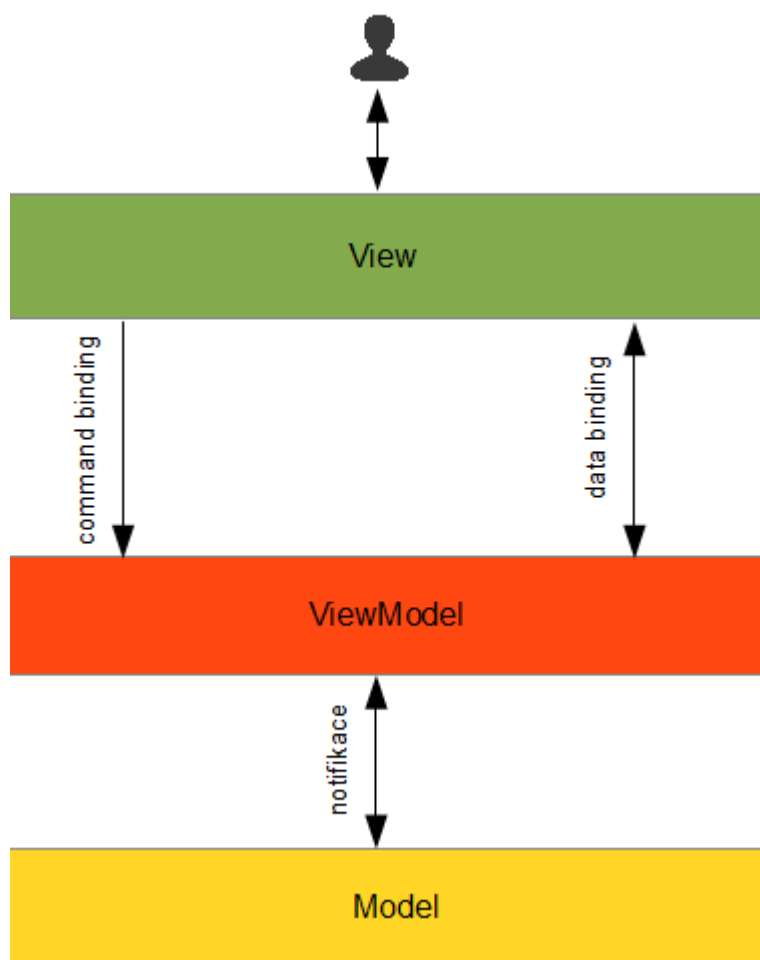
¹ Binding je obvykle automatizovaný, o jeho realizaci se obvykle stará nějaký framework.

2.3.4 Interakce v MVVM

Aby bylo zřejmější vysvětlení fungování MVVM, je uveden obrázek 5 a následující vysvětlující komentář. Spustí-li uživatel MVVM aplikaci, bude mu zobrazeno nějaké View. Předtím, než je View vykresleno, mu musí být nastaven datový kontext (View-Model). Z tohoto datového kontextu jsou následně předány hodnoty (Model) do View k vykreslení.

Pokud uživatel změní hodnoty prvku GUI, např. vyplní textové pole, může být v případě nastaveného obousměrného bindingu informován ViewModel o tom, že má změnit Model, jelikož se změnilo View.

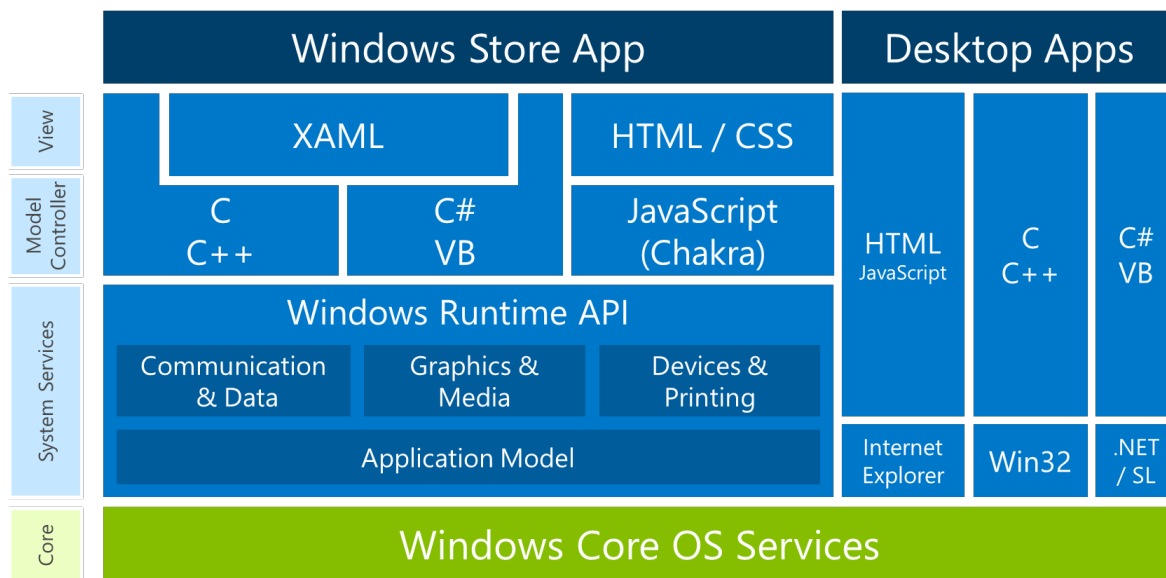
Podobným způsobem, jako nabízí datový binding, je možné informovat ViewModel i o jiných událostech (např. stisknutí tlačítka). K tomu se ale používá prvků Command.



Obrázek. 5: MVVM (překresleno z [12])

3 WINDOWS RUNTIME

Windows Runtime (WinRT) je aplikační architektura, nad kterou mohou být programovány aplikace v operačních systémech Windows 8 a Windows RT, jak ukazuje obrázek 6. K programování aplikací mohou být použity různé programovací jazyky. Těmi-to jazyky jsou standardně C, C++, C#, Visual Basic, ale i JavaScript, sloužící obvykle k programování klientské strany webových stránek. K tvorbě pohledové vrstvy může být použit jazyk XAML známý z WPF. Dále je možné využít HTML a CSS. Aplikace určené pro WinRT je možné spouštět na architekturách procesorů x86, ale i ARM. K šíření aplikací mezi veřejnost slouží Windows Store, což je obchod, kde je možné kupovat, zdarma stahovat, nabízet zdarma, ale i prodávat aplikace. Aplikace pro Windows Store byly původně nazývány „Metro aplikace“, ale později byly přejmenovány na Windows Store aplikace. Důvodem k přejmenování bylo zjištění, že název Metro již vlastnila jistá německá společnost. Aby nebylo jazykových zmatení málo, tak Windows RT je operační systém běžící na mobilních zařízeních s ARM procesory. Windows RT proto není zkratkou Windows Runtime. [13] [14] [15]



Obrázek 6: Začlenění WinRT ve Windows [16]

3.1 Možnosti programování pro WinRT

3.1.1 JavaScript + HTML5 + CSS3

Jedna z možností, jak vytvářet Windows Store aplikace, je založena na technologiích z webového světa. Těmito technologiemi jsou HTML5, JavaScript a CSS3. HTML5 slouží jako deklarativní jazyk grafického uživatelského rozhraní. Pro programování logiky se využívá JavaScript. Aby byl vývoj co nejvíce podobný vývoji webových stránek, ke stylování pohledů může posloužit CSS3.

Oproti vývoji pro IE10 ve standardním režimu existují ovšem drobné rozdíly pramenící například z bezpečnostního modelu. [17]

Tuto novou variantu vývoje aplikací na klientské straně tedy pravděpodobně využijí a ocení mnozí weboví vývojáři, kteří se v podstatě budou muset naučit minimum k tomu, aby mohli vyvíjet aplikace.

3.1.2 XAML + C# 5.0

Další možností tvorby pohledové vrstvy Windows Store aplikací, je varianta s použitím deklarativního jazyka XAML. Tento deklarativní jazyk je založen na standardu XML. První verze jazyka byla zveřejněna v roce 2006. Hlavní použití jazyka XAML je v dnešní době cíleno na technologie Silverlight, WPF a Windows Store aplikace, kde slouží k návrhu GUI. Dalším místem, kde se využívá jazyka XAML je v rámci technologie WF. Na tomto místě je však použit ke konfiguraci aplikace a definici procesů. Jako code-behind pro deklaraci GUI je použit C# 5.0, který přichází s novinkou ulehčující a zpřehledňující práci s asynchronními operacemi. Pro tuto diplomovou práci jsem vybral použití těchto dvou technologií. [18] [19]

3.2 Práce se souborovým systémem

Z důvodu ochrany uživatelů Windows Store mají Windows Store aplikace omezený přístup do souborového systému, kde jsou nainstalovány. Tyto aplikace mohou přistupovat do několika složek systému. Mezi tyto složky patří místo, kam byla aplikace nainstalovaná, složka stažených souborů, roaming složka, lokální složka a složka dočasných souborů. Dále je také možné, aby aplikace měla přístup ke složkám multimediálních knihoven (hudba, obrázky, videa). K tomu je však nutné tyto požadavky uvést v souboru Package.appx-

manifest. [20]

3.2.1 Složka stažených souborů

K vytváření složek a souborů ve složce stažených souborů se používá třída `DownloadsFolder` ze jmenného prostoru `Windows.Storage`. Složky vytváří statická metoda `CreateFolderAsync` a soubory metoda `CreateFileAsync`. V případě, že bude chtít aplikace přistupovat k vytvořeným souborům či složkám i po ukončení opětovném načtení aplikace, musí se přidat položka do tzv. kolekce `FutureAccessList` třídy `StorageApplicationPermissions` ze jmenného prostoru `Windows.Storage.AccessCache`. Následující ukázkový kód ukazuje, jak vytvořit složku, podsložku a soubor ve složce „downloads“. Dále ukazuje způsob, jakým přidat soubor do access listu, aby byl přístupný i po novém spuštění aplikace. Ke konci ukázky je rovněž vidět, jakým způsobem se dají číst data z access listu. [20]

```
//vytvoření složky 'složka A'
StorageFolder slozka_A = await DownloadsFolder.CreateFolderAsync("složka A");

//vytvoření složky 'složka A/podsložka'
StorageFolder podslozka = await slozka_A.CreateFolderAsync("podsložka");

//vytvoření souboru 'složka A/podsložka/soubor1.abc'
StorageFile soubor1 = await podslozka.CreateFileAsync("soubor1.abc");

//získání access listu
StorageItemAccessList accessList =
StorageApplicationPermissions.FutureAccessList;

//zapamatování si souboru1.abc pro budoucí použití
accessList.Add(soubor1);

//získání položek access listu
AccessListEntryView entries =
StorageApplicationPermissions.FutureAccessList.Entries;

//pro všechny položky v access listu
foreach (var entry in entries)
{
    //položka z access listu
    IStorageItem item = await accessList.GetItemAsync(entry.Token);
}
```

3.2.2 Složka instalace

Ze složky instalace aplikace se dá pouze číst. Informace o instalační složce se dá získat z vlastnosti `InstalledLocation` třídy `Package` ze jmenného prostoru `Windows.ApplicationModel`. Aktuální balíček je možné získat ze statické vlastnosti `Current` třídy `Package`. Pro ilustraci toho, jakým způsobem se dá získat instance, pomocí

níž se dá přistupovat do instalační složky, je uveden další úryvek kódu. [20]

```
StorageFolder installFolder = Package.Current.InstalledLocation;  
//získání souboru Diplomka.exe  
StorageFile diplomka = await installFolder.GetFilesAsync("Diplomka.exe");
```

3.2.3 Složky lokální, dočasná a roaming

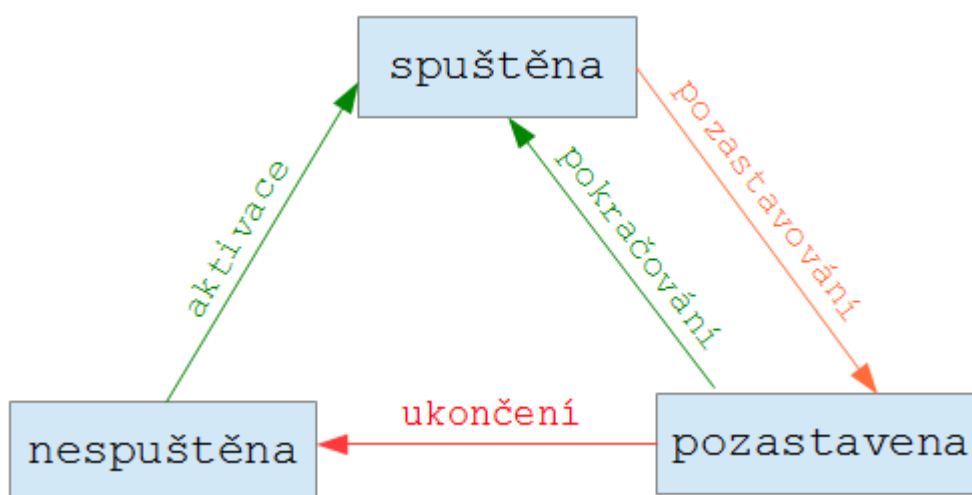
K ukládání dat, která mají být synchronizována mezi všemi zařízeními, kde je aplikace nainstalovaná, slouží úložiště roaming. Oproti tomu pro ukládání dat, která mají být dostupná pouze na zařízení, na kterém byla uložena, se používá lokální úložiště. Jako poslední typ ukládání je možné zvolit dočasnou složku, ze které může systém libovolně mazat data. [20] Níže uvedená ukázka ilustruje, v jakém duchu se nese práce s těmito složkami, jak se vytváří soubory, složky a provádí se operace přejmenování a mazání.

```
//lokální složka  
StorageFolder localFolder = ApplicationData.Current.LocalFolder;  
//vytvoření lokálního souboru  
StorageFile lokalni_soubor = await  
    localFolder.CreateFileAsync("lokalni_sou.xyz");  
//přejmenování souboru  
await lokalni_soubor.RenameAsync("nove jméno");  
//smazání souboru  
await lokalni_soubor.DeleteAsync(StorageDeleteOption.Default);  
  
//roaming složka  
StorageFolder roamingFolder = ApplicationData.Current.RoamingFolder;  
//vytvoření souboru v roaming složce  
StorageFile roaming_file = await  
    roamingFolder.CreateFileAsync("roaming_soub1.xyz");  
  
//dočasná složka  
StorageFolder temporaryFolder = ApplicationData.Current.TemporaryFolder;  
//vytvoření dočasné složky  
StorageFolder temp_subfolder = await  
    temporaryFolder.CreateFolderAsync("subfolder");  
//smazání složky  
await temp_subfolder.DeleteAsync();
```

3.3 Životní cyklus

Windows Store aplikace se může dostat během svého životního cyklu do **3 stavů**, viz. obrázek 7. Prvním, a tedy i startovacím, stavem je stav „**nespuštěna**“, kdy je aplikace nainstalovaná v systému, ale není spuštěná. Z tohoto původního stavu může přejít konečný třístavový automat aplikace pouze do jediného stavu, který je nazván „**spuštěna**“. V tomto stavu aplikace typicky vykonává nějakou činnost (přehrávání muziky, videa, interakce s uživatelem v rámci textového procesoru, apod.). Tento stav je tedy pochopitelně pro uživa-

tele nejdůležitější. Nicméně aplikace se může dostat i do posledního nepopsaného stavu „pozastavena“ přepnutím uživatele do jiné aplikace, přechodem do úsporného režimu či běžným uživatelským ukončením. Pozastavení může být tedy vyvoláno softwarovou, ale i hardwarovou příčinou, např. při nízkém stavu nabití baterie, nedostatku paměti apod. Tento mezistav mezi ukončením a během aplikace je zařazen hlavně z výkonnostních důvodů, jelikož Windows Store aplikace mohou běžet na některých mobilních zařízeních, kde je každý ušetřený watt žádoucí. Pro programátora je důležité smysluplně reagovat na proces pozastavování, jelikož se při něm neví, zda bude moci být aplikace znovu obnovena. Z tohoto hlediska je důležité vždy hledět na persistenci dat, se kterými aplikace pracuje. Vypne-li uživatel aplikaci legitimním způsobem (gesto pro vypnutí či kombinace Alt + F4), je aplikace upozorněna, aby během budoucích **10 sekund** uložila potřebná data. Po oněch 10 sekundách se aplikace vypne, avšak o tom již není informována. Oproti tomu, když se pokračuje v běhu aplikace přechodem do stavu „spuštěna“, aplikace je o tom informována a programátor by v tomto místě měl zajistit načtení relevantních dat, tak aby uživatel nepostřehl, že byla aplikace uspaná. [21]



Obrázek 7: životní cyklus Windows Store aplikace (překresleno z [21])

3.3.1 Zajištění persistence

Jak již bylo zmíněno v kapitole 3.3, měla by být v rámci aplikace zajištěna persistence, kterou by měl zajistit programátor vhodným ukládáním a načítáním dat. K tomuto účelu může použít své vlastní řešení, ale typičtější je využít řešení, které nabízí Visual studio. Toto řešení spočívá v tom, že k řízení persistence je pomocí Visual Studia vygenerová-

na třída `SuspensionManager`. V této třídě je několik metod, mezi nejdůležitější však patří `SaveAsync` a `RestoreAsync`. Tato třída sama o sobě zajišťuje pouze načítání a ukládání dat, ale nezajišťuje provedení operací v potřebný čas. Z tohoto důvodu je nutné, aby se vyvolaly ony metody ve správný moment, a toho může být docíleno uvnitř metod `OnLaunched` a `OnSuspending` v rovněž vygenerované třídě `App`. Tyto dvě metody totiž reagují na potřebné události. Ke zjištění informace, v jakém stavu se aplikace nachází, je předáván jako parametr objekt třídy `LaunchActivatedEventArgs`, pomocí jehož vlastnosti `PreviousExecutionState` je možné zjistit předchozí stav aplikace. Stavů jsou dány výčtovým typem `ApplicationExecutionState`, který může nabývat hodnot `NotRunning`, `Running`, `Suspended`, `Terminated` a `ClosedByUser`. [20] [21]

3.4 Navigace

Pod pojmem navigace se skrývají přechody mezi různými stránkami (pohledy) aplikace a to, jakým způsobem je toho docíleno. Příkladem navigace může být přechod z úvodní stránky do stránky, kde jsou zobrazeny diety. K této navigaci slouží proměnná typu `Frame` ze jmenného prostoru `Windows.UI.Xaml.Controls`, která je obsažena jako vlastnost ve třídě `UserControl` ze stejného jmenného prostoru. K této proměnné je možné se dostat přes stejnojmenný identifikátor `Frame`. Důležitou roli v navigování mezi pohledy hrají metody událostí. Mezi tyto metody patří `OnNavigatingFrom(NavigatingCancelEventArgs)`, `OnNavigatedFrom(NavigationEventArgs)` a `OnNavigatedTo(NavigationEventArgs)` ze třídy `UserControl`. Metoda `OnNavigatingFrom` je volána jako první z uvedených metod při přechodu z View A do View B, kde parametr `NavigatingCancelEventArgs` ze jmenného prostoru `Windows.UI.Xaml.Navigation` v sobě nese informace o navigačním módu ¹ a potřebnou informaci o cílové stránce ². Jako druhá událost je vyvolána `OnNavigatedFrom`, kde instance `NavigatingEventArgs` předaná jako parametr v sobě nese informace jako `NavigatingCancelEventArgs`, ale přidává k tomu ještě další, mezi které patří vlastnost `Content` a `Parameter`. Poslední metodě `OnNavigatedTo` je opět předán parametr jako instance třídy `NavigationEventArgs`. Důležité je podotknout, že první 2 události jsou vyvolány nad stránkou (v code-behind třídě), z které přechází. Poslední 3. metoda je vyvolána již na stránce, na kterou bylo navigováno. Z tohoto vyplývá, že při spuštění aplikace se vykonává pouze poslední z uvedených metod, jelikož ještě neexistuje stránka, z které by

¹ Navigační mód je hodnota výčtového typu `NavigationMode` ze jmenného prostoru

`Windows.UI.Xaml.Navigation`. Tento výčtový typ může nabývat hodnot `New`, `Back`, `Forward` či `Refresh`.

² Informace o cílové stránce je předávána jako `System.Type`.

se přecházelo. Příklad, který ukazuje, jak zadat požadavek na přesměrování, je uveden v níže uvedeném úryvku kódu.

```
//metoda Navigate je instanční, nikoliv statická
Frame.Navigate(typeof(ModulesPage), new { p1 = "první parametr", p2 = 15 });
```

3.5 Notifikace a dlaždice

Ve světě Windows Store aplikací existuje několik specifických způsobů, jakými informovat uživatele. O těchto způsobech je psáno v následujících podkapitolách. [8]

3.5.1 Plánované notifikace dlaždic

Pod pojmem „plánované notifikace dlaždic“ se skrývá mechanismus zobrazování informativního obsahu na dlaždici aplikace v daný čas. U dlaždic je totiž možné plánovat, kdy se na ni obsah zobrazí. Jedna dlaždice může zobrazovat více notifikací, které se za sebou objevují cyklicky. Notifikace se systému předávají v XML formátu. Strukturu ukázkového XML řetězce demonstruje např. následující kód [23]:

```
<tile>
  <visual>
    <binding template="TileWideText09">
      <text id="1">17</text>
      <text id="2">Zprávk</text>
    </binding>
  </visual>
</tile>
```

Microsoft nabízí několik již předvyplněných šablon, které jsou spárované s výčtovým typem `TileTemplateType` ze jmenného prostoru `Windows.UI.Notifications`. Získání příslušné XML struktury se provede pomocí statické metody `GetTemplateContent` třídy `TileUpdateManager` ze stejného jmenného prostoru jako u výčtového typu `TileTemplateType`, jak demonstruje níže uvedený úryvek kódu.

```
//získání patřičné struktury šablony
XmlDocument xml = TileUpdateManager.GetTemplateContent
    (TileTemplateType.TileSquareBlock);
```

Nastavení zobrazovaných hodnot na dlaždici je již práce s XML API, viz. další úryvek kódu:

```
//získání všech elementů text
XmlNodeList elements = xml.GetElementsByTagName("text");

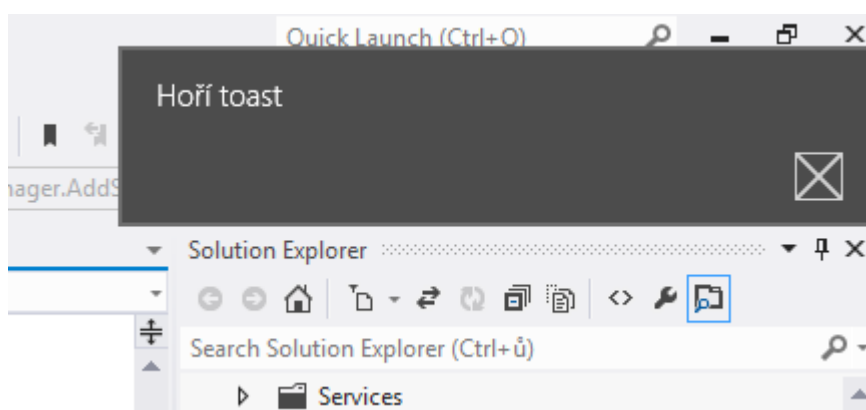
//vlození do 1. nalezeného elementu textu 17
elements[0].AppendChild(xml.CreateTextNode("17"));
```

```
//vložení do 2. nalezeného elementu textu 17  
elements[1].AppendChild(xml.CreateTextNode("zpráva"));
```

Dále je nutné vytvořit instanci třídy `ScheduledTileNotification`. Této instanci je nutné předat v konstruktoru xml strukturu dlaždice a čas, kdy se má dlaždice uživateli zobrazit. K předání instance dlaždice systému se používá instanční metoda `AddToSchedule` třídy `TileUpdater` z výše jmenovaného namespace. K získání instance třídy `TileUpdater` slouží statická metoda `CreateTileUpdaterForApplication` ze třídy `TileUpdateManager`. Příklad procesu vytvoření a předání dlaždice ilustruje níže uvedený kód.

```
//vytvoření instance ScheduledTileNotification  
ScheduledTileNotification tile = new ScheduledTileNotification(xml,  
    DateTime.Now.AddSeconds(1));  
  
//získání updateru a přidání dlaždice do fronty  
TileUpdater tileUpdater = TileUpdateManager.CreateTileUpdaterForApplication();  
tileUpdater.AddToSchedule(tile);
```

3.5.2 Toasty



Obrázek 8: Toast zobrazený v desktopové aplikaci ve Windows 8

Toast notifikace představuje ve Windows Store aplikacích možnost informování uživatele, ať už se nachází kdekoliv v systému. Toasty se totiž zobrazují v prostředí Windows Store aplikací, ale i v klasickém desktopovém prostředí, jak je možné vidět na obrázku 8. K tomu, aby mohla aplikace distribuovat toasty, musí mít uvedeno, že je `ToastCapable`, což znamená, že je atribut `ToastCapable` elementu `VisualElements` nastaven na `true`. Toto nastavení se provádí v souboru `Package.manifest`.

