TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Aplikace SMS brána

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Vojtěch Bartoš**
Vedoucí práce: Mgr. Jiří Vraný Ph.D.

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

SMS gateway application

Diploma thesis

Study programme: N2612 – Electrotechnology and informatics
Study branch: 1802T007 – Information technology

Author: **Bc. Vojtěch Bartoš**
Supervisor: Mgr. Jiří Vraný Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2014/2015**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Vojtěch Bartoš**
Osobní číslo:	**M11000227**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Aplikace SMS brána**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Seznamte se problematikou programování webových služeb s podporou REST api. Dále proveďte rešerši a kritické zhodnocení existujících webových služeb pro SMS brány a k nim příslušných mobilních klientů.
2. Vytvořte návrh webové služby - SMS brány, včetně rozhraní pro přístup z mobilních klientů. Rozhraní by mělo odpovídat principům REST. K webové službě navrhněte také mobilního klienta, včetně způsobu jeho komunikace se službou. Zvolte zda klient bude aplikací nativní pro vybranou platformu či HTML5 multiplatformní.
3. Oba návrhy implementujte a vytvořte webovou službu i mobilní aplikaci pro SMS bránu.

Rozsah grafických prací:	**dle potřeby**
Rozsah pracovní zprávy:	**60 stran**
Forma zpracování diplomové práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] FIELDING, Roy Thomas. Architectural Styles and the Design of Networkbased Software Architectures [online]. University of California, 2000 [cit. 20131008]. Dostupné z: http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm

[2] RICHARDSON, Leonard a Sam RUBY. RESTful web services. 1st ed. Sebastopol: O'Reilly, 2007, xxiv, 419 s. ISBN 9780596529260.

[3] Android API Guides. Developer Android [online]. 2012 [cit. 20121014]. Dostupné z: http://developer.android.com/guide/components/index.html.

[4] MARK, Dave a Jeff LAMARCHE. IPhone SDK: průvodce vývojem aplikací pro iPhone a iPod touch. Vyd. 1. Brno: Computer Press, 2010, 480 s. ISBN 9788025128206.

[5] Miguel Grinberg. Flask Web Development: Developing Web Applications with Python. Vyd. 1. O'Reilly, 2014, 258 s. ISBN 9781449372620

[6] Pilgrim Mark. Dive Into Python. Springer 2004, 436 s. ISBN 9781590593561

[7] Apple Inc. The Swift Programming Language [online]. Dostupné z: https://itunes.apple.com/gb/book/swiftprogramminglanguage/id881256329?mt=11

[8] Videla Alvaro a Jason J. W. Williams, RabbitMQ in Action: Distributed Messaging for Everyone, Vyd. 1. Manning Publications 2012, 312 s. ISBN 9781935182979

Vedoucí diplomové práce:	**Mgr. Jiří Vraný, Ph.D.** Ústav nových technologií a aplikované informatiky
Konzultant diplomové práce:	**Jakub Alimov** Alinet.cz

Datum zadání diplomové práce:	**20. října 2014**
Termín odevzdání diplomové práce:	**15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 20. října 2014

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 11. 9. 2015

Podpis:

Abstrakt

Tato diplomová práce se zabývá návrhem a vývojem aplikace pro SMS bránu, která by měla umožnit rozesílání a příjem SMS textových zpráv pro potřeby různých společností, např. internetových obchodů. Cílem práce je navrhnout a implementovat aplikace jako webovou službu s podporou REST API tak, aby ji bylo možné ovládat i z mobilních klientů.

Práce popisuje návrh a implementaci webové služby, klienta a komunikace s USB modemem, který je schopen odesílat a přijímat SMS textové zprávy. Komunikace mezi webovou službou a klientem je realizována využitím REST API rozhraní a datového formátu JSON.

Klíčová slova: REST, JSON, webová služba, klient, server, React

Abstract

This diploma thesis deals with the design and development of an application for an SMS gateway handling the dispatching and receiving of SMS text messages for companies. The goal of the thesis is the design and implementation of the application as a web service supporting a REST API interface, so that it can be controlled from mobile clients.

This thesis describes the design and implementation of the web service, the client and its communication via a USB modem, which is able to dispatch and receive SMS text messages. The communication between the web service and the client is realized with the help of a REST API interface and the JSON data format.

Keywords: REST, JSON, web service, client, server, React

Poděkování

Na tomto místě bych chtěl poděkovat především vedoucímu diplomové práce panu Mgr. Jiřímu Vranému, Ph.D., konzultantovi práce panu Jakubu Alimovovi a firmě Alinet cz, s. r. o. za ochotu, rady a veškerou pomoc při konzultacích.

Dále bych rád poděkoval své rodině a přítelkyni za oporu, kterou mi pokytovaly po celou dobu mého studia a psaní této diplomové práce.

V neposlední řadě děkuji Mgr. Zuzaně Korfové za korekturu.

Použitý software

Tato práce byla vysázena programem LaTeX (prostředí Texpad ve verzi 1.4.7 a sázecí systém MacTex 2015) pod operačním systémem Mac OS X 10.10.3. Jako nástroj pro tvorbu zdrojových kodů byl použit textový editor Atom ve verzi 1.0.3. Pro uložení dat byl použit MariaDB databazový server a RabbitMQ.

Kontakt

E-mail: hi@vojtech.me

Obsah

Seznam obrázků

Seznam zdrojových kódů

Seznam zkratek

AJAJ	Asynchronous JavaScript and JSON
AJAX	Asynchronous JavaScript and XML
AMQP	Advanced Message Queuing Protocol
CI	Continuous integration
DOM	Document Object Model
ERD	Entity Relationship Diagram
GSM	Global System for Mobile Communications, Groupe Spécial Mobile
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protoco
JS	JavaScript
JSON	JavaScript Object Notation
MIT	Massachusetts Institute of Technology
MVC	Model View Controller
NPM	Node package manager
ORM	Object-relational mapper
REST	Representational State Transfer
SMS	Short Message Service
SPA	Single-page aplikace
SQL	Structured Query Language
SSL	Secure Sockets Layer
TDD	Test Driven Development
USB	Universal Serial Bus
UX	User experience
WSGI	Web Server Gateway Interface

1 Úvod

Žijeme v době moderních komunikačních technologií, bez kterých si většina lidí už nedokáže život představit. Téměř všichni vlastníme mobilní telefony. Ze zařízení, které původně sloužily pouze k telefonování, posílání a přijímání SMS textových zpráv, se postupně staly zařízení, ktere mohou sloužit jako "malý osobní počítač". Tím pádem můžeme být neustále připojeni na internetové síti a využívat její možnosti. Jednou z nejvíce využívaných oblastí je internetové nakupování. Internetové obchody hledají možnosti, jak komunikovat s koncovými zákazníky. V současné době chtějí obchodníci komunikovat se zákazníky co nejjednoduším způsobem, a to jim nabízejí SMS textové zprávy.

Cílem této práce je vytvořit aplikaci SMS brána, která dokáže komunikovat s připojeným zařízením pro zacházení s textovými zprávami pro společnost Alinet cz, s. r. o.

Aktuální využití SMS brány v této společnosti se momentálně realizuje pomocí vlastního řešení, které je vytvořeno specificky pro každý projekt zvlášť, a to není optimální. Hlavním požadavkem firmy je vytvoření aplikace, která bude fungovat odděleně od všech projektů jako webová služba, kterou ostatní systémy budou využívat v případě nutnosti práce s textovými zprávami. Hlavní důraz zadavatele je kladen na vytvoření řešení, které bude možno provozovat na interních serverech běžících pod operačním systémem Linux. Webová služba by měla umět pracovat i s různými uživateli, uchovávat adresář kontaktů či skupin a také mít možnost vytvářet externí přístupy pro aplikace třetích stran.

Hlavní motivací této práce bylo především vytvořit webovou službu a klientskou aplikaci za pomocí moderních technologií, kterou bude využívat nespočet projektů jako svou primární SMS bránu. Velkou výhodou práce byla absolutní volnost při výběru použitých technologií po konzultaci se zadavatelem práce.

Jedním z cílů práce bylo udělat rešerši a zhodnotit již existující řešení. Veškerá řešení musela být vystavěna jako open-source pod licenci pro volné použití.

Dalším cílem práce bylo vytvoření návrhu webové služby s podporou REST [1] API rozhraní a vytvoření návrhu příslušného nativního či HTML5 multiplatformního klienta, který bude spravovat uživatelské požadavky. Požadavkem se rozumí správa kontaktů, skupin, textových zpráv. Administrátor bude mít možnost spravovat i užitatelské účty a jejich oprávnění.

Posledním cílem práce je implementace a vytvoření obou řešení dle navrženého řešení, které bude nasazeno a otestováno na firemních serverech společnosti Alinet cz, s. r. o.

1.1 Jazykové konvence

Práce obsahuje přeložené anglické termíny, pro které překlad postihuje jejich význam. Některé termíny jsou ponechány bez překladu, jelikož pro ně zatím v českém jazyce ustálený význam neexistuje. Zde jsou uvedeny konkrétní termíny s jejich českým překladem:

- *framework* - softwarová struktura sloužící jako podpora při programování a vývoji softwarových projektů,
- *worker* - proces, který vykonává nějakou činnost na pozadí.

2 Existující řešení

Součástí zadání je zhodnotit existující webové služby pro SMS brány a jejich příslušné mobilní klienty.

Hlavním požadavkem při hledání existujících řešení byla možnost nasadit vybrané hotové řešení na firemním serveru s použitím vlastního USB modemu či telefonu, přes který by se posílaly veškeré textové zprávy. Tento požadavek bohužel nesplňují SMS brány, které pokytují firmy třetí strany, a proto byly vyloučeny z rešerše existujících řešení.

Byly nalezeny následující webové služby, které jsou vystaveny open-source na portálu github.com pod licencemi jako GNU GPL [21] nebo Apache 2 [22].

2.1 gosms

Gosms je nedávno vytvořený projekt. SMS brána je naprogramována v jazyce Go a hlavní důraz je kladen na jednoduchost použití a instalace. Podporuje téměř všechny GMS modemy a je možné jej provozovat na všech dnešních operačních systémech jako Windows, GNU\Linux či MacOS. Umožnuje také podporu více USB modemů najednou.

Bohužel tento projekt je jen webová služba pro posílání textových zpráv, nepodporuje žádnou z požadovaných funkcionalit jako správa kontaktů či skupin a příjem textových zpráv. Součástí projektu není také žádný klient, který by mohl být využit pro potřeby uživatelů z mobilních klientů nebo z webového prohlížeče. Webová služba nepodporuje žádnou metodu autentizace, tudíž z pohledu bezpečnosti nelze tento projekt veřejně vystavit na specifické doméně.

Nicméně všechny nedostatky, kterými projekt momentálně disponuje, budou vyřešeny dle plánu projektu vystaveného v oficiálním Git repositáři. Zdrojové kódy projektu jsou dostupné na adrese https://github.com/haxpax/gosms.

2.2 PlaySMS

PlaySMS je zdarma a open-source software pro správu SMS. Flexibilní webově založený systém, který může být využit jako více služeb, např. SMS brána, osobní systém zasílání zpráv, firemní a skupinový nástroj. Projekt byl naprogramován kompletně v jazyce PHP a komunita vývojářů okolo projektu je veliká a velice aktivní. Součást projektu je i velmi podrobná dokumentace ohledně instalace či veškeré

konfigurace.

Tento projekt splňuje veškeré nároky na funkcionalitu, které byly požadovány zadavatelem. Součástí projektu je webový klient, který výborně pracuje v mobilních webových prohlížečích, a dokonce existuje také nativní klient pro platformu Android.

Avšak webová služba, která je navrhnuta a implementována pro komunikaci s klienty třetích stran, nevyhovuje požadavkům zadání. Webová služba má podporu REST API, ale všechny požadavky využívají jediné HTTP metody GET a z toho vyplývá, že veškerá data se posílají jako součást URL adresy. Toto řešení není zcela optimální, bezpečné a jednoduché k použití, avšak dokumentace webové služby je velice podrobná.

Projekt je dostupný na adrese http://playsms.org/.

2.3 Jasmin

Jasmin je open-source SMS brána s mnoha funkcemi pro korporátní sféru. Aplikace Jasmin je vytvořen tak, aby mohl být jednoduše upravena pro specifické potřeby uživatelů. SMS brána je napsána v jazyce Python za použití frameworku Twisted pro vysoce škálovatelné aplikace. Doručování SMS zpráv může být provedeno pomocí HTTP a SMPP protokolů. Web pro správu brány byl naprogramován za použití Python frameworku Django.

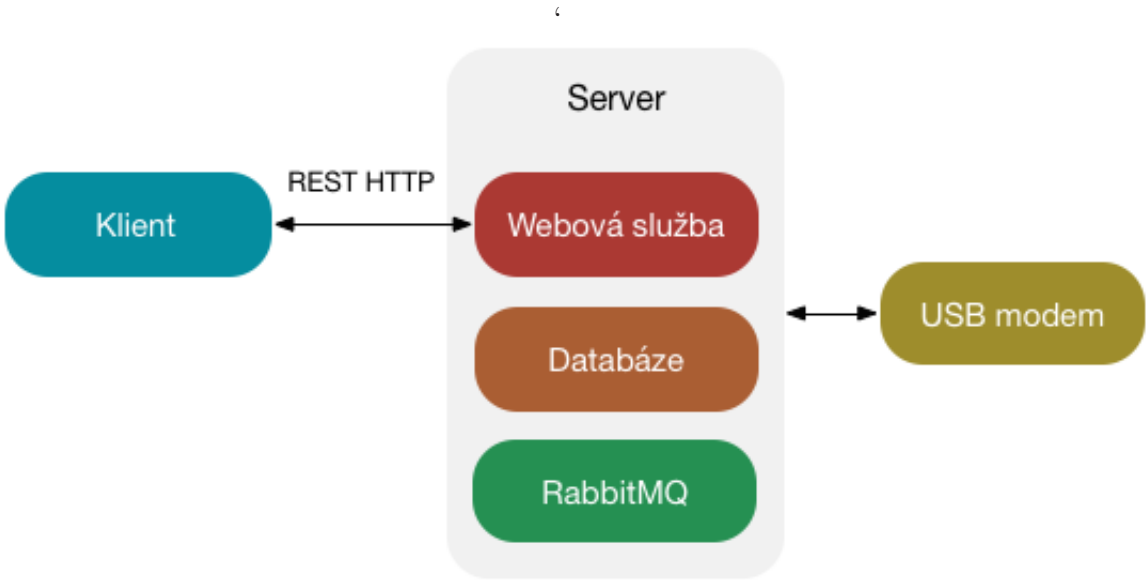
Jasmin splňuje většinu nároků na funkcionality a webovou službu, ať už se jedná o práci se SMS zprávami, nebo možnosti správy kreditů uživatelů. Nicméně neobsahuje žádného klienta pro uživatelské použití, které by podporovalo udržování adresáře lidí, skupiny a klasické jednoduché posílání textových zpráv. Jasmin je webová služba neboli prostředník, který ulehčuje práci s USB modemem či telefonem a implementuje zajímavé funkcionality.

Projekt je dostupný na adrese http://www.jasminsms.com/.

3 Návrh aplikace

Cílem práce je navrhnout a implementovat aplikaci SMS brána, která by byla schopna přijímat a odesílat textové zprávy. Klientská část by měla být implementována jako webová služba s podporou REST API tak, aby ji bylo možno ovládat z mobilních klientů. V této kapitole je popsán způsob odesílání a přijímání textových zpráv, struktura databáze a návrh implementace klienta a webové služby.

Dle návrhu od zadavatele by aplikace měla být klient-server architektury a jako zařízení, které bude odesílat a přijímat textové zprávy, by měl být vybrán jakýkoliv dostupný USB modem.



Obrázek 3.1: Návrh implementace

Výsledná aplikace by měla být napsána jako open-source software pod licencí MIT [10] a dále by měla být veřejně umístěna na webovém portálu github.com.

