VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

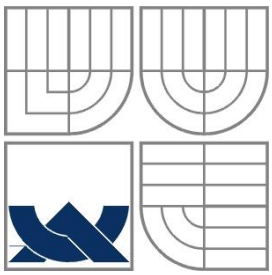
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MIKROBLOG PRO TÝMY

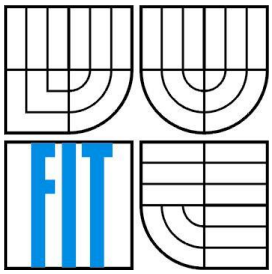
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARTIN ROUBAL
AUTHOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MIKROBLOG PRO TÝMY

TEAM-BASED MICROBLOGGING SERVICE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
MARTIN ROUBAL

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. MICHAL ZACHARIÁŠ

BRNO 2013

Abstrakt

Tato bakalářská práce pojednává o analýze, návrhu a implementaci jednoduchého mikroblogu pro týmy. Hlavním cílem této webové aplikace je usnadnit komunikaci a sdílení informací mezi členy týmu. Aplikace je založena na návrhovém vzoru MVC a implementuje grafické uživatelské rozhraní s rozvržením typickým pro existující řešení.

Abstract

This bachelor's thesis deals with the analysis, design and implementation of a lightweight team-based microblogging service. The main purpose of this web application is to simplify communication and information sharing among team members. The application is based on the Model-View-Controller software architecture pattern and implements the graphical user interface with the typical layout used by existing solutions.

Klíčová slova

mikroblog, webová aplikace, ASP.NET, MVC, C#, Razor

Keywords

microblogging service, web application, ASP.NET, MVC, C#, Razor

Citace

Roubal Martin: Mikroblog pro týmy, bakalářská práce, Brno, FIT VUT v Brně, 2013

Mikroblog pro týmy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Zachariáše.
Další informace mi poskytl pan Ing. Rudolf Kajan.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Roubal
15. května 2013

Poděkování

Dovoluji si poděkovat vedoucímu bakalářské práce panu Ing. Michalu Zachariášovi za poskytnuté rady a čas, který mi věnoval, a panu Ing. Rudolfu Kajanovi za náměty a tipy ke zpracování.

Obsah

1 Úvod

V dnešní době jsou sociální sítě a blogovací služby velmi rozšířeným způsobem elektronické komunikace. Na internetu existuje nepřeberné množství těchto služeb, které se neustále vyvíjejí, rozšiřují svoji funkcionalitu a nabízejí uživatelům stále více možností využití. Současně roste i jejich popularita mezi lidmi. Mikroblogovací služby se osvědčily zpravidla tehdy, kdy je zapotřebí rychlého šíření informací.

Ve srovnání s ostatními způsoby elektronické komunikace se jedná o velice efektivní metodu sdílení informací. Běžná elektronická pošta sice umožňuje zasílání delších textů a příloh, k efektivnímu třídění a vyhledávání je ale nutná velká režie, vytváření složek pošty apod., přesto se mnohé informace ztratí v záplavě nově příchozí pošty, která je zvláště v poslední době přesycena nevyžádanými zprávami. Agilnější způsob komunikace v reálném čase představují různí instant messaging klienti. Tyto aplikace ale trpí zásadním problémem v přístupu k dřívějším zprávám, který je často velice omezený. Aplikace mnohdy historii konverzací neukládají. Alternativou pro dlouhodobé využití jsou právě mikroblogovací služby. V mnoha z nich je možné označit příspěvky klíčovými slovy, což umožňuje nejen rychle a efektivně vyhledat konkrétní příspěvek, ale také snadno třídit informace související s požadovaným tématem.

Mnoho firem má své účty u mikroblogovacích služeb, prostřednictvím kterých zveřejňují stručné a krátké zprávy. Díky jednoduchosti používání těchto služeb se jeví jako vhodné jejich využívání v rámci vnitropodnikových sítí, kde usnadňují komunikaci mezi zaměstnanci nejen v rámci jednotlivých týmů a oddělení, ale často i v rámci celého podniku.

Výstupem této práce je implementace mikroblogovací webové aplikace pro komunikaci v týmu. Mezi klíčové vlastnosti aplikace by mělo patřit efektivní vyhledávání v příspěvcích, snadný přístup k informacím, přehledné a nekomplikované uživatelské rozhraní a v neposlední řadě přístup přes mobilní zařízení srovnatelný s přístupem přes osobní počítač. Uživatelé by měli mít možnost vytvářet a spravovat diskusní skupiny. V rámci zabezpečení by aplikace neměla umožňovat přímou registraci uživatelů, aby nemohlo dojít ke zneužití vnitropodnikových dat.

Následující text této bakalářské práce se věnuje analýze existujícího prostředí, návrhu a implementaci mikroblogovací služby se zaměřením na týmové využití.

Druhá kapitola se zabývá stručnou historií mikroblogovacích služeb a detailní analýzou třech významných zástupců. Kromě rozboru a srovnání implementačních technologií těchto služeb jsou popsány i stěžejní koncepce a přínosy jednotlivých řešení, které jsou následně uplatněny při návrhu aplikace.

Ve třetí kapitole je popsán podrobný návrh aplikace. Konceptuální návrh řeší otázku uzpůsobení služby jejímu využití, v návrhu uživatelského rozhraní je kladen důraz na snadnou a intuitivní ovladatelnost aplikace, volbě implementačních technologií společně s řešením dílčích problémů se poté věnuje návrh implementace.

Samotnou implementací se zabývá čtvrtá kapitola, která je členěna podle funkčních součástí použitého návrhového vzoru. Jsou zde popsány nejen jednotlivé metody tříd ovladačů a jejich vzájemná kooperace, ale také zde uvádím změny a rozšíření oproti návrhu aplikace.

V páté kapitole najde čtenář zhodnocení výsledků práce a srovnání s požadovanými záměry. Také jsou zde uvedeny oblasti možného budoucího rozšíření aplikace.

2 Existující služby

2.1 Mikroblog

Definice podle Symbio[1]: Mikroblog je zmenšenou obdobou klasického webového deníku (blogu) a slouží převážně k publikování textů omezené délky. Hlavní rozdíl mezi blogy a mikroblogy spočívá v délce jednotlivých příspěvků. Pro mikroblogy jsou typické krátké texty (okolo 140–160 znaků), ke kterým můžeme připojit odkazy, fotografie, videa apod. Příspěvky lze obvykle posílat přes webové rozhraní, mobilní telefon, desktopové aplikace, chat nebo třeba e-mail. Mikroblogy je možné sdílet buď se všemi bez omezení, nebo jen s vybranými uživateli. Právě pro stručnost příspěvků si mikroblogování oblíbili lidé, kteří nemají čas na psaní rozsáhlých článků. Využívají jej ale také firmy publikující novinky, zajímavé události a střípky ze života. [1]

2.2 Stručná historie mikroblogovacích služeb

1. března 2006 sociální síť Facebook[2] jako první zavedla možnost aktualizace statusů – vkládání krátkých příspěvků. Tuto funkci můžeme považovat za první mikroblog, zatím ale pouze jako součást rozsáhlejší služby.

První samostatná mikroblogovací služba vznikla 13. července téhož roku. Student New York University Jack Dorsey představil myšlenku projektu využívajícího SMS zpráv pro komunikaci a sdílení informací mezi více lidmi. Tak vznikla služba Twitter[3]. První příspěvek autor zveřejnil již 21. března, úplná služba ale byla představena až téměř o čtyři měsíce později.

V následujícím období přibývalo sociálních sítí, které po vzoru sítě Facebook a služby Twitter zaváděly své vlastní obdoby aktualizace statusů, jednalo se např. o MySpace, LinkedIn a další. Vznikaly též nové čistě mikroblogovací služby, jako např. Tumblr[4], FriendFeed nebo Identi.ca.

13. prosince 2007 byly poprvé použity tzv. Hashtags[5], klíčová slova nebo hesla pro označování příspěvků a usnadnění jejich třídění a vyhledávání.

8. září 2008 byla spuštěna služba Yammer[6], jedna z prvních mikroblogovacích služeb pro komunikaci v rámci organizací. [2]

[1] Citováno z http://www.symbio.cz/slovnik/mikroblog.html

[2] URL služby http://www.facebook.com

[3] URL služby https://www.twitter.com

[4] URL služby http://www.tumblr.com

[5] Další informace dostupné na http://en.wikipedia.org/wiki/Hashtag

[6] URL služby https://www.yammer.com

2.3 Analýza vybraných mikroblogovacích služeb

2.3.1 Twitter

Twitter, jak je uvedeno výše, je první úplnou mikroblogovací službou. Před osamostatněním v roce 2007 byl první prototyp služby, tehdy pod označením twttr, interní službou pro zaměstnance společnosti ODEO. První myšlenky autora orientovaly službu převážně na mobilní zařízení. Osoba, která chtěla informovat své kolegy, zaslala příspěvek maximální délky 140 znaků formou SMS zprávy na telefonní číslo 40404 (USA). Zpráva byla poté přeposlána všem jeho kolegům. Druhou možností pro zasílání a čtení zpráv bylo webové rozhraní služby, které umožňovalo nastavit omezení zasílání SMS zpráv od konkrétních uživatelů nebo zakázat zasílání všech SMS zpráv. Po uvolnění a masovém rozšíření se Twitter stal hojně využívanou službou pro šíření informací v krizových situacích. [3][4]

V současné době má Twitter více než 500 milionů registrovaných uživatelů, mezi nimiž je řada organizací a osobností, denně je odesláno více než 340 milionů zpráv a více než 1,6 miliardy požadavků na vyhledání. Kromě již uvedených možností zasílání a čtení zpráv existuje mnoho aplikací pro mobilní i jiná zařízení, které jsou schopny komunikovat prostřednictvím služby Twitter. [5]

Z implementačního hlediska Twitter vychází z otevřeného softwaru a kombinuje několik technologií. Uživatelské rozhraní je implementováno za použití frameworku Ruby on Rails, využívajícím implementaci jazyka Ruby označenou Ruby Enterprise Edition[1]. Některé služby a procesy původně také implementované v jazyce Ruby byly v letech 2007 až 2008 nahrazeny implementacemi v jazyce Scala[2] využívajícími Java Virtual Machine. Pro zefektivnění vyhledávání v textu byla v roce 2011 využita knihovna Lucene[3], využívající programovací jazyk Java. Ve stejném roce byl implementován nový Java server nazvaný Blender[4], díky kterému došlo ke snížení latence při vyhledávání na třetinovou hodnotu. [6][7][8]

[1] Další informace dostupné na http://www.rubyenterpriseedition.com/documentation.html

[2] Další informace dostupné na http://www.scala-lang.org/node/197

[3] Další informace dostupné na http://lucene.apache.org/

[4] Podrobnosti o technologii Blender dostupné na http://engineering.twitter.com/2011/04/twitter-search-is-now-3x-faster_1656.html

2.3.2 Yammer

Yammer je podniková sociální síť a mikroblogovací služba. Služba je zdarma, pokročilé funkce jsou dostupné za příplatek. Uživatelé mohou sdílet informace s kolegy pouze v rámci své organizace. Přístup do podnikové sítě je umožněn na základě firemní e-mailové schránky. [9]

Služba byla spuštěna 8. září 2008 a v prvních verzích se jednalo pouze o mikroblog pro komunikaci osob v rámci organizace. Od září roku 2010 byla vydána nová verze nazvaná Yammer 2.0, která rozšířila funkce aplikace z mikroblogu na plnohodnotnou podnikovou sociální síť. Současně začali být ze strany tvůrců podporováni vývojáři třetích stran, kteří mohou vytvářet, nabízet a prodávat rozšiřující aplikace. Tou dobou již byl Yammer využíván po celém světě více než jedním milionem uživatelů z osmdesáti tisíc organizací. V červnu 2012 byl Yammer odkoupen společností Microsoft za 1,2 miliardy amerických dolarů. [9][10]