Vyvolání notifikace typu toast se provede obdobným způsobem, jako tomu bylo v kapitole 3.5.1. Práce s tímto oznamovacím prvkem se však nese v mírně odlišném duchu,

např. názvy tříd neobsahují slovo Tile, nýbrž Toast. Další rozdíl mezi těmito prvky spočívá v tom, že toasty nepodporují aktualizace, filosofie práce s těmito prvky zůstává však stejná. Důležitá je třída `ToastNotifier` ze jmenného prostoru `Windows.UI.Notifications`, pomocí které se předávají oznamovací prvky k zobrazení. Instanci této třídy je možné získat pomocí třídy `ToastNotificationManager`. Třída `ToastNotificationManager` rovněž slouží k získávání XML šablony dle výčtového typu. Z důvodu demonstrativnosti možnosti řešení je uveden následující úryvek kódu:

```
XmlDocument xml = ToastNotificationManager.GetTemplateContent
    (ToastTemplateType.ToastText02);

XmlNodeList elements = xml.GetElementsByTagName("text");
elements[0].AppendChild(xml.CreateTextNode("Hoří!"));
elements[1].AppendChild(xml.CreateTextNode("Volejte hasiče!"));

ScheduledToastNotification toast = new ScheduledToastNotification(xml,
    DateTimeOffset.Now.AddSeconds(1));

ToastNotifier notifier = ToastNotificationManager.CreateToastNotifier();
notifier.AddToSchedule(toast);
```

3.5.3 Badge

Badge je prvek, který umožňuje minimalisticky informovat uživatele o stavu aplikace. Tento způsob informování umožňuje zobrazit v rohu dlaždice miniaturní grafiku, jež demonstruje např. zaneprázdněnost aplikace, přijetí zprávy nebo vznik chyby v aplikaci. Dále je možné, aby informace nebyla nesena formou předdefinované grafiky, ale číselnou hodnotou z intervalu 1 – 99. V případě, že je číslo větší než 99, bude zobrazeno jako 99+. [24] Níže uvedený kód ilustruje, jakým způsobem se pracuje s badge prvky. V tomto kódu se pracuje se třídami a výčtovým typem ze jmenného prostoru `Windows.UI.Notifications` a `Windows.Data.Xml.Dom`.

```
//získání xml šablony pro číselný badge nebo grafický
XmlDocument xml = BadgeUpdateManager.GetTemplateContent
    (BadgeTemplateType.BadgeNumber);

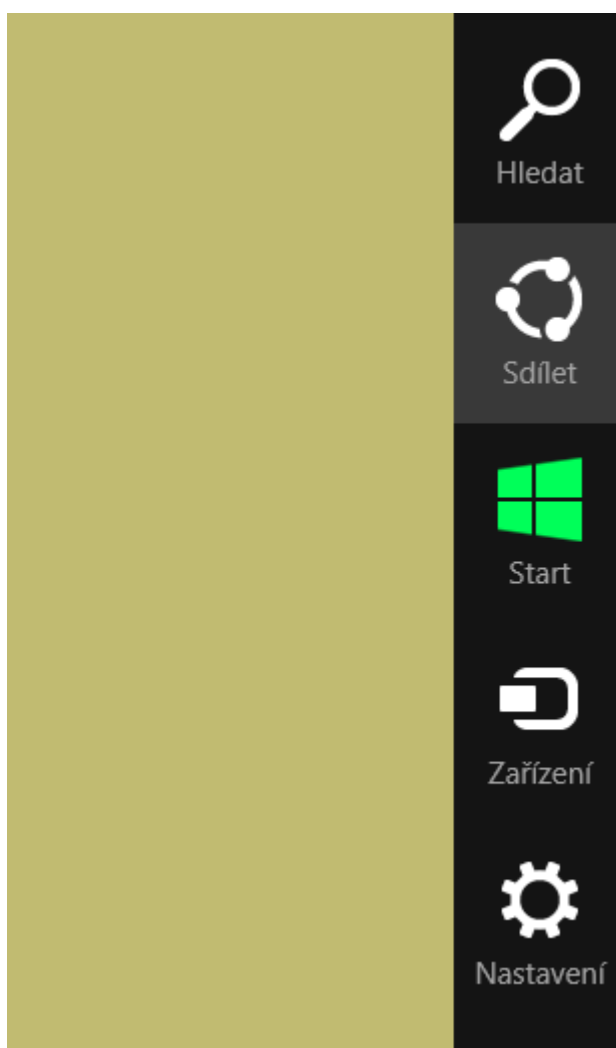
//získání elementu badge
XmlElement badgeElement = (XmlElement)xml.SelectSingleNode("/badge");
//bude zobrazeno číslo 5
badgeElement.SetAttribute("value", 5 + "");

//vytvoření instance badge notifikace
BadgeNotification badge = new BadgeNotification(xml);

//získání updatovacího objektu
BadgeUpdater updater = BadgeUpdateManager.CreateBadgeUpdaterForApplication();
//předání badge k vykreslení
updater.Update(badge);
```

3.6 Kontrakty

Kontrakty jsou mechanismus, který slouží k lepší integraci aplikace do systému s Windows Runtime. Tyto kontrakty rovněž slouží i k zlepšování uživatelského komfortu, jelikož umožňují uživatelům aplikací sdílet data mezi aplikacemi, zahrnovat data aplikace do systémového vyhledávání a jiné další operace. Z uživatelského hlediska je tato integrace realizovaná na úrovni prvku systému, který je nazýván CharmBar. Tento prvek je zobrazen na obrázku 9. Vyvolání se provede např. najetím kurzoru myši do pravého dolního či horního rohu obrazovky. [25]



Obrázek 9: CharmBar

3.6.1 Vyhledávání

Aby bylo možné vyhledávat data z různých aplikací na jednom místě systému s Windows Runtime, musí aplikace, u níž je požadováno vyhledávání, deklarovat tzv. vyhledávací kontrakt. Prvním krokem pro zahrnutí aplikace do vyhledávání operačního systému je uvedení příslušné deklarace na patřičném místě (element Application) v souboru Package.appxmanifest. Potřebný XML fragment má následující podobu:

```
<Extensions>
    <Extension Category="windows.search" />
</Extensions>
```

Dále je potřeba zaregistrovat příslušný handler. Ten se registruje v metodě OnWindowCreated(WindowCreatedEventArgs args) třídy Application. [26] Nalezení výsledků hledání v aplikaci je implementační detail, který zde již není zmiňován. Následující úryvek kódu ilustruje registraci handleru a přesměrování na stránku s vyhledáváním v reakci (uvnitř metody QuerySubmitted) na vyvolanou událost.

```
protected override void OnWindowCreated(WindowCreatedEventArgs args)
{
    base.OnWindowCreated(args);
    SearchPane.GetForCurrentView().QuerySubmitted += QuerySubmitted;
}

void QuerySubmitted(SearchPane sender, SearchPaneQuerySubmittedEventArgs args)
{
    var content = Window.Current.Content;
    var frame = (Frame) content;
    //přesměrování na vyhledávací stránku aplikace
    frame.Navigate(typeof(SearchPage), args.QueryText);
}
```

3.6.2 Sdílení

Sdílení dat je ve Windows Store aplikacích vyřešeno obdobným způsobem jako schránka (ctrl + c → ctrl + v), což znamená, že existuje aplikace, která sdílí (zdroj sdílení) a dále aplikace, která přijímá data (cíl sdílení). K tomu, aby aplikace mohla sdílet, je potřebné, aby měla registrovaný handler, který bude hlídat, zda se uživatel nesnaží sdílet data. V případě, že bude chtít uživatel data sdílet, klikne v liště CharmBar na „Sdílet“, čímž se začne vykonávat kód zavěšený na handleru. [26][27]

Registrace handleru může být provedena například v metodě OnNavigatedTo (viz. kapitola 3.4) a deregistrace v metodě OnNavigatingFrom (viz. kapitola 3.4) konkrétní

stránky. [26] Ukázka kódu, jež demonstruje, jak může vypadat proces registrace, obsluha a odregistrování handleru, je uvedena níže:

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    //registrace handleru
    DataManager.GetForCurrentView().DataRequested += DataRequested;
}

void DataRequested(DataTransferManager sender, DataRequestedEventArgs args)
{
    DataRequest request = args.Request;
    DataPackagePropertySet properties = request.Data.Properties;

    properties.Title = "Titulek sdílející aplikace";
    properties.Description = "Popisek sdílející aplikace";

    request.Data.SetText("Příklad sdíleného textu");
}

protected override void OnNavigatingFrom(NavigatingCancelEventArgs e)
{
    //odregistrování handleru
    DataManager.GetForCurrentView().DataRequested -= DataRequested;
}
```

Výše uvedený kód ukazuje, jakým způsobem sdílet data, což je v podstatě pouze první část řešení. Druhá část (cíl sdílení) kapitoly vysvětluje, jakým způsobem říci systému, že aplikace může přijímat sdílená data. První krok řešení spočívá v překrytí metody `OnShareTargetActivated(ShareTargetActivatedEventArgs args)` třídy `Application` ze jmenného prostoru `Windows.UI.Xaml`. Tato metoda se vyvolá v případě, že se nějaká aplikace snaží sdílet data s aplikací ve které se vykonává kód. Níže je uvedená ilustrativní ukázka naprogramování cíle sdílení:

```
protected async override void OnShareTargetActivated
    (ShareTargetActivatedEventArgs
args)
{
    ShareOperation shareOperation = args.ShareOperation;
    DataPackageView data = shareOperation.Data;
    DataPackagePropertySetView properties = data.Properties;

    await Task.Factory.StartNew(async () =>
    {
        var title = properties.Title;
        var description = properties.Description;

        //ověření zda jsou přijímaná data ve formátu URI
        if (data.Contains(StandardDataFormats.Uri))
```

```
    {
        //získání URI
        Uri uri = await data.GetUriAsync();
    }

    //ověření zda jsou přijímaná data ve formátu RTF
    else if (data.Contains(StandardDataFormats.Rtf))
    {
        //získání rtf
        string rtf = await data.GetRtfAsync();
    }

    //ověření zda mají data textový charakter
    else if (data.Contains(StandardDataFormats.Text))
    {
        //získání textu
        string text = await data.GetTextAsync();
    }

    //ověření zda je formát obrázku
    else if (data.Contains(StandardDataFormats.Bitmap))
    {
        //získání streamu
        RandomAccessStreamReference bitmap = await
            data.GetBitmapAsync();
    }
});
}
```

Přijímá-li výše uvedený kód data od ukázkového kódu, který byl uveden na začátku kapitoly, tak se vykoná třetí blok else if, neboť ve zdroji sdílení je nastaveno `request.Data.SetText("Příklad sdíleného textu")`.

3.6.3 Nastavení

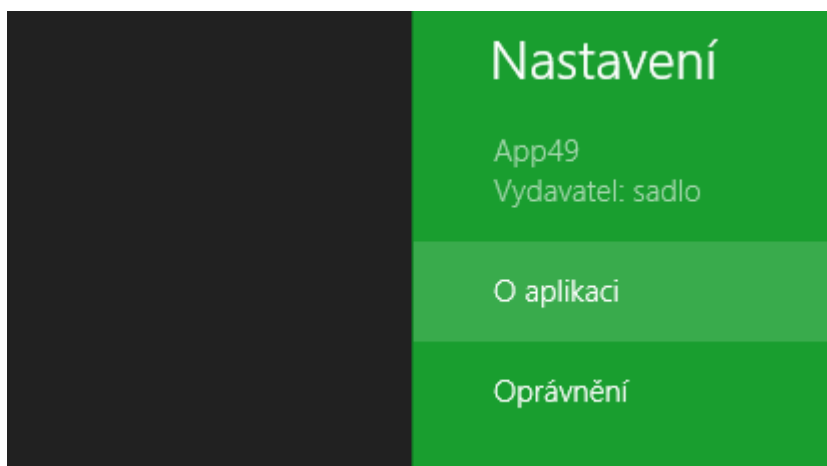
Kontrakt nastavení představuje jediný kontrakt, který musí být implementován jako jediný z kontraktů, aby aplikace mohla projít procesem certifikace pro Windows Store. [26] Níže je uveden úryvek kódu, který ukazuje, jakým způsobem se registruje handler (v těle metody `OnNavigatedTo`), jenž reaguje na snahu uživatelů zobrazit nastavení aplikace. Metoda `SettingsCommandsRequested` demonstruje, v jakém duchu se přidávají tzv. `commandy`.

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    //registrace handleru
    SettingsPane.GetForCurrentView().CommandsRequested +=
        SettingsCommandsRequested;
}

private void SettingsCommandsRequested(SettingsPane sender,
    SettingsPaneCommandsRequestedEventArgs args)
```

```
{  
    var commands = args.Request.ApplicationCommands;  
    //přidání commandu  
    commands.Add(new SettingsCommand("Nějaké ID", "O aplikaci",  
        async (param) =>  
        {  
            MessageDialog dialog = new MessageDialog("Tato mini aplikace vznikla  
                jako demonstrace kontraktů nastavení.");  
            //vyvolání dialogu  
            await dialog.ShowAsync();  
        }));  
}
```

Po přidání výše uvedeného kódu do třídy, která reprezentuje code-behind stránky, bude vypadat sekce nastavení obdobně, jako je vyobrazeno na obrázku 10.



Obrázek 10: Kontrakt nastavení

3.7 Specifické komponenty pro Windows Store aplikace

S novou filosofií grafického uživatelského rozhraní přináší Windows Store aplikace nové komponenty, které jsou typické pro toto prostředí. Nadále je možné používat některé základní komponenty, které jsou známé například ze světa WPF aplikací, ale těmi se práce zabývat nebude z následujících důvodů:

- WPF je poměrně již zaběhnutá technologie, která byla popsána v mnoha akademických pracích a jiných zdrojích.
- Tato práce není primárně zaměřená na tvorbu GUI.

Z těchto důvodů byly vybrány reprezentativní prvky, které jsou popsány v následujících podkapitolách.

3.7.1 AppBar

Hlavní myšlenka tohoto nového uživatelského prvku spočívá v tom, že slouží k obdobným účelům jako menu v desktopových aplikacích. Toto menu se může zobrazovat nahoře nebo dole. U těchto prvků by mělo být dodrženo pravidlo, že ovládací prvky s globálním kontextem (např. ovládání hlasitosti) by měly být uloženy vpravo a prvky vztažené k lokálnímu kontextu (např. editace / mazání vybrané položky) naopak vlevo. Dále je uveden příklad v jazyce XAML, který ukazuje, jak by mohl vypadat horní a spodní AppBar. U obou těchto prvků jsou přidána 2 tlačítka. Jedno vpravo a druhé vlevo. [25]

```
<!-- horní AppBar = horní část obrazovky-->
<Page.TopAppBar>
  <AppBar>
    <Grid>
      <!-- levá horní část obrazovky -->
      <StackPanel HorizontalAlignment="Left">
        <Button Content="levé horní tlačítko" />
      </StackPanel>
      <!-- pravá horní část obrazovky -->
      <StackPanel HorizontalAlignment="Right">
        <Button Content="pravé horní tlačítko" />
      </StackPanel>
    </Grid>
  </AppBar>
</Page.TopAppBar>

<!-- spodní AppBar = spodní část obrazovky-->
<Page.BottomAppBar>
  <AppBar>
    <Grid>
      <!-- levá spodní část obrazovky -->
      <StackPanel HorizontalAlignment="Left">
        <Button Content="levé spodní tlačítko" />
      </StackPanel>
      <!-- pravá spodní část obrazovky -->
      <StackPanel HorizontalAlignment="Right">
        <Button Content="pravé spodní tlačítko" />
      </StackPanel>
    </Grid>
  </AppBar>
</Page.BottomAppBar>
```

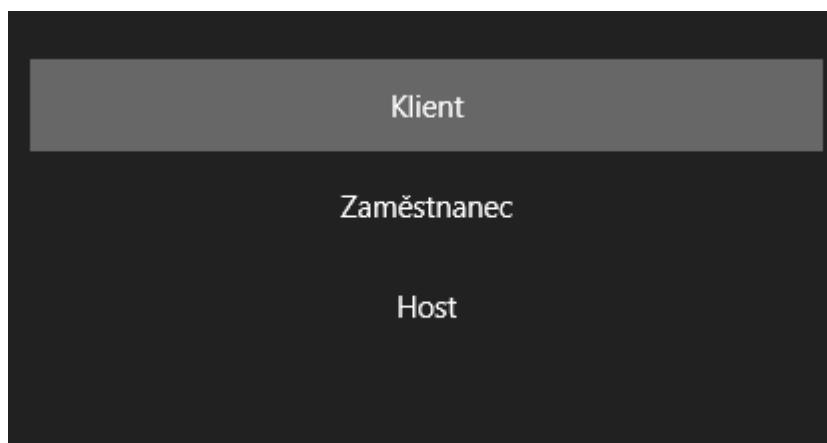
3.7.2 SemanticZoom

Ve Windows Store aplikacích existuje možnost využít tzv. sémantický zoom, který umožňuje na data nahlížet ze dvou úhlů. První pohled na data tak může nabídnout např. pouze důležitý informační extrakt, jak ukazuje kupříkladu obrázek 11, na kterém jsou zobrazeny typy uživatelů. Oproti tomu obrázek 12 ukazuje přiblížený pohled, ze kterého je možné již určit, kdo je klient, zaměstnanec a host. Přepínání mezi přiblíženým a oddá-

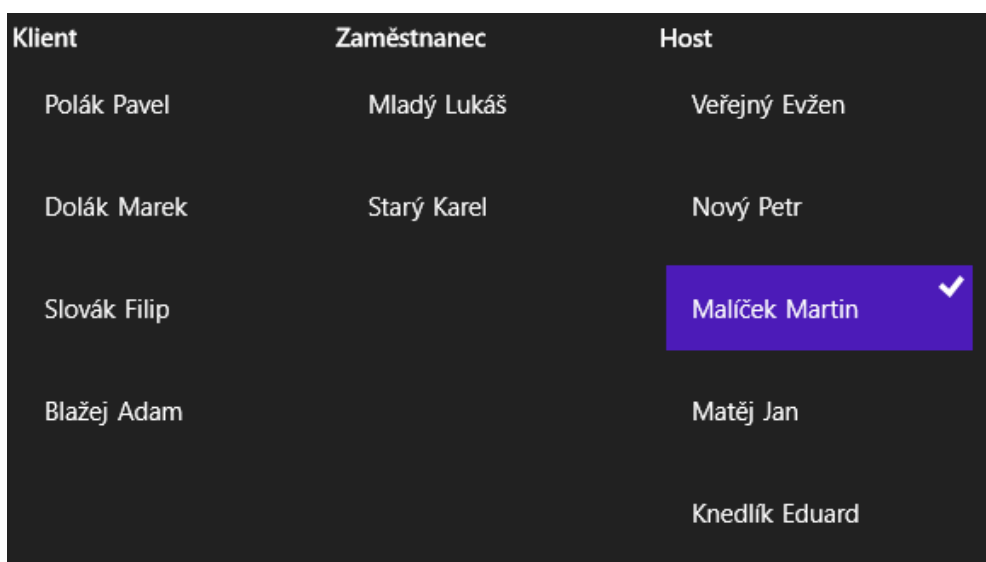
leným pohledem je možné provádět pomocí:

- ctrl + rolování kolečkem myši,
- ctrl + klávesa '+' nebo '-',
- zoom gestem na dotykové obrazovce.

Výhodou tohoto mechanismu je chování, které způsobí, že při výběru položky (např. položka uživatel) v oddáleném pohledu přiblíží systém detailnější pohled (např. seznam uživatelů).



Obrázek 11: Zoomed out



Obrázek 12: Zoomed in

V jazyku XAML se do Windows Store aplikace přidává sémantický zoom pomocí elementu `SemanticZoom`. Obsah přiblížené části se přidá tomuto prvku přes vlastnost `ZoomedInView` typu `ISemanticZoomInformation` ze jmenného prostoru `Windows.UI.Xaml.Cont-`

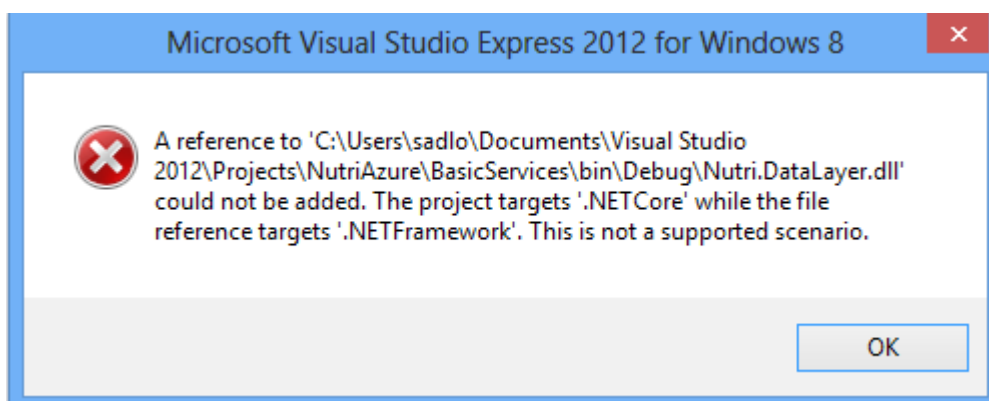
rols. Obdobně se takto přidá i oddálený obsah. K tomuto účelu slouží vlastnost `ZoomedOutView`. V defaultním nastavení je prvně zobrazen přiblížený obsah. Je-li vyžadováno, aby byl zobrazen nejdříve vzdálený obsah, musí být nastavena booleovská vlastnost `IsZoomedInViewActive` na `False`, jak ukazuje následující úryvek kódu.

```
<!-- aktivní je oddálený pohled -->
<SemanticZoom IsZoomedInViewActive="False">
  <!-- oddálený pohled -->
  <SemanticZoom.ZoomedOutView>
    <ListView>
      <TextBlock Text="tlačítka" />
    </ListView>
  </SemanticZoom.ZoomedOutView>

  <!-- přiblížený pohled -->
  <SemanticZoom.ZoomedInView>
    <ListView>
      <Button Content="tlačítko 1" />
      <Button Content="tlačítko 2" />
    </ListView>
  </SemanticZoom.ZoomedInView>
</SemanticZoom>
```

3.8 PCL

Vyvine-li vývojář knihovnu v rámci kterékoliv platformy (Silverlight, .NET, Windows Phone, .NET pro Windows Store aplikace či Xbox 360) společnosti Microsoft, bude možné ji používat pouze v té platformě, pro kterou ji překládal, jelikož všechny technologie nejsou vzájemně plně kompatibilní. Při snaze začlenit do WinRT aplikace dll knihovnu, která byla přeložena pro .NET 4.0, bude informovat Visual Studio o nepodporovaném scénáři, jak demonstruje obrázek 13.