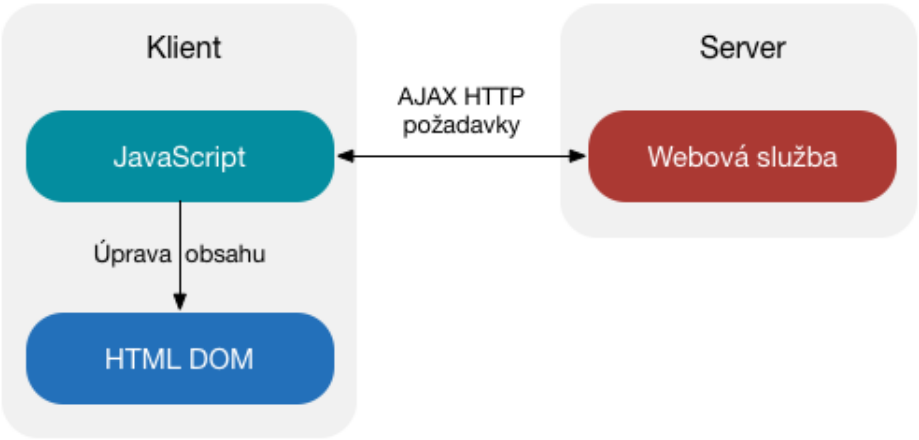
3.1 Klient

Součástí zadání je zvolit, zda bude klient nativní, či HTML5 multiplatformní webová aplikace. Po poradě s konzultantem a zadávajícím práce bylo rozhodnuto,

že klient bude HTML5 multiplatformní z důvodu přístupnosti aplikace ve webovém či mobilním prohlížeči.

Trendem dnešní doby je vytvářet webové aplikace, které běží výhradně na straně prohlížeče a dotazují se webové služby pouze na nezbytná data k zobrazení. Pokud webová aplikace pro každou změnu jakéhokoliv obsahu musí načíst znovu celou stránku, je to nepraktické a jsou přenášeny i části obsahu, které se vůbec nezměnily. Proto je velmi výhodné přenášet data, která mají být na stránce změněna či aktualizována. Webové aplikace, které běží výhradně na straně klienta v prohlížeči, se nazývají single-page aplikace.

Single-page aplikace neboli SPA jsou aplikace, které načtou jednu počáteční HTML stránku a dynamicky aktualizují obsah tak, jak uživatel manipuluje s aplikací. SPA využívají techniky AJAX a technologie HTML5 pro lepší UX bez neustálých opětovných načtení celých stránek. Nicméně to znamená, že veškerá manipulace se stránkou se provádí na straně klienta, tedy v prohlížeči pomocí jazyka JavaScript.



Obrázek 3.2: Schéma single-page aplikace

Klientská aplikace bude vyvinuta kompletně v programovacím jazyce JavaScript, který vychází z návrhu standardu ECMAScript 6 [20] za pomocí knihovny React [12] pro vytváření uživatelských rozhraní, a bude využívat Flux [13] architekturu. Veškeré moduly klienta budou splňovat CommonJS [23] formát zápisu.

3.1.1 React

React je open-source JS knihovna pro vytváření uživatelských rozhraní nejvíce zaměřená pomoci vývojářům vyvíjet single-page aplikace. Knihovna byla vytvořena převážně vývojáři z firem Facebook a Instagram.

React byl vytvořen k tomu, aby pomohl vývojářům vyvíjet velké aplikace, ve kterých se v průběhu mění data. Hlavním cílem je jednoduchost a uspořádanost. Jak bylo řečeno, knihovna řeší pouze uživatelské rozhraní a je považována za View v Model-View-Controller vzoru, tím pádem může být používána samostatně, a nebo může být integrována s větším MVC frameworkem, jako je AngularJS.

Knihovna řeší momentálně největší problém při práci s DOMem, a to je samotný DOM, respektive manipulace. Každá manipulace stojí výkon a není intuitivní, jelikož při použití nativní JavaScript funkce *innerHTML* přidáváme zanořené elementy jako datový typ string. React přichází se systémem komponent, o kterém se ve smyslu budoucnosti vývoje webových aplikací dlouho hovoří, ale bohužel zatím nebylo nic implementováno do standardu HTML5. Každá komponenta má vnitřní stav a hierarchii DOM elementů, kterou představuje dle daného stavu. Na každý element se mohou navazovat nějaké události tak, jako pro nativní komponenty (select, input, div), které mohou měnit vnitřní stav komponenty. Pokud nastane změna stavu, knihovna se pokusí sestavit celou hierarchii elementu v dané komponentě a v zanořených komponentách v takzvaném Virtual DOMu, což je virtuální reprezentace DOMu v paměti, a porovná ho s DOMem, který je vykreslen v prohlížeči, a rozdíly promítne. To znamená, že jsou provedeny jen nejnutnější změny, a to je z hlediska výkonu velice důležité.

```
// syntaxe JSX
var HelloMessage = React.createClass({
  render: function() {
    return (
      <div>
        Ahoj {this.props.name}
      </div>
    );
  }
});

// zkompilovaný soubor JSX do JS kódu
var HelloMessage = React.createClass({
  displayName: "HelloMessage",
  render: function() {
    return React.createElement("div", null, "Ahoj ", this.props.name);
  }
});
```

Zdrojový kód 3.1: Ukázka JSX syntaxe a JavaScript výsledku po kompilaci

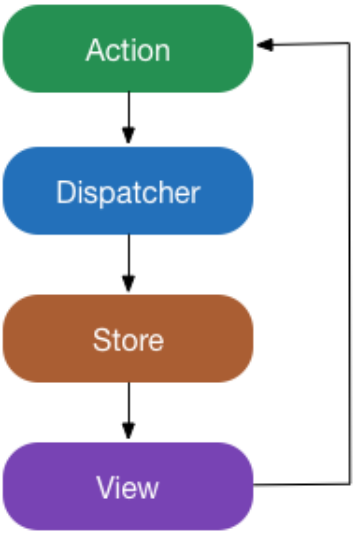
Pro zjednodušení návrhu komponent a hierarchii elementů byla vytvořena syntax JSX, která umožňuje vývojářům používat HTML značky v JS zdrojových kódech. Nicméně toto přináší build proces, protože JSX syntax musí být zkompilována do nativní JavaScript syntaxe, aby zdrojové kódy mohly být použity a spuštěny v prohlížeči.

3.1.2 Flux

Flux je aplikační architektura, kterou využívá firma Facebook pro projekty, ve kterých využívá framework React. Není to framework či knihovna, je to nový druh architektury, který doplňuje React a zavádí koncept jednosměrného toku dat.

Flux se skládá ze 4 základních částí:

- *Action* - funkce, či metody, které umožňují předávání dat do *Dispatcheru*,



Obrázek 3.3: Flux architektura

- *Dispatcher* - přijímá *Action* a odesílá data do všech registrovaných *Store*,
- *Store* - kontejnery pro aplikační stav a logiku,
- *View* - React komponenty vykreslující stav ze *Store*.

3.1.3 CommonJS

V dnešním vývoji webových JavaScript aplikací je velice těžké vytvořit modulovou architekturu aplikace. JavaScript nativně nepodporuje žádné moduly a z tohoto důvodů bude pro klientskou část použitý CommonJS formát zápisu modulů.

Skupina CommonJS definuje formát modulu, který řeší problémy s rozsahem kódu a zajišťuje, že každý modul bude vykonán ve svém jmenném prostoru. Toho je dosaženo pomocí exportování příslušných proměnných či funkcí do okolního světa. Modul má tedy nadefinované rozhraní, které je možno použít, a ostatní funkcionalita je zapouzdřena. Tento formát je použit i na platformě Node a to nám umožňuje psát moduly použitelné jak pro serverovou aplikaci, tak i pro aplikaci běžící v prohlížeči.

```
// module 'setToString'
var rootPrefix = "smsgw";

module.exports = function(prefix, object) {
  Object.keys(object).forEach(function(name) {
    var toStringName = rootPrefix + '/' + prefix + '/' + name;
    object[name].toString = function() {
      return toStringName;
    };
  });
}
```

Zdrojový kód 3.2: Ukázka CommonJS modulu v jazyce JavaScript

Každý modul má přístup k proměnné *module.exports*, která definuje rozhraní modulu. Jakákoliv proměnná či přiřazená funkce bude vystavena k použití a kód se bude vykonávat ve jmenném prostoru modulu. V našem případě modul přepisuje *toString* vlastnost všech vlastností objektu a přidává vlastní prefix před jméno vlastnosti. Proměnná *rootPrefix* je zapouzdřena a nelze ji měnit ani číst zvenčí modulu.

```
var setToString = require('./setToString.js');

setToString('output', {
  'print': function() { ... },
  'log': function() { ... }
});
```

Zdrojový kód 3.3: Ukázka načtení CommonJS modulu v jazyce JavaScript

Výsledný modul je možno importovat do ostatních modulů pomocí funkce *require*, kde první parametr je cesta k souboru či jméno daného modulu, pokud byl nainstalován pomocí NPM balíčkovacího systému.

Použití CommonJS způsobu zápisu modulů nám umožňuje použití celé škály modulů stažených použitím NPM balíčkovacího systému, jelikož všechny moduly splňují CommonJS formát zápisu.

3.1.4 ECMAScript 6

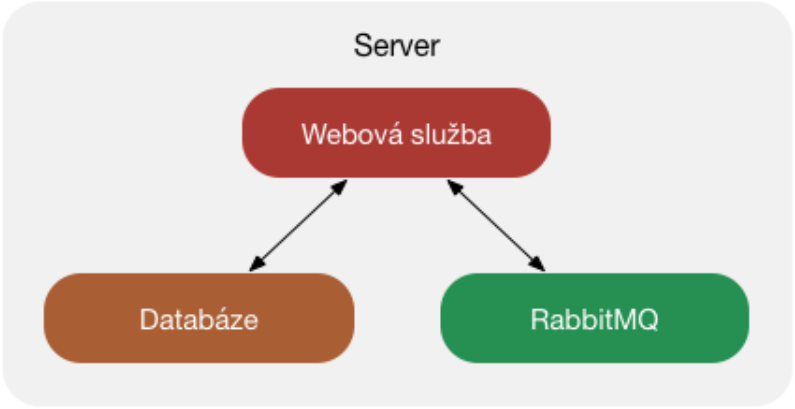
Současná verze JavaScriptu podporovaná všemi staršími a moderními prohlížeči je implementací normovaného skriptovacího jazyka ECMAScript a vychází z verze 5. Nicméně v červenci roku 2012 byl publikován návrh ECMAScript 6 s kódovým jménem *Harmony*, který implementuje podporu nových funkcionalit jazyka jako třídy, konstanty, zlepšení syntaxe jazyka atd. Tento návrh byl oficiálně schválen jako standard v červnu 2015. Bohužel většina moderních prohlížeču tento standard ještě nepodporuje, a tak není možno použít jazyk jako takový. Avšak byl vytvořen kompilátor nazvaný BabelJS, který umožňuje psát a využívat veškerých možností ECMAScript 6 a poté zkomplilovat zdrojové kódy do standardu ECMAScript 5, který je možno bez problémů spustit ve všech moderních webových prohlížečích.

```
// ECMAScript 5
var setToString = require('./setToString.js');

setToString('output', {
  'print': function() { ... },
  'log': function() { ... }
});

// ECMAScript 6
import setToString from './setToString.js';

setToString('output', {
  'print': () => { ... },
  'log': () => { ... }
})
```

Zdrojový kód 3.4: Ukázka rozdílu mezi JS podle standartu ECMAScript 5 a 6

3.2 Webová služba

Požadavkem práce je vytvořit aplikaci jako webovou službu. Webová služba je metoda komunikace mezi dvěma stroji na síti. Služba jako taková může být prezentována softwarovým nebo hardwarovým agentem. Způsob komunikace strojů s webovou službou je vždy dán popisem služby, který definuje protokol komunikace. Během transakce je jeden agent žádajícím o službu a druhý službu poskytuje. Standardy používané webovými službami zajišťují totožnou sémantiku obou agentů. Přínosem webových služeb je možnost využití jejich protokolů jako standardu pro komunikaci mezi systémy, a tak mohou být využity jako prostředník komunikace mezi firemními systémy.



Obrázek 3.4: Schéma komunikace webové služby, databáze a RabbitMQ

Webová služba bude zajišťovat komunikaci v rámci infrastruktury serveru a zároveň bude schopna komunikovat s klienty. Komunikace s klienty bude probíhat výhradně přes REST API rozhraní webové služby tak, jak bylo specifikováno v zadání práce. Komunikace s ostatními částmi infrastruktury jako jsou databáze a RabbitMQ [8] bude probíhat dle protokolu daného pro konkrétní službu.

Jako programovací jazyk pro implementaci webové služby byl vybrán jazyk Python ve verzi 2.7, která je běžně dostupná ve většině linuxových distribucích jako Ubuntu nebo Debian.

Webová služba bude pokryta automatizovanými testy.

3.2.1 REST

REST je architektonický styl rozhraní navržený pro komunikaci mezi webovými službami. Využívá se pro jednotný přístup ke zdrojům, ve kterých nezáleží na reprezentaci dat. Data mohou být reprezentována ve formátu XML nebo JSON. Systémy podporující REST rozhraní většinou komunikují přes protokol HTTP využívající metod GET, POST, PUT, DELETE, atd.

REST využívá základních myšlenek jako bezstavovost, klient-server architektura, kód na vyžádání, využití cache paměti, jednotné rozhraní a vrstvený systém.

3.2.2 Python

Python je interpretovaný a objektově orientovaný programovací jazyk velice podobný jazyku Perl. Tento jazyk klade velký důraz na čitelnost zdrojového kódu a syntax umožňuje programátorům vyjádřit složité koncepty na pár řádků kódu, což by nebylo možné u jazyků jako C++ nebo Java.

Python používá dynamické typování a podporuje několik programovacích paradigmat jako objektově orientované, imperativní, funkcionální a nebo procedurální.

3.2.3 RabbitMQ

RabbitMQ je open-source fronta zpráv napsaná v jazyce Erlang navržená pro robustní aplikace a implementuje protokol AMQP.

RabbitMQ funguje na bázi klient-server. Hlavním úkolem serveru je udržovat informace o vytvořených frontách zpráv v Mnesia databázi. Klient drží spojení se serverem a zpracovává nejaktuálnější zprávy.

V kontextu diplomové práce je RabbitMQ využíván pro asynchronní operace vykonávané na pozadí, respektive posílání e-mailů a dotazování uživatelských webových služeb s informacemi o nejnovějších přijatých textových zprávách.

3.2.4 Testování

Testování funkčnosti aplikačního rozhraní webové služby je velice důležitý proces celého vývoje. Při každé iteraci vývoje přibývají nové funkcionality celé aplikace, a tak je důležité otestovat, zda vše funguje podle zadání. Pokud by proces testování byl prováděn manuálně po každé iteraci, tak by vzniknul prostor pro chyby v testovacím procesu a zabralo by to mnoho času. Z tohoto důvodu byla aplikace pokryta automatizovanými testy.

Webová služba bude vyvíjena testovací metodikou TDD [15], která upřednostňuje nejprve vytvoření testu a až poté psaní zdrojového kódu funkcionality.

Jelikož všechny testy budou automatizovány, tak po každé změně v Git repozitáři se pošle notifikace CI serveru, který všechny testy spustí, a pokud se některý z testů nezdaří, bude vývojář zodpovědný za poslední změny notifikované pomocí e-mailu. CI server použitý pro tento projekt se jmenuje TravisCI a je to cloud služba, která je pro open-source projekty zcela zdarma.

3.3 USB modem

V zadání nebylo specifikováno, jaký USB modem má být použit. Mezi základní funkcionality musí nicméně patřit odesílání a přijímání textových zpráv. Byl vybrán a zakoupen běžně dostupný USB modem Huawei E3131 od firmy T-Mobile, který obsahuje SIM kartu s předplaceným kreditem.

Obrázek 3.5: USB model Huawei E3131 od firmy T-Mobile

3.4 Komunikace s USB modemem

Jedním ze stěžejních faktorů celé aplikace je návrh komunikace s USB modemem. Důraz je kladen na co nejjednodušší implementaci, použití a oddělitelnost od webové služby.