Implementace služby Yammer využívá programovací jazyky Ruby a Scala. Od prosince 2011 byly některé části, obdobně, jako tomu bylo u služby Twitter, reimplementovány za použití programovacího jazyka Java. Od akvizice společností Microsoft je též propojena s Microsoft Active Directory. [11]

2.3.3 Tumblr

Mikroblogovací služba a sociální síť Tumblr byla uvedena do provozu 27. dubna 2007. Služba umožňuje uživatelům vytvořit si bezplatně vlastní blog, na kterém lze kromě krátkých textových zpráv sdílet i multimediální obsah. Uživatelé mohou řídit přístup na svůj blog, mohou vytvářet blogy pro skupiny uživatelů nebo blogy s přístupem chráněným heslem. Na základě tohoto faktu je možné Tumblr považovat za první mikroblog pro týmy. Hlavní předností služby Tumblr je jednoduchost, se kterou mohou uživatelé sdílet požadovaná data. [12][13]

V únoru 2009 byla pro uživatele zdarma zpřístupněna oficiální aplikace pro Apple iPhone, dříve známá jako Tumblerette. [14]

Tumblr je implementován pomocí technologie PHP5. Hlavní vývojář Marco Arment se svým týmem vytvořili vlastní PHP Framework, který mimo jiné implementuje návrhový vzor MVC. [15]

Tumblr poskytuje vývojářům třetích stran oficiální podporu a možnost využít Thumblr API pro vývoj aplikací propojených se službou Tumblr. Další možností je přizpůsobení vzhledu editací jednoduché HTML šablony. [16]

3 Návrh aplikace

3.1 Konceptuální návrh

K návrhu aplikace využívám informací získaných porovnáváním existujících služeb. Výsledná aplikace by měla nabízet obdobné možnosti jako skupinový mikroblog služby Tumblr nebo podnikový mikroblog služby Yammer. Z hlediska administrace a použití jsou možné dva přístupy:

1) Aplikace plní funkci jediného mikroblogu. Administrátor nebo skupina administrátorů mohou povolit přístup dalším uživatelům nebo přidat či odebrat administrátorská práva uživatelům. Každý tým, který hodlá využívat mikroblog, má vlastní webový server a vlastní instanci služby.
2) Existuje pouze jediná služba, která současně spravuje více mikroblogů. Aplikace umožňuje registraci uživatelů, každý uživatel se může stát administrátorem vlastního mikroblogu a ostatním uživatelům povolit či zamezit přístup ke svému mikroblogu.

Volba přístupu záleží na plánovaném nasazení. Pokud by se mělo jednat o službu, která bude využívána zaměstnanci a studenty fakulty pro sdílení informací přímo či nepřímo souvisejících se studiem, výzkumem a prací, vhodnější by byl z hlediska administrace a kontroly vkládaných dat přístup první. Obzvláště efektivní by bylo využívání klíčových slov (obdoba Hashtags služby Twitter), díky kterým by se daly snadno třídit informace týkající se konkrétních kurzů nebo událostí. Pro využití studenty v týmových projektech by mohl být zaveden obdobný model, jakým se řídí vytváření SVN repozitáře pro studenty řešící projekty kurzu Seminář Java[1], tzn. vytvoření a přidělení mikroblogu týmu studentů vyučujícím/cvičícím. Tím by bylo možné i nadále jednoduše dohlížet na zveřejňovaný obsah. Vznikaly by tak pouze oficiální mikroblogy spravované zaměstnanci fakulty. Tento přístup by byl vhodnější také z hlediska bezpečnosti – bylo by možné provozovat mikroblog např. pouze na lokální podsíti či intranetu.

Využití druhého přístupu by snížilo režii nutnou k vytváření mikroblogů, v takovém případě by ale pro zachování možnosti spravovat veškeré příspěvky bylo nutné implementovat více rolí (např. administrátor, moderátor, uživatel). Současně by význam klíčových slov poklesl, neboť by bylo možné hledat příspěvky listováním v konkrétním mikroblogu. Navíc by se služba vzdálila od svého primárního účelu, neboť by umožňovala vytvářet osobní mikroblog.

Vzhledem ke zmíněným negativním vlastnostem druhého přístupu využiji pro implementaci přístup první. Při tomto řešení bude služba více odpovídat koncepci již existujících mikroblogovacích služeb, zejména jednoduchostí přístupu k informacím.

[1] Informace o kurzu dostupné na https://www.fit.vutbr.cz/study/courses/index.php?id=7993

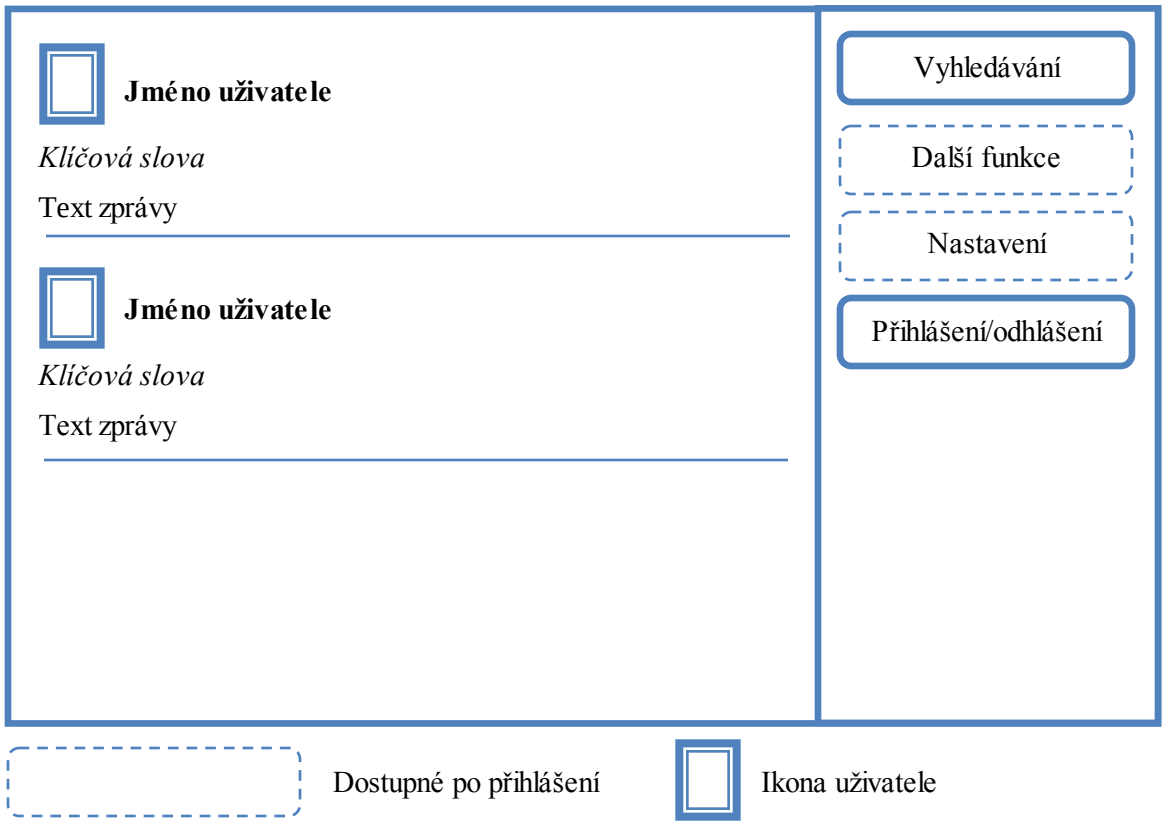
3.2 Návrh uživatelského rozhraní

Uživatelská rozhraní existujících mikroblogů vykazují značnou podobnost. Téměř vždy se člení na dva vertikální sloupce, kde v širším jsou zobrazovány příspěvky a v užším ovládací prvky, např. přihlášení a odhlášení, nastavení nebo vyhledávání. Pořadí sloupců není dáno pravidlem (ve službě Twitter se ovládací prvky nacházejí v levém sloupci, ve službě Tumblr v pravém).

Pro snadnější orientaci uživatelů je vhodné rozmístit prvky uživatelského rozhraní podobně jako u existujících služeb. Orientační schéma uživatelského rozhraní popisuje obrázek 3.1. Pro umístění ovládacích prvků jsem zvolil pravý sloupec, jelikož primárním účelem mikroblogu je sdílení informací a přirozený směr orientace je shodný se směrem čtení textu. Jako první tedy uživatel zaregistruje levý sloupec s informacemi.

Formát příspěvku bude následující:

- První řádek bude obsahovat ikonu a jméno uživatele.
- Na druhém řádku bude uveden seznam klíčových slov.
- Na dalších řádcích bude samotný text příspěvku.



Obr. 3.1: Schematické rozložení uživatelského rozhraní

V pravém sloupci se bude na prvním řádku vyskytovat jednoduchý formulář pro vyhledávání podle klíčových slov. Níže se po přihlášení zobrazí tlačítko pro přístup k nastavení profilu uživatele a další funkce dostupné pouze pro registrované uživatele. Administrátor bude mít k dispozici také tlačítko správy uživatelů. Nejspodnější položkou menu bude tlačítko pro přihlášení nebo odhlášení uživatele.

Možnosti nastavení profilu uživatele se zobrazí v levém sloupci. Uživatel bude moci změnit své zobrazované jméno, změnit heslo pro přihlášení nebo nastavit ikonu zobrazovanou u svých příspěvků. Uživatel nebude mít možnost změnit svoji e-mailovou adresu (fungující jako přihlašovací jméno). Tato funkce bude přenechána pouze administrátorovi, aby měl objektivní přehled o uživatelích a mohl je případně kontaktovat na platnou e-mailovou adresu (uživatelé by mohli zneužívat možnosti měnit si uživatelské jméno a nemuseli by využívat platné e-mailové adresy). Pro změnu přihlašovacího jména tedy uživatel požádá administrátora např. prostřednictvím e-mailu.

Správa uživatelů bude zobrazena formou tabulky uživatelů, kteří mohou číst, vyhledávat a přispívat do mikroblogu. Administrátor bude mít možnost zaregistrovat nového uživatele, vyhledat v tabulce konkrétního uživatele (podle jména nebo e-mailové schránky), upravit jeho údaje, přidat uživateli administrátorská práva a odstranit uživatele ze systému.

3.3 Návrh implementace

3.3.1 Implementační technologie

Pro implementaci jsem se na doporučení vedoucího bakalářské práce rozhodl využít volně dostupnou technologii ASP.NET MVC4 Razor[1] od firmy Microsoft. Jedná se o framework využívající návrhový vzor Model-View-Controller[2], který je vhodný pro prezentačně orientované webové aplikace. Koncepce a možnosti tohoto prostředku vyhovují požadavkům návrhu a nabízejí více způsobů realizace dílčích částí aplikace. [17]

3.3.2 Registrace uživatelů

Jelikož se uživatelé nemohou sami zaregistrovat, vyvstává problém registrace prvního uživatele – administrátora. Lze jej řešit takto: pokud v systému neexistuje žádný uživatel, bude možné se přihlásit k počátečnímu účtu s administrátorskými právy bez registrace. Z tohoto účtu bude možné, stejně jako později ze všech administrátorských účtů, provádět registrace dalších uživatelů. První uživatel si poté změní přihlašovací (implicitně např. uživ. jméno: admin, heslo: admin) i osobní údaje na své vlastní.