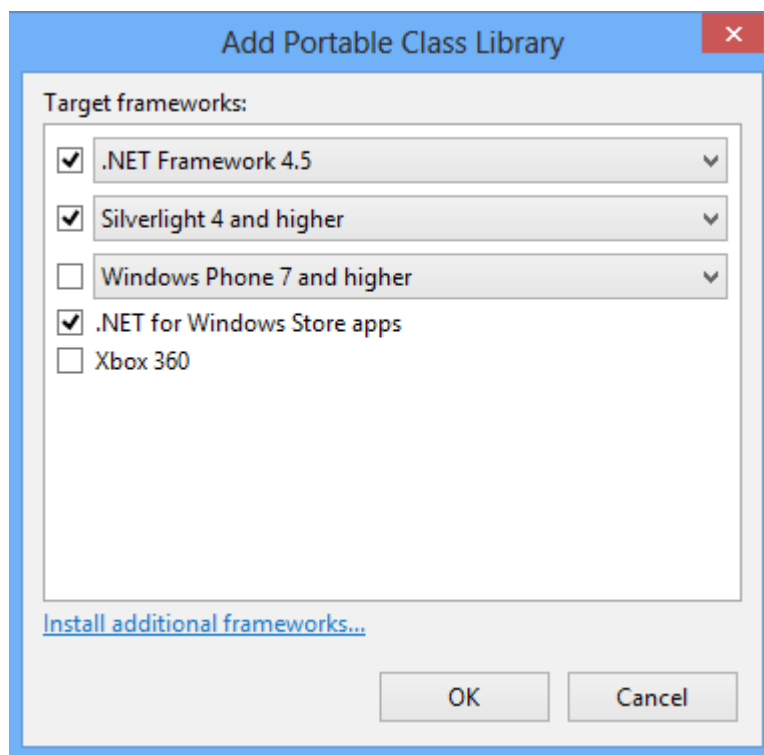
Obrázek 13. upozornění VS2012 reagující na nekompatibilní knihovnu

Z těchto důvodů přišel Microsoft s možností vytvářet portovatelné knihovny, které tuto problematiku řeší a usnadňují tím vývojářům práci. Možnost vytvářet Portable Class Library projekty není zahrnuta v žádné z express edicí Visual Studia 2012. [28] Níže uvedená tabulka ukazuje vybrané technologie a co podporují konkrétní technologie.

*Tabulka 1. Kompatibilita mezi vybranými technologiemi (pozn. * značí, že je vlastnost obsažena v konkrétní technologii [29])*

<i>Vlastnost</i>	<i>Silverlight</i>	<i>Windows Phone 7.5</i>	<i>WinRT aplikace</i>	<i>.NET 4.5</i>	<i>Xbox 360</i>
Jádro BCL	*	*	*	*	*
Jádro XML	*	*	*	*	*
LINQ	*	*	*	*	
IQueryable	*	*	*	*	
Klíč. slovo dynamic	*		*	*	
Jádro WCF	*	*	*	*	
Jádro „síťování“	*	*	*	*	
ViewModely	*	*	*	*	
Data anotace	*		*	*	
XLINQ	*	*	*	*	*
MEF	*		*	*	
Serializace datových kontraktů	*	*	*	*	
XML serializace	*	*	*	*	
JSon serializace	*	*	*	*	
System.Numerics	*		*	*	

Jak z tabulky vyplývá, ve Visual Studiu musí být vybráno, pro které technologie bude portovatelná knihovna použitelná. Do výběru musí být zahrnuty minimálně 2 technologie ze seznamu, který je uveden na obrázku 14.



Obrázek 14. výběr technologií pro PCL

3.9 Publikování aplikací ve Windows Store

S vývojem Windows Store aplikací souvisí i publikování ve Windows Store. Tento obchod totiž slouží jako distribuční místo pro vyvinuté aplikace. Proces publikování se sestává z několika nezbytných kroků. V první řadě je nezbytné, aby existoval účet ve Windows Store, který bude spojen s publikovanými aplikacemi. Při odesílání aplikace je nutné vyplnit údaje o aplikaci. Mezi tyto údaje patří například:

- název a popis aplikace,
- cena,
- zařazení do příslušné kategorie a podkategorie,
- stanovení, pro jaké trhy je aplikace určena,
- zda se používá kryptografie (může být problém na jistých trzích),
- poznámky pro testery.

3.9.1 Certifikace

Aplikace obsažené ve Windows Store prochází certifikačním procesem. Z tohoto důvodu se může publikující dostat do stavu, kdy je aplikace zamítnuta či naopak úspěšně

projde certifikačním procesem a bude možné ji stahovat z Windows Store.

K tomu, aby byla minimalizována pravděpodobnost, že aplikace neprojde certifikačním procesem, může být využito nástroje Windows App Certification Kit. Pomocí této technologie může být provedena předběžná kontrola před odesláním.

V následujícím textu je v bodech stručně uvedeno, co by měla Windows Store aplikace splňovat (dle [30]) pro úspěšnou certifikaci.

1. Přidaná hodnota

- Plně funkční jedinečný a kreativní nástroj ve všech jazycích konkrétních trhů.
- Zkušební verze musí v rozumné míře odpovídat plné.
- Po instalaci aplikace vytvoří pouze 1 dlaždici a obsah dlaždice musí mít souvislost s aplikací.

2. Aplikace nesmí obsahovat pouze reklamy nebo weby.

- Je kladen důraz na umístění reklam (nesmí být v popisu, dlaždici, oznámení).
- Reklamy smí spouštět kód, který pochází pouze od poskytovatele reklamy.

3. Předvídatelné chování aplikací

- Aplikace musí používat pouze Windows Runtime API, což zahrnuje i to, že nesmí komunikovat s jinými aplikacemi a službami.
- Nesmí nabádat k instalování aplikací (výjimku mají udělenou softwarové e-shopy).
- Aplikace nesmí „zatuhnout“.
- Musí být spustitelná na všech podporovaných procesorech Windows Runtime.
- Aktualizace aplikace nesmí snižovat její funkčnost.
- Musí být zajištěna plná podpora ovládání (gesta, myš).
- Aplikace musí podporovat systémové mechanismy (přichycení, nesmí nabízet prvky k ukončení aplikace, zobrazení aplikační lišty dle gesta ruky).

- Aplikace musí splňovat výkonnostní kritéria (např. spuštění do 5 sekund, pozastavení do 2 sekund).

4. Aplikace je řízená uživatelem

- Užívá-li aplikace síť, musí obsahovat prohlášení o zásadách ochrany osobních údajů.
- Při přenášení dat musí být dodrženy zásady ochrany osobních údajů. Uživatelé musí být informováni o přenášení, způsobu přenášení, ukládání a zabezpečení osobních údajů.
- Využívané schopnosti (`internetClient`, `internetClientServer`, `privateNetworkClientServer`) musí být uvedeny v manifestu aplikace.
- V případě sdílení informací osobního charakteru si musí aplikace vyžádat výslovný souhlas ke sdílení.
- V případě sdílení osobních informací osoby, jež nevyvolala sdílení, musí být toto sdílení podloženo písemnou smlouvou s podpisem dotčené osoby.
- Aplikace nesmí ohrozit zabezpečení a ani funkčnost hostitelského operačního systému.
- Aplikace musí umožnit řídit datové proudy ve 2 režimech – neomezený průtok nebo omezený (u videí maximálně 256 kb/s, u zvuku 64 kb/s).
- Pro zprávy získávané prostřednictvím WNS platí stejné podmínky jako pro celou aplikaci.
- Je-li při nákupech použit jiný zprostředkovatel než Windows Store, musí o této skutečnosti aplikace informovat.
- Při shromažďování (aplikace přímo nebo zprostředkovatelská aplikace, již využívá Windows Store aplikace údajů o platebních kartách), musí aplikace splňovat standard PCI DSS.

5. Aplikace musí korespondovat s požadavkem na globální použití

- V aplikacích je zakázáno popuzovat k nenávisti, diskriminaci, násilí, rasismu, porušování lidských práv a práv zvířat.
- Aplikace nesmí podporovat nezákonnou činnost, zobrazovat obscénní, pornografické materiály a používat nadměrné množství vulgarismů.

6. Informace o aplikaci

- Aplikace musí obsahovat jedinečný identifikátor a věkové hodnocení.
- V aplikaci musí být obsažen kontakt na technickou podporu.
- V případě cílení aplikace na specifický region je nutné tento region uvést.
- Ve všech nabízených regionech musí aplikace splňovat požadavek na lokalizaci.
- V aktualizacích aplikace musí být popsány změny.
- Pro každou lokalizaci je nutné uvést snímky z lokalizované aplikace (png větší než 1366 x 768 pixelů).
- Kategorie musí odpovídat charakteru aplikace.

7. Certifikace aplikací klasické plochy

- Je nutné, aby odkaz na stránku se stažením aplikace vedl přímo na stránku, kde se dá aplikace stáhnout. Můžou být 2 odkazy (64 bitová a 32 bitová verze).
- Informace (název, cena a verze aplikace) zadané na Windows Store musí odpovídat informacím na nákupní stránce.

4 WEBOVÉ SLUŽBY

Webové služby jsou prostředek meziprocesové komunikace, který umožňuje komunikaci mezi počítači prostřednictvím počítačové sítě. Důležitou skutečností je, že webové služby nejsou (neměly by být) závislé na konkrétní technologii, což znamená, že může být konzument na klientské straně naprogramován například v Javě a producent na serverové straně v rámci platformy .NET. Parametry, se kterými se volají funkce webových služeb, mohou být klasické objekty klientské technologie či primitivní typy v případě Javy apod. Toto samozřejmě platí rovněž i o návratových hodnotách vzdálených metod.

4.1 REST

REST neboli Representational State Transfer je architektura webových služeb postavená nad protokolem HTTP. Tuto datově orientovanou architekturu navrhl Roy Fielding¹ ve své disertační práci roku 2000. Pomocí architektury REST se pracuje se zdroji, které mají svůj vlastní URI identifikátor. Technologie REST je navržena v návaznosti na CRUD² operace, což jsou operace, kterých se využívá ve většině systémů. Pro mapování jednotlivých CRUD operací slouží jednotlivé metody protokolu HTTP. [31] Formáty, ve kterých jsou data zabalena, jsou obvykle JSON či XML, ale mohou být i jiné. [32] Jelikož se jedná o komunikaci nad protokolem HTTP, používají se i HTTP stavové kódy, které mohou nabývat v rámci technologie REST hodnot, které jsou uvedeny v tabulce 2.

Tabulka 2. Stavové kódy protokolu HTTP

<i>Kód</i>	<i>Hodnota</i>	<i>Vysvětlení</i>
200	OK	Úspěšný požadavek.
201	Created	Byl vytvořen nový zdroj, který je dostupný pod URI předaném v těle odpovědi.
204	No Content	Úspěšné zpracování požadavku bez vracení dat.
404	Not Found	Dokument nebyl nalezen.

4.1.1 Vytváření dat

První operací CRUD kvarteta je Create neboli „vytvoř“. K mapování této metody slouží metoda POST. Příkladem volání může být `http://www.service-xyz.com/customers`,

¹ Spoluautor protokolu HTTP

² Create Retrieve Update Delete

keré vytvoří na vzdáleném serveru zákazníka. Očekávaný stavový kód je 201 (Created), který v sobě nese informaci, pod jakým URI je dostupný nově vytvořený zákazník. [31] [33]

4.1.2 Získávání dat

Druhou operací je Retrieve neboli „získej“. Pro tuto operaci je využito HTTP metody GET. K získávání dat slouží typicky URI podobné `http://www.service-xyz.com/customers/14`, z kterého by měly být logicky vráceny informace spojené se zákazníkem, kterému bylo přiděleno id 14. Stavové kódy by měly být 200 (OK) či 404 (Not Found). [33] Příklad těla XML odpovědi může vypadat podobně jako následující kód.

```
<user id="14">
  <surname>Bálek</surname>
  <firstname>Petr</firstname>
  <phone>420777123456789</phone>
  <address>
    <city>Amsterodam</city>
    <zip>12345</zip>
    <number>87</number>
  </address>
</user>
```

Další oblíbenou variantou obalování dat je formát JSON. Ve formátu JSON by mohlo vypadat tělo odpovědi jako uvedený úryvek kódu.

```
{
  "surname" : "Bálek"
  "firstname" : "Petr"
  "phone" : "420777123456789"
  "address":{
    "city" : "Amsterodam"
    "zip": "12345"
    "number": "87"
  }
}
```

4.1.3 Úprava dat

Pro úpravu dat by měla sloužit HTTP metoda PUT. Této metody však bývá někdy využíváno i k vytváření dat. Nicméně je lepší ji používat pro úpravy. K vytváření dat je vhodné využít metodu POST. Příklad URI může být `http://www.service-xyz.com/customers/14`, kde v těle zprávy zasílané na tuto URI jsou data, se kterými bude nadále disponovat zákazník s ID 14. Stavové kódy by se měly vyskytovat v rozmezí 200 (OK), 204 (No Content), 404 (Not Found). [33]

4.1.4 Odstraňování dat

K signalizaci mazání dat slouží HTTP metoda DELETE. Volání URI *http://www.service-xyz.com/customers/14* by mělo vrátit odpověď s kódem 200 (OK) či 404 (Not Found). [33]

4.2 SOAP webové služby

Webové služby SOAP představují další variantu meziprocesové komunikace. Tato varianta je založená na protokolu XML. Programátor využívající webové služby založené na SOAP obvykle nemusí řešit XML strukturu komunikace, neboť k tomu již existuje ve většině programovacích jazyků dostupné API, které toto řeší. Práce programátora se poté nese v duchu volání metod nad klientským proxy objektem, přes který se volají jednotlivé deklarované operace na webové službě. Tento typ služeb využívá k deklaraci standardu WSDL. Dále mohou být služby vyhledávány pomocí UDDI. [34]

Výhodou tohoto protokolu je možnost provozovat jej nad protokol HTTP na portu 80, který obvykle není blokován firewallem, a tudíž je možné využívat této meziprocesové komunikace, aniž by musely být povolovány jiné protokoly a porty. Naopak nevýhodou těchto služeb je, že obalující metadata zpráv (volání metod / vracení hodnot) zabírají typicky značné množství přenosového pásma v poměru s požadavky (volanými metodami) a odpověďmi (vracenými daty), což může někdy přinést i problémy s rychlostí komunikace na pomalejších sítích. [34] [35]

4.2.1 WSDL protokol

Protokol WSDL neboli Web Services Description Library slouží k deklaraci služby, tj. k tomu, aby klient, který chce využívat webové služby, věděl, co služba umí (jaké nabízí operace), s jakými daty pracuje a kde je dostupná. Tyto informace jsou dostupné v rámci XML elementů, kde pomocí elementu `types` jsou definovány datové typy, s nimiž se v rámci služby pracuje. Element `message` odpovídá volání metody nebo návratu z metody. Dostupné operace datového typu jsou uvedeny v rámci elementu `portType`. Element s názvem `operation` reprezentuje metodu. Pomocí elementu `binding` lze stanovit, jakým způsobem bude možné se službou komunikovat.

4.2.2 SOAP protokol

Oproti WSDL je SOAP protokol použit při každé komunikaci. Tento protokol totiž obaluje přenášená data. Pro lepší pochopení, jak XML komunikace vypadá, nechť existuje třída `User` a metoda `User GetUserByID(int userID)`. Metoda `GetUserByID` vrací dle zadaného identifikačního čísla instanci třídy `User`. Pro zjednodušení, třída `User` má následující tvar:

```
public class User
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string Surname { get; set; }
}
```

Požaduje-li klient služby pomocí operace `GetUserByID`, po službě objekt třídy `User` s `ID` 23, bude požadavek dle vzoru uvedeném ve zdroji [35] vypadat následovně:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <!-- volání getUserByID s předaným parametrem id, který má hodnotu 23 -->
    <getUserByID xmlns="http://jmennyprostor.cz/ws">
      <!-- předaný parametr -->
      <userID>23</userID>
    </getUserByID>
  </soap:Body>
</soap:Envelope>
```

Odpověď na tuto žádost se bude nést v následujícím duchu, jako tomu je například ve zdroji [35], z kterého tato ukázka vychází:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <!-- odpověď na volání metody GetUserByID-->
    <GetUserByIDResponse xmlns="http://jmennyprostor.cz/ws">
      <!-- výsledek volání metody -->
      <GetUserByIDResult>
        <Name>Petr</Name>
        <Surname>Bálek</Surname>
        <ID>23</ID>
      </GetUserByIDResult>
    </GetUserByIDResponse>
  </soap:Body>
</soap:Envelope>
```

4.2.3 UDDI protokol

UDDI neboli Universal Description, Discovery and Integration představuje mechanismus, pomocí kterého je možné služby registrovat, kategorizovat a následně umožnit jejich vyhledávání, např. pomocí Internetu či intranetu. V UDDI registru se

pracuje se čtyřmi druhy prvků (firmy, služby, šablony vazeb a typy služeb). Prvky typu firmy, též zvané bussiness entity, obsahují informace o základních údajích (název, popis, kontakt) společnosti. Dále těmto prvkům mohou být přiřazeny klasifikační údaje, podle náplně podnikání a geografické lokace. Služby neboli bussiness service představují informace o tom, které služby firma nabízí. Šablony vazeb, zvané binding template, odkazují na WSDL deklaraci služby. [36]

4.3 WCF

Windows Communication Foundation (WCF) je technologie .NET frameworku, která byla poprvé zahrnuta ve verzi 3.0 vydané roku 2007. [37] Od této verze je tato WCF součástí dalších verzí. WCF technologie si klade za cíl sjednotit různé způsoby meziprocesové komunikace pomocí servisně orientované architektury (SOA). [38]

4.3.1 Principy WCF

Ve WCF je využíváno tzv. ABC (Address, Binding, Contract). Contract deklaruje, co služba dělá (např. umí sečíst pole čísel nebo smazat uživatele z databáze). Binding deklaruje, jak služba umí komunikovat (např. SOAP, WebSocket, TCP, pojmenované roury apod.). Některé bindingy umožňují službám komunikovat duplexně, jiné pochopitelně ne. Další rozdílností mezi bindingy je podpora stavovosti (session) a kódování (text, binární). Některé bindingy jsou vhodné pro komunikaci v rámci jednoho počítače (např. pojmenované roury), jiné jsou zase vhodné pro vzdálenou komunikaci mezi počítači v síti. Důležitá je však skutečnost, že se neliší v používání (volání v kódu) a vystupují v klient-ském kódu jako jedna vzdálená služba, která je přístupná přes proxy objekt (případ .NET frameworku). Na některé bindingy je možné se připojit i z jiných platforem (Java, Python, PHP apod.). Address deklaruje místo (adresu), kde je služba přístupná. [38] [39]

4.3.2 Hosting WCF služeb

WCF umožňuje hostovat službu ve 2 odlišných prostředích. První, a asi nejběžnější, varianta je v IIS. S využitím této varianty jsou získány důležité vlastnosti. Tyto vlastnosti jsou škálovatelnost, spolehlivost a restart služby v případě neošetřené výjimky. Důležité je připomenout, že starší verze IIS nepodporovaly některé bindingy (např. pojmenované roury). S těmito nedostatky by se programátor v novějších verzích IIS již neměl potýkat. Další možností, jak hostovat aplikace, je tzv. self-hosting, což je využití konzo-

lové aplikace, služby Windows, WPF aplikace či WinForm aplikace k hostování služby. Je-li však použito self-hosting řešení, nezíská aplikace vlastnosti, které jsou přítomny při hostování v IIS. [40]

4.3.3 Výhody WCF

Mezi hlavní výhody WCF patří primárně možnost sjednotit různé druhy meziprocesové komunikace (ASP.NET Web Services, Web Services Enhancements, .NET Remoting, Enterprise Services, Microsoft Message Queuing a jiné další). WCF umožňuje pro jednu službu deklarovat různé bindingy, které umožní používat službu (jeden kód) z více technologií. Například, je-li služba využívána z prostředí, které je typické tím, že jsou komunikující počítače oddělené sítí, a služba se volá z prostředí .NET, tak je vhodné použít například `NetTcpBinding`, který umožní rychlou binární TCP/IP komunikaci. Bude-li však služba používána ze vzdáleného stroje, jako tomu bylo v předchozím příkladě, ale klientská aplikace bude naprogramována např. na platformě Java, bude vhodné nabízet službu například i pomocí `basicHttpBinding`, což je v podstatě SOAP. [38] [39] [40]

Další nespornou kladnou vlastností WCF je možnost využívat různé způsoby zabezpečení (autentizace, autorizace, zabezpečení přenosu).

Dále je možné určit způsob vytváření instancí služby (některé služby nemohou využívat všechny uvedené). Pro tento výběr je důležité stanovit, zda má být služba stavová či nestavová. Na výběr jsou možnosti [38] [42]:

- per call – každý požadavek vytvoří novou instanci,
- per session – pro každé připojení klienta nová instance,
- single – existuje jediná instance služby, která je sdílená mezi klienty.

Mezi další vlastnosti WCF služeb patří možnost řídit počet instancí služby, volání a session. Dále umožňuje WCF nastavit transakční zpracování, skrývat či odkrývat metadata (informace, co služba dělá a s jakými datovými strukturami), nastavit zpracování chyb, vést logování a řídit přístup vláken ke službě. [38] [41] [42]

4.3.4 Podpora WCF ve Windows Store

Jelikož Windows Store aplikace nepodporují některé funkcionality, které jsou běžně dostupné v rámci .NET frameworku, tak tyto změny se pochopitelně týkají i WCF bindingů, bezpečnostních módů a typů ověřování. Z tohoto důvodu jsou uvedeny tabulky 3, 4 a 5, ve kterých je uvedeno, co je podporováno ve Windows Store aplikacích v rámci WCF.