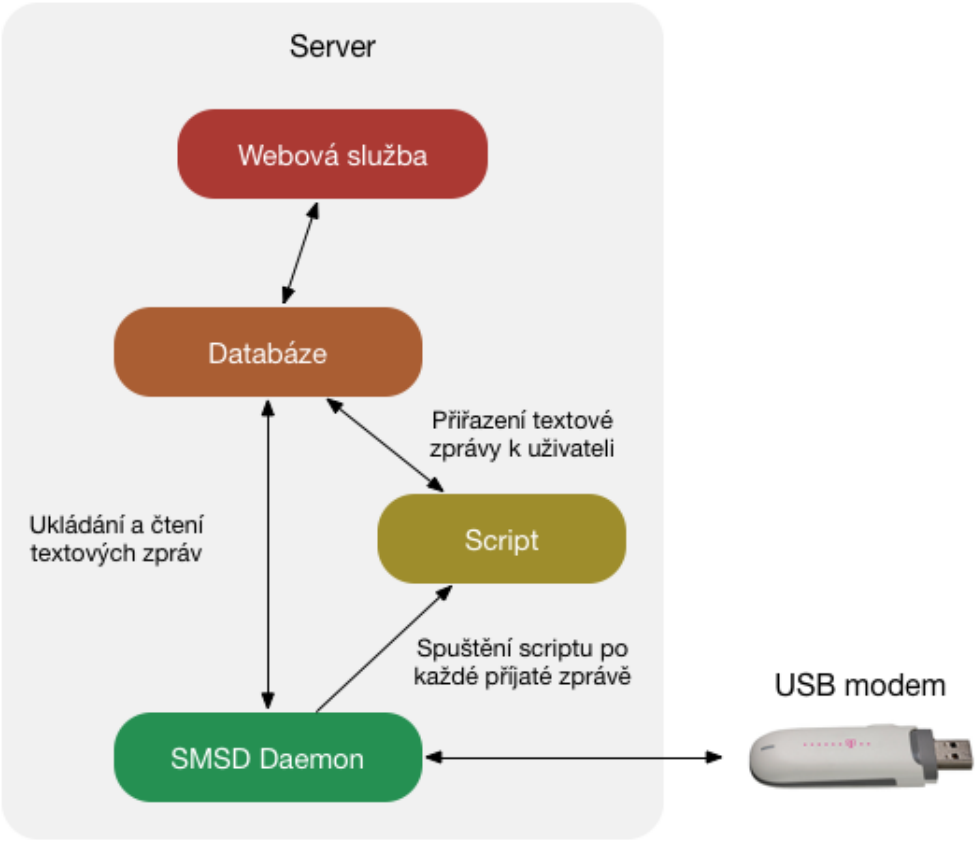
Byl vybrán a použit software Gammu [15] ve verzi 1.34.0. Tento software umožňuje spustit oddělitelný proces v pozadí *SMSD Daemon*, který neustále komunikuje s USB modemem pomocí *AT* příkazů. Proces je naprosto oddělitelný od webové služby, tudíž v případě výpadku jedné ze služeb není ohrožena žádná jiná.

SMSD Daemon má přímý přístup do databáze a provádí dvě hlavní činnosti:

- příjem textových zpráv
  - převezme textovou zprávu a uloží ji přímo do databáze,
- odesílání textových zpráv
  - sleduje frontu zpráv uložených v databázi, převezme zprávy k odeslání a odešle.

Při přijetí nové textové zprávy bude zpráva přímo uložena do databáze a zároveň spustí předem nastavený script, který může provést další operace nad novými přijatými textovými zprávami. Hlavním účelem tohoto scriptu bude přiřazování textových zpráv k uživatelským účtům na základě obsahu textové zprávy.

Webová služba je naprosto oddělena od procesu zpracování zpráv, pouze ukládá a čte data z databáze. To znamená velice jednoduché použití a zároveň bezpečnost v případě výpadku služby.



Obrázek 3.6: Schéma komunikace modemu s databází

3.4.1 Gammu

Gammu je open-source projekt, který umožňuje kontrolovat mobilní telefon nebo USB modem připojený k počítači nebo serveru. Je napsán v jazyce C a je postaven nad knihovnou *libGammu*. Rozhraní umožňuje přístup k rozsáhlým funkcím připojeného telefonu, nicméně podpora pro některé funkce závisí na typu telefonu.

Základní podporované funkce jsou:

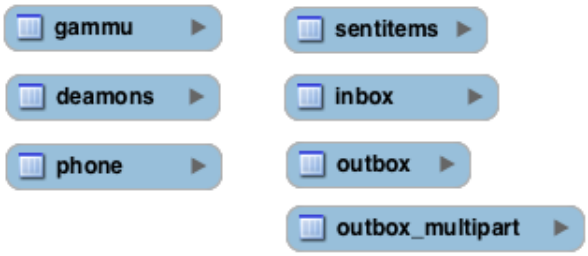
- výpis a řízení hovoru,
- čtení, zálohování a posílání textových zpráv,
- čtení multimediálních zpráv,
- čtení, import a export telefonního seznamu,
- informace o telefonu a připojení do telefonní sítě,
- přístup k souborovému systému telefonu.

3.5 Databáze

Důležitou částí celého projektu je databáze, jelikož je potřeba uchovávat data o všech textových zprávách a také o uživatelských účtech a jejich různých nastaveních.

Databáze by po případném rozšíření aplikace mohla sloužit jako zdroj informací pro statistické reporty odesílání a přijímání textových zpráv.

Jak už bylo řečeno v kapitole 3.4, veškeré textové zprávy budou automaticky odesílány a přijímány pomocí procesu, který poběží v pozadí a který bude automaticky ukládat nové zprávy a mazat již odeslané. Jelikož software Gammu, který spustí tento proces, již očekává specifickou strukturu databáze, kterou nelze jednoduše modifikovat, musí struktura vycházet z již existujícího ERD modelu. Gammu konfigurace umožňuje upravit některé SQL dotazy, které proces na pozadí využívá, což nám umožňuje v rámci možností upravit strukturu databáze.

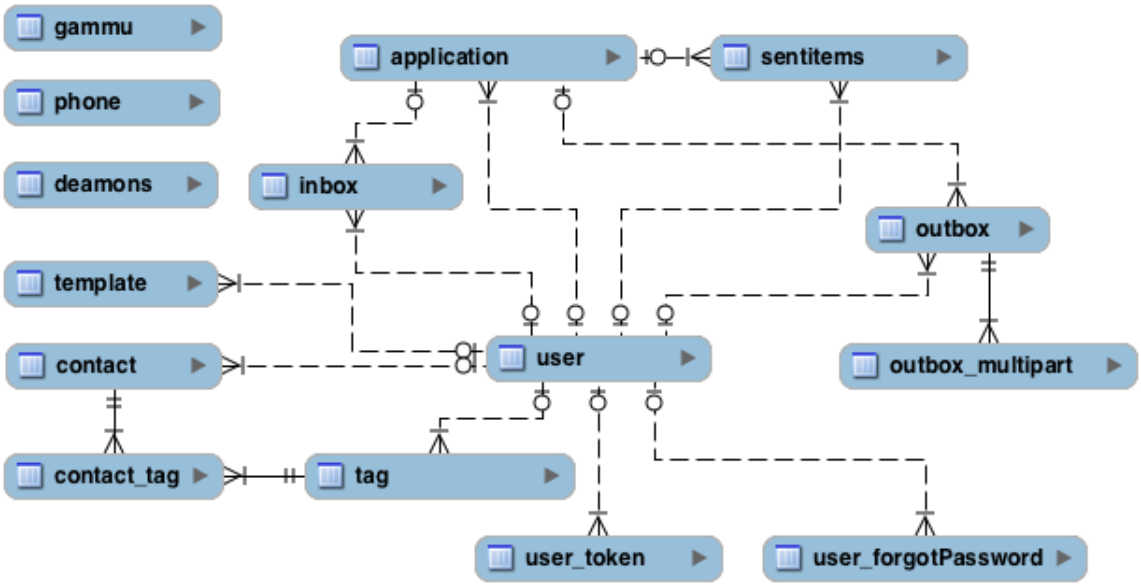


Obrázek 3.7: ERD model Gammu databáze

Nezbytné změny:

- vazby mezi základními entitami,
- změna názvu atributů entit,
- přidání vlastních atributů entit,
- přidání nových entit jako uživatelé, šablony, kontakty atd. a jejich vzájemné vazby.

Po úpravě byl vytvořen následující ERD model databáze zachycující jednotlivé vazby mezi entitami.



Obrázek 3.8: ERD model databáze

Jako databázový server byl vybrán server MariaDB 5.5 [11], a to z důvodu své licence a využití i na jiných projektech.

3.5.1 MariaDB

MariaDB je komunitou vyvíjená větev relační databáze MySQL, která je zdarma distribuována pod licencí GNU GPL. Vývoj je veden původními vývojáři MySQL, kteří byli nuceni vytvořit nezávislou větev pro akvizici MySQL firmou Oracle. Hlavním cílem MariaDB je vytvářet relační databázi, která bude plně kompatibilní s MySQL tak, aby vývojáři mohli nahradit MySQL bez sebemenších problémů.

MariaDB koresponduje mnohem více funkcemi než předchůdce MySQL:

- zvýšení výkonu,
- nové formáty uložiště dat (Aria, XtraDB, atd.),
- nové datové typy,
- dynamické a virtuální sloupce,
- multizdrojové replikace,
- globální transakční identifikátor,
- atd.

3.6 Vývoj s použitím apiary.io

Prvotní vývoj klientské aplikace bude probíhat bez jakékoliv podpory webové služby, jelikož pro návrh REST API rozhraní je potřeba znát specifikace klienta, respektive tok dat mezi klientem a webovou službou.

Implementace bez jakékoliv webové služby není jednoduchá, a proto pro naše potřeby bude použita dočasná webová služba, která nebude obsahovat logiku a bude pouze vracet statická data v těle HTTP odpovědi.

Pro vytvoření dočasného rozhraní bude použita služba apiary.io, která umožňuje navrhnout statické rozhraní bez nutnosti psát jakýkoliv kód. Po vytvoření dokumentu tzv. blueprint, což je specifická syntax pro popis aplikačních rozhraní, služba vygeneruje dočasné statické aplikační rozhraní, které může být použito pro vývoj klientské části. Jak už bylo řečeno, blueprint je dokument, který specifikuje dočasné aplikační rozhraní a zároveň také slouží jako dokumentace, jelikož syntax je odvozena od známého značkovacího jazyka Markdown.

4 Implementace

Celá tato kapitola popisuje implementační prostředky a kroky, včetně instalace potřebných komponent a jejich nastavení na serveru.

Cílem tohoto projektu bylo vytvoření webové služby, včetně rozhraní pro přístup z mobilních klientů, které by mělo splňovat REST API.

4.1 Server

Webová služba byla navržena a implementována jako serverová aplikace, a tudíž bylo potřeba zajistit potřebný server, na kterém bude možné celou aplikaci provozovat.

Jako operační systém serveru byla použita známá linuxová distribuce Debian Wheezy ve verzi 7.0. Z důvodu automatizace instalace všech potřebných softwarovýh součástí byl napsán script, který celý server předpřipraví do produkčního stavu. Tento script byl napsán za pomocí software Ansible a je ho možno zcela zpustit z lokálního prostředí bez nutnosti být vzdáleně připojen k serveru. Nicméně instalace a konfigurace těch nejdůležitějších součástí je vysvětlena níže.

```
ansible-playbook provisioning/machine.yml
```

Zdrojový kód 4.1: Příkaz pro spuštění instalace serveru

4.1.1 Python

Serverová aplikace je napsána v jazyce Python, většina současných linuxových distribucí mají Python ve verzi 2.7 už nainstalovanou jako součást systému, tudíž není potřeba instalovat samotný Python. Je nutné pouze nainstalovat potřebné balíčky pro instalaci a spuštění celé aplikace.

```
apt-get install python
apt-get install python-dev
apt-get install python-setuptools
apt-get install python-virtualenv
apt-get install python-software-properties
```

Zdrojový kód 4.2: Příkazy pro instalaci nezbytných Python balíčků

4.1.2 Supervisor

Celá aplikace bude vždy spuštěna na pozadí jako oddělený proces, který bude naslouchat na lokální IP adrese a portu. Jelikož se jedná o oddělený proces, je potřeba, aby nějaký další software udržoval monitoring. To znamená, pokud z nějakého důvodu bude proces ukončen, tak je nutné, aby byla aplikace znovu spuštěna. K tomuto účelu slouží software supervisor.

Instalace

Instalační balíčku je dostupný v oficiálním repositáři balíčkovacího systému distribuce.

```
apt-get install supervisor
```

Zdrojový kód 4.3: Příkaz pro instalaci supervisor balíčku

Nastavení

Součástí nastavení je nastavení linuxového uživatele, který byl vytvořen pro běh aplikace. Dále se nastavuje příkaz, který spustí aplikaci na pozadí a je vždy zavolán při restartu či při reloadu procesu. Veškeré chybové hlášky jsou ukládány do logu, který je naspecifikovaný jako součást konfiguračního souboru. Konfigurační soubor byl nastaven dle dokumentace softwaru [17].

```
[program:smsgw]
# script pro spusteni aplikace na pozadi
command = /home/smsgw/smsgw/gunicorn.sh production

# proces pobezi pod uzivatelen smsgw
user = smsgw

# log procesu
stdout_logfile = /home/smsgw/smsgw/log/supervisor.log

# presmerovat chybove hlasky to standartniho logu
redirect_stderr = true
```

Zdrojový kód 4.4: Nastavení supervisoru pro aplikaci

Spuštění aplikace se provádí přes příkazovou řádku pomocí klienta supervisoru.

```
supervisorctl start smsgw
```

Zdrojový kód 4.5: Příkaz pro spuštění aplikace na pozadí

4.1.3 Nginx

Aplikace musí být dostupná na základním webovém portu 80, nicméně samotná aplikace nemůže být na tomto portu vystavena. Pro vystavení aplikace na portu 80 by aplikace musela běžet pod uživatelem root (porty pod 1024 musí běžet pod uživatelem root), což není úplně ideální z bezpečnostního hlediska, a na serveru

bychom nemohli provozovat ani webové aplikace či služby na stejném portu. Z tohoto důvodu aplikace musí běžet na privátním portu a na portu 80 bude vystavena proxy, která přesměruje veškerou komunikaci z veřejného portu na ten privátní. Pro tento účel byl vybrán webový server a reverzní proxy nginx.

Instalace

Instalační balíček je dostupný v oficiálním repositáři balíčkového systému distribuce.

```
apt-get install nginx
```

Zdrojový kód 4.6: Příkaz pro instalaci nginx balíčku

Nastavení

Pro účely této práce nebylo potřeba měnit žádné základní nastavení webového serveru či reverzní proxy. Nicméně bylo třeba nastavit server tak, aby poslouchal na portu 80 a na předem nastavené doméně. Všechna komunikace, která bude přijata, bude ihned přesměrována na lokální IP adresu a port, na kterém běží serverová aplikace.

```
server {
    listen 80;
    server_name smsgw.alinet.cz;

    location / {
        proxy_pass http://127.0.0.1:5000/;
        proxy_redirect off;

        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Zdrojový kód 4.7: Nastavení nginx proxy

Poté je potřeba pouze restartovat nginx službu.

```
service nginx restart
```

Zdrojový kód 4.8: Příkaz pro restartování nginx procesu

4.1.4 MariaDB

Relační databáze MariaDB se používá pro ukládání veškerých persistentních dat o uživatelích, textových zprávách atd.

Instalace

Oficiální repositář distribuce bohužel neobsahuje nezbytný balíček. Bylo tak potřebovat přidat potřebný repositář dle dokumentace produktu [11].

```
apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 0
    xcbcb082a1bb943db
add-apt-repository \
  'deb http://mirror.vpsfree.cz/mariadb/repo/5.5/debian wheezy main'
apt-get update
apt-get install mariadb-server
```

Zdrojový kód 4.9: Instalace MariaDB

Nastavení

Po úspěšné instalaci je nezbytné spustit SQL dotazy pro vytvoření databáze a uživatele, pomocí kterého se bude aplikace k databázovému systému připojovat.

```
; vytvoreni uzivatele s heslem
CREATE USER 'smsgw_api'@'%' IDENTIFIED BY '<heslo>';

; vytvoreni databaze
CREATE DATABASE smsgw_api;

; prideleni prav uzivateli na novou databazi
GRANT ALL PRIVILEGES ON `smsgw_api`.* TO 'smsgw'@'%' WITH GRANT OPTION;
```

Zdrojový kód 4.10: Dotazy pro přípravu databázového serveru

4.1.5 Gammu

Software pro zařízení a komunikaci s USB modemem.

Instalace

Distribuce neobsahuje potřebný balíček pro instalaci softwaru. Instalaci je nutné provést ze zdrojových kódů. Nejdříve je potřeba stáhnout, neboli naklonovat zdrojové kódy z oficiálního Git repositáře. Poté je nezbytné zvolit správnou revizi kódu pro získání požadované verze softwaru. Následuje samotná instalace a kompilace zdrojových kódů.

