

MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY



Webové aplikace pracující v reálném čase

DIPLOMOVÁ PRÁCE

Jaromír Nyklíček

Brno, 2013



ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Jaromír Nyklíček

Program, obor (příp. specializace): Informační systémy

Garant oboru (příp. specializace): RNDr. Vlastislav Dohnal, Ph.D.

Vedoucí práce: RNDr. Radek Ošlejšek, Ph.D.

Katedra: KPSK

Název práce: Webové aplikace pracující v reálném čase

Zadání:

Prostudujte techniky a postupy, které v prostředí webu zajišťují dodávání informací klientům v okamžiku jejich zveřejnění bez nutnosti aktivního požadavku klienta. Pomocí webových technologií vytvořte aplikaci pro helpdesk.

Teoretická část práce bude popisovat teoretické koncepty a postupy, které tvorba těchto specifických aplikací vyžaduje. Dále bude popisovat technologie použité pro implementaci aplikace i samotnou aplikaci.

V praktické části bude implementován chat systém pro helpdesk s použitím technologií Node.js a HTML5 Web Sockets resp. frameworku socket.io. Chat bude integrován do Jabberu.

Základní literatura:

CRANE D., McCARTHY P. *Comet and Reverse Ajax: The Next-Generation Ajax 2.0*. Berkeley : aPress, 2008. 148 s. ISBN 978-1-59059-998-3

KUUSKERI, J., MIKKONEN, T. Partitioning web applications between the server and the client. In: *Proceedings of the 2009 ACM symposium on Applied Computing*. ACM, 2009. s. 647-652.

MIKKONEN, T., TAIVALSAARI A. Web Applications — Spaghetti Code for the 21st Century. In: *Proceedings of the 6th ACIS International Conference on Software Engineering Research, Management and Applications (SERA'2008, Prague, Czech Republic, August 20-22, 2008)*. IEEE Computer Society, s.319-328.

Souhlasím se zadáním (podpis, datum)

Nyklíček, 18.3.2013

student(ka)

vedoucí
diplomové práce

garant
oboru

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Jaromír Nyklíček

Vedoucí práce: RNDr. Radek Ošlejšek, Ph.D.

Poděkování

Chtěl bych poděkovat vedoucímu práce Radku Ošlejškovi za odborné vedení, cenné podněty a velmi vstřícný přístup během psaní této práce.

Shrnutí

Cílem práce bylo zmapovat historické a zejména současné trendy, metodiky a postupy v tvorbě webových aplikací, které pracují v reálném čase a umožňují tak příjemci obsahu informace získávat v okamžiku, kdy je autor vydá. Po nastudování problematiky i současných poznatků byla navržen a implementován helpdesk systém Aidbot, který prostřednictvím webového klienta a XMPP serveru umožní poskytovat technickou podporu návštěvníkům a uživatelům webových služeb.

Klíčová slova

Comet, Polling, Long polling, Piggybacking, Ajax push, Server push, Reverse Ajax, Real-time web, Web v reálném čase, klient-server, HTTP, REST, Jabber, XMPP, Node.js, WebSocket

Obsah

1	Úvod	1
2	Základní pojmy	3
2.1	Webová služba	3
2.2	Architektura klient-server v prostředí webu	4
2.3	AJAX	5
2.4	Webová aplikace	6
2.4.1	Bohatá internetová aplikace	7
2.4.2	Webová aplikace pracující v reálném čase	8
2.5	Příklady webových aplikací pracujících v reálném čase	9
2.5.1	Google Wave a Apache Wave	10
2.5.2	Google Apps	11
2.5.3	Office Web Apps	12
2.5.4	Basecamp	12
2.5.5	Sociální sítě	12
3	Techniky zavedení prvku reálného času	14
3.1	Polling	15
3.1.1	Piggybacking	16
3.2	Comet	17
3.2.1	Long polling	19
3.2.2	Specializované Comet servery	19
3.2.3	Protokol Bayeux	20
3.3	Protokol a aplikační rozhraní WebSocket	20
3.3.1	Firewally a proxy servery	21
3.3.2	Podpora prohlížečů	21
4	Návrh řešení	23
4.1	Základní případy užití	23
4.1.1	Další případy užití	23
4.1.2	Motivace	25
4.2	Definice požadavků	25
4.2.1	Funkční požadavky	25
4.2.2	Ostatní požadavky	26
4.3	Architektura systému	27
4.3.1	Serverová část	28
5	Implementace	30
5.1	Použité technologie	30
5.1.1	REST a RESTful webové služby	30
5.1.2	XMPP	32
5.1.3	Node.js	32
5.1.4	Socket.IO	34
5.1.5	AngularJS	35

5.2	Detaily řešení vybraných problémů	36
5.2.1	Zasílání zpráv mezi klientem a serverem	36
5.2.2	RESTful webová služba	37
5.2.3	Ukládání dat	38
5.2.4	Rozhraní pro XMPP server	39
5.2.5	Integrace komunikačního okna	40
5.2.6	Ukládání časových údajů	40
6	Závěr	42
	Literatura	43
A	Dokumentace rozhraní webové služby	47
B	Obsah přiloženého CD	55

Seznam obrázků

2.1	Schéma standardní HTTP komunikace [7].	4
2.2	Schéma komunikace prohlížeče se serverem.	6
2.3	Obrazovka aplikace <i>Google Wave</i> s otevřenou „vlnou“ [24].	11
3.1	Fungování pollingu ve webové aplikaci.	16
3.2	Piggybacking s vynulováním časovače pro snížení počtu HTTP požadavků [7].	17
4.1	Diagram případů užití.	24
4.2	Schéma architektury systému Aidbot.	29
5.1	Uživatelské rozhraní administrace vytvořené v AngularJS.	36

1 Úvod

Systém World Wide Web (dále jen „web“), který prostřednictvím protokolu HTTP umožňuje distribuovat hypertextové dokumenty a další obsah, byl poprvé představen ve výzkumném centru CERN¹ v roce 1990. Patrně ani tvůrce tohoto systému Tim Berners-Lee nepředpokládal, jaký obrovský rozvoj v následujících více než dvou desetiletích prodělá. Web se během velice krátké doby zařadil po bok e-mailu a stal se tak jednou z nejdůležitějších služeb v Internetu.

Jelikož je webový prohlížeč v současné době dostupný v téměř každém zařízení, které disponuje připojením k Internetu, stal se web postupně velmi vyhledávanou platformou pro tvorbu nových aplikací.

Architektura klient-server (v tomto případě navíc ještě zjednodušená tím, že webový prohlížeč je téměř nejtenčí možný klient) přináší tvůrcům aplikací velmi mnoho výhod. Namátkou – multiplatformnost, snadnou údržbu, rychlé aktualizace, do jisté míry i vyšší bezpečnost. Stejně tak ale bohužel přináší i určité nevýhody, které se však v průběhu posledních dvou až tří let daří čím dál tím úspěšněji řešit. Elegantní řešení jsou možná zejména díky novým standardům specifikovaných konsorciem W3C² a také díky moderním prohlížečům, které tyto standardy velmi dobře implementují.

Jedním z problémů, který byl ještě před několika lety řešitelný pouze částečně, byla automatická aktualizace dat na straně klienta v okamžiku jejich změny (nebo vytvoření) na serveru. I toto neúplné řešení s sebou neslo buď určité nepohodlí pro uživatele, nebo zvýšené nároky na systémové zdroje serveru (v prvotní implementaci dokonce oba tyto zápory).

Právě přiblížit řešení problematiky automatických aktualizací dat na klientovi v reálném čase, které je nutné pro tvorbu moderní webové aplikace, je jedním z cílů, které si klade tato diplomová práce. Druhým z nich je praktická implementace webového systému pro technickou podporu zákazníků, který bude nabyté teoretické poznatky z této oblasti uvádět do praxe.

Samotná práce je rozdělena do šesti kapitol. Po úvodu následuje kapitola, která vyjasňuje a definuje základní pojmy, popisuje obecné koncepty webových aplikací a ukazuje rozdíly mezi „klasickými“ webovými aplikacemi a těmi, které fungují v reálném čase. Zároveň uvádí příklady webových aplikací, jejichž alespoň nějaká část v reálném čase pracuje.

1. <http://home.web.cern.ch/>

2. World Wide Web Consortium – hlavní mezinárodní standardizační organizace pro web.

Třetí kapitola ukazuje různé praktické možnosti zavedení prvku reálného času do webových aplikací. Popisuje nejen historické, ale zejména aktuální trendy a postupy řešení této problematiky, zdůrazňuje jejich klady i zápory a navzájem je porovnává.

Čtvrtá kapitola se věnuje převážně analýze vznikajícího softwaru *Aidbot*. V první části obsahuje popis samotného systému, který má sloužit pro přímou technickou podporu uživatelů webu prostřednictvím okna prohlížeče a motivaci pro jeho tvorbu. Těžištěm kapitoly je sběr požadavků a popis případů užití systému. Závěr kapitoly je pak věnován náčrtu celkové architektury systému a rozdělení jednotlivých funkcionalit programu mezi serverovou a klientskou část.

Na čtvrtou kapitolu úzce navazuje pátá část práce, která podává technickou zprávu o vlastní implementaci. Zahrnuje výčet použitých technologií a argumentaci jejich volby. Novější technologie, které nejsou širšímu publiku natolik známy, jsou zde zevrubně popsány a jsou diskutovány výhody, které jejich užití přinese. Dále je tato kapitola věnována zejména popisu jednotlivých detailů aplikace a také obtížím, které se během realizace vyskytly a procesu hledání jejich řešení.

Závěrečné kapitole pak náleží zejména shrnutí dosavadní práce na programu a možnosti dalšího vylepšování a směřování. Druhá část závěrečné kapitoly je pak věnována diskusi možností tvorby webových aplikací s prvky provozu v reálném čase v prostředí současného webu.

2 Základní pojmy

Během poslední dekády se web postupně vyvinul ze systému pro šíření informací do plnohodnotné aplikační platformy [1]. Vývoj aplikací pro takovou platformu ale má, vzhledem k její architektuře i použitým protokolům, určitá specifika a omezení, kterými se velmi liší od vývoje nativních aplikací pro stolní počítače či mobilní zařízení. Právě tyto odlišnosti jsou v následujících odstavcích popsány.

Zároveň tato část práce představuje obecné aspekty webu a webových aplikací a definuje některé pojmy, které jsou v práci dále používány.

2.1 Webová služba

Webovou službu můžeme jednoduše popsat jako způsob, kterým lze zpřístupnit funkcionalitu nějakého softwarového systému (např. informačního) prostřednictvím standardních webových technologií tj. HTTP¹, HTML², webového serveru a webového klienta (typicky prohlížeče). Použití těchto technologií redukuje heterogenitu a zvyšuje interoperabilitu systému [2].

Přesnější definice uváděná konsorciem W3C říká, že webová služba je softwarový systém užívaný pro interoperabilní komunikaci mezi dvěma a více systémy, a to s použitím počítačové sítě; zároveň je podle této definice součástí webové služby i popis rozhraní dané služby ve strojově čitelném formátu (typicky WSDL³). Skrze toto rozhraní s webovou službou komunikují ostatní systémy pomocí XML⁴ serializovaných SOAP⁵ zpráv zasílaných s užitím protokolu HTTP [3].

Jak napovídá definice uvedená v prvním odstavci, webovou službu lze vytvořit a komunikovat s ní i prostřednictvím odlišných a mnohdy i jednodušších způsobů než je WSDL a zasílání SOAP zpráv. Jednou z těchto možností je tzv. RESTful Web API,⁶ které našlo využití mj. i v praktické části této diplomové práce (viz kapitoly 4.3 a 5.1.1). Tento způsob využívá ke komunikaci

-
1. Hypertext Transfer Protocol
 2. Hypertext Markup Language
 3. Web Services Description Language. Jazyk na bázi XML určený k popisu dat obsažených ve webové službě a dotazů, které je možné na tuto službu klást.
 4. XML – zkratka Extensible Markup Language. Obecný značkovací jazyk užívaný k vytváření konkrétních značkových jazyků pro danou problémovou doménu.
 5. Simple Object Access Protocol. Protokol pro výměnu konkrétních objektů přes HTTP.
 6. API – zkratka Application Programming Interface, česky „aplikační rozhraní.“ API slouží pro vzájemnou interakci mezi aplikacemi. Do jisté míry ho tak lze označit za protipól uživatelského rozhraní, které slouží pro interakci člověka a aplikace.

s webovou službou přímo jednotlivé metody (např. GET, POST) HTTP protokolu bez přidání zprávkové vrstvy [4].

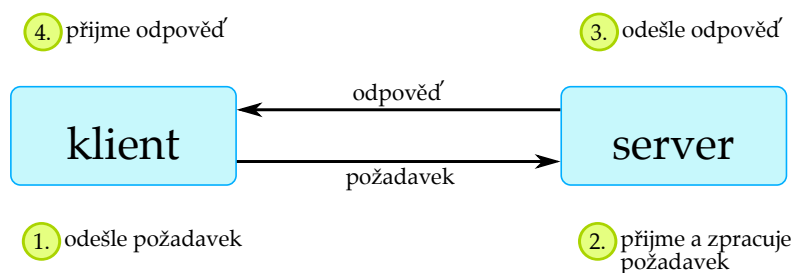
Vzhledem k tomu, že webové služby jsou vystavěny na osvědčených technologiích, poskytují robustní softwarový rámec pro interakci mezi jednotlivými, nejen webovými, aplikacemi. Jednotliví provozovatelé také mohou své webové služby přidat do otevřeného registru UDDI,⁷ což umožní jejich snadné vyhledání a integraci do dalších systémů.

2.2 Architektura klient-server v prostředí webu

Přístup označovaný jako klient-server se od svého vzniku v laboratořích firmy Xerox v sedmdesátých letech 20. století stal jednou z nejrozšířenějších architektur síťových aplikací [5]. Důvodem její popularity je zejména její jednoduchost a srozumitelnost, jasné rozdělení zodpovědností a centralizovaná správa [5]. Za nevýhodu lze označit menší robustnost než je např. u P2P⁸ sítí.

Implementace této architektury v prostředí webu ale na rozdíl od jiných síťových aplikací přináší určitá specifika. Kvůli protokolu HTTP byl web již od svého počátku projektován jako výhradně „jednosměrné“ médium – to znamená, že veškerá komunikace je zahajována ze strany klienta [6] a server pouze pasivně naslouchá jeho požadavkům, které následně odbavuje.

Toto omezení je důsledkem *stateless*⁹ povahy HTTP protokolu. [6]. To znamená, že za každým jednotlivým požadavkem na server ze strany klienta následuje odpověď serveru a tato dvojice je považována za samostatnou transakci, která není v žádném vztahu s předchozími ani následujícími požadavky klienta [6].



Obrázek 2.1: Schéma standardní HTTP komunikace [7].

7. Universal Description, Discovery and Integration, <http://uddi.org>

8. P2P – zkratka Peer-to-Peer. Druh počítačových systémů, kde spolu komunikují přímo jednotliví klienti (uživatelé).

9. „Bezstavovost.“ V některé literatuře, např. v [7], je HTTP označováno jako *request-response* protokol. V zásadě se ale jedná o popis stejné vlastnosti.

2.3 AJAX

V roce 1995 byl společností Netscape Communications vyvinut a představen jazyk Javascript jehož interpret byl nedlouho poté implementován do prohlížeče Navigator. Bylo tak umožněno, aby se součástí webových stránek stal kód v tomto jazyce, který byl následně prováděn přímo na straně klienta. To mimo jiné i dovoluje reagovat na události, které nelze odeslat ke zpracování na server (např. změna polohy kurzoru v textu, označení části stránky myší atd.), čímž bylo dosaženo zvýšené interaktivity a uživatelské přívětivosti webových stránek.

Právě možnost vykonávat kód na straně klienta položila základ pro technologii AJAX¹⁰, která byla představena o několik let později. Pomocí objektu typu XMLHttpRequest [8] umožňuje asynchronně (tzv. „na pozadí“, tedy bez toho, aby se uživateli aktualizovalo okno prohlížeče) vytvořit požadavek, odeslat jej na server a následně odpověď zpracovat a upravit objektový model dokumentu, tak aby reflektoval získaná data [7] [8]. Třída XMLHttpRequest je součástí aplikačního rozhraní všech standardních prohlížečů, stejně jako například objekt typu Window pro manipulaci s oknem prohlížeče.