Tabulka 3. WCF bindingy ve Windows Store aplikaci [43]

<i>Binding</i>	<i>Popis</i>
BasicHttpBinding	WS-I Basic Profile 1.1 [44]
NetTcpBinding	TCP komunikace [45]
NetHttpBinding	Komunikace prostřednictvím HTTP či WebSocket [46]
CustomBinding	Definice vlastního bindingu

Tabulka 4. Bezpečnostní módy WCF ve Windows Store [43]

<i>Bezpečnostní mód</i>	<i>Popis</i>
None	Zabezpečení je ignorováno.
Transport	Zabezpečení na úrovni přenosové trasy.
TransportWithMessageCredentials	Zabezpečení je zajištěno šifrováním přenosu a autentizace je vyřešena pomocí ověřovacích údajů. [47]
TransportCredentialOnly	V tomto módu jsou přenášeny ověřovací údaje v podobě čistého textu, což není příliš bezpečné. [48]

Tabulka 5. Typy podporovaných ověření [43]

<i>Ověřování</i>	<i>Popis</i>
None	Bez ověřování
Basic	Základní autentizace dle RFC2617 – ověřovací údaje jsou přenášeny jako prostý nešifrovaný text. [49]

<i>Ověřování</i>	<i>Popis</i>
Digest	Rozšířená autentizace dle RFC2617 – odesílány jsou pouze kontrolní součty generované algoritmem MD5. [49] [50]
Negotiate	Postupné vyjednávání mezi klientem a službou. [49]
NTLM	Ověřování pomocí autentizačního protokolu NT LAN Manager.
Windows	Intranetová autentizace pomocí protokolu Kerberos. [49]
Username (Message Security)	Služba vyžaduje uživatelské jméno a heslo.

4.3.5 Konfigurace klienta WCF ve Windows Store aplikaci

Při využívání WCF služeb z prostředí Windows Store aplikace je důležité vědět, že není podporována konfigurace klientské aplikace pomocí XML souboru `app.config`, na který jsou někteří vývojáři desktopových aplikací v prostředí .NET zvyklí. Absence tohoto souboru může být pro některé vývojáře problém, obzvláště v případě, když vývojář raději použije vygenerovaný xml konfigurační soubor, který vytvořil pomocí nástroje WCF Test Client nebo upřednostňuje deklarativní způsob před imperativním. Namísto deklarativního XML tedy Visual Studio 2012 automaticky generuje imperativní kód. [43]

5 WINDOWS AZURE

Aby bylo možné zajistit provoz klient/server aplikace, je potřeba vytvořit vedle klientské části (WinRT aplikace) i serverovou. Tuto funkci může zajistit provozovatelům systému, kteří nevládní fyzický server, např. běžný hosting, pronajatý VPS nebo v dnešní době také cloud.

Windows Azure je cloud společnosti Microsoft, v němž je možné hostovat své aplikace. K těmto aplikacím se přistupuje prostřednictvím Internetu. Windows Azure má mnoho možností konfigurací a je možné si jej pronajmout za ceny již od cca 15 \$ za měsíc a používat jej na provoz aplikací.

5.1 Datová centra

Projekt Windows Azure je provozován v 8 datových centrech, která jsou rozmístěna na 3 kontinentech (Asie, Evropa, Severní Amerika), viz. tabulka 6. Data aplikací mohou být rozmístěna ve více regionech z důvodu bezpečnosti či rychlosti. [51]

Tabulka 6. Rozmístění datových center Windows Azure [51]

<i>Region</i>	<i>Stát</i>
Asie	Hongkong
	Singapur
Evropa	Irsko
	Nizozemsko
USA	Illinois
	Texas
	Virginie
	Kalifornie

5.2 Exekuční model

Exekuční model určuje, jestli bude nájemce spravovat celý pronajatý virtualizovaný operační systém, bude mít k dispozici předkonfigurovaný aplikační server IIS či použije tzv. roli.

5.2.1 Virtual Machines

Tato varianta exekučního modelu zajistí tzv. IaaS (Infrastructure as a Service). V rámci tohoto modelu může nájemce spravovat celý operační systém, který nemusí být dokonce ani od společnosti Microsoft. Windows Azure nabízí již vytvořené VHD obrazy k použití. Mezi těmito obrazy jsou:

- Windows Server 2008 R2
- Windows Server 2012
- Windows Server 2008 R2 s SQL Serverem
- Linuxové obrazy

Dále je možnost nahrát vlastní VHD obraz operačního systému, pokud nevyhovují již připravené obrazy. [52]

5.2.2 Web Sites

Další variantou exekučního modelu je možnost Web Sites. Tato varianta se liší od Virtual Machines tím, že není třeba správce operačního systému, neboť v rámci této služby je nabídnut již nakonfigurovaný IIS a není umožněno konfigurovat operační systém, ve kterém IIS běží. Tato varianta je více restriktivní, zato je vhodnější pro uživatele, kteří se nechtějí zabývat konfiguracemi, instalacemi a správou OS. V této variantě je možné provozovat aplikace běžící na platformě:

- .NET
- PHP
- Node.js

Dále v této službě jsou podporovány systémy WordPress, Joomla a Drupal. [52]

5.2.3 Cloud Services

Třetí a poslední variantou exekučního modelu je možnost vytvořit tzv. roli. Ve světě Windows Azure existují 2 typy:

- Web Role (webová role)
- Worker Role (pracovní role)

Uvnitř instance webové role běží IIS, a je tedy vhodná pro nasazení webových aplikací. Kdežto pracovní roli je radno použít v případě náročnějších výpočtů, které by webovou roli zbytečně vytěžovaly. [52]

5.3 Správa dat

Většina aplikací potřebuje obvykle zajistit uchování dat po delší dobu, jelikož pracuje s daty, která mají perzistentní povahu. Konkrétní data mohou nabývat strukturovaného charakteru či nemusí být kladen důraz na strukturovanost. Z těchto důvodů nabízí Windows Azure možnost využívat služeb, které se liší svými vlastnostmi a jsou popsány v následujících kapitolách.

5.3.1 SQL Database

Jak název napovídá, s touto variantou se získá možnost využití relační databáze s jejími vlastnostmi (atomické operace, T-SQL dotazy apod.). Tato služba nezahrnuje však pouze systém řízení báze dat, ale i zautomatizovanou administraci softwarového vybavení, o které se nemusí pronajímající uživatel starat. Z toho vyplývá, že se uživatel nemusí zabývat instalacemi aktualizací databázového serveru, operačního systému, kde je databázový server nasazen, neboť tuto činnost má kupující již v ceně a probíhá na pozadí. Další výhodou je možnost využití distribuce dat mezi více serverů, čímž je umožněno získat lepší výkon. [52]

5.3.2 Tables

Tato varianta umožňuje přístup k datům pomocí klíče. Jedná se tedy o technologii key/value, pomocí které je rovněž možné uchovávat data. Nevýhodou této technologie je však skutečnost, že neumožňuje složitější přístup k datům, což je vlastnost SQL Database. Naopak výhodou použití „služby Tables“ může být využití v aplikaci, ve které není třeba mapovat data na tabulky relační databáze, ale vystačí se s přístupem pomocí klíče. [52]

5.3.3 Blobs

Poslední možností, jak ukládat data, jsou „Blobs“ (Binary Large Objects). Tato varianta je vhodná pro ukládání nestrukturovaných dat (obrázky, video a jiné soubory).

5.4 Síťové služby

5.4.1 Virtual Network

Virtual Network slouží k propojení virtuálního stroje uvnitř Windows Azure s lokální sítí. Využitím VPN je možné získat zabezpečenou síť. [52] Toto je výhoda v situacích, kdy chce nájemce minimalizovat únik informací na přenosové trase. Pomocí této technologie je, mimo jiné, i velice ztížen útok MITM.

5.4.2 Traffic Manager

V případě nasazení aplikace ve více datových centrech může být s inteligentním využitím této technologie zrychlena odezva aplikace, neboť je možné například směřovat požadavek uživatele z USA na jemu nejbližší datové centrum, které je také v USA. Bez tohoto deklarativního směřování by mohl přistupovat k aplikaci v některém z asijských či evropských datových center a mohlo by tudíž vznikat zbytečné zpoždění. Další možností využití této technologie může být směřování na základě rychlosti komunikace. V případě, že je odezva z datového centra pomalá, směřuje se na jiné. [52]

5.5 Analýzy

Velice cennou komoditou informačních systémů jsou analýzy, pomocí kterých jsou získávány různorodé údaje, které mohou velice napomoci k efektivitě softwaru samotného, ale i k efektivitě práce využívajícího subjektu (právnícká či fyzická osoba apod.).

5.5.1 SQL Reporting

SQL reporting, známý z SQL Serveru, je možné použít i ve Windows Azure. Pomocí této služby je umožněno získávat data z SQL Serveru v různých formátech (Excel, HTML, XML, PDF apod.). [52]

5.5.2 Hadoop

Další možností, jak provádět analýzy, je využití technologie Apache Hadoop. Tato technologie umožňuje paralelní zpracování velkého množství (řády petabytů až exabytů [53] ($> 10^{18}$ bytu) [54])) dat na různém HW pomocí programovacího modelu MapReduce. S využitím tohoto modelu se úloha rozdělí na menší úlohy pomocí tzv. Map a část Reduce spojí dílčí výsledky. [53]

5.6 Messaging

Messaging je způsobem asynchronní meziprocesové komunikace, který může být využit například pro komunikaci mezi instancemi virtuálních strojů uvnitř Windows Azure či v komunikaci s klientskou aplikací, která je umístěna mimo cloud.

5.6.1 Fronty

Pomocí front je umožněno, aby 2 azurovské aplikace mezi sebou komunikovaly způsobem, že jedna aplikace vystaví data a druhá je poté čte z fronty. [52]

5.6.2 Service Bus

Sofistikovanější, a pro některé případy vhodnější technologie k meziprocesové komunikaci je technologie Service Bus. Tato technologie umožňuje komunikaci typu one-to-many pomocí návrhového vzoru Publish-subscribe. V tomto vzoru je jeden vydavatel a několik předplatitelů, kteří přijímají data od vydavatele. Důležité je připomenout, že aplikace využívající Service Bus nemusí běžet ve Windows Azure, ale může to být mobilní aplikace, desktopová aplikace apod. Pochopitelně to může být ale i technologie, která běží uvnitř Azure. [52]

5.7 Cachování

Mnohdy aplikace přistupuje stále dokola ke stejným datům, která se takřka nemění. Tato data mohou být uložena či spočtená. Z hlediska efektivnosti je lepší uložit kopii dat do tzv. cache, což je vyrovnávací mezipaměť, která slouží k práci s často využívanými daty. Cache může být na různých úrovních systémů. Webová aplikace může využívat například cache s odpovědmi na často prováděné dotazy, ORM nástroj může používat data uložena v paměti, aniž by přistupoval do relační databáze, atd. Důležitým kritériem pro realizaci cache paměti je, aby přístup k nosnému médium, na kterém je cache realizována byl rychlejší než přístup k datům uloženým mimo cache. Z těchto důvodů je cache obvykle provozována na úrovni operační paměti, jelikož je poměrně rychlá, levná a dnes již dosahuje značných kapacit v řádech GB.

5.7.1 Caching

Tato technologie nabízí možnost využití vyrovnávací paměti, která může být sdílena mezi více virtuálními stroji. Důležitou vlastností je skutečnost, že podporuje zamykání. [52]

5.7.2 CDN

CDN neboli Content Delivery Network je služba, která distribuuje data mezi různými úložišti napříč celým světem. Tato cachovaná data nejsou uložena pouze ve Windows Azure datových centrech, ale i na jiných serverech mimo datová centra, a tudíž mohou sloužit pro rychlejší přenos v různých místech světa. [52]

5.8 Identita uživatele

Důležitým kritériem pro většinu systémů otevřených pro externí subjekty je, aby věděly, kdo s nimi smí komunikovat, do jaké skupiny komunikující patří, jaká má oprávnění, atd. K těmto účelům je nabídnuta adresářová služba Windows Azure Active Directory.

5.8.1 Windows Azure Active Directory

Tato služba poskytuje zdarma (do 500 000 objektů [55]) možnost centralizovaného ověřování, které může být využito napříč aplikacemi. Mnohé aplikace (Office 365, Windows Intune, Dynamics CRM Online, apod.) firmy Microsoft již této službě využívají. Výhodnou vlastností je možnost synchronizování dat s lokální službou Active Directory. [55] [56]

5.9 HPC

HPC je služba, která umožňuje vykonávat paralelní výpočty na více strojích zároveň. Tato technologie pravděpodobně najde využití při modelování složitých fyzikálních procesů, při tvorbě krizových plánů (např. velmi přesné zaplavování území při povodních), luštění šifer, renderování náročných 3D scén či ve zpracování jiných velmi časově náročných operací, které potřebují velmi vysoký výkon. K práci s paralelními procesy je použito standardu MPI. [52]

5.10 Media Services

Další služba, kterou Windows Azure zjednodušuje vývojářům práci, je služba zaměřená na média. S využitím této služby získá vývojář nástroj, který může použít ve svých projektech, kde se pracuje s médii. Služba v sobě zahrnuje kódování, streamování, přidávání reklamy do videa a další možnosti. Dále umožňuje distribuovat média v různých formátech, které jsou vhodné pro různé klientské přehrávače. [52]

II. PRAKTICKÁ ČÁST

6 ANALÝZA

6.1 Stávající IS

Za účelem pomoci řešit podvýživu v České republice vznikl z iniciativy Ing. Michaela Mináře informační systém, který je již dnes využíván na oddělení klinické výživy Thomayerovy nemocnice. Dále je systém nasazen na Střední zdravotnické škole a Vyšší odborné škole v Ostravě. Nejvíce je tohoto systému však využíváno v domovech pro seniory, jmenovitě jej využívají subjekty [57]:

- Sociální služby Lanškroun
- Centrum služeb pro seniory Kyjov
- Domov Magnolie
- Domov důchodců Náchod
- Městská charita Plzeň
- SENIOR centrum Blansko, p. o.

6.1.1 SWOT analýza

Silné stránky

- minimální konkurence,
- již existují stávající zákazníci.

Slabé stránky

- systém je možné využít prakticky pouze ve směru uživatel → systém ve formě webových stránek, ne však ve směru systém → systém, z důvodu absence webových služeb,
- ne všechny části systému, na něž vede odkaz, fungují,
- logicky spojené stránky by bylo lepší oddělit (např. screening a jídelníčky).

Příležitosti

- zavedení systému ve zdravotnických školách (střední, VOŠ, VŠ),
- oslovení domovů pro seniory, jež systém ještě nevyužívají,
- nasazení systému ve vězeních,
- zaujmutí výsadního postavení na českém trhu,
- úspěch na evropském a mezinárodním trhu.

Hrozby

- nezískání dotací,
- nepochopení přínosů systému ze strany zákazníků.

6.2 Plánovaný IS

Ing. Michael Minář se rozhodl pro zásadní restrukturalizaci informačního systému, od které očekává lehčí udržitelnost programového kódu a snazší modulární rozšiřitelnost. Návrh, jakým způsobem provést tyto změny, je i praktickou částí této práce. V budoucím systému bude kladen důraz i na umožnění komunikace se systémy třetích stran (např. dodavatelé zdravotnického materiálu). Z původně webové aplikace se tak stane aplikace, s kterou bude možné jednoduše komunikovat i z jiných prostředí. Dnes je možné tento systém prakticky využívat pouze pomocí webového prohlížeče. Zajišťované funkce, které bude implementovaný systém obsahovat, jsou popsány v kapitolách 6.2.2, 6.2.3,

6.2.4, 6.2.5 a 6.2.6. Důležité je však na tomto místě upozornit, že se správnou realizací jádra systému pak bude možné systém volně rozšiřovat v oblasti nutriční péče či jej aplikovat úplně na jiná odvětví.

6.2.1 Subjekty využívající systém

System druhé generace bude možné nadále využívat v nemocničních zařízeních, domovech pro seniory, zdravotnických školách, ale i věznicích, neboť vězeňská služba potvrdila zájem o využívání tohoto systému. Dále se nabízí možnost lokalizovat tento systém pro zahraniční prostředí a nabízet jej za hranicemi České republiky.

6.2.2 Shromažďování dat

Mezi funkce informačního systému patří shromažďování dat, nad kterými může být prováděna případná analýza. Shromažďovanými daty se rozumí personální údaje (např. nadřizený a podřizený), informace o tom, kdo a kdy zajišťoval péči pacientovi, informace o klientech (hmotnost, výška, nutriční stav, kdy u nich byl prováděn screening, apod.), informace o odděleních, místnostech a obsazených postelích. Dále je třeba uchovávat informace o tom, kdo je k čemu autorizován (co smí v systému provádět).

6.2.3 Screening

Důležitou částí nutričního systému je možnost provádění screeningů, které napomáhají k vyhodnocování stavů klienta. Za stávajícího stavu je možné tvořit screeniny založené na MNA-SF^{® 1} dotaznících. V budoucím systému bude systém rozšířen o další dotazníková šetření, kterými jsou: MNA² dotazníky v plném znění, SGA dotazníky³, NRS 2002⁴ dotazníky a MUST dotazníky⁵.

1 MNA-SF[®] neboli Mini Nutritional Assessment Short Form je nutriční screening se zaměřením na seniory. Tento test obsahuje 6 otázek, kde každá odpověď má určitou váhu vyjádřenou v bodech. Výsledek tohoto testu se určí sečtením bodů na odpovědi, které stanoví, zda je pacient v pořádku, podvýživě nebo v riziku podvýživy. [58]

2 Mini Nutritional Assessment obsahuje oproti krátké formě celkem 18 otázek. [58]

3 Screening SGA (Subjective Global Assessment) se liší od zbývajících uvedených tím, že váhy odpovědí jsou dány subjektivně. Při fyzickém vyšetřování se hodnotí subjektivně ztráta podkožního tuku, rozsah otoků a úbytek svaloviny. V případě tohoto subjektivního hodnocení se volí číslo ze stupnice 0 – 3, kde 0 je normální stav a 3 závažný. [58]

4 NRS 2002 neboli Nutritional Risk Screening je zaměřen na použití pro širokou veřejnost. Tento screeningový nástroj stanovuje výsledky z rozsahu hodnot BMI, ztrátě hmotnosti v období 3 – 6 měsíců a vliv základní choroby se kterou se pacient léčí. [58]

5 Malnutrition Universal Screening Tool se skládá pouze ze 3 vyhodnocovaných hodnot, kterými jsou BMI, úbytek hmotnosti v procentech, skutečnost zda dojde v 5 dnech k nepřijímání potravy normální cestou. [58]

6.2.4 Edukace

Další velmi důležitá část systému bude cílena na využívání ve školách. V tomto případě budou studenti pracovat s reálnými daty, které budou vyhodnocovat, což umožní rychlejší a pohodlnější vzdělávání budoucích nutričních pracovníků. V tomto případě budou skutečné osobní údaje (jméno, příjmení, datum narození, pobytové zařízení) pacientů nahrazeny fiktivními. Důležité naměřené údaje pro vyhodnocování nutričních stavů budou ponechány skutečné a budou sloužit jako trénovací data.

6.2.5 Upozorňování

Z důvodu, aby byla zajištěna včasná komunikace, bude systém navržen tak, aby byl schopen z jednoho místa vysílat informace a na jiném místě informace přijímat. Příkladem toku informací může být načasované upozornění personálu. V tomto načasování zadá nutriční pracovník A v 17:00 hodin, že má být provedena v 19:00 hodin kontrola u pacienta P. V 18:00 hodin končí však nutričnímu pracovníkovi A směna a nastupuje pracovník B, kterému se zobrazí v 19:00 hodin informace o tom, aby provedl kontrolu u pacienta P.

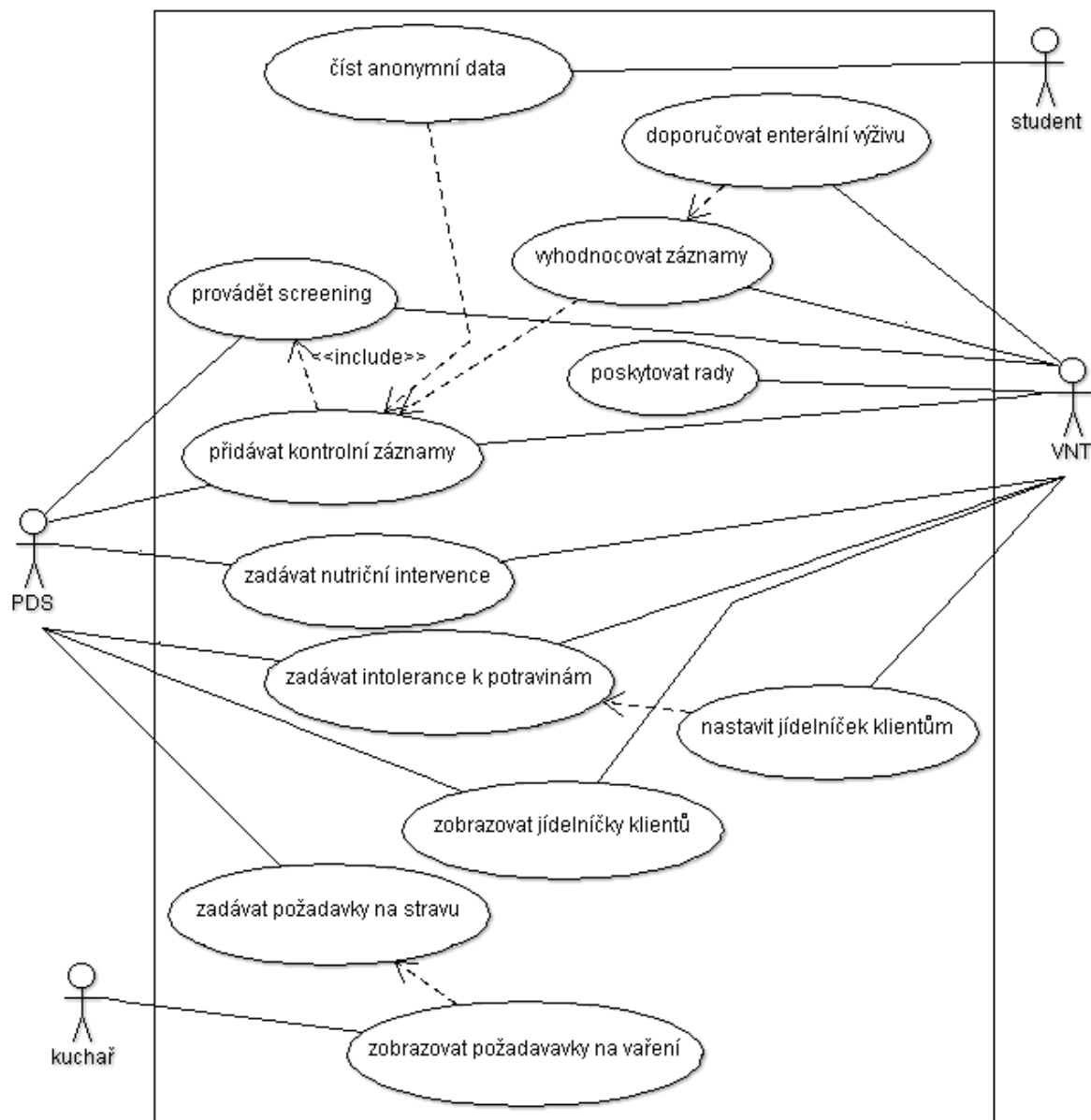
6.2.6 Virtuální nutriční terapeut

Další novinkou, kterou by měl přinést nový systém je funkce, tzv. virtuálního nutričního terapeuta. Virtuální nutriční terapeut je vzdálený odborník (VNT) komunikující pomocí informačního systému. Tento vzdálený terapeut bude pomáhat řešit odborné nutriční problémy uživatelů systémů.