```
git clone https://github.com/gammu/gammu.git gammu
cd gammu/
git checkout 1.34.0
./configure
make
make install
```

Zdrojový kód 4.11: Příkazy nezbytné k instalaci Gammu

Nastavení

Gammu automaticky komunikuje s databází a ukládá přijaté a čte zprávy k odeslání, které jsou zařazené ve frontě. Z tohoto důvodu musí být součástí konfigurace i údaje pro připojení k databázovému serveru. Jelikož dokáže komunikovat s databází, očekává už předem danou strukturu tabulek v databázi, které lze částečně upravit, ale ne zcela změnit. V konfiguračním souboru lze upravit veškeré SQL dotazy, která Gammu používá pro své operace.

Protože daemon komunikuje přímo s USB modemem, je třeba nastavit atribut, který specifikuje, na kterém sériovém portu se modem nachází, a typ komunikace. Dále je nutno nastavit identifikátor telefonu či modemu a PIN kód v případě, že SIM karta je zabezpečena.

```
[gammu]
device = /dev/ttyUSB0
connection = at

[smsd]
phoneid = tmobile-huawei
PIN = 1234
service = sql
driver = native_mysql
user = <uzivatel>
password = <heslo>
pc = 127.0.0.1
database = smsgw_api

deliveryreport = yes
logfile = /home/smsgw/.smsdlog
debuglevel = 2
```

Zdrojový kód 4.12: Nastavení Gammu

4.1.6 RabbitMQ

RabbitMQ je fronta úloh, která je použita pro asynchronní operace, jako je odesílání emailů nebo dotazování uživatelovy webové služby s informací o přijaté zprávě.

Instalace

Instalační balíček je dostupný v oficiálním repositáři balíčkovacího systému distribuce.

```
apt-get install rabbitmq-server
```

Zdrojový kód 4.13: Příkaz pro instalaci RabbitMQ balíčku

Nastavení

Jak bylo řečeno v kapitole 3.2.3, RabbitMQ je frontovací systém, který může spravovat více front ve více virtuálních jmenných prostorech, které je potřeba vytvořit. Dále je třeba vytvořit uživatele, kterým se bude aplikace připojovat a pomocí

kterého bude číst a zapisovat data. Zdrojový kód 4.14 obsahuje příkazy pro vytvoření uživatele, virtuálního jmenného prostoru a nastavení oprávnění.

```
// vytvoreni uzivatele
rabbitmqctl add_user smsgw_api <heslo>

// vytvoreni virtualniho hosta
rabbitmqctl add_vhost smsgw_api

// prirazeni prav novemu uzivately k novemu hostu
rabbitmqctl set_permissions -p smsgw_api smsgw_api ".*" ".*" ".*"
```

Zdrojový kód 4.14: Příkazy pro přípravu RabbitMQ serveru

4.1.7 Node

Klientská aplikace je napsána v jazyce JavaScript za využití knihovny React a je spuštěna výhradně jen ve webovém prohlížeči. Pro pohodlný vývoj aplikace v jazyce JavaScript a možnosti využívat balíčkovací systém pro nezbytné knihovny je potřeba nainstalovat platformu Node, což umožňuje spouštět javascriptové aplikace či skripty na straně serveru. Součástí instalace platformy je i balíčkovací systém NPM, který obsahuje veškeré potřebné knihovny pro běh klientské aplikace.

Instalace

Distribuce neobsahuje balíček pro instalaci platformy. Nicméně dle dokumentace platformy [18] je již připravený skript pro snadnou instalaci.

```
curl -sL https://deb.nodesource.com/setup | bash -
apt-get install nodejs
```

Zdrojový kód 4.15: Příkazy pro instalaci platformy Node

Dále je potřeba instalace nezbytných globálních modulů pro účely testování, podpory ECMAScript 6 či vytváření a kompilace vícezdrojových souborů do jednoho.

```
npm install -g babel
npm install -g gulp
npm install -g jest-cli
npm install -g webpack
```

Zdrojový kód 4.16: Příkazy pro instalaci nezbytných globálních Node modulů

4.2 Webová služba

Serverová aplikace s webovou službou využívající REST API rozhraní byla implementována za pomocí mikroframeworku Flask 0.10.1 [5] v jazyce Python a je integrován s HTTP WSGI serverem Gunicorn 19.0.0, který přijímá HTTP požadavky a předává je samotné aplikaci.

4.2.1 Komunikace

Flask mikroframework poskytuje jen základní funkčnost pro vývoj webových aplikací či služeb. Tudíž je uskutečnitelné, pokud bychom chtěli vytvořit REST API jen na základním frameworku. Nicméně veškerý kód by byl strukturován do funkcí a veškeré URL pro routovací tabulku by musely být vždy nadefinovány. Mikroframework lze rozvíjet o rozšíření, která jsou open-source a lze je instalovat pomocí balíčkovacího systému pip. Pro snadnější vývoj a lepší strukturu kódu bylo vybráno rozšíření Flask-Classy 0.6.10, které implementuje pohledy založené na třídách a automaticky generuje URL pro routovací tabulku. Následně je každá třída a metoda vázána na konkrétní URL a je zavolána po přijetí požadavku.

```
class OutboxResource(FlaskView):

    route_base = '/outbox'

    def index(self):
        return response({})

    @route('/custom', methods=['GET'])
    def custom(self):
        return response({})
```

Zdrojový kód 4.17: Základní ukázka třídy Flask-Classy v jazyce Python

Flask-Classy koresponduje se základními nadefinovanými CRUD metodami, pro které není potřeba nadefinovat HTTP metodu a URL.

- index - GET /
- create - POST /
- get - GET /<id>
- updated - PUT /<id>
- delete - DELETE /<id>

To znamená, že pro metodu *index* není potřeba specifikovat další informace. Nicméně pro metodu *custom*, která není předdefinována, je potřeba naspecifikovat HTTP metodu a URL.

Po přijetí požadavku na URL *http://SERVER:PORT/api/1.0/outbox/custom* je tento požadavek zpracován metodou *custom* a následně vrácena odpověď ve fromátu JSON.

Tímto způsobem je implementováno celé REST API rozhraní poskytující funkčnost pro klientské požadavky.

4.2.2 Formát odpovědi

Jak již bylo zmíněno, tak veškerá komunikace mezi klienty a aplikačním rozhraním se uskutečňuje ve formátu JSON, a je velice důležité, aby formát odpovědi měl

neměnný, standardní formát, který může klient zpracovat. Pokud by formát byl pro každé URL rozhraní rozdílný, bylo by velmi těžké zpracovávat odpovědi na straně klienta či aplikací třetích stran.

Formát byl navržen tak, aby veškeré chybové a nechybové odpovědi byly maximálně rozšiřitelné. Struktura odpovědi se skládá ze dvou částí:

- *meta* – objekt, který obsahuje meta data odpovědi jako HTTP status code, chybovou nebo nechybovou zprávu a je rozšiřitelný o jakékoliv další informace,
- *data* – atribut obsahuje samotná data odpovědi, může obsahovat datový typ objekt nebo kolekce závisející na typu dat, například data o uživateli či list všech uživatelů.

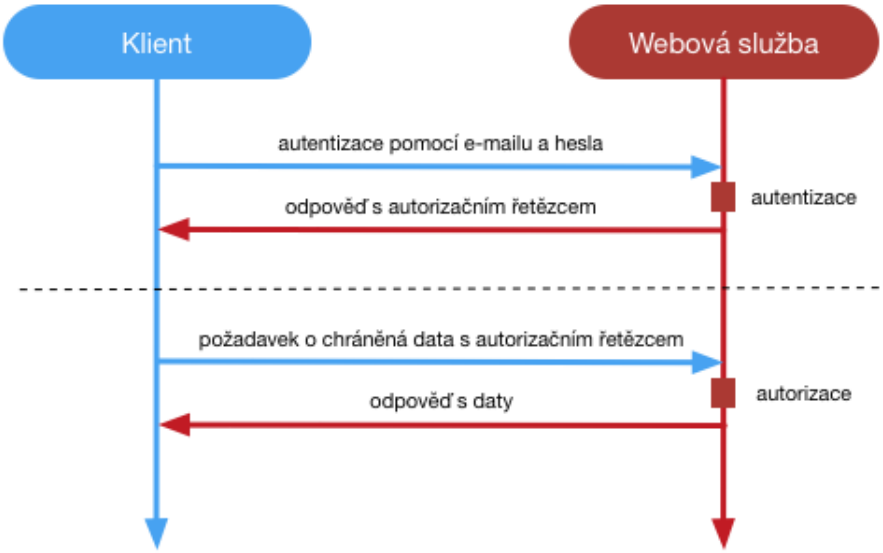
```
{
    "meta": {
        "status": 200,
        "message": "OK."
    },
    "data": {
        "email": "hi@vojtech.me",
        "firstName": "Vojtech",
        "lastName": "Bartos"
    }
}
```

Zdrojový kód 4.18: Ukázka odpovědi webové služby ve formátu JSON

4.2.3 Autentizace a autorizace

Autentizace uživatele se provádí na základě klientovy e-mailové adresy a hesla, které musí obsahovat minimálně 16 znaků. Ověřování probíhá na základě dat, která jsou uložena v databázi. Porovnává se emailová adresa a zašifrované heslo. Ukládání hesel v databázi v čitelné podobě je obrovské bezpečnostní riziko a z toho důvodu je při registraci heslo zašifrováno pomocí šifrovací funkce bcrypt.

Bcrypt je šifrovací funkce, která je založena na šifře Blowfish a zahrnuje v sobě kryptografickou sůl. O autentizaci uživatelů se stará REST API rozhraní na URL adrese *http://SERVER:PORT/api/1.0/auth/login/* a přijímá požadavky pomocí HTTP metody POST.



Obrázek 4.1: Schéma autentizace a autorizace

Po úspěšné autentizaci uživatele se zjistí, zda existuje autorizační řetězec pro daného klienta. Pokud má již uživatel existující řetězec, není třeba vytvářet nový, ale pokud řetězec neexistuje, bude vytvořen a poslán zpět klientovi jako součást HTTP odpovědi. Zde je možnost mít více autorizačních řetězců pro více klientů, což umožňuje uživatelům být přihlášen na více zařízeních najednou.

Autorizační řetězec je ve formátu UUID [24], který je vždy unikátní a umožňuje jednoznačnou identifikaci uživatele. Tento řetězec slouží pro autorizaci veškeré klientovy komunikace s aplikačním rozhraním. REST API rozhraní, které vyžadují autorizaci, očekávají řetězec jako součást HTTP hlavičky požadavku ve formátu *Authorization: Token <retezec UUID>*. V případě, že řetězec není součástí hlavičky nebo řetězec není v databázi, je poslána odpověď klientovi se statusem 401 a chybovou hláškou. Při úspěšné autorizaci rozhraní zpracuje požadavek a vrátí odpověď obsahující požadovaná data.

```
GET /api/1.0/users/@me/ HTTP/1.1
Host: smsgw.alinet.cz
Accept: application/json
Authorization: Token 422511f6-008e-4a00-a15e-23fbac6a9c4c
Content-Type: application/json
```

Zdrojový kód 4.19: Ukázka HTTP POST požadavku s autorizačním řetězcem

4.2.4 Validace

Veškerá data, která aplikační rozhraní přijímá, musí být zkontrolována neboli zvalidována. Validace vstupu je velmi důležitá, protože logika rozhraní očekává a zpracovává určitá data. Pokud by data byla obdržena ve špatném formátu či by některá data nesplňovala nezbytná pravidla, nesmí být tato data zpracována a klient musí být vyrozuměn informací, že data, která poslal pro zpracování, nejsou správná.

Rozhraní bylo navrženo tak, aby přijímala veškerý vstup ve formátu JSON. Pro tento formát existuje standard, který se jmenuje *JSON schema*. Tento standard umožňuje popsat, jak má vypadat validní formát celého vstupu ve formátu JSON. Pomocí jednoduché a strojově čitelné dokumentace je možné navrhnout formát vstupu pro jednotlivá rozhraní. Dokumentace umožňuje určit strukturu celého JSON dokumentu, nastavit pravidla pro atributy jako datový typ, minimální a maximální délku, chybové hlášky či zda je atribut povinný.

```
schema = {
    "description": "Schema for the user POST endpoint",
    "type": "object",
    "method": "POST",
    "required": ["email", "password", "firstName", "lastName"],
    "additionalProperties": False,
    "properties": {
        "email": {
            "type": "string",
            "format": "email",
            "maxLength": 128
        },
        "firstName": {
            "type": "string",
            "maxLength": 16
        },
        "lastName": {
            "type": "string",
            "maxLength": 16
        },
        "company": {
            "type": "string",
            "maxLength": 16
        },
        "password": {
            "type": "string",
            "minLength": 16
        }
    }
}
```

Zdrojový kód 4.20: JSON schema specifikace pro přidání nového uživatele

V jazyce Python již existuje knihovna, která implementuje tento standard. Jmenuje se *jsonschema* a pro tento projekt byl využit ve verzi 2.3.0.

Po přijetí požadavku na rozhraní, které se stará o vytváření nových uživatelů, se extraktuje vstup a použije se výše zmíněná knihovna pro validaci.

```
import jsonschema
import flask from request

jsonschema.validate(request.json, {
    # dokumentace a specifikace vstupu
})
```

Zdrojový kód 4.21: Validace těla požadavku jazyce Python

Při neúspěšné validaci knihovna vyhodí výjimku *jsonschema.ValidationError* typu, která je odchycena chybovým kontrolerem. Kontroler zpracuje výjimku a pošle chybovou odpověď HTTP 403 s chybovou zprávou, která obsahuje informace, z jakého důvodu vstup nesplňuje pravidla.

4.2.5 Model

Pro lepší práci a komunikaci s databází byla vybrána knihovna *SQLAlchemy* 0.9.7. Takto knihovna je open-source SQL nástroj a ORM, který umožňuje vývojářům pracovat na úrovni objektů místo přímo s SQL jazykem. *SQLAlchemy* umožňuje nadefinovat pomocí tříd model, který reprezentuje tabulky v databázi, a specifikovat atributy, které model/tabulka obsahuje. Mezi modely je možné vytvářet relace, které jsou do databáze přeneseny v podobě SQL relace mezi tabulkami. Součástí knihovny je také nástroj, který umožňuje vygenerovat strukturu celé databáze na základě zaregistrovaných modelů.