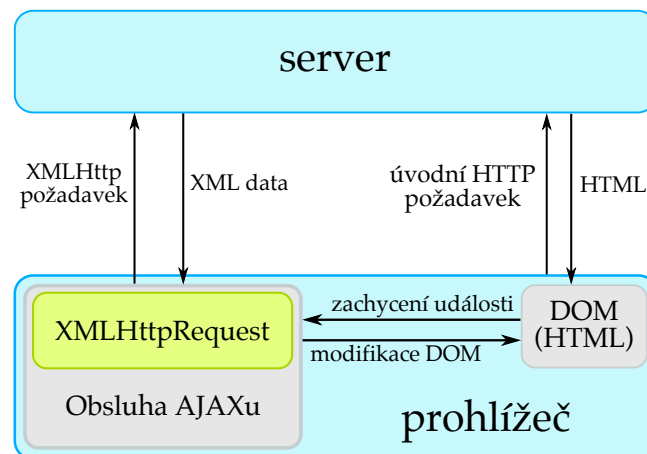
Jádro implementace je velmi triviální (viz obr. 2.2). Nejprve je zachycena událost od uživatele, např. klepnutí na odkaz. Následně je vytvořena nová instance třídy XMLHttpRequest a té jsou prostřednictvím k tomu určených metod nastaveny následující atributy:

- **URL adresa**, na kterou má požadavek proběhnout;
- **HTTP metoda**, jež se má použít;
- **HTTP hlavičky** požadavku;
- **posluchač událostí („event listener“)** – funkce která je automaticky provedena, jakmile se změní vnitřní stav objektu (typicky tedy při ukončení HTTP požadavku). Tato funkce obstarává zpracování přijatých dat a ošetření chybových stavů.

Původně, jak název napovídá, bylo jedinou možností pro přenos dat použití formátu XML. I přes rozpor s názvem technologie je dnes mnohem rozšířenější přenos dat prostřednictvím formátu JSON¹¹ než XML [9].

10. Asynchronous Javascript And XML

11. JSON - zkratka Javascript Object Notation. Textový a na jazyce nezávislý formát pro výměnu dat. Je založen na podmnožině programovacího jazyk JavaScript [10].



Obrázek 2.2: Schéma komunikace prohlížeče se serverem.

2.4 Webová aplikace

Shklar a Rosen v [11] webovou aplikaci definují jako klient-server aplikaci, která na straně klienta využívá webový prohlížeč a poskytuje interaktivní službu prostřednictvím Internetu. Zároveň webová aplikace doručuje obsah, jenž je dynamicky generovaný v závislosti na parametrech požadavku [11].

Tato definice byla patrně platná v kontextu doby svého vzniku (2003), ale pro dnešní vnímání pojmu „webová aplikace“ je příliš široká, z čehož pramení i její relativní nepřesnost. Uvažujeme-li např. libovolný zpravodajský web, můžeme téměř se 100% jistotou říci, že jednotlivé články budou dynamicky zobrazovány prostřednictvím nějakého systému pro správu obsahu na základě čtenářových preferencí nebo popularity článku. Přesto však z dnešního pohledu nemůžeme mluvit o tom, že by zpravodajský portál byl webovou aplikací.

Pro zpřesnění této definice se tak nabízí zavedení následujícího členění webových aplikací:

- **Informačně orientované** – aplikace specializované primárně na zobrazování obsahu, např. zpravodajské portály, blogy apod.
- **Transakčně orientované** – specializované na podporu provádění transakcí, např. elektronické obchody a soutěže, aukční systémy.
- **Aplikačně orientované** – dodávají nějakou netriviální funkcionalitu nebo službu. Např. interaktivní mapy, RSS čtečka, e-mailový klient. Obecně lze říci, že všechny aplikace z této kategorie odpovídají definici *Software as a Service* (více viz kap. 2.5).

World Wide Web Consortium v [12] naproti tomu přináší definici mnohem exaktnější. Webová aplikace pracuje prostřednictvím protokolu HTTP a její interní komunikace je zpracovávána strojově. Webové aplikace jsou obvykle vystavěny nad existující webovou architekturou a infrastrukturou a jsou nezávislé na platformě nebo použitém programovacím jazyce. Zároveň velmi často umožňují znovupoužitelnost své funkcionality, a to prostřednictvím interoperability s jinými webovými aplikacemi či nativními aplikacemi pro osobní počítače nebo mobilní zařízení. V neposlední řadě je vyžadováno, aby obsah jimi zasílaných zpráv byl „sémanticky čistý“, čehož lze dosáhnout použitím samopopisujících formátů jako je JSON nebo XML [12].

Webové aplikace jsou v současné době základním kamenem všech interaktivních webových stránek, neboť dynamicky vytvářejí odpovědi na příchozí HTTP požadavky [13].

Technicky se serverová část webové aplikace skládá z HTTP serveru, aplikačního serveru a databáze, popř. pouze z aplikačního serveru a databáze [2]. Pokud je použit i obecný HTTP server, je využit zejména pro distribuci statického obsahu (soubory, obrázky atp.), zatímco aplikační server odbavuje požadavky na zobrazení dynamického obsahu (stránky a další obsah, který je vytvářen přímo webovou aplikací) [14]. Systém řízení báze dat je nutný jako poskytovatel datového úložiště a zároveň jako nástroj k udržování jeho konzistence.

Současný trend v tvorbě webových aplikací po malých krocích směřuje od klasických relačních databází k tzv. NoSQL neboli bezschémovým databázím. Ty na rozdíl od databází relačních neukládají data do tabulek, ale do tzv. *kolekcí*. Hlavní přidanou hodnotou tohoto druhu databází je rychlost čtení a snadná škálovatelnost, nevýhodou naopak alespoň prozatímni technologická nevyspělost a složitá integrace business intelligence nástrojů pramenící právě z neexistence striktního schématu dat [15].

2.4.1 Bohatá internetová aplikace

Bohaté internetové aplikace (zkráceně RIA – Rich Internet Applications, někdy také označovány jako Rich Web Applications) umožňují uživateli bohatší interakci s aplikací a vyšší uživatelský komfort [16]. Toho dosahují zejména masivním klientským programováním, kdy je velká část funkcionality, kterou běžně obsahuje server, přenesena na klienta.

Příkladem může být např. komponentová logika a šablonovací systém, které jsou v klasických webových aplikacích doménou serveru. Bohaté internetové aplikace umožňují vykonávání logiky na klientovi, čímž dochází k velkému zrychlení odezvy uživatelského rozhraní.

Zvýšení uživatelského prožitku umožňuje také technologie AJAX. V RIA data většinou nejsou odesílána na server v okamžiku, kdy uživatel např. vyplní formulář, ale jsou nejprve uložena lokálně a pak odeslána asynchronně, čímž je pro uživatele snížena doba odezvy aplikace [17].

Cílem tvorby bohatých internetových aplikací je využít výhody webových aplikací (dostupnost, multiplatformnost, minimální nároky na hardware) a zároveň dosáhnout uživatelského komfortu, který je běžný z nativních aplikací.

Tvorba bohatých internetových aplikací s sebou tak přináší zvýšené nároky na počáteční analýzu, neboť je potřeba velmi dobře rozvrhnout, která funkcionality bude vykonávána na klientovi a která na serveru [18]. S nároky na analýzu souvisí také obtíž s udržitelností kódu. U mnoha částí RIA aplikací dochází k duplikování klientské funkcionality na serveru. Důvodem je zejména zajištění alespoň částečné kompatibility pro starší verze prohlížečů [19].

2.4.2 Webová aplikace pracující v reálném čase

Konvenční webové aplikace fungují způsobem popsaným v podkapitole 2.2, tedy tak, že je komunikace zahájena vždy ze strany klienta. Tato metoda bývá v literatuře označována jako tzv. *pull* [7].

Webová aplikace pracující v reálném čase je naopak taková aplikace, která umožňuje, aby byla komunikace zahájena ze strany serveru. Tato metoda získávání dat bývá označována jako tzv. *push* [7]. *Push* je zaslání dat ze serveru klientovi bez toho, aniž by si je vyžádal, a to právě v okamžiku jejich vydání nebo s naprosto minimálním zpožděním.¹²

Crane a McCarthy v [7] uvádějí několik základních případů užití, pro které je vhodné použít webovou aplikaci pracující v reálném čase; jako nejdůležitější z nich můžeme zmínit následující:

- **Sledování změn**

V prostředí webových aplikací může snadno dojít k situaci, kdy HTTP požadavek na serveru vyvolá spuštění dlouhotrvajícího procesu. Dodávat uživateli v reálném čase průběžné zprávy a dílčí výsledky je efektivní způsob, kterým jej lze informovat o aktuálním stavu procesu.

Dalším podobným příkladem mohou být sociální sítě, neboť v nich každým okamžikem vzniká velké množství obsahu od jednotlivých jejich uživatelů. Kombinace webové aplikace pracující v reálném čase s vhodným algoritmem, který bude dané příspěvky filtrovat podle osobních

12. Wikipedia na stránce http://en.wikipedia.org/wiki/Real-time_web uvádí maximální prodlevu v délce jedné sekundy. Článek bohužel neobsahuje žádné relevantní primární zdroje, proto je nutné tento údaj brát s rezervou.

preferencí, je téměř ideálním způsobem jak uživateli doručovat obsah, který jej zajímá, a to v okamžiku, kdy je aktuální.

Zajímavým uplatněním takových aplikací jsou také webové aplikace, které mají přímou návaznost na finanční sektor. Příkladem může být sledování růstu či poklesu cen obchodovaných akcií, monitorování vývoje na komoditních burzách apod.

- **Komunikace a vzdálená spolupráce**
Charakteristickou vlastností všech nástrojů pro vzdálenou spolupráci je, že v jednom okamžiku může být aktuální stav aplikace změněn více než jedním uživatelem. Veškeré provedené změny je pak potřeba v co nejkratším možném čase distribuovat mezi všechny připojené uživatele a zajistit tak, aby měli k dispozici aktuální verzi.

Z výše uvedeného vyplývá jeden z důležitých rysů aplikací pracujících v reálném čase – a to, že uživatel se může v každém okamžiku spolehnout na aktuálnost dat, která právě vidí. Tyto nástroje jsou tak velmi vhodné pro on-line spolupráci více lidí, e-learningové aktivity, sledování chování návštěvníků webu (např. internetového obchodu), rychlou organizaci větších skupin lidí apod.

2.5 Příklady webových aplikací pracujících v reálném čase

Jedním z největších soudobých trendů v IT je tzv. *cloud computing*. Definice tohoto pojmu je prozatím nepříliš ustálená¹³, ale bez újmy na přesnosti lze říci, že se jedná o službu, která na požádání umožňuje využívat výpočetních zdrojů prostřednictvím počítačové sítě [20].

Rozhodujícím faktorem, kvůli kterému firmy přecházejí na cloudové počítání, je zejména ekonomická stránka věci. Veškeré náklady související s vybudováním, provozem a údržbou infrastruktury totiž nese její poskytovatel namísto uživatele. Službou ale nemusí být pouze infrastruktura (Infrastructure as a Service, IaaS) nebo platforma (Platform as a Service, PaaS) k provozu aplikací, ale také aplikace sama (Software as a Service, dále jen SaaS).

Základní myšlenkou SaaS je, že zákazník dostane pouze tenkého klienta, který slouží k zobrazování dat a informací a k provádění elementárních operací s nimi [21]. Hlavní výpočetní jádro provádějící prakticky veškerou modifikaci dat je součástí samotné služby. Klient tak vytváří požadavky na službu, aby získal výsledky, a ty následně zobrazuje.

13. Tento fakt nejlépe ilustruje článek v on-line magazínu Cloud Computing Journal, kde 21 oslovených expertů poskytlo 21 různých odpovědí na otázku „Co je to cloud computing?“. Viz <http://cloudcomputing.sys-con.com/node/612375>

Vzhledem ke své architektuře, způsobu užití a faktu, že naprosto nejrozšířenějším tenkým klientem je webový prohlížeč, je poskytování softwaru jako služby odvětvím, kde naleznou nejširší uplatnění právě webové aplikace. A to obzvlášť ty, které alespoň z části pracují v reálném čase, což potvrzují ukázky aplikací, které v této kapitole následují.

2.5.1 Google Wave a Apache Wave

Google Wave vytvořená společností Google byla platforma, jež obsahovala patrně první webovou aplikaci, která masivně využívala zpracování událostí v reálném čase. Jejím hlavním cílem bylo umožnit rychlou on-line spolupráci mezi jednotlivci a týmy. V praxi tento nástroj fungoval tak, že sjednocoval e-mailovou komunikaci, instant messaging, wiki a sociální síť do jednoho sdíleného komunikačního prostoru¹⁴, čímž stíral rozdíly mezi preferencemi způsobů komunikace u jednotlivých členů týmu [22]. To umožňovalo efektivní kooperaci, neboť každý jednotlivec mohl využít svůj oblíbený způsob komunikace a zároveň existovalo jedno místo, kde bylo možné sledovat celou konverzaci včetně její historie [22].

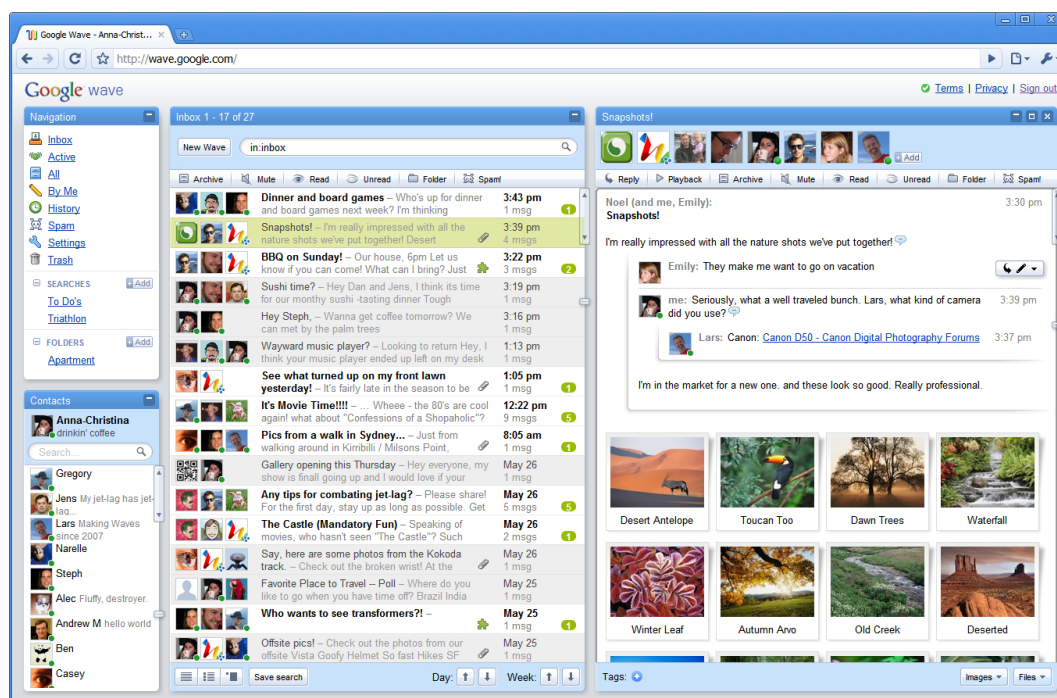
Google Wave kromě samotné webové aplikace obsahovala také vlastní protokol založený na XMPP¹⁵ a rozsáhlé aplikační rozhraní pro tvorbu vlastních aplikací a nástrojů. Nejednalo se tedy pouze o SaaS službu, ale také o poskytování kompletní aplikační platformy, tedy PaaS.

Google Wave jako komunikační nástroj bohužel nikdy nebyl přijat kritickou většinou uživatelů, a proto jej firma Google v červenci 2010 označila za neperspektivní a rozhodla se jeho další vývoj zastavit [23]. Utlumení provozu proběhlo v několika fázích. Nejprve byla zastavena možnost registrovat nové uživatele, následně byla aplikace dostupná pouze v režimu pro čtení a export dat a nakonec byla v dubnu 2012 definitivně ukončena.

Většina kódu tohoto nástroje byla později uvolněna jako open source a o jeho údržbu se starají dobrovolníci z Apache Software Foundation, která převzala správu projektu jako celku [23]. Repozitář obsahující zdrojový kód byl poté rozdělen na dvě části – první z nich je *Apache Wave*, což je samotný softwarový rámec pro vývoj aplikací a druhou je *Wave in a Box* (zkráceně WIAB). WIAB je samotný výsledný produkt, serverová implementace *Apache Wave*, obsahující webovou aplikaci.

14. Tzv. „vlny“, odtud název celé aplikace.

15. Extensible Messaging and Presence Protocol, protokol původně navržený pro síť Jabber, v současné době využívaný pro širokou paletu úkolů (vzájemná komunikace programů, ovládání automatických služeb apod.). Více viz kap. 5.1.2.



