

Instant messaging založený na webových službách

Web-service based instant messaging

Bc. Ondřej Hošák

Diplomová práce
2012



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Ondřej HOŠÁK**
Osobní číslo: **A10709**
Studijní program: **N 3902 Inženýrská informatika**
Studijní obor: **Informační technologie**

Téma práce: **Instant Messaging založený na webových službách**

Zásady pro vypracování:

1. Vypracujte rešerši pro oblast instant messagingu.
2. Diskutujte výhody a nevýhody webových služeb jako komunikačního protokolu.
3. Provedte návrh protokolu pro instant messaging založený na webových službách.
4. Zpracujte analýzu požadavků na prototyp serveru a klienta.
5. Provedte vhodnou implementaci celého systému.
6. Řešte zabezpečení při přenosu zpráv.
7. Vyhodnoťte navržený systém a rámcově jej porovnejte s jinými instant messagingovými systémy.

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **SCHACKOW, Stefan. Professional ASP.NET 2.0 Security, Membership, and Role Management. [s.l.] : [s.n.], 2006. 648 s. ISBN 978-0-7645-9698-8**
2. **FERRACCHIATI, Fabio Claudio. Linq For Visual C 2008. [s.l.] : Apress, 2008. 200 s. ISBN 1430215801**
3. **CHIARETTA, Simone, NAYYERI, Keyvan . Beggining ASP.NET MVC 1.0. [s.l.] : WROX, 2009. 539 s. ISBN 047043399X**
4. **RATTZ, Joseph C. Pro LINQ : Language Integrated Query in C 2008. [s.l.] : Apress, 2007. 600 s. ISBN 1590597893**
5. **Stephen Walther, ASP.NET MVC Framework Unleashed. 2009. 723 s. ISBN 06723329980**

Vedoucí diplomové práce:

Ing. Radek Šilhavý, Ph.D.

Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce:

24. února 2012

Termín odevzdání diplomové práce:

21. května 2012

Ve Zlíně dne 24. února 2012



prof. Ing. Vladimír Vašek, CSc.
děkan



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

ABSTRAKT

Tato diplomová práce se zaměřuje na problematiku instant messagingu. Dílčím cílem této práce je analyzovat stávající IM systémy, zhodnotit jejich výhody a nevýhody. Další část se zabývá vývojem nového protokolu, který bude možné využít pro IM. Součástí diplomové práce je také implementace prototypu serveru a webového klienta pro přenos zpráv pomocí navrženého protokolu. Práce obsahuje zhodnocení a porovnání navrženého protokolu s již existujícími protokoly.

Klíčová slova: Instant messaging, ASP MVC, .NET, webová služba

ABSTRACT

This thesis is focused on instant messaging issues. Partial goal of this thesis is to analyze existing IM systems and evaluate their advantages and disadvantages. Another part deals with new protocol development that would be able to use for IM. Server prototype and web client for message transmission via designed protocol implantation is also part of this thesis. Thesis contains evaluation and comparison of designed protocol with already existing protocols.

Keywords: Instant messaging, ASP MVC, .NET, web service

Rád bych poděkoval svému vedoucímu diplomové práce panu Ing. Radku Šilhavému, Ph.D. za jeho vstřícný přístup a cenné rady při zpracovávání této práce.

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 | 10 |
| I TEORETICKÁ ČÁST | 12 |
| 1 PROTOKOLY POUŽÍVANÉ V SOUČASNOSTI | 13 |
| 1.1 SKYPE | 13 |
| 1.2 ICQ (I SEEK YOU)..... | 14 |
| 1.3 JABBER..... | 14 |
| 1.4 MSM (MICROSOFT MESSENGER)..... | 15 |
| 1.5 GOOGLE TALK | 15 |
| 1.6 OSTATNÍ PROTOKOLY | 16 |
| 2 WEBOVÁ SLUŽBA JAKO KOMUNIKAČNÍ PROTOKOL | 17 |
| 2.1 SOAP..... | 17 |
| 2.2 REST | 17 |
| 2.3 XML-RPC | 18 |
| II PRAKTICKÁ ČÁST | 19 |
| 3 POUŽITÉ TECHNOLOGIE | 20 |
| 3.1 PLATFORMA .NET..... | 20 |
| 3.2 ASP MVC .NET | 21 |
| 3.2.1 Modely | 21 |
| 3.2.2 Kontrolery | 22 |
| 3.2.3 Pohledy..... | 22 |
| 3.2.4 Popis práce ASP MVC .NET | 22 |
| 3.3 LINQ | 22 |
| 3.4 MEMBERSHIP..... | 23 |
| 3.5 MS SQL SERVER | 23 |
| 3.5.1 Jazyk SQL | 24 |
| 3.6 JAVASCRIPT..... | 24 |
| 3.6.1 JQuery | 25 |
| 3.7 KASKÁDOVÉ STYLY..... | 26 |
| 4 VLASTNÍ PROTOKOL | 27 |
| 4.1 NÁVRH NOVÉHO PROTOKOLU | 27 |
| 4.1.1 Odeslání zprávy..... | 28 |
| 4.1.2 Přijetí zprávy | 29 |
| 4.1.3 Změna nastavení..... | 30 |
| 4.1.4 Kontaktní list..... | 30 |
| 5 ANALÝZA SYSTÉMU | 32 |
| 5.1 NEFORMÁLNÍ SPECIFIKACE..... | 32 |
| 5.2 POŽADAVKY NA SYSTÉM | 33 |
| 5.2.1 Klientská část | 33 |
| 5.2.2 Serverová část | 33 |
| 5.2.3 Webová služba | 33 |

| | | |
|----------|--|-----------|
| 5.3 | INFRASTRUKTURA | 34 |
| 5.3.1 | Klient..... | 34 |
| 5.3.2 | Server | 34 |
| 5.3.3 | Webová služba | 34 |
| 5.3.4 | Společné knihovny | 35 |
| 5.3.5 | Databáze..... | 35 |
| 6 | NÁVRH SYSTÉMU | 36 |
| 6.1 | ZABEZPEČENÍ | 36 |
| 6.1.1 | Zabezpečení přenosu zpráv | 36 |
| 6.1.2 | Zabezpečení přenosu informací | 36 |
| 6.2 | USE-CASE DIAGRAMY UŽITÍ..... | 37 |
| 6.3 | NÁVRH TŘÍD | 38 |
| 6.3.1 | Klient..... | 38 |
| 6.3.1.1 | Kontrolery | 38 |
| 6.3.1.2 | Modely | 39 |
| 6.3.1.3 | Pohledy (Views) | 40 |
| 6.3.2 | Webová služba | 40 |
| 6.3.2.1 | Rozhraní služby | 41 |
| 6.3.2.2 | Třídy vnějšího rozhraní..... | 42 |
| 6.3.2.3 | Třídy pro vnitřní použití | 43 |
| 6.4 | DATABÁZOVÝ NÁVRH | 43 |
| 6.4.1 | Membership schéma..... | 44 |
| 6.4.1.1 | Tabulka aspnet_Users | 45 |
| 6.4.1.2 | Tabulka aspnet_Profile | 45 |
| 6.4.1.3 | Tabulka aspnet_Membership..... | 45 |
| 6.4.1.4 | Tabulka aspnet_Applications..... | 45 |
| 6.4.1.5 | Tabulka aspnet_Roles | 45 |
| 6.4.1.6 | Tabulka aspnet_UsersInRoles..... | 46 |
| 6.4.2 | Rozšířené tabulky specifické pro navrhovaný systém | 46 |
| 6.4.2.1 | Tabulka Message | 47 |
| 6.4.2.2 | Tabulka User | 47 |
| 6.4.2.3 | Tabulka ContactListRequest..... | 47 |
| 6.4.2.4 | Tabulka UserSettings | 47 |
| 6.4.2.5 | Tabulka ContactList..... | 47 |
| 6.5 | PRŮCHOD POŽADAVKU SYSTÉMEM..... | 48 |
| 6.6 | DIAGRAM AKTIVITY UŽIVATELE PŘI ODESLÁNÍ ZPRÁVY | 49 |
| 7 | IMPLEMENTACE SYSTÉMU | 50 |
| 7.1 | IMPLEMENTACE KLIENTA..... | 51 |
| 7.1.1 | Kontroler Home | 51 |
| 7.1.2 | Kontroler Account..... | 52 |
| 7.1.3 | Ověření identity uživatele | 52 |
| 7.1.4 | Stránka komunikace | 53 |
| 7.1.5 | Stránka kontaktní list..... | 55 |
| 7.1.6 | Stránka nastavení | 56 |
| 7.2 | IMPLEMENTACE WEBOVÉ SLUŽBY..... | 57 |
| 7.2.1 | Zpracování a obsluha požadavku | 57 |
| 7.2.2 | Přístup k databázi | 58 |

| | |
|--|-----------|
| 7.3 IMPLEMENTACE ADMINISTRACE | 59 |
| ZÁVĚR | 60 |
| ZÁVĚR V ANGLIČTINĚ..... | 61 |
| SEZNAM POUŽITÉ LITERATURY..... | 62 |
| SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK..... | 63 |
| SEZNAM OBRÁZKŮ | 64 |

ÚVOD

V dnešním světě počítačů a rychlého internetu je komunikace důležitou součástí lidské interakce. Lidé potřebují dostávat zprávy o novinkách, jak nejrychleji je to jen možné. Tímto vzniká tlak na tvorbu stále rychlejších metod pro výměnu informací.

V dřívějších dobách se lidé dorozumívali pomocí ohňů, kde se navzájem varovali před nepřítelem nebo jinou hrozbou. S rozvojem moderní civilizace se varování a výměna zpráv stále zrychluje, od ohňů jsme přešli k poště, pak k elektronické poště a nakonec k instant messaging, kde je rychlost limitována pouze rychlostí internetového spojení mezi jednotlivými komunikujícími uzly.

Tato diplomová práce je zaměřena na analýzu stávajících IM (Instant messaging) systémů, zhodnocení jejich parametrů a uživatelské základny. Dále na zhodnocení bezpečnostních rizik při používání jednotlivých protokolů. Hlavními cíly jsou analýza stávajících IM systémů a návrh nového protokolu, společně s implementací prototypu webové služby a klientské aplikace, které budou nový protokol používat.

Projekt, který jsem si zvolil realizovat jako svou diplomovou práci, by mi měl dopomoci k rozšíření mých znalostí o technologii ASP.NET (implementované v platformě .NET) s rozšířením MVC (Model-View-Controller). Jazykem pro implementaci prototypu byl zvolen C#.

První kapitola je zaměřena na analýzu stávajících IM systémů. Jsou zde rozepsány jejich přednosti a zápory. Dále je v této kapitole uvedeno srovnání jednotlivých systémů.

Druhou kapitolou je zhodnocení výhod a nevýhod webových služeb jako komunikačního protokolu.

Třetí kapitola je věnována použitým technologiím při vývoji prototypu klientské aplikace a webové služby.

Ve třetí kapitole je popis vlastního protokolu využívaný navrženým IM systémem.

Čtvrtá kapitola popisuje návrh nového protokolu. Jsou zde uvedeny diagramy pro přenos zpráv a ostatní operace. Pro každou operaci je uveden i konkrétní návrh datového schématu pro přenos.

Pátá kapitola je zaměřena na analýzu systému a uvádí přehled požadavků na navrhovaný systém. Analýza je rozdělena logicky podle jednotlivých částí prototypu.

V šesté kapitole je důkladně rozepsán návrh systému.

Sedmá kapitola popisuje implementaci systému. Jsou zde popsány jednotlivé části systému včetně implementačních detailů.

I. TEORETICKÁ ČÁST

1 PROTOKOLY POUŽÍVANÉ V SOUČASNOSTI

V dnešní době má takřka každý aktivnější uživatel internetu svůj vlastní a oblíbený IM systém. Ve valné většině uživatelů českého internetu je to *ICQ*. To je dáno do jisté míry tím, že *ICQ* bylo jedním z prvních takových systémů. Tím si získalo z počátku velkou uživatelskou základnu. Nicméně *ICQ* není vhodnou volbou zejména pro jeho nedostatky a uzavřenost. Naštěstí ale existují i jiné alternativy. V této části diplomové práce je uveden přehled nejpoužívanějších protokolů současnosti.

1.1 Skype

Skype je uzavřený systém, který vznikl v srpnu roku 2003. V roce 2011 pak došlo ke koupi *Skype* společností Microsoft. Nejedná se o čistě IM systém, ale spíše o systém, který je zaměřený na volání mezi uživateli. Pomocí *Skype* lze telefonovat i do ostatních sítí s výjimkou čísel integrovaného záchranného systému (rychlá pomoc, hasiči, police). Propojení s ostatními sítěmi, ať už pevných linek nebo mobilních telefonů, probíhá přes služby *SkypeIn* (příjem hovorů) a *SkypeOut* (odchozí hovory). Služby *SkypeIn* a *SkypeOut* jsou zpoplatněny.

Největší nevýhodou protokolu *Skype* je jeho uzavřenost, za celou dobu jeho existence se zatím nepodařilo vytvořit alternativního klienta, který by mohl být připojen do sítě *Skype*. Protokol *Skype* je necentralizovaný, z čehož plynou jeho výhody i nevýhody. Využívá peer-to-peer přenos pro zprávy i volání. Jeho necentralizovanost je výhodná v nezávislosti na serverech, které pouze ověřují uživatele, dále pak komunikace probíhá přes ostatní připojené uživatele (uzly) k systému *Skype*. Program *Skype* pak obsahuje i další funkce, které lze najít na oficiálních stránkách *Skype* <http://www.skype.com>.

U *Skype* jsou uživatelé jednoznačně identifikováni podle *Skype Caller ID*, což je jednoznačný identifikátor, který může být složen pouze z čísel a písmen. Toto ID si uživatel volí při registraci a je limitované pouze tím, aby žádný jiný uživatel neměl zadané ID již registrované.

Jelikož je *Skype* protokol uzavřený, nelze provést jeho bližší zkoumání pro potřeby této diplomové práce.

1.2 ICQ (I seek you)

ICQ bylo vyvinuto izraelskou společností *Mirabilis* již v roce 1996. O dva roky později došlo k odkoupení společností *AOL* za částku 287 miliónů amerických dolarů. V dnešní době *ICQ* umožňuje odesílat zprávy jak v online tak i offline (bez přístupu k internetu) režimu, odesílat soubory a provádět hlasové i video hovory. V neposlední řadě lze také přes *ICQ* odesílat SMS. Od roku 2006 je oficiálním partnerem v České republice pro *ICQ* portál Atlas (<http://www.atlas.cz>), který také dodává lokalizovanou verzi pro oficiálního klienta.

U protokolu *ICQ*, jako u jednoho z mála, není možné šifrovat zprávy mezi uživateli. To je jedním z hlavních důvodů, proč se nedoporučuje užívat *ICQ* zejména pro firemní komunikaci. Uživatelé jsou identifikováni podle tzv. UIN (*Unified Identification Number*). V současné době je toto číslo devítimístné, což znamená, že uživatelská základna se pohybuje v řádech stovek miliónů. UIN je přiřazeno uživateli automaticky po registraci. Registraci lze provést na neoficiálních stránkách <http://www.icq.com>.

Oficiálním klientem pro tento protokol je stejnojmenný program *ICQ* (verze 7.7). Protokol *ICQ* je sice uzavřený, ale vzhledem k jeho slabému zabezpečení se podařilo protokol analyzovat a došlo ke vzniku dalších alternativních klientů.

1.3 Jabber

Jabber jako projekt vznikl v roce 1998 Jeremiem Millerem jako open-source projekt. *Jabber* je založený na protokolu XMPP (*eXtensible Messaging Presence Protocol*) [7], který je standardizován podle normy RFC (RFC 3920, RFC 3921, RFC, 3922, RFC 3923). Filozofie *Jabber* je od ostatních protokolů odlišná, nejedná se o komerční protokol to znamená, že není provozován žádnou konkrétní společností, a proto nepodléhá jejím účelům a cílům.

Na světě existují tisíce serverů, na kterých je protokol *Jabber* provozován. Takový server si může zřídit i uživatel sám doma nebo lze využít služeb některého z větších poskytovatelů.

Identifikace uživatelů na *Jabber* je pomocí *Jabber ID* (JID). Toto ID je podobné emailové adrese – nickname@domain.tdl. Velkou výhodou *Jabber* jsou tzv. transporty. Pomocí těchto transportů je možné komunikovat s uživateli jiných protokolů. Další velkou

výhodou je uložení seznamu kontaktů na serveru. Společně s tím je možné být přihlášen na více místech současně k uživatelskému účtu.

Díky nezávislosti protokolu a možnosti šifrování komunikace je *Jabber* považován za moderní protokol. Vzhledem k tomu, že *Jabber* není provozován žádnou konkrétní společností, neexistuje pro něj také žádný oficiální klient. Existuje ale velká spousta neoficiálních. Domovskou stránkou pro Českou republiku je <http://www.jabim.cz>. Oficiální stránkou pro protokol *Jabber* je pak <http://www.jabber.org>.

1.4 MSM (Microsoft Messenger)

Protokol MSM byl uveden do světa v roce 1999 jako další z možných služeb portálu MSM. Poté došlo k přejmenování služby na Windows Live. Název MSM se vžil mezi uživateli a stále se používá. Samotný protokol MSM je opět uzavřený a pod správou firmy Microsoft. MSM lze využít pro posílání zpráv IM, hlasové i video hovory. Jeho výhodou a zároveň i nevýhodou je komerční zastřešení. Výhodou je, že o vývoj oficiálního klienta se stará výhradně firma Microsoft, nicméně jeho uzavřenost neumožňuje napojení alternativních klientů a tím jeho spojení s jiným klientem, který umí zpracovat více protokolů (např. *Jabber* nebo *Miranda*).