```
class Tag(BaseModel, DateMixin):
    """ Tag model """

    id = db.Column(mysql.INTEGER(10, unsigned=True), primary_key=True)
    userId = db.Column(mysql.INTEGER(10, unsigned=True),
                       ForeignKey('user.id'))
    uuid = db.Column(mysql.CHAR(36), unique=True, nullable=False,
                     default=generate_uuid)
    reference = db.Column(db.String(32), nullable=False)
    label = db.Column(db.String(32), nullable=False)
    note = db.Column(db.String(255))
```

Zdrojový kód 4.22: SQLAlchemy Tag model v jazyce Python

4.2.6 Fronta úloh

Frontu úloh webová služba využívá zejména pro operace jako odesílání e-mailu nebo informování uživatelovy webové služby o přijaté textové zprávě. Nicméně fronta by měla být i v budoucnu použita pro více operací, které je dobré zpracovat asynchronně tak, aby nezdržovaly vrácení odpovědi klientovi.

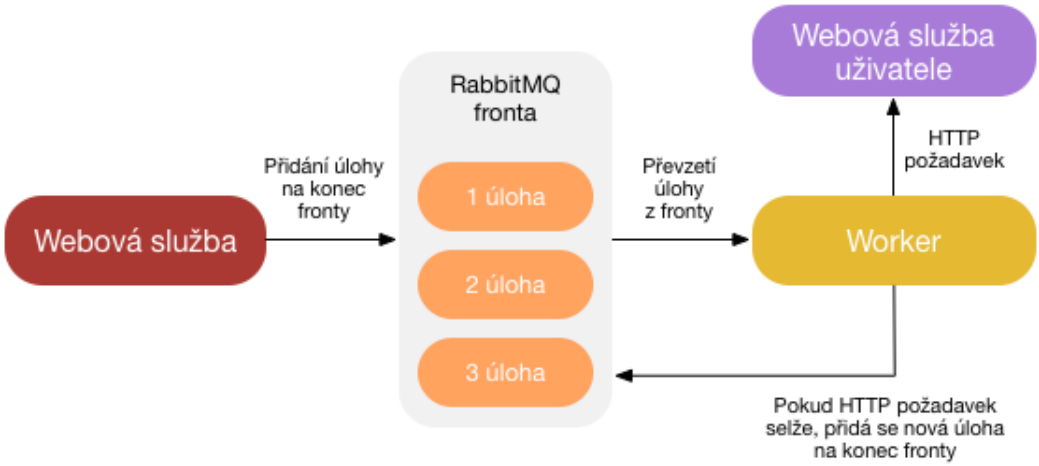
Pro frontovací systém byl vybrán framework *Celery* ve verzi 3.1.17. *Celery* je distribuovaná fronta úloh, která umožňuje spuštění asynchronních úloh na pozadí.

Jak už bylo popsáno v kapitole 3.2, jako uložiště pro fronty a úlohy byl vybrán a použit software RabbitMQ.

Implementace frontovacího systému se skládá ze dvou částí, které běží ve vlastním procesu:

- webová služba - vytváří a ukládá nové úlohy pro asynchronní zpracování,
- worker - oddělený proces, který se stará o převzetí a vykonání úlohy v pozadí.

Webová služba se stará pouze o vytváření a ukládání úloh pro pozdější zpracování. Každá nová úloha je přidána na konec fronty, a tím to pro webovou službu končí.



Obrázek 4.2: Schéma fronty operací

Worker běžící na pozadí je naprosto oddělený proces od webové služby. Udržuje komunikaci s uložištěm RabbitMQ, přebírá a zpracovává úlohy, které jsou na začátku fronty. Po zpracování úlohy zjistí, zdali je ve frontě další úloha ke zpracování, a pokud ano, tak ji převezme a začne zpracovávat. Pokud je fronta prázdná, worker stále udržuje komunikaci s RabbitMQ a čeká na novou úlohu.

Jednou z úloh, které se zpracovávají na pozadí, je odesílání e-mailových zpráv. Důvod, proč se tato operace provádí na pozadí, je, že odeslání HTTP odpovědi klientovi nezávisí na tom, jestli byl e-mail odeslán, či nikoliv. E-mail může být odeslán o pár sekund déle a vše bude naprosto v pořádku. Třída, která definuje úlohu, se jmenuje *MailTask* a obsahuje jednu metodu *run*, která se stará o vykonání operace odeslání e-mailové zprávy, a jeden atribut *routing_key*, který definuje cílovou frontu úlohy. Metoda *run* obsahuje argumenty, které jsou uloženy jako součást zprávy v uložišti RabbitMQ, a logiku celé úlohy. Logika úlohy je v tomto případě velmi jednoduchá, jelikož veškeré e-maily jsou vytvořeny dle předdefinované šablony a poté je obsah předán pouze k odeslání.

```
class MailTask(BaseTask):
    """ Task for sending emails """

    routing_key = 'mails'

    def run(self, to, template, params, sender=None, **kwargs):
        # render subject and body from template files
        subject = render_template("{0}-subject.html".format(template),
                                  **params)
        body = render_template("{0}.html".format(template), **params)

        # create message
        msg = Message(subject)
        msg.sender = sender or current_app.config \
                                            .get('DEFAULT_MAIL_SENDER')
        msg.recipients = to
```

```
msg.body = body

# send mail
mail.send(msg)

return {'to': to,
        'template': template,
        'params': params}
```

Zdrojový kód 4.23: Úloha pro odesílání e-mailů napsaná v jazyce Python

Poslední implementovanou úlohou je úloha specifikovaná třídou *CallbackTask*. Ta zajišťuje komunikaci s uživatelovou webovou službou. Informuje webovou službu třetí strany o nově přijaté zprávě pro uživatelovu aplikaci. Webová služba třetí strany musí splňovat REST API rozhraní, jelikož požadavek bude poslán protokolem HTTP pomocí metody POST. Pokud odpověď od webové služby bude chybová, úloha vytvoří novou úlohu, která se pokusí o zaslání požadavku za určitý časový interval. Každá úloha má maximálně 3 opravné pokusy. Pokud všechny pokusy selžou, úloha zanikne. Každý pokus je zpožděn z důvodu možnosti dočasného výpadku webové služby uživatele.

- 1 opravný pokus - 5 minut,
- 2 opravný pokus - 10 minut,
- 3 opravný pokus - 15 minut.

4.2.7 Testování

Jak už bylo řečeno v kapitole 3.2.4, webová služba je pokryta automatickými testy, které pomáhají udržovat stabilitu služby.

Jako knihovna usnadňující testování byla vybrána knihovna *nose* ve verzi 1.3.3. Tato knihovna rozšiřuje základní framework *unittest* pro snadnější testování a spouští se z příkazové řádky pomocí jednoduchého příkazu *nosetests*.

Veškeré testy se automaticky spouští i na portále TravisCI, který dostane notifikaci pokaždé, když nastane změna v Git repositáři hostovaném na github.com. Nicméně bylo potřeba nastavit *travis.yml* konfigurační soubor dle dokumentace [16]. TravisCI automaticky přečte konfiguraci z repositáře a nastaví prostředí dle potřeb.

Jak je možno vidět ve zdrojovém kódu 4.24, součástí nastavení testovacího prostředí pro tuto webovou aplikaci je:

- požadovaná verze programovacího jazyka Python 2.7,
- nastavení prostředí,
- instalace python modulů,
- vytvoření databáze,
- příkaz pro spuštění testů.

```
services:
  - rabbitmq

language: python
python:
  - 2.7

env:
  - SMSGW_ENV=ci

install: "pip install -r requirements.txt"
before_script:
  - mysql -e 'create database smsgw_test;'

script: nosetests
```

Zdrojový kód 4.24: TravisCI YAML konfigurace

Kvůli snadnějšímu vytváření testů byla vytvořena základní třída *SmsgwIntegrationTestCase* jako předchůdce pro všechny testy. Tato třída obsahuje metodu *setUp*, která je zavolána před každým testem. Metoda zajistí vyprázdnění celé testovací databáze a nahrání dat, která jsou nezbytná pro každý test. Každý test musí pracovat jenom s daty, která nezbytně potřebuje, a při spuštění dalšího testu se databáze musí vyprázdnit z důvodu nezasahování do jiného testu.

```
class SmsgwIntegrationTestCase(FlaskTestCase):

    def create_app(self):
        return factory.create_app(name="smsgw_testing",
                                  env=os.environ.get('SMSGW_ENV'))

    def setUp(self):
        super(SmsgwIntegrationTestCase, self).setUp()
        # flask instance
        self.app = self.create_app()

        # prepare DB. first dump DB afterwards re-create it,
        # because if tests gonna fail in DB stay latest
        # state
        db.drop_all()
        db.create_all()

        # import base datesets
        self.user = User(**datasets.user.USER)
        self.user.tokens = [UserToken(agent="Command-line")]
        db.session.add(self.user)
        db.session.commit()
        db.session.refresh(self.user)
```

Zdrojový kód 4.25: Základní integrační TestCase v jazyce Python

4.3 Klient

Tato kapitola popisuje implementaci klientské části aplikace a uvádí hlavní prvky implementace moderních webových aplikací, které byly vyvinuty v programovacím jazyce JavaScript.

4.3.1 Build

Klientská aplikace využívá CommonJS a ECMAScirpt 6, které nejsou podporovány ve všech moderním prohlížečích. Z toho důvodu musel být zaveden proces kompilace a buildu. Veškerý zdrojový kód musí být transformován do jazyka JavaScript vycházejícího ze standardu ECMAScript 5 tak, aby bylo možné spustit kód v moderních prohlížečích. Po transformaci je stále zdrojový kód rozdělen do modulů dle CommonJS syntaxe, a tak je potřeba vytvořit jeden spustitelný soubor, který bude spuštěn v prohlížeči.

Pro kompilaci a sestavení spustitelného souboru byl použit NPM modul *webpack* ve verzi 1.12.0. Tento modul sestavuje moduly do jednoho spustitelného balíku, který je možno spustit v prohlížeči. Rovněž je schopen transformace zdrojového kódu, sestavování balíčků pro různé zdroje jako jsou kaskádové styly, obrázky a také minifikaci zdrojového kódu pro zredukování velikosti výsledného souboru.

```
{
  cache: (isDevelopment),
  watch: (isDevelopment),
  devtool: (isDevelopment) ? 'eval' : '',
  output: {
    filename: "js/[name].js"
  },
  module: {
    preLoaders: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        loader: "eslint-loader"
      }
    ],
    loaders: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        loader: 'babel-loader',
        query: { stage: 2 }
       }
    ]
}
```

Zdrojový kód 4.26: Konfigurace modulu webpack v jazyce JavaScript

Konfigurace pro modul *webpack* obsahuje:

- souborovou adresu, která určuje, kde má být vytvořen finální spustitelný balíček,

- preprocesor *jslint-loader*, který kontroluje správnost formátování a detekuje chyby v kódu,
- procesor *babel-loader*, který transformuje JavaScript standard ECMAScript 6 do standardu ECMAScript 5.

Po spuštění nastaveného *webpack* modudulu se používá task manager NPM modul *gulp* ve verzi 3.9.0 a vyvolává se pomocí příkazu spuštěného z příkazové řádky.

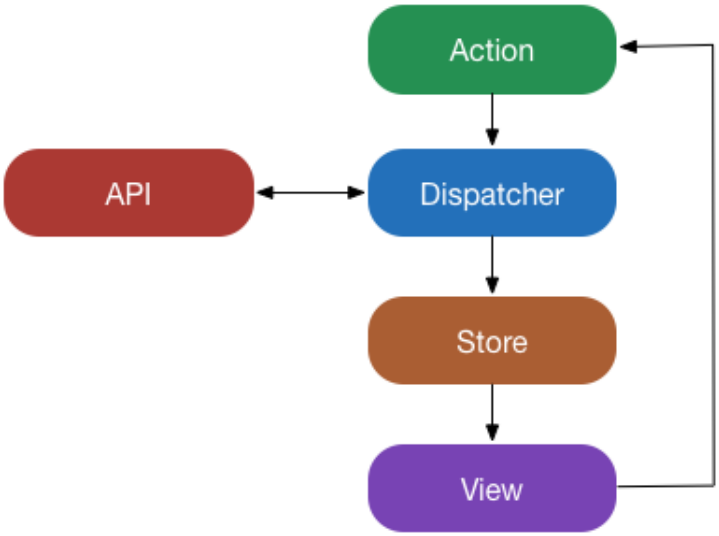
```
gulp -e production
```

Zdrojový kód 4.27: Příkaz pro sestavení výsledného souboru

4.3.2 Flux

Pro vytváření uživatelských rozhraní byl použit framework React, který se považuje za pomyslné View z navrhovaného vzoru Mode-View-Controller. Avšak využití navrhovaného vzoru MVC s frameworkem React není doporučováno z důvodu škálovatelnosti. Doporučováno je pak využít Flux architekturu.

Nicméně tato architektura, která byla popsána v kapitole 3.1.2, byla použita jako předloha pro strukturu aplikace a byla upravena dle potřeb. Struktura je rozdělena do komponent dle kontextu operací, které vykonává, např. uživatelé, aplikace, kontakty, tagy atd. Každá z těchto komponent obsahuje *Action*, *Store* a *View*, které pracují a komunikují v kontextu celé komponenty.



Obrázek 4.3: Flux architektura s API

Action obsahuje funkce neboli metody, které umožňují předávání dat do *Dispatcheru*. Součástí může být i logika, která upraví a předá data. Avšak ve většině případů daná akce připravuje požadavek pro aplikační rozhraní serveru, které posléze předá do *Dispatcheru*.

```
/**
 * Fetch current user
 */
export function getLoggedIn() {
  let request = api.get(users.get('@me'), { token: _token });

  return Dispatcher.dispatch(getLoggedIn, request);
}
```

Zdrojový kód 4.28: Akce pro získání informací o přihlášeném uživateli v jazyce JavaScript

Dispatcher je v podstatě manažer všech procesů. Je to centrální prvek pro celou aplikaci. Přijímá *Action* a odesílá data do všech registrovaných *Store*. Základní funkcionalita metody *dispatch* byla rozšířena o možnost zpracovávat předpřipravené požadavky pro aplikační rozhraní serveru. To znamená, že pokud metoda *dispatch* obdrží předpřipravený požadavek pro aplikační rozhraní serveru, tak vykoná tento požadavek a po obdržení odpovědi odešle přijatá data do všech registrovaných *Store*.

```
class Dispatcher extends flux.Dispatcher {

  dispatch(action, data) {
    if (data && typeof data.then === 'function')
      return this.dispatchAsync(action, data);
    else
      super.dispatch({action, data});
  }

  dispatchAsync(action, promise) {
    const name = action.toString();

    return promise.then(body => {
      const {data, meta} = body;

      super.dispatch({action, data, meta});

      return body;
    }).error(err => {
      super.dispatch({
        action: flash,
        data: { text: err.message, type: 'danger' }
      });

      throw err;
    });
  }
}
```

Zdrojový kód 4.29: Dispatcher v jazyce JavaScript

Store je kontejner pro aplikační stav a logiku, kde je zaregistrovaná callback funkce pro *Dispatcher*. *Dispatcher* volá všechny callback funkce všech zaregistrovaných *Store* a je už na každém z nich, zda data přijme, či nikoliv. Data, která *Store* přijme, se skládají z dat ke zpracování a jména funkce či metody, která byla vy-

volána v *Action*. Díky znalosti původce akce je velice jednoduché rozpoznat, která data mají být zpracována a která ne.

```
/**
 * Registerting to dispatcher
 */
export const dispatchToken = Dispatcher.register(({action, data}) => {
  switch (action) {
    case actions.getAll:
      \\ ...
      break;

    case actions.update:
    case actions.get:
      \\ ...
      break;

    case actions.remove:
      \\ ...
      break;
  }
});
```

Zdrojový kód 4.30: Registrace Store v Dispatcheru aplikace naprogramovaném v jazyce JavaScript

View už jsou samostatné React komponenty, které se vykreslují dle vnitřního stavu či předaných atributů. Oproti základní architektuře Flux nemusí každá komponenta, která závisí na stavu *Store*, naslouchat změnám dat. Pouze root komponenta naslouchá změnám všech specifikovaných *Store* a pokud dojde ke změně dat v jakémkoliv uložišti, bude stav hlavní komponenty změněn. Hlavní komponenta předává všem nižším komponentám reference na všechna *Store* pomocí atributu. To znamená, že při každé změně stavu hlavní komponenty všechny nižší komponenty dostanou nové atributy a vygenerují nový DOM.

4.3.3 Local storage

Po úspěšné autentizaci uživatele pomocí přihlašovacího formuláře klient obdrží autorizační řetězec, který je použit pro veškerou komunikaci s aplikačním rozhraním. Tento řetězec musí být persitentně uložen v klientovi a jako uložiště bylo vybráno local storage.

Podpora pro uložiště local storage je dnes ve všech moderních prohlížečích nevyjímaje Internet Explorer verze 8. Data jsou persistována pro webovou doménu, kde je aplikace spuštěna, tudíž data nejsou přístupná pro ostatní aplikace. Data uložená v uložišti jsou reprezentována jako datová struktura slovníku.

Pro komunikaci s datovým uložištěm byl použit NPM modul *localStorage* ve verzi 1.0.3. Modul zajišťuje základní aplikační rozhraní jako přidání a odebrání hodnoty pro klíč nebo vymazání celého uložiště.

```
import localStorage from 'localStorage';
```

```
let _token = localStorage.getItem('token');
```

Zdrojový kód 4.31: Uložení autorizačního řetězce v local storage v jazyce JavaScript

4.3.4 Komunikace

Veškerá komunikace s aplikačním rozhraním se provádí pomocí AJAX, respektive AJAJ techniky. Komunikace probíhá pomocí asynchronních požadavků, ve kterých se odpověď očekává ve formátu JSON.

Pro vytváření asynchronních požadavků byl použit NPM modul *superagent* ve verzi 1.2.0. Modul je možno použít jak v prohlížeči, tak i v Node aplikaci běžící na serveru. Aplikační rozhraní modulu je velmi intuitivní a velice jednoduché k použití.

Modul *superagent* byl použit jako základ pro vytvoření vlastního modulu *api*, který řídí veškerou komunikaci s aplikačním rozhraním.

