FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Našeptávání a validace e-mailových adres
Student:	Beňadik Štrba
Vedoucí:	Ing. Jan Herold
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je vytvořit službu, která bude pomáhat uživatelům zadat e-mail ve webovém formuláři správně a bez překlepů a autorům webových aplikací zajistí, že zadané e-mailové adresy opravdu existují.

1. Seznamte se možnostmi, jak lze ověřit existenci e-mailové adresy – jak lze poznat, že na zadanou e-mailovou adresu lze doručovat e-maily.
2. Navrhněte a implementujte nástroj, který bude ověřovat existenci e-mailové adresy a bude umět navrhnout opravu dojde-li k překlepu v názvu domény.
3. Navrhněte a implementujte nástroj, který bude umět navrhovat dokončení rozepsaných e-mailových adres – doménovou část u nejčastěji používaných domén.
4. Pro oba nástroje navrhněte a implementujte webovou službu.
5. Navrhněte a implementujte javascriptovou knihovnu, která za pomoci výše popsané webové služby bude uživatelům navrhovat, jak dokončit rozepsanou e-mailovou adresu a po dopsání e-mailové adresy ověří její existenci. Výsledek validace zobrazí přímo ve formuláři.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 20. října 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Našepkávanie a validácia e-mailových adries

Beňadik Štrba

Katedra softwarového inžinierstva
Vedúci práce: Ing. Jan Herold

12. mája 2021

Poďakovanie

Chcel by som sa poďakovať predovšetkým vedúcemu práce Ing. Janovi Heroldovi za trpezlivosť, cenné rady, vždy pozitívny prístup a obetovaný čas, počas tvorby tejto bakalárskej práce. Ďakujem aj kamarátovi Lukášovi za ochotu a skvelé nápady. Rovnako patrí vďaka celej mojej rodine a priateľom za podporu a v neposlednom rade aj mojej priateľke Laure.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov. Ďalej prehlasujem, že som s ČVUT uzavrel dohodu, na základe ktorej sa ČVUT vzdalo práva na uzavrenie licenčnej zmluvy o používaní tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona. Táto skutočnosť nemá vplyv na ust. § 47b zákona č. 111/1998 Sb. o vysokých školách.

V Prahe 12. mája 2021 . . . . . .

Odkaz na túto prácu

Štrba, Beňadik. *Našepkávanie a validácia e-mailových adries*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Bakalárska práca sa zaoberá službou, ktorá bude pomáhať užívateľom zadávať vo webových formulároch e-mailové adresy a po zadaní zisťovať ich existenciu. Výsledná služba bude rozšírením existujúcej služby smartform, ktorá aktuálne poskytuje našepkávanie a validáciu adries. Cieľom práce je zistiť, ako a či vôbec je možné určiť existenciu e-mailových adries a následne implementovať vybrané riešenie. Na základe analýzy možných riešení validácie e-mailových adries som navrhol a implementoval službu, využívajúcu pri validácií SMTP protokol.

Pri testovaní vytvorenej implementácie sa vyskytlo niekoľko problémov. Služba nie je schopná rozhodnúť o existencií všetkých e-mailových adries, ktoré dostane na vstupe. Na druhej strane, je funkčná pre väčšinu populárnych poštových serverov, a preto má praktické využitie.

Kľúčové slová implementácia služby, API, validácia e-mailových adries, smtp, editačná vzdialenosť, java, Spring Boot

Abstract

This bachelor thesis focuses on a service, which would help users to enter e-mail addresses in electronic forms and verify their existence. The resulting service will be an extension of the existing smartform service, which currently provides e-mail address suggestions and validation. The aim of this thesis is to find out, whether and how it would be possible to determine the existence of e-mail addresses and then to implement a preferred solution. I designed and implemented a service, using the SMTP protocol, based on the analysis of the possible solutions for e-mail address validation.

There happen to be several problems when I was testing the implementation. The service cannot verify the existence of every e-mail address it receives. On the other hand, the service works correctly for most popular mail servers and therefore it has a practical use.