Jako jednoznačná identifikace uživatele v tomto protokolu slouží email, který uživatel vyplní při registraci na portálu MSM. Přeždívkou, pod kterou je také možné uživatele vyhledávat, je možné dodatečně doplnit. Oficiální stránkou je MSM je <http://explore.live.com>.

1.5 Google talk

Google Talk (zkráceně GTalk) je jedním z nejnovějších přírůstků mezi protokoly a IM systémy. Jeho vznik se datuje k roku 2005. Google Talk vychází z *Jabberu* a používá stejný komunikační protokol XMPP. Na rozdíl od *Jabberu* jsou, ale použita jako přihlašovací jména pouze emailové adresy s doménou google.com. Tedy uživatel pro přihlášení používá svůj účet na googlu nebo se musí zaregistrovat na <http://www.gmail.com>.

Tento protokol nabízí stejně pestrou paletu funkcí jako výše uvedený *Jabber*. Pro připojení k tomuto protokolu lze využít stejné klienty jako pro připojení k *Jabberu*. Oficiální klient, který je momentálně nabízen společností Google zatím nepodporuje žádné rozšířené funkce. Výhodou tohoto klienta je ale možnost využít propojení s emailovým účtem od

společnosti Google. S rozšiřováním funkcí klientské aplikace pak tedy bude možné mít jednu komplexní aplikaci, která bude sloužit pro komunikaci mezi lidmi.

Výhodou tohoto protokolu je možnost použít Google API (Application Interface), pro odesílání zpráv i jejich příjem. Google Talk je určen především těm uživatelům, kteří hledají komerčně zastřešenou implementaci protokolu XMPP.

1.6 Ostatní protokoly

Ve světě se samozřejmě nevyskytují pouze výše uvedené protokoly. Existuje poměrně velké množství dalších protokolů, mnohdy i relativně povedených. Valná většina z nich se ovšem zaměřuje pouze na konkrétní země. Velkou oblibu v tomto má zejména Polsko. Další nevýhodou jiných protokolů je poměrně malá uživatelská základna.

Příkladem ostatních protokolů může být Yahoo, jež patří stejnojmennému internetovému portálu. K zemi jej však sráží zejména špatný klient a nedostatečný okruh uživatelů. Za zmínku ještě stojí protokol AIM, který je hlavním protokolem společnosti AOL. Tato společnost vlastní i ICQ.

2 WEBOVÁ SLUŽBA JAKO KOMUNIKAČNÍ PROTOKOL

Webové služby se v dnešní době těší velké oblibě. Je to zejména pro jejich jednoduchost, snadnou implementaci a širokou podporu ze strany programovacích jazyků. Před vznikem SOA (Service Oriented Architecture) se využívalo protokolu http k odesílání dat v nestrukturované podobě nebo ve formě souboru. Teprve s rozvojem XML a vytvořením knihoven pro čtení toho typu souboru se začaly webové služby dostávat do popředí zájmu vývojářů.

Pro každý řešený problém, ale nemusejí být webové služby vhodným nástrojem, protože fungují na principu dotaz-odpověď. Nejsou tedy vhodné pro tzv. aktivní spojení. Aktivní spojení je takové, kde je vyžadován neustálý datový tok. Typickým příkladem aktivního spojení může být hlasový hovor. U takového typu služby není princip dotaz-odpověď velmi použitelný, či dokonce je přímo nevhodný.

Pro přístup k webovým službám se používají tři základní protokoly. Nejstarším z nich je XML-RPC, který dnes již není velmi používán. Dalším a nejrozšířenějším je protokol SOAP. Tento typ přenosu informací bude využit i pro potřeby navrhovaného systému. Posledním a nejmladším typem je REST, který je spíše vhodný pro přenos menších dat v rámci jedné webové aplikace (typickým použitím je javascriptové volání metody v rámci stránky).

2.1 SOAP

SOAP znamená Simple Object Access Protocol. Z názvu tedy plyne, že se jedná o protokol, který je zaměřený na přenos menších objektů. Každá webová služba, která využívá tento typ protokolu, obsahuje i wsdl (web service definition language) soubory. Tyto soubory popisují strukturu webové služby, její metody včetně parametrů a formát přenášených dat. Pomocí těchto souborů lze generovat šablony pro požadavky a odpovědi služby. SOAP má jednu nevýhodu oproti ostatním protokolům. Ta tkví v tom, že se k přenášeným datům přidává obálka a celý přenos dat je zabalen do formátu XML. To usnadňuje čtení a vytváření požadavků, podporuje i kontrolu proti chybám, avšak navyšuje kapacitu přenášených dat.

2.2 REST

REpresential State Transfer (REST) je protokol navržený pro distribuované prostředí. Hlavním rozdílem oproti protokolům SOAP a RPC je orientace datová, nikoliv

procedurální. Přenášená data tedy nejsou zatížena dalšími informacemi jako je tomu u SOAP. Tohoto je využíváno zejména u webových aplikací, kdy je potřeba asynchronně získávat data nebo odesílat malé požadavky zpět na server. Velkou výhodou REST je jeho otevřenost. Není přesně stanoveno, ve kterém z podporovaných formátů budou data odeslána zpět jako výsledek. Tento formát je zpravidla uveden v dokumentaci. Nejčastěji se pro tyto účely využívá JSON (JavaScript Object Notation) a XML, méně často pak html nebo pdf. Z těchto důvodů se protokol REST nepoužívá pro vzdálené volání webových služeb ale právě pro zmíněné posílání požadavků v rámci jedné webové aplikace.

2.3 XML-RPC

Remote Procedure Call (RPC) je označení protokolu, který slouží pro vzdálené volání procedur, jak už napovídá jeho název. Tento protokol nebyl svým způsobem nějak průlomový, spíše se jednalo o sjednocení pravidel. Tato pravidla říkají jak použít stávající a standardizované technologie pro použití RPC. Data jsou zapouzdřena pomocí značkovacího jazyku XML. Odtud XML v názvu protokolu. Požadavky tohoto protokolu jsou přenášeny přes http. Tento koncept ve své době umožňoval komunikaci různých aplikací mezi různými operačními systémy. V současné době je již vývoj tohoto protokolu zastaven a byl nahrazen novějším typem SOAP.

II. PRAKTICKÁ ČÁST

3 POUŽITÉ TECHNOLOGIE

Tato diplomová práce je implementována pomocí technologií představených firmou *Microsoft*. Volby padla na Microsoft zejména z důvodu velmi dobré technické podpory a dokumentace. Dalším důvodem je i vývojové prostředí Microsoft Visual Studio 2010, které je pro vývoj aplikací na platformě .NET nejlepším dostupným nástrojem.

3.1 Platforma .NET

O platformě .NET [13] většinou mluvíme jako o kompletním řešení pro vývoj aplikací jak webových tak i desktopových. Většinou máme na mysli její infrastrukturu. Infrastruktura se skládá celkem ze čtyř součástí. Tyto součásti jsou: .NET Framework , Microsoft Visual Studio [12] , .NET Enterprise Servers a Microsoft Windows .NET. Infrastruktura platformy zahrnuje všechny technologie, které tvoří prostředí pro vytváření a spouštění aplikací. Ta část platformy, která umožňuje vytváření aplikací, se nazývá .NET Framework.

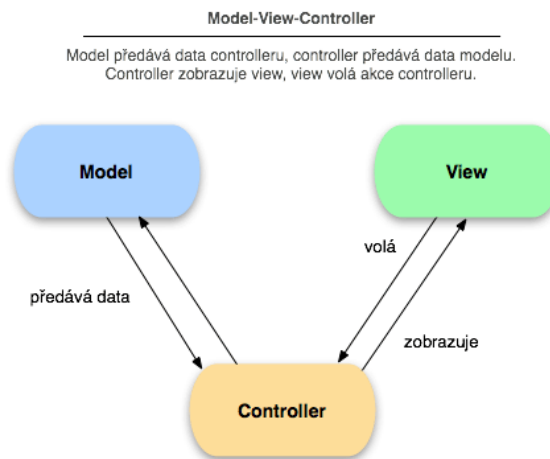
System .NET Framework je složen ze dvou částí: běhového systému CLR (*Common Language Runtime*) a knihoven tříd. Tyto knihovny tříd jsou známé pod zkratkou BCL (*Base Class Library*). Knihovna je přehledně členěna do jmenných prostorů (každý obsahuje sadu tříd) rozdělených podle použití. K této knihovně mají přístup všechny programovací jazyky. Běhový systém si můžeme představit jako virtuální stroj, ve kterém pracují aplikační funkce platformy .NET a je jádrem celé infrastruktury. V tomto běhovém prostředí mohou být bez problémů spouštěny aplikace psané v různých programovacích jazycích. Jde o princip, který je někdy nazýván „spolupráce mezi jazyky“ (*cross-language interoperability*). Všechny jazyky, které chceme využít pro vývoj aplikací běžících na CLR, musí mít překladač dodržující specifikaci CLS (*Common Language Specification*). Jde o jakýsi souhrn pravidel a pokud překladač jazyka tato pravidla nedodržuje, pak není kompatibilní s platformou .NET. Mezi jazyky dodržující CLS patří např. C#, C++ .NET nebo Visual Basic .NET.

Společnost Microsoft se snažila usnadnit práci výrobcům překladačů, a proto vyvinula „mezijazyk“ MSIL (*Microsoft intermediate language*). Pro kompilaci programů napsaných v platformě .NET je vstupem pro překladače zdrojový kód v daném jazyce a výstupem právě tento „mezijazyk“. Při prvním spuštění aplikace zajistí CLR její překlad do strojového kódu počítače pomocí překladače JIT (*Just In Time*). MSIL je tedy sám o sobě

plnohodnotný jazyk podobný assembleru, vyjma krajních situací se ale sám o sobě k programování nepoužívá.

3.2 ASP MVC .NET

Architektura MVC je návrhový vzor, který je přímo implementován firmou Microsoft. Jedná se o alternativu k ASP.NET (*Active server pages*), která slouží pro vývoj webových aplikací na platformě .NET. ASP MVC odstraňuje valnou část problémů při vývoji webových aplikací v ASP.NET. Vydání nástavby MVC se datuje k roku 2007. Nyní již existuje verze číslo 4, která přišla zároveň se čtvrtou verzí frameworku .NET. Tento návrhový vzor rozděluje projekt do tří separátních částí: modely, kontrolery a pohledy. Propojení mezi jednotlivými částmi je dobře vidět na následujícím obrázku.



Obrázek 1 – Architektura MVC

zdroj: blog.karmi.cz/assets/2000/6/17/mvc-pattern.png

3.2.1 Modely

Model je doménově specifická reprezentace informací, s nimiž aplikace pracuje. Modely obecně slouží pro uložení dat, které jsou převáděna z kontrolerů do pohledů. Kontroler vytvoří model a naplní ho daty. Případně se může model sám naplnit daty, ale to už záleží na konceptu vyvíjené aplikace. Tedy jestli bude data poskytovat pouze kontroler nebo bude mít model sám přístup k databázi a bude se moci sám plnit daty např. pomocí zadaného ID. Tento model je pak zobrazen pomocí pohledu. Při vývoji v ASP MVC platí pravidlo, že jeden pohled může zobrazit pouze jeden model. Modely jsou tedy entity, které slouží pro předávání dat mezi pohledy a kontrolery.

3.2.2 Kontrolery

Kontrolery obecně reagují na požadavky, které přicházejí do systému. V prostředí webových aplikací je to typicky reakce na http požadavek, který může a nemusí pocházet od uživatele. Přes kontrolery procházejí veškeré akce a starají se o generování výstupu respektive zvolení cílového pohledu a předání jeho modelu. Jejich hlavní funkcí, avšak ne jedinou, je zajistit změnu dat v pohledech a modelech.

3.2.3 Pohledy

Pohledy převádí data reprezentovaná modelem do podoby vhodné k interaktivní prezentaci uživateli. Ve webových aplikacích je to typicky HTML dokument.

3.2.4 Popis práce ASP MVC .NET

ASP MVC.NET pracuje na základě cest (angl. routes). Tyto cesty jsou definovány v konfiguračním souboru. Při příchodu požadavku na server, je požadavek přeměřován na cílovou aplikaci. Při startu aplikace se nejprve provede převedení požadavku na cestu. Tato cesta pak určuje, který kontroler bude obsluhovat událost a následně cesta určuje akci, kterou kontroler vykoná a také obsahuje parametry pro tuto akci. Jednotlivé cesty lze zadávat obecně i přímo specificky pro akci a její parametry. Více v následujícím příkladu. V případě, že aplikace není schopna najít pro zadaný požadavek odpovídající cestu, vrací se standardní http odpověď 404 NOT FOUND.

Ukázka jak může vypadat obecně zadaná cesta: `<controller>/<action>/<id>`. Tato cesta obecně říká, že aplikace má zkusit http požadavek rozložit podle požadovaného url na kontroler akci a parametr id.

Specifická cesta pak může vypadat následovně: `Home/Zbozi/<id>`. Tato cesta nám pak jasně adresuje kontroler Home, akci Zbozi a parametrem je id. V prohlížeči pak adresa může vypadat takto: `http://www.nejakastranka.cz/Home/Zbozi/15`.

3.3 LINQ

LINQ (*Language INtegrated Query*) [1] je technologie, která přináší objektově orientovaný přístup k neobjektovým datům. Typicky jsou tyto data SQL tabulky nebo XML soubor, ale existují i jiné možnosti. Cílem této technologie není podpora pro

specifický datový zdroj, nýbrž určení obecného přístupu k datům. Stejným způsobem se pak programují dotazy nad těmito daty.

Tato technologie pracuje s databázovými schémata nebo xml schémata. Pro tyto schémata se vygenerují proxy třídy. Přes tyto proxy třídy je pak zprostředkován přístup k datům. Vygenerované třídy má k dispozici programátor a může jejich prostřednictvím zadávat dotazy nad daty.

Tato technologie ovšem není aplikovatelná pouze pro schémata. Dotazy napsané v tomto jazyce se dají aplikovat i obecně nad jakoukoliv kolekcí, která splňuje určitá pravidla specifická pro platformu .NET.

3.4 Membership

Rozšíření Membership [5] je novinka, která byla představena v platformě .NET 2.0. Jedná se o sadu funkcí pro práci s uživatelskými účty. Zásadní výhodou Membershipu je jeho nezávislost na datovém zdroji. Připojení k datovému zdroji probíhá přes tzv. providery. Jejich podpora je rozsáhlá, existuje provider pro SQL, Oracle , AD (Active directory) a další. Případně má vývojář aplikace možnost naprogramovat si vlastní provider.

Membership zapouzdřuje veškeré operace, které jsou potřebné pro práci s uživatelskými účty, jako je vytvoření účtu, ověření přihlašovacích informací apod. Membership má velkou podporu v celé platformě .NET a lze ho použít pro propojení s rolemi nebo kontrolu uživatelů administrátorem.

Mezi další přednosti patří velká škálovatelnost nastavení. Pomocí konfigurace, která se zadává do konfiguračního souboru celé webové aplikace, lze nastavit od počtu pokusů k přihlášení (než dojde k zablokování účtu) až po minimální délku hesla a počet speciálních znaků.

3.5 MS SQL Server

MS SQL Server je databázový nástroj vytvořený firmou Microsoft umožňující ostatním programům (na lokálním počítači nebo přes síť) ukládat, zpracovávat a vyhledávat větší objemy dat. Pro vývoj IM systému aplikace je použita jeho Express verze, která je dodávána společně s instalací Microsoft Visual studio Express Edition. Express edice je volně dostupná a její součástí je SQL Server Express Edition. Při instalaci se všechny produkty obsažené v této edici samy nastaví a není potřeba žádného dalšího zásahu ze

strany uživatele. Další nastavení serveru je nutné pouze pokud by bylo potřeba mít SQL server dostupný z internetu. Express edice plně vystačí se svými omezenými funkcemi oproti placené verzi. Systém řízení báze dat MS SQL Server je spolehlivý a robustní natolik, aby mohl tvořit databázovou vrstvu aplikací podnikových informačních systémů strategického významu. Celý systém obsahuje mnoho užitečných aplikací s grafickým uživatelským rozhraním (např. SQL Management Studio) usnadňujícím vývojáři jeho práci, tyto nástroje však nejsou dostupné v jeho Express edici.

3.5.1 Jazyk SQL

V 70. letech minulého století probíhal ve firmě IBM výzkum relačních databází. Vývojáři potřebovali vytvořit sadu příkazů pro práci s těmito systémy, které by byly syntakticky podobné přirozené řeči. Vznikl tak jazyk SEQUEL, který byl později standardizován a přejmenován na SQL (*Structured Query Language*) [3]. Prvním standardizovaným jazykem bylo v roce 1986 SQL-86. Pro jeho nedostatky byl později upraven a vznikly postupně jazyky SQL-92 a SQL-99. Tento jazyk je tedy přímo navržen pro dotazy nad strukturovanými daty.

3.6 Javascript

Javascript [14] je skriptovací jazyk se základním objektovým modelem, který se interpretuje až na straně klienta. To znamená, že se jeho kód odešle společně s HTML dokumentem a jeho interpretaci zajišťuje internetový prohlížeč. Universální jádro Javascriptu je obsaženo ve všech současných internetových prohlížečích.

