VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

JABBER BOT OVLÁDANÝ PŘES REST API

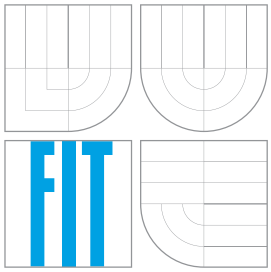
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VIKTOR STÍSKALA

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

JABBER BOT OVLÁDANÝ PŘES REST API

JABBER BOT REMOTED BY REST API

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VIKTOR STÍSKALA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MARTIN MILIČKA

BRNO 2013

Abstrakt

Tato bakalářská práce popisuje implementaci a využití bota ovládaného přes REST API a komunikujícího prostřednictvím XMPP protokolu. Aplikace je navržena s ohledem na minimální náročnost na systémové prostředky, možnosti rozšíření a škálování, čímž je zajištěna možnost napojení na neomezený počet uživatelů. Práce rovněž obsahuje stručný popis vlastností XMPP protokolu, který je základním stavebním kamenem celé aplikace. V dalších kapitolách je popsána implementace webové aplikace demonstrující chování bota a také možnosti rozšíření.

Abstract

This bachelor's thesis describes the implementation and use of Jabber bot controlled via REST API. The application is designed to meet the minimal demands on system resources, extensibility and scalability thus providing the ability to accept connections from an unlimited number of users. The work also contains a brief characterization of XMPP protocol, which is the cornerstone of the entire application. In subsequent chapters is described the implementation of a web application demonstrating the bot behavior and also the possibility of further extensions.

Klíčová slova

XMPP, Jabber, Python, Vzdálené ovládání, REST API

Keywords

XMPP, Jabber, Python, Remote control, REST API

Citace

Viktor Stískala: Jabber bot ovládaný přes REST API, bakalářská práce, Brno, FIT VUT v Brně, 2013

Jabber bot ovládaný přes REST API

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Martina Miličky.

.....
Viktor Stískala
15. května 2013

Poděkování

Touto cestou bych chtěl poděkovat vedoucímu práce, Ing. Martinovi Miličkovi za ochotu a odbornou pomoc při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Základní elektronickou komunikaci tak, jak ji známe dnes odstartovaly e-maily. Umožnily rychlé zasílání zpráv a rychle se dostaly velké oblibě. Dnes jsou i neodmyslitelnou součástí různých aplikací a tak se nemůžeme divit, že se denně celosvětově odešle okolo 144 miliard e-mailů [7]. Pokud ale chceme komunikovat v reálném čase, je mnohem výhodnější použít Instant Messaging (IM). Většina služeb v dnešní době rozesílá automatické informační e-maily, přitom by pro nás bylo mnohem příjemnější dostat tuto jednoduchou informační zprávu ve formě IM. Pro internetové služby je to navíc přínosem, protože je více pravděpodobné, že si uživatel přečte tuto zprávu rychleji, než e-mail.

Instant messaging služeb dnes existuje celá řada, od komerčních po open source. Mezi open source protokoly je nejznámější XMPP (Extensible Messaging and Presence Protocol), známý také pod názvem Jabber. Vzhledem k tomu, že XMPP je otevřený a decentralizovaný protokol [17], je možné si zřídit vlastní, např. firemní XMPP server, a to nejen pro interní komunikaci. Mnoho firem používá interní informační systémy, které jsou propojené na další služby, a to je právě jeden z příkladů, kde se vyplatí použít Jabber bota.

Cílem této bakalářské práce je vytvoření tzv. „bota“, ovládaného přes REST API umožňujícího interakci s XMPP protokolem. Bot je registrovaný jako samostatný XMPP uživatel a umožňuje tímto mimo jiné zasílání zpráv, zjišťování prezence (někdy také nazývané jako „stav“), komunikaci a reakci na příkazy ve skupinovém chatu, ukládání historie, apod. Chová se jako prostředník mezi aplikací a uživatelem, škála různých druhů využití je proto poměrně obsáhlá.

V současné době existuje několik podobných programů pracujících s protokolem XMPP, jejich použití je ale většinou pouze jednostranné a nepříliš upravitelné, například reakce na příkazy zasílané pomocí přímých zpráv. Jabber bot vyvinutý jako produkt této bakalářské práce se snaží těmto omezením vyhnout a nabízí programovatelné rozhraní, díky čemuž je jeho případná úprava či rozšíření jednoduchá.

Kapitola 2

Použité technologie

2.1 Python

Python je vysokoúrovňový objektově orientovaný dynamický skriptovací programovací jazyk. Standardní knihovna jazyka obsahuje mnoho nástrojů a knihoven, které usnadňují, ale hlavně urychlují práci programátora. Kromě standardní knihovny je možné doinstalovat mnoho balíčků třetích stran z různých zdrojů, z nichž nejpoužívanějším je dnes PyPI – Python Package Index.

Python byl vytvořen v roce 1990 Guido Van Rossumem, který dodnes zůstává jeho hlavním autorem. Jazyk je vyvíjen jako open source projekt, vydávaný pod licencí, která je kompatibilní s licencí GPL. Všechny doposud vydané verze jazyka jsou open source a až na drobné výjimky jsou i GPL kompatibilní. [6]

Jazyk má v současnosti dvě vývojové větve rozdělené podle verze a to `2.x.x` a `3.x.x`, přičemž hlavní úsilí je dnes již věnováno novější verzi. Starší verze je však především kvůli velkému rozšíření stále vyvíjená a získává i některé funkce z Pythonu verze `3.x.x`. [11]

V Projektu je použit jazyk Python verze 2.7, protože jsou na něm závislé některé použité knihovny (především Gevent).

2.2 HTTP

HTTP (Hypertext Transfer Protocol) je protokol pracující na aplikační vrstvě, původně navržený k přenosu dokumentů ve formátu HTML, dnes se mu však dostává mnohem širšího využití. Klíčovou vlastností tohoto protokolu je bezstavovost, kterou však v případě potřeby mohou suplementovat některé dodatečné hlavičky[1] [5].

V projektu je použit pro přenos informací ve formátu JSON[2], pro vzdálené ovládání Jabber bota. Samotná komunikace s XMPP serverem je však zajišťována prostřednictvím odlišného protokolu – XMPP, viz kapitola 3.

2.3 REST

Pro vzdálenou komunikaci mezi jednotlivými programy je zapotřebí určité rozhraní. Za historicky první typ tohoto rozhraní se považuje RPC (Remote Procedure Call), přičemž

[1] **HTTP Headers** – dodatečná data zasílaná spolu s požadavkem na server

[2] **JavaScript Object Notation** – standard využívající datové struktury odvozené z jazyka JavaScript, známé jako objekty

první zmínka o tomto typu rozhraní se datuje do roku 1976 [18]. Od tohoto typu rozhraní později začaly vznikat další, například XML-RPC využívající protokol HTTP pro komunikaci a XML pro kódování zpráv. Za zmínku stojí také SOAP (Simple Object Access Protocol), které je považováno za nástupce XML-RPC a nově není závislé na protokolu HTTP. Nasazení je rovněž poměrně časté na protokolu SMTP sloužícího pro e-mailovou komunikaci. Pro kódování zpráv SOAP využívá rovněž XML, ale jednotlivé zprávy jsou navíc zabalené v tzv. SOAP obálce. To s sebou přináší výhody např. v možnosti přidat k samotnému volání vzdálené procedury dodatečné informace, ale zároveň s sebou nese velkou nevýhodu v podobě nutnosti přenosu velkého množství dat i při krátkých předávaných zprávách.

REST (Representational State Transfer) je architektura rozhraní, kterou poprvé popsal Roy Fielding v části své disertační práce [4]. Mezi hlavní přednosti této architektury patří především bezstavovost, která umožňuje jednodušší škálování, menší náročnost na systémové prostředky (mohou se uvolnit ihned po ukončení požadavku a není potřeba zachovávat kontext mezi jednotlivými požadavky) a jednodušší zotavení z chyb. Při vývoji REST API se rovněž často používá datově nenáročné kódování zpráv, například ve formátu JSON[3], bez dodatečných obálek.

V projektu je REST API využito s použitím HTTP protokolu, což zahrnuje veškeré výhody tohoto protokolu, jako je například autentizace, nebo cachování. Tyto funkce se tudíž nemusí implementovat znovu jako součást API, ale je využito již stávajícího a především dobře otestovaného řešení. Pro kódování zpráv je použit JSON[4], implementace jiného typu kódování zpráv, je však vzhledem k návrhu aplikace možná.

2.4 Redis

Redis je open source distribuovaná `key-value` databáze. Data jsou ukládána v paměti s volitelnou perzistencí na pevný disk. Redis je také často označován jako *server pro datové struktury*, protože umožňuje ukládat řetězce, seznamy, asociativní pole, množiny a uspořádané množiny. Zkratka Redis pochází ze spojení *REmote Dictionary Server*, ale od doby vzniku tento program prošel velkým vývojem, takže toto označení je dnes již nepřesné.

Ukládání dat v paměti přináší velkou výhodu – rychlost. V praxi je Redis často využíván jako vyrovnávací paměť, nebo „pomocná databáze“ [8]. I přesto, že je Redis poměrně jednoduchou databází s minimem datových struktur, podporuje některé pokročilé funkce databázových systémů, jako jsou transakce, nebo návrhový vzor pro odesílání zpráv publish – subscribe. Publish – subscribe je variantou návrhového vzoru Observer [12].

2.4.1 NoSQL

Zařazení vůči NoSQL databázím je v současné době stále sporné. Redis disponuje velice jednoduchým jazykem pro dotazy, což některé vede i k tvrzení, že žádný jazyk pro dotazy nemá [10]. Rovněž podpora datových typů je oproti jiným NoSQL databázím značně omezená. Na druhé straně ale patří mnohdy mezi nesmírné výhody podpora *ACID*[5], přičemž je zajištěno, že každá operace v Redisu je atomická. Řadí se tak někde mezi jednoduché

[3] Viz poznámka 2

[4] Viz poznámka 2

[5] ACID – zkratka ze slov Atomicity, Consistency, Isolation a Durability, tedy zajištění Atomicity, Konzistence, Izolace a Trvanlivosti

systémy vyrovnávací paměti jako *Memcache* a databáze bohaté na funkce, jako například *MongoDB*[6].

2.4.2 Komunikace

Redis komunikuje s klienty i ostatními redis instancemi pomocí protokolu TCP. Protože komunikace není nijak šifrována, jsou při reálném nasazení na více oddělených strojích používány SSH nebo VPN tunely, aby nemohlo dojít k odposlouchávání komunikace.

2.4.3 Využití v projektu

Protože je Jabber bot nenáročný na množství ukládaných dat a zároveň je očekávána velmi rychlá odezva, byl jako úložiště dat vybrán právě Redis. Využívány jsou jak základní datové struktury, tak i funkcionalita publish – subscribe, která je využívána zejména pro informování modulů při různých událostech jako jsou přijetí zprávy, odpovědi, informace o změně stavu, apod. Protože je návrh Jabber bota modulární, je možno využít informování o událostech i v zásuvných modulech.