Keywords service implementation, API, e-mail address validation, smtp, edit distance, java, Spring Boot

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov kódu

Úvod

V dnešnej dobe existuje nespočetne veľa web stránok, ktoré požadujú od nás naše e-mailové adresy. Môžu to byť napríklad e-shop, bazár, fórum alebo ľubovoľná stránka s možnosťou registrácie. Či už je to kvôli našej identifikácii, odoslaniu údajov o objednávke, alebo má na to táto webová stránka akýkoľvek iný dôvod, vždy potrebuje existujúcu e-mailovú adresu.

Validácia e-mailových adries je dôležitá, najmä kvôli tzv. reputácii odosielateľa. Túto reputáciu je potrebné udržať čo najvyššiu, pretože čím vyššia bude, tým je väčšia šanca, že e-mail bude doručený a nebude zaradený napr. ako spam. Reputácia závisí od mnohých faktorov, no jeden z hlavných je odosielanie e-mailov na existujúce e-mailové adresy. V neposlednom rade je možné vďaka tejto validácií eliminovať falošné registrácie, či objednávky. Mnoho webových stránok nedisponuje žiadnou validáciou e-mailových adries, poprípade iba základnou kontrolou syntaktickej stránky adresy, čo sa dá jednoducho obísť a nie je to dostačujúci spôsob validácie.

Táto bakalárska práca sa zaoberá vytvorením služby, ktorá bude validovať a našepkávať e-mailové adresy a pokúsi sa rozhodnúť, či daná e-mailová adresa existuje alebo nie, pretože nie vždy to je možné určiť.

Hlavnou motiváciou výberu tejto témy bolo, že výsledná služba bude rozšírením existujúcej služby smartform, ktorá momentálne poskytuje našepkávanie a validáciu adries pre rôzne webové stránky, čo je z môjho pohľadu veľmi praktické a prospešné.

Z hľadiska štruktúry práca pozostáva zo šiestich hlavných kapitol. Prvá kapitola je teoretická a vysvetľuje pojmy používané v ďalších kapitolách. Druhá kapitola sa zaoberá analýzou validácie e-mailových adries, súčasnými riešeniami a nakoniec aj analýzou našepkávania a opravy doménovej časti adries. Tretia kapitola sa zameriava na popis použitých technológií, návrh a architektúru hlavných častí služby a na spôsob ako bude výsledná služba integrovaná do klientskych aplikácií. Štvrtá kapitola nadväzuje na predchádzajúcu s procesom implementácie služby a jej nasadením. V piatej kapitole je popísané

testovanie služby, dosiahnuté výsledky a pretrvávajúce problémy. Posledná kapitola naznačuje možné rozšírenia tejto služby.

Ciele práce

Primárnym cieľom práce je vytvoriť službu, ktorá bude užívateľom pomáhať dokončiť dopísať doménovú časť e-mailovej adresy, poskytovať jednoduché opravy preklepov a validovať zadanú e-mailovú adresu.

K dosiahnutiu tohoto cieľa je potrebné najskôr vykonať analýzu, ako je možné overiť existenciu e-mailovej adresy, bez toho aby sa odoslal na túto adresu e-mail. Taktiež je žiadúce zrealizovať aj analýzu dostupných algoritmov pre dopĺňanie a opravu doménovej časti e-mailovej adresy. Na základe analýzy je ďalej potrebné vytvoriť softwarový návrh služby, ktorý poslúži pri implementácií a javascriptovú knižnicu, ktorá bude prepájať klientský web s výslednou službou. Ako posledný bod si táto práca nesie za cieľ otestovať validáciu e-mailových adries a vyvodiť záver z dosiahnutých výsledkov.

Cieľom tejto práce nie je vytvoriť produkčnú službu, ale poskytnúť získané znalosti a základnú implementáciu, ktorá rozšíri existujúcu službu smartform.

Kapitola 1

Teória

Cieľom úvodnej kapitoly je objasniť vybrané odborné pojmy, ktoré používam v nasledujúcich kapitolách práce. Najskôr sa zameriam na pojmy súvisiace s validáciou e-mailových adries a v druhej časti kapitoly vysvetlím pojmy súvisiace s dopĺňaním a opravou doménovej časti e-mailových adries.