Tento jazyk se využívá zejména pro validaci formulářů, tzn. kontroluje obsah polí podle určitých pravidel a nepovolí odeslání formuláře, dokud nejsou všechna pole řádně vyplněna. V poslední době se také rozmáhá jeho využití pro asynchronní dotazy pro dodatečná data na server. Mezi jeho další využití lze uvést možnost dynamické změny vzhledu HTML prvků na stránce. Syntaxe Javascriptu je velice podobná jazyku C, ale jedná se o objektový jazyk, který využívá především objektů prohlížeče a zabudovaných objektů.

Využívání Javascriptu má i jednu podstatnou nevýhodu. Uživatel může zakázat svému prohlížeči interpretování Javascriptu. Proto je důležité s touto možností počítat zejména při zpracování formulářů. Je vhodné a doporučeno ošetřit vstupy od uživatele pomocí kódu, který se zpracovává na straně serveru.

JavaScript se také využívá při použití technologie AJAX (Asynchronous JavaScript And Xml) [2]. Tato technologie se používá u takových typů stránek, kde je potřeba nějak aktivně a přitom asynchronně zpracovávat uživatelské požadavky. Dobrým příkladem může být hlasování v anketě. S využitím AJAXu se pak nemusí překreslovat celá HTML stránka, ale pouze ta část, která se změnila. Tento postup šetří i síťovou zátěž, protože se odesílají a přijímají pouze dílčí části původní stránky. Nevýhodou ovšem může být, že uživatel tuto komunikaci nějakým způsobem nemůže sledovat, proto se často při použití AJAXu objevují různé dialogy, které informují uživatele o probíhající požadavku, a ten pak čeká a neodesílá další požadavky v domnění, že jeho původní požadavek byl ignorován. Další využití AJAXu spočívá v načítání dat, která jsou vázána na uživatelský vstup.

3.6.1 JQuery

JQuery [6] je javascriptová knihovna, jejíž důraz je kladen na interakci mezi HTML dokumentem a javascriptem. Jedná se o open-source projekt, který byl představen v roce 2006 Johnem Resingem. JQuery zapouzdřuje nejčastěji používané funkce javascriptu s ohledem na rozdílné chování internetových prohlížečů. Mezi největší výhody této knihovny patří tzv. selektory. Selektory se používají pro výběr DOM objektů v dokumentu, nad kterými je pak možné provádět další operace. Pro funkci selektorů byl vytvořen speciální cross-browser engine Sizzle.

S možností výběru jednotlivých DOM objektů přichází i možnost napojit události na různé akce pocházející od uživatele. Typicky kliknutí nebo označení. JQuery dále přináší rozšířené události, které lze obsluhovat u jednotlivých HTML prvků. Stejně tak je možné pomocí této knihovny manipulovat s kaskádovými styly a obecně měnit atributy jednotlivých HTML elementů. V neposlední řadě jsou součástí této knihovny i funkce pro vytváření různých animací a efektů jako jsou vysouvací panely nebo vyskakovací okna, případně rotující prvky. Velkou výhodou této knihovny je její velká uživatelská komunita, která se stará o odstraňování různých chyb, ale také o nováčky a začátečníky při práci s JQuery. Z toho důvodu jsou vedle základní knihovny k dispozici i nejrůznější rozšíření, jako jsou validační masky, dialogy pro výběry barev a mnoho dalších.

V JQuery je také zjednodušení používání a napojení AJAXových dotazů a jejich zpracování.

3.7 Kaskádové styly

Kaskádové styly [4] slouží pro vylepšení vzhledu internetových stránek, protože samotný značkovací jazyk HTML neumožňuje výrazně ovlivnit vzhled prvků na stránce a ani to není jeho cílem. Jazyk HTML poskytuje určité HTML tagy, s jejichž pomocí se dá vzhled výsledné stránky ovlivnit (BR, TABLE, H1, ...) nicméně se zpočátku vývoje internetových stránek příliš nepočítalo s grafickým ztvárněním stránek. Ovšem s rostoucí oblibou internetu bylo pro firmy a tedy i vývojáře potřeba nějakým způsobem své dílo prezentovat a upozornit na něj.

Proto se do HTML jazyka začaly postupně přidávat atributy značek, které umožnily změnit vzhled jednotlivých prvků (barva, písmo apod.). Velkou nevýhodou tohoto způsobu ale byla nemožnost nastavit hodnotu nějakého atributu pro všechny prvky určitého typu a potřeba nastavovat atributy pro každý prvek jednotlivě. To vedlo ke vzniku značně nepřehledného kódu, jehož jakákoliv úprava znamenala nemalé úsilí pro vývojáře.

Jako reakce na tyto problémy vznikly v roce 1996 CSS (Cascade Style Sheets) neboli kaskádové styly. Na počátku se jim nedostávalo velké podpory ze strany internetových prohlížečů, protože starší verze nedokázaly plně spolupracovat s touto novou technologií.

Hlavní výhodou CSS je možnost oddělení kódu (obsahu) a grafického vzhledu stránky. Kaskádové styly se připojí ke stránce pomocí direktivy `<link rel="stylesheet" type="text/css" href="cesta k souboru" />`, která se vloží do hlavičky dokumentu HTML `<head></head>`. Po připojení stylů ke stránce je možné jednotlivým prvkům na stránce přiřadit třídu, jejíž vlastnosti se určují v souboru s kaskádovými styly. Tyto soubory lze opět využít při tvorbě jiných HTML stránek.

4 VLASTNÍ PROTOKOL

Každý systém pro rychlé odesílání a přijímání zpráv mezi uživateli používá nějaký protokol pro přenos dat. V první části této práce byl popsán způsob, jakým pracují nejznámější a nejpoužívanější systémy pro přenos zpráv a jejich protokoly, pokud jsou tyto protokoly známé. Bohužel jsou tyto protokoly většinou skryté, avšak pomocí reverzního inženýrství se dá zjistit, jaká data jsou přenášena a jak jsou chráněna. Případem, kdy byl protokol takto odhalen je protokol ICQ. Jediným čistě otevřeným systémem je již zmiňovaný protokol XMPP. Proto při návrhu vlastního protokolu z něj bude vycházeno.

Protokol XMPP je velmi robustní a stavěný na decentralizovaný přístup. Z počátku návrhu i vývoje vlastního systému se zatím nepočítá s jeho decentralizací, nicméně v pozdější fázi a zejména při jeho rozšíření to bude jistě zapotřebí. Tento protokol je standardizován pomocí RFC (Request For Comments). Nabízí poměrně velkou část služeb včetně přenosu hlasu, videa a souborů.

Navrhovaný protokol se zabývá pouze přenosem zpráv a klade důraz na bezpečnost. Není ale vyloučeno, že při dalším rozšíření dojde k využití hlasu, videa nebo přenosu souborů.

4.1 Návrh nového protokolu

Každý protokol zabývající se přenosem zpráv v reálném čase potřebuje některé základní operace, aby byl použitelný. Základem je možnost přijímání a odesílání zpráv. Dále pak možnost nastavení některých parametrů (zejména v zabezpečení) a zamezení nevyžádané pošty. Při každém požadavku na server nebo webovou službu je v první řadě potřeba autorizace. Jakmile je požadavek autorizován, pak teprve dochází ke zpracování požadavku. V navrhovaném protokolu je součástí každého požadavku uživatelské jméno a heslo. Proto jsou diagramy v následujících částech zkráceny o tuto část. Protokol bude komunikovat přes webové služby, konkrétně přes protokol SOAP. U každé části je ukázka požadavku a odpovědi. Každý požadavek a odpověď má svůj specifický formát. Obecně bude formát pro požadavek obsahovat typ (požadavek nebo odpověď) akce, která se má provést a její parametry.

```
<IMerRequest>  
  <Action>  
    <Parametr1 />  
  ...  
  </Action>  
</IMerRequest>
```

Odpověď již nemá pevný formát, ale bude vždy obsahovat dva základní prvky informující o provedené akci.

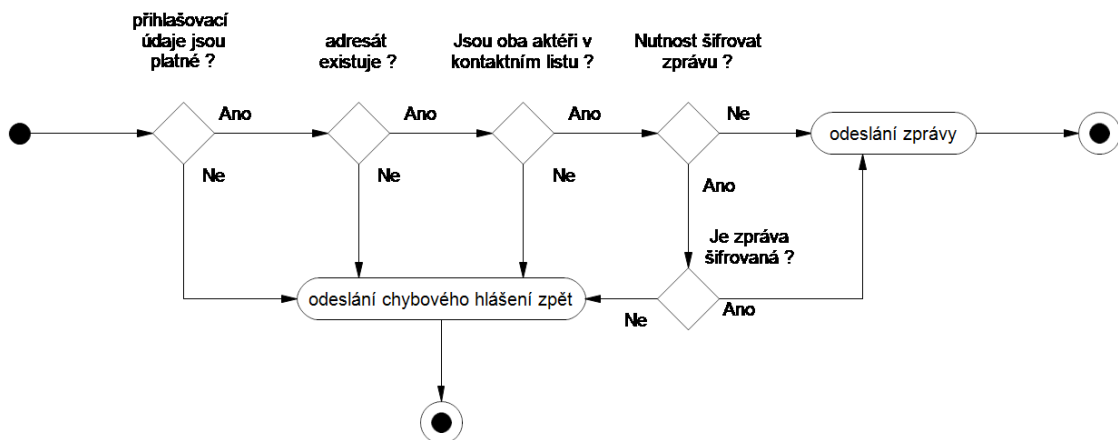
```
<IMerResponse>
  <OperationSuccess>true</OperationSuccess>
  <ErrorMessage></ErrorMessage>
  ...
</IMerResponse>
```

4.1.1 Odeslání zprávy

Pro odeslání zprávy je zapotřebí několik parametrů. Toto je reflektováno i v navrhovaném protokolu. Mezi požadované informace pro odeslání patří:

- Identifikace uživatele, který zprávu odesílá
- Identifikace uživatele, který zprávu přijímá
- Obsah zprávy

Každý z těchto parametrů obsahuje ještě dodatečné operace. Vše je názorně zobrazeno v následujícím diagramu.



Obrázek 2 – workflow protokolu: odeslání zprávy

Z obrázku je patrné, že protokol musí provést několik validací pro úspěšné odeslání zprávy a její přijetí adresátem. Tyto akce nejsou součástí přímo přenášených informací, ale jejich jednotlivý stav resp. chybový stav se přenáší zpět. Pro každou zprávu se tedy musí zkontrolovat, jestli existuje adresát a jsou navzájem propojeni přes kontaktní seznam. Toto opatření je zavedeno zejména proti nevyžádané poště. Poslední akcí je kontrola, zda je zpráva šifrována. Toto nastavení si určuje každý adresát sám. Pokud si adresát přeje

dostávat pouze šifrované zprávy, tak zprávy v nešifrované podobě nejsou přijaty. Formát dotazu pak vypadá následovně:

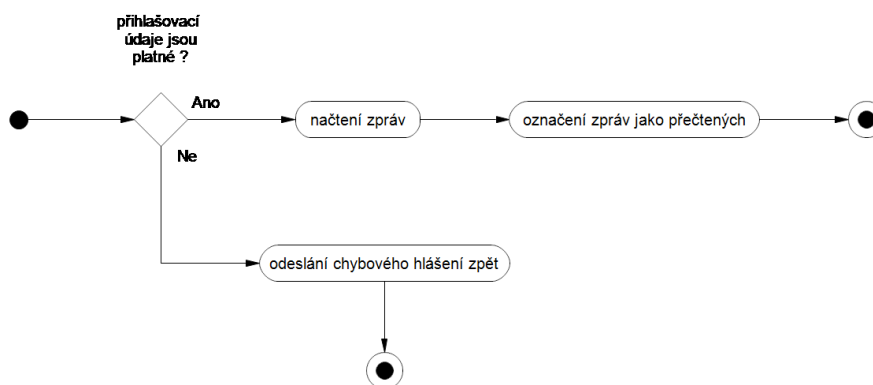
```
<IMerRequest>
  <SendMessage>
    <UserName>user</UserName>
    <Password>password</Password>
    <Receipient>receptient_guid</Receipient>
    <Text>message</Text>
    <IsCiphered>>false</IsCiphered>
  </SendMessage>
</IMerRequest>
```

Odpověď pomocí protokolu, kde došlo k chybě nenalezení uživatele, potom vypadá takto:

```
<IMerResponse>
  <OperationsSuccess>>false</OperationSuccess>
  <ErrorMessage>Receipient not found</ErrorMessage>
</IMerResponse>
```

4.1.2 Přijetí zprávy

Přijetí zprávy je nejjednodušším úkolem celého systému. Pro vyzvednutí zprávy stačí pouze ověření totožnosti uživatele. Jakmile je totožnost uživatele ověřena, zprávy jsou odeslány zpět uživateli a nastaveny jako odeslané. Popis provedených akcí je znázorněn na digramu.



Obrázek 3 – workflow protokolu: přijetí zprávy

Požadavek na příchozí zprávy:

```
<IMerRequest>
  <GetNewMessages>
    <UserName>user</UserName>
    <Password>password</Password>
  </GetNewMessages>
```

```
</IMerRequest>
```

Odpověď pomocí navrhovaného protokolu pak vypadá následovně:

```
<IMerResponse>
  <OperationSuccess>true</OperationSuccess>
  <ErrorMessage></ErrorMessage>
  <Messages>
    <Message>
      <Sender>user2</Sender>
      <SenderGuid>user2_guid</SenderGuid>
      <Text>message_text</Text>
      <DateSend>date</DateSend>
    </Message>
  </Messages>
</IMerResponse>
```

4.1.3 Změna nastavení

Změna nastavení uživatele je v prvotní fázi cílena pouze na změnu nastavení šifrování, veřejného klíče a hesla. Operace jsou členěny do dvou požadavků. První pro změnu šifrování, který obsahuje nové nastavení a případně nový veřejný klíč. Druhým je pak požadavek na změnu hesla, který obsahuje pouze nové heslo.

```
<IMerRequest>
  <ChangePassword>
    <NewPassword>new_password</NewPassword>
  </ChangePassword>
</IMerRequest>
```

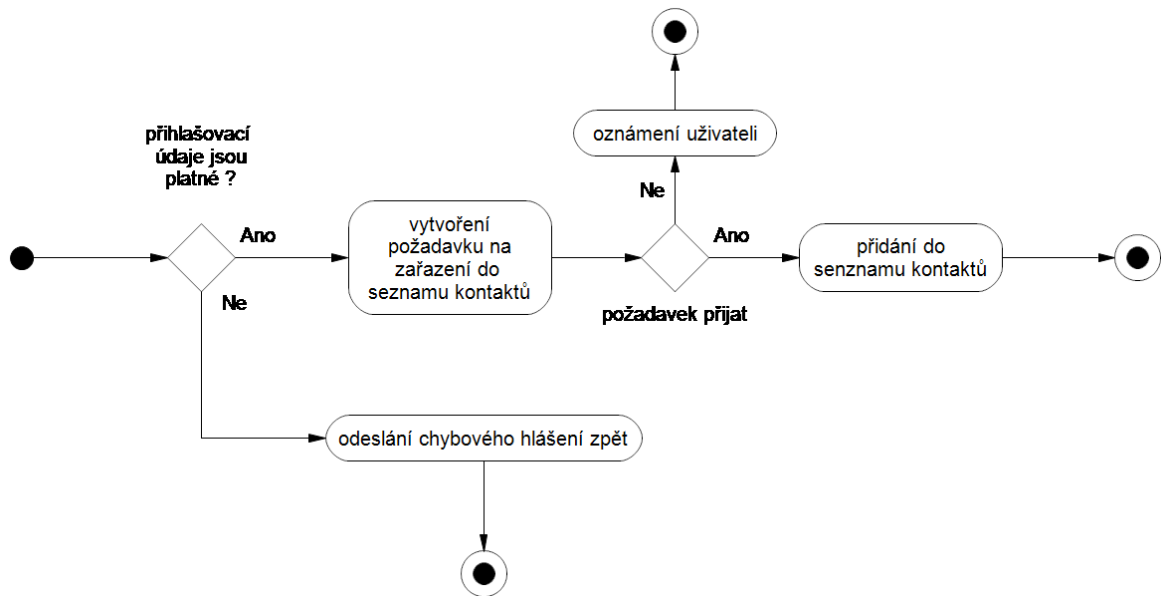
```
<IMerRequest>
  <ChangeEncryptionSettings>
    <UseCiphredMessages>true</UseCiphredMessages>
    <PublicKey>public_key</PublicKey>
  </ChangeEncrzptionSettings>
</IMerRequest>
```

Odpověď pro tento typ požadavku není nikterak speciální. Obsahuje pouze informaci o tom, zda se změna nastavení provedla v pořádku či nikoliv.

4.1.4 Kontaktní list

Akce spojené s kontaktním listem jsou rozděleny na dvě části. První částí je zaslání požadavku na přijetí do kontaktního listu. U této akce je uveden identifikátor uživatele, který je přidáván a přezdívka pod kterou bude kontakt zobrazován.

Druhou částí je pak získání všech požadavků na přidání do kontaktního seznamu. U každého požadavku se uživatel rozhodne, zda kontakt přijme či nikoliv. Pro odeslání rozhodnutí je pak speciální akce protokolu. Na následujícím diagramu je zobrazen průchod žádosti o zařazení do kontaktního listu.