2.5 Gevent

Gevent je síťová knihovna pro jazyk Python založená na koprogramech. Poskytuje vysokoúrovňové synchronní API založené na knihovně *libevent*. Tím, že je tato knihovna založena na koprogramech, je zajištěna mnohem větší rychlost síťových operací, protože odpadá režie navíc, jako je tomu při použití vláken, nebo podprocesů. Tato knihovna zároveň umožňuje „záplatování“ některých funkcí ze standardní knihovny jazyka Python (tzv. „monkey-patching“), což umožňuje její použití i u některých stávajících projektů bez nadměrného úsilí pro optimalizace. Toto je zajištěno pomocí nahrazení částí standardní knihovny s kompatibilním protějškem z knihovny *gevent* [1].

2.5.1 Greenlet

Gevent je založen na jednoduchých pseudovláknech nazývaných *Greenlet*. Mezi jednotlivými „pseudovlákny“ je za dobu běhu programu přepínáno buď automaticky (po dokončení činnosti), nebo manuálně (při použití některé z metod pro přepnutí kontextu). Díky tomuto zároveň odpadají problémy se synchronizací jednotlivých vláken či procesů, protože se veškerá aktivita odehrává v jednom vlákně či procesu, pokud jich samozřejmě úmyslně nespustíme více. Každý *Greenlet* tak má přístup ke globálním proměnným a jiným částem programu, které může i upravovat bez rizika souběhu (race-condition). V praxi to potom vypadá tak, že můžeme psát klasický synchronní kód bez ohledu na pozdější úpravy a teprve poté přidat direktivy, které umožní souběžný běh různých částí programu.

Hlavní nevýhodou tohoto přístupu je, že žádný z „Greenletů“ by neměl být příliš náročný na výpočetní výkon, protože by svým výpočtem blokoval ostatní. Tuto situaci je možné řešit spouštěním výpočetně náročných částí programu v podprocesech. K blokování ale může dojít i nechtěně (např. při zacyklení části programu), proto je většina funkcí uvnitř knihovny opatřena dodatečným argumentem *timeout*, který slouží k přepnutí na další *Greenlet*, pokud by operace trvala příliš dlouho. Nutno podotknout, že výpočetně náročné operace není

[6] `http://www.mongodb.org/`

možné (až na výjimky při použití speciálního interpretu) v jazyce Python řešit ani pomocí vláken, kvůli přítomnosti GIL[7].

Knihovna rovněž disponuje vlastní DNS knihovnou i TCP, SSL, HTTP či WSGI[8] serverem. Jsou obsaženy i některé základní synchronizační primitivy jako události, fronta, nebo semafory. Pro omezení počtu souběžně běžících koprogramů je možné použít třídu `Pool`.

2.5.2 Využití v projektu

Protože nejvíce omezující a nejpomalejší částí programu je síťová komunikace a samotný program má minimální nároky na výpočetní výkon, jevilo se použití této knihovny jako optimální. Samostatně blokující části aplikace, jako je komunikace s XMPP serverem, nebo naslouchání na TCP portu pro příjem příkazů takto mohou snadno běžet společně.

[7] Global Interpreter Lock – globální zámek interpretu
[8] Web Server Gateway Interface – jednoduché rozhraní pro komunikaci mezi webovým serverem a aplikací

Kapitola 3

XMPP protokol

Extensible Messaging and Presence Protocol (XMPP), známý také jako Jabber je otevřený protokol na bázi jazyka XML pro komunikaci téměř v reálném čase[1], dále pro předávání informací o stavu uživatele a pro služby založené na principu dotaz – odpověď. [13]

Protože je tento protokol poměrně mladým, má výhodu v tom, že při návrhu mohli jeho vývojáři přebírat užitečnou funkcionalitu z jiných protokolů. Jádro tohoto protokolu není vázáno na žádnou konkrétní síťovou architekturu, v praxi se ale využívá TCP protokol jak pro komunikaci XMPP klienta se serverem, tak pro komunikaci jednotlivých serverů mezi sebou [17].

3.1 Identifikace uživatelů

Pro identifikaci uživatelů používá XMPP protokol schéma podobné emailu. Unikátní identifikátor uživatele se nazývá Jabber Identifier (JID), přičemž tento název je v současnosti používán spíše z historických důvodů. Všechny JID jsou ve tvaru `[uzel]@[doména]/[zdroj]`, přičemž *uzel* je nejčastěji uživatelské jméno, *doména* je plně kvalifikované doménové jméno a *zdroj* je textový řetězec, identifikující jednoho klienta. Příkladem JID může být účet `marie.example@jabber.cz`, který je zároveň v projektu použit jako testovací.

3.1.1 Internacionalizace

V JID je možno použít libovolné unicode znaky, což jej odlišuje například od emailu. Emailová adresa `václav@novak.cz` by platná nebyla, kdežto Jabber ID v tomto tvaru je použitelné a zcela validní. Nastupuje zde však otázka vhodnosti využití tohoto JID z důvodu možnosti napsat jej např. na anglické klávesnici.

3.1.2 Zdroje

Z výše uvedených informací je patrné, že XMPP protokol umožňuje připojení více klientů s jedním JID. Každý z těchto klientů má vlastní „zdroj“, což je textový řetězec, který jej identifikuje. V praxi se nejčastěji používá název klientského programu plus uživatelský popis. Zdrojem ovšem může být také jméno uživatele v místnosti pro chat mezi více uživateli. Název zdroje zajišťuje klientský program, záleží tedy pouze na jeho implementaci,

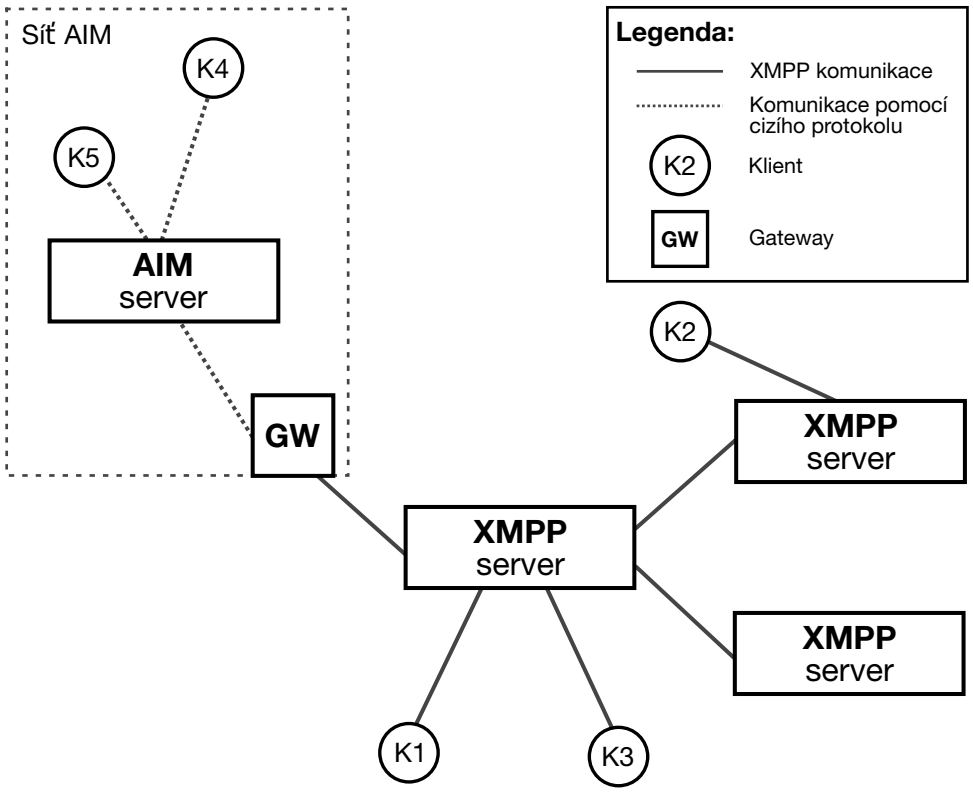
[1] Komunikace je zpožděna kvůli latenci síťového připojení a dalším faktorům – např. výkonu procesoru a síťových zařízení

jestli si název zdroje bude moci nastavit uživatel, nebo jestli bude přiřazen automaticky vygenerovaný název.

3.1.3 URI

RFC 5122 [14] definuje schéma, které je možné použít pro komunikaci s určitým JID prostřednictvím XMPP protokolu. Toto schéma je ve tvaru `xmpp:example@jabber.cz` a existují rozšíření, která umožňují přidat příkazy pro vyvolání určité akce. Příkladem využití pro tyto identifikátory mohou být například webové stránky, kde by jinak Jabber ID mohlo být nerozlišitelné například od emailu, nebo různé databáze, kde se uvádí kontakt. Použití URI[2] je určeno pouze pro externí aplikace, protože ve specifikaci XMPP protokolu se výslovně uvádí, že schéma nesmí být obsaženo pro adresaci v rámci protokolu [14].

3.2 Architektura



Obrázek 3.1: Schéma architektury XMPP protokolu

AIM je zkratka pro AOL Instant Messenger, program pro komunikaci v reálném čase a informaci o stavu uživatele, využívající proprietární protokol *OSCAR*.

XMPP používá decentralizovanou klient – server architekturu, podobnou webu, nebo emailu. Tato architektura má výhodu v úplném oddělení serveru a klienta, tudíž je možné aby se vývojáři klientů soustředili na uživatelské prostředí, dodatečné poskytované funkce, apod., a vývojáři serverů zase na škálování a zajištění dostupnosti. Přitom klient je nezávislý

[2] **Uniform Resource Identifier** – jednotný identifikátor zdroje

na konkrétní aplikaci implementující funkci serveru a server je zase nezávislý na použitém klientovi. [17]

3.2.1 Server

Server je základem pro veškerou komunikaci mezi uživateli. Jeho úkolem je udržovat spojení s autorizovanými klienty, servery a ostatními entitami podílejícími se na komunikaci, ve formě XML proudu. Zajišťuje také směrování a zasílání zpráv (tzv. *XML sloky*[3]) mezi těmito entitami. Po navázání spojení je možné zasílat neomezený počet XML slok v obou směrech.

Server také často zajišťuje další funkce a služby, například ukládání „rosteru“[4], zprostředkování chatu mezi více uživateli, aj. V tomto případě server zpracovává zprávy samostatně a již je dále nepřeposílá.

Mnoho provozovatelů existujících XMPP serverů umožňuje využívání svých služeb pro uživatele zcela zdarma, zpoplatněny bývají často pouze pokročilé funkce, jako je ukládání historie zpráv a následný přístup k nim pomocí webového rozhraní nebo transportní služby, které umožňují komunikaci s uživateli na jiných protokolech (AIM, ICQ, ...) prostřednictvím protokolu XMPP. Transportní služby fungují pro klienta zcela transparentně a takto přenesené účty se zobrazují jako standardní kontakty. Tyto služby nejčastěji zprostředkovává gateway (GW), která může, ale nemusí být oddělená od XMPP serveru.

XML proud

XML proud slouží jako tzv. kontejner pro výměnu informací mezi libovolnými dvěma entitami na síti [13]. Pro zjednodušení je možné představit si jej jako nekonečný XML dokument, který se postupně vytváří tím, jak probíhá komunikace. Jeho obsahem mohou být jak zprávy pro udržování aktuálního spojení, tak i jednotlivé XML sloky, které mohou obsahovat zprávy, informace o stavu uživatele, oznámení o změně stavu (uživatel píše zprávu), apod.

Navázání spojení

Pokud chceme navázat spojení se serverem, klient otevře dlouhotrvající TCP spojení pro komunikaci se serverem. Po dokončení vyjednávání otevře server taktéž jedno dlouhotrvající TCP spojení pro komunikaci s klientem. Každé spojení se serverem tudíž využívá dvě TCP spojení (jedno v každém směru).