Obrázek 2.3: Obrazovka aplikace Google Wave s otevřenou „vlnou“ [24].

2.5.2 Google Apps

Google Apps¹⁶ je balík on-line aplikací od společnosti Google. Obsahuje nástroje pro práci s e-mailovou poštou (*Gmail*), kancelářské nástroje (*Dokumenty Google*), online úložiště dat (*Disk Google*), kalendář (*Kalendář Google*), chatovací program (*Google Talk*) a několik dalších, méně významných, nástrojů.

U služby *Google Talk* jsou okamžité odezvy nutnou podmínkou pro její kvalitní použitelnost, ale i u téměř každé z ostatních aplikací této sady nalezneme alespoň nějakou její část, která pracuje v reálném čase, a neprodleně tak zásobuje uživatele aktualizacemi se změnami jeho dat. Např. kancelářské nástroje ze sady *Dokumenty Google* umožňují souběžnou práci několika uživatelů na jednom otevřeném souboru. Ta funguje tak, že je pro každého aktivního uživatele zobrazen v editoru jeden (barevně odlišný) kurzor a veškeré změny, které provede, se okamžitě promítnou i do prohlížečů ostatních spolupracujících. Toto chování umožňuje velmi kvalitní kooperaci při přípravě dokumentů.

16. Google Apps jsou dostupné na adrese <http://www.google.com/enterprise/apps/business/> ve verzi pro firmy a na adrese <http://drive.google.com/> ve verzi pro běžné uživatele.

2.5.3 Office Web Apps

O sadě kancelářských nástrojů *Office Web Apps*¹⁷ lze říci, že jsou odpovědí společnosti Microsoft na úspěch *Google Apps*. Obsahuje webové ekvivalenty programů známých z balíku *Microsoft Office* pro operační systém Windows. Tímto krokem navíc firma Microsoft zvýšila počet svých potenciálních uživatelů, neboť zatímco *Office Web Apps* jsou dostupné pro všechny platformy, Microsoft Office je nabízen pouze pro operační systémy Windows a Mac OS X.

Stejně jako konkurenční sada, i *Office Web Apps* umožňují on-line spolupráci více uživatelů nad jedním dokumentem. Ačkoliv nejsou tak široce používány jako *Google Apps*, do přehledu významných aplikací pracujících v reálném čase je jistě nutné je uvést, a to nejen kvůli jejich funkcionalitě, ale také kvůli jejich historii.

Office Web Apps byly vytvořeny na základech jiné webové aplikace *Outlook Web Access*. A právě pro Outlook Web Access byl vývojáři společnosti Microsoft navržen a implementován prvotní koncept XMLHttpRequest (viz kapitola 2.3), tedy aplikační rozhraní, které umožnilo vznik technologie AJAX a potažmo i dalších webových aplikací.

2.5.4 Basecamp

*Basecamp*¹⁸ od společnosti 37signals je webová aplikace sloužící pro projektové řízení. Na rozdíl od výše zmíněných balíčků, které obsahují pro každou specifickou činnost jednu aplikaci, se jedná o velký integrovaný nástroj, jenž obsahuje mnoho komponent pracujících v reálném čase – např. textový editor pro kolaborativní tvorbu dokumentů, seznamy úkolů, interaktivní časová osa.

Basecamp také umožňuje propojení se službou *Campfire*, dalším produktem firmy 37signals. Campfire je webovou aplikací, která umožňuje on-line rozhovor. Na rozdíl od ostatních podobných programů si ale klade za cíl poskytnout efektivní komunikaci uvnitř vícečlenného týmu. Kromě běžných funkcí tak obsahuje i možnost moderovat probíhající diskusi, hlasování apod.

2.5.5 Sociální sítě

Tzv. *sociální sítě*, které přenáší společenské vazby mezi jednotlivými návštěvníky do prostředí Internetu, jsou beze sporu jedním ze současných trendů. Jsou to rozsáhlé webové aplikace umožňující uživatelům sestavovat okruh svých známých a přátel a vzájemně s nimi komunikovat (prostřednictvím přímých

17. <http://office.live.com>

18. <http://basecamp.com>

zpráv nebo tzv. „statusů“, které mají charakter mikroblování¹⁹), sdílet fotografie, odkazy a další obsah.

Aktuálně jsou nejpoužívanějšími sociálními sítěmi *Facebook*²⁰ a *Google+*²¹ [25]. Obě obsahují mnoho komponent, které uživateli v reálném čase doručují informace ze serveru, a to zejména oznámení o aktivitách jiných připojených uživatelů, upozornění na jejich příspěvky apod. Zároveň v obou těchto sociálních sítích nalezneme on-line chat.

Sociální síť *Facebook*, podobně jako původní služba *Google Wave* (viz kap. 2.5.1), obsahuje platformu pro tvorbu vlastních aplikací a používání prvků z *Facebooku* na webových stránkách třetích stran.

19. Mikroblov je, podobně jako blog, webovým zápisníkem. Na rozdíl od blogu jsou ale jednotlivé záznamy velmi krátké, typicky obsahují pouze několik desítek znaků. Více viz <http://en.wikipedia.org/wiki/Microblogging>.

20. <http://facebook.com>

21. <http://plus.google.com>

3 Techniky zavedení prvku reálného času

Ačkoliv princip fungování webových aplikací pracujících v reálném čase do jisté míry popírá způsob použití, který byl navržen pro HTTP protokol, existuje množství postupů, jakými ho implementovat. Tato kapitola si klade za cíl představit ty nejdůležitější z nich a ukázat, že s jejich pomocí lze řešit jednotlivé případy užití popsané v části 2.4.2.

Přestože většina těchto technik nějakým způsobem porušuje jeden ze dvou základních principů HTTP protokolu, není ve skutečnosti příliš nutné se zabývat otázkou, zdali je to správné nebo vhodné. Vedle zahájení komunikace klientem je dalším klíčovým rysem HTTP protokolu i to, že je striktně bezstavový (viz kap. 2.2) a nikdy nebyl navržen tak, aby uchovával nějaké informace o aktuálním sezení uživatele. I přesto se princip HTTP session¹ velice rychle prosadil do povědomí webových vývojářů a nikdo jej nepovažuje za zneužívání HTTP protokolu. Naopak se užití HTTP session stalo naprosto standardní součástí vývoje webových aplikací a umožnilo využít webové aplikace k obslužení mnohem širší škály případů užití [7].

Nutnost tvořit webové aplikace, které umožňují doručovat klientům změny v reálném čase, se poprvé ve větším měřítku objevila během tzv. „internetové horečky.“² Jednalo se zejména o webové nástroje ke komunikaci („chatovací místnosti“) a on-line hry [26]. Jejich implementace byla většinou založena na periodickém obnovování okna prohlížeče nebo využívala vlastních protokolů vytvořených prostřednictvím jiné technologie (např. Flash nebo Java) a ke svému běhu vyžadovaly rozšíření nainstalované do prohlížeče uživatele [26].

Jak už bylo řečeno v kapitole 2.3, AJAX je velmi jednoduchá technologie, která ale přinesla do světa webového vývoje velké změny a přepracování mnoha zažitých postupů [7]. Jedna z těchto změn se týká i zavedení prvku reálného času do webových aplikací. Po roce 2000 tak bylo poprvé možné vyvíjet takovéto aplikace jenom za použití standardních rozhraní poskytovaných prohlížečem a protokolem HTTP bez nutnosti spoléhat se na přítomnost proprietárních rozšíření [26].

1. HTTP session dává webovému serveru možnost uložit si informace o klientovi, který k němu přistupuje. V principu funguje tak, že údaje jsou uloženy na serveru pod nějakým unikátním identifikátorem, tento identifikátor je klientovi sdělen v odpovědi na požadavek a každý další požadavek tento identifikátor obsahuje.

2. Tzv. *Dot-com bubble*. Období rozkvětu a masivních investic do internetových firem a služeb v letech 1997 až 2001. Internetová horečka se týkala zejména USA a v menší míře i států západní Evropy. Většina firem vlivem poklesu trhu a ztráty zájmu investorů po roce 2001 zkrachovala, neboť nebyla schopna účinně tvořit zisk [27].

Aby webová aplikace mohla pracovat alespoň částečně v reálném čase, je nutné vyřešit problém vytvoření trvalého spojení mezi klientem a serverem. Právě s pomocí technologie AJAX můžeme vytvořit takové spojení, které sice persistentní v pravém slova smyslu slova není, ale svými vlastnostmi se mu podobá. Tímto způsobem využití technologie AJAX se zabývá první část této kapitoly.

Druhá polovina kapitoly poté představuje aktuální trend, protokol WebSocket vytvořený konsorciem W3C. WebSocket poskytuje *full-duplex* komunikaci³ prostřednictvím TCP protokolu na portu 80 a je součástí aplikačního rozhraní nejmodernějších prohlížečů. V současnosti je považován za budoucnost pro tvorbu webových aplikací s prvky reálného času [28].

3.1 Polling

Polling (někdy též označován jako „heartbeat“ [29]) je ve světě informačních technologií dobře známý pojem. Spočívá v opakovaném pokládání dotazu na zjištění stavu např. vstupně-výstupního zařízení a jeho aplikaci téměř s jistotou nalezneme mj. ve starších operačních systémech.

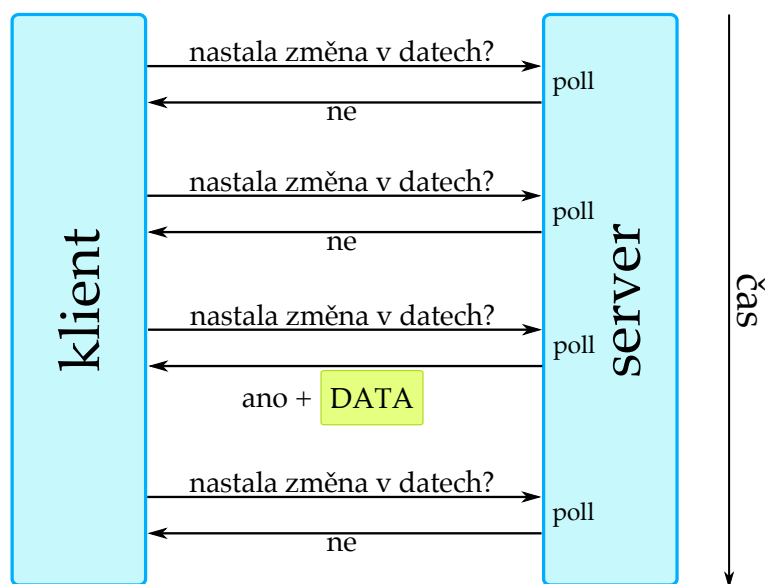
Využití pro tuto techniku v prostředí webových aplikací nalezneme na téměř libovolné kolaborativní aplikaci nebo hře popsané v kapitole 2.4.2. Představme si on-line implementaci známé hry „Šibenice.“⁴ Aplikace ze slovníku vybere slovo, které se budou hráči snažit po jednotlivých písmenech uhodnout. Po každém z tipů je nutné seznámit všechny hráče s výsledkem – tedy buď zapsat písmeno na správnou pozici ve slově, nebo dokreslit další část do obrázku oběšence. Každý tip představuje jeden požadavek na server, hádající hráč tedy obdrží výsledek svého pokusu v odpovědi na požadavek. Jádrem problému spočívá v tom, jak o výsledku informovat i další hráče.

Nejjednodušší způsob, jak tuto funkcionalitu implementovat, je využít právě pollingu, neboli cyklického dotazování serveru, zdali došlo k nějakým změnám v datech (viz obrázek 3.1) [7]. Jednotliví klienti tak kladou na server běžné požadavky a server na ně odpovídá, ve většině případů tak, že nedošlo k žádným změnám. Velkou nevýhodou pollingu je plýtvání zdroji – server je zatížen velkým množstvím zbytečných požadavků, které musí obsloužit, a stejně tak jsou po síti posílána nadbytečná data [29].

Aplikaci pro hraní „Šibenice“ kvůli zvýšení uživatelského prožitku navrhneme jako bohatou internetovou aplikaci, jednotlivé dotazy na stav proto budeme zasílat formou XMLHttpRequest požadavků (tj. pomocí technologie

3. Taktéž *double-duplex*; systém (resp. protokol) umožňující současnou komunikaci probíhající v obou směrech.

4. Anglicky známa jako *Hangman*. Pro podrobná pravidla viz [http://en.wikipedia.org/wiki/Hangman_\(game\)](http://en.wikipedia.org/wiki/Hangman_(game)).



Obrázek 3.1: Fungování pollingu ve webové aplikaci.

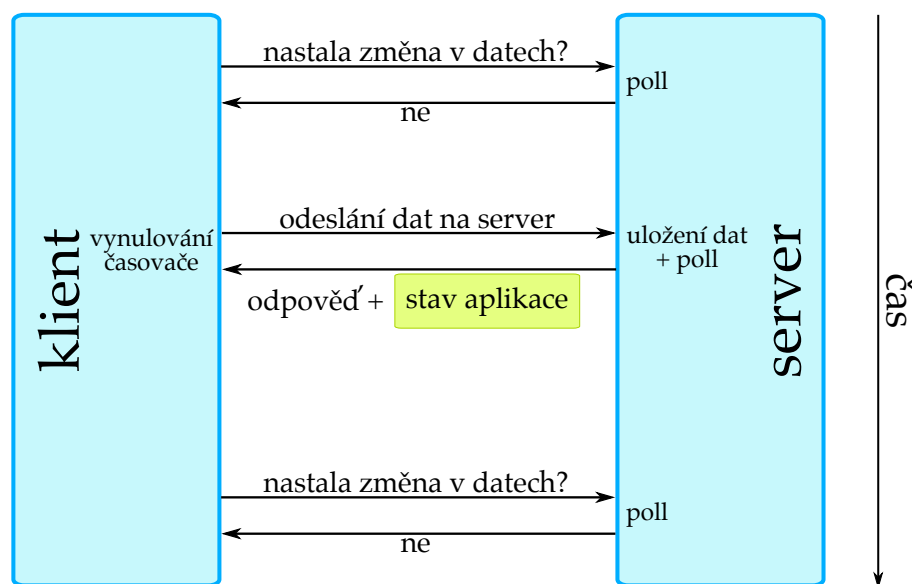
AJAX „na pozadí“, viz 2.3). Jazyk Javascript pak sám obsahuje vestavěnou funkci `setTimeout()`, která po zadaném počtu milisekund provede jinou funkci, která jí je předána jako argument. Implementace celého problému pak spočívá pouze v naprogramování dvou funkcí. První sestaví `XMLHttpRequest` požadavek a odešle jej na server a druhá pak zpracuje výsledek tohoto volání.

I přes přílišnou náročnost na množství použitých zdrojů může být polling dobrou volbou, a to zejména v případech, kdy očekáváme nízký počet připojených klientů. Pokud je interval nastaven na dostatečně krátkou dobu, jsou data v aplikaci aktuální a její zobrazení je konzistentní mezi jednotlivými klienty. Platí ale nepřímá úměra, že čím kratší je interval obnovení, tím větší je zátěž serveru a naopak. Každé řešení s využitím pollingu tedy vždy bude kompromisem mezi aktuálností dat v aplikaci a plýtváním zdroji [7].

3.1.1 Piggybacking

Aplikace pro hraní hry „Šibenice“ může být rozšířena o funkci, která hráčům umožní obohacovat slovník, z něhož jsou vybírána slova pro další kola hry. V aplikaci tak přibude textové pole, do kterého hráči i během probíhající hry mohou vepsat další slovo a to bude odesláno na server, kde bude také uloženo do databáze.

Stejně jako všechny ostatní HTTP požadavky v aplikaci bude i tento proveden prostřednictvím objektu `XMLHttpRequest`. Odpověď na něj tedy bude obsahovat XML strukturu, ve které bude patrně pouze hlášení o úspěšném



Obrázek 3.2: Piggybacking s vynulováním časovače pro snížení počtu HTTP požadavků [7].

nebo neúspěšném uložení slova do slovníku. Pokud do takovéto odpovědi serveru bude přidán i aktuální stav hry, umožní nám to optimalizovat počet pollingových volání.

K tomu v zásadě existují dvě strategie – pokud je interval pollingu dlouhý, můžeme tyto legitimní požadavky využít jako prostředek zrychlení reakce aplikace. Naopak, je-li interval krátký a generuje vysokou zátěž serveru, můžeme požadavek využít namísto pollingového dotazu, a to tím způsobem, že se při každém zpracování odpovědi na legitimní požadavek resetuje časovač pollingu (viz obr. 3.2) [7].