[1] Další informace dostupné na http://www.asp.net/mvc/mvc4

[2] Základní informace o návrhovém vzoru dostupné na http://cs.wikipedia.org/wiki/Model-view-controller

3.3.3 Klíčová slova

Pro vyhledávání a třídění příspěvků bude možné označit vkládané příspěvky jedním nebo více klíčovými slovy. Bude se jednat o obdobu hashtags používaných službou Twitter. Klíčová slova budou uložena spolu s textem příspěvku, autorem, datem a časem v jednom záznamu. Zároveň bude existovat tabulka klíčových slov, do které se nově vložená klíčová slova uloží také. Tato tabulka bude sloužit jako zdroj dat pro našeptávače (viz níže). Při vyhledávání podle klíčových slov se zobrazí příspěvky, které obsahují všechna uvedená klíčová slova – při hledání podle dvou klíčových slov současně nebudou zobrazovány částečné výsledky odpovídající pouze jednomu klíčovému slovu, ale pouze výsledky s plnou shodou. Vyhledávání podle obsahu textu příspěvku možné nebude, důraz je kladen na značení příspěvků vystihujícími klíčovými slovy, které kromě funkce vyhledávací slouží i k tematickému propojení příspěvků.

3.3.4 Našeptávač klíčových slov

Při zápisu klíčových slov u vkládání příspěvků a u vyhledávání bude přítomný našeptávač. Jednou z možností pro implementaci našeptávače je využití technologie AJAX[1], která splňuje požadavky na dynamické dotazování na výsledky porovnání. Z existujících řešení dostupných pro ASP.NET framework je možné uvažovat o volně šiřitelném projektu ASP.NET AJAX Control Toolkit[2], který vhodným způsobem implementuje funkci AutoComplete[3], která funguje jako našeptávač a je možné ji propojit s různými datovými zdroji, např. s databází. Druhou možností je použití prvku jQuery UI Autocomplete widget[4], u tohoto přístupu by ale v budoucnu velké množství klíčových slov v databázi mohlo způsobit problémy s latencí a rychlostí reakce služby. [18][19]

3.3.5 Skupiny uživatelů

Uživatelé budou mít možnost vytvářet diskusní skupiny. Ty budou primárně sloužit k jednoduššímu třídění souvisejících příspěvků. Skupinu bude moci vytvořit každý uživatel. Poté se skupina zviditelní ostatním uživatelům, kteří budou moci zažádat o členství ve skupině. Žádosti spravuje výhradně zakladatel skupiny. Potvrzením žádosti vzniká uživateli možnost přispívat do dané skupiny a zobrazit seznam všech ostatních uživatelů, kteří jsou jejími členy. V případě zamítnutí žádosti nemůže uživatel opakovaně odeslat žádost o členství, dokud zakladatel skupiny nezruší

[1] Asynchronní JavaScript a XML, detaily dostupné na http://www.w3schools.com/ajax/ajax_intro.asp

[2] Další informace dostupné na http://www.asp.net/ajaxlibrary/AjaxControlToolkitSampleSite/

[3] Popis použití včetně demonstrace dostupný na http://www.asp.net/ajaxLibrary/AjaxControlToolkitSampleSite/AutoComplete/AutoComplete.aspx

[4] Detailní popis včetně demonstrace dostupný na http://api.jqueryui.com/autocomplete/

zamítnutí, čímž uživateli umožní celý proces opakovat. Pokud žádá o členství administrátor, jeho žádost je automaticky schválena. Opustit skupinu může každý člen kromě zakladatele. Dále mohou být členové ze skupiny vyloučeni, a to zakladatelem skupiny nebo členem s administrátorskými právy. Pokud administrátor vyloučí se skupiny jejího zakladatele, převezme všechna jeho práva. Zrušit diskusní skupinu může pouze její zakladatel. Příspěvky odeslané do zrušené diskusní skupiny nebudou smazány a bude je možno stále zobrazit.

3.3.6 Hlasování v příspěvcích

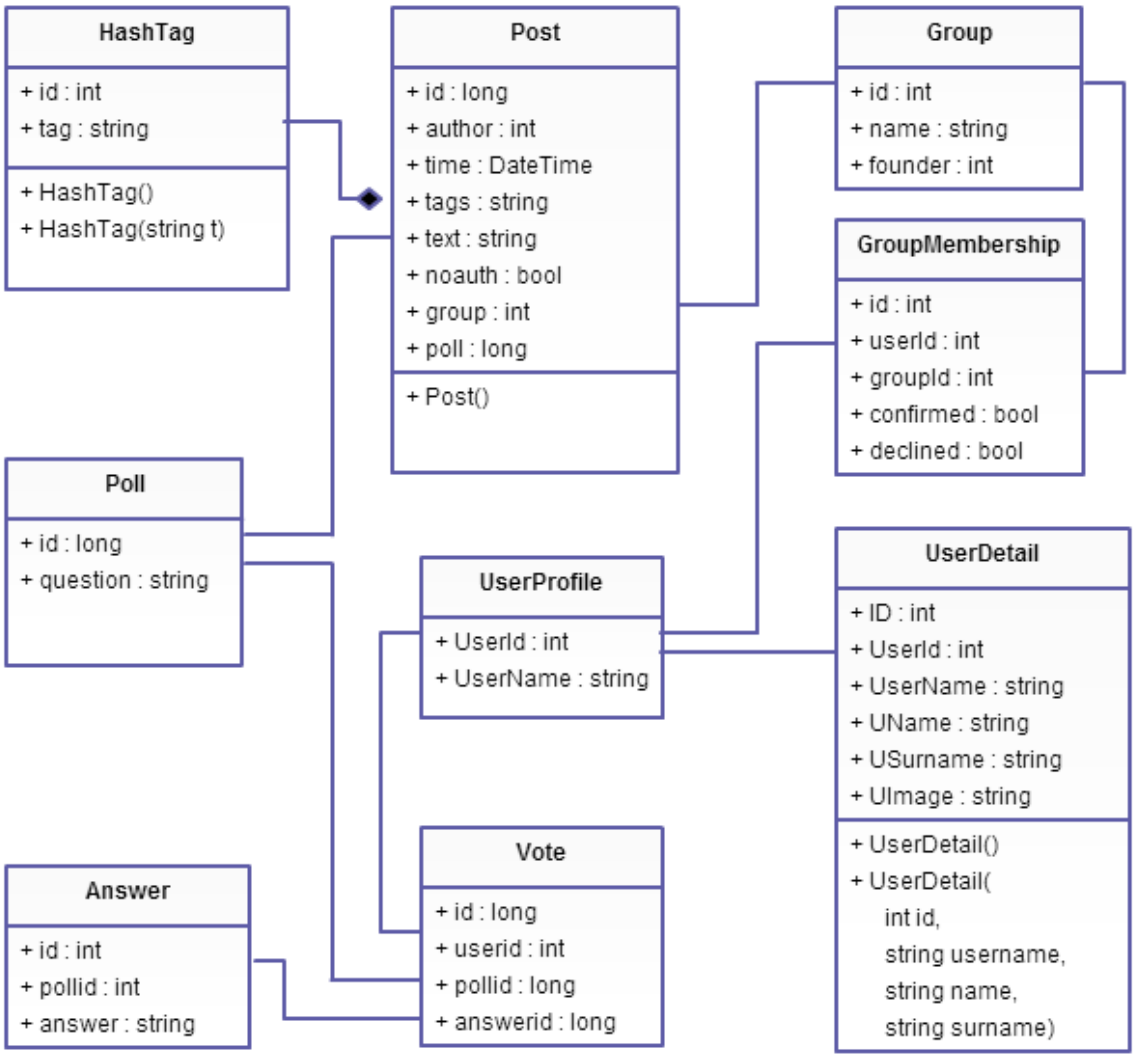
K příspěvkům bude možné připojit jednoduché hlasování. Každý přihlášený uživatel bude mít jeden hlas pro volbu jedné z několika odpovědí na otázku. Průběžné výsledky hlasování budou interpretovány graficky (formou jednoduchého sloupcového grafu) i numericky. Vzhledem k počtu možných odpovědí mohou existovat dva přístupy: možnost předem definovat počet odpovědí teoreticky s neomezenou horní hranicí nebo předpřipravený formulář s omezeným počtem polí. Pokud je jedním z hlavních kritérií aplikace rychlost její obsluhy, druhý přístup představuje výhodu z hlediska počtu kroků nutných k vytvoření příspěvku. Je ale třeba zvolit vhodný horní limit počtu odpovědí.

4 Implementace

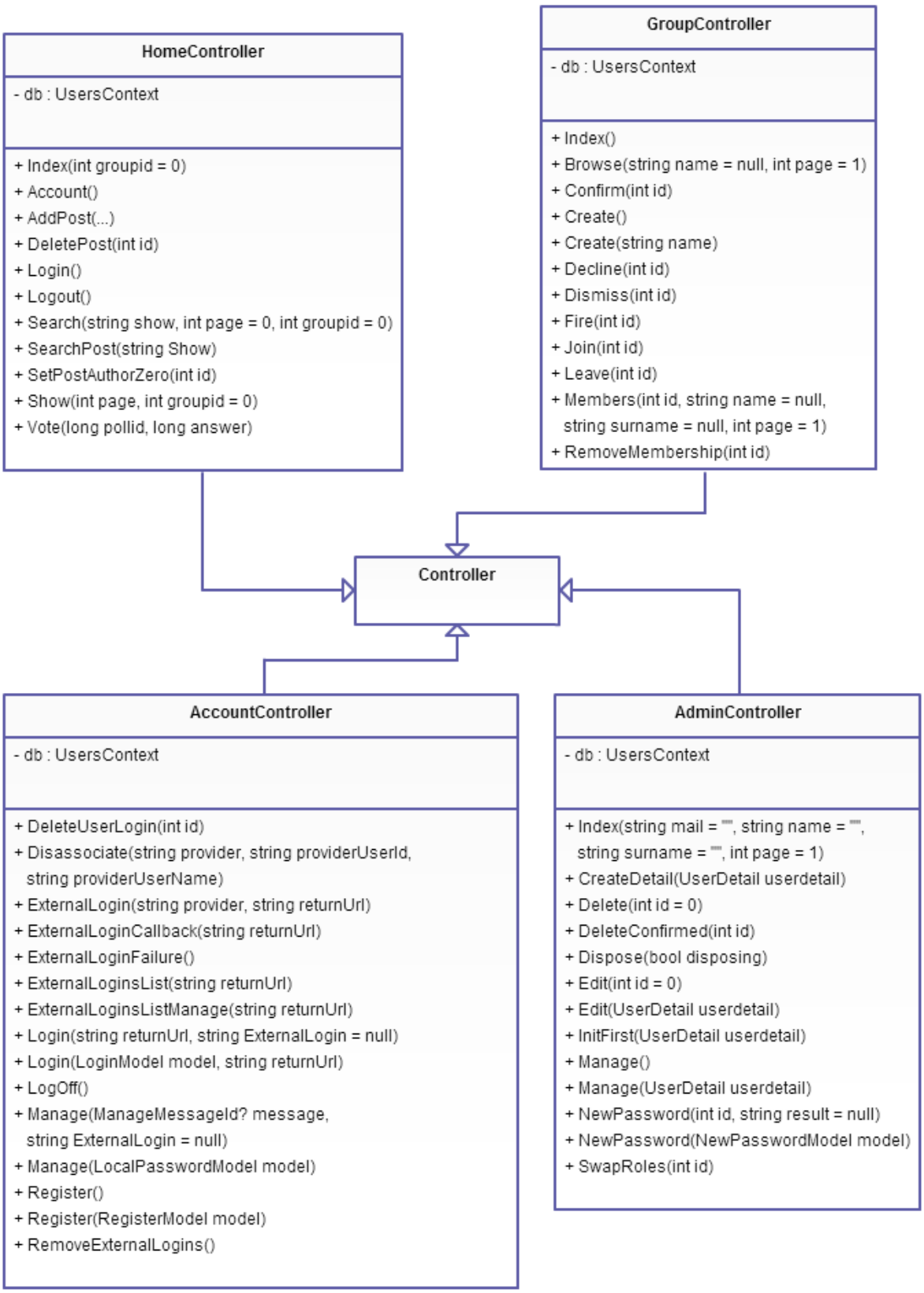
Pro implementaci jsem využil technologii ASP.NET a programovací jazyk C#. Dotazování nad databází jsem implementoval s použitím programovacího jazyka LINQ, který je s výše uvedenými plně kompatibilní. Pro studium těchto programovacích jazyků jsem čerpal ze zdrojů [20] a [21].