3.2.2 Klient

Většina klientů se pouze připojuje k serveru, který zprostředkovává samotnou funkcionalitu a klient zde slouží pouze jako uživatelské rozhraní. Možnosti jednotlivých klientů se poměrně výrazně liší, od jednoduchých textových klientů až po obsáhlé klienty s grafickým rozhraním a celou škálou funkcí.

Pro využívání služeb serveru je potřeba registrovaný uživatelský účet. Tato registrace probíhá nejčastěji přímo prostřednictvím klienta, který zasílá požadavek na server.

[3] **XML Stanza** (sloka) slouží jako obálka pro informace zasílané mezi entitami podílejícími se na komunikaci

[4] Roster – seznam kontaktů

3.2.3 Gateway

Gateway, česky také „brána“ poskytuje transportní služby popisované v sekci 3.2.1. Primární funkcí brány je překlad XMPP zpráv do a z protokolu používaného jinou službou. Příkladem mohou být brány pro e-mail, SMS, nebo jiné IM protokoly. Příklad brány pro IM protokol AIM je na obrázku 3.1.

3.3 Bezpečnost

XMPP server podporuje mechanismy k zajištění bezpečnosti komunikace, tj. k zamezení odposlouchávání, nebo změně zasílaných zpráv. Šifrování využívá protokol TLS[5] spolu s rozšířením STARTTLS[6], podobně jako je tomu u protokolů IMAP nebo POP3. [13] Přestože je využití šifrování komunikace doporučeno, není pro komunikaci nutné, pokud není vynuceno na straně klienta, nebo serveru. Vynucení šifrování na straně klienta je však dnes z důvodu bezpečnosti výchozím nastavením většiny klientů. [9]

Pro autorizaci uživatelů je využit framework SASL[7]. Pokud si strana zahajující spojení vyžádá použití TLS, je pro zajištění bezpečnosti autentizace vynuceno, aby SASL autorizace proběhla až po dokončení TLS inicializace (handshake). Toto chování je pro TLS protokol standardní, tj. komunikace na aplikační vrstvě, v tomto případě SASL autentizace, je povolena až po dokončení inicializace. [3]

[5] **TLS** (Transport Layer Security) – kryptografický protokol poskytující možnost zabezpečené komunikace na internetu

[6] STARTTLS – rozšíření podporující povýšení nešifrované komunikace na šifrovanou [15]

[7] Simple Authentication and Security Layer – framework pro ověřování klientů na serverech podporující různé ověřovací mechanismy

Kapitola 4

Návrh aplikace

V první části této kapitoly jsou podrobně popsány podporované příkazy a zasílané zprávy. Druhá část kapitoly se věnuje popisu objektového návrhu aplikace a ukazuje možnosti rozšíření, včetně příkladů.

4.1 Množina příkazů a zasílaných zpráv

4.1.1 Příkazy přijímané prostřednictvím HTTP serveru

Příkazy popsané v této podkapitole mají ve svémo popisu uvedenu vždy URL adresu spolu s typem HTTP požadavku. Objekt, který se může vyskytovat pod nimi slouží jako POST data, která můžou být zaslána v podobě serializovaného JSON objektu, nebo jako data ve formátu `application/x-www-form-urlencoded`, což je preferovanou možností. Pro druhou možnost existují knihovny v naprosté většině programovacích jazyků.

Zpráva

Slouží pro odeslání obyčejné zprávy (XMPP typ `chat`) na adresu příjemce.

`text`	text zprávy
`to`	příjemce zprávy

```
POST /message/
{
    "text": "Hello world!",
    "to": "viktor@stiskala.cz",
}
```

Otázka

Odeslání zprávy (identifikovatelné pomocí jednoznačného ID), na kterou má uživatel možnost odpovědět.

`text`	text zprávy
`id`	identifikátor zprávy sloužící pro spárování s odpovědí
`to`	příjemce zprávy
`postback_url`	URL adresa, na kterou bude odeslána odpověď pomocí metody POST
`expire_on_offline`	Pokud je nastaven tento příznak, zpráva expiruje, jakmile uživatel přejde do stavu *offline*. Pokud je otázka zaslána v době, kdy je uživatel *offline*, tak expiruje okamžitě.
`only_if_status`	Otázka bude zaslána pouze v případě, že stav uživatele odpovídá jednomu ze specifikovaných. Více hodnot je možno oddělit čárkou.

```
POST /question/
{
    "text": "John sent you a request in bug #344, you can reply right here",
    "id": "bug_344",
    "to": "viktor@stiskala.cz",
    "postback_url": "http://example.com/post/bugtracker/344"
    "expire_on_offline": "true"
    "only_if_status": "online,away"
}
```

Monitorování chat místnosti

Příkaz způsobí, že Jabber bot vstoupí do místnosti a veškeré příchozí zprávy bude přeposílat na uvedenou `postback_url`. Příklad využití tohoto příkazu je popsán v sekci 7.2.

`room`	identifikátor chat místnosti
`nickname`	jméno, pod kterým bude Jabber bot v místnosti vystupovat
`password`	volitelné heslo pro vstup do místnosti
`postback_url`	URL adresa, na kterou budou zasílány všechny zaznamenané zprávy z místnosti

```
POST /monitor_chatroom/
{
    "room": "test-room@chat.example.com",
    "nickname": "Jabber bot",
    "password": "let me in",
    "postback_url": "http://example.com/handle-chatroom-messages/"
}
```

Souvisejícími příkazy jsou `cancel_monitoring` a `cancel_all_monitoring`, které zruší nastavené monitorování pro vybranou místnost, případně pro všechny místnosti.

Funkce `cancel_monitoring` přijímá jediný parametr (`room`), kterým je identifikátor místnosti pro kterou se má monitorování zrušit.

Odpověď

V tomto případě se jedná o odchozí požadavek ze serveru: odpověď na otázku zaslanou pomocí typu zprávy `question`. Zpráva je zasílána HTTP metodou `POST` na `postback_url` definovanou v otázce. Objekt níže specifikuje formát zasílané odpovědi.

`text`: text odpovědi
`id`: identifikátor zprávy
`from`: odesílatel zprávy
`answered_after`: doba v sekundách, po které uživatel odpověděl, od doby odeslání zprávy

```
{
    "type": "answer",
    "id": "bug_344",
    "text": "Hey John, this bug was caused by temporary failure.",
    "from": "viktor@stiskala.cz",
    "answered_after": "3.456653"
}
```

Textová zpráva z chat místnosti

Jedná se rovněž o odchozí požadavek ze serveru, který se provede vždy, když je přijata textová zpráva ve skupinovém chatu, pro který bylo nastaveno monitorování. Vlastnosti níže popisují obsah zprávy zasílané na `postback_url` metodou POST.

`from`	jméno uživatele, který poslal zprávu do místnosti
`room`	identifikátor místnosti, do které zpráva přišla
`text`	text zprávy
`received`	datum a čas, kdy byla zpráva přijata ve formátu ISO 8601

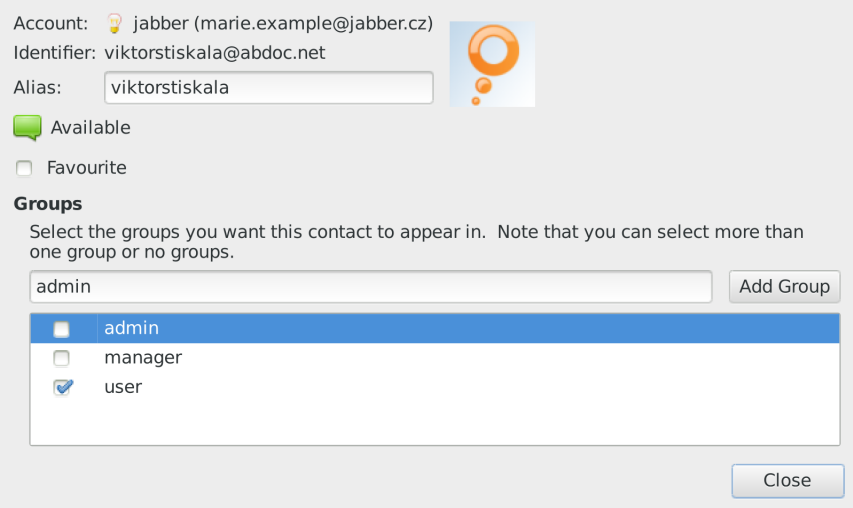
```
{
    "from": "Petr",
    "room": "testroom@chat.example.com",
    "text": "Message logging test",
    "received": "2013-05-15T00:03:54.875140"
}
```

4.1.2 Příkazy zasílané prostřednictvím XMPP protokolu

Tato sekce popisuje příkazy podporované aplikací již v základu. Cílem aplikace není poskytnout co největší množinu příkazů, ale rozhraní pro jejich snadné přidávání, proto slouží většina integrovaných příkazů pouze pro demonstraci využití. Vlastní příkazy je možné přidávat jednoduchým způsobem, jak je popsáno v sekci 4.1.3.

Každý z příkazů popsaných v této sekci je možné spouštět pouze tehdy, pokud má uživatel k tomu potřebná oprávnění. Oprávnění jsou detekována podle skupiny, do které uživatel spadá, ve třech úrovních: *user*, *manager* a *admin*, přičemž uživatelé spadající do vyšší skupiny mají zároveň všechna oprávnění ze skupiny nižší. Nastavení těchto oprávnění je možné s použitím libovolného XMPP klienta, kde stačí uživatele „přetáhnout“ do správné skupiny. Jelikož protokol umožňuje, aby byl uživatel připojen vícekrát, není potřeba pro

změnu oprávnění a tudíž přihlašování v klientském programu Jabber bot vypínat. Ukázka zařazení do skupin je vidět na obrázku 4.1.



Obrázek 4.1: Zařazení uživatelů do skupin v komunikačním programu Empathy

Seznam příkazů

`join_room` – způsobí připojení Jabber bota do chat místnosti, přijímá 3 parametry: `room` a volitelné `nickname` a `password`.
`users_in_roster` – vypíše seznam uživatelů, které má Jabber bot v seznamu kontaktů (*rosteru*).
`user_status` – vypíše aktuální stav všech uživatelů, případně při zadaných JID oddělených mezerou stav těchto uživatelů. Vypisuje *offline* i pro neautorizované uživatele, stejně jako IM klienti.
`server_uptime` – zašle zpět informaci o době běhu a vytížení serveru, na kterém bot běží (výsledek volání programu `uptime`).
`bot_uptime` – zašle zpět informaci o době běhu Jabber bota. Vyžaduje minimálně oprávnění úrovně *manager*.

4.1.3 Dekorátor `bot_command`

Jabber bot podporuje kromě vzdáleného ovládání přes externí protokol také spouštění příkazů po zaslání zprávy přes protokol XMPP, který se jinak v projektu používá pro zasílaní zpráv, nebo odpovědí na otázku. Kromě integrovaných příkazů uvedených v sekci 4.1.2 je možné program rozšířit o další s využitím připraveného dekorátoru.

Mezi základní rozšíření patří přidání nového příkazu, na který bude Jabber bot reagovat. K tomu se využívá dekorátor `bot_command`, který zároveň jako volitelný argument přijímá třídu pro zpracování parametrů zadávaných k tomuto příkazu.

Chceme-li například přidat nový příkaz `weekday`, který zobrazuje den v týdnu podle data (které může být předáno jako volitelný parametr), můžeme si napsat vlastní třídu pro zpracování parametrů využívající funkci `strptime` vytvářející datum z textového řetězce.