```
import * as api from '../api'
import localStorage from 'localStorage';

const promise = api.post('... url rozhrani', {
  token: localStorage.getItem('token'),
  data: {
    // data pozadavku jako informace o novem uzivately
  }
});

promise.then({meta, data} => {
  // `data` obsahuje data prijmute od API
  console.log(data);
}).error(err => {
  // v pripade chyby
  console.log(err);
});
```

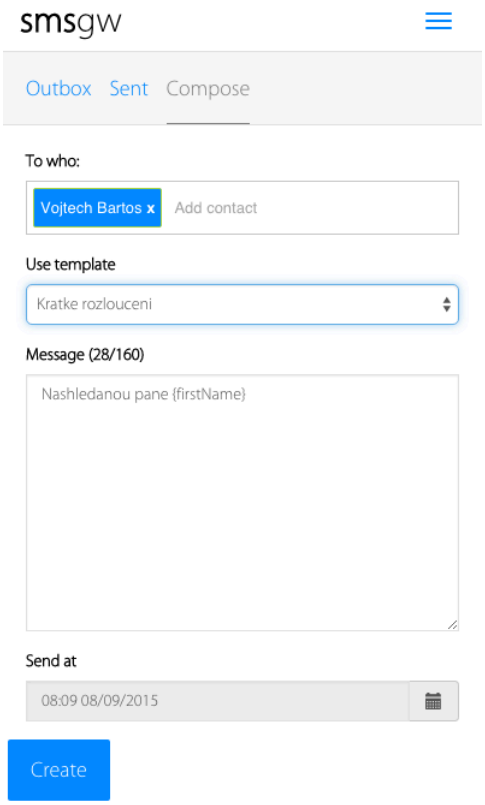
Zdrojový kód 4.32: Ukázka komunikace s webovou službou v jazyce JavaScript

Nový modul implementuje základní funkce nezbytné pro vytváření validních požadavků pro aplikační rozhraní.

- Autorizační řetězec – přidává řetězce do hlavičky každého HTTP požadavku.
- Implementace návrhového vzoru Promise [19] – zavádí možnost využití navrhovaného vzoru Promise, který pomáhá reprezentovat operaci, která ještě nebyla dokončena, ale očekává se její dokončení v blízké budoucnosti. Použit byl NPM modul *bluebird* ve verzi 2.9.6.
- Content-Type: application/json – nastavuje typ obsahu odpovědi.
- Accept: application/json – nastavuje povolený typ obsahu odpovědi.
- Zpracování chyb – pokud aplikační rozhraní skončí chybou, ta bude zpracována, analyzována a reprezentována pomocí výjimky typu *ApiError*.

4.3.5 Responsivní design

Jelikož se jedná o multipratformní HTML5 webovou aplikaci, je třeba zajistit, aby uživatelská přívětivost byla stejná v různých rozlišeních obrazovek na různých zařízeních. Způsob docílení nejlepší uživatelské přívětivosti se nazývá responsivní design.



Obrázek 4.4: Vytvoření textové zprávy na mobilním zařízení do 600px

Responsivní design je způsob stylování HTML dokumentů, které zaručí, že zobrazení stránky bude optimalizováno pro různá zařízení. Díky *Media Queries*, které jsou zahrnuty ve specifikaci *CSS 3*, lze rozpoznat vlastnosti zařízení cílového zařízení.

Veškerý design implementovaný na klientovi je přizpůsoben třem různým rozlišením:

- Do šířky *600 px* – mobilní zařízení typu smart-phone jako iPhone nebo Samsung Galaxy S4, viz obrázek 4.4.
- Do šířky *1023 px* – mobilní zařízení typu tablet jako iPad nebo Samsung Galaxy Tab 2, viz obrázek 4.5.
- Více než *1023 px* – veškerá desktop zařízení.

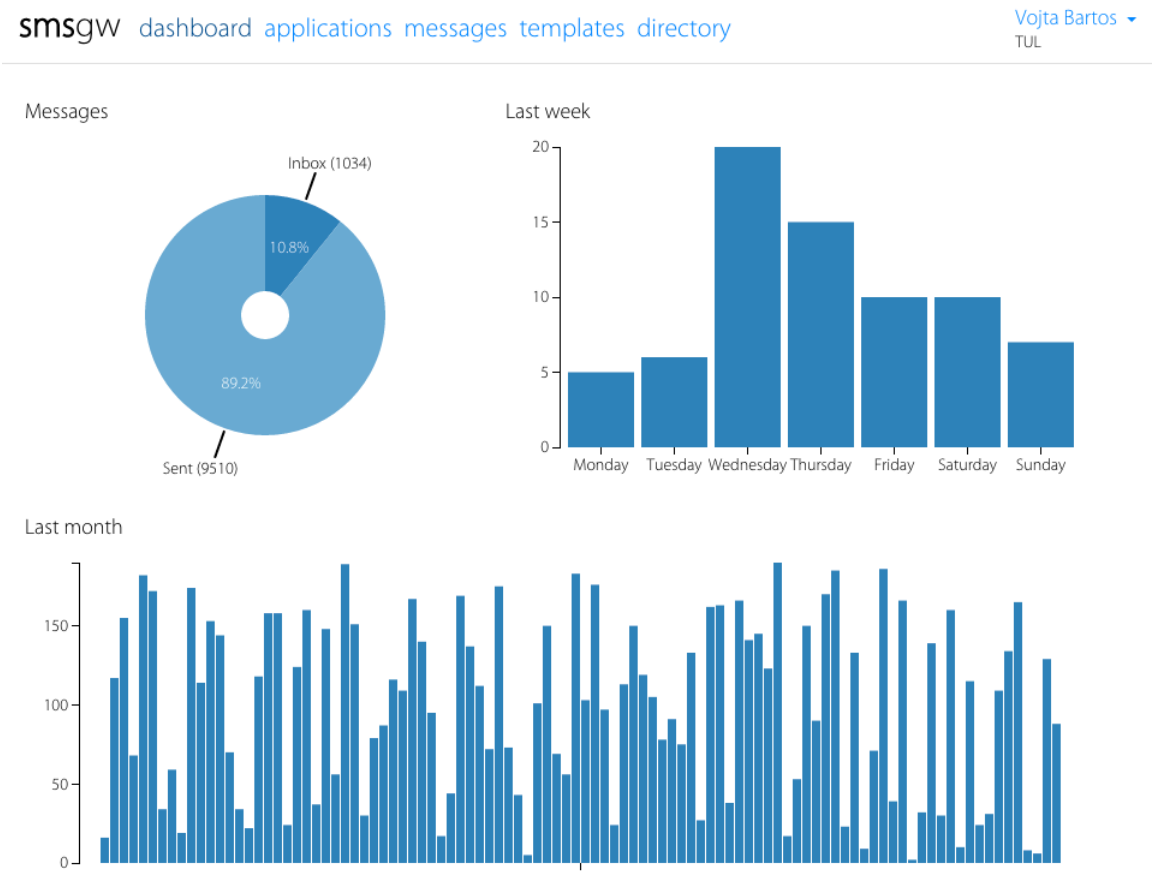
4.3.6 Možnosti klienta

Klientská aplikace je vždy dostupná na URL adrese, která byla vybrána při instalaci webové služby na server. Po úspěšné instalaci je možno aplikaci spustit v moderních webových prohlížečích – jak na desktopu, tak na mobilním zařízení.

Uživatelské rozhraní bylo navrženo s důrazem na intuitivní orientaci v aplikaci s cílem snadného a rychlého ovládání a bylo přizpůsobeno mobilním zařízením.

Dashboard

Dashboard je hlavní stránka, která se uživateli zobrazí po úspěšné autentizaci. Tato stránka obsahuje základní statistiky ohledně odeslaných a přijatých textových zpráv jednak za období jednoho měsíce, jednak za období jednoho týdne.



Obrázek 4.5: Úvodní obrazovka

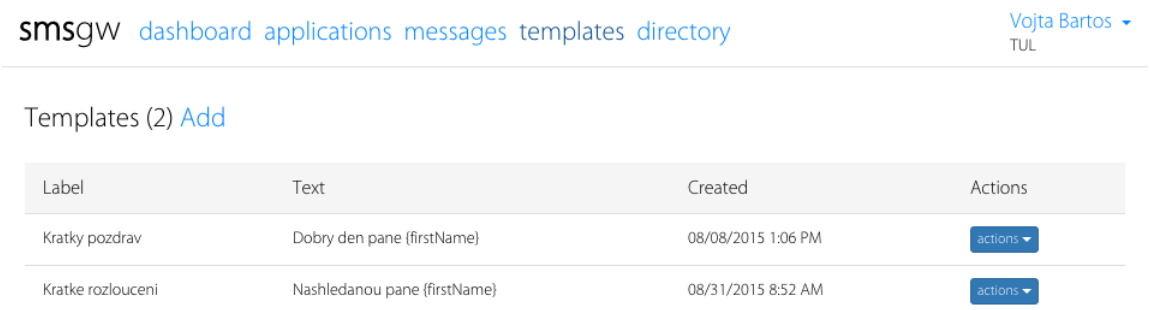
Šablony

Komponenta, která umožňuje uživateli vytvářet, upravovat a mazat šablony textových zpráv. Tyto šablony mohou být vybrány při vytváření textových zpráv jako předloha a mohou být poté upraveny. Každá šablona musí obsahovat název a textový

obsah šablony. Textový obsah může obsahovat klíčová slova, která budou zaměněna za data kontaktu při odeslání textové zprávy.

Nynější aplikace obsahuje základní klíčová slova:

- *{firstName}* – křestní jméno pro daný kontakt,
- *{lastName}* – příjmení pro daný kontakt.



Obrázek 4.6: Seznam šablon

Aplikace

Komponenta umožňující vytváření, úpravu a mazání záznamů o aplikacích třetích stran. Tento záznam slouží k tomu, aby uživatel mohl zasílat textové zprávy přes webovou službu z vlastní aplikace.

Pro každou zaregistrovanou aplikaci je vygenerován unikátní autorizační řetězec, který je použit pro autorizaci požadavků zaslaného z uživatelovy aplikace. Aplikace třetí strany zasílá požadavky na zdroj REST API rozhraní, které očekává autorizační řetězec pro registrované aplikace. Autorizační řetězec musí být specifikován v hlavičce požadavku označen jako *Authorization*.