4.1 Návrhový vzor MVC

Využití návrhového vzoru MVC umožňuje rozčlenění aplikace na jednotlivé kontextové oddíly. Oddíly jsou vždy tvořeny jedním ovladačem, jenž je objektem podtřídy třídy *Controller* (soubory s příponou *.cs*), která zajišťuje reakce na události vyvolané interakcí uživatele, dále příslušnými pohledy zajišťujícími zobrazení dat uživateli (soubory s příponou *.cshtml*) a třídami tvořící datové modely (soubory s příponou *.cs*, obr. 4.1). Metody a proměnné tříd ovladačů jsou zobrazeny v diagramu tříd na obr. 4.2.



Obr. 4.1: Diagram tříd datových modelů



Obr. 4.2: Diagram tříd Microblog.Controllers

4.2 Sdílené pohledy

Sdílené pohledy slouží pro vykreslení statických a společných částí uživatelského rozhraní aplikace. Rozložení uživatelského rozhraní je popsáno v souboru *_Layout.cshtml*. Jedná se o kód psaný jazykem HTML 5 s vnořenými příkazy zobrazovacího jádra Razor. Výchozí rozložení vygenerované jako šablona bylo zcela nevyhovující a neodpovídalo mému návrhu uživatelského rozhraní aplikace, proto jsem vytvořil rozložení se dvěma sloupci.

Díky vnořeným příkazům zobrazovacího jádra bylo možné do sdíleného pohledu jednoduše implementovat dynamické prvky. Tímto způsobem je realizováno menu: pro nepřihlášeného uživatele je k dispozici pouze vyhledávání a tlačítko pro přihlášení, přihlášenému uživateli se kromě vyhledávání zobrazí tlačítka pro přístup ke skupinám uživatelů, nastavení a odhlášení. Přihlášenému administrátorovi se zobrazí navíc ještě tlačítko pro správu uživatelů. Pod tlačítkem skupin uživatelů se pro rychlý přístup zobrazuje až pět skupin, v nichž má uživatel platné členství, seřazených podle času posledních příspěvků. Zakladatelům skupin se navíc pod menu zobrazuje upozornění, že jiný uživatel žádá o členství ve skupině.

Po zvážení jsem se rozhodl do sdíleného pohledu zahrnout i formulář pro psaní příspěvků. Tento krok byl nutný z hlediska použití našeptávačů AJAX Control Toolkit třídy AutoCompleteExtender. Každý formulář (soubor s příponou *.aspx*) obsahující našeptávač musí také obsahovat objekt třídy *ScriptManager*[1], který zajišťuje mj. obsluhu prvků využívajících Ajax. V jednom pohledu může být aktivní pouze jeden objekt třídy ScriptManager. Po přihlášení, kdy jsou zobrazeny dva formuláře jako částečné pohledy (vyhledávání a vložení nového článku), tímto způsobem nelze zajistit funkčnost obou našeptávačů současně. Bylo tedy nutné vytvořit jeden formulář obsahující dva blokové prvky a pomocí kaskádových stylů tyto bloky umístit na požadované pozice. Tím bylo možné dosáhnout požadované funkčnosti obou našeptávačů nezávisle na sobě.

4.3 Třída HomeController

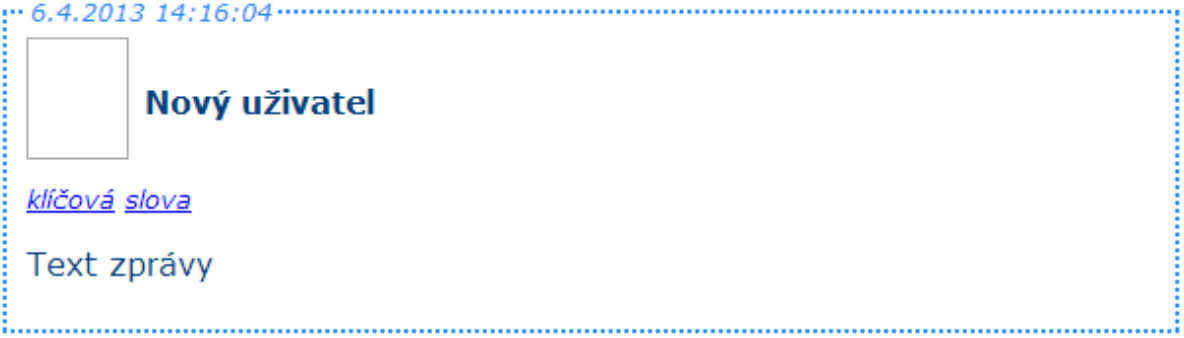
Třída *HomeController* implementuje ovladač pro zobrazení, vkládání a vyhledávání příspěvků. Třída, stejně jako ostatní třídy ovladačů, obsahuje proměnnou `db`, která je instancí podtřídy třídy `DbContext`, která zajišťuje obousměrný přístup k datům v databázi. Z ovladače HomeController je konkrétně přistupováno k datům modelů `Post`, `UserDetail`, `HashTag`, `Group` pro přístup k příspěvkům a jejich detailům a modelů `Poll`, `Answer` a `Vote` svázané s hlasováním v příspěvcích.

Metody `Index`, `Show` a `Search` slouží pro předání určité části seznamu příspěvků příslušnému pohledu, který příspěvky zobrazí uživateli. Jako horní limit zobrazení na stránku jsem zvolil

[1] Další informace dostupné na http://msdn.microsoft.com/cs-cz/library/system.web.ui.scriptmanager.aspx

15 příspěvků. Pokud požadovanému zobrazení odpovídá více příspěvků, je na spodním okraji pohledu zobrazen odkaz pro přechod na následující stránku – zobrazení starších příspěvků. V případě metody `index` volané bez argumentů jsou zobrazeny nejnovější příspěvky. Pokud je metoda volána s argumentem `groupid`, který nabývá hodnoty čísla skupiny, jejímž je uživatel členem, zobrazí se nejnovější příspěvky v dané skupině.

Zobrazení příspěvků odpovídá návrhu. Pro grafické oddělení jednotlivých příspěvků jsem se rozhodl obalit strukturu příspěvku elementem `fieldset` s orámováním definovaným kaskádovým stylem. Takové řešení má výhodu v možnosti použití vnořeného elementu `legend` pro zobrazení času odeslání příspěvku. Konečný grafický formát příspěvku je zachycen na obrázku 4.3.



Obr. 4.3: Formát zobrazení příspěvku včetně kaskádových stylů.

Samotná klíčová slova příspěvku fungují jako odkazy pro vyhledávání. Kliknutím na příslušný odkaz se zobrazí pouze příspěvky obsahující dané klíčové slovo. Pokud se akce opakuje, můžeme kliknutím na další klíčová slova u příspěvků zúžit výběr. Takto vybraná klíčová slova jsou poté zobrazena v horní části pohledu. Fungují zde jako odkazy pro odstranění daného klíčového slova z vyhledávání, tudíž k rozšíření výběru. Pokud je použito vyhledávání pomocí formuláře, hledaný řetězec klíčových slov je rozdělen a výsledné zobrazení je shodné. Pokud je vícenásobně zadáno stejné klíčové slovo, první výskyt je považován za platný a ostatní jsou ignorovány.

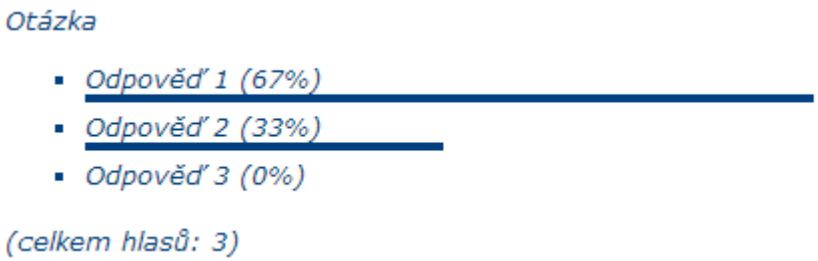
Pokud uživatel vyplní formulář pro nový příspěvek, odesláním formuláře je zavolána metoda `AddPost`, jejímiž argumenty jsou obsahy polí formuláře. Po kontrole neprázdnosti vkládaného textu je vytvořena nová instance třídy `Post`. Klíčová slova připojená k textu příspěvku jsou rozdělena na pole řetězců jednotlivých slov. Jako oddělovač pro klíčová slova v příspěvku je použit znak mezerník[1]. Poté jsou všechna slova uložená v poli porovnána s existujícími instancemi třídy `HastHag`. Pokud se slovo v databázi dosud nevyskytuje, je instanciován nový objekt třídy HashTag obsahující nenalezený řetězec. Pokud uživatel ve formuláři vyplnil hlasování, jednotlivé odpovědi jsou vloženy do pole řetězců, přičemž jsou vynechávány prázdné řetězce a řetězce obsahující pouze

[1] Znak s ASCII hodnotou 32. Použití všech bílých znaků jako oddělovačů jsem v daném kontextu považoval za zbytečné.

bílé znaky. Pokud tedy uživatel hodlá uvést pouze tři odpovědi, může tak provést vyplněním libovolných tří polí. Pro otázku i pro všechny prvky seznamu jsou vytvořeny příslušné objekty tříd `Poll` a `Answer`. Hodnota instanční proměnné `poll` v objektu `Post` odkazuje na číselnou hodnotu identifikátoru objektu třídy `Poll`. Pokud se v daném příspěvku hlasování nevyskytuje, nabývá nulové hodnoty.

Pro rozdělení hlasování do objektů dvou tříd jsem se rozhodl, aby se snížila paměťová náročnost. Kdyby byly všechny odpovědi uloženy v proměnných jednoho objektu společně s otázkou, obsahoval by objekt v případě zadání menšího počtu odpovědí nevyužitelnou část paměti[1]. S rostoucím počtem příspěvků by tak docházelo ke stále rozsáhlejšímu neefektivnímu využívání paměti.

Hlasování je zobrazeno pod textem příspěvku. Jednotlivé možnosti tvoří položky nečíslovaného seznamu (obr. 4.4). Pokud uživatel doposud k dané otázce nehlasoval, odpovědi jsou zobrazeny formou odkazu, který volá metodu `Vote`. Tato metoda zajišťuje zabezpečení celé operace a samotné uložení hlasu. Výsledky hlasování jsou graficky zobrazeny pomocí kaskádových stylů, jež určují šířku bloku dané odpovědi v procentech korespondujících s počtem hlasů pro danou odpověď. Text odpovědi je zobrazen pomocí přetékání textu (CSS vlastnost overflow). Současně je za odpovědí počet hlasů vyjádřen procentuálně a pod hlasováním je zobrazen celkový počet hlasů. Hlasování je nevratná operace, jednou udělený hlas již nelze zrušit a hlasovat pro jinou odpověď.