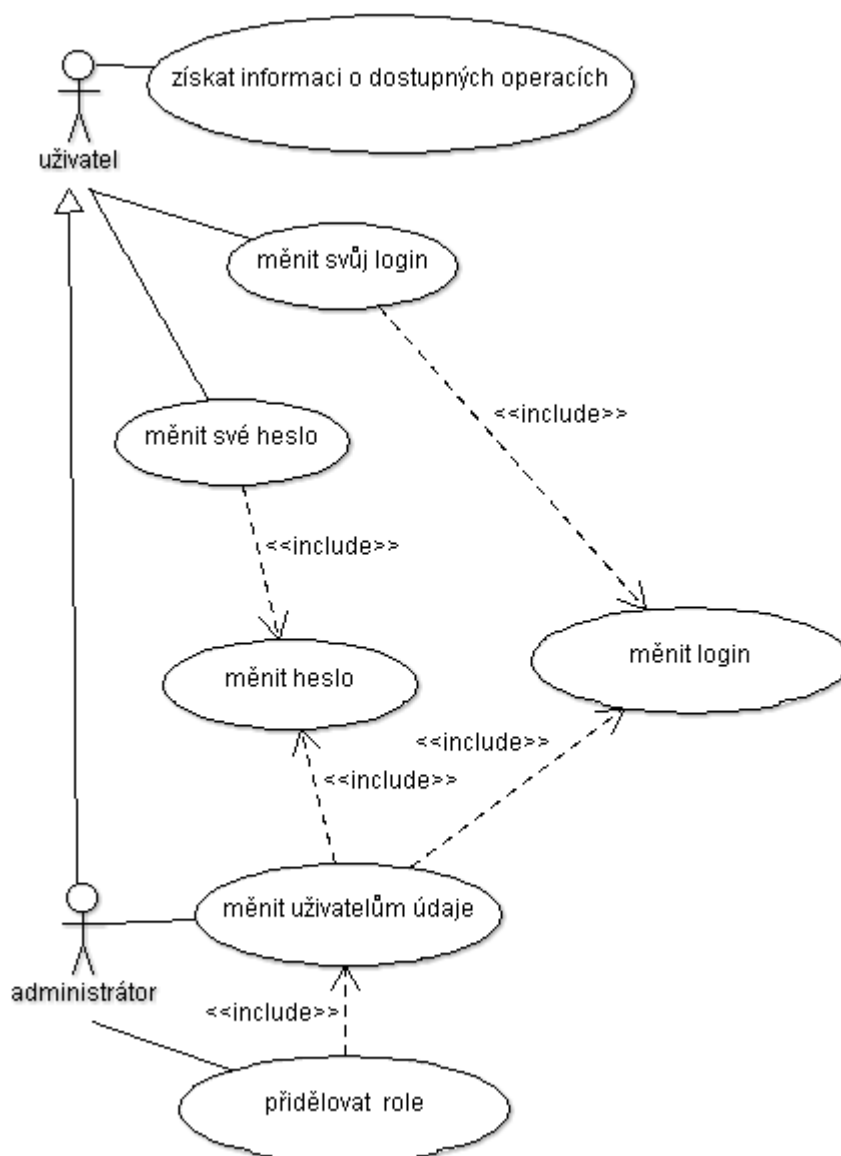
6.3 Případy užití

Pro lepší představu toho, co by měl systém pro nutriční péči zajišťovat, jsou uvedeny v této kapitole obrázky s diagramy případů užití. Ve většině informačních systémů existuje mnoho uživatelů. Přístup do částí systému by měl být řízen osobou administrátora. Tento obecný případ užití je vystižen na obrázku 16. Případ použití systému pro podporu nutriční péče by měl zajišťovat operace, jež jsou uvedeny na obrázku 15. Na tomto diagramu vystupují 4 aktéři. PDS neboli personál domova pro seniory může provádět screening, přidávat kontrolní záznamy, přidávat nutriční záznamy, zadávat nutriční intervence, zadávat intolerance k potravinám, zobrazovat jídelníčky a zadávat požadavky na stravu. Další aktér je kuchař, který může zobrazovat pouze požadavky na vaření. Student může číst anonymní data a věnovat se díky tomu studiu na praktických příkladech. Nejdůležitější

roli v systému však má aktér VNT neboli virtuální nutriční terapeut. VNT může doporučovat enterální výživu, vyhodnocovat záznamy, poskytovat rady provádět screening, přidávat kontrolní záznamy, zadávat nutriční intervence, zadávat intolerance k potravinám, zobrazovat jídelníčky klientů a nastavovat jim povolená jídla.



Obrázek 15: Příklad užití systému pro nutriční péči v rámci domova pro seniory.



Obrázek 16: Příklad užití z hlediska zabezpečení.

6.4 Potřebné zdroje k realizaci

6.4.1 Software

K nasazení serverové části je třeba zajistit server s Windows (např. Windows 2008 či Windows 2012) operačním systémem, ve kterém poběží IIS 7.0 či vyšší. Databázová vrstva může být řešena nad MS SQL serverem. Pro práci se soubory je potřeba FTP server. Rovněž je potřeba nasadit poštovní server. Vedle těchto nástrojů je žádoucí použití pro potřeby správy a vývoje, aby bylo možné se k serveru připojit pomocí „vzdálené plochy“.

Při vývoji systému je vhodné použít vývojové nástroje, které mohou být zdarma. Mezi tyto vývojářské nástroje patří např. Microsoft Visual Studio Express 2012 for Web,

pro vývoj databází Microsoft SQL Server Management Studio. Pro práci se soubory je nezbytný FTP klient. Dále je vhodné použít verzovací systém. Může to být, např. GIT nebo TFS.

6.4.2 Hardware

Pro hardwarové potřeby serverové části je potřeba mít pro nasazení k dispozici fyzický server. Tuto úlohu může však zajistit i VPS či např. webová role Windows Azure. Serverová strana by měla disponovat alespoň 1024 MB RAM na jednu instalaci (např. pro 1 domov důchodců). Vhodné je však, aby paměti bylo více pro případy vysokého payloadu. Pro jednotlivé databáze by mělo existovat alespoň 1 GB místa na disku.

Osobní vývojové počítače by měly disponovat alespoň 2048 MB RAM, neboť vývojové prostředí spotřebuje pravděpodobně více paměti z důvodu instalace emulátorů a spuštěných vývojových prostředí.

6.4.3 Vývojový tým

Pro implementaci systému je určeno, aby pracovali 2 programátoři (junior a senior), 1 databázový specialista a 1 tester. Úloha testera je v týmu nepostradatelná, neboť náklady na předělávání systému mohou výslednou cenu velice prodražit, viz. tabulka 7. Tato tabulka ukazuje, jakým způsobem chyby prodraží vývoj. Tabulka zohledňuje čas zavedení a odhalení chyb v určité fázi vývoje.

Tabulka 7. Cena oprav chyb [59]

<i>Náklady na opravu</i>		<i>Čas detekce</i>				
		<i>požadavky</i>	<i>architektura</i>	<i>konstrukce</i>	<i>sys. test</i>	<i>po uvolnění</i>
<i>Čas zavedení</i>	<i>požadavky</i>	1×	3×	5 – 10×	10×	10 – 100×
	<i>architektura</i>	–	1×	10×	15×	25 – 100×
	<i>konstrukce</i>	–	–	1×	10×	10 – 25×

6.4.4 Čas

Pro implementaci a nasazení projektu je vymezeno období 4 měsíců při práci 2 vývojářů, 1 testera a 1 databázového specialisty, což je cca 16 týdnů. Při pracovní době 8 hodin denně a 5 dní v týdnu vychází délka doby implementace na 640 hodin práce.

6.4.5 Finance

Tabulka 8. Mzdy vývojového týmu

<i>Role</i>	<i>Plat [Kč/hod.]</i>	<i>Hodin</i>	<i>Celkový plat Kč</i>
Junior programátor	200	640	128 000,00 Kč
Senior programátor	270	640	172 800,00 Kč
Tester	200	320	64 000,00 Kč
Databázový specialista	300	640	192 000,00 Kč
Celkové výdaje na mzdy			556 800,00 Kč

7 NÁVRH

7.1 WCF služby

Velmi důležitou součástí navrhovaného systému tvoří WCF služby. Filosofie tvorby jednotlivých služeb se nese v duchu „WCF služba pro každou jednotlivou entitu datové vrstvy“. Toto znamená, že například pro entitu User existuje WCF služba `IUsersService`, pro entitu Diet existuje `IDietsService`, atd. K této filosofii vedlo několik následujících důvodů. Nebudou existovat „super služby“, které by nabízely příliš mnoho operací (např. 300), které spolu nesouvisí. Tyto služby by totiž byly těžko pochopitelné pro externí uživatele, se kterými se počítá.

7.1.1 Generické CRUD rozhraní

Nad většinou dat budou vykonávány tzv. CRUD operace. Z tohoto důvodu bylo navrženo rozhraní, které bude tyto operace u vytvářených tříd deklarovat. Toto rozhraní je uvedené v níže uvedeném úryvku kódu.

```
[OperationContract]
public interface IGenericCrudService<T> where T : Entity
{
    [OperationContract]
    T FindByID(int id);

    [OperationContract]
    T UpdateByID(int id, T updatedEntity);

    [OperationContract]
    T Create(T entity);

    [OperationContract]
    bool Delete(int ID);

    [OperationContract]
    T[] FindAll();
}
```

7.1.2 Generická abstraktní CRUD třída

Aby nemusel být ve všech třídách, které slouží jako implementace nabízených WCF služeb, programován stále dokola stejný kód, je k tomuto účelu navržena abstraktní třída, která je uvedena níže.

```
public abstract class GenericCrudService<T> where T : Entity
{
    public T FindByID(int id) {
```

```
        //najde a vrátí entitu z DB
        return GenericServiceLocal<T>.FindByID(id);
    }

    public T UpdateByID(int id, T updatedEntity) {
        //upraví entitu v DB
        T saved = GenericServiceLocal<T>.Update(id, updatedEntity);
        return saved;
    }

    public T Create(T entity) {
        //vytvoří a vrátí entitu v DB
        T created = GenericServiceLocal<T>.Create(entity);
        return created;
    }

    public bool Delete(int ID) {
        //smaže entitu z DB a vrátí true, když se smaže
        return GenericServiceLocal<T>.Delete(ID);
    }

    public T[] FindAll() {
        //najde všechny entity v databázi
        T[] all = GenericServiceLocal<T>.FindAll();
        return all;
    }
}
```

7.1.3 Konkrétní CRUD služba

Jako ukázka konkrétní služby využívající abstraktní CRUD třídu `GenericCrudService` necht' poslouží třída `UserService`. Tato třída navíc implementuje specifické operace (pro ukázku pouze 1 metoda) pro práci s entitami typu `User`. Tyto specifické metody jsou deklarovány v rozhraní `IUserService`.

```
[ServiceContract]
public interface IUserService : IGenericCrudService<User>
{
    //operace specifické pro práci se entitou User
    [OperationContract]
    void AddAddress(User u, Address a);

    //další operace...
}

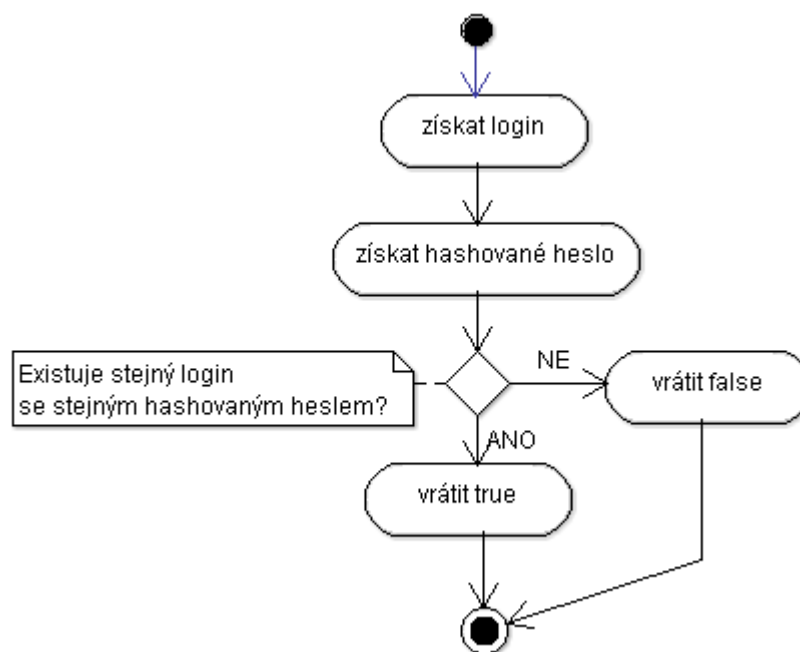
public class UserService : GenericCrudService<User>, IUserService
{
    public void AddAddress(User user, Address address)
    {
        //konkrétní implementace
    }
}
```

7.1.4 Konfigurace WCF

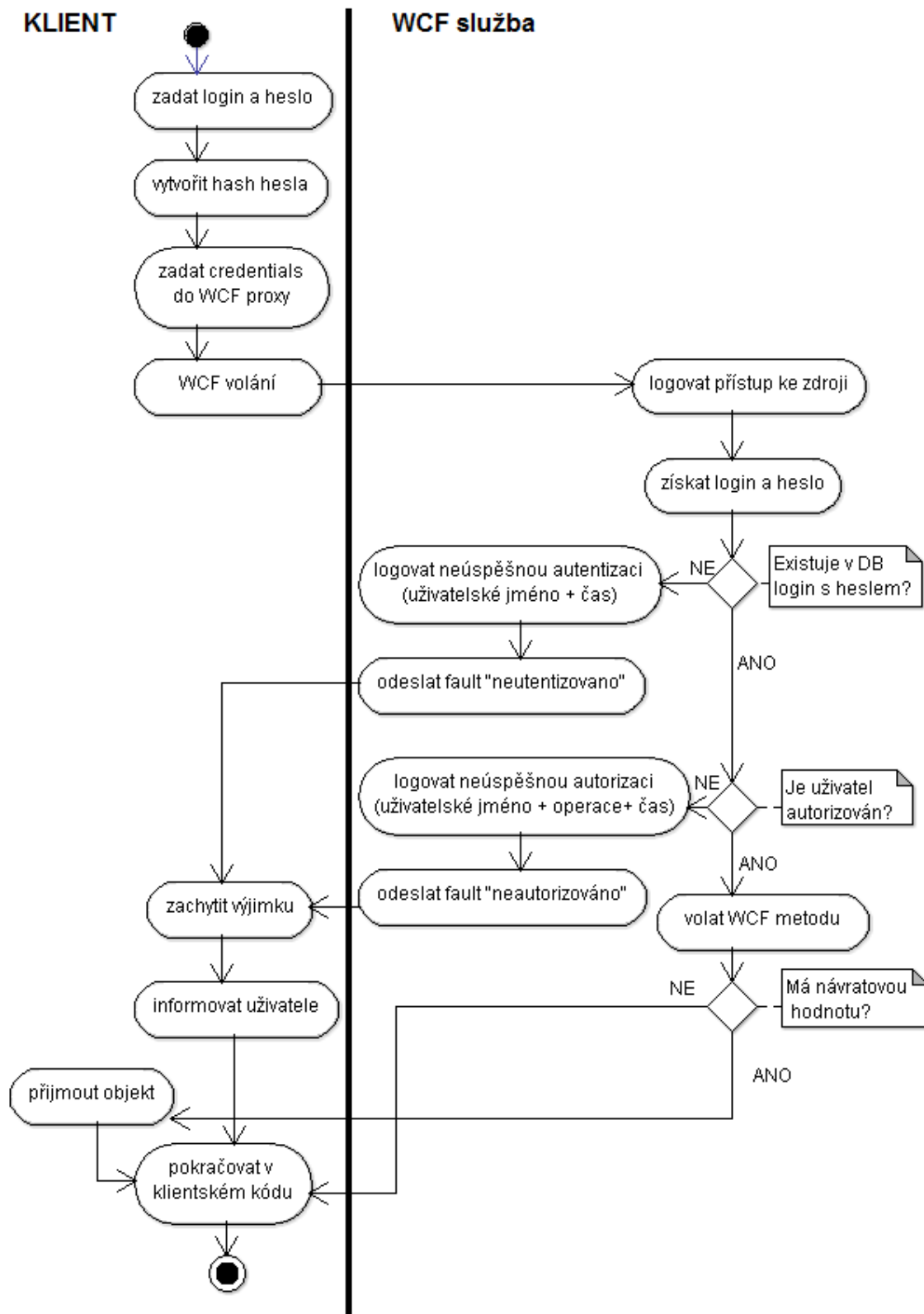
Z důvodu propojitelnosti systému s klientskými technologiemi je vhodné použít různé bindingy, jelikož v různých technologiích existuje různá podpora pro připojení. Z tohoto důvodu je navrženo využít možnosti bindingů `netNamedPipeBinding`, `netTcpBinding` a `basicHttpBinding`. Pomocí připojení typu `netNamedPipeBinding` je možné využít rychlou komunikaci systému s webovými stránkami, které budou psány v prostředí .NET a hostovány ve stejném IIS. Pro rychlou komunikaci .NET ↔ .NET bude použit `netTcpBinding`. Aplikace, které využijí `basicHttpBinding` mohou být vytvořeny např. v Javě, ale rovněž se může jednat o Windows Store aplikace, které `netTcpBinding` nepodporují.

7.2 Zabezpečení dat

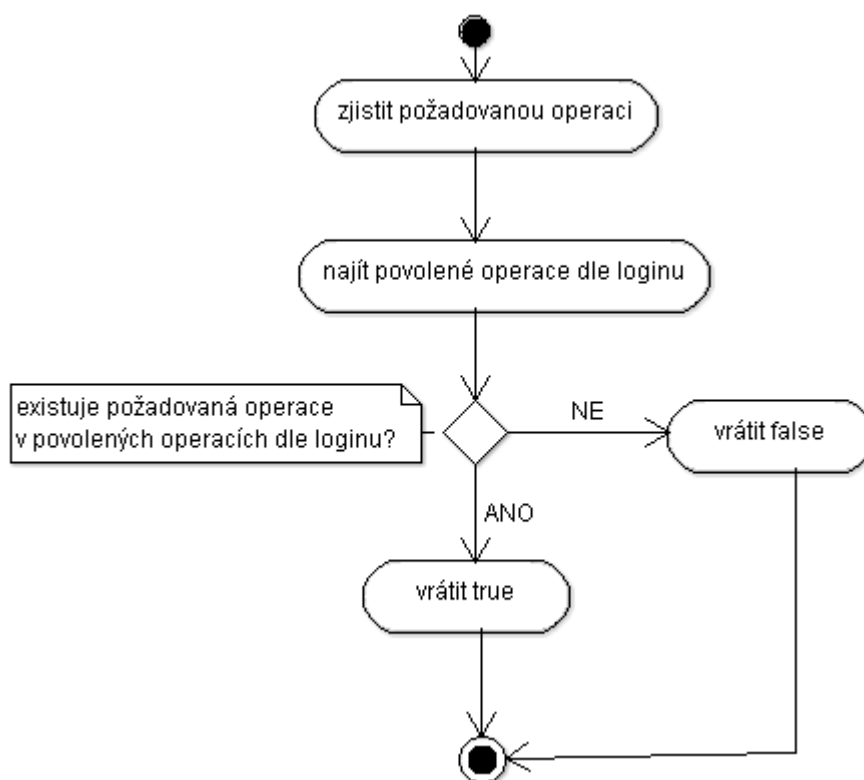
Pro autentizaci klienta je navrženo použití uživatelského jména s heslem. Dále je navrženo ukládat hesla jako SHA-1 hash a přenášet hesla z klienta na server rovněž ve formě otisku, čímž se sníží pravděpodobnost případného rozluštění hesla při zachycení příslušného paketu s heslem. Pro zabezpečení přenosové trasy mezi klientem a serverem je vhodné využít SSL/TLS, které umožní šifrování přenosu. Diagram aktivit obsažený na obrázku 17 ukazuje způsob, jakým se provádí autentizace na serveru. Obrázek 19 zachycuje proces autorizace. Na obrázku 18 je zachycen průběh obecného požadavku klienta volajícího WCF službu.



Obrázek 17: Autentizace na serveru (vytvoreno v ArgoUML)



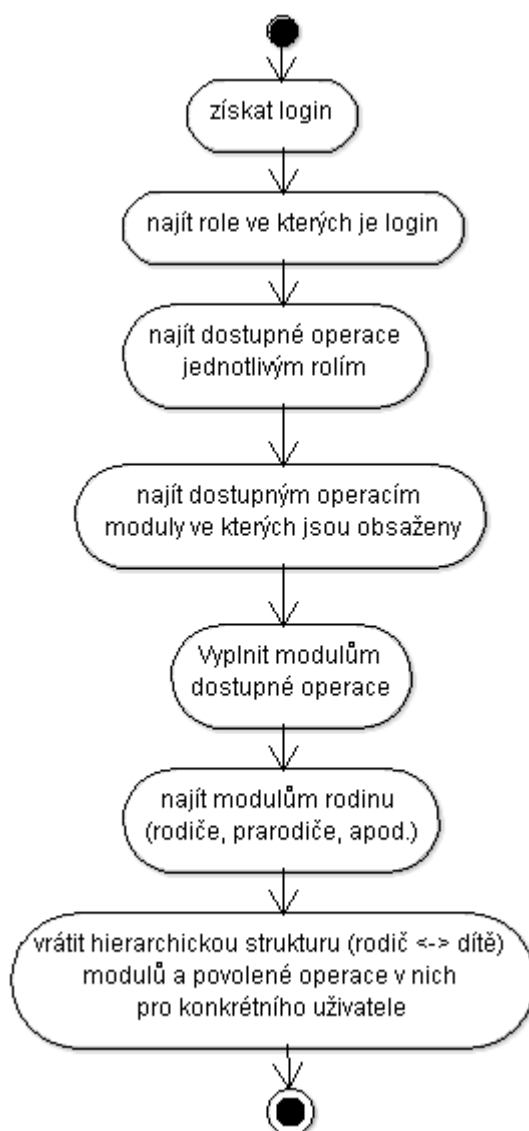
Obrázek 18: WCF klient volající WCF službu (vytvořeno v ArgoUML)



Obrázek 19: Autorizace na serveru (vytvoření v ArgoUML)

7.2.1 Řízení obsahu

V případě systémů, kde se počítá s přístupem uživatelů, kteří mají rozdílné pravomoci, je potřebné zajistit, aby každý uživatel systému měl možnost zjistit, které operace (např. mazání uživatelů, prohlížení diet) smí v systému vykonávat. Toto zjišťování musí být zajištěno dynamicky, na úrovni databáze, jelikož uživatelé mohou být povoleny či zakázány operace administrátorem, tj. bez zásahu programátora. Operace, které v systému můžou uživatelé provádět, mohou mít podobný rys. Z tohoto důvodu je žádoucí slučovat jednotlivé operace do tzv. modulů (např. nastavení, screening). Dále je vhodné tyto skupiny operací dělit, např. screening rozdělit na MNA, NSR, atd. Tento požadavek na rozdělení může být realizován pomocí algoritmu zobrazeného na obrázku 20. V tomto algoritmu se využívá datových struktur tříd `Module`, `Operation`, `Role` a `User`, o kterých je psáno v kapitole 7.3.