```
GET /api/1.0/outbox/ HTTP/1.1
Host: smsgw.alinet.cz
Accept: application/json
Authorization: 422511f6-008e-4a00-a15e-23fbac6a9c4c
Content-Type: application/json
```

Zdrojový kód 4.33: Ukázka HTTP požadavku pro externí použití

Webová služba podporuje i zasílání notifikací o přijatých zprávách přiřazených k registrované aplikaci na webovou službu třetí strany. Nicméně je potřeba specifikovat nezbytné atributy v nastavení konkrétní aplikace, a to *prefix* a *callback URL*.

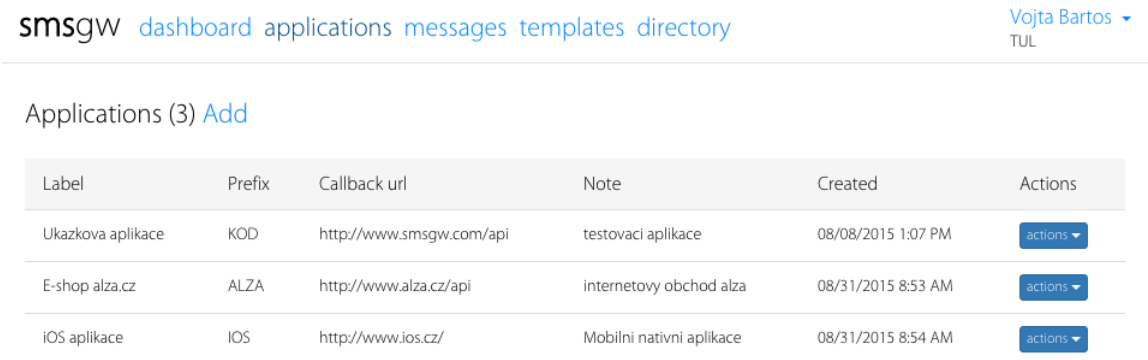
- prefix
  - řetězec délky 2 až 5 znaků,

  - prefix zprávy používaný pro identifikaci aplikace; pokud webová služba přijme textovou zprávu, která začíná prefixem zaregistrovaným k aplikaci, bude textová zpráva přiřazena uživateli a aplikaci,

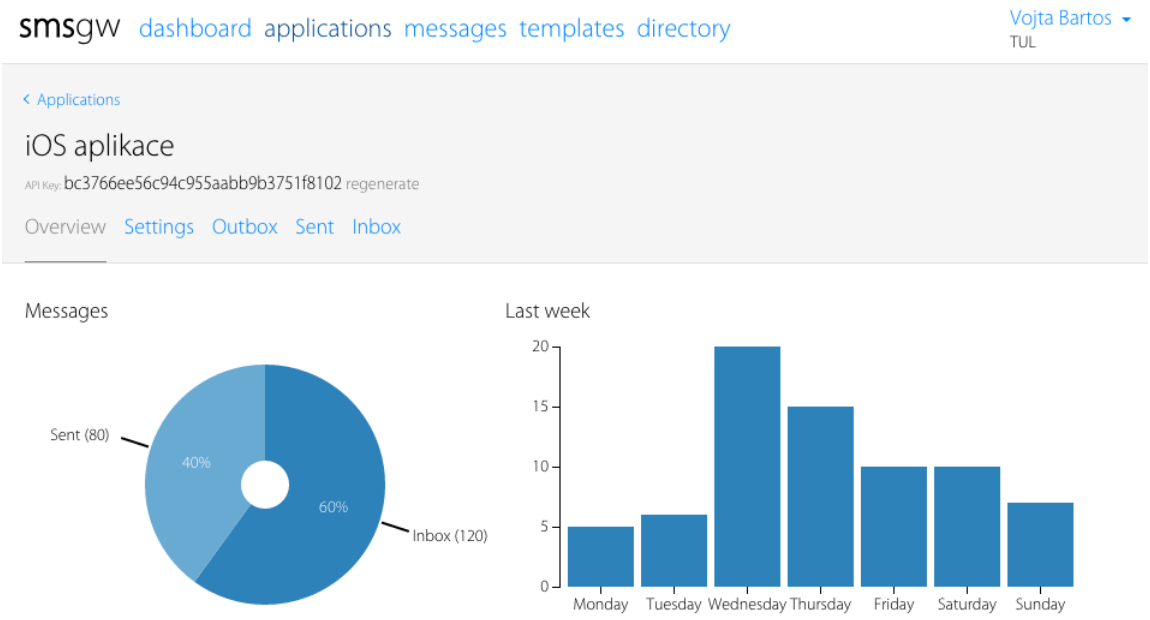
- callback URL adresa
  - URL řetězec neomezené délky,
  - pokud dojde k přiřazení textové zprávy k aplikaci pomocí prefixu, bude vytvořen HTTP POST požadavek, který bude odeslán na tuto callback URL adresu; součástí požadavku budou veškeré informace o textové zprávě jako obsah, telefonní číslo či datum a čas přijetí; každý vytvořený požadavek bude zařazen do fronty RabbitMQ, která se stará o odesílání všech požadavků, a budou vykonány asynchronně na pozadí.

Součástí komponenty je možnost zobrazit uživateli detail registrované aplikace, který se skládá z více složek:

- Overview - statistiky pro registrovanou aplikaci zobrazující odeslané a přijaté textové zprávy,
- Settings - nastavení aplikace, kde lze změnit prefix, callback URL či popisek,
- Outbox - fronta již zaregistrovaných textových zpráv čekajících na odeslání,
- Sent - seznam všech již odeslaných zpráv,
- Inbox - seznam všech přijatých či přiřazených zpráv registrované aplikace.



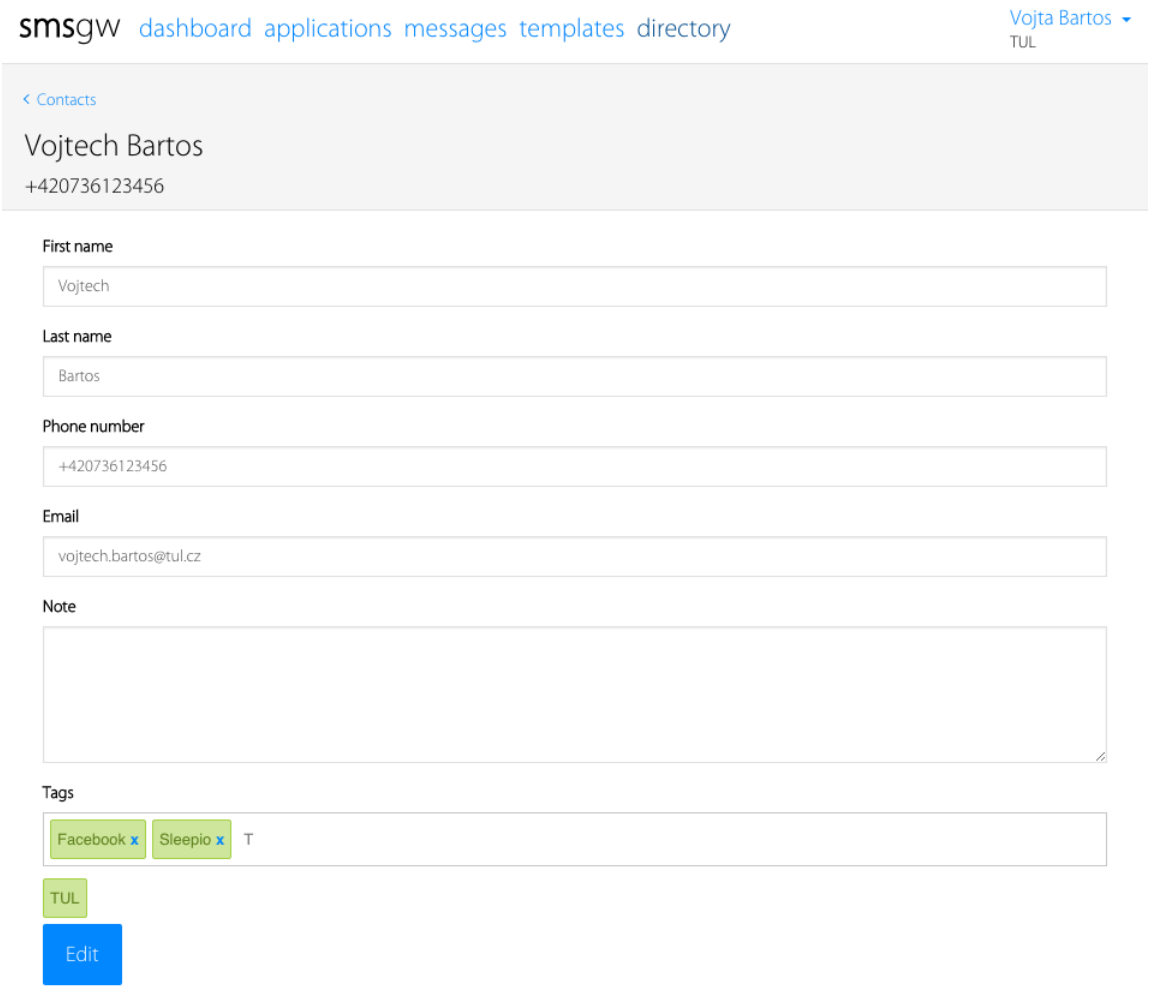
Obrázek 4.7: Seznam aplikací



Obrázek 4.8: Detail aplikace

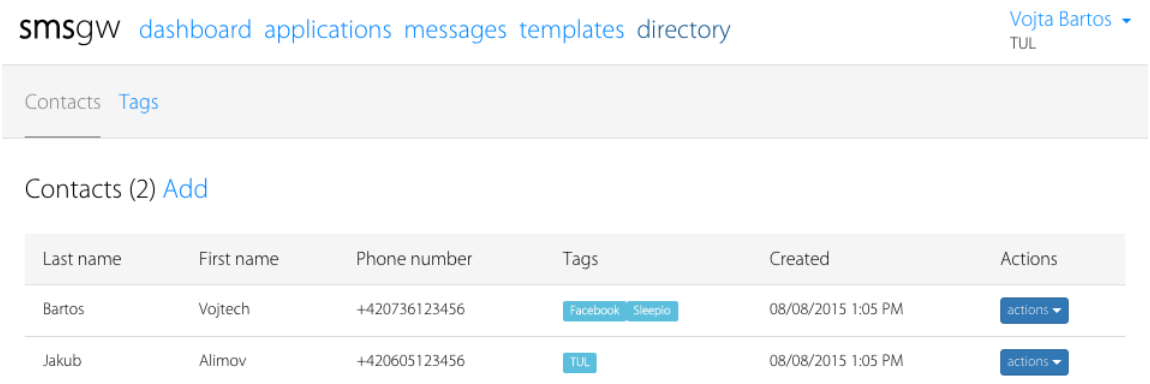
Adresář

Komponenta, která spravuje uživatelův adresář, který obsahuje sekce kontakty a tagy, neboli skupiny, do kterých může být kontakt přiřazen. Každá sekce je reprezentovaná vlastní komponentou a je přístupná pomocí tabů neboli záložek.

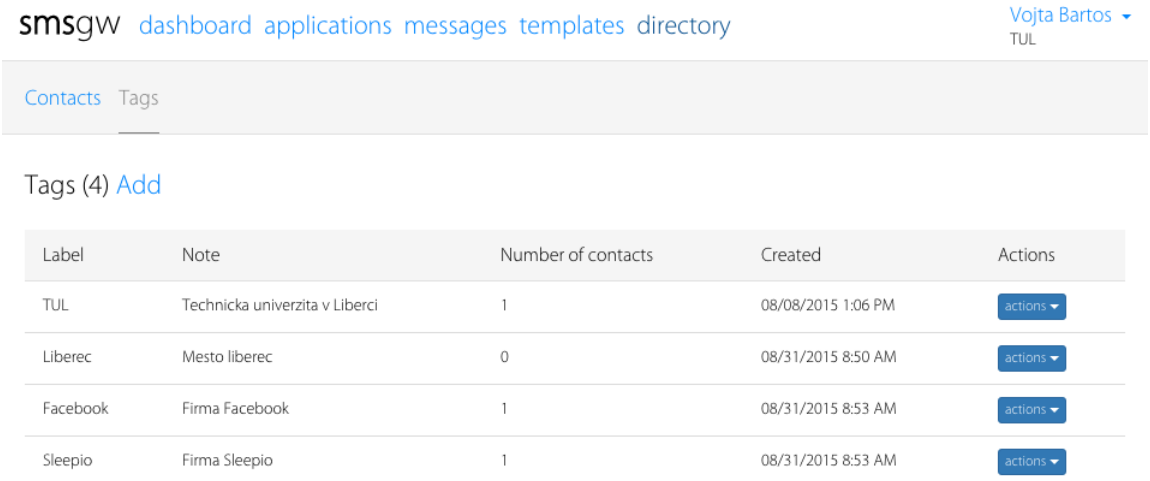


Obrázek 4.9: Formulář pro editaci kontaktu s interaktivním políčkem

- Kontakty
  - Komponenta umožňující uživateli vytvářet, upravovat a mazat kontakty. Pro každý kontakt je možno nastavit telefonní číslo, jméno, příjmení, e-mail a krátký popisek. Kontakt může být zařazen do různých skupin pomocí tagů, kterých může mít nespočet. Kontakt je vždy přiřazen k přihlášenému uživateli, tudíž ostatní uživatelé nemají přístup ke kontaktům jiného uživatele. Formulář pro vytvoření kontaktu obsahuje interaktivní políčko pro tagy, které napovídá existující tagy po stisknuté klávese. Pokud již tag neexistuje, bude automaticky vytvořen a přiřazen ke kontaktu.
- Tagy
  - Uživatel má přehled o všech vytvořených tazích a má možnost přidat nový či upravit stávající. Uživatel má možnost zobrazit uživatele, kteří mají přiřazený daný tag.



Obrázek 4.10: Seznam kontaktů

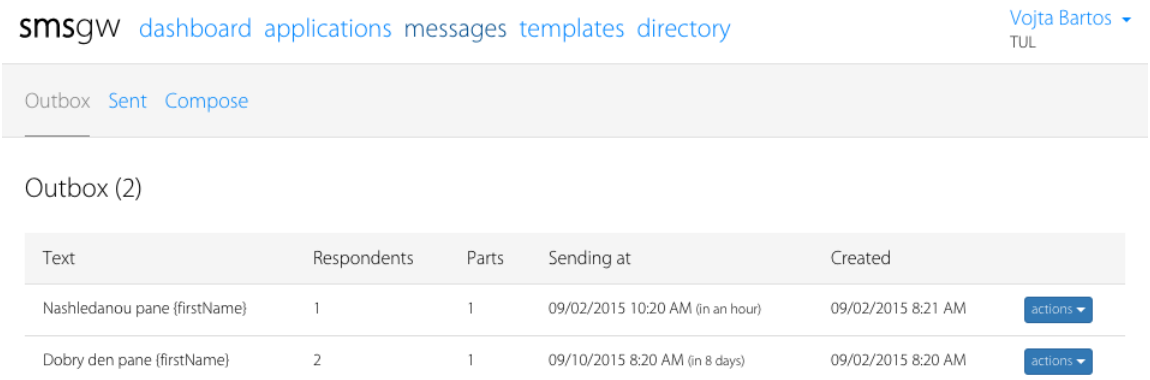


Obrázek 4.11: Seznam tagů

Zprávy

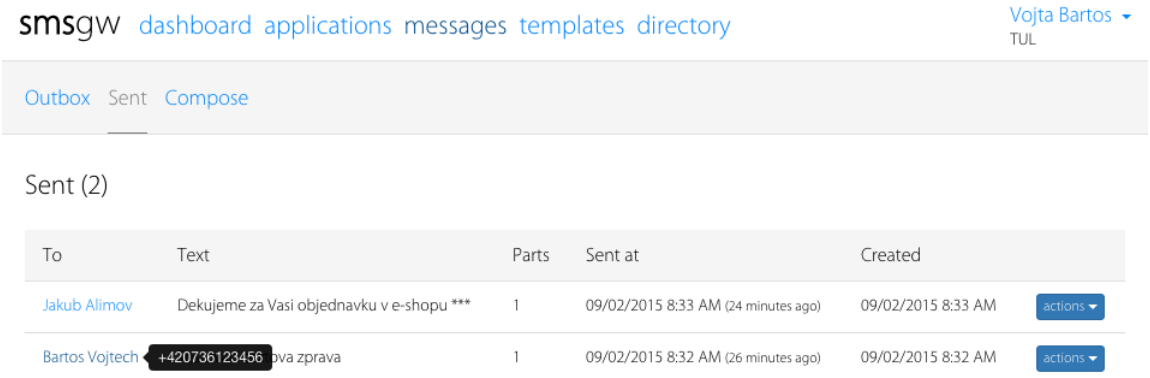
Komponenta starající se o veškeré zprávy, které čekají na odeslání nebo které byly odeslány. Funkcionality jsou rozděleny do sekcí a každá sekce je reprezentována vlastní komponentou pomocí tabů neboli záložek.

- Outbox
  - Komponenta zobrazující frontu zpráv, které čekají na odeslání. Uživatel má možnost zprávu smazat či zobrazit detail textové zprávy. Detail textové zprávy obsahuje informace jako kontakty či telefonní čísla, časové informace, obsah textové zprávy a informace, zda je textová zpráva rozdělena na více zpráv.



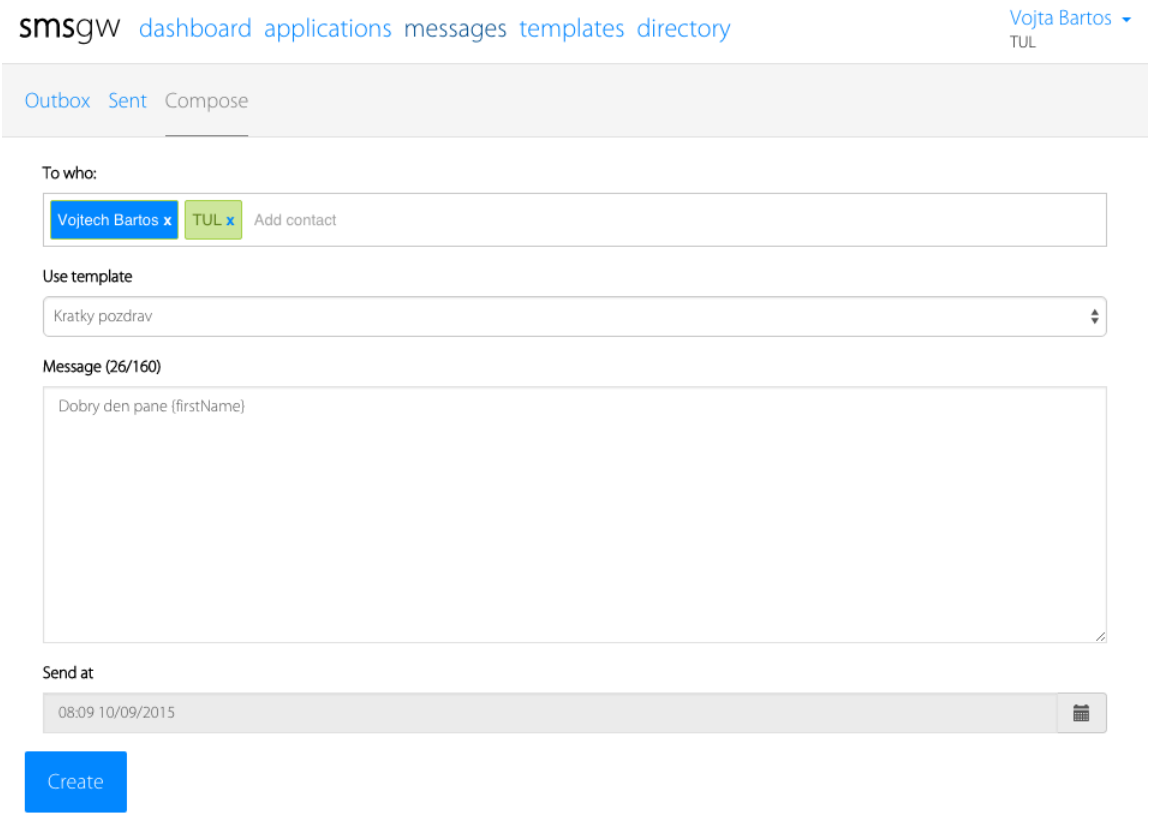
Obrázek 4.12: Outbox

- Odeslané zprávy
  - Komponenta, která zobrazuje všechny zprávy, které byly odeslány z uživatelova účtu. Každý záznam zobrazuje, jaká zpráva byla odeslána, časové údaje a kontakt či telefonní číslo.



Obrázek 4.13: Odeslané zprávy

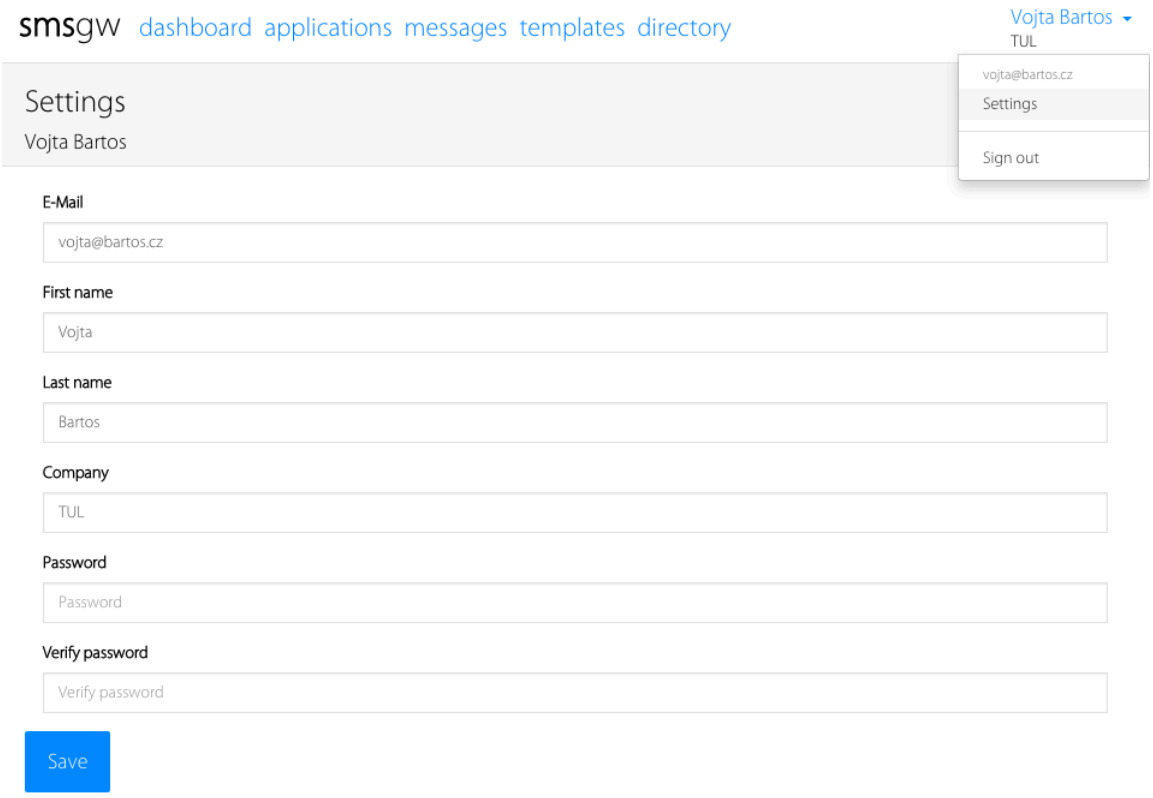
- Vytvoření zprávy
  - Komponenta umožňující uživateli odeslat textovou zprávu kontaktům, které mají přiřazený specifický tag neboli skupinu. Součástí komponenty je formulář, který nabízí interaktivní hledání tagu či kontaktu. Dále nabízí možnost vybrat předem připravenou šablonu textové zprávě. Textová zpráva může být odeslána ihned nebo je možné nastavit specifický čas odeslání.



Obrázek 4.14: Vytvoření zprávy

Nastavení

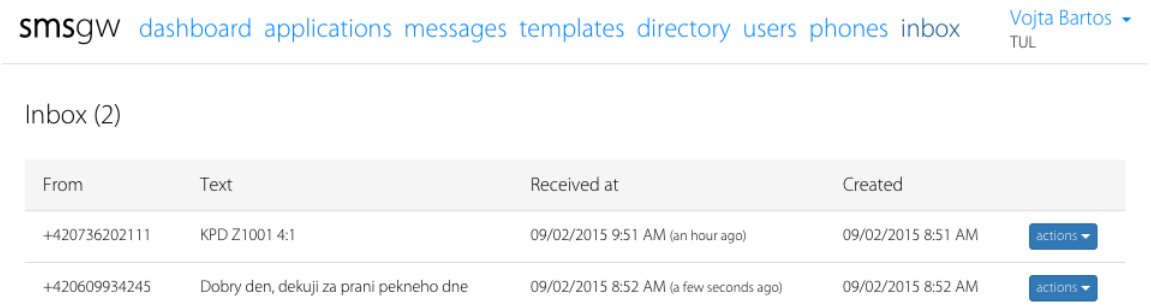
Komponenta umožňující změnit uživatelovy informace a možnost změny hesla. Jedná se o jednoduchou stránku obsahující pouze formulář pro změnu dat. Nastavení je přístupné po kliku na jméno v pravém horním rohu obrazovky.



Obrázek 4.15: Formulář pro změny uživatelových informací

Inbox

Komponenta, která je přístupná pouze uživateli s rolí *admin*. Zobrazuje textové zprávy, které nebyly přiřazeny k žádnému uživatelskému účtu. Administrátor může zprávu vymazat či upravit. Tato komponenta slouží výhradně pro administrátora k roztřídění textových zpráv k uživatelskému účtu, pokud došlo k překlepu v textové zprávě.



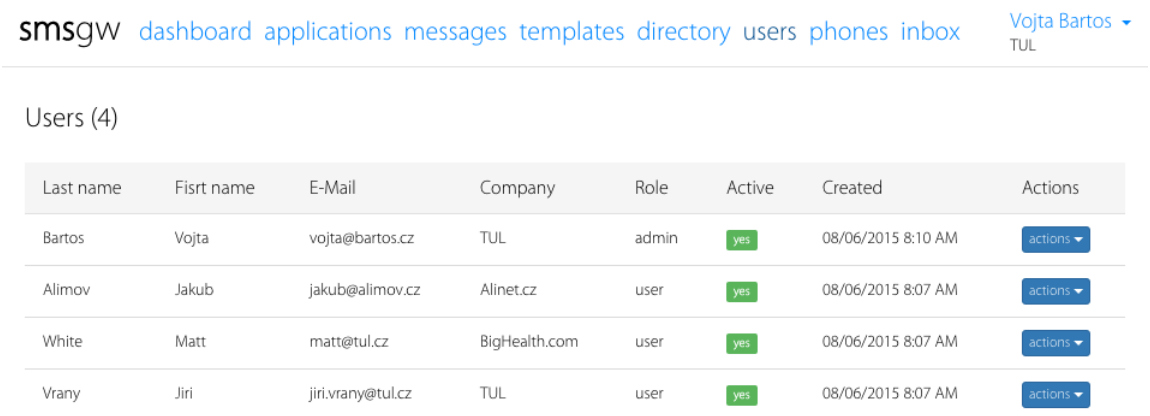
Obrázek 4.16: Seznam přijatých zpráv v Inboxu

Uživatelé

Komponenta, která je přístupná pouze uživateli s rolí *admin*. Je to administrační stránka pro správu uživatelských účtů, která umožňuje přidávat, upravovat a mazat uživatele. Pro každého uživatele může administrátor vidět základní statistiku účtů, jako je počet šablon, aplikací, odeslaných a přijatých zpráv.

Administrátor má možnost měnit role uživatelů. Současné verze podporuje dvě základní role:

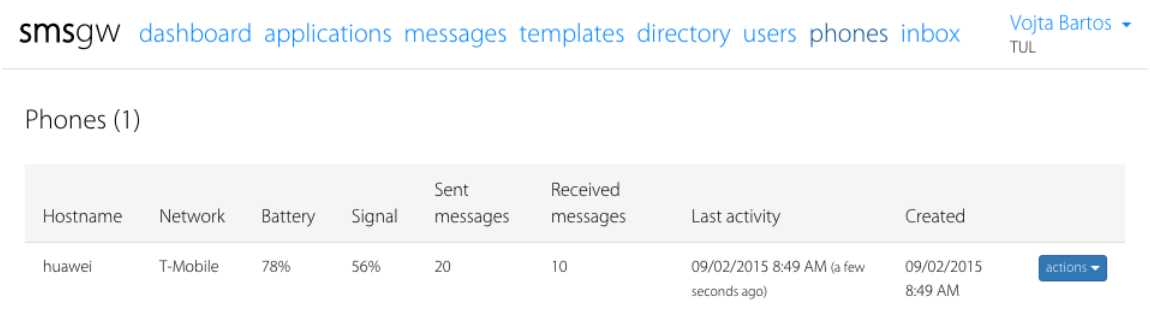
- *user*
  - klasický uživatel bez administrátorských oprávnění,
- *admin*
  - oprávnění pro přístup do administrátorské sekce.



Obrázek 4.17: Seznam registrovaných uživatelů

Telefony nebo modemy

Komponenta, která je přístupná pouze uživateli s rolí *admin*. Zobrazuje připojené telefony, které aktivně komunikují s běžícím procesem na pozadí *SMSD Daemon*. Pro každé zařízení jsou dostupné informace jako síla signálu, stav baterie, počet odeslaných a přijatých textových zpráv, název operátora a název zařízení. Tato komponenta zobrazuje pouze informace, které jsou automaticky aktualizované procesem *SMSD Daemon*.



Obrázek 4.18: Seznam aktivních připojených modemů či telefonů

5 Možnosti rozšíření

Aplikace byla implementována jako multiplatformní HTML5 aplikace, a tím byl splněn požadovaný cíl práce. Avšak během návrhu a vývoje byla zohledňována možnost následného rozšíření aplikace.

První možností rozšíření je vytvoření nativních klientů pro platformy Android a iOS díky lepší uživatelské přívětivosti. Multiplatformní HTML5 aplikace je sice optimalizována pro mobilní zařízení, nicméně nativní aplikaci se ještě v dnešní době z pohledu uživatelské přívětivosti nevyrovná. Jednalo by se pouze o implementaci nativních klientů využívajících existující webové služby.