Obrázek 4 – workflow protokolu: požadavek do kontaktního seznamu

Ukázka odpovědi pro potvrzení kontaktů:

```

<IMerResponse>
  <OperationSuccess></OperationSuccess>
  <ErrorMessage></ErrorMessage>
  <AddContactRequests>
    <Contact>
      <UserGuid>user_guid</UserGuid>
      <Name>user</Name>
      <Message>message</Message>
      <Date>date</Date>
    </Contact>
  </AddContactRequests>
</IMerResponse>
  
```

Potvrzení požadavku má tento formát:

```

<IMerRequest>
  <AddContactResult>
    <UserGuid>user_guid</UserGuid>
    <Name>name</Name>
    <Approved>true</Approved>
  </AddContactResult>
</IMerRequest>
  
```

5 ANALÝZA SYSTÉMU

Součástí každého rozsáhlejšího projektu by měla být i analýza systému. Tato analýza se zpravidla provádí na základě specifikace od zákazníka v několika iteracích v závislosti na použitém vzoru. Analýza toho systému je rozdělena do tří samostatných částí, protože i samotný projekt je rozdělen do tří nezávislých částí. První částí je analýza serveru, druhou pak analýza klienta a poslední pak analýza společných (sdílených) knihoven.

5.1 Neformální specifikace

Navrhovaný systém je zaměřen na rychlý přenos zpráv mezi uživateli. Celý systém se bude skládat celkově z pěti částí:

- Klientská část
- Serverová část
- Sdílené knihovny
- Webová služba
- Databáze

Důležitým aspektem, který je potřeba brát v úvahu při navrhování systému je jeho škálovatelnost a zejména pak využití sdílených knihoven pro tvorbu alternativních klientů případně i serverů. Cílem systému je masivní rozšíření a z toho důvodu je potřeba použít otevřený systém. Jiný systém by případná komunita nemusela přijmout, nebyla by pak schopna vyvinout alternativní řešení pro klienta nebo server.

Každý registrovaný uživatel v systému musí mít možnost přijímat i odesílat zprávy v textovém formátu. Dále musí mít přístup i ke zprávám, které byly poslány v minulosti. Z tohoto důvodu budou zprávy uloženy v databázi pro pozdější využití v zašifrované podobě. Zprávy budou šifrovány klíčem, který se načte z certifikátu, jenž uživatel zadá při registraci.

Uživatel bude při registraci vyplňovat pouze emailovou adresu a přezdívku. Vyhledávání uživatelů bude probíhat na základě emailové adresy, protože přezdívka nemusí být v rámci systému unikátní. Stejně tak si každý, kdo bude využívat výsledný systém, může přidávat ostatní uživatele do svého kontaktního seznamu. U každého přidaného kontaktu si uživatel může zvolit jméno, pod kterým bude kontakt uveden v kontaktním listu. Zprávy si můžou zaslat pouze dva lidé, kteří se předtím navzájem přidali do kontaktního listu.

Z důvodu zabezpečení se během registrace po uživateli požaduje i jeho certifikát, který ale nemusí být podepsaný CA (certifikační autorita).

5.2 Požadavky na systém

Z neformální analýzy plynou požadavky na výsledný systém. Celý systém bude rozdělen na klientskou a serverovou část. Každá z těchto částí má specifické požadavky, avšak obě budou využívat společných knihoven pro přístup k databázi a pro validování uživatelských vstupů.

5.2.1 Klientská část

Klientskou částí systému je myšlena ta část, ke které mají přístup pouze registrovaní uživatelé. Z neformální analýzy plyne, že uživatelé budou rozlišeni unikátně pomocí emailů a každý z nich si bude moci zvolit přezdívku, pod kterou bude vystupovat.

Dále bude umožněno odesílat, přijímat a vyhledávat starší zprávy jimi odeslané. Pro autorizaci do systému bude použito uživatelské jméno (emailová adresa) a heslo volené uživatelem při registraci. Každý účet musí mít možnost smazání a zablokování (pro nevhodné chování, při ztrátě hesla apod.).

5.2.2 Serverová část

Serverovou částí systému jsou služby a činnosti, které souvisí s obsluhou zpráv, jejich doručováním, mazáním a vyhledáváním. Stejně tak s registrací uživatelů a jejich blokováním. Serverová část systému bude mít také GUI (graphic user interface), jenž bude velmi jednoduché a určené pouze pro administrátory.

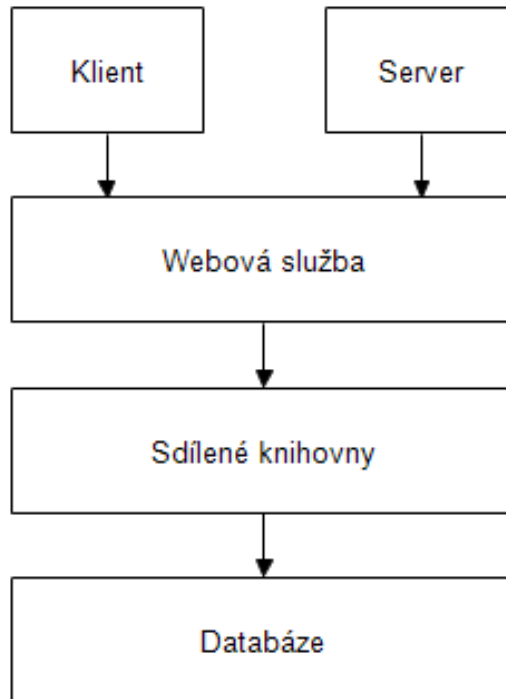
Součástí serverové části budou i dlouhotrvající služby. Jako příklad může být změna certifikátu uživatele, kdy je potřeba znovu zašifrovat všechny jeho zprávy, jak odeslané tak i přijaté.

5.2.3 Webová služba

Webová služba je ta část systému, která zpracovává veškeré požadavky od klientské i serverové části systému. Webová služba výhradně obstarává databázové dotazy a provádí změny v databázi. Komunikace s webovou službou probíhá přes protokol SOAP.

5.3 Infrastruktura

Výsledná infrastruktura systému je dobře zřetelná na následujícím obrázku. V obrázku jsou znázorněny jednotlivé části výsledného systému.



Obrázek 5 – Schéma vrstev v systému

5.3.1 Klient

Klientem se myslí část, zodpovědná za prezentování a sběr dat od uživatele. Jedná se o tenký klient, který může být formou webové nebo desktopové aplikace. V tomto je kladen důraz na přesný a silný návrh společných knihoven.

5.3.2 Server

Server bude také prezentován jako tenký klient. Opět by měl využívat pouze služeb společných knihoven.

5.3.3 Webová služba

Webová služba je samostatná část, která je spuštěna v rámci internetu, aby byla dostupná pro server a i klient. Její běh je nezávislý na ostatních systémech s výjimkou databáze.

5.3.4 Společné knihovny

Společné knihovny jsou vrstvou aplikace, kde jsou zapouzdřeny nejrůznější metody, které využívají zejména webové služby. Jsou zde metody pro validování uživatelských vstupů. Dále pak proxy vygenerované třídy, které využívá technologie LINQ pro přístup k databázi. Veškeré databázové dotazy jsou prováděny přes tuto vrstvu.

5.3.5 Databáze

Databáze je samostatná část. Jedná se klasickou relační databázi. Pro databázové schéma se vygenerují pomocí technologie LINQ proxy třídy. Využitím této technologie dosáhneme nezávislosti na typu použitého databázového stroje. Avšak musí být dodržena struktura databáze.

6 NÁVRH SYSTÉMU

Celý systém bude rozdělen do několika dílčích částí popsaných v jednotlivých podkapitolách.

6.1 Zabezpečení

Zabezpečení systému je rozděleno na dvě části. První částí je zabezpečení přenosu zpráv mezi uživateli. Druhou částí pak šifrování komunikace pro veškeré operace, které bude uživatel provádět.

6.1.1 Zabezpečení přenosu zpráv

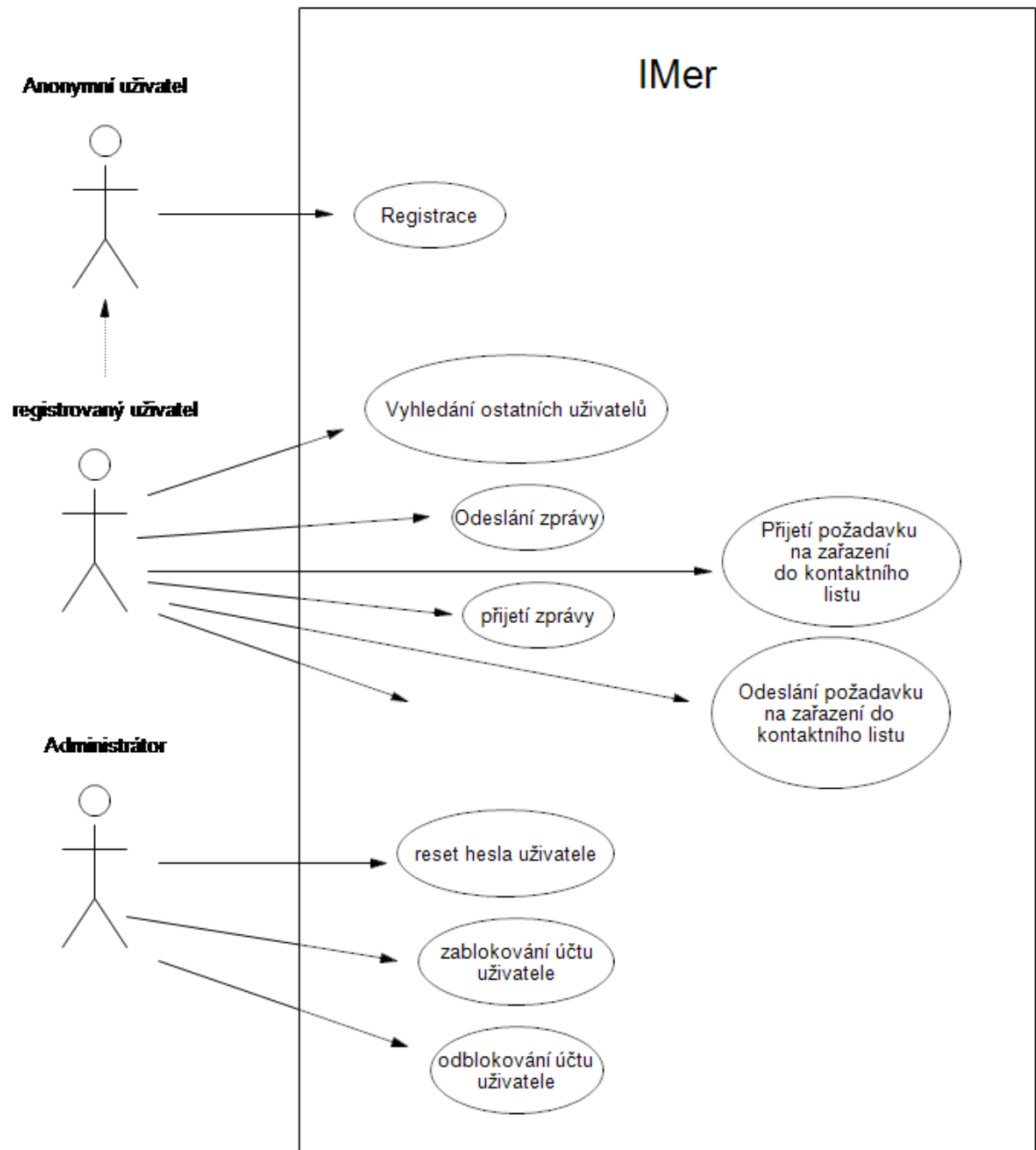
Přenos zpráv bude zabezpečen pomocí asymetrického šifrování. Každý uživatel systému si může zvolit, zda se jeho zprávy budou nebo nebudou šifrovat. Avšak je doporučeno, aby se jeho zprávy šifrovaly. Při volbě šifrování zpráv musí uživatel při registraci poskytnout svůj veřejný klíč, kterým se budou zprávy šifrovat. Pro zjednodušení se bude využívat pouze šifrovacího algoritmu RSA. Forma dodání veřejného klíče je libovolná. Doporučení je vygenerování vlastního certifikátu (nemusí být podepsán) a jeho nahrání na server. Veřejný klíč pak bude uložen v databázi. Při každém přihlášení ovšem bude muset uživatel poskytnout stejný certifikát, aby bylo možné přijaté zprávy číst.

Algoritmus RSA vnikl před více než desíti lety. I přesto je matematický základ této šifry dostačující a není v reálném čase prolomitelná. Asymetrická šifra znamená, že k šifrování a dešifrování se používá jiný klíč (privátní – soukromý a veřejný klíč).

6.1.2 Zabezpečení přenosu informací

Přenos informací mezi klientskou aplikací a webovou službou, která bude obstarávat data, bude probíhat přes protokol https. Pro tuto funkčnost musí být vystaven certifikát pro webovou službu a následně podepsán jednou z uznávaných CA. Šifrované spojení mezi klientskou aplikací a webovou službou pak zaručuje dostatečné zabezpečení proti odposlouchávání a dalším typům útoků. Pro uživatele není potřeba měnit žádné nastavení ani instalovat speciální software pro využití https.

6.2 Use-case diagramy užití



Obrázek 6 – use-case diagram užití systému z pohledu uživatele

Na obrázku 6 je znázorněn diagram užití pro různé role uživatelů v systému. Základní rolí je anonymní uživatel. Tímto uživatelem je myšlen jakýkoliv návštěvník klientské aplikace. Tato role je zde uvedena proto, že klientská aplikace umožňuje i registrování uživatelů. Předpokládá se její veřejná dostupnost pro webovou verzi. Verze pro osobní počítač by tuto akci neposkytovala, avšak obsahovala by odkaz do internetu, kde je možné registraci provést.

Další rolí je registrovaný uživatel. Tato role zastřešuje dvě entity v systému, jež jsou ekvivalentní. Jedná se o registrovaného a přihlášeného uživatele. Z pohledu aplikace nemá registrovaný uživatel možnost dalších akcí oproti uživateli přihlášenému. Pro tuto roli jsou dostupné operace pro zprávy (přijetí a odeslání), vyhledávání dalších uživatelů, jejich zařazení do kontaktního seznamu a potvrzení požadavků na zařazení do kontaktního listu od ostatních uživatelů.

Poslední rolí v systému je administrátor. Administrátor je v počátku systému určen zejména pro správu uživatelských účtů. Při rozšiřování systému se tato role uplatní i pro další účely. Cílem role administrátora je tedy resetovat uživatelům hesla. Nová hesla se pak odešlou na email, který uživatel vyplní při registraci. Dalšími akcemi pro tuto roli je blokování/odblokování některého z účtů. Zablokování účtu může být využito, pokud registrovaný uživatel bude obtěžovat ostatní uživatele hromadnými zprávami nebo dojde k napadení jeho účtu.

6.3 Návrh tříd

Návrh jednotlivých tříd systému je rozdělen podle vrstev, které je budou využívat. Obecně je cílem využívat objektově orientovaný přístup jak k datům, tak i k operacím s daty. Jelikož je jazyk C# objektově orientovaný je tento přístup nejen žádaný, ale i poměrně zjednodušen.

6.3.1 Klient

Klientská aplikace je postavena na návrhovém vzoru MVC. Je tedy implicitně rozdělena podle tohoto návrhového vzoru. Klientská aplikace je v této práci pouze pro demonstrační účely fungování systému. Proto její implementace není nikterak rozsáhlá a obsahuje pouze základní funkčnost. Nicméně připravené modely a kontrolery lze využít při vývoji desktopové aplikace nebo při rozšiřování stávající webové aplikace. Při použití pro desktopovou aplikaci by její implementace také musela vycházet z návrhového vzoru MVC. Dále jsou navrženy třídy určené pro jazyk C# v platformě .NET 4.0.

6.3.1.1 Kontrolery

Klientská aplikace bude pro základní funkčnost obsahovat pouze dva kontrolery. Při rozšiřování aplikace by samozřejmě bylo zapotřebí kontrolerů více, avšak aplikace může být obsluhována i pouze jedním kontrolerem. Rozdělení se však používá pro lepší přehlednost a pro možnost nastavení specifických požadavků pro různé kontrolery. Jako

příklad lze uvést rozdělení v prezentované aplikaci. Pro užití akcí kontroleru *Home*, musí být uživatel přihlášen. Naproti tomu pro užití akcí kontroleru *Account* uživatel nemusí být přihlášen (s výjimkou akce změny hesla).

Prvním kontrolerem je *Home*. Jedná se o základní kontroler, přes který zpracovává všechny požadavky, jenž má k dispozici registrovaný uživatel. Pro každou akci je vytvořena metoda, která je mapována přes http požadavek. Většina metod je nastavena tak, aby šel jejich vyprodukovaný výsledek převzít jazykem javascript. Klientská aplikace využívá mnoho volání asynchronního javascriptu pro odesílání a přijímání zpráv.

Druhým kotrolerem v aplikaci je *Account*. Tato část je určena pro operace s uživatelskými účty. Zejména pak pro přihlášení a registraci. Kontroler obsahuje metody pro přihlášení, registraci a změnu hesla uživatele.

6.3.1.2 Modely

Modely slouží pro zobrazování (v kombinaci s pohledy) informací uživateli v reakci na jeho požadavky. Rozdělení modelů tedy odpovídá jednotlivým akcím v aplikaci. Modely můžou sloužit pro sběr dat i pro jejich prezentaci. Příkladem může být *LogOnModel*, tento model obsahuje vlastnosti *UserName* a *Password*. V pohledu, který tento model zpracovává jsou pak určená místa pomocí speciálních značek, kde uživatel tyto hodnoty vyplní a jsou odeslány zpět na server. O transformaci dat na modely a modelů na data se stará sama platforma .NET.