Obr. 4.4: Formát zobrazení hlasování včetně kaskádových stylů.

K odstranění příspěvku slouží metoda `DeletePost`. Příspěvek může odstranit pouze jeho autor nebo administrátor. Společně s objektem příspěvku jsou odstraněny i související objekty hlasování, tj. objekt třídy `Poll`, objekty třídy `Answer` a objekty třídy `Vote`, které zaznamenávají jednotlivé hlasy uživatelů. Pokud odstraněním příspěvku dojde k situaci, že existuje klíčové slovo, na nějž není odkazováno z žádného dalšího příspěvku, je odstraněn i objekt reprezentující dané klíčové slovo.

Metoda `SetPostAuthorZero` je volána v kontextu odstranění uživatele. Jejím smyslem je všem nesmazaným příspěvkům odstraňovaného uživatele vynulovat hodnotu instanční proměnné `author`. Při zobrazení těchto příspěvků je poté místo jména autora uveden text *Bývalý uživatel*.

Ostatní metody (`account`, `login`, `logout`) slouží jako alias metod v jiných ovladačích pro snadnější volání z ovladače HomeController a jeho pohledů.

[1] Objekty dědící z třídy DbContext nemohou mít instanční proměnné vyšších datových typů (pole, seznam, aj.).

4.4 Třída AccountController

Třída AccountController implementuje ovladač pro autentizaci a registraci uživatelů. Společně s ovladačem AdminController se stará o kompletní správu uživatelských účtů. Poskytuje metody pro přihlášení, odhlášení, registraci a odstranění uživatelů, změnu hesla a externí přihlášení. Třída pomocí vnitřní proměnné `db` přistupuje k datovým modelům `UserDetail`.

Tato třída byla částečně vytvořena v rámci šablony pro nový projekt ve vývojovém prostředí Microsoft Visual Web Developer 2010 Express[1]. Bylo ale nutné provést mnoho změn souvisejících s omezením přístupu do aplikace, které budou u jednotlivých metod popsány.

První metodou, kterou bylo nutné přepracovat, byla metoda `Login`. Metoda je přetížena, protože reaguje na metody HTTP protokolu GET i POST. Při volání metodou GET je zobrazen přihlašovací formulář, při volání metodou POST se provádí samotná autentizace uživatele. Při prvním volání metody `Login` metodou GET je provedena inicializace uživatelských rolí a registrace prvního uživatele podle návrhu implementace. Následně je, stejně jako při dalších voláních, zobrazen přihlašovací formulář.

Autentizace uživatele byla do značné míry předimplementována v šabloně. Použité řešení využívající pro všechny operace s uživatelskými účty metody třídy `WebSecurity`[2] se zpočátku zdálo být vhodné. Třída `WebSecurity` poskytuje přístup k metodám třídy `Membership` na vyšší úrovni abstrakce, podporuje ale jen podmnožinu těchto metod. Neimplementuje např. uživatelské role, ke kterým musí být paralelně k uživatelským účtům přistupováno metodami třídy `Roles`[3]. Další limitující vlastností je nemožnost změnit přihlašovací jméno uživatele. Tento problém se týká nejen třídy `WebSecurity`, ale i třídy `Membership`. Situaci jsem tedy vyřešil následovně: Při registraci nového uživatele jsou uživatelské údaje uloženy v instanci třídy `UserAlternativeProfile`, která poskytuje přístup ke stejným datům, která jsou uložena v instancích třídy `MembershipUser` reprezentující uživatelské účty třídy `Membership`. To umožňuje změnit přihlašovací jméno uživatele. Způsob přihlášení demonstruje následující kód:

```
1:    UserDetail detail;
2:    if ((detail = detaildb.Users.SingleOrDefault(
3:          user => user.UserName == model.UserName)) != null) {
4:          if (WebSecurity.Login(logindb.profiles.SingleOrDefault(
5:              profile => profile.UserId == detail.UserId).UserName,
6:              model.Password, persistCookie: model.RememberMe))
7:                  return RedirectToAction("Index", "Home");
8:    }
```

Při autentizaci se vyhledá identifikační číslo uživatele podle uživatelského jména v modelu `UserDetail` (řádky 2–3), podle nějž se vyhledá původní uživatelské jméno v modelu

[1] Software ke stažení na http://www.microsoft.com/visualstudio/cze/products/visual-studio-2010-express

[2] Další informace dostupné na http://msdn.microsoft.com/en-us/library/webmatrix.webdata.websecurity.aspx

[3] Další informace dostupné na http://msdn.microsoft.com/en-us/library/system.web.security.roles.aspx

`UserAlternativeProfile` (řádky 4–5), které se použije pro autentizaci. Proměnné `detaildb` a `logindb` jsou instancemi podtříd třídy `DbContext` pro přístup k výše zmíněným modelům. Uživatelská jména v modelu `UserDetail` je tedy možné měnit s omezením na hodnoty bývalých uživatelských jmen ostatních uživatelů. Takové případy jsou ale použitím e-mailové adresy pro přihlašování principiálně vyloučeny.

Registraci uživatelů zajišťuje metoda `Register`, která je rovněž přetížena pro volání metodami GET a POST. Jelikož může registrace uživatelů provádět pouze administrátor, je tento fakt reflektován zabezpečením metody proti neoprávněnému volání. Pohled vytvořený ze šablony bylo nutné přizpůsobit potřebám uchovávat o uživateli více informací. Administrátor takto vytvoří uživatelský účet s vyplněným přihlašovacím jménem – e-mailovou adresou, heslem, jménem a příjmením uživatele a rolí, již bude uživatel v systému zastávat. Ikonu uživatele administrátor nenastavuje. Vlivem potřebných úprav popsaných výše u metody `Login` je současně s vytvořením nového uživatelského účtu instanciován objekt třídy `UserAlternativeProfile`, který poskytuje přístup k prvotnímu uživatelskému jménu a identifikačnímu číslu uživatele.

Změnu hesla zajišťuje metoda `Manage`. I tato metoda je přetížena a implementována obdobným způsobem jako metody `Login` a `Register` s rozdílem, že zobrazený pohled obsahuje Formulář pro změnu hesla v odděleném částečném pohledu. Samotná změna hesla probíhá voláním metody `ChangePassword` třídy `WebSecurity`. Tato operace nevyžadovala přílišné zásahy do původní struktury, pouze byla odstraněna část kódu, jež sloužila k vytvoření nového přihlášení pomocí uživatelského jména a hesla, jelikož byla v kontextu mého návrhu aplikace zbytečná.

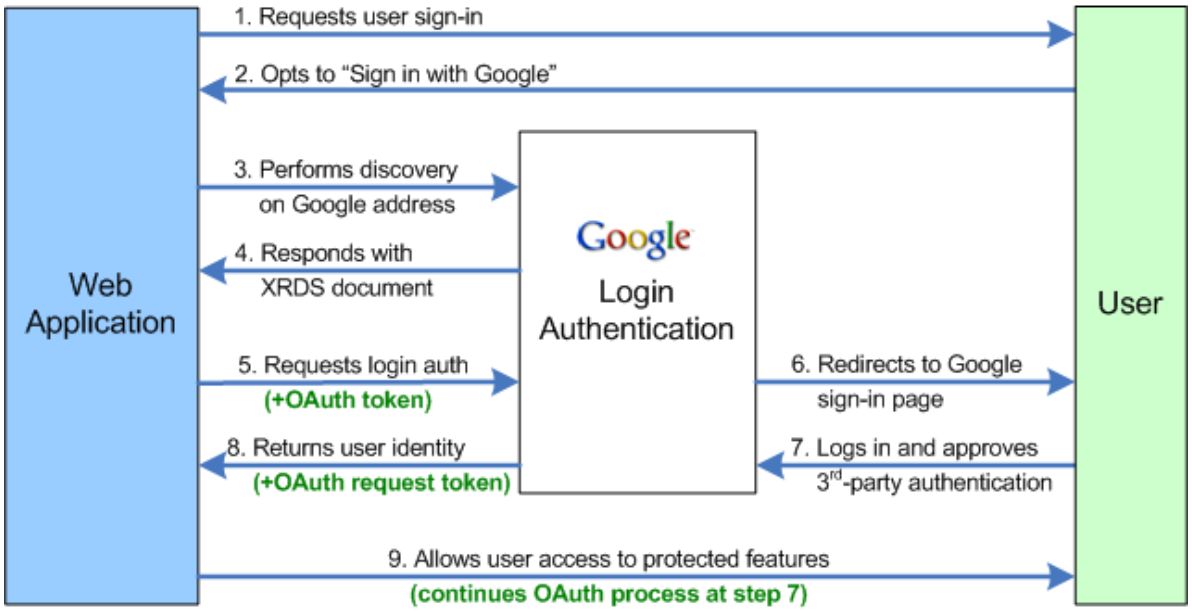
Ke smazání uživatele, resp. objektu třídy `UserAlternativeProfile`, slouží metoda `DeleteUserLogin`. Jedná se o bezpečnou metodu, která je volána nepřímo, a tudíž nemusí být testována validita vstupního parametru `id` – číselného identifikátoru uživatele. Proces mazání uživatele bude popsán v kapitole 4.5.

4.4.1 Externí autentizace

Pro přihlášení uživatele jsem implementoval také externí autentizaci pomocí služby OpenID[1]. Aplikace využívá rozhraní OpenID poskytované službou Google API[2]. Jedná se o otevřený protokol, popisující vzdálenou autentizaci uživatelů. OpenID popisuje komunikaci mezi klientem, službou, ke které klient požaduje přístup (*Relying party – RP*), a autentizačním serverem (zde server Google). Komunikace mezi jednotlivými subjekty probíhá předáváním identifikátorů ve formě URL, jedná se tedy o protokol aplikační vrstvy modelu ISO-OSI. [22]

[1] Další informace dostupné na http://openid.net/get-an-openid/what-is-openid/

[2] Podrobné informace o službě dostupné na https://developers.google.com/accounts/docs/OpenID



Obr. 4.5: Schéma komunikace pomocí protokolu OpenID (převzato z [23]).

Komunikační protokol Google OpenID (Obr. 4.5) tvoří posloupnost devíti operací:

1) Aplikace (RP) požaduje autentizaci uživatele.
2) Uživatel zvolí možnost autentizace pomocí OpenID (Přihlášení účtem Google).
3) Aplikace zašle požadavek na server Google (*OpenID Provider – OP*) pro získání adresy koncového zařízení poskytujícího OpenID (*OP Endpoint URL*)
4) OP na požadavek aplikace odpoví odesláním dokumentu XRDS[1], který obsahuje adresu koncového zařízení.
5) Aplikace odešle požadavek na autentizaci uživatele na adresu koncového zařízení získanou z dokumentu XRDS.
6) Uživatel je přesměrován na stránku požadující přihlášení k účtu Google.
7) Uživatel je upozorněn, že webová aplikace požaduje přihlášení pomocí Google OpenID, a je vyzván k povolení vzdálené autentizace pro danou aplikaci.
8) Uživatel je přesměrován zpět do aplikace, současně je aplikaci prostřednictvím URL předán identifikátor uživatele (parametr *openid.claimed_id*).
9) Aplikace rozpozná uživatele na základě identifikátoru v URL a povolí uživateli přístup. [23]

Metody pro externí autentizaci podporující Google Open ID byly v ovladači AccountController částečně předimplementovány. Samotný protokol OpenID implementuje třída `OAuthWebSecurity`[2], která umožňuje vzdálenou autentizaci nejen pomocí Google OpenID,

[1] Podrobnější informace o dokumentu a jeho formátu dostupné na http://en.wikipedia.org/wiki/XRDS

[2] Podrobné informace dostupné na http://msdn.microsoft.com/en-us/library/microsoft.web.webpages.oauth.oauthwebsecurity(v=vs.111).aspx

ale také pomocí účtů Windows Live, Facebook nebo Twitter. Pro výběr autentizačního serveru bylo zapotřebí editovat soubor AuthConfig.cs ve složce App_Start, který zajišťuje správnou inicializaci objektu třídy `OAuthWebSecurity`.