Tato technika bývá označována jako tzv. „piggybacking“, protože obsah odpovědi je do jisté míry nesouvisející s důvodem provedení původního HTTP požadavku [7] [30].

V popisované aplikaci bude vliv piggybackingu na celkový počet požadavků zanedbatelný, neboť obsahuje silnou disproporci mezi počtem pollingových a ostatních HTTP požadavků. V aplikacích, kde je tento poměr vyrovnaný nebo jsou dokonce pollingové HTTP požadavky v menšině, může tato technika velmi snížit celkový počet požadavků, které musí server odbavit.

3.2 Comet

Termín Comet poprvé představil Alex Russell v článku [31] v roce 2006. Zatímco polling (a jeho modifikace piggybacking) vychází z klasického modelu

HTTP komunikace, Comet již tuto myšlenku opouští. Předpokládá, že server data nezašle v odpovědi na požadavek, který o ně explicitně žádá, ale poskytne je sám, a to přesně v okamžiku jejich vytvoření nebo modifikace [31]. Sám Russell Comet doslovně definuje jako „*event-driven server-push data streaming*“ [31], což lze volně přeložit jako „událostmi řízené zaslání dat (*push*, viz kap. 2.4.2) ze serveru“.

Při implementaci tohoto způsobu komunikace tak odpadá zásadní problém pollingu, tedy hledání rovnovážného stavu mezi šetřením systémovými zdroji na jedné straně a aktuálností dat u klientů na straně druhé. Comet bez zvýšení zátěže serveru a dalších výkonnostních obtíží zkvalitňuje fungování aplikace, která má za úkol v reálném čase dodávat data k uživatelům [31].

Možnost, jak vytvořit takové spojení, je v mantinelech současných webových technologií prakticky jediná. Klient naváže se serverem spojení jako při standardním HTTP požadavku, server na něj odpoví, ale spojení neuzavře a ponechá ho nadále otevřené [29] [7]. Tímto způsobem vznikne komunikační kanál, prostřednictvím kterého může server klientovi zasílat své zprávy.

Toto, na první pohled poněkud zvláštní, využití HTTP protokolu má jednu zásadní výhodu – využívá naprosto standardních prostředků, a proto je dostupné ve všech webových prohlížečích bez nutnosti jakýchkoliv doplňků nebo dalšího softwaru [29].

Nejjednodušší metodou, jak podobné chování implementovat, je vytvořit ve stránce neviditelnou HTML značku `<iframe>`. Atribut `src` této značky obsahuje takovou adresu, na které nedojde po odbavení HTTP požadavku k uzavření spojení. Tím server získá komunikační kanál, kterým může klientovi zasílat data. A to jednoduše tak, že dojde-li k jejich změně, zašle tuto změnu jako javascriptový kód v HTML značce `<script>`. Prohlížeče zpracovávají příchozí data okamžitě, nečekají na uzavření spojení, tento kód tedy bude ihned vykonán.

Tato triviální technika je v literatuře označována jako tzv. „streaming“. Její výhodou je zejména popsáná snadnost implementace, která na straně prohlížeče nevyžaduje téměř žádné klientské skriptování. Veškerý kód je vygenerován na straně serveru a v prohlížeči je pouze vykonán.

Nevýhodou streamingu je přímo jeho hlavní rys, tedy samotná délka jednotlivých HTTP spojení. První problém by mohl nastat na straně klienta, neboť ve většině aktuálně používaných prohlížečů je maximální počet současných požadavků na jednu doménu omezen na dva [7]. Jeden dlouhotrvající požadavek tak vyčerpá polovinu této kvóty. V případě, že i druhý požadavek na stejnou doménu trvá delší dobu (např. může čekat na výsledek složitého databázového dotazu nebo na nějakou jinou časově náročnou operaci na serveru), dojde k situaci, že aplikace dočasně přestane reagovat na další uživa-

telské akce (požadavky se budou řadit do fronty a čekat, až některý z běžících skončí). S tímto faktem souvisí i druhá část problému – prohlížeče se snaží spojení, po kterých dlouhou nepřišla žádná data, samy ukončit právě z důvodu, aby neblokovala další požadavky [7].

Druhý problém pak nastává na straně serveru, resp. po cestě od klienta k serveru. Překážkami v komunikaci jsou v tomto případě některé síťové prvky, zejména se jedná o různé firewally a proxy servery. Ty v rámci šetření vlastních systémových zdrojů mohou tato velmi dlouhá a z jejich pohledu nevyužitá HTTP spojení (typicky takovýmto spojením během desítek minut projde pouze zanedbatelné množství paketů) automaticky uzavírat [29].

3.2.1 Long polling

Řešením obou výše zmíněných obtíží je technika zvaná „long polling“. Ta, jak název napovídá, v sobě kombinuje polling popsáný v kapitole 3.1 a dlouhotrvající HTTP požadavky. Pojem „dlouhotrvající“ v tomto případě znamená obvykle několik desítek sekund.

Prakticky long polling funguje tak, že klient vytvoří HTTP požadavek typu XMLHttpRequest, server potvrdí navázání spojení, ale neodešle žádnou odpověď, takže spojení zůstane otevřené. Jakmile jsou na serveru k dispozici data, server je odešle klientovi a spojení uzavře. Klient data zpracuje a vytvoří další dlouhotrvající HTTP spojení. Tímto způsobem spolu klient a server komunikují po celou dobu běhu aplikace.

Druhou možností je, že mají jednotlivá HTTP spojení přesně stanovenou maximální délku. Takové spojení je pak ukončeno a nahrazeno novým spojením i v případě, že server nedodal žádná data.

3.2.2 Specializované Comet servery

Další nevýhoda tohoto postupu je také zřejmá – server musí udržovat všechna takto vzniklá spojení neustále otevřená a v aktivním stavu, čímž blokuje své volné prostředky.

Na rozdíl od pollingu sice server není vystaven přílivu velkého množství požadavků, a tedy vysoké zátěži CPU, každé otevřené spojení ale znamená alokaci určitého množství operační paměti. Zároveň má také každý webový server limit na počet uživatelů, které dokáže souběžně obsloužit⁵ [7]. V praxi by tak mohlo dojít k situaci, kdy všechnu tuto kapacitu obsadí Comet požadavky a server nebude schopen odpovídat na žádné další.

5. Např. v hojně rozšířeném webovém serveru Apache se tato konfigurační direktiva nazývá MaxClients.

Další potíží pak je, že obsluha a zpracování dlouhotrvajících HTTP požadavků je v rozporu se základním rysem všech webových serverů; tedy rychle obsloužit klienta a uvolnit zdroje pro další spojení [29].

Pro implementaci Comet serveru se tak používají speciální webové servery, např. *Ajax Push Engine*⁶ nebo rozšiřující moduly pro ty existující jako je např. *HTTP Push Module*⁷ pro NGiNX⁸.

3.2.3 Protokol Bayeux

Protokol Bayeux je de-facto standardem pro tvorbu Comet aplikací [7]. Byl vytvořen Alexem Russellem a dalšími vývojáři sdruženými okolo javascriptového rámce *dojo*⁹ v roce 2007 [32].

Jeho primárním cílem je zjednodušit a do jisté míry standardizovat tvorbu aplikací, které fungují v reálném čase. Za tímto účelem specifikuje metody komunikace mezi klientem a serverem, serverem a klientem a také mezi dvěma klienty (skrze server). Veškerá komunikace probíhá prostřednictvím asynchronních zpráv a to zejména za použití běžných HTTP prostředků a technologie AJAX [32].

Existuje i referenční implementace tohoto protokolu s názvem *cometD*. Sami autoři tento software označují jako tzv. Comet sběrnici [33]. Nejedná se tedy o webový server jako takový, spíše o jistou mezivrstvu mezi ním a klientem, která se stará o udržování spojení mezi serverem a klientem a doručování zpráv ze serveru na klienta.

3.3 Protokol a aplikační rozhraní WebSocket

Vzhledem k Russellově původní definici z [31] bychom i technologii WebSocket mohli označit jako další možnost, jak implementovat Comet. Vzhledem k její důležitosti pro praktickou část této práce a také pro její naprostou odlišnost od všech výše zmíněných postupů jí je ale věnována speciální kapitola.

Jak už bylo zmíněno v předcházejících odstavcích, WebSocket je webová technologie, která umožňuje obousměrnou komunikaci mezi prohlížečem a serverem prostřednictvím jediného TCP spojení [34]. Protokol WebSocket byl standardizován v roce 2011 organizací Internet Engineering Task Force [35] a aplikační rozhraní tohoto protokolu bylo standardizováno konsorciem W3C taktéž v roce 2011 [36].

6. <http://www.ape-project.org/home.html>

7. <http://pushmodule.slact.net/>

8. <http://nginx.org/>

9. <http://dojotoolkit.org/>

Se síťovými sokety se zejména při tvorbě síťových aplikací setkáváme ve valné většině programovacích jazyků. Jedná se o rozhraní (obvykle poskytované přímo operačním systémem), které slouží ke spojení mezi dvěma komunikujícími stranami prostřednictvím technologie IP. Velmi zjednodušeně řečeno je soket koncovým bodem síťové komunikace a jeho adresa je popsána kombinací IP adresy, protokolu a portu, který je pro danou komunikaci použit [34]. Soket sám je pouze síťovým rozhraním na velmi nízké úrovni, aplikace a služby na vyšší úrovni jej dále používají k přenášení dat.

Protokol WebSocket (často zmiňován pouze zkratkou WS) je pak přenesením tohoto konceptu do světa webového vývoje. Mezi prohlížečem a serverem se pouze za použití standardních prostředků, nikoliv pomocí prohlížečových rozšíření od třetích stran nebo jiného softwaru, vytvoří obousměrné spojení, po kterém mohou navzájem komunikovat. Podle specifikace je možné, použít i tzv. WSS¹⁰, tedy šifrované spojení vhodné i k přenosu citlivých dat. Velice elegantně se tak vyhneme oběma výše popsaným možnostem – pollingu a různým implementacím metody Comet, což přináší, zejména pro vývojáře, jasnou přidanou hodnotu.

3.3.1 Firewally a proxy servery

Pro aplikace založené na dlouhotrvajících HTTP požadavcích byly často velmi velkou překážkou firewally, proxy servery atp. (kapitola 3.2). Technologie WebSocket tímto neduhem netrpí – během navazování spojení mezi klientem a serverem dochází k automatické detekci proxy serverů. Pokud je nějaký takový server nalezen, protokol WebSocket samočinně vytvoří tunel pomocí HTTP příkazu CONNECT, který otevře TCP/IP spojení s daným serverem na definovaném portu [34]. Jakmile je tunel vytvořen, informace mohou být bez překážek obousměrně přenášeny. Analogicky je vytvářeno i zabezpečené spojení protokolem WSS.

3.3.2 Podpora prohlížečů

Jelikož technologie WebSocket pro svou funkčnost potřebuje rozhraní na straně webového prohlížeče, je její rozšířenost a využitelnost přímo závislá na vůli výrobců ji do svých programů implementovat.

V době psaní této práce už není WebSocket pouze teoretickým konceptem, ale již je možné jej použít i v praxi. Všechny majoritní prohlížeče pro osobní počítače (viz tab. 3.1) ve svých posledních verzích obsahují aplikační rozhraní protokolu WebSocket.

10. WebSocket over SSL – zabezpečená varianta protokolu WebSocket.

Prohlížeč	Podpora od verze	Podíl na trhu
Firefox	11	35,98 %
Chrome	16	15,95 %
Safari	6	2,95 %
MS Internet Explorer	10	2,93 %
Opera	12.10	1,48 %

Tabulka 3.1: Dostupnost protokolu WebSocket v prohlížečích.

Tabulka 3.1 ukazuje tržní podíl jednotlivých prohlížečů, které implementují standard WebSocket. Data pochází z nezávislého měření společnosti Net Applications [37] a ukazují, že protokol WebSocket je dostupný na 59,29 % klientů. Pro aplikaci, která by měla být plně funkční pro naprostou většinu potenciálních uživatelů, se ale jedná o příliš malou rozšířenost.

Pro vytvoření takové aplikace však lze použít např. softwarový rámec *socket.io* určený právě pro tvorbu aplikací pracujících v reálném čase. Obsahuje v sobě API, které slouží jako abstrakce nad všemi v této kapitole popsanými technikami a podle toho, v jakém prohlížeči je aplikace používána, vybere nejvhodnější z nich. Podrobnější popis tohoto nástroje následuje v kapitole 5.1.4.

4 Návrh řešení

První část kapitoly je věnována popisu programu Aidbot, který je vytvářen jako součást této diplomové práce. Zmiňuje motivaci pro vytvoření podobného systému a definuje jednotlivé případy jejího užití. Na jejich základě je následně důkladně popisuje stěžejní požadavky na vznikající aplikaci, a to zejména z hlediska jednotlivých funkcí programu.

Se sběrem a následným vyhodnocením požadavků pak úzce souvisí i návrh odpovídající architektury aplikace, která bude dostatečně robustní, a umožní tak budoucí snadnou rozšiřitelnost programu.

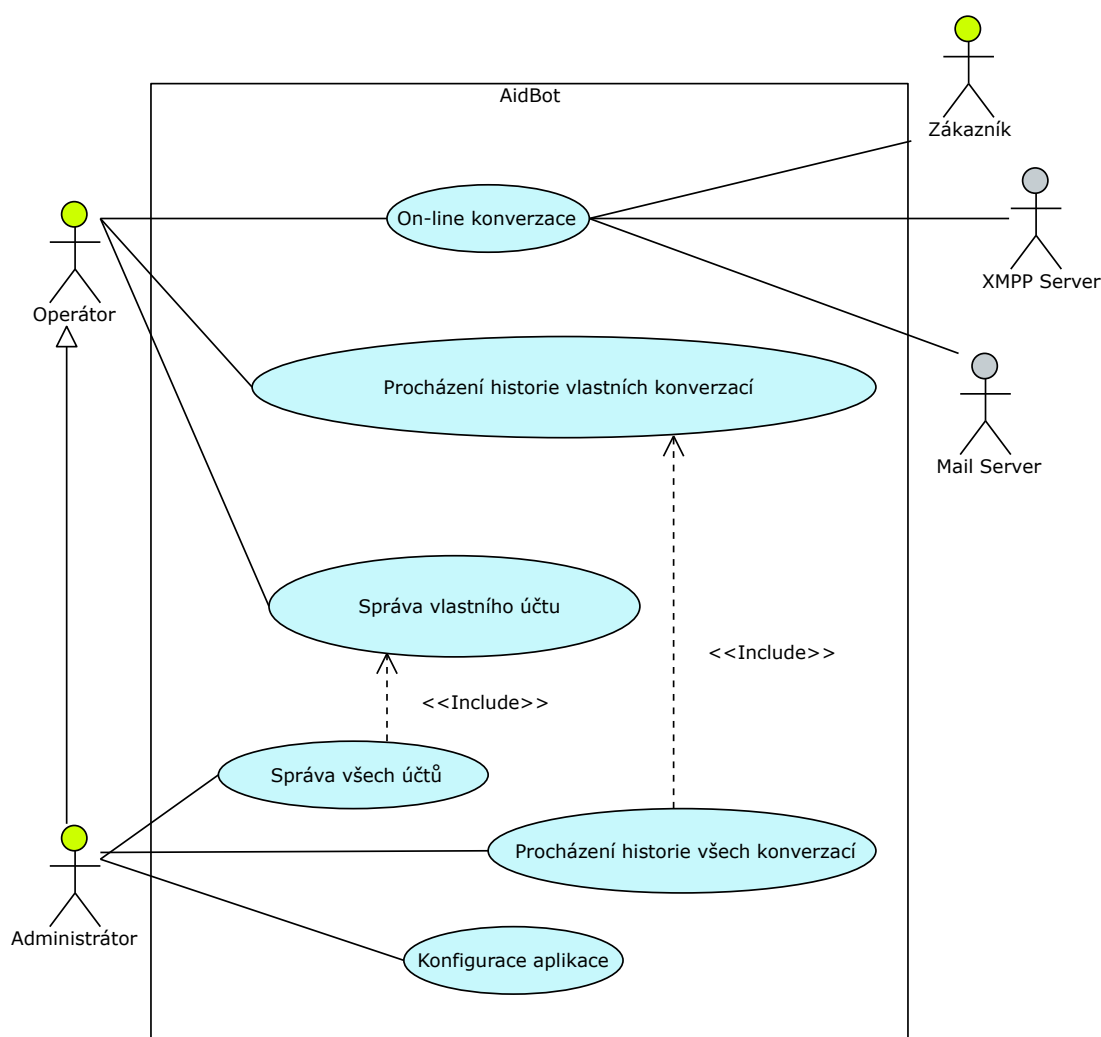
4.1 Základní případy užití

Systém Aidbot by měl být softwarem, který umožní efektivně podporovat zákazníky resp. uživatele jiných webových aplikací. Základní a nejdůležitější případ užití můžeme demonstrovat na příkladu nákupu v internetovém obchodu. Zákazník, procházející produkty v tomto obchodě, se může dostat do situace, kdy narazí na zboží, o jehož koupi uvažuje, ale nenalezl o něm na stránkách prodejce dostatečné množství informací. Jednou z možností, jak tuto situaci řešit, je kontaktovat zákaznickou podporu, která bývá běžně dostupná prostřednictvím infolinky nebo k tomu určené e-mailové adresy. Oba tyto způsoby s sebou přinášejí určité obtíže, ať už se jedná o poplatek za telefonní hovor nebo nejistou dobu odpovědi danou asynchronní povahou služby elektronické pošty.