Kapitola 5

Implementace

Tato kapitola se věnuje popisu implementačních detailů Jabber bota. První část kapitoly popisuje rozdělení aplikace do funkčních celků, zbytek kapitoly je potom věnován detailnějšímu popisu architektury aplikace.

Celý projekt je možno rozdělit do dvou funkčních celků. První z nich obstarává komunikaci prostřednictvím XMPP protokolu, druhým funkčním celkem jsou tzv. „listeners“, kteří komunikují prostřednictvím jiného protokolu, ale umožňují s aplikací přímo interagovat.

5.1 XMPP komunikace

Pokud je aplikaci zaslána zpráva prostřednictvím XMPP protokolu, zpracuje se podle diagramu 5.1 na straně 18. Hlavním faktorem ovlivňujícím způsob zpracování zprávy je skutečnost, jestli byla uživateli zaslána otázka. Pokud ano, přeskakuje se logika pro zpracování příkazu a zpráva se vyhodnocuje jako odpověď na otázku.

5.1.1 Vyhodnocení příchozí zprávy jako „příkaz“

Pokud uživatel zašle zprávu přímo na JID, které využívá Jabber bot, interpretuje se tato zpráva jako příkaz. Příkazy mohou mít volitelně nastavený tzv. *prefix*, tj. znak nebo znaky, kterými musí zpráva začínat, aby byla jako příkaz interpretována. Tento prefix se nastavuje při inicializaci třídy `XMPPBot`, respektive `EventBot` (která je přímým potomkem) a je nastavitelný odděleně pro zprávy přijaté v chat místnosti a pro zprávy zaslané přímo bot uživateli.

Ve výchozím stavu není nastaven žádný prefix pro přímé zprávy a znak „!“ pro zprávy v chat místnosti, kde může být přítomno více uživatelů a mohlo by tak docházet k nechtěnému vyhodnocení zprávy jako příkazu. Pro rozlišení se používá typ zprávy (*chat*, *normal* a *groupchat*), který XMPP protokol specifikuje.

I přes relativně matoucí název jsou zprávy přijaté v chat místnosti typu *groupchat* a typy *normal* nebo *chat* označují typy zpráv používané při komunikaci jeden na jednoho. Aby aplikace odpovídala XMPP specifikaci jsou zprávy, které neobsahují atribut *type* považovány za typ *normal*.

Zpracování příkazu

Pro definici příkazu se používá dekorátor `bot_command`, který nastavuje „dekorované“ metodě atribut (příznak) `_bot_command` na `True`. Když má zpráva správný typ a obsahuje

odpovídající prefix, je tento prefix nejprve odstraněn a poté je vyhledána metoda s odpovídajícím názvem. Protože může být dekorátoru explicitně předán parametr `name`, probíhá vyhledání metody s použitím mapovacího slovníku. Pokud má tato metoda nastaven příznak `_bot_command`, jsou ji předány parametry, které uživatel poslal spolu s příkazem a v odděleném koprogramu je spuštěno vyhodnocení této metody. Jestliže metoda po vyhodnocení skončí s návratovou hodnotou jinou než `None`, je tato hodnota zaslána zpět uživateli. Tohoto přístupu se využívá pro zaslání výsledku příkazu zpět uživateli. Jelikož zpracování probíhá asynchronně, je však při zaslání více příkazů najednou možné, že odpovědi nepřijdou zpět ve stejném pořadí. Proces zpracování je zjednodušeně popsán algoritmem 5.1.

Pro vynucení čekání po určitou dobu se využívá metoda `gevent.sleep(time)`, která neblokujícím způsobem zajistí čekání pomocí přepínání kontextu. Využití metody `time.sleep` je možné pouze pokud byl proveden tzv. „monkey-patching“ standardní knihovny (viz sekce 2.5), jinak by docházelo k blokování celého programu.

```
if message.type in ("normal", "chat", "groupchat") then
    if message.body starts with prefix then
        remove prefix;
        command, params = split(message.body, " ");
        find method with command name;
        if method has attribute bot_command then
            parse arguments;
            process_command(command, parsed_args);
        end
    end
end
```

Algoritmus 5.1: Pseudokód popisující zpracování příkazu

Zpracování parametrů, které mohou být zaslány spolu s příkazem zajišťuje třída, kterou je možné specifikovat v dekorátoru `bot_command`. Pokud není explicitně uvedeno jinak, použije se výchozí třída `SepArgsParser`, která rozdělí parametry do datového typu `list` podle bílých znaků.

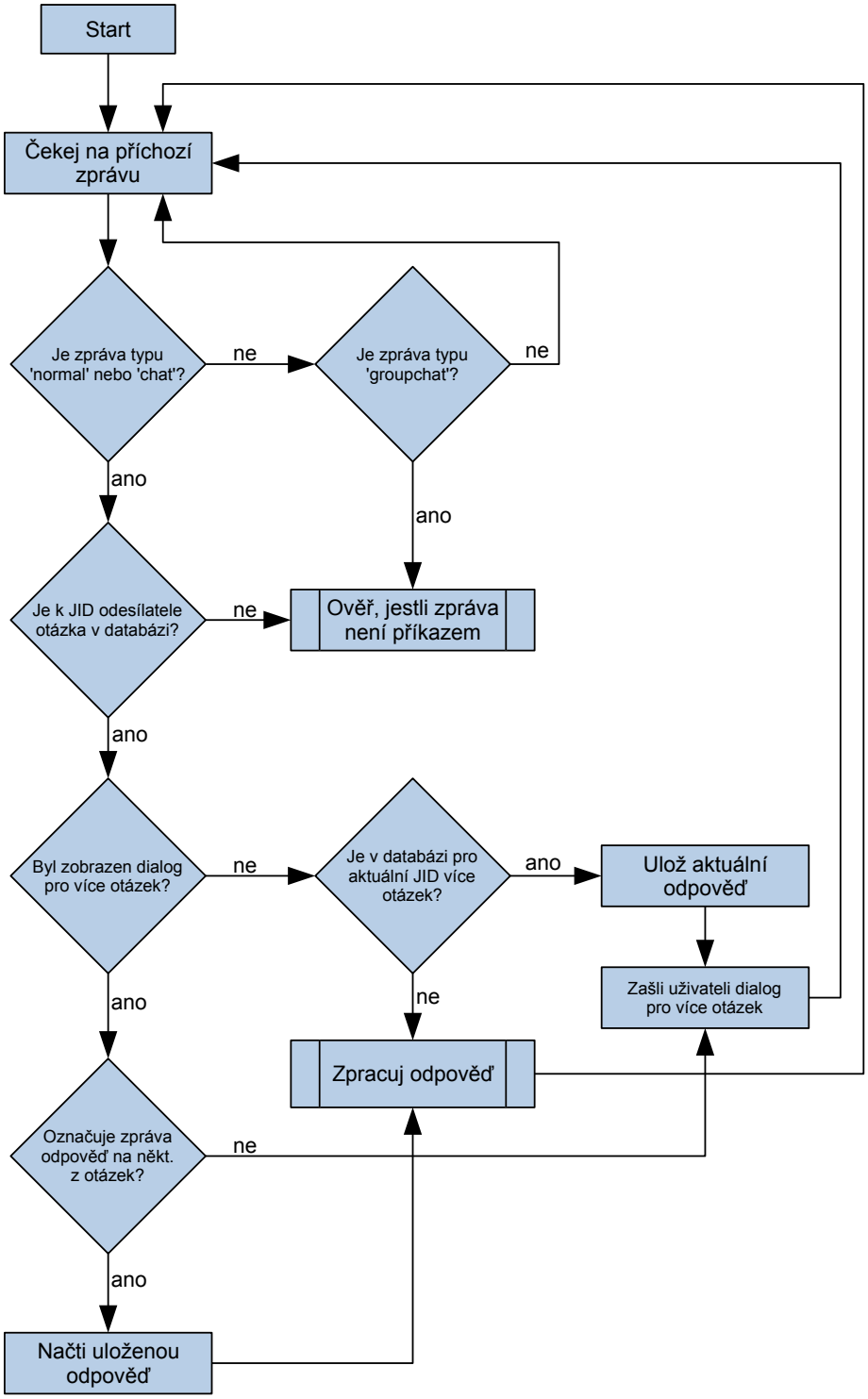
Například při zaslání příkazu `uptime server1.example.com newserver.example.org` obdrží metoda zajišťující zpracování tohoto příkazu argumenty ve tvaru seznamu zobrazeném na následujícím příkladu:

```
["server1.example.com", "newserver.example.org"]
```

Příklad 5.1: Seznam (list) argumentů příkazu

5.1.2 Vyhodnocení příchozí zprávy jako „odpověď na otázku“

Příchozí zpráva je vyhodnocena jako odpověď na otázku v případě, že je pro Jabber ID uživatele, který zprávu odeslal v databázi uložena otázka. Do procesu ale dále vstupují další proměnné, které jsou vidět na vývojovém diagramu 5.1, například dialog pro více otázek.



Obrázek 5.1: Vývojový diagram znázorňující zpracování příchozí XMPP zprávy

Diagram je pro přehlednost zjednodušen, neobsahuje tedy kontrolu, zda nevypršela platnost otázky, ověření stavu uživatele, aj. Tyto vlastnosti jsou popsány dále v kapitole.

5.2 Listeners

Listener (česky „posluchač“) je část aplikace, která čeká, až nastane určitá událost, aby pro tuto událost mohla vyvolat obslužnou rutinu. Jak již z předchozí věty vyplývá, část aplikace poskytující programovatelné rozhraní pro jednotlivé posluchače je založena na systému událostí. Tento systém je implementován z toho důvodu, že jednak může k událostem docházet asynchronně (např. přijetí odpovědi na otázku), ale také je tímto způsobem snadné zaregistrovat více posluchačů na stejnou událost. Tím je umožněno pozdější rozšíření aplikace bez zbytečně vynaloženého úsilí navíc.

5.2.1 Systém událostí

Systém událostí je navržen v třídě `EventBot`, přičemž události jsou implementovány jako typ `defaultdict` z balíku `collections`, který je součástí standardní knihovny jazyka Python. Tento typ se chová podobně jako slovník, vytvoří však klíč s předem nastavenou výchozí hodnotou pokaždé, kdy se přistupuje k neexistujícímu klíči. V aplikaci je pro události použit `collections.defaultdict(list)`, který jak je pravděpodobně již patrné, vytvoří při prvním přístupu k neexistujícímu klíči typ seznam. Toho využívá metoda `register_callback`, která tak nemusí mít předem omezenou množinu názvů obslužných rutin.

Aplikační rozhraní pro registraci událostí

Ve třídě `EventBot` jsou v základu implementovány dvě události – `answer_received` a `question_expired`. První z těchto událostí je vyvolána pokaždé, kdy bot přijme odpověď na otázku, druhá z nich je vyvolána při expiraci otázky.

Událost expirace nenastává ihned po vypršení otázky, ale pouze když přijde odpověď na otázku, které již vypršela platnosti, nebo v situaci, kdy je nastaven příznak `expire_on_offline`. Ve druhém případě dochází k expiraci při změně stavu uživatele, nebo pokud je otázka odeslána a uživatel se již nachází ve stavu *offline* (expiruje okamžitě).