Metoda `ExternalLogin` zajišťuje třetí krok výše uvedené posloupnosti akcí. Po provedení čtvrtého kroku ze strany serveru je volána metoda `ExternalLoginCallback`, která provede zbývající operace. V případě selhání dojde k zavolání metoda `ExternalLoginFailure`, která chybu ohlásí uživateli. Pokud metodu `ExternalLoginCallback` volá následkem své akce přihlášený uživatel, provede se propojení vzdáleného účtu s uživatelským účtem tohoto uživatele.

Metody `ExternalLoginsList` a `ExternalLoginsListManage` slouží pro zobrazení všech dostupných služeb, pro jejichž vzdálenou autentizaci je aplikace konfigurována, v našem případě zobrazí pouze tlačítko jako odkaz pro přihlášení prostřednictvím Google, příp. tlačítko pro propojení uživatelského účtu s účtem Google. Metoda `RemoveExternalLogins` zobrazí všechny servery, s nimiž má uživatel propojené účty, jako odkazy pro odebrání těchto propojení. Tlačítko pro přihlášení se zobrazuje ve společném pohledu se standardním formulářem pro přihlášení, tlačítka pro propojení účtů a odebrání propojení se zobrazují v jednom pohledu společně s formulářem pro změnu uživatelského hesla.

Metoda `Disassociate` zajišťuje odebrání externí autentizace uživatele. Argument `providerUserId`, reprezentující identifikátor uživatele (viz popis protokolu OpenID, krok 8.), je porovnán s databází, zda náleží uživateli volajícímu metodu. Poté je záznam tohoto uživatele z databáze externích autentizací odebrán. Z původní metody vytvořené šablonou jsem odebral provádění operace formou transakce a test, zda se jedná o jediný způsob autentizace daného uživatele, neboť v kontextu mého návrhu se jednalo o redundantní operace.

4.4.2 Lokalizace validace modelů

V souvislosti s ovladačem AccountController bylo zapotřebí provést lokalizaci chybových hlášení modelů přihlášení, změny hesla a registrace uživatele. Validace modelů probíhá např. jako kontrola neprázdnosti povinných polí, kontrola shody hesla ve dvou polích při změně uživatelského hesla nebo kontrola jedinečnosti uživatelského jména.

K těmto účelům slouží v definicích modelů příkazy `[Key]` (klíčová hodnota modelu – jedinečnost, automatická inkrementace), `[Required]` (povinná hodnota – testování neprázdnosti), `[DataType]` (datový typ – testování formátu vstupních dat), `[StringLength]` (omezení délky řetězce – minimální délka hesla) nebo `[Compare]` (porovnání hodnot proměnných – stejná hesla).

Při porušení validity jsou uživateli zobrazena chybová hlášení. Implicitně jsou chybová hlášení zobrazena anglicky ve tvaru „The `{číslo}` field is required.“ Číslo zde udává pořadí proměnné, jejíž hodnota je vložena do textu. Lokalizaci těchto hlášení je možné provést dvěma způsoby.

První možností je rozšíření validačního příkazu, např. v modelu pro změnu uživatelského hesla s proměnnými Password a ConfirmPassword je možné pro druhou proměnnou zapsat příkaz ve tvaru:

```
[Compare("Password", ErrorMessage = "Hesla se neshodují.")]
```

V případě chyby se zobrazí hlášení zadané proměnnou ErrorMessage. Zmíněné řešení je vhodné pro validace, které nejsou využívány opakovaně.

Pro lokalizaci četnějších hlášení, např. validace povinných položek, existuje vhodnější postup s využitím souborů prostředků[1]. Ve složce Properties jsem vytvořil soubor zdrojů Resources.resx, který obsahuje dvojice název prostředku – chybové hlášení. Na tato hlášení je možné odkazovat z příkazů validace následujícím způsobem:

```
[Required(ErrorMessageResourceType = typeof(Properties.Resources),
ErrorMessageResourceName = "Required_cs")]
```

`Required_cs` je v tomto případě název požadovaného prostředku. Při chybě validace je poté vypsáno hlášení odpovídající prostředku daného jména, v našem případě „Pole {0} je povinné." Symbol `{0}` je vždy interpretován jako hodnota proměnné `Name` definovaná příkazem `[Display(Name = „název proměnné")]`. [24]

S využitím obou metod lokalizace je tedy definice modelu pro změnu hesla následující:

```
public class LocalPasswordModel
{
  [Required(ErrorMessageResourceType = typeof(Properties.Resources),
  ErrorMessageResourceName = "Required_cs")]
  [DataType(DataType.Password)]
  [Display(Name = "Současné heslo")]
  public string OldPassword { get; set; }

  [Required(ErrorMessageResourceType = typeof(Properties.Resources),
  ErrorMessageResourceName = "Required_cs")]
  [StringLength(100, ErrorMessage = "{0} musí mít aspoň {2} znaků.",
  MinimumLength = 5)] // {2} značí hodnotu proměnné MinimumLength
  [DataType(DataType.Password)]
  [Display(Name = "Nové heslo")]
  public string NewPassword { get; set; }

  [DataType(DataType.Password)]
  [Display(Name = "Nové heslo znovu")]
  [Compare("NewPassword",ErrorMessage = "Nová hesla se neshodují.")]
  public string ConfirmPassword { get; set; }
}
```

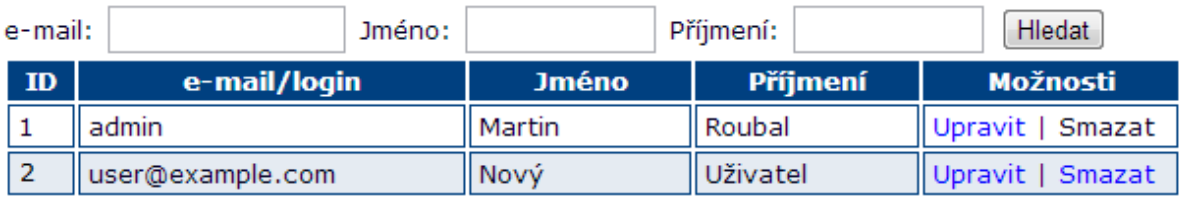
Obdobně jsou definovány i ostatní modely pro správu uživatelských účtů, registraci uživatelů či přihlášení.

[1] V originále Resource Files, detaily dostupné na http://msdn.microsoft.com/en-us/library/ms227427.aspx

4.5 Třída AdminController

Třída AdminController je implementací ovladače plnícího dvě zásadní funkce: správu uživatelů a editaci uživatelského profilu. Podobně jako v ovladači AccountController je volání metod podmíněno autentizací uživatele. Pro metody realizující správu uživatelů je navíc nutné přihlášení uživatele s rolí administrátora. Vnitřní proměnná `db`, kterou třída obsahuje, je využita k přístupu k modelům tříd `UserDetail` a `UserAlternativeProfile`.

Metoda `Index` je volána odkazem „Uživatelé“ z hlavního menu aplikace. Slouží k zobrazení hlavního pohledu pro správu uživatelů. Pohled obsahuje tabulku uživatelů s možnostmi editovat profil uživatele či odstranit uživatele ze systému. Počet záznamů v tabulce je stránkovaný po 25 uživatelích. Záznamy jsou řazeny podle číselného identifikátoru uživatele a je možné je filtrovat podle e-mailové adresy, jména, příjmení uživatele či libovolné kombinace uvedených parametrů. Řetězce znaků pro filtraci záznamů společně se společně s číslem stránky předávají ovladači formou nepovinných parametrů metody. Formát zobrazení tabulky zachycuje obrázek 4.6. Pohled dále obsahuje odkaz pro vytvoření nového uživatele, který volá metodu `Register` ovladače `AccountController`.



Obr. 4.6: Tabulka uživatelů včetně kaskádových stylů.

Změnu uživatelského profilu administrátorem implementuje metoda `Edit`, která je přetížena a reaguje na metody HTTP protokolu GET a POST. Při volání metodou GET je nejprve vyhledán uživatelský profil předaný parametrem `id` pomocí číselného identifikátoru. Pokud daný uživatel existuje, zobrazí se pohled s formulářem, který umožňuje změnit všechna data včetně přihlašovacího jména uživatele – e-mailové adresy. Pokud uživatel nemá zvolenou ikonu, tudíž používá ikonu výchozí, příslušné pole se zobrazí prázdné. Ikona je nastavována vložením URL požadovaného obrázku do příslušného formulářového prvku. Pohled dále obsahuje odkaz pro změnu uživatelské role a vytvoření nového hesla.

Při volání metody `Edit` metodou GET se parametrem předá objekt třídy `UserDetail` obsahující aktualizovaný uživatelský profil. Nejprve je ověřena jedinečnost přihlašovacího jména uživatele, poté je prázdná hodnota URL ikony uživatele nahrazena URL výchozí ikony a záznam je aktualizován v databázi. Výsledek operace je formou chybového kódu předán metodě GET `Edit` jako parametr `result`. Ten může nabývat hodnoty `success` v případě úspěšného provedení

operace, `failure` v případě porušení jedinečnosti přihlašovacích jmen nebo `error` v případě jiné chyby (např. chyba zápisu do databáze).

Výše uvedená změna uživatelské role je realizována voláním metody `SwapRoles`. Metoda má jeden parametr `id` – číselný identifikátor uživatele. Pokud daný uživatel existuje, metoda otestuje, do které role uživatel přísluší. Poté je jeho role nahrazena rolí opačnou. V souvislosti s metodou `SwapRoles` bylo nutné vyřešit situaci, kdy se administrátor sám vzdá své role. Implementace metody výše zmíněné volání akceptuje. Není však možné, aby se aplikace vyskytovala ve stavu bez administrátorů. Řešením je ponechat prvotnímu uživateli roli administrátora bez možnosti odebrání role či smazání uživatele, neboť se zpravidla jedná o zřizovatele aplikace a neočekává se, že by svoji roli opustil.

Pokud uživatel ztratí heslo ke svému účtu, administrátor má možnost zřídit uživateli nové heslo. Z editace profilu uživatele je možné zavolat metodu `NewPassword`. Jedná se opět o metodu přetíženou. Při volání metodou GET je zobrazen pohled s formulářem pro zadání hesla. Odesláním je volána metoda `NewPassword` metodou POST. Samotná změna hesla je realizována prostřednictvím volání metod třídy `WebSecurity`, která umožňuje obnovení hesla. Metoda `GeneratePasswordResetToken` s parametrem předaným uživatelským jménem vrátí jedinečný řetězec, který je předán jako první parametr metodě `ResetPassword`. Druhým parametrem je nové heslo. Jelikož lze následkům ztráty hesla zabránit využíváním externí autentizace, není informování uživatele o novém hesle automatizováno. Administrátor by tedy měl uživatele o změně hesla neprodleně informovat.