Aidbot se snaží tuto nesnáz řešit. Pokud je do internetového obchodu integrován, umožní zákazníkovi klepnutím na příslušné tlačítko otevřít přímo na dané stránce (tedy bez toho, aby byl zákazník přesměrován někam jinam) komunikační okno, prostřednictvím kterého může přímo v prohlížeči zahájit konverzaci s operátorem zákaznické podpory. Tato konverzace bude probíhat v reálném čase. Uživatel tak může beze změny kontextu získat potřebné informace. Navíc hned při zahájení konverzace je operátorovi doručena adresa, na které se zákazník aktuálně nachází. Tímto způsobem je k operátorovi alespoň částečně přenesen zákazníkův aktuální kontext, což zvyšuje jeho schopnost poskytnout mu relevantní odpovědi na jeho otázky.

4.1.1 Další případy užití

Ambicí systému Aidbot je řešit podporu uživatelů komplexně. Komunikační okno pro konverzaci mezi zákazníkem a operátorem je tak pouze jednou z jeho částí. Další případy užití souvisí s administračním rozhraním celého systému, jež umožňuje přihlášení ve dvou různých uživatelských rolích.



Obrázek 4.1: Diagram případů užití.

Uživatel s rolí „operátor“ může v administračním rozhraní procházet konverzace se zákazníky, kterých se sám zúčastnil, a spravovat nastavení svého účtu. Odpovědný pracovník s rolí „administrátor“ může procházet historii všech již ukončených konverzací. Jejich důsledné vyhodnocování může vést zejména ke zkvalitnění uživatelské podpory. V případě nasazení Aidbotu do internetového obchodu (nebo jiné transakční webové aplikace) může vést také k lepšímu poznání nákupního chování zákazníků, a tím pádem mít i ekonomické cíle.

Posledním, neméně důležitým případem užití, je správa účtů jednotlivých operátorů. Jakmile administrátor takový účet vytvoří, dojde k jeho založení i v XMPP serveru. Operátor tak používá jedno heslo pro připojení k Jabberu i pro přihlášení do administračního rozhraní.

4.1.2 Motivace

Služby podobného druhu již samozřejmě existují¹ a vesměs jsou nabízeny jako SaaS. Základní motivací pro tvorbu aplikace Aidbot tak je poskytnout takovou alternativu těmto systémům, která bude mít otevřený zdrojový kód a bude dostupná pod některou ze svobodných licencí, konkrétně MIT licenci².

Ačkoliv některé z podobných služeb umožňují, aby ze strany operátora komunikace probíhala prostřednictvím Jabberu (resp. XMPP, viz kap. 5.1.2) či jiného programu pro instant messaging, žádné z těchto řešení tento způsob nepovažuje za výchozí, a jako rozhraní pro operátora využívají speciální webovou aplikaci. Druhým motivačním faktorem tedy je vytvořit takový systém, pro který bude protokol XMPP primárním komunikačním kanálem. Jeho použití přináší několik zajímavých výhod – za nejdůležitější z nich můžeme označit, fakt že nevyžaduje stálou přítomnost operátora v aplikaci, aby mohl okamžitě reagovat na příchozí zprávy; dalšími výhodami jsou pak jeho rozšiřitelnost a možnost používání různých klientů podle preferencí daného operátora.

4.2 Definice požadavků

Pro fungování systému jsou klíčové zejména dvě skupiny uživatelů. Tou první jsou uživatelé nějaké aplikace, kteří potřebují určitou formu technické podpory. Druhou skupinou jsou operátoři, kteří technickou podporu poskytují. V systému zároveň existuje i třetí skupina uživatelů – administrátoři. Administrátor je operátorem, který má rozšířené pravomoci a prostřednictvím administračního rozhraní může kontrolovat konfiguraci celého systému.

4.2.1 Funkční požadavky

- **On-line konverzace v reálném čase**
Základním případem užití je konverzace mezi zákazníkem a operátorem. Ze strany zákazníka probíhá prostřednictvím webového prohlížeče a je nutné, aby její zpoždění způsobené technickými vlivy bylo zanedbatelné, tj. menší než jedna sekunda.
- **Podpora pro IM software**
Jak bylo zmíněno v předchozí kapitole (4.1.2), existující programy řešící stejný problém vesměs pro komunikaci ze strany operátora, používají webovou aplikaci. To se může stát problematickým zejména v malých firmách, kde zaměstnanec pověřený podporou zákazníků má i další

1. Např. Zopim (<http://zopim.com>), LiveZilla (<http://livezilla.net>) nebo Livechatoo (<http://livechatoo.com>).

2. Licence vytvořená na Massachusetts Institute of Technology, viz http://en.wikipedia.org/wiki/MIT_License.

pracovní povinnosti (v případě internetového obchodu se může jednat např. o pracovníka obchodního oddělení). IM program v tomto případě poskytuje ideální řešení. Může být v počítači spuštěn na pozadí a na příchozí zprávy operátora upozorňovat. Ten tak není nucen neustále sledovat webové rozhraní a může se věnovat i ostatním činnostem.

- **Historie konverzací**

System nabídne prostřednictvím administračního rozhraní uživatelům s patřičným oprávněním možnost procházet již ukončené konverzace. Tato funkcionality je nutná zejména pro řídicí pracovníky a umožňuje jim vyhodnocovat efektivitu podpory a určovat další komunikační strategii pro operátory.

- **Uživatelské role**

System bude umožňovat členění operátorů do nejméně dvou úrovní. Podle těchto úrovní jim bude zpřístupněna příslušná funkcionality.

4.2.2 Ostatní požadavky

- **Snadná integrace**

Vzhledem k faktu, že bude program používán společně s velkým množstvím heterogenních systémů (internetové obchody, firemní intranetové systémy, rezervační systémy. . .) je jedním ze zásadních požadavků jeho jednoduchá integrace. Za ideální lze považovat takový stav, kdy bude začlenění do cizího systému probíhat pouze vložení hypertextového odkazu na komunikační okno do HTML šablony hostitelského systému.

- **Multiplatformnost, nezávislost**

Je pravděpodobné, že uživatelé budou pro komunikaci s operátory využívat velké množství rozdílných zařízení a prohlížečů. Je proto nutné, aby komunikační okno programu (a s ním související komunikace v reálném čase) nebylo závislé na žádné nestandardní vlastnosti některé z těchto platforem a uspokojivě fungovalo na každé z nich.

Stejně tak je nutné, aby kód programu nespolehal na přítomnost některého z proprietárních rozšíření prohlížeče. Pro implementaci je nutné vystačit pouze se standardizovanými prostředky.

- **Použitelnost, srozumitelnost rozhraní**

Komunikační okno bude používáno zejména laickými uživateli, je tedy vhodné, aby jeho ovládání bylo srozumitelné a intuitivní a bylo použitelné s maximálním možným uživatelským komfortem. Jelikož neustále roste počet mobilních zařízení mezi běžnými uživateli, je také důležité, aby bylo lehce ovladatelné i na přístrojích s dotykovou obrazovkou.

Požadavek na intuitivní a snadno použitelné uživatelské rozhraní lze vztáhnout i na administraci celého systému. Zde však lze předpokládat, že uživatel bude dostatečně proškolen, tudíž může být ovládání navrženo spíše s ohledem na efektivitu práce než na jeho jednoduchost.

- **Robustnost, snadná rozšiřitelnost a údržba**

Samozřejmým požadavkem je také schopnost aplikace vyrovnat se s neočekávanými stavy a chybami ve vnějším prostředí. Té lze dosáhnout zejména kvalitní architekturou a využitím stabilních a osvědčených technologií.

Stejně tak je nutné předpokládat, že se jednotlivé případy užití mohou v budoucnosti měnit, a je tedy vhodné počítat s budoucími úpravami. Aplikace musí být alespoň do určité míry připravena na změny, čehož lze dosáhnout zejména aplikací vhodných návrhových vzorů.

- **Dokumentace**

Vzhledem k tomu, že Aidbot bude uvolněn jako software s otevřeným zdrojovým kódem a lze tedy předpokládat, že ostatní uživatelé jej budou různě modifikovat, je také vhodné, aby byl precizně zdokumentován. Zkušenosti z ostatních open-source projektů ale ukazují, že kvalitní dokumentace je během na dlouhou trať.

4.3 Architektura systému

Celý systém Aidbot bude rozdělen do dvou nezávislých komponent. Serverové části, jejíž úkolem je především ukládat data a zprostředkovávat propojení s XMPP serverem, a dvou klientských aplikací.

Obě klientské aplikace, tj. **komunikační okno** (které slouží ke konverzaci zákazníka s operátorem a bude integrováno do ostatních webových systémů, pro něž je potřeba technická podpora) a **administrační rozhraní** (sloužící pro správu operátorských účtů a proběhlých konverzací) budou implementovat architektonický vzor MVC (Model View Controller) v jazyce JavaScript přímo na klientovi (viz kap. 5.1.5). Při prvním HTTP požadavku dojde ke stažení celé aplikace do prohlížeče, kde dále poběží.

Vzor MVC je v mnoha moderních webových aplikacích velmi rozšířeným. Tato architektura rozděluje aplikaci do tří skupin zodpovědnosti. Model je reprezentací dat a vnitřních stavů, s nimiž aplikace pracuje, view se stará o zobrazení informací uživateli a controller zachytává události (typicky pocházející od uživatele či jiného systému) a v reakci na ně mění stav modelu nebo view.

Z toho důvodu, že se celá implementace aplikace nachází na straně klienta, nebude modelová vrstva propojena přímo s databází, ale bude prostřednic-

tvím HTTP požadavků komunikovat s k tomu určenou webovou službou (typu RESTful, viz kap. 5.1.1), která bude primárním zdrojem dat.

Použití architektury MVC s sebou nese značnou výhodu pro budoucí změny a celkovou udržitelnost kódu. Každá ze součástí implementuje definované rozhraní, což s sebou přináší nezávislost jednotlivých komponent, která umožňuje jednoduchou změnu nebo výměnu každé z nich bez dopadu na ostatní.

Hlavním argumentem pro tuto klientskou implementaci MVC a její napojení na webovou službu je zejména budoucí rozšiřitelnost programu. Jednou z uvažovaných možností je i vytvoření speciálního softwaru pro operátory. Jednalo by se o nativní aplikaci pro operační systém Microsoft Windows, která by fungovala jako XMPP klient pro konverzaci samotnou, a zároveň byl obsahovala i veškerou konverzační historii přihlášeného operátora tak, aby ji mohl procházet a dále s ní pracovat (např. za účelem správy kontaktů na zákazníky apod.). Pokud takováto aplikace vznikne, nebude nutné celý systém jakkoliv upravovat, aplikace se pouze připojí k webové službě, která pro ni bude zdrojem dat.

4.3.1 Serverová část

S přihlédnutím ke komplexnosti celé serverové části můžeme její jádro rozdělit do tří základních subsystémů.

1. **Webový server a RESTful webová služba** – Úkolem webového serveru je zpracovávat požadavky z obou v systému obsažených bohatých internetových aplikací. Jedním druhem odpovědí na tyto požadavky budou HTML stránky, resp. jejich části, neboť obě webové aplikace, jsou tzv. „jednostránkové“. Jednostránková webová aplikace je taková, jejíž veškerý kód se načte s prvním požadavkem [38] a další změny v ní probíhají pomocí AJAXu.

Druhým typem odpovědí budou samotné reakce webové služby, kterou jádro systému navenek poskytuje. Tato webová služba obsahuje tzv. RESTful rozhraní³. Jejím prostřednictvím komunikují s jádrem systému webové aplikace běžící na klientovi. Webová služba umožňuje např. procházet databázi zpráv a konverzací.

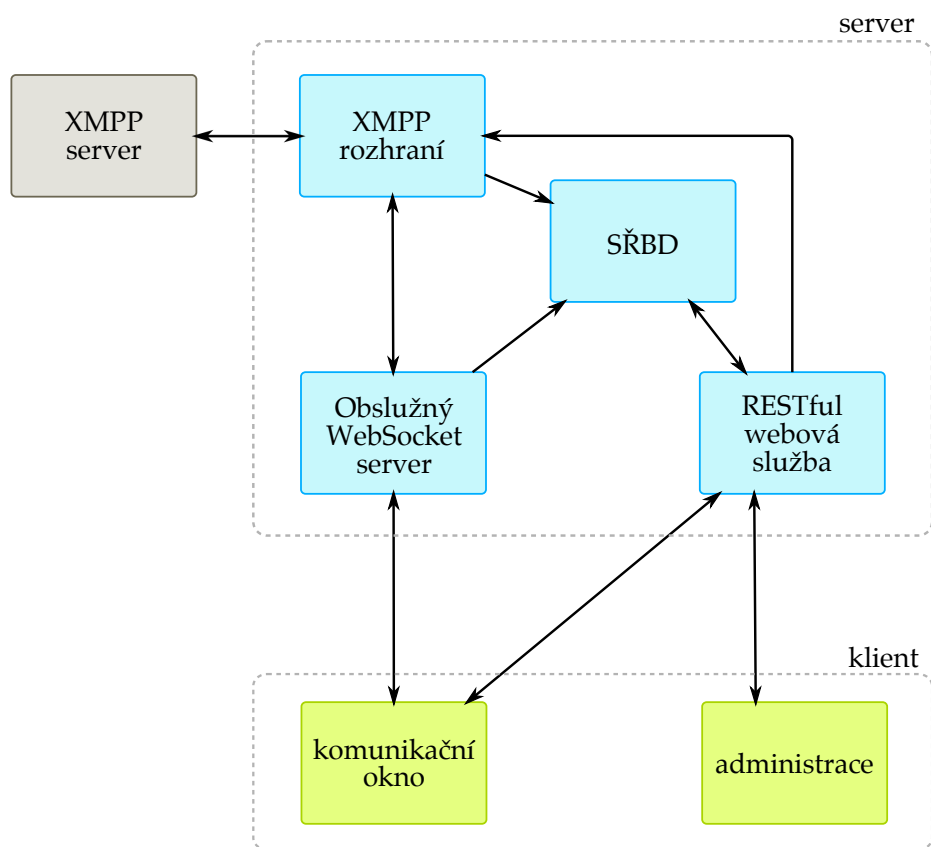
2. **Obslužný server pro WebSocket** – Aplikace komunikačního okna bude fungovat v reálném čase. S prvním požadavkem z komunikačního okna dojde k vytvoření webového soketu, prostřednictvím kterého je realizo-

3. Aplikační rozhraní implementované v souladu s principy RESTu na protokolu HTTP. Více viz kap 5.1.1.

vána veškerá komunikace v reálném čase mezi uživatelem a operátorem. Obslužný server se stará o správu těchto spojení.

3. **XMPP rozhraní** – Částečná implementace XMPP protokolu uvnitř Aidbotu, která zajišťuje napojení na XMPP server. Ze zpracované zprávy od zákazníka vytvoří validní XMPP zprávu a tu zašle příslušnému serveru. Stejně tak převezme odpověď od operátora a předá ji k dalšímu zpracování a odeslání zákazníkovi.

Hlavním úkolem serverové části systému je tedy převzetí zprávy od zákazníka, její zpracování (zejm. uložení do databáze) a předání XMPP serveru. Jakmile operátor na zprávu zareaguje, převezme tuto odpověď od XMPP serveru a předá ji dále na klienta.



Obrázek 4.2: Schéma architektury systému Aidbot.