Základní rozdělení je na *HomeModels* a *AccountModels*. Do *HomeModels* jsou zařazeny všechny modely, jež jsou spojeny s registrovaným uživatelem respektive s prezentováním dat přihlášenému uživateli. Naproti tomu do *AccountModels* jsou zařazeny pouze modely spjaté s akcemi pro registrování, přihlášení a změnu hesla. Mezi jednotlivými modely není pevně dána žádná hierarchie. Jsou vytvářeny podle potřeb pohledů a prezentace dat uživateli. Z toho důvodu nelze uvést žádný návrhový diagram. Všechny modely jsou prezentovány formou tříd. Následuje ukázka modelu:

```
Public class LogOnModel
{
    [Display(Name="Uživatelské jméno:")]
    [Required(Name="Uživatelské jméno je nutné vyplnit")]
    public string UserName { get; set;}

    public string Password { get; set;}
}
```

Z ukázky plyne, že pomocí modelů lze generovat i textový výstup. Tento výstup může být lokalizován do různých jazyků přes pomocí tzv. *resource files*. Lokalizace ale nejsou součástí navrhované aplikace. Dále lze vidět, že přes vlastnosti modelu lze nastavit i datový typ a určit, kde uživatel musí některou vlastnost vyplnit.

6.3.1.3 Pohledy (Views)

Pohledy jsou jedinou specifickou částí systému závislou na formě aplikace. Toto je právě velkou výhodou návrhového vzoru MVC. Teoreticky by se dalo říct, že stačí pouze vyměnit pohledy a z webové aplikace můžeme mít desktopovou, avšak to není úplně pravda a bylo by potřeba provést více změn. Protože vyvíjená aplikace je webová, jsou pohledy ve formě html stránek. Každý pohled může mít nadřazenou stránku (tzv. master). Tyto nadřazené stránky se využívají hlavně pro hlavičky a patičky stránek, které jsou shodné pro všechny stránky.

V navrhovaném systému jsou dva typy nadřazených stránek *Layout* a *Layout_login*. *Layout* slouží pro stránky dostupné po přihlášení, *Layout_login* pak pro stránky přihlášení a registrace. Pohledy jsou rozděleny podle akcí. Pro každou akci v systému by mělo být odpovídající pohled. Pohled může a nemusí zobrazovat model. Stejně tak pohled nemusí být zařazen k akci, ale může patřit ke sdíleným pohledům (typicky informace o chybě nebo potvrzení úspěšnosti odeslání informací).

V navrhované aplikaci jsou pohledy rozděleny do složek podle kontrolerů a samostatné složky pro sdílené pohledy.

6.3.2 Webová služba

Webová služba je stěžejní část navrhovaného systému, protože její služby budou využívat všechny ostatní části systému. Důraz při jejím návrhu je tedy kladen na konkurentnost a transakční zpracování požadavků. Pro tyto účely jsou již připravené nástroje v platformě .NET, které jsou implementované a zaručené firmou Microsoft, která zajišťuje platformu .NET. Není proto potřeba řešit návrh pro podporu těchto požadavků v rámci toho návrhu.

Mezi další požadavky na webovou službu patří rychlost zpracování příchozích požadavků. Zejména pro úkoly na další dobu budou využity vlákna. Posledním důležitým aspektem je stability služby. V této kritické části aplikace nemůžeme dovolit, aby některý vstup nebo operace způsobila pád služby a tudíž její nedostupnost. Toto bude zaručeno odchycením

všech akcí služby do programového bloku *try-catch-finally*. Služby tedy bude vracet odpověď i v případě výjimky. Návrh toho systému je popsán v kapitole zaměřené na třídy vnějšího rozhraní.

6.3.2.1 Rozhraní služby

Navrhovaná služba bude fungovat na protokolu SOAP. Tento protokol je charakteristický tím, že komunikace probíhá ve formátu XML. Tato forma komunikace je náročnější na přenášena data (přenáší se více dat) oproti formátu RESP, kde se přenáší pouze data. Využití SOAP je ale výhodnější pro vývoj alternativních klientů. V současné době mají již skoro všechny vyšší programovací jazyky podporu pro komunikaci s webovými službami právě přes tento formát. Tato podpora je umožněna přes soubory typu wsdl. Tyto soubory jsou typicky umístěny jako součást služby na webovém serveru, ale lze je stáhnout i samostatně. Přes tyto soubory pak lze vygenerovat proxy třídy, ze kterých se budou skládat požadavky a čteny odpovědi.

Požadavek i odpověď přes protokol SOAP je zabalen do obálky (SOAP envelope). Tato obálka je standardizována. Schéma lze nalézt na odkazu uvedeném v seznamu literatury [11]. Obsah obálky je pak složen ze dvou částí.

První částí je hlavička, která není povinná. V hlavičce jsou obsaženy dodatečné informace k požadavku a další pomocné údaje jako je hash zprávy, aby nedošlo k jejímu podvržení. Pro aplikaci je ale více důležitá další část a tou je tělo požadavku.

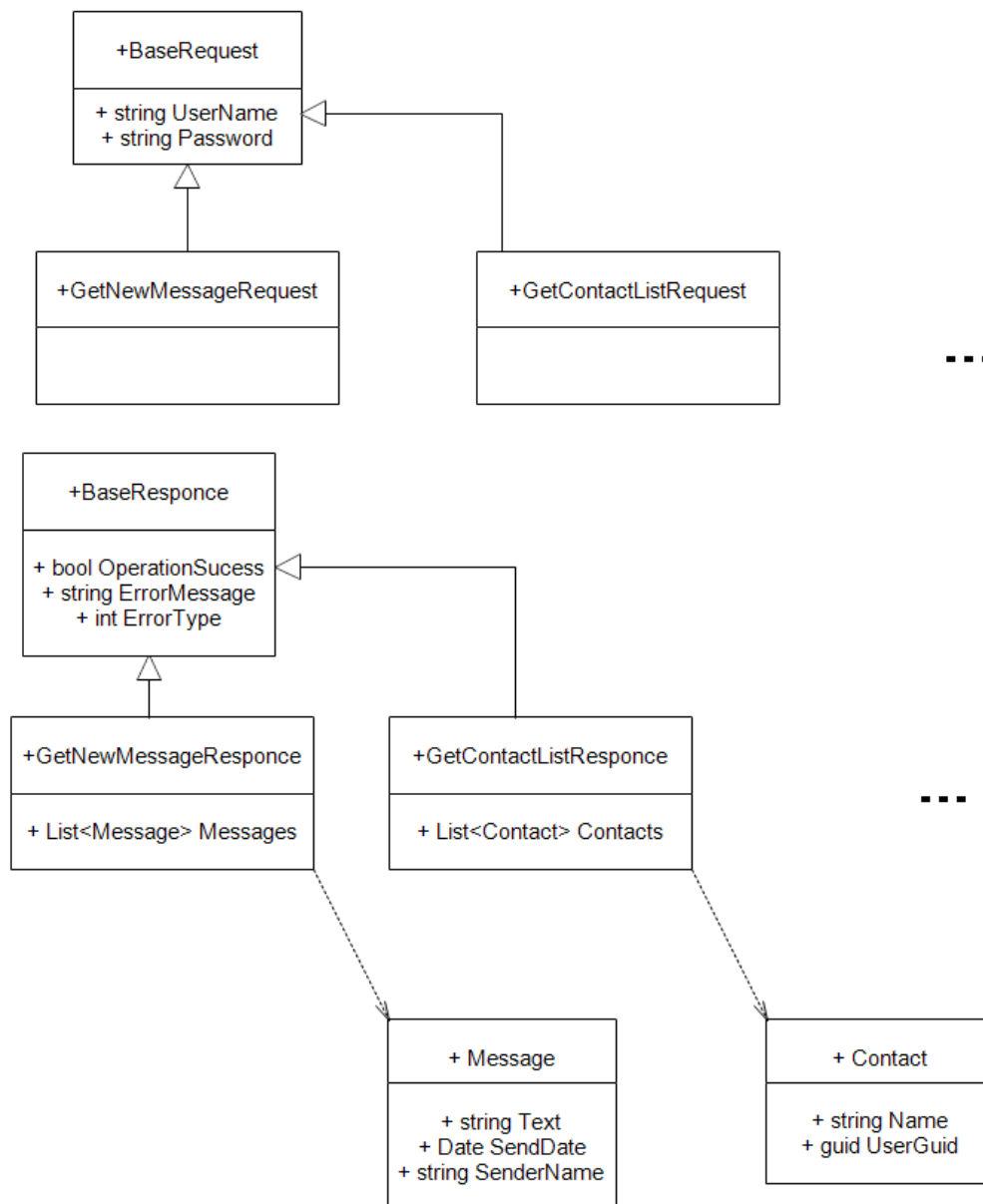
Druhou částí je tedy tělo požadavku. V této části je již samotný požadavek na metodu služby a parametry pro metodu. Následuje ukázka SOAP požadavku:

```
<soap:Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope">
  <soap:Body>
    <ValidateUser xmlns="http://imer.org/ws">
      <UserName>login</UserName>
      <Password>password</Password>
    </ValidateUser>
  </soap:Body>
</soap:Envelope>
```

Takto tedy vypadá konkrétní požadavek pro webovou službu pomocí protokolu SOAP na autorizaci uživatele. *ValidateUser* je již metoda služby a její parametry jsou *UserName* (uživatelské jméno) a *Password* (heslo). Odpověď od služby má stejný formát. Tento formát je opět umístěn v souborech wsdl.

6.3.2.2 Třídy vnějšího rozhraní

Pro třídy vnějšího rozhraní je využito dědičnosti. Každý dotaz na službu musí být autorizován, aby bylo zřejmé, pro kterého uživatele se bude požadovaná metoda provádět. Z tohoto důvodu je využito základního požadavku *BaseRequest*, který je základem pro všechny ostatní operace. *BaseRequest* třída obsahuje dvě vlastnosti, které při přenostu budou naplněny uživatelským jménem (*UserName*) a heslem (*Password*). Návrh je zobrazen na následujícím obrázku.



Obrázek 7 – hierarchie tříd pro vstupní a výstupní požadavky služby

Obdobně jako pro vstupní požadavky je pro výstupní požadavky také využito dědičnosti. Jak bylo popsáno výše, tak navrhovaná webová služba monitoruje svůj vnitřní stav. Proto

každá odpověď obsahuje stavovou proměnnou *OperationSuccess* datového typu *bool*, která určuje, zda se operace zdařila či nikoliv. Dále obsahuje informaci o chybě, pokud k ní došlo. Tyto proměnné jsou zapouzdřeny do třídy *BaseResponse*.

6.3.2.3 Třídy pro vnitřní použití

Třídy pro vnitřní použití jsou použity pro interní operace webové služby. Úkolem těchto tříd je převod mezi uživatelskými vstupy a třídami sdílené knihovny, které jsou určeny pro zápis nebo čtení z databáze. Jako příklad lze uvést třídu *Message*, která má určité vlastnosti, které jsou odesílány zpět v odpovědi. Zároveň je ale potřeba vyfiltrovat informace uložené v databázi. Všechny atributy nejsou použity v odpovědi a proto je potřeba tato převodní třídy. Pro každou třídu, která je v databázi a zároveň se vyskytuje v odpovědi nebo požadavku existuje převodní třída.

6.4 Databázový návrh

Součástí každého rozsáhlejšího systému je i databáze. V navrhovaném systému se z počátku počítá s centralizovanou databází, avšak návrh je tvořen tak, aby bylo v budoucnosti možné využít více databázových strojů a decentralizované databáze. V případě masivního rozšíření systému by totiž nebylo možné využívat pouze jedné webové služby. Její zátěž by byla neúnosná a v podstatě by docházelo DoS (Denial Of Service) útoky relevantními požadavky. Z těchto důvodů se počítá do budoucnosti o necentralizovaném systému. S tímto je velmi úzce spjat i databázový návrh, který musí umožňovat výměnu informací mezi jednotlivými databázemi.

Z těchto důvodů nejsou voleny jako primární klíče celočíselné identifikátory, ale unikátní identifikátory o velikosti 128 bitů (většinou jsou uložena jako čísla). Tyto unikátní identifikátory se nazývají GUID (Globally Unique Identifier). GUID lze generovat v rámci výkonného kódu programu nebo lze nastavit jejich generování v samotné databázi. Pro vytváření těchto identifikátorů existuje jednoznačný algoritmus, který zaručuje jejich jedinečnost.

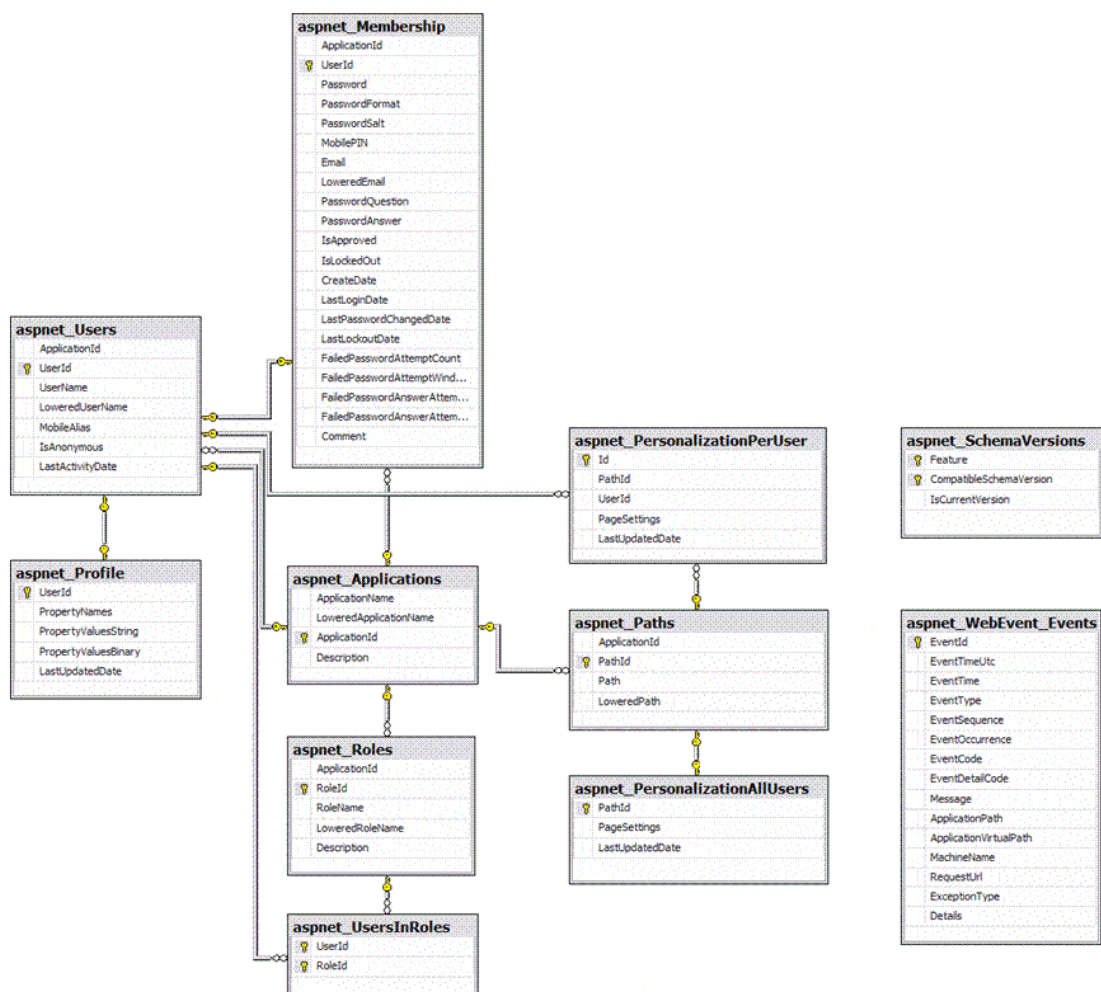
Databázové schéma je rozděleno do dvou částí. V praxi budou všechny tabulky v jedné databázi, ale jelikož se jedná o dvě samostatné části, spojené právě jednoznačným identifikátorem uživatele je vhodnější je prezentovat zvlášť.

6.4.1 Membership schéma

Membership je technologie, která zapouzdřuje operace k uživatelům a jejich účtům. Databázové schéma je poměrně rozsáhlé a připravené pro využití více databází i více aplikací. Součástí této technologie je i možnost profilování uživatelů a jejich přiřazení k rolím.

Toto uspořádání je dáno přímo firmou Microsoft. Je tedy nutné při využití membershipu dodržet přesně dané schéma. SQL skripty pro vytvoření tabulek a uložených procedur lze nalézt na stránkách společnosti.

Na následujícím obrázku je zobrazené databázové schéma technologie Membership. Následuje popis jednotlivých tabulek a jejich využití.



Obrázek 8 – Databázové schéma tabulek technologie Membership

6.4.1.1 Tabulka aspnet_Users

Základní tabulka, která obsahuje seznam všech uživatelů. Primárním klíčem je sloupec *UserID* datavého typu GUID. Dalším důležitým sloupcem je *UserName*, kde je uloženo v textové podobě uživatelské jméno. Uživatelské jméno musí být také jednoznačné v kombinaci s *ApplicationID*. *ApplicationID* určuje ze které aplikace je uživatel pořízen a pro každou aplikaci může být jen jeden uživatel s daným uživatelským jménem.