Stávající řešení autentizace a autorizace není ideální. Je sice velice jednoduché na implementaci i použití, nicméně není jedním z nejbezpečnějších. Existuje mnoho již hotových, bezpečných a otestovaných řešení. Jedním z hojně využívaných protokolů pro autentizaci a autorizaci je protokol Oauth2.

Bezpečnost celé aplikace by mohla být vylepšena zabezpečením komunikace mezi klienty a serverem. Veškerá komunikace se provádí ve veřejné síti, a proto by měla být zabezpečena protokolem SSL.

Stávající vzhled webové aplikace by měl být změněn na vzhled navržený profesionálním grafikem. Vzhled by měl být navržen pro všechny platformy a zařízení a měl by být zhotoven na základě *User Guidelines* každé platformy.

Veškerý text aplikace je momentálně výhradně v anglickém jazyce. Je tomu tak z důvodu vystavení aplikace jako open-source projektu. Velkým přínosem by byla implementace více jazykových mutací, především české, a možnosti přepínat mezi nimi.

6 Závěr

Cílem této diplomové práce bylo vytvořit aplikaci, která by měla umožnit rozesílání SMS pro potřeby různých společností, např. internetových obchodů.

Nejprve bylo nezbytné vytvořit rešerši a zhodnotit již existující webové služby pro zpracovávání textových zpráv a k nim příslušných klientů (viz kapitola 2).

Dalším krokem se stal návrh samotné aplikace, který byl prodiskutovám s konzultantem diplomové práce, a poté byly určeny základní technologie a postupy. Jedním z nejdůležitějších prvků návrhu byl výběr vhodného softwaru pro komunikaci s připojeným USB modemem. Po vybrání tohoto softwaru byla zvolena struktura databáze, která musela vycházet z vybraného softwaru pro komunikaci. Aplikace SMS brána byla navržena jako klient-server aplikace, ve kterém klientská aplikace byla zvolena jako HTML5 multiplatformní single-page aplikace z důvodu nutnosti přistupu z webového prohlížeče. Kompletní návrh databáze, návrh komunikace s USB modemem a výsledný návrh klientské aplikace a webové služby jsou popsány v kapitole 3.

V první fázi implemetace byl vyvinut klient (4.3) typu singe-page aplikace bez zhotovené webové služby. Jako dočasná webová služba byla zvolena služba se statickým obsahem, která byla vygenerována pomocí služby apiary.io. Dočasná webová služba velmi ulehčila a urychlila vývoj klientské aplikace bez nutnosti zhotovení realné webové služby.

Po úspěšné implementaci klientské aplikace, která komunikovala s dočasnou webovou službou, byla dostupná dokumentace webové služby ve formátu blueprint společnosti apiary.io. Tato dokumentace sloužila jako specifikace REST API rozhraní webové služby při její implementaci. Postup implemetace webové služby je popsán v kapitole 4.2.

Konečným krokem implementace po zhotovení webové služby a klientské aplikace bylo připravení a nainstalování potřebného serveru (4.1) pro běh celé aplikace, která je přístupná odkudkoliv na světě. Příprava serveru byla provedena až jako konečný krok po znalosti všech požadavků webové služby a klienta.

V kapitole 5 jsou popsány případné možnosti rozšíření. Při tvorbě aplikace již bylo počítáno s dalšími možnostmi rozšíření, které se mohou stát základem dalších prací.

Práce byla vyvíjena podle specifikace společnosti Alinet cz, s. r. o. Během vývoje aplikace bylo použito několik open-source knihoven a z tohoto důvodu je výsledná aplikace zveřejněna pod open-source licencí MIT [10].

Seznam literatury

[1] FIELDING, Roy Thomas. *Architectural Styles and the Design of Networkbased Software Architectures* [online]. University of California, 2000 [cit. 20131008]. Dostupné z: http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm

[2] RICHARDSON, Leonard a Sam RUBY. *RESTful web services*. 1st ed. Sebastopol: O'Reilly, 2007, xxiv, 419 s. ISBN 9780596529260.

[3] Android API Guides. *Developer Android* [online]. 2012 [cit. 20121014]. Dostupné z: http://developer.android.com/guide/components/index.html.

[4] MARK, Dave a Jeff LAMARCHE. *IPhone SDK: průvodce vývojem aplikací pro iPhone a iPod touch.* Vyd. 1. Brno: Computer Press, 2010, 480 s. ISBN 9788025128206.

[5] GRINBERG, Miguel. *Flask Web Development: Developing Web Applications with Python.* Vyd. 1. O'Reilly, 2014, 258 s. ISBN 9781449372620

[6] PILGRIM, Mark. *Dive Into Python.* Springer 2004, 436 s. ISBN 9781590593561¨

[7] Apple Inc. *The Swift Programming Language* [online]. Dostupné z: https://itunes.apple.com/gb/book/swiftprogramminglanguage/id881256329?mt=11

[8] VIDELA, Alvaro a Jason J. W. WILLIAMS, *RabbitMQ in Action: Distributed Messaging for Everyone*, Vyd. 1. Manning Publications 2012, 312 s. ISBN 9781935182979

[9] CROCKFORD, Douglas. *The application/json Media Type for JavaScript Object Notation (JSON).* [online] 2006 RFC editor [cit. 2015-08-17]. Dostupné z: http: //www.rfc-editor.org/rfc/rfc4627.txt.

[10] Open Source Initiative. *The MIT License (MIT).* [online]. 2015 [cit. 2015-08-05]. Dostupné z: http://opensource.org/licenses/MIT.

[11] MariaDB Corporation. *MariaDB.* [online] 2015 [cit. 2015-08-17]. Dostupné z: https://mariadb.com/.

[12] Facebook Inc. *React.* [online] 2015 [cit. 2015-08-17]. Dostupné z: https://facebook.github.io/react/.

[13] Facebook Inc. *Flux*. [online] 2015 [cit. 2015-08-17]. Dostupné z: https://facebook.github.io/flux/.

[14] ČÍHAŘ, Michal. *Gammu*. [online] 2015 [cit. 2015-09-04]. Dostupné z: http://wammu.eu/gammu/.

[15] BACK, kent. *Test Driven Development: By Example* Vyd. 1. Addison-Wesley Professional, 2012, 240 s. ISBN 978-0321146533

[16] Travis CI, GmbH. *Travis CI documentation*. [online] 2015 [cit. 2015-09-05]. Dostupné z: http://docs.travis-ci.com/

[17] Agendaless consulting and contributors. *Supervisor: A process control system*. [online] 2015 [cit. 2015-09-05]. Dostupné z: http://supervisord.org/

[18] NodeJS Foundation. *Node.JS*. [online] 2015 [cit. 2015-09-05]. Dostupné z: https://nodejs.org/en/

[19] CAVALIER, Brian a Domenic DENICOLA. *Primises/A+*. [online] 2015 [cit. 2015-09-06]. Dostupné z: https://promisesaplus.com/

[20] Ecmascript.org *Draft Specification for ES.next*. [online] 2015 [cit. 2015-09-09]. Dostupné z: http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts

[21] Open Source Initiative. *GNU General Public License Versions* [online]. 2015 [cit. 2013-09-07]. Dostupné z: http://opensource.org/licenses/gpl-license.

[22] Open Source Initiative. *Apache License, Version 2.0.*. [online] 2015 [cit. 2013-09-07]. Dostupné z: http://opensource.org/licenses/Apache-2.0.

[23] DANGOOR, Kevin. *CommonJS*. [online] 2015 [cit. 2015-09-07]. Dostupné z: http://www.commonjs.org/

[24] The Internet Society. *A Universally Unique IDentifier (UUID) URN Namespace*. [online] 2015 [cit. 2015-09-07]. Dostupné z: https://tools.ietf.org/html/rfc4122

Příloha A - Obsah CD

Na přiloženém CD se nachází:

- VojtechBartos.dp.pdf - tato práce ve formátu PDF,
- smsgw.zip - zdrojové kódy aplikace.