5 Implementace

Následující kapitola se věnuje především popisu samotné implementace programu Aidbot, ukazuje její zajímavá místa a popisuje klíčová rozhodnutí, která během vývoje musela být udělána. Zároveň přibližuje i problémy, které nastaly a bylo nutné řešit. První část kapitoly se také zabývá představením jednotlivých technologií, které jsou v systému použity.

5.1 Použité technologie

Jelikož je velká část technologií použitých pro implementaci programu Aidbot poměrně mladá (s výjimkou REST a XMPP, které jsou ovšem v kontextu implementovaného systému poměrně zásadní), je jejich představení věnována tato podkapitola. Zároveň je u každé z představovaných technologií diskutováno, proč byla zvolena a zdali je její použití vhodné.

5.1.1 REST a RESTful webové služby

Representational State Transfer byl vyvinut souběžně s návrhem specifikace HTTP protokolu verze 1.1. Termín samotný byl představen v roce 2000 v disertační práci Roye Fieldinga, jednoho ze spoluautorů této specifikace. REST je komplexní způsob návrhu architektury distribuovaných systémů [39]. Největším implementovaným systémem, o kterém můžeme říci, že splňuje požadavky REST architektury, je World Wide Web.

Klíčovou abstrakcí v RESTu je tzv. „zdroj“ (angl. resource). Zdrojem může být téměř libovolný objekt, ve většině případů se ale jedná buď o data nebo o stav aplikace. Platí však, že každý zdroj má unikátní identifikátor [40]. Aby mohly jednotlivé části systému se zdroji manipulovat, musí být schopny spolu navzájem komunikovat prostřednictvím nějakého standardizovaného rozhraní, a vyměňovat si tak reprezentace jednotlivých zdrojů (tedy zprávy obsahující informace o zdroji v nějakém konkrétním formátu, např. JSON nebo XML).

Jakákoliv z částí systému může provést požadavek na manipulaci se zdrojem bez toho, aby znala jakékoliv informace o cestě, která ke zdroji vede. Nemusí znát umístění serveru, na kterém se zdroj nachází, veškeré potenciální překážky (např. proxy servery, firewally atp.) jsou považovány za součást systému. Jediné, co k úspěšnému provedení požadavku potřebuje, je identifikátor zdroje a akce, kterou chce nad zdrojem vykonat [4].

Zdroj	Kolekce dat	Jedna datová položka
HTTP metoda	URI: /resources/	URI: /resources/item8
GET	Seznam URI identifikátorů jednotlivých položek.	Konkrétní položka v definovaném formátu.
PUT	Nahrazení celé kolekce jinou kolekcí.	Editace konkrétní položky. Pokud neexistuje, bude vytvořena.
POST	Vytvoření nové položky v kolekci a vrácení její URI.	<i>Obvykle nepoužíváno.</i>
DELETE	Odstranění celé kolekce.	Odstranění konkrétní položky.

Tabulka 5.1: Data vrácená po provedení jednotlivých akcí na zdroje.

Implementaci principů REST v prostředí webu a protokolu HTTP označujeme jako tzv. **RESTful webovou službu**¹. Identifikátorem zdroje je pak jeho URI² a komunikačním rozhraním mezi částmi systému je protokol HTTP. Jednotlivými akcemi, které je možno nad zdroji dat provádět, jsou metody samotného HTTP protokolu, tedy GET, POST, PUT a DELETE [4]. Příklad jejich užití pro specifické operace ukazuje tabulka 5.1.

Je zřejmé, že mezi klasickými a RESTful webovými službami (viz kap. 2.1) existují významné odlišnosti. Prvně, běžné webové služby definují vzdálené procedury a protokol, kterým je možné je volat, naopak REST určuje, kde se data nachází a jak k nim přistupovat. Můžeme tedy říci, že se jedná o datové, nikoliv procedurálně orientovaný přístup [4].

Druhý rozdíl pak pramení z odlišné povahy obou přístupů. Zatímco tradiční webové služby jsou standardizovanými protokoly (např. už zmíněný SOAP) a jsou formálně specifikované, RESTful webové služby, ač používají standardizované součásti (HTTP, URI, XML aj.), jako celek standardizovány nejsou. Neexistují tedy žádná závazná pravidla, jak tyto služby tvořit, a implementace závisí zejména na obecných konvencích.

Vzhledem k zamýšlené architektuře celého systému, se RESTful webová služba jako základní aplikační rozhraní jeví být ideální volbou. Její používání

1. Také RESTful Web API, popř. RESTful aplikační rozhraní.

2. Uniform Resource Identifier, viz např. http://en.wikipedia.org/wiki/Uniform_resource_identifier/.

se nijak zásadně neliší od běžných HTTP požadavků, jedná se pouze o jejich zobecnění. To snižuje nároky na klienta prakticky pouze na schopnost pracovat s HTTP protokolem, což poskytuje velice jednoduchou možnost, jak k rozhraní připojovat další, navzájem heterogenní aplikace.

5.1.2 XMPP

XMPP (Extensible Messaging and Presence Protocol) je komunikační protokol založený na XML [41]. Původně byl navržen pro použití v komunikační síti *Jabber* a v prvních letech své existence byl znám pod shodným názvem [41].

Na rozdíl od většiny protokolů pro instant messaging je XMPP utvářen jako otevřený standard. Tato otevřenost umožňuje, aby byl snadno implementován i do softwarových systémů, které nejsou primárně určeny pro instant messaging a mohou jej dále využívat k zasílání zpráv pro svou vzájemnou komunikaci [41]. To je společně s jeho rozšiřitelností hlavním důvodem, proč byl tento protokol (a potažmo síť *Jabber*) zvolen jako součást řešení praktické části této práce.

Síť *Jabber* je od ostatních komunikačních sítí³ odlišná také v tom, že není nijak centralizovaná. Podobně jako služba elektronické pošty sestává *Jabber* z nezávislých serverů, které jsou vzájemně federalizovány a doručují si jednotlivé zprávy. Z tohoto důvodu je také využit obdobný identifikátor uživatele (tzv. „*Jabber ID*“), který je ve tvaru uživatelské.jméno@server.tld. V první fázi přenosu je zpráva předána na server uvedený za symbolem „zavináče“ a zde je poté doručena konkrétnímu uživateli.

Systém *Aidbot* ve výchozím stavu předpokládá existenci vlastního XMPP serveru. Může být ale nakonfigurován i pro fungování prostřednictvím již existujícího (např. veřejně dostupného) serveru. Je však otázkou, nakolik je takové řešení vhodné, neboť množství zasílaných zpráv a připojených uživatelů může v extrémním případě způsobit zvýšené zatížení XMPP serveru a snížit tak kvalitu služby pro ostatní uživatele. Podobné konfiguraci by tak měla předcházet dohoda se správcem dotčeného serveru.

5.1.3 Node.js

Node.js je platforma, která umožňuje pro vývoj serverových aplikací (především webových serverů a podobných) využít jazyka Javascript. Základním rysem tohoto aplikačního prostředí je, že aplikace nad ním vytvořené jsou jednovláknové, událostmi řízené a velmi hojně používají neblokující asynchronní zpracování vstupu a výstupu, což vede k minimalizaci režie procesoru a maxi-

3. Např. ICQ (<http://icq.com/>), Skype (<http://skype.com/>), Gadu-Gadu (<http://gadu-gadu.pl/>) a mnoho dalších.

malizaci výkonu. To je také hlavní rozdíl od standardních webových serverů, které pro každé HTTP spojení vytváří speciální proces nebo vlákno [42].

Tento rozdíl je nejlépe patrný, přijde-li na server požadavek na nějakou časově náročnou operaci. Tou může být například čtení souboru z disku nebo dlouhotrvající dotaz do databáze. V případě klasického modelu je v jednom vlákně webového serveru přijat požadavek i přečten obsah souboru a ten je následně zaslán klientovi jako odpověď. V závislosti na konfiguraci pak může být vlákno ukončeno, nebo uloženo zpět k ostatním předpřipraveným vláknům k dalšímu použití (tzv. „thread-pool“). Důležité ale je, že během čtení souboru z disku nemůže toto vlákno obsluhovat žádné další požadavky.

Je-li HTTP server implementován na platformě Node.js, mohou být v jednom vlákně během čtení obsahu souboru z disku zpracovány i další požadavky ostatních klientů. V okamžiku, kdy je dokončeno čtení souboru, je vyvolána událost, ta je zachycena definovaným posluchačem událostí a obsah souboru je zaslán klientovi, který o něj požádal [42]. Samozřejmě je možné nad Node.js implementovat webový server, který funguje i způsobem popsaným v předchozím odstavci. Vzhledem k tomu, že je ale Node.js tzv. „single-threaded“, tedy běží pouze v jediném vlákně, by vytvoření blokujícího HTTP serveru způsobilo velké problémy s výkonem.

Základem tohoto prostředí je jeden z v současné době nejvýkonnějších dostupných interpretů [43] Javascriptu, V8 od společnosti Google. Ten je napsán v jazyce C++ a původně byl navržen pro použití ve webovém prohlížeči *Chrome*. Nad interpretem je tenká vrstva dalšího kódu v C, knihovna *libUV*, která mj. poskytuje abstrakci nad jednotlivými operačními systémy, na nichž je možno Node.js provozovat a minimální nutné zázemí pro fungování celé platformy – smyčku pro zpracování a vyhodnocení příchozích událostí, obsluhu vstupně-výstupních zásobníků atd.

S Node.js je nedělitelně spojen také systém *NPM*⁴. Tento balíčkovací systém slouží pro správu závislostí ve vznikající aplikaci a je instalován společně s Node.js. Během vývoje aplikace může nastat situace, kdy je potřeba použít nějaký kód či komponentu, které již byla vytvořeny jiným autorem. NPM poskytuje jednoduchý způsob, jak všechny tyto závislosti nainstalovat a zároveň dále aktualizovat a spravovat. Ačkoliv pro NPM existují alternativy (velmi efektivně lze např. pro správu závislostí používat *Makefile* známé zejména z UNIXových systémů), je tento systém prakticky jediným, který je v komunitě vývojářů masově používán.

Node.js byl pro implementaci systému *Aidbot* zvolen zejména pro svou univerzálnost. Geisendörfer v [44] popisuje základní případy, kdy je vhodné Node.js použít a kdy je naopak správné se jeho užití vyvarovat. Mezi aplikace,

4. Node Package Manager, viz <http://npmjs.org/>

které je vhodné v Node.js implementovat zahrnuje mj. i software fungující v reálném čase, jednostránkové webové aplikace a tvorbu aplikačních rozhraní. Jak bylo popsáno v kap. 4.3, celý Aidbot bude rozdělen do tří subsystémů, které budou mít tyto přesně popsané vlastnosti. I z tohoto důvodu je vhodné Node.js použít, neboť existuje oprávněný předpoklad, že budou přesně využity jeho silné stránky. Univerzálností zmíněnou v začátku tohoto odstavce je pak myšlen fakt, že ačkoliv se jedná o tři heterogenní subsystémy, Node.js umožní jejich implementaci na jedné platformě, prostřednictvím jednoho programovacího jazyka.

5.1.4 Socket.IO

Socket.IO je rámec pro podporu tvorby webových aplikací pracujících v reálném čase. Jeho hlavním přínosem (a tedy také tím, co rozhodlo pro jeho použití v Aidbotu) je, poskytnutí možnosti využívat síťové sokety i v prohlížečích, které rozhraní protokolu WebSocket neobsahují [45]. Na rozdíl od podobných knihoven je jeho předností, že kromě javascriptového kódu, který se vykonává na straně klienta, obsahuje také modul pro Node.js.

Díky tomu, že klientskou i serverovou část přenosu obsluhuje jedna softwarová struktura, může být pro komunikaci využit vlastní protokol. Ten poskytuje abstrakci nad jednotlivými možnostmi vytvoření persistentního spojení mezi klientem a serverem a jeho aplikační rozhraní je velmi podobné protokolu WebSocket [45]. Obsahuje všechny možnosti popsané v kapitole 3, jejich výčet rozšiřuje ještě o technologii *Adobe Flash Socket* (Tato technologie pro vytvoření komunikačního kanálu potřebuje uživatelem nainstalované rozšíření prohlížeče od společnosti Adobe.).

Protokol sám pak funguje podobně jako většina ostatních síťových protokolů. Na začátku proběhne tzv. „handshake“, během kterého dá klient serveru najevo, že je připraven komunikovat. Poté následuje fáze nazvaná „transport connection“. V průběhu této fáze klient serveru oznámí, jaké způsoby komunikace podporuje, a podle toho je se serverem dohodnut způsob další komunikace. Server pak sdělí klientovi URL, na které bude přenos dat probíhat [46]. Dále je již chování odlišné v závislosti na použité metodě pro komunikaci. Rozhraní pro programátora ale zůstává neměnné – nezávisle na metodě je vytvořen soket, do kterého zapisuje data a ta jsou přenášena ke klientovi.

Pro implementaci Aidbotu byl zvolen protokol WebSocket zejména kvůli předpokladu, že během několika málo let se stane prakticky jedinou možností pro psaní webových aplikací pracujících v reálném čase. Vzhledem k tomu, že tato doba ještě nenastala, byl zvolen rámec Socket.IO pro doplnění této funkcionality do některých starších prohlížečů. Ačkoliv je Socket.IO v této práci primárně užít k obalení protokolu WebSocket (resp. dalších způsobů komuni-

kace v reálném čase), sám o sobě umožňuje i mnohé další funkce, např. broadcastové vysílání zpráv připojeným klientům, ukládání metadat o jednotlivých klientech apod.

5.1.5 AngularJS

AngularJS je rámec vystavěný na architektonickém vzoru MVC a slouží pro tvorbu jednostránkových javascriptových aplikací. V systému AidBot bude využit pro tvorbu obou webových aplikací, které komunikují s centrální webovou službou.

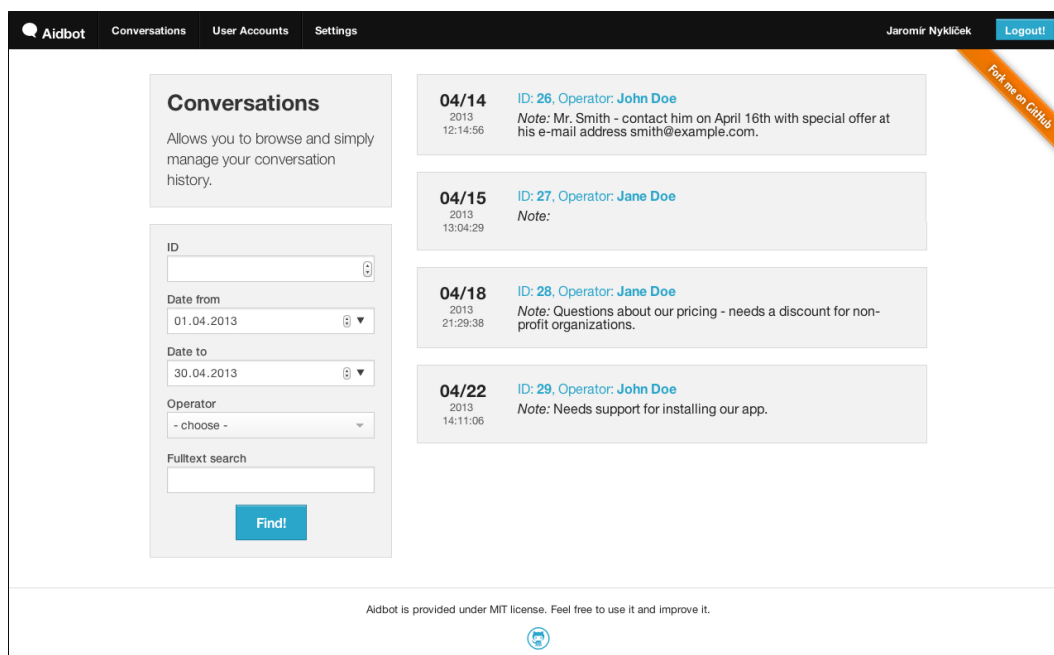
Předpokladem, ze kterého bylo vycházeno při vývoji AngularJS je, že deklarativní programování je výhodné použít pro tvorbu uživatelských rozhraní, zatímco imperativní přístup je vhodný pro vyjádření doménové logiky [47]. Cílem tohoto rámce je rozšířit HTML, které samo je velmi kvalitním deklarativním jazykem pro tvorbu statických dokumentů, o podporu tvorby webových aplikací takovým způsobem, aby jej bylo možné použít jako samostatný šablonovací jazyk [47]. Toho je dosaženo rozšířením HTML o další atributy jednotlivých značek (tzv. „direktivy“, v kódu jsou označeny prefixem ng-), které jsou během sestavení objektového modelu dokumentu přechteny a na jejich základě je tento model upraven.