6.4.1.2 Tabulka aspnet_Profile

Využití této tabulky spočívá v profilování uživatelů, pokud je jim potřeba nastavit speciální vlastnosti nebo k nim uložit informace, které jsou specifické pro vyvíjenou aplikaci. Bohužel je implementace přístupu k těmto datům poměrně pomalá a proto se v praxi používá externí tabulka pro tyto informace. Tento typ tabulky je součástí druhého schématu. V navrhované aplikaci nebude tato tabulka využita.

6.4.1.3 Tabulka aspnet_Membership

Tato tabulka obsahuje veškeré informace o uživateli. Je zde uloženo heslo ve formátu, který je zvolen a také uložen. Je na výběr z možností prostého textu, šifrování pomocí klíče a hashování. Výchozí hodnotou je hashování. Dále je zde uložen i email, který ovšem nemusí být jedinečný (lze nastavit) a informace o stavu účtu. Zda je zablokovaný nebo ne. Membership také umožňuje odeslání hesla nebo vygenerování nového přes bezpečnostní otázku. Otázka a její odpověď jsou také uloženy v této tabulce. U účtů se také rozlišuje, zda je uzamčený a povolený. K uzamčení dojde při překročení maximálního počtu pokusů o přihlášení. Povolení účtu nastavuje administrátor.

6.4.1.4 Tabulka aspnet_Applications

Tabulka určená k uložení všech aplikací, které využívají danou instanci technologie membership. Tato tabulka nebude v první fázi implementace systému používána. Její využití přijde zejména při decentralizování jednotlivých databází. Pak bude možné jednotlivé uživatele registrovat na více serverech a pak zpětně ověřovat jejich uživatelská jména. Případně s využitím této tabulky bude možné spojovat různé databáze.

6.4.1.5 Tabulka aspnet_Roles

V navrhované aplikaci se s využitím rolí nějak zásadně nepočítá ani do dalších fází. Jediným rozdělením rolí je normální uživatel a administrátor. Pokud by bylo potřeba

využit v aplikaci role, tak k jejich uložení slouží právě tato tabulka. Role mohou být specifické podle aplikace a každý uživatel může mít jiné role v různých aplikacích (databázích).

6.4.1.6 Tabulka *aspnet_UsersInRoles*

Jedná se o vazební tabulku mezi rolemi a uživateli přes primární klíče tabulek pro uživatele a role.

6.4.2 Rozšířené tabulky specifické pro navrhovaný systém

Databázové schéma uvedené na následujícím obrázku je rozšíření tabulek technologie Membership o informace, které souvisejí s navrhovanou aplikací. Spojení s tabulkami pro uživatele je přes tabulku *User*, která je zde schematicky naznačena.

Při dalším rozvoji systému se bude upravovat a rozšiřovat pouze níže uvedená část databázového schématu. V zobrazeném schématu jsou navrženy pouze základní tabulky, aby bylo možné dosáhnout požadované funkcionality výsledného systému. Je pravděpodobné, že při dalším rozvoji dojde ke změnám i v tomto schématu.



Obrázek 9 – Databázové schéma aplikace

6.4.2.1 Tabulka Message

Základní tabulka rozšířeného schématu. V této databázové tabulce budou uloženy všechny zprávy, které si uživatelé mezi sebou odešlou. Základem je primární klíč, který bude mít datový typ GUID. Tímto není počet zpráv v systému limitován a je možné jejich spojení z více databází do jedné. Samozřejmě při převodu je nutné převést i uživatele, aby byla zachována vazba na cizí klíče. Dále je samozřejmě uložen text zprávy a to v podobě jakou má uživatel nastavenou. Posledními dvěma sloupci jsou pak jednotlivé časy odeslání a přijetí zprávy.

6.4.2.2 Tabulka User

Tato tabulka odpovídá tabulce *User* z předchozího schématu. V tomto schématu je uvedena pouze pro potřeby modelování vazeb na uživatele, jakožto cizích klíčů.

6.4.2.3 Tabulka ContactListRequest

Přes tuto tabulku probíhá rozšiřování kontaktních listů jednotlivých uživatelů. Pokud uživatel chce přidat druhého do kontaktního listu, musí nejprve poslat žádost. Po potvrzení této žádosti se pak uživatelé spárují a uloží do kontaktního listu. U každého požadavku je uložena informace o tom o které dva uživatele se jedná a který z nich žádost vyvolal. Dále je možné připojit k žádosti zprávu pro cílového uživatele. Dále je uloženo datum odeslání žádosti, rozhodnutí uživatele o přijetí/nepřijetí a datum rozhodnutí.

6.4.2.4 Tabulka UserSettings

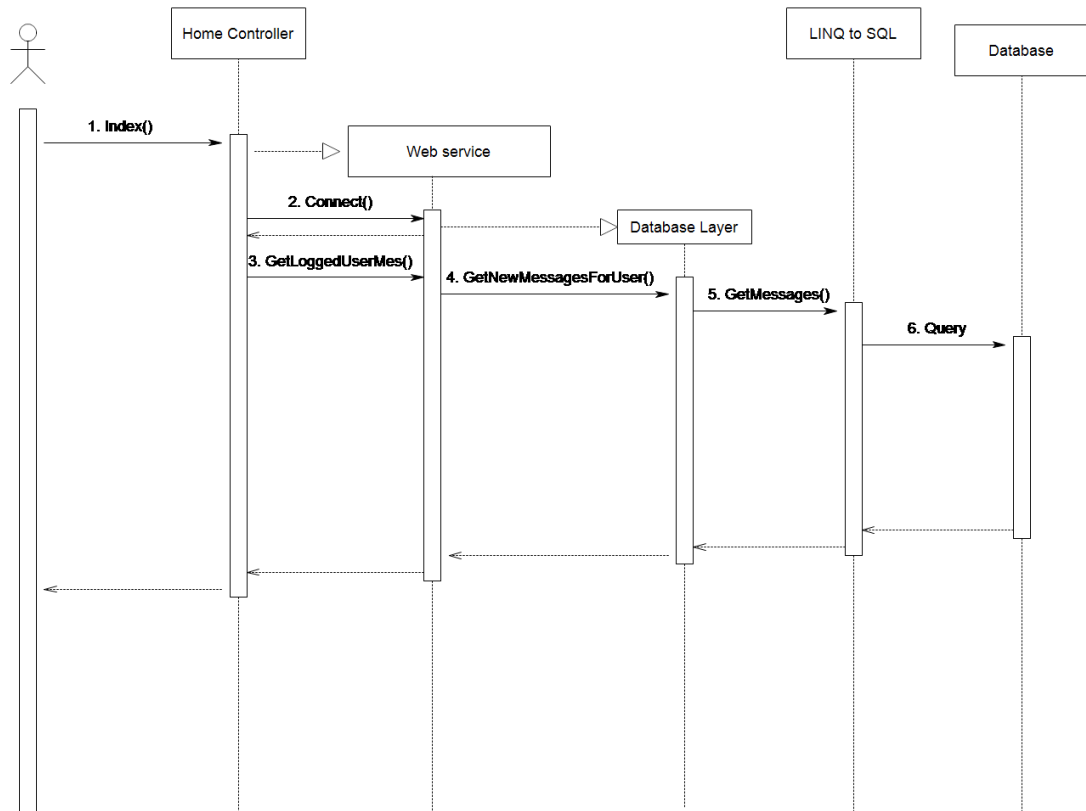
Tabulka je určena pro uložení nastavení pro konkrétního uživatele. V této fázi návrhu systému se zatím počítá s uložením příznaku o tom, jestli uživatel chce příchozí zprávy šifrovat či nechce. Pro tyto účely je zaveden i sloupec *PublicKey*, kde je uložen veřejný klíč uživatele. Tento klíč je určen k šifrování příchozích zpráv. Zejména u této tabulky se počítá s rozšířením při rozšiřování systému.

6.4.2.5 Tabulka ContactList

ContactList je databázová tabulka, kde jsou vazby mezi uživateli. V systému je možné posílat pouze zprávy uživatelům, které má uživatel ve svém kontaktním listě. Opět jsou zde vazby na tabulku *User* a to pro vlastníka kontaktního listu a jeho kontaktu. Pro každý kontakt si uživatel může zvolit jméno, pod kterým se mu bude jeho kontakt zobrazovat.

6.5 Průchod požadavku systémem

Každý požadavek v systému prochází přes několik vrstev. Vrstvy jsou strukturovány podle pořadí jejich využití. V následujícím diagramu je ukázka požadavku uživatele na nově přichozí zprávy. V diagramu je pro zjednodušení vynechána část s přihlášením uživatele.



Obrázek 10 – Průchod požadavku systémem

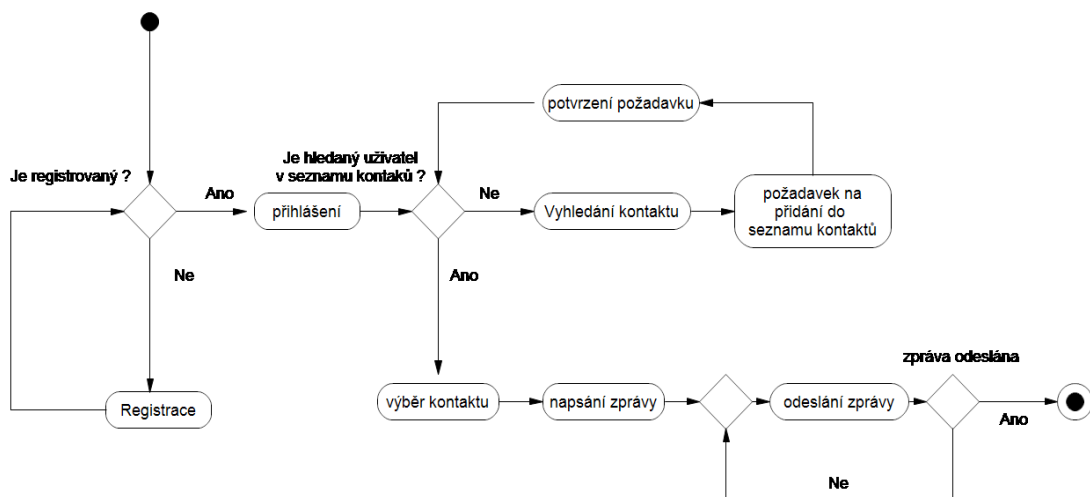
Po přihlášení se uživateli zobrazí hlavní okno, kde je možné posílat zprávy. V krátkých časových úsecích se systém periodicky dotazuje na nové zprávy. Nejprve se tedy odešle požadavek na kontroler *Home* s žádostí o nové zprávy. Kontroler obslouží požadavek zavoláním specifické metody. V rámci této metody dojde k připojení k webové službě a odeslání požadavku na připojení. Následuje odeslání požadavku na nově přichozí zprávy. Pro přečtení SOAP požadavku webová služba zavolá požadovanou obslužnou metodu. Tato metoda provede přes sdílené knihovny dotaz na nové zprávy. Ve vrstvě sdílených knihoven se dotaz transformuje na dotaz ve tvaru LINQ a poté je převeden do SQL dotazu a odeslán na databázový server. Výsledek dotazu je pak postupně předáván zpět k webové

službě, ta vytvoří SOAP odpověď a odešle ji je zpět. Kontroler tuto odpověď zpracuje a odešle zpět webovému prohlížeči.

Akce 1,2 a 3 tedy náleží klientské aplikaci. Kde 2 a 3 jsou prováděny pomocí SOAP. V pořadí 4 akce je již prováděna samotnou webovou službou, která volá služby vrstvy sdílených knihoven. Vrstva sdílené knihovny provede transformaci dotazu (akce 5) na pseudo databázový dotaz ve formátu LINQ to SQL. Šestá akce je pak již převedení pseudo dotazu na SQL dotaz a jeho vykonání na databázovém stroji.

6.6 Diagram aktivity uživatele při odeslání zprávy

Základní funkcí navrhovaného systému je odesílání a přijímání zpráv. Pro odeslání zprávy musí být uživatel přihlášený (registrovaný). Po přihlášení musí zvolit druhého uživatele, kterému chce odeslat zprávu. Pokud nemá cílového uživatele ve svém kontaktním listu, musí si ho přidat, aby mohl odeslat zprávu. Jakmile je zvolen kontakt otevře se okno, kde je možné napsat text zprávy. Tato zpráva se následně odešle, kde je testováno její převzetí. Odesílání se je zkoušeno, dokud se zpráva neodešle nebo uživatel neukončí odesílání. Celá operace je znázorněna na následujícím aktivním diagramu.



Obrázek 11 – diagram aktivity uživatele při odeslání zprávy

7 IMPLEMENTACE SYSTÉMU

Celý systém je realizován v jazyce C# na platformě .NET 4.0. Implementace je rozdělena do několika samostatných řešení. Každé řešení zastupuje nějakou část systému a je tedy možné ji spustit samostatně. Ale pro správnou práci klientské aplikace je zapotřebí mít spuštěnou webovou službu (funguje i lokálně). To stejné platí i pro serverovou (administrační) část. Celkově je tedy řešení složeno ze tří částí, z nichž některá obsahují více projektů.

První částí řešení je webová služba, která je samostatně spustitelná. Projekt s řešením sestává z několika dílčích projektů. Mezi tyto projekty patří samotná webová služba, sdílené knihovny a proxy třídy pro přístup k databázi.

Druhou částí je klientská aplikace. Tato část je sice samostatně spustitelná, ale pro její správné fungování je vyžadována i spuštěná webová služba, ze které jsou získávána data. Klientská aplikace obsahuje základní prototyp pro demonstraci komunikačních možností a základních funkcí systému. Je rozdělena podle návrhového vzoru MVC a je tedy možné ji využít pro vývoj dalších alternativních klientů (pouze v platformě .NET 4.0).

Poslední částí je pak administrace. Pro tuto část platí stejné pravidla spuštění jako pro klientskou aplikaci. V administraci je k dispozici pouze přihlášení a základní formulář pro obsluhu uživatelských účtů. U živitelských účtů je možné provést reset hesla a jeho odeslání na registrovaný email, blokování a odblokování účtu.

Při implementaci byl kladen důraz na ukázkou základní funkčnosti a zejména na možnost dalšího rozvoje systému. Samozřejmostí je dodržení objektově orientovaného přístupu k vývoji všech částí výsledného systému. Každá část systému je uzavřena do vlastního jmenného prostoru. Jmenné prostory jsou pak dále děleny ještě dle zařazení do jednotlivých projektů. Jmenné prostory a jejich názvy byly voleny tak, aby umožnily rychlé zorientování ve zdrojových kódech.

Celý projekt je umístěn do jmenného prostoru *IMer*. Tento název vzniknul zkrácením a složením slov Instant Messenger. Další dělení je pak podle použití v systému. Klientská aplikace je tedy zapouzdřena do jmenného prostoru *IMer.Client*. Webová služba do *IMer.Webservice*. Administrace je pak *IMer.Admin*.

7.1 Implementace Klienta

Klientská aplikace byla implementována na základě návrhového vzoru MVC v jazyce C# v platformě ASP MVC .NET 4.0. Aplikace je členěna do jednotlivých složek podle jejich obsahu. Kontrolery jsou obsaženy ve složce *Controllers*, modely ve složce *Models*. Pohledy jsou zařazeny do složky *Views* a dále pak členěny do dalších složek podle názvu kontroleru, pro jehož akce jsou jednotlivé pohledy užity. Pro každou akci na straně kontroleru je vytvořen speciální pohled, který prezentuje výsledek volané akce. Pohledy jsou tedy rozděleny do složek podle názvů kontrolerů. Speciální složkou je pak složka *Shared*, kde jsou zařazeny pohledy sdílené mezi kontrolery. Mezi takové pohledy patří například pohled *Master*. Tento pohled je použit pro zobrazení hlavičky a patičky na každé stránce. Každý jiný pohled pak využívá toho *Master* pohledu pro zobrazení hlavičky a patičky, aby ji nemusel obsahovat sám. Součástí projektu jsou samozřejmě také obrázky a kaskádové styly, které se nachází ve složce *Content*.

V prototypu aplikace se nacházejí tři záložky pro akce uživatele. Jsou členěny a nazvány logicky podle akcí, které na nich lze provést. Záložka komunikace tedy slouží k odesílání a přijímání zpráv. Záložka kontaktní list obsluhuje potřeby uživatele pro operace s kontaktním listem. Poslední je pak záložka nastavení, kde lze měnit nastavení klientské aplikace.

7.1.1 Kontroler Home

Základním kontrolerem je *Home*, jež má na starost vykreslování všech stránek uvnitř systému a zpracovává požadavky od aplikace i uživatele. Pro každou požadovanou akci je vytvořena metoda na kontroleru. Tyto metody mohou přijímat jednoduché parametry, jako jsou řetězce nebo celá čísla přímo pomocí mapování cest přes konfigurační soubor *global.asax*. Parametry ovšem nemusí být pouze základní datové typy nýbrž celé třídy (modely). Těch parametrů se využívá zejména při odesílání formulářů metodou POST. Pro nově přichodící zprávy existuje metoda *GetNewMessages*, pro odeslání zprávy metoda *SendMessage*. Změna nastavení se provádí pomocí metody *ChangeSettings*. Právě u této metody je využito speciálních parametrů, protože se změna odehrává pomocí formuláře. Pro operace kontaktního listu jsou metody *SendContactListRequest* (pro zobrazení požadavků) a *SendContactListResult* (pro zaslání vyjádření k požadavku).