Ke smazání uživatele slouží přetížená metoda `Delete`. Parametrem metody je číselný identifikátor uživatele. Při volání metody je nejprve zamezeno odstranění prvního uživatele. Poté je testována existence odstraňovaného uživatele. Následně je uživateli odebráno členství z role, již zastává, a na závěr je odstraněn samotný přihlašovací účet uživatele. Poté jsou v kaskádě volány metody `RemoveMembership` ovladače GroupController (viz kapitola 4.6), `DeleteUserLogin` ovladače AccountController a `SetPostAuthorZero` ovladače HomeController.

Při registraci nového uživatele je volána metoda `CreateDetail`, která vytvoří profil uživatele. Pokud se jedná o inicializaci prvního uživatele, je volána rozšířená metoda `InitFirst`, která zabezpečí vytvoření nového objektu pro správu databáze.

Druhou základní funkcí ovladače AdminController je změna uživatelského profilu prováděná uživateli samotnými. K tomuto účelu slouží přetížená metoda `Manage`. Při volání metodou GET se zobrazí pohled s formulářem pro změnu jména, příjmení a ikony uživatele. Metoda je přístupná z hlavního menu odkazem „Nastavení“ a poté odkazem „Změna údajů“ z pohledu pro změnu uživatelského hesla. Po odeslání formuláře je volána přetížená metoda metodou POST. Pokud jsou zadaná data platná, provede se aktualizace databáze.

4.6 Třída GroupController

Pomocí třídy GroupController je implementován ovladač pro správu diskusních skupin a členství uživatelů ve skupinách. Jelikož se jedná o služby určené přihlášeným uživatelům aplikace, je veškerý přístup k ovladači podmíněn autentizací uživatele. Vnitřní proměnná ovladače `db` je využita pro přístup k datovým modelům tříd `Group`, `GroupMembership` a `UserDetail`.

Metoda `Index` slouží k zobrazení pohledu pro správu skupin uživatele. Metoda nejprve získá seznam všech skupin souvisejících s uživatelem a předá je pohledu. Ten je zobrazen formou tabulky (viz obr. 4.7), v níž jsou zobrazeny skupiny, které patří uživateli nebo ve kterých má uživatel platné členství. Pro každou zde zobrazenou skupinu má uživatel možnost vstoupit (pokračováním na hypertextový odkaz zobrazený jako jméno skupiny), zobrazit ostatní uživatele, kteří jsou členy dané skupiny, nebo opustit skupinu, pokud však není jejím zakladatelem. V takovém případě je tato volba nahrazena možností zrušit skupinu. Skupiny jsou v tabulce řazeny abecedně a za účelem snadnějšího přístupu není tabulka stránkována.

Název	Členství	Možnosti
Moje skupina	Zakladatel	Členové | Zrušit skupinu
Testovací skupina	Člen	Členové | Opustit skupinu

Obr. 4.7: Tabulka skupin včetně kaskádových stylů.

Pohled dále obsahuje hypertextové odkazy volající metody, které slouží pro vytvoření nové skupiny a procházení existujících skupin.

Přetížená metoda `Create` implementuje vytváření nových skupin. Při volání metody metodou GET je zobrazen pohled obsahující formulář s jediným polem – názvem skupiny. Při odeslání formuláře je volána metoda `Create` metodou POST. Nejprve je testována validita názvu skupiny – neprázdnost a jedinečnost. V případě invalidního názvu je uživateli zobrazeno chybové hlášení prostřednictvím nepovinného parametru `status` metody reagující na metodu GET. Pokud je název v pořádku, je instanciován objekt třídy Group, který je následně naplněn hodnotami názvu skupiny a číselného identifikátoru vytvářejícího uživatele a uložen do databáze.

Procházení existujících skupin je realizováno prostřednictvím metody `Browse`. Metoda má dva nepovinné parametry: textový řetězec `name` pro filtrování zobrazených výsledků a celé číslo `page` pro zobrazení požadované stránky výstupu. Bez nepovinných parametrů je v pohledu tabulkově zobrazeno až dvacet skupin řazených abecedně podle názvu. Tabulka má obdobný formát jako v případě pohledu metody `Index`, není však zobrazen sloupec „Členství.“ Ve sloupci Možnosti je pro skupiny, kterých je uživatel platným členem, zobrazen hypertextový odkaz na výpis seznamu členů a dále jedna z následujících hodnot:

- Hypertextový odkaz „Přidat se“ v případě, kdy uživatel není členem skupiny.
- Text „Žádost odeslána“ v případě, kdy uživatel čeká na schválení zakladatelem skupiny.
- Text „Jste členem“ v případě platného členství.
- Text „Nemůžete se přidat“ v případě zamítnutí žádosti zakladatelem skupiny.

Nad tabulkou je zobrazen formulář pro filtrování výsledků podle jména skupiny. V případě většího počtu skupin než dvacet je pod tabulkou zobrazena navigace pro listování stránek.

Pro zobrazení seznamu uživatelů skupiny slouží metoda `Members`. Metoda má celkem čtyři parametry: povinný celočíselný parametr `id` reprezentující číslo skupiny, nepovinné řetězcové parametry `name` a `surname` pro filtrování výsledků a nepovinný celočíselný parametr `page` pro zobrazení požadované části stránkovaného obsahu. Při volání metody se nejprve ověří práva uživatele zobrazit členy skupiny – pouze administrátor nebo platný člen. Poté je vytvořen seznam členů dané skupiny, který je s respektováním hodnot parametrů filtru omezen na nejvýše dvacet položek, které jsou předány pohledu prostřednictvím parametru. V pohledu je zobrazena jedna ze dvou variant tabulky uživatelů. První variantou je řádkový výpis jmen a příjmení členů, který se zobrazuje uživatelům bez administrátorských nebo zakladatelských práv. Druhou variantou je výpis jmen a příjmení rozšířený o možnosti potvrdit či zamítnout žádost o členství nebo odstranit uživatele ze skupiny. Nad tabulkou je zobrazen formulář pro filtrování, pod tabulkou, obdobně jako u procházení skupin, navigace pro listování stránkovaným obsahem.

Vstup uživatele do skupiny zajišťuje metoda `Join`, která má jeden povinný celočíselný parametr `id`, který reprezentuje číslo skupiny. Metodu lze volat pouze v případě první interakce uživatele a skupiny. Algoritmus metody se dále větví podle role uživatele: administrátor se stává členem okamžitě, běžný uživatel pouze odešle žádost o členství zakladateli skupiny. V obou případech je instanciován objekt třídy `GroupMembership` obsahující identifikátory uživatele a skupiny a proměnné logických hodnot `confirmed` a `declined`, které uchovávají stav vyřízení žádosti o členství.

Komplementární metodou k výše uvedené metodě `Join` je metoda `Leave`, jež má sémanticky totožný parametr. Metodu může volat pouze uživatel s platným členstvím nebo čekající či zamítnutou žádostí o členství. Ve všech případech se uživatel vrátí do stavu bez interakce se skupinou, tj. objekt třídy `GroupMembership` reprezentující vztah uživatele a skupiny je smazán.

Pro zpracování žádosti o členství slouží dvojice komplementárních metod `Confirm` a `Decline`. Obě metody mají jeden celočíselný parametr `id`, který nabývá hodnoty číselného identifikátoru objektu reprezentujícího vztah uživatele a skupiny. Pokud zadaný vztah existuje, je metodami nastavena logická hodnota `true` příslušné proměnné objektu.

Ke zrušení členství jinému uživateli zakladatelem skupiny nebo administrátorem slouží metoda `Fire` s celočíselným parametrem `id` stejné sémantiky jako u metod `Confirm` a `Decline`. Pokud

parametrem předaný vztah existuje a volající uživatel má dostatečné oprávnění, je objekt reprezentující vztah uživatele a skupiny smazán.

Poslední metodou ovladače je metoda `RemoveMembership`, která je jednou z kaskádově volaných metod odstraňování uživatele. Celočíselný parametr `id` metody reprezentuje číselný identifikátor uživatele. Algoritmus metody je rozdělen do tří částí. Nejprve je všem skupinám (objektům třídy `Group`, jejichž zakladatelem je odstraňovaný uživatel) nastavena hodnota zakladatele na identifikátor přihlášeného uživatele – administrátora. Poté jsou odstraněny všechny objekty vztahů mezi odstraňovaným uživatelem a skupinami. V závěru jsou instanciovány objekty vztahů mezi přihlášeným uživatelem a skupinami, jež patřily odstraňovanému uživateli, ale přihlášený uživatel nebyl jejich členem. Tímto postupem jsem zajistil, že žádná skupina nezůstane v nekonzistentním stavu bez zakladatele, a tedy bez možnosti vyřizovat žádosti o členství.

4.7 Soubor Web.config

Pro úspěšné spuštění aplikace na serveru je nutné provést konfiguraci databázového připojení a jednotlivých součástí aplikace. K tomuto účelu je určen soubor Web.config. Jedná se o XML dokument obsahující popis nastavení ASP.NET serveru. Soubor byl vygenerován společně se šablonou, bylo však nutné provést jeho úpravy.

ASP.NET aplikace mají dva režimy zobrazování chybových hlášení:

- Plné zobrazení s detailním popisem chyby a výpisem zásobníku, které je vhodné pro vývoj a ladění aplikace.
- Uživatelské zobrazení využívající sdíleného pohledu Error.cshtml, které je vhodné pro běžný chod aplikace.

Pro volbu zobrazování chyb slouží značka `<customErrors mode="Off/On" />`, kterou bylo nutné přidat do sekce `configuration – system.web`. [25] Výhodou souboru Web.config v kontextu chybových hlášení je, že se nepřekládá do binární formy, ale je využíván ve svém standardním textovém formátu. Mohl jsem tedy při výskytu chyby mimo rámec ladění pouze změnit hodnotu `customErrors` bez nutnosti nového překladu a kopírování dat na server.

Pro našeptávač klíčových slov bylo zapotřebí oznámit serveru požadavek na využívání služeb AjaxControlToolkit, což bylo provedeno přidáním následujících značek do sekce `configuration – system.web – pages`:

```
<namespaces>
  <add namespace="System.Web.Mvc.Ajax" />
</namespaces>
<controls>
  <add tagPrefix="ajaxToolkit" assembly="AjaxControlToolkit"
       namespace="AjaxControlToolkit" />
</controls>
```

Při instalaci balíku AjaxControlToolkit prostřednictvím služby NuGet[1] se provedlo přidání výše uvedeného kódu do souboru Web.config automaticky.

Pro připojení k databázovému serveru slouží tzv. připojovací řetězce. Ty jsou uvedeny v sekci `configuration – connectionStrings`. Značka pro zapsání připojovacího řetězce má následující tvar:

```
<add name="název_připojení"
     connectionString="Data Source=URL databázového serveru;
                       Initial Catalog=název databázového katalogu;
                       User ID=přihlašovací jméno;
                       Password=heslo;
                       MultipleActiveResultSets=True/False
                       PersistSecurityInfo=True/False"
     providerName="přístup k poskytovateli dat" />,
```

kde položka `MultipleActiveResultSets` udává, zda je možné pracovat s více výsledky databázových dotazů současně, a `PersistSecurityInfo` rozhoduje, zda se mají citlivá data přenášet jako součást připojovacího řetězce. Parametr `providerName` určuje jmenný prostor tříd a metod pro komunikaci s poskytovatelem dat, např. v případě databázového serveru Microsoft SQL Server, který jsem využíval pro ladění aplikace, nabývá parametr `providerName` hodnoty `System.Data.SqlClient`. [26]

4.8 Shrnutí

Během implementace jsem se potýkal s několika problémy. Prvním problémem byla výše popsaná lokalizace validačních zpráv modelů, kde část hlášení byla zobrazena v podobě snadno modifikovatelného textového řetězce, zatímco jiná hlášení byla skryta v souborech programátorovi nepřístupných a vyžadovala zcela jiný systém řešení pomocí souboru zdrojů, přičemž tento fakt nebyl v oficiální dokumentaci ASP.NET zmíněn.