Deklarativním přístupem k tvorbě uživatelského rozhraní je docíleno vyšší úrovně abstrakce, neboť aplikační logika je odstíněna od manipulace s objektovým modelem dokumentu. To zajišťuje vyšší čitelnost a přehlednost zdrojového textu a zároveň vylepšuje testovatelnost aplikace.

Další vlastnost, která přispívá ke snížení objemu kódu aplikace, a tedy i ke zvýšení jeho čitelnosti, je označována jako „two-way data binding“ (neboli „oboustranná vazba dat“). Ta automaticky zajišťuje aktualizaci dat v modelu při jejich změně skrze uživatelské rozhraní a naopak. Implementačně je toto chování řešeno prostřednictvím návrhového vzoru Observer. AngularJS obsahuje službu, která sleduje veškeré změny dat ve view. V případě, že je zachycena událost, značící provedení změny, je vyvolána akce controlleru, která upraví stav modelu. Opačné chování pak vyplývá ze samé podstaty vzoru MVC.

Ačkoliv podobných frameworků existuje několik, AngularJS byl už od počáteční analýzy jasným kandidátem pro vývoj uživatelského rozhraní. Zvolen byl zejména kvůli početné komunitě vývojářů, která s ním pracuje, a je tedy možné nalézt velké množství ukázek kódu a doporučení k vývoji. Dalším důvodem byla velmi propracovaná dokumentace s ukázkami jednotkových testů, které do jisté míry dávají návod, jak psát testy vlastní. Část dokumentace a především množství záznamů z vývojářských konferencí jsou dostupné

i v českém jazyce, neboť AngularJS je vyvíjen malým týmem tvořeným českým a dvěma slovenskými programátory.



Obrázek 5.1: Uživatelské rozhraní administrace vytvořené v AngularJS.

5.2 Detaily řešení vybraných problémů

Následující odstavce ve stručnosti popisují klíčové problémy, které bylo během vývoje nutné vyřešit nebo rozhodnout. Zmíněny jsou zejména takové obtíže, které nebyly zcela zřejmé v průběhu návrhu a poprvé se objevily až ve fázi samotné realizace.

5.2.1 Zasílání zpráv mezi klientem a serverem

Pro zasílání zpráv v reálném čase byl zvolen protokol WebSocket, resp. rámec *Socket.IO*, který nad ním vytváří abstrakci a poskytuje podobnou funkcionalitu i prohlížečům, které jej přímo nepodporují (viz kap. 5.1.4).

V okamžiku otevření komunikačního okna ve webovém prohlížeči zákazníka je na této straně komunikace otevřen nový soket s adresou serveru a vyvolána událost `openConversation`. Ta je na straně serveru zachycena příslušným posluchačem událostí, který otevře i druhou stranu spojení. Tímto způsobem je vytvořen persistentní komunikační kanál, který je využíván po celou dobu spojení.

```

1 // klient
2 var socket = io.connect('http://localhost:8080');
3 socket.emit('sendmessage', "Hello_World");
4
5 // server
6 io.sockets.on('connection', function (socket) {
7     var cm = new ConversationManager(socket);
8     cm.persist();
9
10    socket.on('sendmessage', function (data) {
11        cm.saveMessage(data);
12        cm.moveToXmpp(data);
13    });
14 }

```

Zdrojový kód 5.1: Ukázka vyvolání události na klientovi a její zpracování na serveru.

I komunikace samotná je poté událostmi řízená. Každou akci, ať už na straně serveru nebo klienta, lze doprovodit vyvoláním vlastní události, na kterou má komunikační partner možnost reagovat prostřednictvím přiřazeného posluchače (viz ukázku kódu 5.1). Vzhledem k tomu, že klient vyvolává pouze dvě události (`openConversation` a `sendMessage`), zajišťujících jejich obsluhu na serveru třída `ConversationManager`. Pokud by v budoucnu došlo k rozšíření této sady událostí, bylo by vhodné jejich obsluhu vyčlenit do speciální třídy (např. `EventManager`) z důvodu zachování tzv. „single responsibility“⁵ principu.

5.2.2 RESTful webová služba

RESTful webová služba, jak už bylo zmíněno v kapitole 5.1.1, využívá HTTP protokolu a jeho metod pro přístup k jednotlivým zdrojům. Její běh zajišťuje jednoduchý webový server napsaný v jazyce JavaScript, který je dostupný v balíku `http` v základní instalaci Node.js. Tento stejný webový server zároveň slouží i k obsluhování požadavků na statický obsah.

Po úvaze byla nakonec i pro architekturu webové služby zvolena implementace vzoru MVC. Hlavním argumentem pro toto rozhodnutí byl zejména fakt, že webová služba použitím tohoto vzoru dostane jasně definovanou strukturu, která přinese výhody zejména při údržbě kódu nebo jeho rozšiřování. Protože webová služba je určena ke vzájemné interakci mezi aplikacemi, nikoliv mezi aplikací a uživatelem, není v tomto případě nutné implementovat i view vrstvu. Data budou odesílána přímo `controllery` ve formátu JSON.

5. Jeden z pěti základních principů objektově orientovaného programování, který říká, že každá třída by měla zapouzdřovat právě jednu zodpovědnost. Více viz např. http://en.wikipedia.org/wiki/Single_responsibility_principle.

Kód	Význam	Popis
200	OK	Vracen v případě korektního zpracování HTTP požadavku.
304	Not Modified	Obsah nebyl změněn, klient má u sebe jeho aktuální verzi.
401	Unauthorized	Neautorizovaný přístup. Klient zaslal chybné přihlašovací údaje.
403	Forbidden	Klient požaduje informace o zdroji, ke kterému nemá přístup.
404	Not Found	Klient požaduje zdroj z URI, která neexistuje.
500	Internal Server Error	Chyba na straně serveru, která nesouvisí s požadavkem klienta.

Tabulka 5.2: HTTP kódy použité ve webové službě.

Pro návrh rozhraní webové služby byl použit nástroj *apiary.io*. Ten umožňuje vznikající rozhraní popsat pomocí pro tento účel navrženého doménově specifického jazyka, a to včetně zadání testovacích dat. K tomuto prototypu je poté možné se z klientské aplikace rovnou připojit, a vyzkoušet tak pohodlnost a účelnost jeho použití. Z dokumentačních komentářů prototypu je navíc automaticky vygenerována kompletní dokumentace popisující rozhraní webové služby. Takto získaná dokumentace je dostupná jako příloha A této práce.

Největší přínos podobného nástroje ale patrně nastává při vývoji ve větším týmu. Po sestavení prototypu může jedna část vývojářů pracovat na klientské aplikaci a druhá paralelně s ní vyvíjet webovou službu. Vzhledem k tomu, že k prototypu vždy existuje aktuální dokumentace, může jedna ze skupin rozhraní služby během vývoje modifikovat a druhá je o této úpravě okamžitě informována a má možnost na ni ve svém kódu reagovat.

Pro korektní chování RESTful webové služby bylo také nutné nastudovat významy jednotlivých HTTP kódů, které jsou vráceny klientovi v odpovědi na požadavek. Přehled kódů, které jsou v aktuální verzi webové služby využity, obsahuje i s jejich popisem tabulka 5.2.

5.2.3 Ukládání dat

Přestože je momentálním trendem ve vývoji Node.js aplikací využívat zejména bezschémové databáze, bylo poměrně záhy rozhodnuto, že pro uklá-

dání dat bude použita klasická relační databáze. Kromě objektivních měřítek jako jsou například výkon, možnosti škálování, celková vhodnost řešení apod. vstupují do volby systému řízení báze dat také subjektivní pohnutky. Těmi jsou zejména znalost dané platformy a efektivita práce s ní. Zejména z těchto osobních důvodů byl zvolen databázový software *MySQL Community Server*, který momentálně spravuje a vyvíjí firma Oracle.

Jako databázová vrstva byla vybrána knihovna `node-mysql`. Její hlavní předností je, že je napsána pouze v JavaScriptu, což umožňuje její snadnou instalaci prostřednictvím nástroje `npm`, není tedy nutné instalovat ji na úrovni celého systému pomocí balíčkovacího systému nebo kompilací ze zdrojových textů. Společně s vhodnou architekturou celé aplikace, která izoluje práci s databází do jednoho místa, je toto základním předpokladem pro snadnou budoucí výměnu databázového serveru, dojde-li ke změně požadavků.

5.2.4 Rozhraní pro XMPP server

Rozhraní, které zajišťuje komunikaci mezi XMPP serverem a zbytkem aplikace, se nachází ve třídách `XmppAdapter` a `XmppClient`. Třída `XmppClient`, jak název napovídá, je vlastní implementací XMPP klienta nad knihovnou `node-xmpp`. Tato knihovna zajišťuje základní úroveň práce s protokolem. Vývojář je tak odstíněn od nutnosti sestavovat jednotlivé XML zprávy jako textové řetězce, namísto toho může využít k tomu připravené třídy, jejichž instance pouze naplní daty. Třída `XmppAdapter`, která je zároveň potomkem systémové třídy `EventEmitter`, slouží k úpravě jejího rozhraní tak, aby události pocházející z XMPP serveru byly snadno zpracovatelné ve zbytku systému.

Samotné zasílání zpráv mezi XMPP serverem a zbytkem systému je realizováno poměrně přímočarým a jasným způsobem. Konverzace je považována za zahájenou v okamžiku, kdy je v prohlížeči klienta otevřeno komunikační okno, neboť v tuto chvíli je otevřen nový soket a na serveru je vytvořena instance třídy `ConversationManager`, která řídí tok zpráv v konverzaci.

Tato instance v sobě asociuje jak daný soket, tak instanci entitní třídy `Conversation`, která reprezentuje právě probíhající konverzaci. Tato entita je uložena do databáze, kde jí je přiřazen unikátní číselný identifikátor, tzv. `ConversationID`. Kromě objektu typu `Conversation` v sobě `ConversationManager` obsahuje také instanci `XmppClientu`. Tento objekt zadá XMPP serveru příkaz k registraci nového uživatele s JID ve tvaru `aidbot.[ConversationID]@server.tld` a náhodně vygenerovaným heslem. Hned po provedení úspěšné registrace tohoto uživatele se `XmppClient` připojí pod jeho identitou k XMPP serveru.

Tímto způsobem je vytvořena cesta, po které mohou putovat zprávy z komunikačního okna v prohlížeči zákazníka až do Jabber klienta na počítači ope-

rátora a zpět. Ve chvíli, kdy zpráva dorazí skrze soket na server, je převzata ConversationManagerem, který z ní vytvoří instanci entitní třídy Message a uloží ji do databáze s vazbou na aktuální konverzaci. Po uložení ji předá své instanci XmppClientu, jenž ji zašle připojenému operátorovi prostřednictvím XMPP serveru. Komunikace v opačném směru (tedy od operátora k zákazníkovi) probíhá analogicky.

Propojení webové aplikace s XMPP serverem byla patrně nejsložitějším úkolem praktické části diplomové práce. Bylo nutné detailně nastudovat celý princip fungování tohoto protokolu a poté de-facto implementovat vlastního XMPP klienta v jazyce JavaScript.

5.2.5 Integrace komunikačního okna

Jedním z požadavků na Aidbot kladených byla snadná integrace do systémů třetích stran. Propojení těchto dvou aplikací je realizováno prostřednictvím HTML struktury, jež je vložena do šablony webu, do kterého je Aidbot integrován (viz ukázkou kódu 5.2).

```

1 <div id="aidbot">
2   <div id="aidbot-frame"></div>
3   <div id="aidbot-button" onclick="aidbot.toggle()">Open</div>
4 </div>
5
6 <script type="text/javascript"
7   src="http://server.tld/window/window.js"></script>
8 <link rel="stylesheet"
9   type="text/css" href="http://server.tld/window/window.css" />

```

Zdrojový kód 5.2: Integrační kód.

Integrační kód obsahuje klientský program v jazyce JavaScript, jenž obstarává vytvoření komunikačního soketu a zároveň také zobrazování a skrývání samotného komunikačního okna na webové stránce. Jeho součástí je i kaskádový styl, který definuje základní vzhled a formátování okna. Tento styl není pro korektní funkčnost programu nutný, integrátor má možnost se rozhodnout jej nepoužít a nahradit ho vlastní implementací.

5.2.6 Ukládání časových údajů

Vzhledem k povaze systému Aidbot může snadno nastat situace, kdy jsou operátor a uživatel, jemuž je poskytována technická podpora, geograficky velmi vzdáleni. S tím souvisí i problematika jednotlivých časových pásem. Pokud bychom jednotlivé časové známky v konverzaci ukládali tak, jak jsou doručeny od klienta resp. operátora, byl by časový rámec celé konverzace znehodnocen. Mohlo by dokonce docházet k paradoxním situacím, kdy při pouhém pohledu na čas odeslání odpověď předchází otázce.

Přístup, který vede k řešení tohoto problému, je poměrně prostý. Časový údaj získaný od uživatele převedeme na GMT, tedy univerzální čas. S tímto formátem časového razítka dále pracujeme a jakmile data vypisujeme do uživatelského rozhraní, dojde k jejich konverzi na časové pásmo uživatele, které získáme např. z nastavení prohlížeče.

6 Závěr

Hlavním cílem této práce bylo analyzovat postupy a techniky sloužící k tvorbě webových aplikací pracujících v reálném čase a poté implementovat systém Aidbot, který bude těchto poznatků využívat. Systém sám slouží pro poskytování okamžité technické podpory uživatelům webové aplikace, která jej integruje. Analytické části záměru se věnují především druhá a třetí kapitola, návrhu architektury aplikace a její implementaci pak kapitoly 4 a 5. Výstupem a hlavním přínosem celé diplomové práce tak je zejména základní implementace tohoto softwaru, která je uvolněna s otevřeným zdrojovým kódem a pod svobodnou licencí.

Při tvorbě architektury a z ní vycházejícího návrhu programu bylo postupováno s důrazem na využití ověřených architektonických a návrhových vzorů. Aplikace těchto postupů je do jisté míry zárukou proveditelnosti budoucích úprav a rozšíření. To je vzhledem k tomu, že je Aidbot uvolněn jako open-source, poměrně zásadní, neboť lze předpokládat, že postupem času bude dále vylepšován nejen autorem, ale i komunitou dalších programátorů. Zdrojový kód je momentálně uložen v repozitáři na serveru Github¹, odkud je možné jej volně stáhnout a modifikovat jej. Z tohoto důvodu je tedy pravděpodobné, že budou vznikat i alternativní větve tohoto programu (tzv. „forky“).

Praktickou implementací softwaru byla také ověřena domněnka, že v prostředí současného webu lze pouze s využitím standardních prostředků efektivně tvořit aplikace, které pracují v reálném čase. Node.js a ekosystém příbuzných nástrojů představuje zajímavou a funkční alternativu k zavedeným serverovým technologiím. Výhoda událostmi řízeného modelu, který Node.js využívá, vynikne zejména v kombinaci s protokolem WebSocket, protože umožní nastavit síťový soket jako posluchače těchto událostí a tím minimalizovat zpoždění v komunikaci. Zároveň se ukázalo, že ačkoliv není rozhraní tohoto protokolu v prohlížečích ještě dostatečně rozšířeno, existují nástroje (viz 5.1.4), které je v případě nedostupnosti nahrazují jinými technikami, a to při zachování konzistentního rozhraní pro vývojáře.

Systém Aidbot zatím rozhodně není ve svém cíli, a to zejména z důvodu, že u podobného projektu je téměř nemožné označit jej za dokončený. Vývoj bude nadále pokračovat a jednou z budoucích cest je i možnost provozu Aidbotu jako SaaS.

1. Služba pro sdílení zdrojového kódu. <http://github.com/>, adresa repozitáře je <https://github.com/jaromirnyklicek/aidbot>.