1.1 Elektronická pošta

Elektronická pošta existuje už od ranných začiatkov internetu. Ako konkrétne elektronická pošta funguje, je uvedené v nasledujúcom texte.

1.1.1 E-mailová adresa

E-mailová adresa je špecifický internetový identifikátor, obsahujúci lokálnu časť, ktorá sa nachádza pred znakom zavináč „@“ a následne internetovú doménu, napríklad: lokalnacast@domena.

Lokálna časť je závislá na doménovej časti a v e-mailových adresách reprezentuje meno konkrétnej poštovej schránky. Doménová časť identifikuje cieľ, na ktorý je e-mail doručený [1].

1.1.2 Mail Agents

Podľa [2] existuje niekoľko poštových agentov, zabezpečujúcich prenos e-mailu.

Mail User Agent (MUA)

MUA je lokálna aplikácia u klienta, ktorá užívateľovi umožňuje vytvárať, editovať a tak isto aj odosielať a prijímať e-maily.

Mail Submission Agent (MSA)

MSA predstavuje serverovú aplikáciu, ktorá prijíma e-mail od MUA a preposiela ho ďalej MTA.

Mail Transfer Agent (MTA)

MTA je serverová aplikácia zodpovedná za prenos a smerovanie e-mailov od odosielateľa k prijímateľovi, ktorý ale nemusí byť nutne finálny.

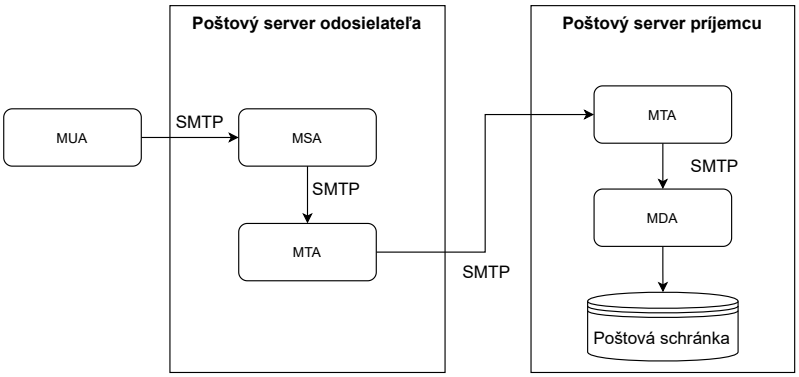
Mail Delivery Agent (MDA)

MDA je serverová aplikácia, ktorá prijíma e-mail od MTA a uloží ho do poštovej schránky.

1.1.3 Životný cyklus e-mailu

E-mail na obr. 1.1 od jeho vytvorenia odosielateľom, až po príchod do prijímateľovej poštovej schránky, prechádza cez viacerých agentov. Vytvorený e-mail MUA odošle MS agentovi, ktorý e-mail predá MT agentovi. Ten ho príjme a odošle jednému alebo viacerým MT agentom. To, ako bude MTA smerovať e-mail záleží od toho, či prijímateľova poštová schránka je lokálna alebo nie. Pokiaľ je adresa prijímateľa lokálna, MTA pošle e-mail ďalej MD agentovi, ktorý už e-mail uloží do prijímateľovej poštovej schránky. Prijímateľ sa k e-mailu dostane opäť pomocou MU agenta. Pokiaľ e-mailová adresa nie je lokálna, tak doménová časť e-mailovej adresy sa použije na zistenie poštového servera, ktorý prijíma e-maily pre danú doménu. MT agentovi tohoto servera je potom preposlaný e-mail [2].

Komunikáciu zabezpečuje SMTP protokol. Medzi MUA a MSA prebieha komunikácia prevažne na porte 587, z historických dôvodov je podporovaná stále aj na porte 25. Komunikácia medzi MSA a MTA rovnako ako aj medzi jednotlivými MT agentmi prebieha už len na porte 25.



Obr. 1.1: Životný cyklus e-mailu

1.2 SMTP protokol

1.2.1 Predstavenie

SMTP alebo Simple Mail Transfer Protocol je protokol, operujúci na aplikačnej vrstve osi modelu. Protokol slúži na prenos e-mailov medzi odosielateľom (klientom) a prijímateľom (serverom). Pokiaľ má klient správu, ktorú chce odoslať, nadviaže TCP spojenie so SMTP serverom. SMTP server je buď cieľový server prijímateľa alebo ide o server figurujúci ako prostredník.