Registrace obslužné rutiny i vyvolání události zpracovává třída `EventBot`. Všechny ostatní části programu, které potřebují být informovány o tom, že nastala určitá událost mohou použít metodu `EventBot.register_callback(self, event, callback)`, kde `event` je název události a `callback` je obslužná rutina.

Jakmile nastane určitá událost, je vyvolána metoda `EventBot._trigger_event(self, event_name, data)`, která spustí veškeré registrované obslužné rutiny v oddělených koprogramech prostřednictvím Greenletu (viz. 2.5.1). Obslužné rutině jsou navíc předána dodatečná data: například v případě události `answer_received` je to n-tice dvou slovníků, z nichž první obsahuje původní otázku a druhý příchozí odpověď uživatele. Zpracování těchto dat je ponecháno čistě na obslužné rutině.

Implementování vlastní události ve třídě, která využívá `EventBot` jako svého předka je možné pouze s pomocí metod `register_callback` a `_trigger_event`.

Implementační detaily

Události jsou implementovány jako slovník, kde klíč je název události a hodnotou je pole (seznam) obsahující obslužné rutiny pro tuto konkrétní událost. Slovník obsluhy událostí pro Jabber bot s aktivním modulem HTTP Listener je vidět na příkladu 5.2. Vyvolání událostí je prováděno postupně s využitím pseudovláken (Greenlet), tudíž není vhodné, aby obslužné

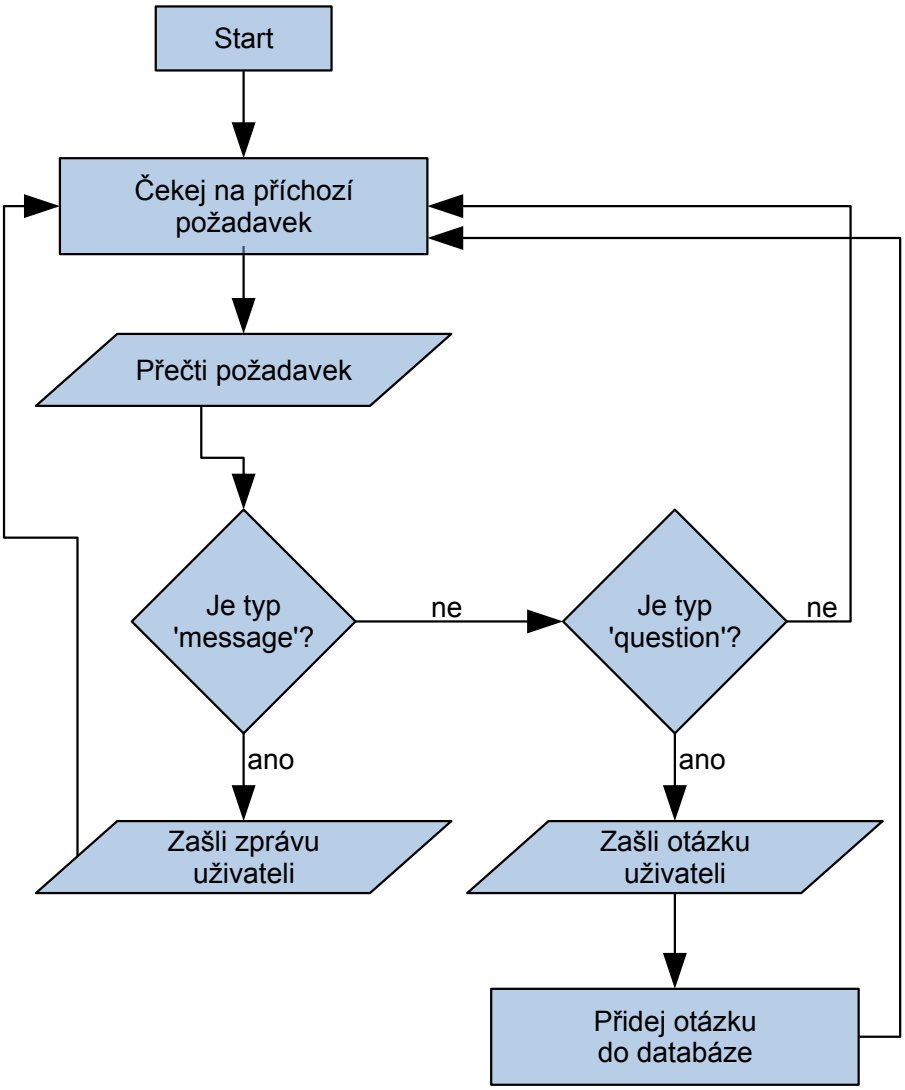
rutiny obsahovaly výpočetně náročný kód. Využití blokujícího kódu, který využívá síťovou komunikaci zde naopak na škodu není, protože Greenlet se postará o asynchronní běh.

```
{
    'answer_received': [<bound method HttpListener.answer_received
    of <HttpListener at 0xcbd410>>],
    'question_expired': [<bound method HttpListener.handle_expired_question
    of <HttpListener at 0x205d410>>]
}
```

Příklad 5.2: Struktura slovníku pro obsluhu událostí

5.2.2 HTTP Listener

HTTP Listener je část programu, která naslouchá na specifikovaném portu a čeká na příchozí příkazy. Zpracování příkazu je implementováno asynchronně, tudíž může program v jednu dobu obsloužit více požadavků i od různých uživatelů. Průběh zpracování příchozího požadavku je vidět na vývojovém diagramu níže (obrázek 5.2).



Obrázek 5.2: Vývojový diagram zpracování příkazu přijatého prostřednictvím HTTP

Propojení s rozhraním pro registraci událostí

Modul HTTP Listener využívá především událost `answer_received`, aby ji mohl dále zpracovat a odeslat metodou POST na `postback_url` uvedenou v původní otázce. Pokud původní otázka tuto url neobsahuje, je přijatá odpověď ignorována, čímž je zajištěno, že Jabber bot může podporovat další moduly pro zpracování odpovědi, aniž by to nějak ovlivnilo funkčnost HTTP listeneru. Při odesílání odpovědi na `postback_url` jsou parametry odpovědi před samotným odesláním serializovány do formátu JSON.

Oprávnění a autentizace

HTTP Listener nemá, narozdíl od přímého zasílaní zpráv, implementovány úrovně oprávnění. Nastavení autentizace je možné více způsoby, pomocí externích aplikací. Protože se jedná o HTTP protokol, je ideální využít k těmto účelům reverzní proxy. Nastavení reverzní proxy na webovém serveru Nginx je podrobně popsáno v příloze C.

Pomocí obdobného způsobu je možné nastavit rovněž různé úrovně oprávnění. Každý z příkazů, které aplikace přijímá prostřednictvím HTTP modulu má oddělenou adresu, čímž je umožněno nastavení oprávnění pro každý z příkazů zvlášť. Stačí v nastavení proxy serveru filtrovat příchozí požadavky podle URL adresy a na každý z nich vyžadovat jinou úroveň oprávnění uživatelů, případně filtrovat pouze pomocí jména autentizovaného uživatele. Nastavení oprávnění uživatelů na proxy serveru je možno řešit s využitím skupin a dosáhnout tak stejného způsobu nastavení filtrování, jako je tomu u příkazů přijímaných Jabber botem popsaných v sekci 4.1.2.

5.3 Problém více otázek

Při zasílání otázek uživateli může dojít k problému, kdy je uživateli zasláno více otázek za sebou dříve, než stihne odpovědět na předchozí. Aby tento problém nastal navíc není nutné zaslat několik otázek rychle po sobě, protože uživateli může trvat i několik minut, než na předchozí otázku odpoví, případně na ni nemusí odpovědět vůbec (např. ji může přehlédnout). V této situaci se namísto zaznamenání odpovědi pošle uživateli dialog, který se jej ptá, na kterou z otázek právě odpověděl a teprve až uživatel zašle zpět číslo identifikující otázku, je jeho předchozí odpověď vyhodnocena.

V průběhu vývoje jsem se snažil navrhnout několik metod, jak problém více otázek řešit, žádná z nich však nebyla bezchybná, proto jsem se rozhodl zůstat u jednoznačné, avšak méně uživatelsky příjemné metody – zaslání dialogu. Předchozí metody spočívaly v odhadování na kterou z otázek uživatel odpověděl podle proměnných jako čas, změny stavu uživatele, apod.

Podle změn stavů uživatele v čase je možné odhadnout jeho aktivitu, protože mnoho klientů nastavuje tzv. „auto away“ stav, když uživatel odejde od PC – nevykazuje žádnou aktivitu. Společně s tímto faktem je možno brát v úvahu také další rozšíření, které mnozí IM klienti implementují, například XEP-0085, které specifikuje zprávy o stavu chatu jako „uživatel píše zprávu“, „uživatel se aktivně podílí na chatu“, „uživatel aktivně ukončil svou účast na chatu“, apod. Zde je však problém v nejednoznačnosti specifikace a tím pádem v rozdílné implementaci v různých IM klientech. Příkladem může být knihovna *libpurple* (využívaná IM klienty Pidgin a Adium), která při zavření okna konverzace zasílá stav *gone*, kdežto přímo ve specifikaci o tomto využití není zmínka. Naopak je zde uveden příklad použití vysvětlený jako „uživatel neprovedl žádnou interakci s rozhraním pro rozhovor, systémem, nebo zařízením po poměrně dlouhou dobu – např. 10 minut“ [16].

Metody popsané v předchozím odstavci umožňují do jisté míry predikovat, na kterou otázku uživatel odpovídá, ale jejich správnost nikdy nebude stoprocentní. Po vyladění určitých parametrů by pravděpodobně bylo možné použít tyto metody v produkčním prostředí, nicméně použití metod v této práci by mohlo zavádět nejednoznačnosti do jiných systémů, které by byly na této aplikaci závislé. V sekci 8.1 na straně 30 je popsáno, jakým způsobem bylo testováno nastavení parametrů tak, aby se dosahovalo lepších výsledků predikce.

5.4 Pomocné rutiny pro start a konfiguraci aplikace

Jabber bot je implementován jako Python balíček s názvem *marie*, což je kódové označení pro tuto aplikaci. Spolu s aplikací je však distribuován také soubor `start.py`, jehož úkolem je konfigurace a spouštění jak hlavní aplikace, tak i ostatních rozšíření.

V základu je aplikace spouštěna spolu s HTTP listenerem (viz. sekce 5.2.2), který ve výchozím nastavení naslouchá na všech dostupných zařízeních a portu 8088.

Protože bot i *listeners* potřebují pracovat souběžně, je potřeba nejprve všechny komponenty spustit a poté vyvolat čekání, aby aplikace neskončila předčasně. K tomuto účelu obsahuje modul `marie` dekorátor `serve_forever`, jehož použití je vidět na následujícím příkladu.

```
with marie.serve_forever() as m:
    bot = EventBot('username', 'password')
    m.start(bot)

    listener = HttpListener(bot, 8088)
    m.start(listener)
```

Příklad 5.3: Využití dekorátoru `serve_forever`

Dekorátor si interně ukládá jednotlivé komponenty do datového typu `list` a při výstupu z kontextu tohoto dekorátoru vyvolává metodu `joinall` modulu `gevent`, čímž zajistí běh aplikace až do vyvolání přerušení. Metoda `start` na příkladu 5.3 zajišťuje zároveň spuštění a přidání komponenty do pole pro pozdější využití.

Aplikace taky ve výchozím nastavení vypisuje informace a chyby přímo do konzole. Pro tento výpis byl použit modul `logging` ze standardní knihovny jazyka Python, je tedy možné způsob výstupu a úroveň chybových zpráv snadno změnit.