Obrázek 20: Získávání modulů (vytvořeno v ArgoUML)

7.3 Entity jádra

Třídy, které jsou mapovány do databáze, dědí od třídy `Entity`, která obsahuje vlastnosti (`ID`, `Description`, `Name`), jež jsou typické pro většinu entit. Třída `Module` v sobě může nést informaci o operacích, které jsou dostupné v rámci modulu. Dále si uchovává informaci o modulech na nižší hierarchické úrovni (např. `nastavení` → `uživatelé` → `klienti`). Zároveň může nést informaci o nadřazeném modulu. Instance třídy `Operation` ví o tom, do jakého modulu patří, z jaké je služby a pro jaké role je dostupná. `Role` musí vědět o tom, jaké operace má povolené a jakých uživatelů se týká. U uživatele je potřeba vědět, v kterých je rolích, jaké má příjmení, jméno, heslo a login. Obrázek 21 ukazuje, jakým

způsobem může být provedeno mapování tříd na tabulky relační databáze.

```
[KnownType(typeof(ICollection<Module>))]
public class Module : Entity {
    public Module() {
        SubModules = new List<Module>();
        Operations = new List<Operation>();
    }

    [DataMember]
    public virtual ICollection<Module> SubModules {get; protected set; }
    [DataMember]
    public virtual Module Owner { set; get; }
    [DataMember]
    public virtual ICollection<Operation> Operations { set; get; }
    public virtual string RepreImage { get; set; }
}

public class Operation : Entity {
    public virtual Module Module{ get; set; }
    public string InService { get; set; }
    public string OperationName { get; set; }
    public List<Role> Roles { get; set; }
}

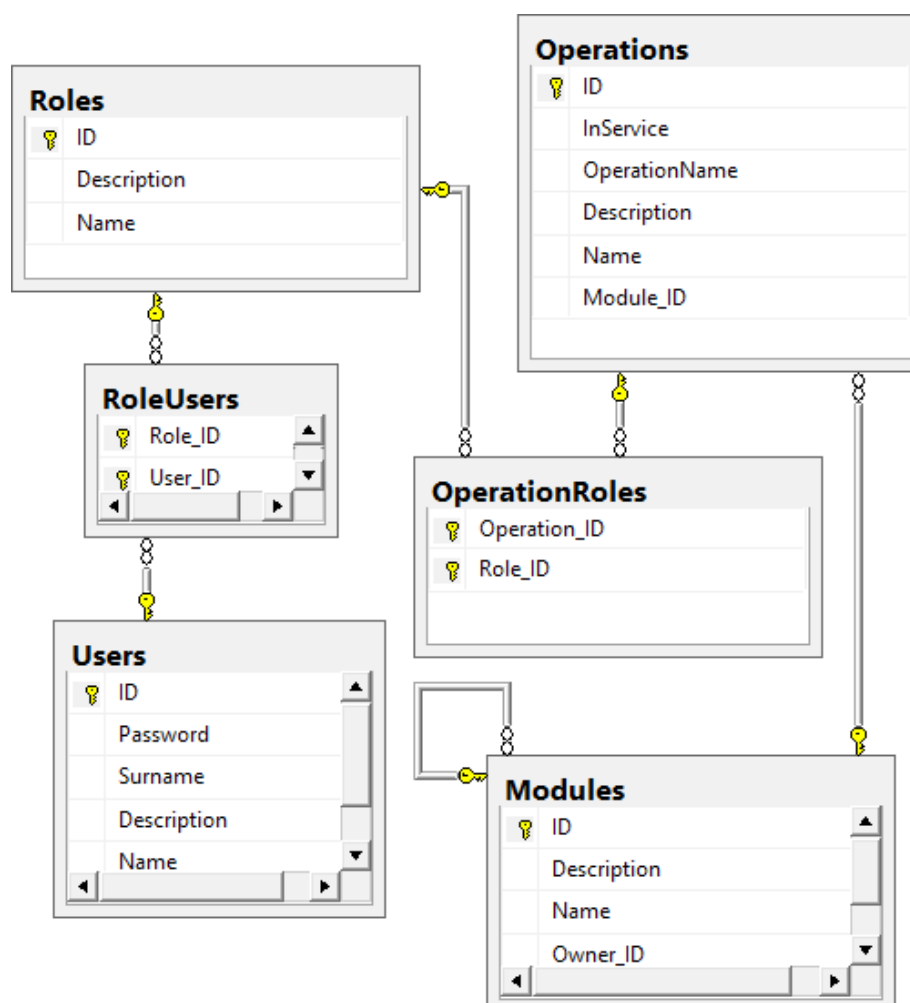
public class Role : Entity {
    public Role() {
        EnabledOperations = new List<Operation>();
        Users = new List<User>();
    }

    [DataMember]
    public virtual List<Operation> EnabledOperations { get; set; }
    public virtual List<User> Users { get; set; }
}

public class User : Entity {
    public User() {
        Roles = new List<Role>();
    }

    public virtual List<Role> Roles { get; set; }
    public string Password { get; set; }
    [DataMember]
    public string Surname { get; set; }
    [DataMember]
    public string Login { get; set; }
}

public class Entity {
    [Key]
    [DataMember]
    public int ID { get; internal set; }
    [DataMember]
    public string Description { get; set; }
    [DataMember]
    public string Name { get; set; }
}
```

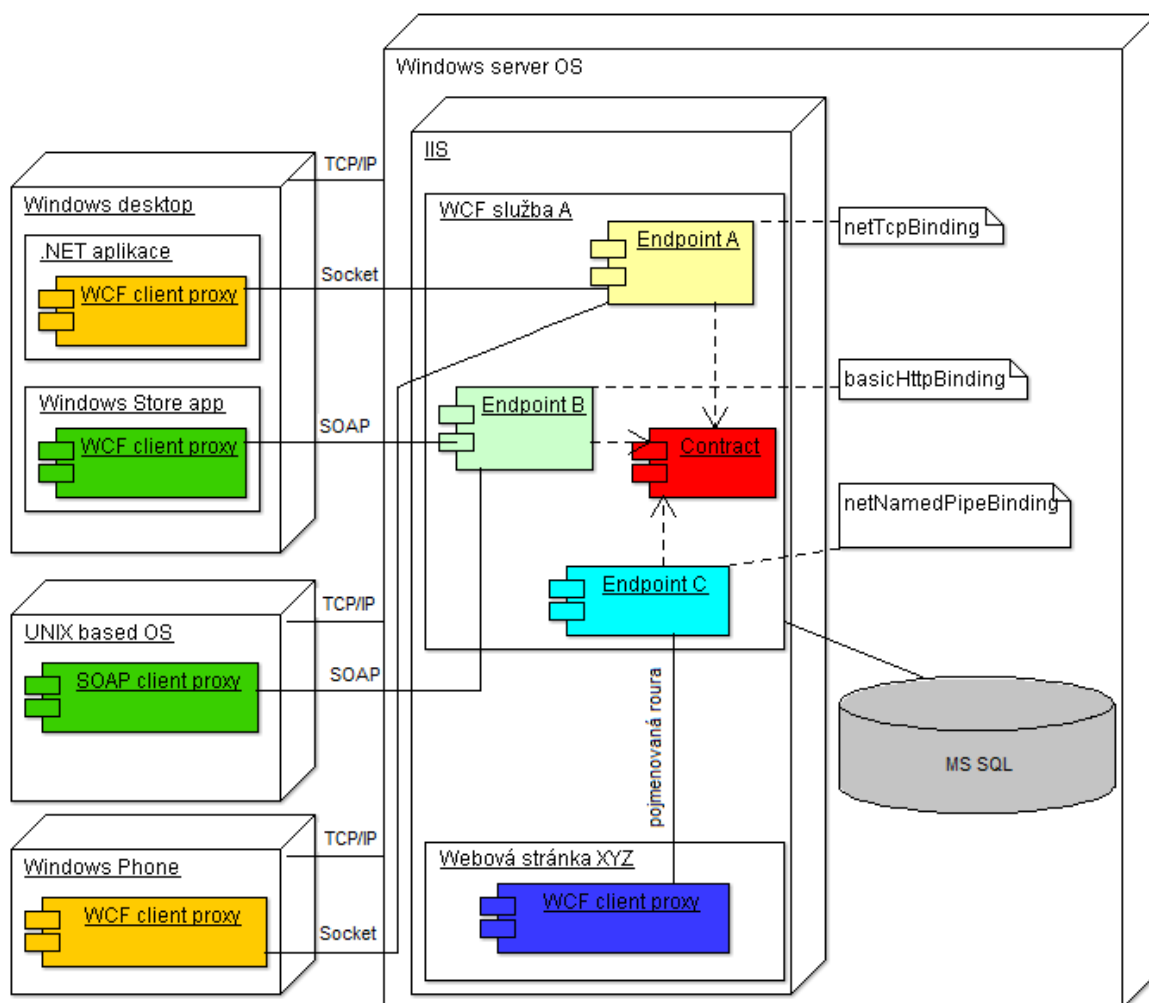


Obrázek 21: Mapování jádra systému na tabulky (vytvořeno pomocí nástroje Microsoft SQL Server Management Studio)

7.4 Nasazení klient / server

7.4.1 Vlastní server

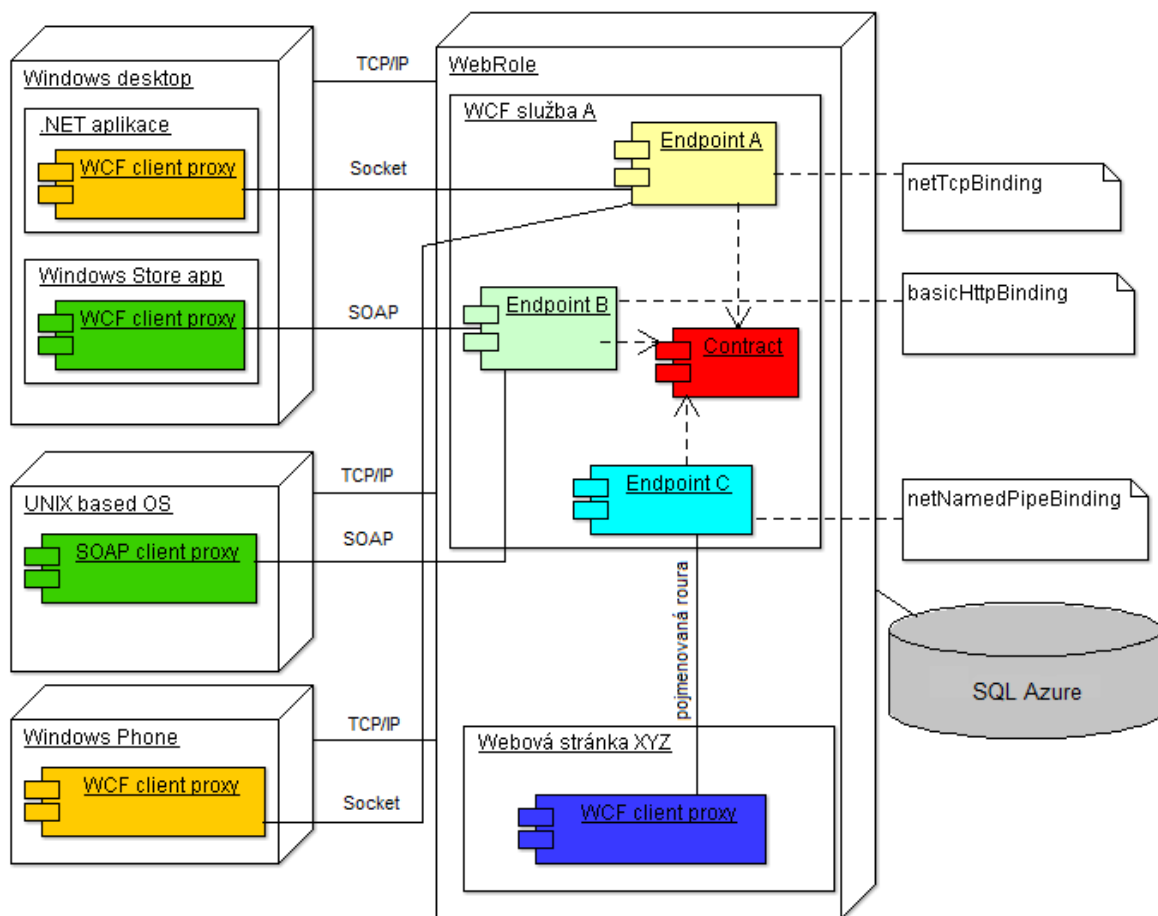
Důvodem pro nasazení serverové části na vlastním serveru mohou být např. obavy z přístupu nepovolané osoby k datům, obava z nedostupnosti systému v případě výpadku internetového spojení nebo prostý fakt, že organizace, která bude systém využívat, vlastní server, kde je Windows server OS a IIS 7.0. V tomto scénáři vypadá nasazení tak, jak je uvedeno na obrázku 22 s diagramem nasazení. Na tomto diagramu jsou v levé části klientské aplikace, které využívají jednotlivých služeb systému prostřednictvím WCF. V pravé části je zachycená serverová strana s IIS. K persistenci dat na serveru může být využit např. MS SQL server.



Obrázek 22: Nasazení – varianta s vlastním serverem (vytvořeno v ArgoUML)

7.4.2 Windows Azure

Typičtější scénář nasazení tohoto systému bude s využitím Windows Azure. V tomto případě budou služby hostovány v rámci tzv. WebRole. Persistence v této variantě bude zajištěna pomocí SQL Azure. Diagram této varianty nasazení je uveden na obrázku 23.



Obrázek 23: Nasazení – varianta s Windows Azure (vytvořeno v ArgoUML)

8 KLIENTSKÁ APLIKACE

Pro vytvoření ukázkové klientské aplikace byla zvolena varianta Windows Store aplikace a návrhový vzor MVVM.

8.1 Struktura

Vytvořený projekt obsahuje složky Assets, Behaviors, Commands, Common, Controls, Locators, Services, Styles, ViewModels a Views.

8.1.1 Složka Assets

V této složce jsou zařazeny obrázky, které jsou využívány z různých míst aplikace. Mezi tyto obrázky patří např. logo aplikace, splashscreen, obrázky typů jídel a jiné. Do této složky jsou zařazeny ikony, o kterých je více psáno v kapitole 8.5.

8.1.2 Složka Behaviors

Složka Behaviors obsahuje třídy, které rozšiřují chování prvků uživatelského rozhraní. Vybraná třída z této složky je popsána v kapitole 8.3.8.

8.1.3 Složka Commands

Ve složce Commands jsou uloženy jednotlivé Command prvky vzoru MVVM, které implementují rozhraní `ICommand` ze jmenného prostoru `System.Windows.Input`. Příklad této konkrétní implementace je uveden v kapitole 8.3.4.

8.1.4 Složka ViewModels

Do této složky jsou řazeny části ViewModel použité ve vzoru MVVM. O filosofii, ve které se nesl vývoj těchto částí, je psáno v kapitolách 8.3.2 a 8.3.3.

8.1.5 Složka Views

Obsah prvního „V“ ze vzoru MVVM, tedy jednotlivá View, jsou umístěna ve složce Views.

8.1.6 Složka Controls

Jelikož je v aplikaci použito více podobných prvků, tak jsou tyto prvky sloučeny do uživatelských komponent, které jsou uloženy ve složce Controls. O jedné z komponent je psáno podrobněji v kapitole 8.3.5.

8.1.7 Složka Services

Ve složce Services jsou uvedena rozhraní služeb, které je možné využít v rámci projektu. Důležité je zdůraznit, že tato složka obsahuje složku Local a Remote. Ve složce Local jsou umístěny lokální implementace služeb a ve složce Remote implementace volání vzdálených služeb. O filosofii servisní vrstvy je psáno více v kapitole 8.2.

8.1.8 Složka Styles

Do složky Styles jsou zařazeny styly, které jsou používány v aplikaci.

8.1.9 Složka Locators

Složka Locators obsahuje 2 třídy. Tyto třídy jsou ViewModelLocator a ServiceLocator. O třídě ServiceLocator je zmínka v kapitole 8.2, o druhé třídě v kapitole 8.3.1. Tyto dvě třídy mají společný rys, kterým je fakt, že v sobě drží reference příslušných prvků. Třída ViewModelLocator je zaměřena na ViewModel prvky vzoru MVVM a třída ServiceLocator na jednotlivé služby. V těchto třídách je rovněž prováděna instanciací příslušných tříd.

8.2 Servisní vrstva

Servisní vrstva v této klientské aplikaci slouží k využití služeb, které mohou být vzdálené či lokální. Každá služba je vždy prvotně deklarovaná jako rozhraní, čímž říká, co umí, a je tedy jedno, jakým způsobem toho dosáhne a co k tomu potřebuje. Příkladem může být rozhraní IDietsService, které deklaruje, že pomocí služby, která jej implementuje, je možné najít všechny klienty a jejich diety nebo přidat či odebrat surovinu do diety.

```
public interface IDietsService : ICrudService<Diet>
{
    Task<ObservableCollection<GroupedItem<Client>>> GetClientsAndDiets();
    void AddFood(Diet diet, Food food);
    void RemoveFood(Diet diet, Food food);
}
```

Tato filosofie založená na rozhraních je důležitá, neboť je tímto způsobem možné pružně

měnit různé implementace (implementující třídy) služeb od různých dodavatelů. Služby díky tomu mohou být kupříkladu emulované lokálně, čehož se využije v situaci, když je plánováno vytvořit WCF službu, která bude přistupovat do databáze, ale není přítomen databázový specialista, jenž by dokázal implementovat výkonné databázové dotazy. Konkrétní lokální implementace tedy může v tomto případě používat relativně nevýkonné operace nad seznamy, slovníky a jinými datovými strukturami a vývojář klientské aplikace se může soustředit na tvorbu GUI.

Na tento kompromis bylo přistoupeno i při vývoji klientské aplikace. Důležitou roli v určení, která implementace služby bude použita, hraje třída `ServiceLocator`. Ukázka toho, jak může vypadat inicializace této třídy, je uvedena v úryvku kódu níže.¹ Na tomto úryvku je uvedeno, že lokální služba musí mít vždy repositář dat, nad kterými pracuje. Oproti tomu vzdálená služba mít žádný repositář nemusí, jelikož to je implementační detail serverové strany.

To, zda je použita vzdálená či lokální varianta služby, pro funkčnost klientské aplikace nehraje roli.

```
//repositář lokální služby
public static dynamic usersRepository = new ObservableCollection<User>();
//varianta lokální služby
public static IUserService Users = new UsersService(usersRepository);
//varianta vzdálené služby
public static IUserService Users = new RemoteUsersService();
```

8.2.1 Vzdálené služby

Rozhraní `IRemoteService`, které je uvedeno níže, přikazuje všem třídám, které implementují toto rozhraní, aby vrátily v metodě `GetClient()` klientský proxy objekt, pomocí kterého bude konkrétní implementující služba komunikovat se vzdáleným počítačem prostřednictvím WCF.

```
public interface IRemoteService
{
    dynamic GetClient();
}
```

Jako příklad implementující třídy může posloužit například třída služby `RemoteModulesService`, která v metodě `GetClient` vytváří proxy objekt WCF služby a předává mu nezbytné uživatelské jméno a heslo, které je uloženo v instančních

¹ Je však nezbytné, aby byl identifikátor `Users` použit pouze jednou, jelikož by se nepodařilo spustit aplikaci v případě stejných identifikátorů. Stejný identifikátor `Users` je zde totiž použit pouze z důvodu demonstrace. Důležité je rovněž podotknout, že třídy `UserService` i `RemoteService` implementují stejné rozhraní `IUserService`.

vlastnostech `Username` a `Password` třídy `LoginsViewModel`. Dále tato třída implementuje metodu `GetMyModules` z rozhraní `IModulesService`. Tato implementace je provedená ve dvou krocích. V kroku prvním, je získána instance proxy objektu. V druhém kroku, je vrácen výsledek pomocí volání metody `GetAllAsync` nad klientským proxy objektem¹ WCF služby.

```
public class RemoteModulesService : IModulesService, IRemoteService
{
    public dynamic GetClient()
    {
        var client = new BasicServiceClient();
        client.ClientCredentials.UserName.Password =
            ViewModelLocator.LoginsViewModel.Username;
        client.ClientCredentials.UserName.UserName =
            ViewModelLocator.LoginsViewModel.Password;

        return client;
    }

    public Task<ObservableCollection<Module>> GetMyModules()
    {
        return GetClient().GetAllAsync();
    }
}
```

8.2.2 CRUD služby

V projektu je na mnoha místech využíváno CRUD operací, z tohoto důvodu bylo deklarováno rozhraní, které tyto společné operace slučuje do jednoho rozhraní.

```
public interface ICrudService<T> where T : Entity
{
    Task<T> FindByID(int id);

    Task<T> UpdateByID(int id, T updatedEntity);

    Task<T> Create(T entity);

    Task<bool> Delete(int ID);

    Task<ObservableCollection<T>> FindAll();
}
```

8.2.3 Třída `RemoteCrudService<T>`

Na této abstraktní generické třídě je zajímavé, že implementuje veškeré CRUD metody deklarované rozhraním `ICrudService<T>`. Tyto operace jsou v podstatě totožné pro

¹ Tento WCF proxy je vytvořen ze třídy, která byla vygenerována pomocí Visual Studia, po přidání reference na WCF službu.

všechny instance tříd, jež dědí od třídy `Entity` ze jmenného prostoru `Nutri.DataLayer.Entities`. Jinými slovy tato jedna třída slouží pro základní vyhledávání podle ID, upravování, mazání a vytváření entit pomocí WCF. Těmito entitami se rozumí zaměstnanci, klienti, postele, oddělení, apod. Zároveň je důležité připomenout, že deklaruje abstraktní metodu `GetClient()`, která musí být implementovaná v konkrétní třídě, jež dědí od této. Metoda `GetClient()` vrací proxy objekty, přes které se přistupuje na server, kde se volají vzdálené metody pomocí WCF.

```
public abstract class RemoteCrudService<T> : IRemoteService, ICrudService<T>
where T : Entity
{
    public Task<T> FindByID(int id){
        return GetClient().FindByIDAsync(id);
    }

    public Task<T> UpdateByID(int id, T updatedEntity){
        return (Task<T>)GetClient().UpdateByIDAsync(id, updatedEntity);
    }

    public Task<T> Create(T entity){
        return GetClient().CreateAsync(entity);
    }

    public Task<bool> Delete(int ID){
        return GetClient().DeleteAsync(ID);
    }

    public Task<ObservableCollection<T>> FindAll(){
        return GetClient().FindAllAsync();
    }

    public abstract dynamic GetClient();
}
```

8.2.4 Lokální služby

Z důvodů, které byly zmíněny v kapitole 8.2 jsou v projektu použity lokální služby. Jen s tím rozdílem, že to dělají emulovaně a nevolají vzdálený server přes WCF. Většina již implementovaných tříd lokálních služeb dědí od třídy `LocalCrudService<T>`, která v podstatě zastává stejnou funkci jako třída `RemoteCrudService`, již se věnuje kapitola 8.2.3. Část třídy `EmployeesService` nechť poslouží jako ukázková třída, která třídu `LocalCrudService<T>` rozšiřuje. U těchto služeb je důležité, aby měly v repositáři (kolekce) nějaké hodnoty, z tohoto důvodu se volá metoda `Create` v konstruktoru, která přidá vždy jednu instanci do kolekce.

```
public class EmployeesService : LocalCrudService<Employee>, IEmployeesService
```

```
{
    public EmployeesService(object repository)
        : base(repository)
    {
        Create(new Employee { Name = "Viktor", Surname = "Pracovitý",
                               Login="pracantXY", Password = "abcd" });

        Create(new Employee { Name = "Petr", Surname = "Bálek",
                               Login = "Petr", Password="Petr" });
    }

    //zbytek třídy je úmyslně vynechán...
}
```

8.3 Vybrané části aplikace

V následujících podkapitolách jsou vysvětleny stěžejní části aplikace. V některých částech jsou uvedeny i ilustrativní části kódu, které by čtenáři této DP měly nastínit fungování výsledné klientské Windows Store aplikace.