Prenos správy môže byť realizovaný v jedinom spojení medzi originálnym odosielateľom a cieľovým prijímateľom alebo tento prenos môže vyzerať ako séria skokov medzi servermi správajúcich sa ako prostredníci. Po nadviazaní spojenia, klient iniciuje prenos správy. Tento prenos pozostáva zo série príkazov, týkajúcich sa špecifikovania odosielateľa, prijímateľa a obsahu správy. Server odpoveďou na každý jednotlivý príkaz indikuje, či bol príkaz akceptovaný, či nastala dočasná, prípadne trvalá chyba alebo či server očakáva ďalšie príkazy. Po prenose e-mailu môže klient ukončiť spojenie alebo iniciovať prenos ďalšej správy [3].

1.2.2 ESMTP

SMTP protokol bol doplnený o rozšírenie, umožňujúce dohodnúť sa klientovi a serveru na funkcionalite, ktorú budú medzi sebou využívať a ktorá nie je implementovaná v pôvodnom SMTP protokole. Potreba rozšíriť pôvodný SMTP protokol sa objavila najmä preto, aby vyhovoval dnešným požiadavkám týkajúcich sa hlavne bezpečnosti e-mailovej komunikácie. Súčasná SMTP implementácia musí podporovať základný rozširujúci mechanizmus. Hlavnou myšlienkou je predstaviť rozšírený `HELO` príkaz, nazvaný `EHLO`, na základe ktorého, server rozozná klienta, ktorý podporuje ESMTP.

Server musí implementovať `EHLO` príkaz, aj keď neimplementuje žiadnu funkcionalitu nad rámec SMTP protokolu [3, 4].

1.2.3 Nájdenie cieľového príjemcu

Pri hľadaní cieľového príjemcu, ako prvé klient lexikálne identifikuje doménu, na ktorú má byť správa doručená. Tá musí mať plne kvalifikovaný názov, musí to byť tzv. FQDN. Prevedie sa DNS preklad domény a najskôr sa pozerá na to, či doména disponuje MX záznamom. Pokiaľ je nájdený CNAME záznam, vyhodnocovanie pokračuje rovnako, akoby bola obdržaná počiatočná doména. Ak doména nedisponuje žiadnym MX záznamom, tak je použitý adresný A záznam.

V prípade, že žiaden z MX záznamov nie je použiteľný alebo doména vôbec neexistuje, tak je situácia vyhodnotená ako chyba. MX záznam musí obsahovať adresný záznam pre nájdenie IP adresy, na ktorú má byť e-mail smerovaný [3].

1.2.4 Štruktúra komunikácie

Komunikácia prebieha po „riadkoch“. To znamená, že každá jednotlivá požiadavka od klienta alebo odpoveď od servera tvorí jeden riadok. Na SMTP príkaz prichádza SMTP odpoveď od odosielateľa. Odpoveď predstavuje pozitívne alebo negatívne potvrdenie od servera. Všeobecná forma odpovede je tvorená číselným kódom, zvyčajne nasledovaná textom. Riadky sú ukončené sekvenciou <CRLF> [3].

1.2.5 SMTP príkazy

Podľa [3] sú SMTP príkazy reťazce charakterov, ukončené medzerou <SP>, ak za nimi nasledujú nejaké parametre, inak sú ukončené <CRLF>.

Príkazy EHLO a HELO

Tieto príkazy slúžia na identifikáciu klienta pred serverom a inicializáciu komunikácie. Klient začína komunikáciu poslaním `HELO`/`EHLO` príkazu, na ktorý sa server predstaví v odpovedi.

Pri použití `EHLO` príkazu bude odpoveď od servera viacriadková a každý riadok reprezentuje podporované rozšírenie SMTP protokolu.

Ak sa klient rozhodne začať komunikáciu s `HELO` príkazom, tak je možné používať iba základné príkazy bez rozšírení.