5.5 Možná rozšíření projektu

Součástí zadání této práce je navržení možných rozšíření projektu, které jsou popsány v této podkapitole.

5.5.1 Propojení se školním systémem

Sekce 7.1 na straně 28 popisuje možné využití Jabber bota jako notifikačního systému, kde je uvažováno (mimo jiné) zasílání notifikací ze školního systému. Pro využití této funkcionality je potřeba vytvořit externí aplikaci, která bude v pravidelných intervalech přistupovat ke školnímu systému a kontrolovat změny. Aplikace může použít některou z databází na ukládání aktuálního stavu a při změně některého ze sledovaných parametrů (body, zápočty, nové termíny) použít REST API bota k zasílání notifikací.

Při kombinaci popsaného notifikačního systému pro školní systém a příkazů přijímaných přímo Jabber botem je možné dosáhnout určité automatizace pro přihlašování na nové termíny i v době, kdy ještě nebylo přihlášení oficiálně zahájeno. Stačí využít systémového plánovače, nebo aplikace čekající na pozadí, která v době začátku přihlašování odsimuluje webový prohlížeč a zašle požadavek na přihlášení. Po tomto požadavku je samozřejmě možné zpětně upozornit uživatele, zda se přihlášení zdařilo, či ne.

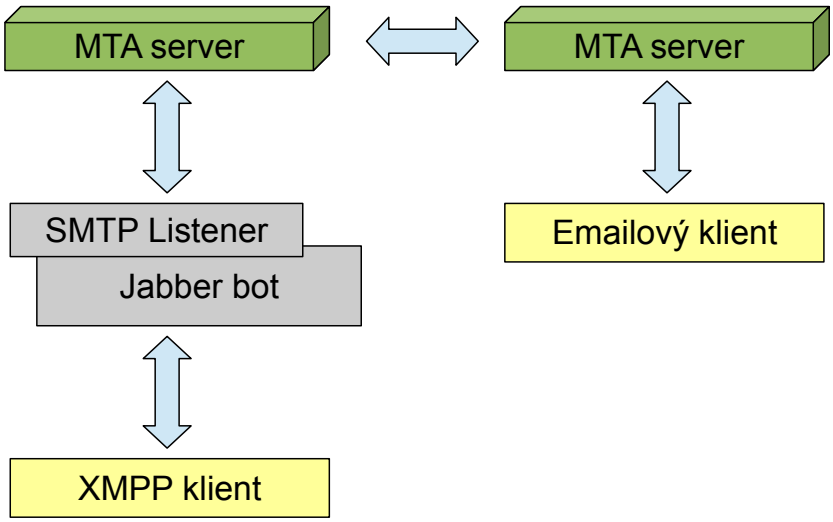
5.5.2 SMTP Listener

Toto rozšíření by zajišťovalo propojení se SMTP[1] protokolem, čímž by umožňovalo přeposílat emaily jako Jabber zprávy. Jak *Listeners* fungují popisuje podkapitola 5.2.

[1] Simple Mail Transfer Protocol – protokol určený pro přenos zpráv elektronické pošty (tzv. „emaily“)

Pro jednodušší implementaci je vhodné propojit listener s některým z existujících „přepravců elektronické pošty“, takzvaným MTA[2]. Většina z existujících MTA dnes podporuje velmi podrobnou konfiguraci, kde by zde popsané řešení nemělo být problémem. MTA potom komunikuje přímo se SMTP listenerem a naopak, takže by byla umožněna i odpověď na příchozí email přímo prostřednictvím XMPP klienta. Bylo by však nutné rozlišit odpoveď na email od jiných příkazů zasílaných přímo botovi, například použitím *prefixu*, nebo právě specifickým příkazem. Schéma komunikace je vidět na obrázku 5.3.

Při použití tohoto rozšíření nemusí být emailová adresa nutně stejná jako Jabber ID, je možné použít jinou doménu či subdoménu, ale i úplně odlišný identifikátor. Příkladem může být Jabber ID ve tvaru `user@example.com` a emailová adresa s automatickým přeposíláním na Jabber ve tvaru `user@xmpp.example.com`.



Obrázek 5.3: Schéma komunikace XMPP klienta s SMTP protokolem

[2]Mail Transfer Agent – software určený pro přenos elektronické pošty mezi servery

Kapitola 6

Uživatelské rozhraní

Jabber bot nemá (kromě konzolového výstupu) žádné uživatelské rozhraní, proto jsou součástí této práce také dvě dodatečné aplikace demonstrující využití na reálném příkladu. Tato kapitola se věnuje popisu obou těchto aplikací.

6.1 XMPP rozšíření pro nástroj Sentry

Sentry je nástroj pro vizualizaci chybových zpráv ve webovém rozhraní. Dokáže ukládat chyby téměř v reálném čase a vzhledem k nízkým nárokům na systémové prostředky je vhodný i pro nasazení v produkčním prostředí. [2]

Sentry dokáže zobrazovat chybové zprávy včetně všech podrobných informací o tom, jak k události došlo a tyto chybové zprávy umí rovněž seskupit podle jejich typu. Nestane se tak, že by kvůli stejné chyby, která nastala vícekrát po sobě bylo celé rozhraní zahlceno informacemi o této jedné chybě, zatímco ostatní méně frekventované chyby by mohly být přehlédnuty. Toto je problém, který se mnohdy stává při „obyčejném“ ukládání chybových zpráv do souboru.

Jednotlivé chyby, resp. skupiny chyb mohou být po opravení v aplikačním rozhraní označeny za opravené, čímž se ze seznamu skryjí. Pokud však určitá chyba nastane opětovně, bude znovu zobrazena celá skupina, včetně všech informací o předchozích výskytech stejného typu chyby, aby byly snadněji dohledatelné souvislosti.

Kromě zobrazení ve webovém rozhraní umí Sentry zasílat informace o chybě na email, nebo jiným způsobem notifikovat vývojáře pomocí dodatečných zásuvných modulů jak od autorů tohoto nástroje, tak i od třetích stran. Nestane se však, že by bylo zasláno obrovské množství zpráv při mnoha výskytech chyby, protože pro každou skupinu se informace odesílá pouze při prvním výskytu, nebo pokud znovu nastane chyba náležící skupině chyb, která již byla označena za vyřešenou. Tímto způsobem je poměrně elegantně vyřešen způsob, jak neinvazivním způsobem informovat vývojáře o chybách, které v monitorované aplikaci nastaly.

Nástroj umožňuje propojení pomocí dodatečných knihoven s většinou dnes používaných programovacích jazyků, včetně některých aplikačních *frameworků* (např. Django[1]). Pokud však pro některý z programovacích jazyků ještě neexistuje knihovna, je možno k propojení použít HTTP API.

[1]Framework napsaný v jazyce Python určený pro vývoj webových aplikací. Více informací o tomto projektu najdete na webové stránce `https://www.djangoproject.com/`

6.1.1 Popis rozšíření

Rozšíření, které je součástí této práce umožňuje odesílat informace o chybě, která právě nastala pomocí protokolu XMPP. Rozšíření se po instalaci zaregistruje do Sentry, kde je možné jej pro určitý projekt povolit. V hlavním nastavení je pro správnou funkcionalitu potřeba nastavit URL pro komunikaci s Jabber botem, včetně volitelné HTTP autentizace. Tu je možno využít v případě, kdy se bot nachází za reverzním proxy serverem se zapnutou autentizací, tak jak je popsáno v sekci 5.2.2 na straně 21.

Po povolení rozšíření pro vybraný projekt je možno v nastavení projektu vyplnit Jabber ID, na které budou notifikace odesílány. Ukázka tohoto rozhraní je na obrázku 6.1. Jakmile je nastavení uloženo, přijde na specifikovaná JID při výskytu chyby zpráva, obsahující krátkou informaci o typu chyby a odkaz do Sentry rozhraní, kde se vývojář může dozvědět podrobnosti o konkrétní chybě.

Protože je propojení Sentry s Python kódem jednoduchou záležitostí, je možné jej aktivovat i v samotné aplikaci Jabber bota. Tímto způsobem může sentry odesílat prostřednictvím bota informace i o jeho interních chybách, pokud chyba není kritická natolik, že by neumožnila odeslání zprávy.

Obrázek 6.1: Konfigurace XMPP rozšíření pro Sentry

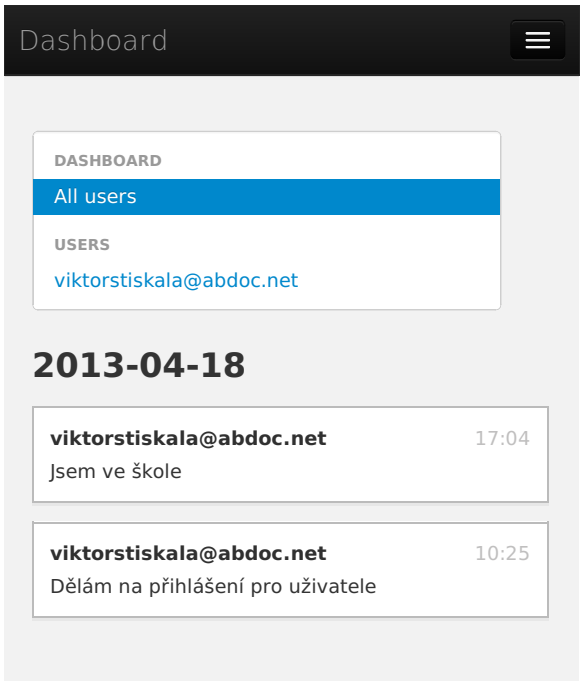
6.2 Webová aplikace s informacemi o práci týmu

Tato aplikace sbírá od uživatelů informace o tom, na čem právě pracují členové týmu a zobrazuje je v přehledné webové aplikaci. V administračním rozhraní aplikace je možné přidat seznam uživatelů a vytvořit otázky, které budou uživatelům zasílány v nastavitelném časovém intervalu.

Část aplikace, která se stará o odesílání otázek je spouštena *cronem*, nebo obdobným systémovým plánovačem. Při každém spuštění se tak kontroluje čas, kdy byla naposledy uživateli otázka odeslána a pokud překročí nastavený interval, je otázka odeslána znovu.

Otázky jsou zasílány s nastaveným příznakem `expire_on_offline`, aby uživatelům nechodily, když jsou ve stavu *offline*.

Webové rozhraní je nakódováno tak, aby vypadalo dobře i na menším displeji, například na mobilním telefonu. Uživatelé se tak mohou jednoduše podívat na aktuální informace i když jsou mimo počítač. Vzhled rozhraní na mobilním telefonu je vidět na obrázku 6.2.



Obrázek 6.2: Prostředí webové aplikace zobrazené na mobilním telefonu

Kapitola 7

Integrace

Tato kapitola se věnuje možností integrace Jabber bota do jiných systémů a podrobně rozebírá možnosti využití. Volně tak navazuje na kapitolu 6, která popisuje využití v praxi na dvou konkrétních příkladech.

7.1 Nástroj pro zasílání notifikací

Jednou z možností použití je využití jako nástroj pro zasílání notifikací na různé akce. Může tak snadno nahradit emailové notifikace krátkými informačními zprávami o stavu.