Literatura

- [1] JABLONSKI, S., PETROV, I., MEILER, C. a MAYER, U. *Guide to Web Application and Platform Architectures*. Heidelberg : Springer-Verlag, 2004. 255 s. ISBN 3-540-00947-7.
- [2] ALONSO, G., CASATI, F., KUNO, H. a MACHIRAJU, V. *Web Services: Concepts, Architectures and Applications*. Heidelberg, Germany, Springer-Verlag, 2004. 354 s. ISBN 3-540-44008-9.
- [3] WORLD WIDE WEB CONSORTIUM. HAAS, H., BROWN, A. *Web Services Glossary* [online]. 8. 8. 2003, poslední revize 11. 11. 2004. [cit. 14. 1. 2013]. Dostupné z: <http://www.w3.org/TR/ws-gloss/>
- [4] RICHARDSON, L., RUBY, S. *RESTful Web Services: Web services for the real world*. Sebastopol : O'Reilly Media, 2007. 454 s. ISBN 978-0-596-52926-0.
- [5] GALLAUGHER, J., RAMANATHAN, S. *The Critical Choice of Client Server Architecture: A Comparison of Two and Three Tier Systems* [online]. Poslední revize 28. 7. 1995. [cit. 3. 2. 2013]. Dostupné z: <https://www2.bc.edu/~gallaugh/research/ism95/cccsa.html>
- [6] FIELDING, R. , GETTYS, J., MOGUL, J., et al. *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1* [online]. June 1999. [cit. 13. 1. 2013]. Dostupné z: <http://tools.ietf.org/html/rfc2616>
- [7] CRANE, D., MCCARTHY, P. *Comet and Reverse Ajax: The Next-Generation Ajax 2.0*. Berkeley : aPress, 2008. 148 s. ISBN 978-1-59059-998-3
- [8] WORLD WIDE WEB CONSORTIUM. AUBOURG, J., SONG, J., STEEN R. M. H. *XMLHttpRequest* [online]. 4. 5. 2006, poslední revize 6. 12. 2012. [cit. 14. 1. 2013]. Dostupné z: <http://www.w3.org/TR/XMLHttpRequest/>
- [9] MAZZETTI, P., NATIVI, X., SACCO, A. AND BIGAGLI, L. Integration of REST style and AJAX technologies to build Web applications. In *Third International Conference on Information and Communication Technologies: From Theory to Applications, Damascus, Syria, April, 7-11 2008*. s. 1-6.
- [10] CROCKFORD, D. *The application/json Media Type for JavaScript Object Notation (JSON)* [online]. July 2006. [cit. 1. 2. 2013] Dostupné z: <http://www.ietf.org/rfc/rfc4627.txt>
- [11] SHKLAR, L., ROSEN, R. *Web Application Architecture: Principles, Protocols and Practices*. Chichester : John Wiley & sons, Ltd., 2003. 357 s. ISBN 0-471-48656-6.

- [12] WORLD WIDE WEB CONSORTIUM. HADLEY, M. *Web Application Description Language* [online]. 31. 8. 2009. [cit. 1. 2. 2013]. Dostupné z: <http://www.w3.org/Submission/2009/SUBM-wadl-20090831>
- [13] DOC FORGE. *Web Application* [online]. [cit. 1. 2. 2013] Dostupné z: http://docforge.com/wiki/Web_application
- [14] CECCHET, E., CHANDA, A., ELNIKETY, S., MARGUERITE, J. a ZWANEPOEL, W. Performance comparison of middleware architectures for generating dynamic webcontent. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware Rio de Janeiro, Brazil, June 16-20, 2003*. New York : Springer-Verlag, 2003. s. 242-261. ISBN 3-540-40317-5.
- [15] HARRISON G. *10 things you should know about NoSQL databases* [online]. 26. 8. 2010. [cit. 28. 4. 2013] Dostupné z: <http://www.techrepublic.com/blog/10things/10-things-you-should-know-about-nosql-databases/1772>
- [16] TŮMA, J. *RIA* [online]. 30. 7. 2012. [cit. 3. 2. 2013] Dostupné z: <http://www.webcesky.cz/ria/>
- [17] MARTÍNEZ-RUIZ, J. F., MUÑOZ ARTEAGA, J., VANDERDONCKT, J. a GONZÁLEZ-CALLEROS, M. J. *A First Draft of a Model-driven Method for Designing Graphical User Interfaces of Rich Internet Applications* [online]. [cit. 3. 2. 2013]. Dostupné z: <https://lilab.isys.ucl.ac.be/bchi/publications/2006/Martinez-LAWeb2006.pdf>
- [18] KUUSKERI, J., MIKKONEN, T. Partitioning web applications between the server and the client. In: *Proceedings of the 2009 ACM symposium on Applied Computing*. ACM, 2009. s. 647-652.
- [19] MIKKONEN, T., TAIVALSAARI, A. Web Applications — Spaghetti Code for the 21st Century. In *Proceedings of the 6th ACIS International Conference on Software Engineering Research, Management and Applications (SERA'2008, Prague, Czech Republic, August 20-22, 2008)*. IEEE Computer Society. s. 319-328.
- [20] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *The NIST Definition of Cloud Computing* [on-line]. September 2011, poslední revize 27. 4. 2012. [cit. 14. 2. 2013]. Dostupné z: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>
- [21] SOFTWARE & INFORMATION INDUSTRY ASSOCIATION. *Software as a Service: Strategic Background* [online]. February 2001. [cit. 14. 2. 2013] Dostupné z: <http://www.siia.net/estore/ssb-01.pdf>
- [22] FERRATE, A. *Google Wave: Up and Running*. Sebastopol : O'Reilly Media, Inc., 2010. 304 s. ISBN 1449390803.

- [23] DAWSON, CH. *How will Google Wave be reincarnated?* [online]. 4. 8. 2010. [cit. 14. 2. 2013]. Dostupné z: <http://www.zdnet.com/blog/google/how-will-google-wave-be-reincarnated/2344>
- [24] CRUNCHBASE.COM. *Google Wave* [online]. 28. 5. 2009. [cit. 14. 2. 2013]. Dostupné z: <http://www.crunchbase.com/product/google-wave>
- [25] GLOBAL WEB INDEX. *SOCIAL PLATFORMS GWI.8 UPDATE: Decline of Local Social Media Platforms* [online]. 22. 1. 2013. [cit. 14. 2. 2013]. Dostupné z: <http://globalwebindex.net/thinking/social-platforms-gwi-8-update-decline-of-local-social-media-platforms/>
- [26] ALINONE, A. *Comet and Push Technology* [online]. 19. 10. 2007. [cit. 18. 2. 2013]. Dostupné z: <http://cometdaily.com/2007/10/19/comet-and-push-technology/>
- [27] LOWENSTEIN, R. *Origins of the Crash: The Great Bubble and Its Undoing*. New York : Penguin Books, 2004. 272 s. ISBN 1594200033.
- [28] LUBBERS, P., GRECO, F. *HTML5 Web Sockets: A Quantum Leap in Scalability for the Web?* [online]. [cit. 19. 4. 2013] Dostupné z: <http://www.websocket.org/quantum.html>
- [29] MALÝ, M. *Kometa přináší web v reálném čase* [online]. 23. 8. 2010. [cit. 15. 2. 2013]. Dostupné z: <http://www.zdrojak.cz/clanky/kometa-prinasi-web-v-realnem-case/>
- [30] DOKUMENTACE DIRECT WEB REMOTING. *Reverse ajax* [online]. [cit. 18. 2. 2013]. Dostupné z: <http://directwebremoting.org/dwr/documentation/reverse-ajax/index.html>
- [31] RUSSELL, A. *Comet: Low Latency Data for the Browser* [online]. 3. 3. 2006. [cit. 7. 4. 2013]. Dostupné z: <http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/>
- [32] RUSSELL, A., WILKINS, G., DAVIS, D. a NESBITT, M. *The Bayeux Specification* [online]. 2007. [cit. 7. 4. 2013] Dostupné z: <http://svn.cometd.com/trunk/bayeux/bayeux.html>
- [33] DOKUMENTACE PROJEKTU COMETD. *Preface* [online]. 2011, poslední revize 4. 4. 2013. [cit. 7. 4. 2013] Dostupné z: <http://docs.cometd.org/reference/>
- [34] MALÝ, M. *Web Sockets* [online]. 14. 12. 2009. [cit. 27. 2. 2013]. Dostupné z: <http://www.zdrojak.cz/clanky/web-sockets/>
- [35] FETTE, I., MELNIKOV, A. *The WebSocket Protocol* [online]. December 2011. [cit. 27. 2. 2013]. Dostupné z: <http://tools.ietf.org/html/rfc6455>

- [36] WORLD WIDE WEB CONSORTIUM. HICKSON, I. *The WebSocket API* [online]. 29. 9. 2011. [cit. 27. 1. 2013]. Dostupné z: <http://www.w3.org/TR/2011/WD-websockets-20110929/>
- [37] NET APPLICATIONS. *Desktop Browser Version Market Share* [online]. March 2013. [cit. 7. 4. 2013]. Dostupné z: <http://www.netmarketshare.com/browser-market-share.aspx?qprid=2&qpcustommd=0>
- [38] FLANAGAN D. *JavaScript - The Definitive Guide*. 6th Edition. Sebastopol : O'Reilly Media, Inc., 2011. 1100 s. ISBN 978-0-596-80552-4.
- [39] FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures* [online]. Irvine, 2000. 162 s. Disertační práce na University Of California. Školitel Richard N. Taylor. [cit. 15. 4. 2013] Dostupné z: http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
- [40] FIELDING, R. T. Principled Design of the Modern Web Architecture. *Journal of ACM Transactions on Internet Technology*. 2002. Roč. 2, č. 2., s. 115-150. ISSN 1533-5399.
- [41] SAINT-ANDRE, P., SMITH, K., TRONCON, R. *XMPP: The Definitive Guide: Building Real-Time Applications with Jabber Technologies*. Sebastopol : O'Reilly Media, 2009. 288 s. ISBN: 978-0-596-52126-4.
- [42] FINLEY, K. *Wait, What's Node.js Good for Again?* [online]. 25. 1. 2011. [cit. 7. 4. 2013] Dostupné z: <http://readwrite.com/2011/01/25/wait-whats-nodejs-good-for-aga>
- [43] RESIG, J. *JavaScript Performance Rundown* [online]. 3. 9. 2008. [cit. 7. 4. 2013] Dostupné z: <http://ejohn.org/blog/javascript-performance-rundown/>
- [44] GEISENDÖRFER, F. *Felix's Node.js Convincing the boss guide* [online]. 2011. [cit. 11. 4. 2013] Dostupné z: http://nodeguide.com/convincing_the_boss.html
- [45] WALSH, D. *WebSocket and Socket.IO* [online]. 29. 11. 2010. [cit. 11. 4. 2013] Dostupné z: <http://davidwalsh.name/websocket>
- [46] DOKUMENTACE PROJEKTU SOCKET.IO. *Socket.IO protocol* [online]. [cit. 11. 4. 2013] Dostupné z: <https://github.com/LearnBoost/socket.io-spec>
- [47] DOKUMENTACE PROJEKTU ANGULARJS. *What Is Angular?* [online]. [cit. 16. 4. 2013] Dostupné z: <http://docs.angularjs.org/guide/overview>

A Dokumentace rozhraní webové služby

Aidbot RESTful Web API

Aidbot's RESTful webservice: documentation & examples.

TABLE OF CONTENTS

1. Conversation Resources
2. User Resources
3. Settings Resource

1 Conversation Resources

The following is a section of resources related to the conversation. It allows you to list a and modify conversations realized through the XMPP and WebSocket chat system.

1.1 GET /conversations

List of all conversations.

Returns a collection of JSONs, each representing a single conversation.

Each conversation includes a list of all messages.

REQUEST

raw

RESPONSE

200 (OK)
Content-type: application/json

```
[
  {
    "id": 26,
    "operator": 2,
    "operatorName": "John Doe",
    "date": "2013-04-14T12:14:56",
    "note": "Mr. Smith - contact him on April 16th with special offer at his e-mail address smith@example.com.",
    "messages": [
      {
        "author": "user",
        "date": "2013-04-14T12:14:57",
        "text": "Hi, can you help me please?"
      }
    ]
  },
  {
    "id": 27,
    "operator": 3,
    "operatorName": "Jane Doe",
    "date": "2013-04-15T13:04:29",
    "note": "",
    "messages": [
      {
        "author": "user",
```

```
[
  {
    "date": "2013-04-14T12:14:57",
    "text": "Hi, can you help me please?"
  }
]
```

1.2 GET /conversations/{id}

Returns a single conversation identified by id passed in URL.

REQUEST

raw

RESPONSE

200 (OK)
Content-type: application/json

```
{
  "id": 26,
  "operator": 2,
  "operatorName": "John Doe",
  "date": "2013-04-14T12:14:56",
  "note": "...",
  "messages": [
    {
      "author": "user",
      "date": "2013-04-14T12:14:57",
      "text": "Hi, can you help me please?"
    },
    {
      "author": "op",
      "date": "2013-04-14T12:16:21",
      "text": "Of course, what can I do for you?"
    }
  ]
}
```

1.3 PUT /conversations/{id}

Updates existing conversation.

Updated conversation is identified by id passed in URL, not by in request body. Returns the same object as the one that was passed.

REQUEST

raw

Content-type: application/json

```
{
  "id": 26,
  "operator": 2,
  "operatorName": "John Doe",
  "date": "2013-04-14T12:14:56",
  "note": "updated",
  "messages": [
    {
      "author": "user",
      "date": "2013-04-14T12:14:57",
      "text": "Hi, can you help me please?"
    },
    {
      "author": "op",
      "date": "2013-04-14T12:16:21",
      "text": "Of course, what can I do for you?"
    }
  ]
}
```

RESPONSE

200 (OK)
Content-type: application/json

```
{
  "id": 26,
  "operator": 2,
  "operatorName": "John Doe",
  "date": "2013-04-14T12:14:56",
  "note": "updated",
  "messages": [
    {
      "author": "user",
      "date": "2013-04-14T12:14:57",
      "text": "Hi, can you help me please?"
    },
    {
      "author": "op",
      "date": "2013-04-14T12:16:21",
      "text": "Of course, what can I do for you?"
    }
  ]
}
```

2 User Resources

Complete CRUD for user management.

Server implementation also creates and modifies users at XMPP server.

2.1 GET /users

List of all users. Returns a collection of JSONs, each representing a single user.

REQUEST*raw***RESPONSE**

200 (OK)
Content-type: application/json

```
[
  {
    "id": 1,
    "username": "john.doe",
    "name": "John Doe",
    "role": "Admin"
  },
  {
    "id": 2,
    "username": "jane.doe",
    "name": "Jane Doe",
    "role": "op"
  }
]
```

2.2 GET /users/{id}

Single user identified by URL.

REQUEST*raw***RESPONSE**

200 (OK)
Content-type: application/json

```
{
  "id": 2,
  "username": "john.doe",
  "name": "John Doe",
  "role": "op"
}
```

2.3 PUT /users/{id}

User update.

Because of XMPP server, it's not possible to pass a new username, username is immutable.

REQUEST

raw

Content-type: application/json

```
{
  "id": 2,
  "password": "",
  "name": "John Doe",
  "role": "op"
}
```

RESPONSE

200 (OK)
Content-type: application/json

```
{
  "id": 2,
  "username": "john.doe",
  "name": "John Doe - modified",
  "role": "admin"
}
```

2.4 POST /users

Creates new user.

First step in server implementation is check if given username is unique. If not, webservice returns code "409 Conflict".

REQUEST

raw

Content-type: application/json

```
{
  "id": 2,
  "password": "",
  "name": "John Doe",
  "role": "op",
  "username": "john.doe"
}
```

RESPONSE

200 (OK)
Content-type: application/json

```
{
  "id": 2,
  "username": "john.doe",
  "name": "John Doe - new",
  "role": "admin"
}
```

RESPONSE

```
409 (Conflict)
Content-type: application/json
```

```
{
  "status": "Username is not unique"
}
```

2.5 DELETE /users/{id}

Deletes user.

REQUEST

raw

RESPONSE

```
200 (OK)
```

3 Settings Resource

Enables modification of server settings (such as offline messages, Aidbot URL, etc.)

3.1 PUT /settings

API is not very RESTful because of the fact, that there is no collection of settings, but only one record which is globally used.

REQUEST

raw

```
Content-type: application/json
```

```
{
  "url": "http://aidbot.c3po.cz/",
  "offlineMessage": "Sorry, we are no available at the moment, please leave us a
message.",
  "cookieExpiration": 7
}
```

RESPONSE

200 (OK)
Content-type: application/json

```
{
  "url": "http://aidbot.c3po.cz/",
  "offlineMessage": "Sorry, we are no available at the moment, please leave us a
message.",
  "cookieExpiration": 7
}
```

B Obsah přiloženého CD

V kořenovém adresáři přiloženého CD se nachází následující tři adresáře:

- `repo` – obsahuje klon repozitáře ze serveru Github s kompletními zdrojovými kódy celé aplikace. Více informací o systémových požadavcích a instalaci obsahuje soubor `README.md`.
- `doc` – obsahuje API dokumentaci vygenerovanou z dokumentačních komentářů ve zdrojovém textu.
- `thesis` – obsahuje elektronickou verzi této práce ve formátu PDF včetně příloh.