8.3.1 Třída ViewModelLocator

Třída ViewModelLocator v sobě zahrnuje instanciaci jednotlivých ViewModel tříd. Z důvodu relativně malého počtu ViewModel tříd, kterých není ani 20, jsou konkrétní instance vytvářeny hned v konstruktoru a jsou zpřístupněny zbytku aplikace pomocí veřejných statických vlastností této třídy.

8.3.2 Třída BasicViewModel

Jednotlivé ViewModel třídy dědí přímo či nepřímo od třídy BasicViewModel. Kód této třídy je uveden níže. Pomocí metody NotifyPropertyChanged je informováno View, o změněné vlastnosti. Jméno změněné vlastnosti je předáváno jako parametr této metody. V této třídě je deklarována i vlastnost typu Frame, aby bylo možné provádět z ViewModel navigaci na jiné View. Abstraktní metoda InitCommands je volána v konstruktoru třídy BasicViewModel. V jednotlivých implementacích této třídy tato metoda slouží k inicializaci jednotlivých příkazů. Ukázka implementace této metody je uvedena v kapitole 8.3.3.

```
public abstract class BasicViewModel : INotifyPropertyChanged
{
    public Frame Page { get; set; }
    public event PropertyChangedEventHandler PropertyChanged;

    public BasicViewModel() {
        InitCommands();
    }
}
```

```
protected void NotifyPropertyChanged(string propertyName) {
    var handler = PropertyChanged;
    if (handler != null) {
        handler(this, new PropertyChangedEventArgs(propertyName));
    }
}
protected abstract void InitCommands();
}
```

8.3.3 Metoda InitCommands třídy CRUDViewModel

Metoda uvedená v této kapitole, je konkrétní implementace abstraktní metody ze třídy `BasicViewModel`. Třídy `CRUDViewModel` je využito jako datového kontextu `View`, kde se počítá s přidáváním, označováním a mazáním prvků, které jsou zobrazeny typicky v `GridView`. V této metodě jsou implementované některé společné `Command` prvky.

Pro lepší představu necht' poslouží příklad, kdy existuje ve `View` (které má datový kontext nastaven na instanci `CRUDViewModel`) `AppBar` s tlačítky „přidat“, „upravit“ a „smazat“. Příkaz `SelectedObjectCommand` nastaví vybraný objekt k editaci a případně skryje/odkryje `AppBar`. Viditelnost `AppBar` prvku je vázána na hodnotu vlastnosti `IsAppBarOpen`.

Když uživatel klikne na „upravit“, vykoná se `EditingCommand`, který nastaví vlastnost `IsEditing` na `true`, čímž řekne `View`, aby zobrazilo editační panel s editačními komponentami (např. vstupní pole), neboť zobrazení tohoto editačního formuláře je vázáno na hodnotu `IsEditing`. V obdobném duchu se nesou i příkazy `DeletingCommand`, `AddingCommand` a `StopDeletingCommand`, které slouží k ovládání zobrazování či skrývání formulářů.

```
protected override void InitCommands()
{
    //IS SELECTED OBJECT
    SelectedObjectCommand = new ParamCommand((selected) =>
    {
        EditObject = (Entity)selected;
        NotifyPropertiesOnSelected();
        if (selected != null)
        {
            IsSelected = Visibility.Visible;
            IsAppBarOpen = true;
        }
        else IsSelected = Visibility.Collapsed;
    });

    //USER WANTS ADD NEW OBJECT
    AddingCommand = new Command(() =>
    {
```

```
        IsAdding = true;
        NewObject = (Entity) Activator.CreateInstance(CurrentType);
    });

    //USER WANTS EDIT OBJECT
    EditingCommand = new Command(() =>
    {
        IsEditing = true;
    });

    //USER WANTS DELETE OBJECT
    DeletingCommand = new Command(() =>
    {
        IsDeleting = true;
    });

    StopDeletingCommand = new Command(() =>
    {
        IsDeleting = false;
    });
}
```

8.3.4 Command

K tvorbě Command částí návrhového vzoru MVVM je zapotřebí implementovat rozhraní ICommand ze jmenného prostoru System.Windows.Input. V projektu toto rozhraní implementují 2 třídy – Command a ParamCommand. Třída ParamCommand je uvedena níže. Tento příkaz se liší od předchozího pouze tím, že při jeho vyvolání může být předán i parametr. Parametrem v tomto případě může být jakákoliv instance.

```
public class ParamCommand : ICommand
{
    protected Action<object> Action;
    public event EventHandler CanExecuteChanged;

    public ParamCommand(Action<object> action)
    {
        this.Action = action;
    }

    public bool CanExecute(object parameter)
    {
        return true;
    }

    public void Execute(object parameter)
    {
        Action(parameter);
        var handler = CanExecuteChanged;
        if (handler != null)
            handler(this, EventArgs.Empty);
    }
}
```

8.3.5 HeaderControl

V projektu se na mnoha místech pracuje s podobnými skupinami prvků, proto je vhodné nevytvářet redundantní kód a využít možnosti vytvořit vlastní ovládací komponentu, která tyto opakující se prvky sloučí a navenek se bude tvářit jako jeden prvek. Z tohoto důvodu je zde uveden ilustrující kus kódu, který reprezentuje hlavičku, na kterou může být navázán text pomocí dependency property. Tento text může být, např. „Nacházíte se na stránce editace uživatelů“ nebo v kratší formě „Uživatelé“. Dále tato třída, která dědí od třídy `UserControl` ze jmenného prostoru `Windows.UI.Xaml.Controls`, obsahuje metodu `GoBack` události, která je vyvolaná po stisknutí tlačítka, jenž je součástí hlavičky. V těle této metody je provedeno přesměrování na předchozí View.

```
public sealed partial class HeaderControl : UserControl
{
    public static readonly DependencyProperty TextProperty =
        DependencyProperty.Register("Text", typeof(string),
            typeof(HeaderControl), new
                PropertyMetadata(null));

    public HeaderControl() {
        this.DataContext = this;
        this.InitializeComponent();
    }

    public object Text{
        get { return (string)GetValue(TextProperty); }
        set { SetValue(TextProperty, value); }
    }

    private void GoBack(object sender, RoutedEventArgs e) {
        var pageElement = Utils.FindPageElement((Button)sender);
        pageElement.Frame.GoBack();
    }
}
```

Jelikož se jedná v případě uživatelských komponent, o grafické uživatelské rozhraní, je vhodné, aby měla komponenta i svůj vlastní XAML soubor. Tento XAML kód může vypadat obdobně, jako následující, který patří k výše zmiňované třídě `HeaderControl`.

```
<Grid>
    <StackPanel Style="{StaticResource GreenTopStackPanelStyle}">
        <Button x:Name="backButton" Click="GoBack" IsEnabled="true"
            Style="{StaticResource NutriBackButtonStyle}"/>
        <TextBlock x:Name="pageTitle" Text="{Binding Path=Text}"
            Style="{StaticResource NutriPageHeaderTextStyle}"/>
    </StackPanel>
</Grid>
```

Pro použití vlastní komponenty, je zapotřebí provést 2 operace:

1. Uvést v kořenovém elementu patřičný xmlns using, který bude odkazovat do jmenného prostoru, kde je komponenta vytvořena. V případě této komponenty může vypadat xmlns, např. xmlns:control="using:Diplomka.Controls".
2. Provést vykreslení, které může vypadat jako v níže uvedeném XAML kódu.

```
<control:HeaderControl Text="Název stránky XY" />
```

8.3.6 Třída GroupedItem

Třída GroupedItem byla vytvořena z důvodu, aby bylo možné držet v paměti sloučené prvky, které bude využívat prvek GridView s nastaveným slučováním.

Pro lepší vysvětlení použití této třídy jsou uvedeny příklady. V prvním příkladě je vyžadováno sloučení uživatelů do skupiny podle prvního znaku příjmení. Jedna skupina může být například skupina „B“, což je obsah vlastnosti GroupName. V seznamu Items pak budou uživatelé (instance třídy User) s příjmením Bálek, Berka a Bosák. Dále je vhodné v některých skupinách mít reprezentativní obrázek. Adresa tohoto obrázku je uložena ve vlastnosti ReprImage. Vlastnosti OriginalEntity je využíváno v případě, že existuje společná entita pro všechny položky, což v příkladě se jmény není. Této vlastnosti se využije například při vykreslování diet a klientů, jenž mají nastavenou dietu. Toto vykreslení pak vypadá tak, že ve vlastnosti OriginalEntity je instance třídy Diet a ve vlastnosti Items jsou jednotlivé instance třídy Client.

```
public class GroupedItem<T> where T : class
{
    public string GroupName { get; set; }
    public List<T> Items { get; set; }
    public string ReprImage { get; set; }
    public Entity OriginalEntity { get; set; }

    public GroupedItem() {
        Items = new List<T>();
    }
}
```

8.3.7 Metoda NavigateByUser třídy Navigator

Třída Navigator byla do projektu zařazena z důvodu navigování mezi jednotlivými View. Tato třída obsahuje metody podobné níže uvedené metodě. V těle této metody se vybírá cílové View podle parametru typu User. Cíl kam z této metody může být aplikace

nasměrována, je určen pomocí datového typu předaného v 2. parametru. Důležité u těchto metod je, že před vstupem na konkrétní View je nastaven příslušný ViewModel.

```
public async static void NavigateByUser(Frame frame, User user)
{
    switch (user.GetType().Name)
    {
        case "Employee":
            ViewModelLocator.EmployeeDetailViewModel.User = user;
            frame.Navigate(typeof(EmployeeDetailPage));
            break;

        case "User":
            ViewModelLocator.UserDetailViewModel.User = user;
            frame.Navigate(typeof(UserDetailPage));
            break;

        case "Client":
            ViewModelLocator.ClientDetailViewModel.User = (Client)user;
            ViewModelLocator.ClientDetailViewModel.Diets =
                await ServiceLocator.Clients.FindOptimalDiets((Client)user);
            frame.Navigate(typeof(ClientDetailPage));
            break;
    }
}
```

8.3.8 Vlastní chování

Jelikož některé události není možná jednoduše navázat na Command, je potřeba vytvořit řešení, jež se opírá o tzv. dependency property. Při navazování z View na dependency property je možné na tento stav reagovat pomocí callback metody, která byla uvedena při registraci dependency property. V této metodě je pak potřeba zajistit vyvolání příslušného příkazu při žádoucí události.

Níže uvedený úryvek ze třídy `ListViewBaseMultipleSelect` ukazuje callback metodu, která slouží k informování ViewModel části aplikace pomocí prvků Command. Konkrétně tato metoda slouží k navázání události `SelectionChanged` na instanci typu `ListViewBase` (např. `GridView`). V případě, že je vybrán prvek ze seznamu, je o tom informován příslušný ViewModel pomocí spuštění prvku Command, jež je navázán na vlastnost, která je pojmenována „SelectedCommand“. Za předpokladu, že je nějaký prvek seznamu odznačen, je o tom ViewModel informován pomocí příkazu, který je navázán na vlastnost „UnselectedCommand“.

```
private static void CommandPropertyChanged(DependencyObject d,
                                           DependencyPropertyChangedEventArgs e)
{
    ListViewBase listViewBase = ((ListViewBase)d);
    listViewBase.SelectionChanged += (sender, ev) =>
```

```

{
    ParamCommand selected = GetSelectedCommand(listViewBase);
    ParamCommand unselected = GetUnselectedCommand(listViewBase);

    //když se nerefreshuje stránka
    if (listViewBase.SelectedIndex != -1)
        foreach (var removed in ev.RemovedItems)
            unselected.Execute(removed);

    foreach (var it in ev.AddedItems)
        selected.Execute(it);
};
}

```

8.3.9 App.xaml

V XAML částech aplikace je možné přistupovat ke třídě `ViewModelLocator` pomocí jména `ModelViewLocator`, čehož je využito pro mapování datových kontextů (viz. kapitola 8.3.10). Jméno aplikace je nastaveno na hodnotu „Nutriční péče“, čehož může být využito na stránkách, kde není vhodné využít výstižnější nadpis. V rámci aplikace je možné využít styly, které byly automaticky přidány při vytváření projektu nebo mohou být použity styly obsažené v souboru `NutriStyles.xaml` ve složce `Styles`, jenž byly vytvořeny v rámci projektu.

```

<Application.Resources>
    <ResourceDictionary>
        <local:ViewModelLocator x:Key="ModelViewLocator" />
        <x:String x:Key="AppName">Nutriční péče</x:String>

        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary Source="Styles/StandardStyles.xaml"/>
            <ResourceDictionary Source="Styles/NutriStyles.xaml"/>
        </ResourceDictionary.MergedDictionaries>

    </ResourceDictionary>
</Application.Resources>

```

8.3.10 Nastavení datové kontextu

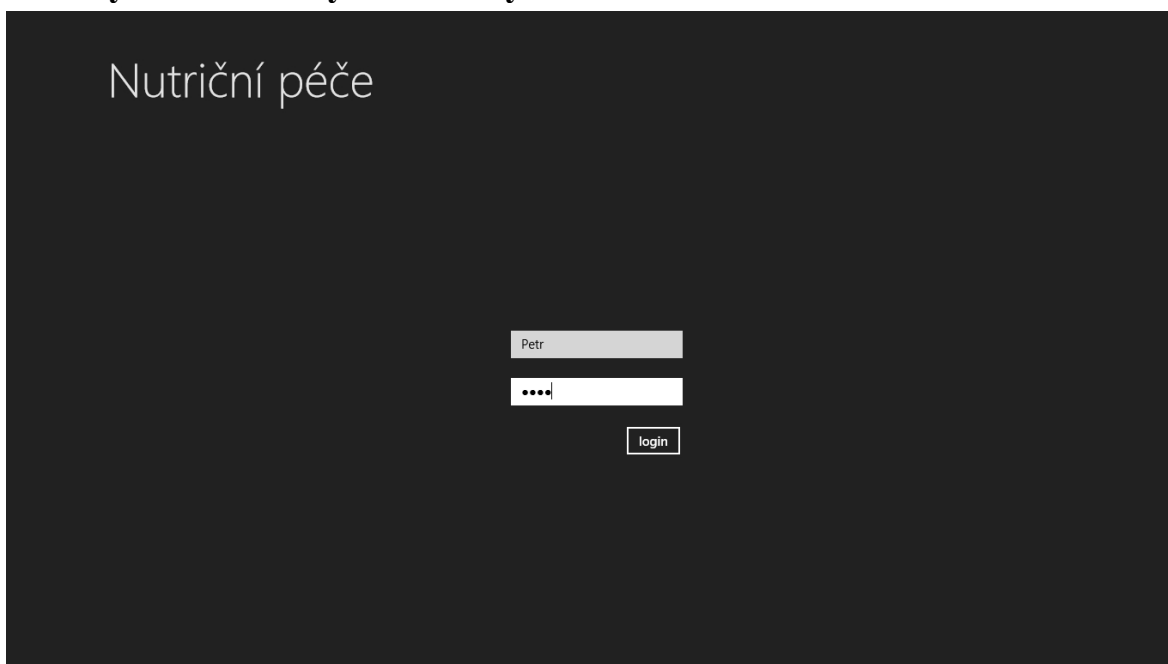
Nastavování datového kontextu je prováděno v XAML části `View`, způsobem jenž je uveden v úryvku kódu. Toto nastavování se provádí v elementech typu `Page` a vypadá obdobně jako tomu je např. v souboru `ClientDetailPage.xaml`.

```

DataContext="{Binding ClientDetailViewModel, Source={StaticResource
                                ModelViewLocator}}"

```

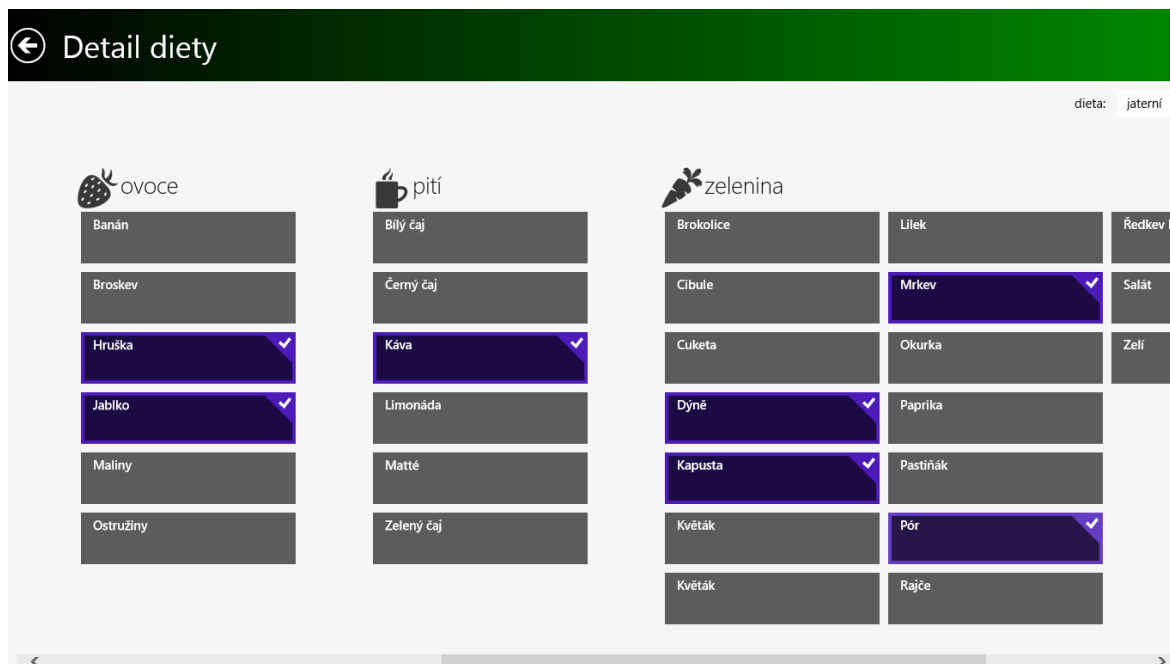
8.4 Vybrané snímky obrazovky



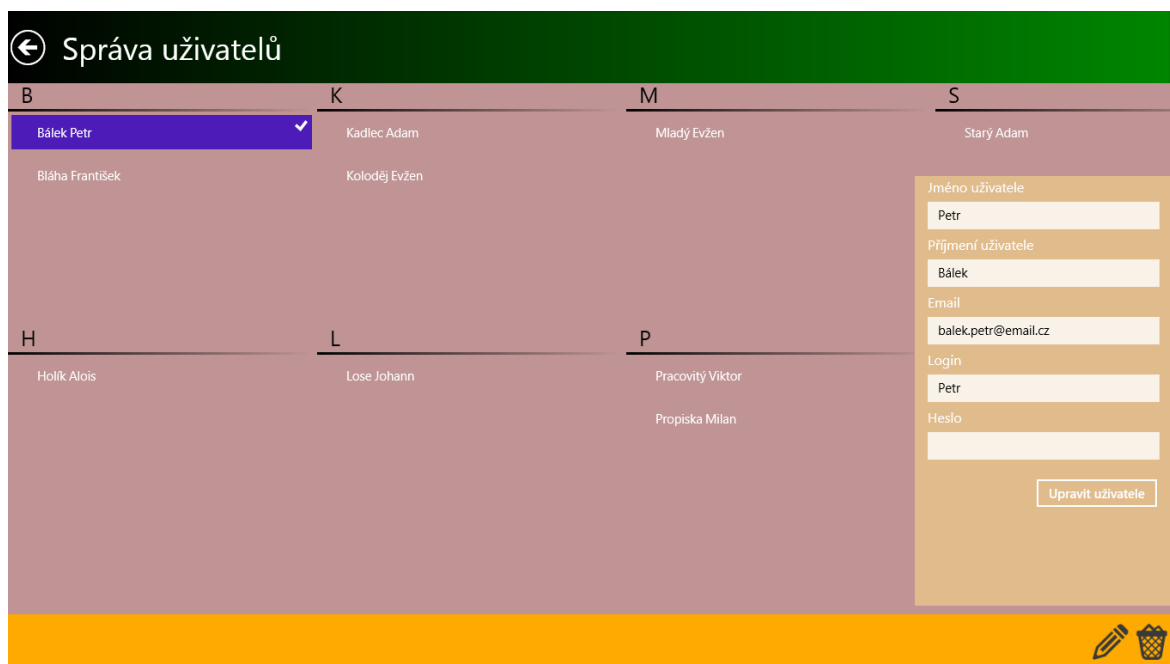
Obrázek 24: Přihlašovací obrazovka



Obrázek 25: Rozcestník modulů



Obrázek 26: Detail diety



Obrázek 27: Sekce uživatelů

8.5 Ikony použité v aplikaci

V této aplikaci jsou na různých místech použity ikony, které je možné používat zdarma i v případě komerčního využití za podmínky zanechání odkazu na stránky tvůrců.

Tyto ikony vytvořili Anna, Alexander a Ivan. K ikonám je možné se dostat přes url <http://icons8.com/download-huge-windows8-set/>.

8.6 Zjištěné problémy

Při vypracovávání klientské aplikace, byly zjištěny drobné nedostatky, které ztěžují vývoj Windows Store aplikací. Mezi tyto problémy patří, např. nemožnost použít některých bindingů ve směru z View do ViewModel části. Tento problém musí být řešen, např. pomocí dependency property a týká se kupříkladu komponent TextBox a PasswordBox, z kterých nebylo možné informovat ViewModel o změně vstupu. Další nepříjemnost v sobě nese GridView komponenta, u kterého když je nastaveno, že může být vybrán pouze jeden prvek, tak je defaultně vybraná první položka. Tento problém však nebyl vyřešen.