Toto využití se hodí do různých informačních systémů, nebo eshopů. Eshop může tímto způsobem jednoduše informovat zákazníka o aktuálním stavu objednávky a nahradit tak alespoň pro určitou část uživatelů drahé textové zprávy. Dalším příkladem pro využití notifikací může být školní systém, který takto může informovat o získaných bodech z projektů, udělení zápočtu, apod.

Při využití typu zprávy *otázka* je možné využít i odezvu uživatele k provedení akce. Po vyhodnocení zkoušky může být student informován, že neprospěl a může mu být přímo ve zprávě nabídnuta možnost zapsat se automaticky na další termín. Toto využití navíc nemusí být implementováno přímo v informačním systému, ale může jej zajišťovat externí aplikace. Tomuto propojení se věnuje sekce 5.5.1.

Některé webové stránky obsahují kontaktní formulář, který většinou odesílá email na zadanou adresu. Tyto formuláře je možné nahradit, nebo jen doplnit zasíláním XMPP zprávy na zadané Jabber ID.

V knize Jabber Developer's Handbook [9] je popsán způsob, jak využít notifikačního systému pro zasílání informací o systémových událostech na serveru. Externí aplikace může sledovat chybové logy běžících aplikací i samostného systému a při výskytu neočekávané události odeslat upozornění uživateli. Jednodušší způsob, jak řešit informování o chybách však poskytují specializované, přímo k tomuto účelu vytvořené systémy. Sentry, který je zástupcem jednoho z nich v základu XMPP notifikace neobsahuje, jejich odesílání je ale možné prostřednictvím XMPP rozšíření, které je součástí této práce (viz. sekce 6.1).

7.2 Zpracovávání událostí z chat místnosti

Jabber bot umožňuje připojení se do chat místnosti, neboli MUC[1], kde může jak reagovat na příkazy, tak zpracovávat jednotlivé zprávy a další informace a tyto odesílat na externí

[1]Multi User Chat – rozhovor s více uživateli, vlastnost podporovaná XMPP protokolem

URL adresu. Díky této funkci je možné vytvářet externí systémy, které události v těchto místnostech zpracovávají.

Často jsou chat místnosti spojené s nějakou službou, či webem, o které je možné prostřednictvím bota účastníky chatu informovat. V chat místnosti určené vývojářům určitého softwaru je například možné informovat účastníky o tom, že byla právě vypuštěna nová verze, nahlášena nová chyba, apod.

Taktéž je prostřednictvím bota možné ukládat historii konverzace a tuto zpřístupňovat na webových stránkách.

Kapitola 8

Testování

8.1 Testování přesnosti metody pro řešení problému více otázek

V průběhu vývoje aplikace byla vyvinuta metoda pro predikci, na kterou ze zaslaných otázek uživatel odpověděl. O tomto problému více pojednává sekce 5.3 na straně 22.

8.1.1 Popis metody pro predikci

Testovaná metoda bere v úvahu 2 proměnné: *čas* a *stav uživatele* přičemž stav uživatele je při každé změně ukládán do záznamu pro pozdější analýzu. Nejsou uvažována další rozšíření poskytující dodatečné informace o stavu, ukládán je tedy pouze stav tak, jak jej popisuje samotná specifikace protokolu.

Jakmile aplikace obdrží zprávu, která je vyhodnocena jako odpověď a v databázi je pro konkrétního uživatele uložena více než jedna otázka, vyvolá se funkce zajišťující predikci. Ta nejprve zhodnotí časový rozestup mezi zaslanými otázkami a poté zkontroluje změny stavu mezi jednotlivými otázkami. Například byl-li uživatel po celou dobu online, nebo jesli těsně po zaslání otázky změnil stav na jeden z nedostupných, apod. Pokud je časový rozestup mezi zaslanými otázkami dostatečně malý, očekává se, že na otázky uživatel odpoví ve správném pořadí. Všechny faktory a proměnné je však potřeba testovat v reálném nasazení a podle aktuálních výsledků správnosti predikce tyto proměnné upravovat.

8.1.2 Testování v reálném nasazení

Aplikace byla mezi jednotlivými iteracemi spuštěna vždy po dobu přesně 12 hodin a externí testovací skript zajišťoval zasílání připravených otázek celkem 9 lidem v takových intervalech, aby napomáhaly ověření mezních hodnot. Otázky byly zasílány v běžné pracovní době s nastaveným příznakem `expire_on_offline`, tudíž uživatelům kteří byli v dobu zaslání otázky ve stavu *offline* tato otázka nepřišla. Protože test probíhal v rámci jedné firmy, obsahovaly otázky pro uživatele vesměs dotazy na to, na čem právě pracuje, přičemž uživatelé nevěděli o tom, že jsou jejich odpovědi použity mimo jiné také k testování této metody. Otázky však byly kladeny takovým způsobem, aby odpovědi na ně nebyly vzájemně zaměnitelné.

Vzhledem k tomu, že byl testovací vzorek poměrně malý a při jednotlivých iteracích nedocházelo pouze k malým změnám, ale mnohdy k přepracování celé logiky pro predikci, probíhalo vyhodnocení testu spíše úsudkem, než podle přesných dat.

Výsledky se radikálně lišily uživatel od uživatele, což může znamenat, že každý uživatel má odlišné způsoby odpovědi. Například část uživatelů pokud neodpověděla na první otázku přibližně do půl hodiny a mezitím jim přišla otázka druhá, tak odpověděli pouze na tuto druhou otázku a na první z nich pravděpodobně zapomněli. Jiní uživatelé ale i po takto dlouhé době odpověděli na obě otázky v pořadí.

Z toho vyplývá, že je tato metoda příliš nepřesná pro nasazení v reálném systému, kde by přiřazením špatné odpovědi mohlo docházet k závažným chybám. Dodatečné upřesnění výsledků by bylo možné vytvořením profilů pro jednotlivé uživatele, kdy by se aplikace „učila“, jakým způsobem konkrétní uživatele odpovídají.

8.2 Automatické a manuální testování aplikace

Tato sekce popisuje metody použité při vývoji aplikace, pomocí kterých se ověřovala správnost implementace, reakce na nespecifikované zprávy, apod.

Jabber bot byl testován po každé iteraci vývoje metodou „unit testing“. Použitím automatických testů samozřejmě nemůžeme rozumně pokrýt kompletní specifikaci, včetně vzájemného ovlivňování se jednotlivých funkcionalit, proto byl program otestován také manuálně.

8.2.1 Unit testing

Unit testing je důležitou součástí vývojové strategie orientované na testování. Pokud je používáno správně, pomáhá předcházet chybám ve všech fázích vývoje. Umožňuje otestovat, zda jsou všechny testované funkce kompletní a aplikace funkční při rozsáhlejší úpravách kódu, ale zároveň pomáhá při vývoji, kde s pomocí nástrojů dokáže snadno simulovat uživatele. Drobnou nevýhodou však je, že je potřeba testy udržovat a aktualizovat po celou dobu vývoje. [11]

Pro program byly vytvořeny automatické testy ověřující správnost výsledku jednotlivých částí programu. Pro tvorbu testů byl použit modul `unittest`, který je součástí standardní knihovny jazyka Python.

Aby byla hloubka záběru co nejvyšší, byly kromě testování explicitně určených případů užití vytvořeny testy zahrnující méně časté stavy programu. K tomuto účelu byl použit nástroj `coverage` měřící pokrytí kódu, tzv. „code coverage”. Touto metodou je možné zjistit, které řádky kódu byly během použití alespoň jednou vykonány a které ne. Výstup z tohoto nástroje je rovněž možno vizualizovat a zvýraznit tak řádky kódu, které nebyly vykonány přímo v editoru, za předpokladu, že to editor, nebo IDE[1] podporuje. Je však důležité zdůraznit, že 100% pokrytí kódu nutně neznamená, že byly otestovány všechny možnosti testované aplikace.

8.2.2 Manuální testování

Manuální testy měly záběr na části programu, které není možné jednoduše otestovat testy automatickými. Byly například testovány reakce na nestandardní zprávy zaslané bot uživateli přes XMPP protokol. Těmito zprávami jsou myšleny ty, pro které není implementována žádná reakce a někteří IM klienti je mohou zasílat spolu se zprávou.

V počátcích vývoje byly tímto způsobem testování zjištěny chyby, jednalo se například o zprávu informující o zavření okna konverzace, která byla interpretována jako obyčejná

[1] Vývojové prostředí – nejčastěji s cílem usnadnit práci programátora a zvýšit jeho produktivitu.

zpráva, avšak bez textu. Při odeslání příkazu typu `question` (viz. 4.1.1), na který se očekává odpověď uživatele tak docházelo k zaslání odpovědi bez textu HTTP požadavkem na `postback_url`, aniž by uživatel odpověděl. Toto chování by nemělo ovlivnit funkčnost aplikací, ale je nežádoucí.

8.2.3 Způsob testování aplikace

Protože aplikace sama o sobě nemá uživatelské rozhraní, bylo potřeba pro testování functionality použít externí nástroje. Pro odesílání požadavků byly použity program `curl` a HTML stránka s formulářem, který měl nastaveny všechny parametry ve skrytých polích. Pro testování, zda aplikace zasílá správná data na `postback_url` používané u některých typů příkazů byl vytvořen jednoduchý HTTP server, který vypisoval přijatá data a informace o požadavku do konzole.

Kapitola 9

Závěr

V této bakalářské práci jsem vyvíjel aplikaci – Jabber bota ovládaného přes REST API. Po důkladném studiu XMPP protokolu z dostupných materiálů jsem implementoval aplikaci, která je přínosnou pro mnoho způsobů využití, které jsou v této práci podrobně probírány.

Protože aplikace má více součástí, které si udržují vlastní síťové spojení, bylo potřeba vymyslet způsob, jak tyto části mohou fungovat samostatně. V průběhu vývoje jsem nejprve vyzkoušel použití vláken, od kterého jsem však později upustil, především z důvodu složité synchronizace. Kvůli změny bylo sice potřeba změnit celou stávající architekturu aplikace, ale z dlouhodobého hlediska se nepochybně vyplatila. Použitím knihovny Gevent pro asynchronní síťovou komunikaci navíc došlo ke znatelnému zlepšení odezvy aplikace.

Nevýhodou při komunikaci s aplikací je při odeslání více otázek nemožnost spolehlivě identifikovat, na kterou z otázek uživatel odpověděl. V tomto případě se uživateli odešle dialog, ve kterém může správnou otázku vybrat. Byla testována i možnost využití automatického řešení pro rozpoznání na kterou otázku uživatel odpověděl, to ale vedlo k občasným chybám, které by se mohly projevit na funkčnosti aplikace, která Jabber bot využívá. XMPP protokol sice obsahuje prostředky, které by tento problém umožňovaly řešit, ale implementace IM klientů tento problém zohledňuje jen velice zřídka.

Jabber bot je aplikací, která bude pravděpodobně často rozšiřována o další funkce specifické pro účel nasazení a proto byl při vývoji brán ohled na jednoduchost těchto úprav. Velkým kladem je i modulární návrh, který v případě potřeby umožňuje libovolně měnit, nebo rozšiřovat functionalitu integrovaných metod.

Vzhledem k tomu, že se Jabber bot implementovaný v této práci chová jako klient, je navíc zcela nezávislý na použitém XMPP Serveru, nebo klientech.

Literatura

[1] BILENKO, D.: gevent.monkey – Make the standard library cooperative. Dostupné z `http://www.gevent.org/gevent.monkey.html`, [cit 2013-05-10].