7.1.2 Kontroler Account

Druhým kontrolerem je *Account*. Tento kontroler má na starost vykreslení stránek pro přihlášení a registraci. Dále zpracovává události spojené s těmito formuláři. Jelikož jsou obě výše zmíněné stránky formulářové je potřeba oddělit jejich zobrazení a zpracování. Na tomto kontroleru je tedy k dispozici vždy dvojice akcí *LogOn* a *Register*. Pro tyto akce platí stejná pravidla pro parametry jako u kontroleru *Home*. *LogOn* metoda tedy obsluhuje přihlášení. První implementací této metody je bezparametrické volání, které pouze nalezne požadovaný pohled a zobrazí ho zpět uživateli. Druhou implementací je volání s parametrem modelu, který je naplněn v předtím zobrazeném pohledu. Z důvodu rozpoznání volané metody je nutné k druhé implementaci připojit klíčové slovo *HttpPost*, aby bylo jednoznačně určené při jakém požadavku se má která implementace volat. Pro druhou akci *Register* platí stejná pravidla jako pro předchozí metodu, včetně parametrů s jediným rozdílem. A to v názvu.

7.1.3 Ověření identity uživatele

Jelikož klientská část systému je zabezpečená a požaduje autorizaci uživatele pro její využití, je nutné aplikaci zabezpečit proti neoprávněnému přístupu. Tento problém lze řešit několika různými způsoby, ale na platformě MVC .NET je nerozšířenějším řešením pomocí technologie *Membership*. Pro aplikaci je potřeba nastavit několik parametrů. Nejdůležitějším je *LoginPage*, kde se nastaví přihlašovací stránka. Dále je potřeba nastavit složky, kde je vyžadována autorizace a kde mají přístup anonymní uživatelé. Pro všechny složky kromě složky *Content* je nastaveno požadování autorizace. Dále lze explicitně uvést u každého kontroleru a stejně tak u každé jeho akce nastavit, zda je požadována autorizace. Toto lze provést klíčovým slovem *Authorize*. Kromě akcí pro přihlášení a registraci je tato možnost využita u všech akcí v systému. Tímto nastavením je zabezpečení aplikace kompletní a nyní se o něj stará technologie *Membership*.

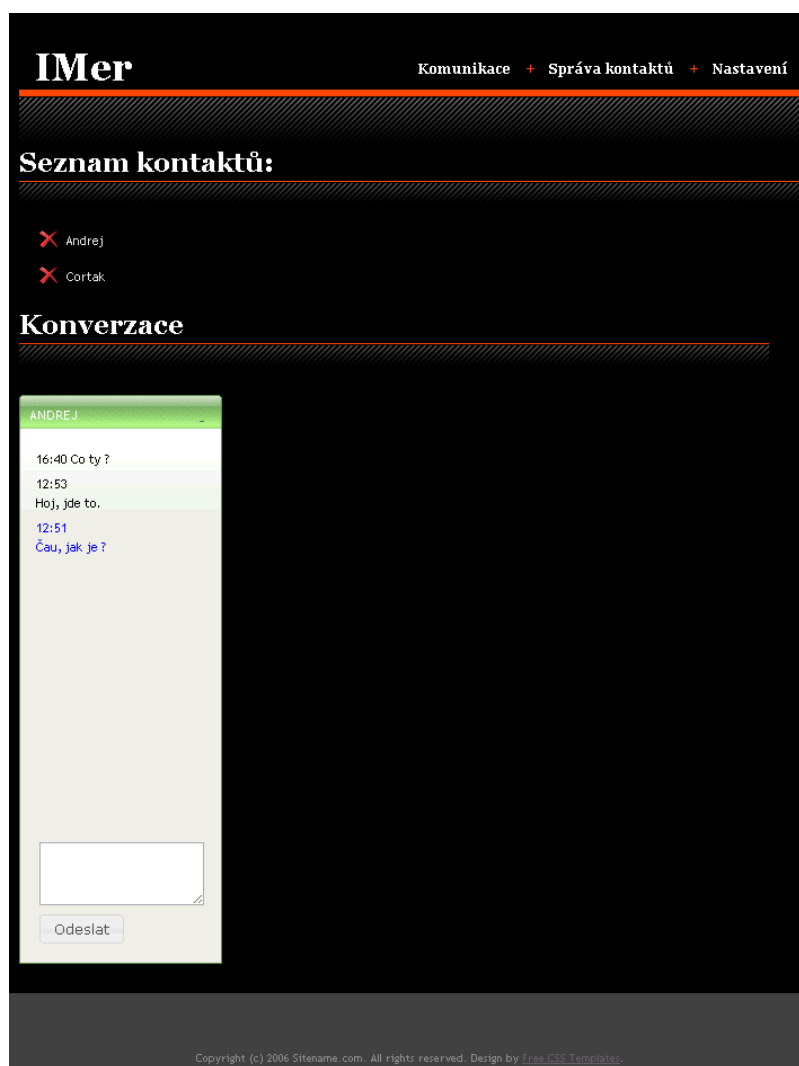
Jediným problémem technologie *Membership* je, že předpokládá databázi se svým datovým schématem dostupnou z aplikace, pro kterou obstarává zabezpečení. Často je to řešeno pouze nastavením tzv. connection string pro připojení k databázi.

Ve vyvíjeném systému ovšem není databáze k dispozici a proto je ověřování uživatelů a jejich správa řešena pomocí webové služby. Z těchto důvodů se při implementaci muselo přistoupit k vytvoření vlastního poskytovatele (provider) technologie *Membership*. Aby

mohl být nově vytvořený provider použit, musí být zděděn ze základní třídy *MemberProvider*. Děděním z této třídy je nový provider povinen implementovat základní metody pro operace s uživateli. Není nutné implementovat všechny metody, když je používaná aplikace nebude využívat. Pro potřeby systému bylo nutné implementovat pouze metody *ValidateUser* (pro ověření uživatele), *CreateUser* (pro vytvoření – registraci), *GetUser* (pro získání dat o uživateli) a *ResetPassword* (pro změnu hesla).

Každá z těchto metod má pevně dané parametry a výstup, který je nutný dodržet. Přepsáním chování těchto metod spočívalo hlavně ve změně zdroje dat. V každé z výše jmenovaných metod bylo nutné místo připojení k lokální nebo i vzdálené databázi vytvořit připojení k webové službě. Webová služba pak provedla požadované akce nebo poskytla potřebná data nebo informace.

7.1.4 Stránka komunikace



Obrázek 12 – ukázka aplikace: stránka komunikace

Na předchozím obrázku je zobrazena záložka pro komunikaci přihlášeného uživatele. Ta je ve své podstatě jedním z nejdůležitějších prvků v systému. Proto byl při jejím návrhu kladen důraz hlavně na jednoduchost, intuitivnost a bezproblémové ovládání. Celé ovládání je navrženo a implementováno s co největším využitím javascriptu resp. javascriptové knihovny JQuery.

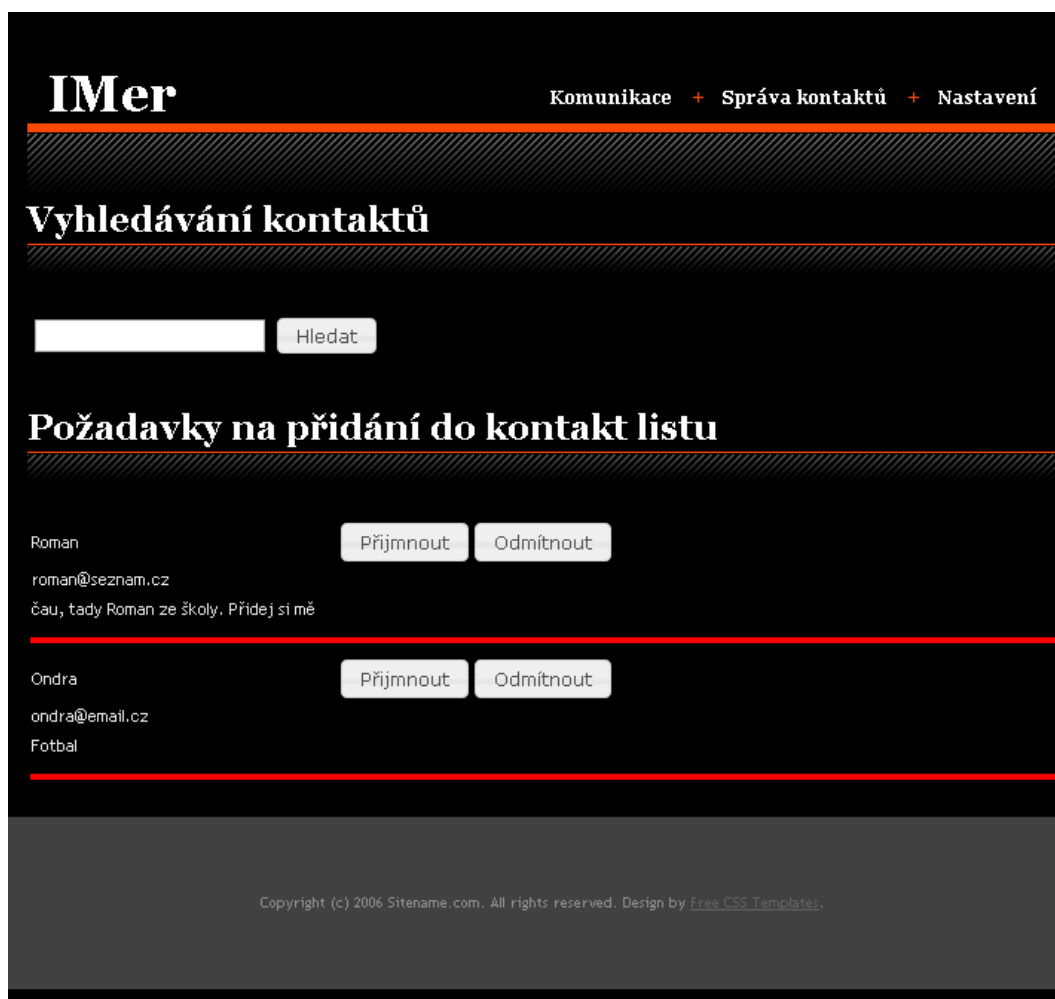
V horní části obrazovky se nachází aktuální kontaktní list se všemi spárovanými kontakty a jejich aktuálním stavem (zda jsou online či nikoliv). Kontaktní list je obnovován každých 10 sekund pomocí AJAXového volání metody na straně kontroleru *Home*. Tato metoda se přes webovou službu dotáže na kontaktní list a odešle ho zpět. Pomocí javascriptu je pak odpověď rozložena a je provedena aktualizace u jednotlivých kontaktů. Zahájení konverzace se provede dvojklikem na jméno kontaktu. Po dvojkliku na kontakt vyskočí okno komunikace nebo je nastaveno do popředí okno s tímto kontaktem, pokud existuje. Komunikaci lze zahájit s kontaktem, který není aktuálně připojený.

Hlavním doménou této stránky jsou ale jednotlivá okna pro komunikaci s vybraným kontaktem (dalším uživatelem). Pro každou instanci komunikace je vždy otevřeno nové okno. Ve vytvořeném okně je vidět název kontaktu, přehled komunikace i s časy odeslání jednotlivých zpráv. Posledním prvkem je textové pole pro odeslání další zprávy. Každé okno lze uzavřít v pravém horním rohu okna.

Odesílání zpráv je opět řešeno pomocí AJAXového volání metody kontroleru *Home*. Odeslání se provádí stiskem tlačítka „Odeslat“, kde dojde k načtení zprávy a informací o cíli zprávy. Získané informace jsou následně odeslány a je sledován stav požadavku. Pokud dojde k chybě je akce opět opakována, dokud nedojde k jejímu kladnému vyřízení nebo uživatel nezruší odesílání.

Na stránce je důležité sledovat i příchozí zprávy. Sledování příchozích zpráv probíhá přes časovač, který je nastaven na 10 sekund. Tedy každých 10 sekund se pomocí AJAXového volání dotáže stránka kontroleru *Home*, zda nejsou k dispozici nové zprávy. Pokud jsou k dispozici nové zprávy, tak jsou postupně čteny. Pro každou čtenou zprávu se nejprve ověří, jestli není otevřeno okno komunikace s kontaktem, který zprávu odeslal. Pokud takové okno existuje, tak je do něj zpráva připojena. Pokud takové okno zatím neexistuje, tak je vytvořeno nové a zařazeno na konec seznamu s okny konverzace.

7.1.5 Stránka kontaktní list



Obrázek 13 – ukázka aplikace: stránka kontaktní list

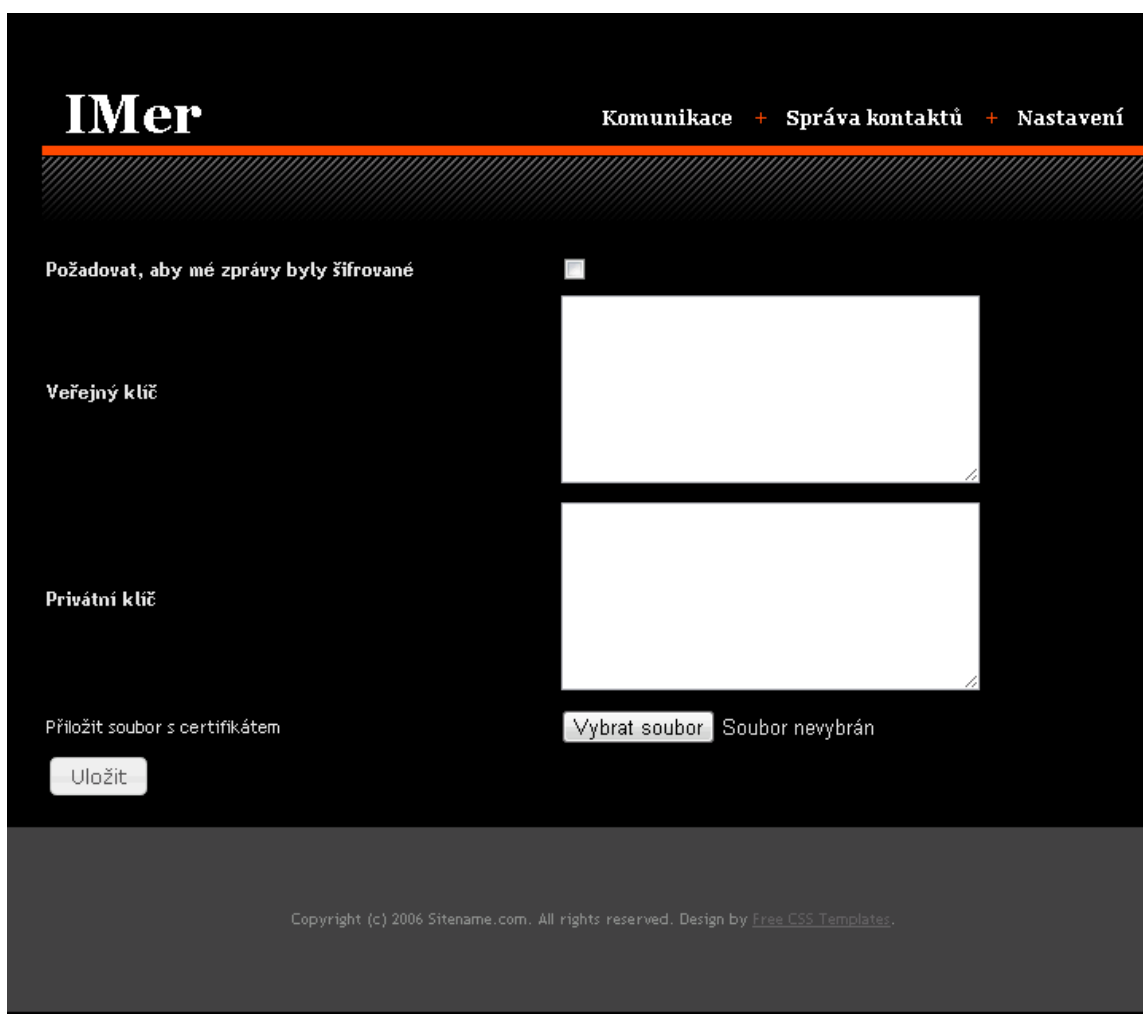
Na obrázku je vidět další záložka v systému, kterou je kontaktní list. Cílem této stránky je poskytnutí uživateli kontroly nad svým kontaktním listem. Stránka je rozdělena do dvou logických celků.

V horní části stránky lze vyhledávat nové kontakty. Kontakty lze hledat na základě dvou parametrů a to přihlašovacího jména nebo emailu. Vyhledávání je zjednodušené a nabízí pouze jedno vyhledávací pole, přes které se hledá nad oběma parametry. Výsledky jsou pak zobrazeny v tabulce, kde je údaj o uživatelském jménu a emailu. U každého takto vyhledaného kontaktu je možnost přidání stiskem tlačítka „Přidat“. Po stisku vyskočí dialogové okno. V okně je možné vyplnit přezdívku, pod kterou bude kontakt uložen a zprávu pro kontakt, aby mohl snáze rozpoznat, o koho se jedná. Oba údaje jsou nepovinné, a pokud uživatel nevyplní přezdívku, je použito uživatelské jméno.

Druhou sekcí stránky jsou pak aktuální požadavky na spárování od ostatních uživatelů. U každého záznamu je uvedeno uživatelské jméno, email a zpráva připojená při žádosti. Uživatel nevidí, jak je u druhého kontaktu uložen. Pro každý požadavek na přidání do kontaktního listu se může uživatel rozhodnout, zda požadavek na párování přijme či nepřijme.