Dalším problémem byla kombinace dvou formulářů s našeptávači v jednom pohledu. Pro čistotu návrhu by bylo nejvhodnější, aby každý formulář byl implementován ve vlastním souboru a do výsledného pohledu byly tyto formuláře zahrnuty jako dva částečné pohledy. Při všech mnou testovaných způsobech takové implementace ale fungoval vždy jen první našeptávač. Nabízelo se tedy řešení mít funkční vždy pouze jeden našeptávač, což považuji za nepřípustné, nebo vytvořit jeden společný formulář a využitím absolutního pozicování kaskádovými styly rozmístit dva bloky formuláře na požadované pozice. Nevýhodou tohoto přístupu je situace, kdy uživatel nemá k dispozici prohlížeč s kaskádovými styly. V takovém případě se hlavní menu aplikace zobrazí na jiném místě než pole pro vyhledávání, stránka ovšem nadále zůstává bez sebemenších problémů plně čitelná.

[1] Další informace dostupné na http://nuget.org/

Posledním a nejvýznamnějším problémem byla nemožnost změny uživatelského jména. Třída `Membership` sice implementuje metody pro vrácení a modifikaci objektu uživatele, ale většina proměnných tohoto objektu, včetně proměnné `UserName`, je určena jen ke čtení, nikoliv k zápisu. Mnou navržené řešení tohoto problému s dvěma uživatelskými jmény pro jednoho uživatele zdaleka není univerzální a funguje jen díky faktům, že uživatelským jménem je e-mailová adresa a jeden uživatel nemá více účtů. Technicky nejčistším způsobem autentizace je tedy externí přihlášení pomocí OpenID.

Vzhledem ke stále hojnějšímu využívání mobilních zařízení pro přístup k internetu jsem aplikaci testoval na některých mně dostupných mobilních zařízeních, konkrétně na běžném mobilním telefonu, mobilním telefonu s operačním systémem Android a v experimentálním webovém prohlížeči čtečky elektronických knih Amazon Kindle 4. Ve všech případech se aplikace i přes všechny výše uvedené problémy zobrazovala konzistentně a fungovala bez omezení, což považuji za její silnou stránku. Pro mobilní přístup tedy nebude nutné vyvíjet aplikace ve stylu služeb Twitter či Tumblr, ale bude umožněn okamžitě.

5 Závěr

V úvodu této práce jsem si stanovil cíl, kterým bylo vytvořit webovou aplikaci, mikroblog, pro týmovou komunikaci. Studiem a analýzou existujících služeb, které obsahovaly pro moji práci důležité myšlenky, jsem se zabýval ve druhé kapitole, která je výstupem prvního bodu zadání. Ve třetí kapitole jsem vytvořil návrh aplikace, čímž jsem zpracoval druhý bod zadání. Zabýval jsem se stěžejní koncepcí z hlediska plánovaného využití aplikace. Grafické uživatelské rozhraní jsem navrhnul tak, aby se uživatel, zvyklý na již existující služby tohoto typu, se v aplikaci snadno orientoval. Z implementačního hlediska jsem srovnával různé přístupy k realizaci důležitých částí aplikace a vyhodnocoval jejich přínosy a možná rizika. Třetí bod zadání jsem vypracoval ve čtvrté kapitole, kde jsem se zabýval popisem samotné implementace. V podkapitole Shrnutí jsem popsal problémy, se kterými jsem se při implementaci setkal, a výsledek testování aplikace na mobilních zařízeních.

Po analýze prostředí, zpracování návrhu, vytvoření struktury aplikace, implementaci jednotlivých tříd s příslušnými metodami a vzájemném propojení všech součástí se mi podařilo vytvořit funkční aplikaci, která splňuje všechny mnou stanovené cíle a požadavky na výslednou funkcionalitu. Dle mého názoru je zvláště důležitá použitelnost v mobilních zařízeních, jejichž popularita stále stoupá.

I přes splnění všech mnou kladených požadavků aplikace stále nabízí mnoho možností k dalšímu rozšíření. Mezi nejpřínosnější rozšíření by mohla patřit např. integrace s operačním systémem Google Android. Jelikož aplikace využívá autentizaci Google OpenID, byla by tato možnost pro uživatele zcela jistě vítaná.

Další rozšíření by se mohlo týkat sdílení obrázků, videí nebo souborů v příspěvcích. V prostředí ASP.NET by bylo možné danou funkčnost doplnit např. s využitím tzv. snippetů, krátkých úseků kódu doplňovaných do textu příspěvků.

V neposlední řadě by bylo možné rozšířit aplikaci o vytváření událostí, k nimž by se mohli uživatelé přihlašovat obdobným systémem, jakým se uživatelé stávají členy diskusních skupin. Z implementačního hlediska by takové rozšíření vyžadovalo vytvořit novou třídu ovladače s příslušnými pohledy a datovými modely.

Vývojem by také mohlo projít uživatelské rozhraní, které by se mohlo stát ještě intuitivnějším a v kombinaci s výše uvedeným rozšířením o události by mohlo včas zobrazovat upozornění na blížící se termín události.

6 Literatura

[1] Mikroblog. In: *Symbio: Internetová agentura* [online]. [cit. 2012-12-12]. Dostupné z: http://www.symbio.cz/slovnik/mikroblog.html

[2] NAONE, Erica. A Brief History Of Microblogging. In: *Technology review: MIT's magazine of innovation* [online]. 2008 [cit. 2012-12-11]. ISSN 1099-274x. Dostupné z: http://www.technologyreview.com/sites/default/files/legacy/forward.pdf

[3] LENNON, Andrew. A Conversation With Twitter Co-Founder Jack Dorsey. In: *The Daily Anchor: Marketing Blog* [online]. 2009 [cit. 2012-12-12]. Dostupné z: http://www.thedailyanchor.com/2009/02/12/a-conversation-with-twitter-co-founder-jack-dorsey/

[4] ARRINGTON, J. Michael. Odeo Releases Twttr. In: *TechCrunch* [online]. 2006 [cit. 2012-12-12]. Dostupné z: http://techcrunch.com/2006/07/15/is-twttr-interesting/

[5] Twitter. *Wikipedia, the free encyclopedia* [online]. 2012, 2012-12-10 [cit. 2012-12-11]. Dostupné z: http://en.wikipedia.org/wiki/Twitter

[6] BUSCH, Michael. Twitter's New Search Architecture. In: *Twitter: Engineering Blog* [online]. 2010 [cit. 2012-12-12]. Dostupné z: http://engineering.twitter.com/2010/10/twitters-new-search-architecture.html

[7] GADE, Krishna. Twitter Search is Now 3x Faster. In: *Twitter: Engineering Blog* [online]. 2011 [cit. 2012-12-12]. Dostupné z: http://engineering.twitter.com/2011/04/twitter-search-is-now-3x-faster_1656.html

[8] VENNERS, Bill. Twitter on Scala: A Conversation with Steve Jenson, Alex Payne, and Robey Pointer. In: *Artima developer: Best practises in enterprise software development* [online]. 2009 [cit. 2012-12-12]. Dostupné z: http://www.artima.com/scalazine/articles/twitter_on_scala.html

[9] Yammer. *Wikipedia, the free encyclopedia* [online]. 2012, 2012-10-31 [cit. 2012-12-11]. Dostupné z: http://en.wikipedia.org/wiki/Yammer

[10] RAO, Leena. Yammer Debuts A Facebook For The Enterprise. In: *TechCrunch* [online]. 2010 [cit. 2012-12-12]. Dostupné z: http://techcrunch.com/2010/09/28/yammer-debuts-a-facebook-for-the-enterprise/

[11] Scala at Yammer. In: *Yammer: Engineering* [online]. 2010 [cit. 2012-12-12]. Dostupné z: http://eng.yammer.com/scala-at-yammer/

[12] Tumblr. *Wikipedia, the free encyclopedia* [online]. 2012, 2012-12-04 [cit. 2012-12-11]. Dostupné z: http://en.wikipedia.org/wiki/Tumblr

[13] Managing and creating blogs. In: *Tumblr* [online]. [cit. 2012-12-12]. Dostupné z: http://www.tumblr.com/docs/en/blogs

[14] Tumblerette is now the official Tumblr iPhone App! (and it's free!). In: *Tumblr Staff* [online]. 2009 [cit. 2012-12-12]. Dostupné z: http://staff.tumblr.com/post/81463386/iphone-app

[15] Re: PHP Framework?. In: *Marco.org* [online]. 2008 [cit. 2012-12-12]. Dostupné z: http://www.marco.org/2008/10/20/re-php-framework

[16] API. In: Tumblr [online]. [cit. 2012-12-12]. Dostupné z: http://www.tumblr.com/docs/en/api/v2

[17] ASP.NET MVC Overview. *The Official Microsoft ASP.NET Site* [online]. 2012 [cit. 2012-12-22]. Dostupné z: http://www.asp.net/mvc/tutorials/older-versions/overview/asp-net-mvc-overview

[18] AutoComplete Sample. *ASP.NET AJAX Control Toolkit* [online]. 2006-2011 [cit. 2012-12-13]. Dostupné z: http://www.asp.net/ajaxLibrary/AjaxControlToolkitSampleSite/AutoComplete/AutoComplete.aspx

[19] How Do I Create an Auto-Complete TextBox?: ASP.NET Ajax Library. *The Official Microsoft ASP.NET Site* [online]. 2012 [cit. 2012-12-13]. Dostupné z: http://www.asp.net/ajaxlibrary/jquery_mvc_autocomplete.ashx

[20] EVJEN, Bill. *ASP.NET 3.5 v jazycích C# a Visual Basic: programujeme profesionálně*. Vyd. 1. Brno: Computer Press, 2009, 791 s. ISBN 978-80-251-2069-9.

[21] PIALORSI, Paolo a Marco RUSSO. *Microsoft LINQ: kompletní průvodce programátora*. Vyd. 1. Brno: Computer Press, 2009, 615 s. ISBN 978-80-251-2735-3.

[22] OpenID. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-04-14]. Dostupné z: http://cs.wikipedia.org/wiki/OpenID

[23] Federated Login for Google Account Users: Google Accounts Authentication and Authorization. GOOGLE. *Google Developers* [online]. 2013, 2013-02-26 [cit. 2013-04-14]. Dostupné z: https://developers.google.com/accounts/docs/OpenID

[24] Localizing ASP.NET MVC Validation. In: HAACK, Phill. *Haacked* [online]. 2009 [cit. 2013-04-17]. Dostupné z: http://haacked.com/archive/2009/12/07/localizing-aspnetmvc-validation.aspx

[25] <customErrors> Element. In: *Msdn* [online]. 2013 [cit. 2013-04-23]. Dostupné z: http://msdn.microsoft.com/en-us/library/h0hfz6fc(v=VS.71).aspx

[26] SQL Server Connection Strings for ASP.NET Web Applications. In: *Msdn* [online]. 2013 [cit. 2013-04-23]. Dostupné z: http://msdn.microsoft.com/en-us/library/jj653752.aspx

Seznam obrázků

A. Obsah CD

- Složka bin:
  - Přeložená aplikace pro spuštění na ASP.NET serveru.
- Složka doc:
  - Dokumentace ve formátu DOC a PDF.
  - Soubor README.txt s popisem spuštění aplikace na serveru.
- Složka src:
  - Zdrojové kódy aplikace.
  - Složka sql:
    - Soubory s příkazy jazyka SQL pro inicializaci databáze.