[2] CRAMER, D.: Sentry documentation. Dostupné z `http://sentry.readthedocs.org/en/latest/index.html`, 2011 [cit 2013-05-12].

[3] DIERKS, T.: The Transport Layer Security (TLS) Protocol Version 1.2 [online]. `http://tools.ietf.org/html/rfc5246`, 2008-08.

[4] FIELDING, R. T.: Architectural Styles and the Design of Network-based Software Architectures [online]. Dostupné z `http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`, 2000.

[5] FIELDING, R. T.; et. al.: Hypertext Transfer Protocol – HTTP/1.1 [online]. Dostupné z `http://tools.ietf.org/html/rfc2616`, 1999-06, [cit. 2013-05-02].

[6] Foundation, P. S.: Python History and License. Dostupné z `http://docs.python.org/2/license.html`, [cit 2013-05-14].

[7] HOANG, Q.: Email Statistics Report, 2012-2016 [online]. Technická zpráva, The Radicati Group, Inc., 1900 Embarcadero Road, Palo Alto, CA, 2012.

[8] MACEDO, T.; OLIVEIRA, F.: *Redis Cookbook*. O'Reilly Media, Inc., první vydání, 2011, ISBN 978-1-449-30504-8.

[9] MOORE, D.; WRIGHT, W.: *Jabber Developer's Handbook*. Sams Publishing, 2004, ISBN 0-672-32536-5.

[10] NASEVIC, D.: Redis in the NoSQL Ecosystem. Dostupné z `http://blog.siyelo.com/redis-in-the-nosql-ecosystem`, [cit 2013-05-01].

[11] PILGRIM, M.: *Dive Into Python 3*. Apress Media LLC, 2009, ISBN 978-1-4302-2415-0.

[12] RICHTA, K.: SWI041: Návrhové vzory [online]. `http://www.ksi.mff.cuni.cz/~richta/NSWI041/SWI041-6.pdf`, Naposledy navštíveno 23. 3. 2013.

[13] SAINT-ANDRE, P.: Extensible Messaging and Presence Protocol (XMPP): Core [online]. `http://tools.ietf.org/html/rfc3920`, 9 2004.

[14] SAINT-ANDRE, P.: Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for the Extensible Messaging and Presence Protocol (XMPP) [online]. Dostupné z `http://tools.ietf.org/html/rfc5122`, 2 2008, [cit. 2013-05-06].

[15] SAINT-ANDRE, P.: Extensible Messaging and Presence Protocol (XMPP): Core [online]. Dostupné z `http://tools.ietf.org/html/rfc6120`, 3 2011, [cit. 2013-05-03].

[16] SAINT-ANDRE, P.; SMITH, D.: XEP-0085: Chat State Notifications. Dostupné z `http://xmpp.org/extensions/xep-0085.html`, 2009-09-23 [cit 2013-05-11].

[17] SAINT-ANDRE, P.; SMITH, K.; TRONÇON, R.: *XMPP: The Definitive Guide*. O'Reilly Media, Inc., první vydání, 2009, ISBN 978-0-596-52126-4.

[18] WHITE, J. E.: A High-Level Framework for Network-Based Resource Sharing [online]. `http://tools.ietf.org/html/rfc707`, 1976-01-14.

Příloha A

Obsah CD

`src/gmarie/`	Zdrojové soubory Jabber bota
`src/dashboard/`	Zdrojové soubory testovací webové aplikace
`src/sentry_xmpp/`	Zdrojové soubory Sentry XMPP rozšíření
`text/`	Zdrojové soubory této práce ve formátu LaTeX
`bp.pdf`	Tato práce v elektronické podobě ve formátu PDF

Příloha B

Instalační manuál

Zdrojové kódy jsou k dispozici na přiloženém CD (viz. příloha A), nebo online na webové stránce `https://github.com/hareevs/marie-jabber-bot/`. Pro získání zdrojového kódu z online zdroje je vhodné použít nástroj GIT.

B.1 Postup instalace pro Jabber Bot

Pro tento postup se předpokládá operační systém Linux s nainstalovaným kompilátorem (ideálně GCC[1]) a balíčky potřebnými pro kompilování Python knihoven. Postup níže se snaží být co nejvíce obecný, některé kroky však mohou být specifické pro Linuxovou distribuci Arch Linux (`https://www.archlinux.org/`), pro ostatní distribuce bude však postup obdobný.

U distribucí založených na Debianu[2] je potřeba instalovat také *dev* balíčky, protože standardní instalace balíků neobsahuje hlavičkové soubory potřebné pro kompilaci některých knihoven.

Aplikace vyžaduje mít v systému nainstalovaný Python verze 2.7, dále `Redis`, `pip` (novější alternativa pro `easy_install`) a doporučuje se použít `virtualenv`. Instalace jednotlivých součástí je popsána níže.

B.1.1 Redis

Redis je `key-value` databáze, pro instalaci nejčastěji dostupná jako systémový balíček. Balík pro Arch Linux se nazývá `redis`, pro Debian a jeho derivace potom `redis-server`. Jabber bot využívá standardní nastavení, tedy hostname `localhost`, port `6379` a databázi `0`. Toto nastavení je možné změnit úpravou slovníku `Eventbot.REDIS_CONFIG`.

B.1.2 Pip

Pip je nástroj pro instalaci a správu Python knihoven z PyPi (Python Package Index). Instalace na Arch Linuxu je možná pomocí balíčku `python2-pip`, ostatní způsoby instalace jsou popsány v dokumentaci k tomuto nástroji. Dokumentace je dostupná na webové stránce `http://www.pip-installer.org/en/latest/installing.html`.

[1] **GCC** (Gnu Compiler Collection) – sada překladačů, ve většině Linuxových distribucí dostupná jako systémový balíček

[2] Debian, Ubuntu, Linux Mint, apod.

B.1.3 Dodatečné Python knihovny

Seznam dodatečných knihoven je v souboru `requirements.txt`. Pro instalaci je možné použít i připravený shellový skript `install_requirements.sh`. Pokud kompilace probíhá na GCC verze 4.8, je potřeba kvůli chybě v knihovně `greenlet` verze 0.4.0 nejprve nastavit `CFLAGS` pomocí `export CFLAGS="-O2 -fno-tree-dominator-opts"`, jinak by mohlo docházet k náhodným pádům aplikace. Připravený shell skript toto dělá automaticky.

Doporučuje se použít `virtualenv`, aby se knihovny nemusely instalovat do systému. Pro instalaci slouží příkaz `pip install virtualenv`.

Instalace knihoven do systému probíhá následovně:

```
git clone git@github.com:hareevs/marie-jabber-bot.git
cd marie-jabber-bot
virtualenv venv --distribute
source venv/bin/activate
./install_requirements.sh
```

Pokud při posledním kroku dojde k chybě, je potřeba doinstalovat do systému potřebné balíčky (případně *dev* balíčky pro hlavičkové soubory) a poslední krok opakovat. Chybový výpis nástroje `pip` by měl být dostatečným zdrojem informací, které pomůžou při hledání správného balíku.

B.1.4 Spuštění aplikace

Po úspěšném dokončení předchozích kroků je možné prohlédnout si a případně upravit konfiguraci v souboru `gmarie/start.py` a poté spustit následujícím způsobem:

```
cd marie-jabber-bot
source venv/bin/activate # pouze při použití virtualenv
cd gmarie
python start.py
```

B.2 Instalace testovací webové aplikace

Webová aplikace využívá webový framework *Django*, knihovnu *requests* a databázi *SQLite*. Podobně jako pro Jabber bot je využit nástroj pip pro instalaci dodatečných knihoven (instalace viz sekce B.1.2) a jsou vyžadovány určité systémové balíky, které jsou blíže popsány v sekci B.1.

Zdrojové kódy testovací aplikace jsou dostupné na přiloženém CD (viz. příloha A). Postup pro instalaci knihoven a spuštění aplikace je následující:

```
cd dashboard
virtualenv venv --distribute    # pouze při použití virtualenv
source venv/bin/activate        # pouze virtualenv
pip install -r requirements.txt
cd dashboard
python manage.py syncdb --noinput

python manage.py runserver
```

Po spuštění posledního kroku by aplikace měla být dostupná prostřednictvím webového prohlížeče na adrese `http://127.0.0.1:8000`. Komunikace s Jabber botem probíhá podle konfigurace uložené v souboru `settings.py`, konkrétně `BOT_URL` a `POSTBACK_URL`.

Pro přístup k administračnímu rozhraní na adrese `http://127.0.0.1:8000/admin/` je potřeba mít vytvořený administrátorský účet. Pro vytvoření je možno použít příkaz `python manage.py createsuperuser` ve složce `dashboard/dashboard/` a potom následovat interaktivního průvodce v konzoli.

B.3 Instalace XMPP rozšíření pro Sentry

Pro instalaci tohoto rozšíření je potřeba nejprve nainstalovat sentry do odděleného virtualenv prostředí a při aktivovaném prostředí nainstalovat aplikaci ze složky `sentry_xmpp`. Zdrojové soubory se nacházejí na CD, viz. příloha A.

Postup pro instalaci je následující:

```
source venv/bin/activate
pip install -e sentry_xmpp
```

Po instalaci je potřeba v nastavení projektu povolit rozšíření XMPP a konfigurovat.

Příloha C

Ukázka nastavení autentizace pro Jabber bot

Jabber bot nemá implementovánu autentizaci uživatelů z důvodu snadnějšího přizpůsobení pro různé účely. Přidání autentizace je však jednoduchým krokem navíc a je možné například pomocí webserveru Nginx[1], který je popsán v této příloze. V této metodě je využita reverzní proxy.

C.1 Konfigurace Nginx

Tato sekce popisuje nastavení *jednoduchého ověření přístupu* (Basic access authentication) HTTP protokolu, pomocí webového serveru Nginx. Pro toto nastavení se předpokládá nastavení HTTP listeneru Jabber bota tak, aby naslouchal na IP adrese `127.0.0.1` a portu `8088`, tj. nebude k němu přístup zvenčí. O přístup zvenčí se postará Nginx server, který zároveň přidá autentizační vrstvu. Webserver bude pro jednoduchost ukázkové konfigurace zpřístupňovat Jabber bot REST API na portu `8080`.

Část konfiguračního souboru s definicí serveru může vypadat následovně:

```
server {
    listen 8080;        # naslouchání na všech IP adresách a portu 8080

    location / {
        proxy_pass              http://127.0.0.1:8088/;
        proxy_redirect          off;
        proxy_buffering         off;

        auth_basic              "Jabber bot REST API - authorised access only";
        auth_basic_user_file    auth/htpasswd;
    }
}
```

[1] `http://wiki.nginx.org/Main`

C.1.1 Oprávnění pro přístup

Pro přístup k API je nyní potřeba znát uživatelské jméno a heslo. Tyto se nacházejí v souboru `auth/htpasswd`. Pro jeho konfiguraci je možné použít nástroj *htpasswd*.

Obsah `htpasswd` souboru může vypadat například takto:

```
viktor:$apr1$oz7VWFk8$q8V5X3N9hDydH61xrrGm61
petr:$apr1$UEuK1gDj$bsft8hZlM2DDnAvvixHdp1
josef:$apr1$k/Q.sAsy$Na6BSCT2A8QQLpvhvnNNQ/
```

Z ukázky je patrné, že heslo je uložené v hašovaném tvaru s použitím soli. Kromě ukázkového příkladu je možné přidat například LDAP autentizaci, nebo jiné, které podporují webservery, nebo jiné reverzní proxy servery.