Formulář pro vyhledávání je obslužen samostatnou metodou na kontroleru *Home* se speciálním parentrem. Po zadání vstupu se výstup přesměruje zpět na tuto stránku a je zobrazen seznam s výsledky. Akce na potvrzení nebo odmítnutí zařazení jsou implementovány pomocí AJAXového volání pro zjednodušení ovládání.

7.1.6 Stránka nastavení



Obrázek 14 – ukázka aplikace: stránka nastavení

Poslední sekcí v klientské části aplikace je záložka nastavení. Tato záložka je koncipována jako jednoduchý formulář, kde lze nastavit možnost šifrování přichozích zpráv. Pokud se

uživatel rozhodne své zprávy šifrovat, pak musí poskytnout privátní a veřejný klíč. Tyto klíče mohou být dodány v textové formě nebo ve formě certifikátu. Veřejný klíč je uložen do databáze pro pozdější užití. Privátní klíč je uložen do lokální paměti internetového prohlížeče. Pro zachování bezpečnosti je důležité po ukončení práce se systémem kompletně vypnout prohlížeč, zejména pokud je počítač používán více osobami.

Je vhodné pro aplikaci používat pouze jednu dvojici privátního a veřejného klíče, protože zprávy mohou být odeslány i pokud není uživatel přihlášen. Pokud by pak změnil své nastavení, původní zprávy nelze s novým privátním klíčem přečíst.

Celá odeslaný formulář je zpracován jednou metodou *ChangeSettings* na straně kontroleru *Home*. Po ověření dvojice privátní a veřejný klíč pomocí testovacího řetězce je nové nastavení uloženo a uživateli je oznámen výsledek. Ať kladný nebo záporný, pokud je problém v dvojici klíčů nebo s načtením informací z příloženého certifikátu.

7.2 Implementace Webové služby

Webová služba je pilířem celého vyvíjeného systému. Implementována je v jazyce C# v platformě .NET 4.0. Při implementaci byl kladen důraz zejména na stabilitu a schopnost obsloužit požadavky v co nejrychlejší čas. Rozhraní služby bylo implementováno pomocí standardu SOAP, který obaluje nově vyvinutý protokol. Celá implementace je rozdělena na tři části. První z nich je samotná služba, která obsahuje metody pro obsluhu všech navržených požadavků protokolu. Další částí je soubor tříd, které slouží jako vstupní a výstupní parametry pro metody služby. U těchto tříd je maximálně využito objektově orientovaného přístupu. Poslední částí jsou sdílené knihovny pro přístup k databázi. Přes tuto knihovnu jsou zprostředkovány všechny databázové dotazy a je navržena tak, aby byla v budoucnosti i samostatně použitelná, například při vývoji administrátorské konzole.

7.2.1 Zpracování a obsluha požadavku

Pro každý příchozí požadavek na webovou službu je potřeba nejdříve ověřit uživatele. Samozřejmě je to důležité pro ověření, jestli se má příchozí požadavek vůbec zpracovávat, ale je důležité si uvědomit, že stejně důležité je to i pro rozpoznání identity uživatele.

Každá metoda implementována v rámci webové služby tedy provede nejprve ověření identity požadavku. Tato operace je implementována samostatnou metodou, kde jsou vstupními parametry uživatelské jméno a heslo a výstupem stavová proměnná datového

typu *boolean*. Dalším důležitým výstupem je jednoznačný identifikátor uživatele (GUID), který se předává pomocí odkazu (klíčové slovo *out*).

Pokud autorizace uživatele selže, pak je využito chybných stavů v navrženém protokolu. Konkrétně dojde k nastavení příznaku *OperationSuccess* na hodnotu *false*. Vlastnost *ErrorMessage* se pak vyplní chybovou zprávou. Jestliže je autorizace provedena v pořádku, dojde k provedení požadavku. Přes datovou vrstvu se provedou potřebné změny v databázi a výsledek je odeslán zpět k uživateli.

7.2.2 Přístup k databázi

Pro přístup k databázi slouží vrstva sdílených knihoven. V této vrstvě jsou vygenerovány třídy, které reprezentují databázové tabulky. Je to zprostředkováno technologií LINQ To SQL. Pro každou tabulku je tedy vygenerována třída se shodným názvem, jako je název tabulky. Do této třídy jsou převedeny všechny sloupce jako vlastnosti se shodnými datovými typy jako jsou v definici tabulky. Výhodou toho objektového přístupu je možnost využití silné typové kontroly pro ukládaná data a pro kontrolu vstupních dat od uživatele. Jediným úskalím využití technologie je LINQ je přímá závislost vytvořených tříd na databázových tabulkách. Při jakékoliv změně v databázovém schématu je tedy nutné znovu vygenerovat třídy pro tabulky, které se změnily. Nelze tedy vytvářet nové tabulky za běhu programu a pak k nim mít zprostředkovaný objektově orientovaný přístup technologií LINQ.

Pro každý databázový dotaz je nejdříve nutné otevřít databázový kontext. V některých aplikacích se dá využít návrhového vzoru *singleton*, kde je vytvořena pouze jedna instance databázového kontextu a přes tuto instanci jsou směřovány všechny dotazy, aktualizace nebo odstranění řádku v tabulce. Implementace databázové kontextu přes *singleton* ale není příliš vhodná pro webovou službu. Při využití toho vzoru pro webové služby by docházelo k nekonzistenci v databázi a zdlouhavému čekání na obsloužení předchozích požadavků.

Z výše uvedeného důvodu je pro každý databázový dotaz vytvořen nový kontext pomocí klíčového slova *using*. Tímto zaručíme, že každý databázový dotaz bude mít konzistentní pohled na databázi. Využití takového přístupu je náročnější na použitou paměť a větší zátěž na databázový stroj. Tyto zvýšené požadavky ale nejsou natolik zásadní, aby bylo kompaktnějšího a bezpečnějšího přístupu k databázi použito.

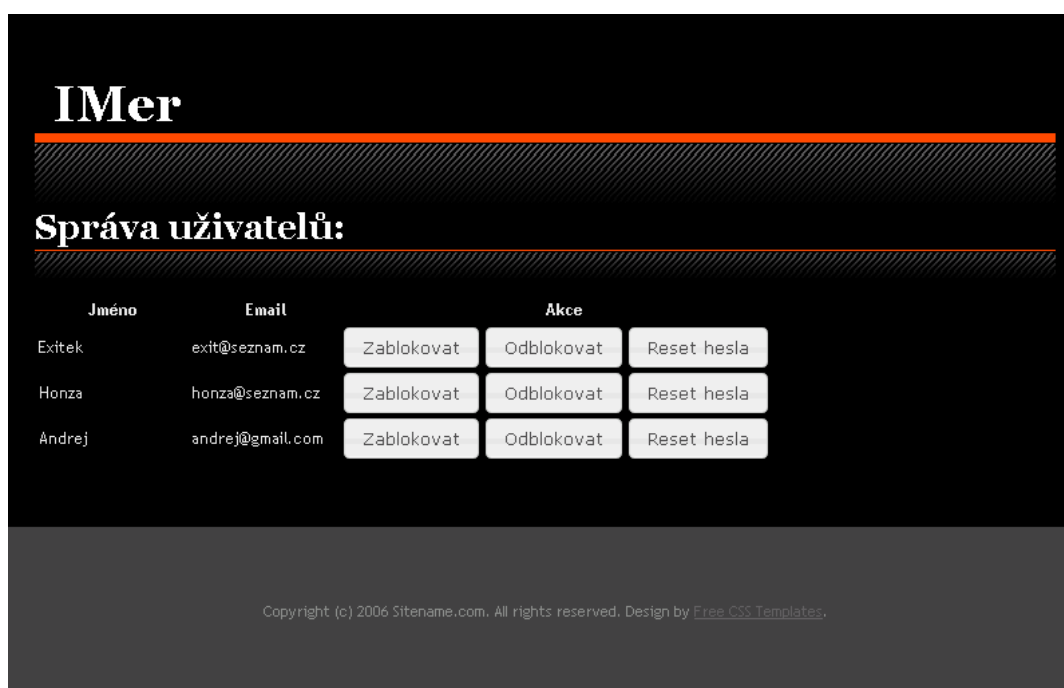
7.3 Implementace Administrace

Administrace je důležitou součástí každého fungujícího systému. Jelikož je systém centralizovaný není zatím potřeba žádná rozsáhlá administrace systému. Která by poskytovala převod uživatelů nebo globální nastavení systému.

Pro účely prototypu systému stačí pouze administrace uživatelů. U každého uživatele je tedy možné jeho účet zablokovat, odblokovat nebo mu resetovat heslo a nové heslo odeslat na vyplněný email. K zablokování účtu dochází při vyčerpání možností oprav u zadávání hesla. Toto je poměrně běžný problém u uživatelů, proto je důležité mít možnost účet znovu aktivovat.

Přehled všech uživatelů je koncipován do tabulky. Pro každého uživatele systému jsou k dispozici zmiňované akce. U administrace není tak důležité dbát na uživatelsky přátelské ovládání. Proto jsou pro jednotlivé akce vytvořeny metody, které jsou volány pomocí formuláře, na rozdíl od implementace v klientské aplikaci.

Administrace je samostatný projekt, který je také naprogramovaný v jazyce C# v platformě MVC .NET 4.0. Na následujícím obrázku je ukázka jediné obrazovky z administrace systému. Samozřejmostí je nutnost přihlášení do systému pod takovým účtem, který má administrátorská práva. Část aplikace nutná pro přidávání uživatelů mezi administrátory byla z prototypu vypuštěna.



Obrázek 15 – ukázka aplikace: administrace

ZÁVĚR

V teoretické části této diplomové práce jsou podrobně popsány nejznámější protokoly pro IM. V první řadě jejich nedostatky sloužily jako inspirace pro návrh vlastního protokolu. Na tuto problematiku navazovalo navržení nového systému pro IM, založeném na webové službě.

Prototyp vytvořeného IM systému umožňuje v současné verzi pouze základní funkce pro demonstraci použití. Ve srovnání s existujícími systémy nabízí pouze možnost odesílání a přijímání zpráv. Tyto zprávy je ale možné šifrovat, což jako jediný ze srovnávaných protokolů umožňuje *Jabber*. Pro masivní rozšíření nového protokolu by bylo potřeba rozšířit funkcionalitu zejména o možnost přenosu souborů a hlasových hovorů. Tyto dvě funkce poskytují všechny protokoly zmíněné v úvodu této diplomové práce. Mezi další rozdíly mezi popsány protokoly a nově navrženým je necentralizovanost. Jediným centralizovaným protokolem, stejně jako navržený protokol, je *ICQ*, ostatní jsou necentralizované. Nicméně vytvořený protokol byl navržen s ohledem na možnost v budoucnu pracovat jako necentralizovaný.

Jelikož je prototyp systému omezen pouze na odesílání a přijímání zpráv, tak je jeho využití určeno spíše pro menší lokální síť nebo malé firmy. Prototyp je navržen tak, aby bylo možné navázat na jeho vývoj nebo použít navržené jádro klientské aplikace pro vytvoření dalšího alternativního klienta.

Výsledkem této práce je tedy funkční prototyp systému pro přenos textových zpráv přes síť (lokální i internet) pomocí webové služby. Prototyp využívá nově navržený protokol, který umožňuje přenos těchto zpráv a další služby mezi které patří například registrace uživatele, změna uživatelského nastavení a operace s kontaktním listem.

Mezi hlavní přínosy při vypracování této diplomové práce patří hlubší seznámení s platformou .NET, využití nejnovějších technologií jako je jQuery nebo MVC.NET a také implementace webové služby.

ZÁVĚR V ANGLIČTINĚ

In the theoretical part of this thesis are described in details the most popular IM protocols. At first place their shortcomings have served as inspiration for the design of own protocol. This issue was linked to design a new system for IM based on web service.

Current version of the IM system allows only basic functionality for demonstration purposes. It offers only sending/ receiving messages in comparison with already existing systems. It is possible to encrypt these messages, which only *Jabber* supports. It would need extended functionality (file transfer and voice calls in particular, which all protocols mentioned in the abstract of this thesis support) for massive expansion. Among, other differences between studied and newly created protocol belongs decentralization. Only centralized protocol, as well as designed, is *ICQ* – others are decentralized. However, newly created protocol was designed to be used in future as decentralized.

Since the prototype system is limited to sending and receiving messages, its use is intended rather for smaller local networks or small businesses. The prototype is designed to coincide with the development or use of its core client application to create another alternative client.

The result of this work is a functional prototype of the system for transmission text messages over a network (local or internet) via web service. The prototype uses a newly designed protocol that allows transmission of these messages and other services like user register, changing user settings or operations with contact list.

Main benefits in the development of this thesis are a deeper acquaintance with the .NET platform, using the latest technologies such as jQuery or MVC.NET and web service implementation.

SEZNAM POUŽITÉ LITERATURY

- [1] RATTZ, Joseph C. Pro LINQ : Language Integrated Query in C# 2008. [s.l.] : Apress, 2007. 600 s. ISBN 1590597893
- [2] GIBS, Matt, WAHLIN, Dan. Professional Asp.Net 2.0 Ajax. [s.l.] : Wiley India, 2007. 328 s. ISBN 8126513241
- [3] VIEIRA, Robert. Professional Microsoft Sql Server 2008 Programming. [s.l.] : John Wiley & Sons Inc, 2004. 1000 s. ISBN 0470257024
- [4] Jak Psát Web [online]. 2009 , 24.3.2012 [cit. 2012-03-24]. Dostupný z WWW: www.jakpsatweb.cz
- [5] SCHACKOW, Stefan. Professional ASP.NET 2.0 Security, Membership, and Role Management. [s.l.] : [s.n.], 2006. 648 s. ISBN 978-0-7645-9698-8
- [6] jQuery [online]. 1.5.2012 [cit. 2012-05-01]. Dostupný z WWW: www.jquery.com
- [7] XMPP protocol [online]. Cit. 1.5.2012. Dostupný z WWW: xmpp.org
- [8] FERRACCHIATI, Fabio Claudio. Linq For Visual C 2008. [s.l.]: Apress, 2008. 200 s. ISBN 1430215801
- [9] CHIARETTA, Simone, NAYYERI, Keyvan. Beginning ASP.NET MVC 1.0.[s.l.]: WROX, 2009. 539 s. ISBN 047043399X
- [10] STEPHEN, Walter. ASP.NET MVC Framework Unleashed. 2009. 723 s. ISBN 06723329980
- [11] Schéma protokolu SOAP [online]. Cit. 1.5.2012. Dostupný z WWW: schemas.xmlsoap.org/soap/envelope/
- [12] SHARP, John. Microsoft Visual C# 2010. Computer press, 2010. 696 s.
- [13] NAGEL, Christian, EVJEN, Bill, GLYNN, Jay, WATSON, Karli, SKINNER, Morgan. Professional C# 4.0 and .NET 4. WROX, 2010. 1536 s. ISBN 978-0-470-50225-9
- [14] ZAKAS, Nicolas. JavaScript pro webové vývojáře. Computer press, 2019. 832 s. ISBN 978-80-251-2509-0

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|-------|---|
| IM | Instant messaging – rychlý přenos zpráv po síti, typicky ICQ. |
| UIN | Unified Identification Number |
| GUI | Graphic user interface |
| HTTPS | Hyper text transfer protokol s podporou SSL vrstvi pro šifrování komunikace |
| SOAP | Simple Object Access Protocol – jedná se standardizovaný protokol pro přenos informací přes webové služby. |
| CA | Certifikační autorita |
| SOA | Service Oriented Architecture – architektura projektu, která je zaměřená na webové služby jako hlavní zdroj dat. |
| MVC | Model – View - Controller. Jeden ze známých návrhových vzorů. |
| RFC | Request For Comments – označení řady standardů a norem pro většinu internetových protokolů |
| SSL | Secure socket layer – jedná se o vrstvu v síťové komunikace, kde je přenášena informace chráně pomocí asymetrického šifrování |
| DOS | Denial Of Service – typ útoku na webový server. Spočívá v zahlcení požadavky a nedostupnosti požadované služby. |
| GUID | Globally Unique IDentifier – jednoznačný identifikátor v rámci světa. Jedná se 128 bitů velkou strukturu, složenou z času a dalších proměnných. |

SEZNAM OBRÁZKŮ

| | |
|---|----|
| Obrázek 1 – Architektura MVC..... | 21 |
| Obrázek 2 – workflow protokolu: odeslání zprávy | 28 |
| Obrázek 3 – workflow protokolu: přijetí zprávy | 29 |
| Obrázek 4 – workflow protokolu: požadavek do kontaktního seznamu | 31 |
| Obrázek 5 – Schéma vrstev v systému | 34 |
| Obrázek 6 – use-case diagram užití systému z pohledu uživatele | 37 |
| Obrázek 7 – hierarchie tříd pro vstupní a výstupní požadavky služby | 42 |
| Obrázek 8 – Databázové schéma tabulek technologie Membership | 44 |
| Obrázek 9 – Databázové schéma aplikace..... | 46 |
| Obrázek 10 – Průchod požadavku systémem | 48 |
| Obrázek 11 – diagram aktivity uživatele při odeslání zprávy | 49 |
| Obrázek 12 – ukázka aplikace: stránka komunikace | 53 |
| Obrázek 13 – ukázka aplikace: stránka kontaktní list..... | 55 |
| Obrázek 14 – ukázka aplikace: stránka nastavení | 56 |
| Obrázek 15 – ukázka aplikace: administrace..... | 59 |