9 DALŠÍ VÝVOJ

Po sestavení požadovaného vývojového týmu bude možné implementovat návrh v plném rozsahu ve stanovém horizontu cca 4 měsíců. V následujících fázích by měl být kladen důraz rovněž na navržení testů a postupné programování jednotlivých částí systému. Vývoj klientských částí bude pravděpodobně zaměřen na mobilní zařízení a nasazení na tabletech.

ZÁVĚR

V této práci byl vytvořen návrh klient / server řešení na základě vypracované analýzy. Tento návrh systému může být dále použit pro realizaci serverové i klientské části. Důležitou roli v navrženém systému hraje servisně orientovaná architektura a komunikace prostřednictvím WCF.

Pro potřeby této práce byl rovněž vymyšlen i algoritmus, jenž je možné použít pro řízení zobrazovaného obsahu na klientských aplikacích. Dále bylo v úvodu zpracováno rešerši zaměřené na architektonické vzory, webové služby, Windows Azure a prostředí aplikací pro Windows Store. V práci byla zpracována i ukázková Windows Store aplikace využívající vzor MVVM.

ZÁVĚR V ANGLIČTINĚ

In this thesis was created proposal of client / server solution. This proposal is based on the analysis. This system design can be used for the realization of both server and client side. Important role in the proposed system plays a service-oriented architecture and communication through WCF.

For the purpose of this study was also invented an algorithm that can be used to control the displayed content on the client applications. In introduction of this thesis was created Search aimed at problem of architectural patterns, web services, Windows Azure and application for Windows Store. There was also created Windows Store application. The application is based on MVVM pattern.

SEZNAM POUŽITÝCH ZDROJŮ

- [1] Nutriční terapeut versus výživový poradce. *Zdravě.cz* [online]. 1. 3. 2011 [cit. 2013-05-02]. Dostupné z: <http://zdrava-vyziva.zdrave.cz/nutricni-terapeut-versus-vyzivovy-poradce/>
- [2] Podvýživa. *Lepší péče* [online]. [2010] [cit. 2013-05-02]. Dostupné z: <http://www.lepsipecce.cz/podvyziva>
- [3] Na podvýživu každou minutu umírá pět dětí. *Svět — ČT24 — Česká televize* [online]. 15. 2. 2012 [cit. 2013-05-03]. Dostupné z: <http://www.ceskatelevize.cz/ct24/svet/164557-na-podvyzivu-kazdou-minutu-umi-ra-pet-deti/>
- [4] Nutriční péče dnes... z pohledu dietní sestry. *Sestra+* [online]. 1.6.1999 [cit. 2013-05-03]. Dostupné z: <http://zdravi.e15.cz/clanek/sestra/nutricni-pecce-dnes-z-pohledu-dietni-sestry-121085>
- [5] Podvýživa nemusí jít ruku v ruce s depresí. *Výživa v nemoci* [online]. © 2012 [cit. 2013-05-03]. Dostupné z: http://www.vyzivavnemoci.cz/index.php?id=13&tx_ttnews%5Btt_news%5D=1395&cHash=49135cb2a1da45adbdfded55aaf9cab9
- [6] Podvýživa. *Vitalion.cz* [online]. © 2012 [cit. 2013-05-03]. Dostupné z: <http://nemoci.vitalion.cz/podvyziva/>
- [7] Trygve Reenskaug. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 30 April 2013 [cit. 2013-05-05]. Dostupné z: http://en.wikipedia.org/wiki/Trygve_Reenskaug
- [8] Trygve M. H. Reenskaug coordinates. *ScienceNet.cn* [online]. © 2007-2012 [cit. 2013-05-05]. Dostupné z: <http://blog.sciencenet.cn/blog-89075-242398.html>
- [9] Model-View-Controller z pohledu webových aplikací. *DunLog* [online]. [2012] [cit. 2013-05-05]. Dostupné z: <http://blog.milde.cz/post/257-model-view-controller-z-pohledu-webovych-aplikaci/>
- [10] Úvod do architektury MVC. *Zdroják* [online]. 7.5.2009 [cit. 2013-05-06]. Dostupné z: <http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/>

- [11] Prezentační vzory z rodiny MVC. In: *Zdroják* [online]. 11.5.2009 [cit. 2013-05-06]. Dostupné z: <http://www.zdrojak.cz/clanky/prezentacni-vzory-zrodiny-mvc/>
- [12] Switching to MVVM. *CodeProject* [online]. 19 May 2011 [cit. 2013-05-06]. Dostupné z: <http://www.codeproject.com/Articles/199063/Switching-to-MVVM>
- [13] Microsoft Renames Metro Apps To Windows Store Apps. *WebProNews* [online]. September 13, 2012 [cit. 2013-05-08]. Dostupné z: <http://www.webpronews.com/microsoft-renames-metro-apps-to-windows-store-apps-2012-09>
- [14] Windows Runtime. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-08]. Dostupné z: <http://en.wikipedia.org/wiki/WinRT>
- [15] Windows RT. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-08]. Dostupné z: http://en.wikipedia.org/wiki/Windows_RT
- [16] Začínáme psát aplikace pro Windows Store v HTML5. *Zdroják* [online]. 5.4.2013 [cit. 2013-05-09]. Dostupné z: <http://www.zdrojak.cz/clanky/zaciname-psat-aplikace-pro-windows-store-v-html5/>
- [17] HTML, CSS, and JavaScript features and differences (Windows Store apps). *Windows* [online]. 11/29/2012 [cit. 2013-05-09]. Dostupné z: <http://msdn.microsoft.com/en-us/library/windows/apps/hh465380.aspx>
- [18] *Xaml Object Mapping Specification 2006*. 2006. Dostupné z: <http://download.microsoft.com/download/0/A/6/0A6F7755-9AF5-448B-907D-13985ACCF53E/%5BMS-XAML%5D.pdf>
- [19] Jazyk XAML. *Vbnet.cz* [online]. 2. 2. 2012 [cit. 2013-05-10]. Dostupné z: http://www.vbnet.cz/clanek--198-jazyk_xaml.aspx
- [20] File access and permissions in Windows Store apps. *Windows* [online]. 11/29/2012 [cit. 2013-05-10]. Dostupné z: <http://msdn.microsoft.com/en-us/library/windows/apps/hh967755.aspx>
- [21] Část 2: Správa životního cyklu a stavu aplikace (aplikace pro Windows Store s využitím C#/VB a XAML). *Windows* [online]. [2013] [cit. 2013-05-11]. Dostupné z: <http://msdn.microsoft.com/cs-cz/library/windows/apps/hh986968.aspx>

- [22] Hands-on labs for Windows 8. *Windows* [online]. September 2012 [cit. 2013-05-11]. Dostupné z: <http://go.microsoft.com/fwlink/?LinkID=265117>
- [23] The tile template catalog (Windows Store apps). *Windows* [online]. © 2013 [cit. 2013-05-11]. Dostupné z: <http://msdn.microsoft.com/en-us/library/windows/apps/hh761491>
- [24] Badge overview (Windows Store apps). *Windows* [online]. 10/26/2012 [cit. 2013-05-11]. Dostupné z: <http://msdn.microsoft.com/cs-cz/library/windows/apps/hh779719.aspx>
- [25] Aplikací lišta a multimédia. *MSTV* [online]. [2013] [cit. 2013-05-12]. Dostupné z: <http://www.mstv.cz/vyvojari/videos/777/Windows-8-HOL---Aplikacni-lista-a-multimedia>
- [26] Windows Store App Contracts. *Simple talk* [online]. 14 January 2013 [cit. 2013-05-12]. Dostupné z: <https://www.simple-talk.com/dotnet/.net-framework/windows-store-app-contracts/>
- [27] Aplikace pro Windows Store v HTML5 – kontrakty. *Zdroják* [online]. 26.4.2013 [cit. 2013-05-13]. Dostupné z: <http://www.zdrojak.cz/clanky/aplikace-pro-windows-store-v-html5-kontrakty/>
- [28] Visual Studio 2012 Express with Portable Class Library?. In: HEUER, Tim. *Windows* [online]. August 23, 2012 [cit. 2013-05-13]. Dostupné z: <http://social.msdn.microsoft.com/Forums/en-US/toolsforwinapps/thread/9c7fe120-14b5-49f8-ad09-f48dc80fa5c4/>
- [29] Targeting Multiple Platforms with Portable Code: Overview. *.NET Framework Blog* [online]. 6 Jul 2012 [cit. 2013-05-13]. Dostupné z: <http://blogs.msdn.com/b/dotnet/archive/2012/07/06/targeting-multiple-platforms-with-portable-code-overview.aspx>
- [30] Požadavky na certifikaci aplikací pro Windows 8. *Windows* [online]. 20. března 2013 [cit. 2013-05-14]. Dostupné z: <http://msdn.microsoft.com/cs-cz/library/windows/apps/hh694083.aspx>
- [31] REST: architektura pro webové API. MALÝ, Martin. *Zdroják* [online]. 3.8.2009 [cit. 2013-05-14]. Dostupné z: <http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/>

- [32] JSON and REST. *Slideshare* [online]. © 2013 [cit. 2013-05-14]. Dostupné z: <http://www.slideshare.net/rmaclean/json-and-rest>
- [33] Using HTTP Methods for RESTful Services. *HTTP Methods for RESTful Services* [online]. 2012 [cit. 2013-05-15]. Dostupné z: <http://www.restapitutorial.com/lessons/httpmethods.html>
- [34] Tutoriál Web Services. *Ústav výpočetní techniky* [online]. [2005] [cit. 2013-05-15]. Dostupné z: <http://dior.ics.muni.cz/~makub/soap/tutorial.html>
- [35] SOAP. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-15]. Dostupné z: <http://cs.wikipedia.org/wiki/SOAP>
- [36] Využití webových služeb a protokolu SOAP při komunikaci. *Domovská stránka Jirky Koska -- "VŠE O WWW"* [online]. © Jiří Kosek 1999-2013 [cit. 2013-05-16]. Dostupné z: <http://www.kosek.cz/diplomka/html/websluzby.html>
- [37] .NET. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-16]. Dostupné z: <http://cs.wikipedia.org/wiki/.NET>
- [38] KOŠŤÁL, Marián. Začínáme s WCF. *Vyvojar.cz* [online]. 1. února 2007 [cit. 2013-05-16]. Dostupné z: <http://www.vyvojar.cz/Articles/452-zaciname-s-wcf.aspx>
- [39] WCF - Bindings. *Vyvojar.cz* [online]. 24. února 2007 [cit. 2013-05-16]. Dostupné z: <http://www.vyvojar.cz/Articles/459-wcf-bindings.aspx>
- [40] WCF - Hosting služieb. *Vyvojar.cz* [online]. 23. března 2007 [cit. 2013-05-16]. Dostupné z: <http://www.vyvojar.cz/Articles/467-wcf-hosting-sluzieb.aspx>
- [41] WCF - Instancing. *Vyvojar.cz* [online]. 5. března 2007 [cit. 2013-05-17]. Dostupné z: <http://www.vyvojar.cz/Articles/462-wcf-instancing.aspx>
- [42] WCF - Nastavenie správania služieb. *Vyvojar.cz* [online]. 12. března 2007 [cit. 2013-05-17]. Dostupné z: <http://www.vyvojar.cz/Articles/465-wcf-nastavenie-spravania-sluzieb.aspx>
- [43] Accessing WCF Services with a Windows Store Client App. *Msdn* [online]. © 2013 [cit. 2013-05-17]. Dostupné z: <http://msdn.microsoft.com/cs-cz/library/hh556233.aspx>

- [44] BasicHttpBinding Class. *Msdn* [online]. © 2013 [cit. 2013-05-18]. Dostupné z: <http://msdn.microsoft.com/en-gb/library/system.servicemodel.basichttpbinding.aspx>
- [45] NetTcpBinding – třída. *Msdn* [online]. © 2013 [cit. 2013-05-18]. Dostupné z: <http://msdn.microsoft.com/cs-cz/library/system.servicemodel.nettcpbinding.aspx>
- [46] NetHttpBinding – třída. *Msdn* [online]. © 2013 [cit. 2013-05-18]. Dostupné z: <http://msdn.microsoft.com/cs-cz/library/system.servicemodel.nethttpbinding.aspx>
- [47] Msdn. In: *Windows vs. Basic credential type for basicHttpBinding?* [online]. February 26, 2009 [cit. 2013-05-19]. Dostupné z: <http://social.msdn.microsoft.com/Forums/en-US/wcf/thread/2d51f9ae-7981-49b6-b9c5-6af97a86eccc>
- [48] Wcf TransportCredentialOnly mode: Basic clear text credentials over unsecured transport. *Http://developers.de/* [online]. Jul 31 2006 [cit. 2013-05-19]. Dostupné z: http://developers.de/blogs/damir_dobric/archive/2006/07/31/890.aspx
- [49] Selecting a Credential Type. *Msdn* [online]. © 2013 [cit. 2013-05-19]. Dostupné z: <http://msdn.microsoft.com/en-us/library/ms733836.aspx>
- [50] Autentizace na webu. *Ústav výpočetní techniky* [online]. prosinec 1999 [cit. 2013-06-03]. Dostupné z: <http://www.ics.muni.cz/bulletin/articles/173.html>
- [51] Privacy. *Windows Azure Trust Center* [online]. © 2013 [cit. 2013-05-15]. Dostupné z: <http://www.windowsazure.com/en-us/support/trust-center/privacy/>
- [52] Introducing Windows Azure. *Windows Azure: Microsoft's Cloud Platform | Cloud Hosting | Cloud Services* [online]. © 2013 [cit. 2013-05-22]. Dostupné z: <http://www.windowsazure.com/en-us/develop/net/fundamentals/intro-to-windows-azure/>
- [53] Apache Hadoop. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-23]. Dostupné z: http://cs.wikipedia.org/wiki/Apache_Hadoop
- [54] Exabajt. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-23]. Dostupné z: <http://sk.wikipedia.org/wiki/Exabajt>

- [55] Active Directory. *Windows Azure: Microsoft's Cloud Platform | Cloud Hosting | Cloud Services* [online]. © 2013 [cit. 2013-05-26]. Dostupné z: <http://www.windowsazure.com/en-us/pricing/details/active-directory/>
- [56] Windows Azure Active Directory dokončena. *TechNet Blogs - TechNet Blogs* [online]. 17 Apr 2013 [cit. 2013-05-22]. Dostupné z: <http://blogs.technet.com/b/technet-czsk/archive/2013/04/18/windows-azure-active-directory-dokoncena.aspx>
- [57] Reference. *Nutriční péče* [online]. 2013 [cit. 2013-05-25]. Dostupné z: <http://www.nutricni-pece.cz/reference>
- [58] NÁSTROJE K HODNOCENÍ NUTRIČNÍHO STAVU HOSPITALIZOVANÝCH PACIENTŮ. *Státní zdravotní ústav* [online]. 2011 [cit. 2013-05-24]. Dostupné z: <http://www.szu.cz/svi/hygiena/archiv/h2011-1-04-full.pdf>
- [59] Software testing. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-29]. Dostupné z: https://en.wikipedia.org/wiki/Software_testing
- [60] HTTP Authentication: Basic and Digest Access Authentication. *The Internet Engineering Task Force* [online]. June 1999 [cit. 2013-05-28]. Dostupné z: <http://www.ietf.org/rfc/rfc2617.txt>
- [61] Web Services Introduction. ODSTRČIL, Jan. *KP FEL ČVUT Praha - Distribuované systémy a sítě* [online]. [2005] [cit. 2013-05-30]. Dostupné z: https://dsn.felk.cvut.cz/wiki/_media/vyuka/cviceni/x36mti/prezentace2007/xodstr-ci-prezentace.ppt

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface -- Aplikační programové rozhraní.
ARM	Advanced RISC Machine -- architektura procesorů.
Async	Klíčové slovo uváděné v hlavičce metody, která používá ve svém těle klíčové slovo await.
Await	Klíčové slovo signalizující, že se volá asynchronní metoda.
BCL	Base Class Library
C#	C sharp -- programovací jazyk.
Cache	Rychlá odkládací mezipaměť.
CDN	Content Delivery Network
CRUD	Create Retrieve Update Delete – běžné databázové operace.
CSS3	Cascading Style Sheets 3 – kaskádový styl verze 3.
Deklarativní	Deklarativní programování umožňující vývojáři deklarovat, co má kód udělat, ale o způsob provedení se stará jiná (imperativní) vrstva.
Desktopový	Desktopová aplikace používaná na stolním počítači, např. program na psaní, tabulkový procesor, apod..
DLL	Dynamic-link library – dynamicky linkovaná knihovna s programovým kódem.
Duplex	Obousměrný provoz.
Entita	Třída, která slouží pro mapování dat do databáze.
GB	Gigabajt – jednotka množství dat rovnající se 10 ⁹ bajtu.
Hosting	Běžové prostředí aplikace.
HPC	High-performance computing – technologie Windows Azure umožňující velmi výkonné výpočty.
HTML5	HyperText Markup Language 5 – specifikace jazyka HTML dokumentů.
HTTP	Hyper Text Transfer Protocol – internetový protokol.
HTTPS	Hyper Text Transfer Protocol Secured – zabezpečená verze HTTP.
IE10	Internet Explorer 10 – webový prohlížeč.
IIS	Internet Information Services – webový server od společnosti Microsoft.
Imperativní	Imperativní programování je opak deklarativního. S využitím této filosofie musí vývojář stanovit požadavek na to, co bude kód dělat, ale zároveň i jak to bude dělat.
Instance	Konkrétní datový objekt.
JavaScript	Interpretovaný programovací jazyk.
JSON	JavaScript Object Notation – způsob zápisu dat.
LINQ	Language Integrated Query – dotazovací jazyk.
MD5	Message-Digest algorithm slouží ke generování otisků zpráv.

MEF	Managed Extensibility Framework – knihovna pro vytváření rozšiřitelných aplikací.
MITM	Man in the middle – typ útoku na počítačový systém.
MPI	Message Passing Interface
MVC	Model-view-controller – návrhový vzor
MVP	Model-view-presenter – návrhový vzor
MVVM	Model View ViewModel – návrhový vzor
ORM	Object-relational mapping – objektově relační mapování, kdy jsou mapovány objekty na relační tabulky.
Paralelní	Běžící současně.
PCL	Portable Class Library – přenositelná knihovna mezi různými technologiemi firmy Microsoft.
Persistence	Stálost dat i po uspaní aplikace nebo klidně i po vypnutí počítače.
Pseudoparalelní	Běžící zdánlivě paralelně na jednom procesoru.
Remote	Vzdálený
REST	Representational State Transfer – architektura pro distribuované aplikace.
RFC2617	HTTP Authentication: Basic and Digest Access Authentication – dokument popisující metody autentizace založené na HTTP protokolu. [60]
RISC	Reduced Instruction Set Computing – redukováná instrukční sada procesoru.
SOAP	Simple Access Object Protocol – standard webových služeb.
SQL	Structured Query Language – jazyk využívaný v relačních databázových systémech.
SSL	Secure Sockets Layer – vrstva zajišťující šifrování a autentizaci.
TCP	Transmission Control Protocol – spojový protokol transportní vrstvy TCP/IP.
TLS	Transport Layer Security – následovník SSL.
UML	Unified Modeling Language – modelovací jazyk.
URI	Uniform Resource Identifier – jednotný identifikátor zdroje.
VHD	Virtual Hard Disk – soubor, který má strukturu jako pevný disk. Tyto soubory jsou používány při virtualizaci.
VNT	Virtuální nutriční terapeut
VPN	Virtual Private Network – soukromá virtuální síť typicky realizovaná skrz veřejnou síť.
VPS	Virtual Private Server – virtuální server.
WAAD	Windows Azure Active Directory
WCF	Windows Communication Foundation – technologie zajišťující různé způsoby meziprocesové komunikace .
WF	Workflow Foundation

Windows RT	Verze operačního systému Windows 8, která je zaměřená pro mobilní zařízení. [15]
WinRT	WinRT je zkratka pro Windows Runtime, což je softwarová architektura obsažená v operačních systémech Windows 8 a Windows RT. [14]
WPF	Windows Presentation Foundation – technologie firmy Microsoft pro definování grafického uživatelského rozhraní.
WS-I Basic Profile 1.1	Profil webových služeb založených na SOAP. [61]
XAML	Extensible Application Markup Language – značkovací jazyk založený na XML.
XLINQ	Language Integrated Query for XML – LINQ pro XML.
XML	Extensible Markup Language – využívaný značkovací jazyk.

SEZNAM OBRÁZKŮ

Obrázek 1. Trygve Reenskaug – přejato z [8].....	14
Obrázek 2: Interakce uživatele s MVC systémem (překresleno z [11]).....	16
Obrázek 3: Interakce uživatele s MVP systémem (překresleno z [11]).....	17
Obrázek 4: MVP variace Passive View (překresleno z [11]).....	17
Obrázek. 5: MVVM (překresleno z [12]).....	19
Obrázek 6: Začlenění WinRT ve Windows [16].....	20
Obrázek 7: životní cyklus Windows Store aplikace (překresleno z [21]).....	24
Obrázek 8: Toast zobrazený v desktopové aplikaci ve Windows 8.....	27
Obrázek 9: CharmBar.....	29
Obrázek 10: Kontrakt nastavení.....	33
Obrázek 11: Zoomed out.....	35
Obrázek 12: Zoomed in.....	35
Obrázek 13. upozornění VS2012 reagující na nekompatibilní knihovnu.....	36
Obrázek 14. výběr technologií pro PCL.....	38
Obrázek 15: Příklad užití systému pro nutriční péči v rámci domova pro seniory.....	61
Obrázek 16: Příklad užití z hlediska zabezpečení.....	62
Obrázek 17: Autentizace na serveru (vytvořeno v ArgoUML).....	67
Obrázek 18: WCF klient volající WCF službu (vytvořeno v ArgoUML).....	68
Obrázek 19: Autorizace na serveru (vytvoření v ArgoUML).....	69
Obrázek 20: Získávání modulů (vytvořeno v ArgoUML).....	70
Obrázek 21: Mapování jádra systému na tabulky (vytvořeno pomocí nástroje Microsoft SQL Server Management Studio)	72
Obrázek 22: Nasazení – varianta s vlastním serverem (vytvořeno v ArgoUML).....	73
Obrázek 23: Nasazení – varianta s Windows Azure (vytvořeno v ArgoUML).....	74
Obrázek 24: Přihlašovací obrazovka.....	87
Obrázek 25: Rozcestník modulů.....	87
Obrázek 26: Detail diety.....	88
Obrázek 27: Sekce uživatelů.....	88

SEZNAM TABULEK

Tabulka 1. Kompatibilita mezi vybranými technologiemi (pozn. * značí, že je vlastnost obsažena v konkrétní technologii [29]).....	37
Tabulka 2. Stavové kódy protokolu HTTP.....	42
Tabulka 3. WCF bindingy ve Windows Store aplikaci [43].....	48
Tabulka 4. Bezpečnostní módy WCF ve Windows Store [43].....	48
Tabulka 5. Typy podporovaných ověření [43].....	48
Tabulka 6. Rozmístění datových center Windows Azure [51].....	50
Tabulka 7. Cena oprav chyb [59].....	63
Tabulka 8. Mzdy vývojového týmu.....	64