Argumentom príkazu je plne kvalifikovaná doména klienta. Pokiaľ klient nedisponuje doménou, tak by mala byť argumentom jeho IP adresa. Preferovaný je `EHLO` príkaz namiesto `HELO` príkazu, ale kvôli spätnej kompatibilite musí byť `HELO` príkaz stále podporovaný všetkými SMTP servermi.
Príklad: `HELO fully.qualified.domain.com<CRLF>`.

Príkaz MAIL

Príkazom `MAIL` začína proces prenosu e-mailovej správy, v ktorom je doručená do jednej alebo viacerých poštových schránok. Prvý parameter `MAIL` príkazu je kľúčové slovo `FROM` a nasledujúci parameter je adresa odosielateľa správy.
Príklad: `MAIL FROM:<sender@mailexample.com><CRLF>`.

Príkaz RCPT

Príkaz `RCPT` identifikuje jedného alebo viacerých príjemcov e-mailovej správy. Príkaz môže byť opakovaný viac krát, pokiaľ je e-mailová správa určená pre viacerých príjemcov. Prvým parametrom je kľúčové slovo `TO` a nasledujúci parameter je e-mailová adresa príjemcu.
Príklad: `RCPT TO:<receiver@mailexample.com><CRLF>`.

Príkaz DATA

Príkazom `DATA` začína prenos textu e-mailovej správy. Po zadaní príkazu `DATA` server očakáva text správ, ktorý bude ukončený špeciálnou sekvenciou <CRLF>.<CRLF>.
Príklad: `DATA<CRLF>`.

Príkaz RSET

Príkazom `RSET` sa zruší aktuálne prebiehajúca transakcia. Všetky uložené dáta o odosielateľovi, prijímateľovi a text správy musia byť vymazané. Príkaz môže byť klientom poslaný v ľubovoľnom bode komunikácie.
Príklad: `RSET<CRLF>`.

Príkaz VRFY

Príkaz slúži na overenie, či zadané užívateľské meno alebo užívateľské meno aj s doménou existuje na serveri ako užívateľ alebo poštová schránka. Pokiaľ užívateľ existuje, odpoveďou je celé meno používateľa a jeho e-mailová adresa.

`VRFY` príkaz môže slúžiť ako zdroj e-mailových adries a v skutočnosti bol tento príkaz aj často zneužívaný na ich získavanie. Získané adresy boli potom použité napríklad na šírenie spamu. Z uvedeného dôvodu býva tento príkaz často vypnutý alebo povolený len pre overených používateľov. Je veľmi užitočný pre odladovanie a zbavovanie sa chýb na poštovom serveri.
Príklad: `VRFY username<CRLF>`.

Príkaz EXPN

Príkaz slúži na overenie, či zadaný názov zoznamu adries existuje na serveri. Pokiaľ tento zoznam existuje, server vráti zoznam mien jednotlivých užívateľov aj s ich e-mailovými adresami.

Z hľadiska bezpečnosti platí pre príkaz `EXPN` to isté, čo pre príkaz `VRFY`, a preto býva z bezpečnostných dôvodov vypnutý.
Príklad: `EXPN test-list<CRLF>`.

Príkaz NOOP

Príkaz `NOOP` nijakým spôsobom neovplyvňuje príkazy zadané pred odoslaním príkazu. Nešpecifikuje žiadnu akciu a server na tento príkaz odošle odpoveď „250 OK“. Zadané parametre sú ignorované serverom. Príkaz slúži napríklad na overenie, či spojenie medzi klientom a serverom je stále aktívne.
Príklad: `NOOP<CRLF>`.

Príkaz QUIT

Príkaz `QUIT` slúži na ukončenie spojenia medzi klientom a serverom. Server po prijatí tohoto príkazu musí odoslať odpoveď „221 OK“ a ukončiť spojenie s klientom. Server nesmie úmyselne uzavrieť spojenie skôr ako príjme `QUIT` príkaz od klienta, aj keď by sa v ich komunikácií vyskytla chyba. Príkaz môže byť klientom poslaný v ľubovoľnom bode komunikácie.
Príklad: `QUIT<CRLF>`.

1.2.6 SMTP odpovede

SMTP odpovede, zo strany servera, slúžia na zaistenie synchronizácie medzi požiadavkami klienta a akciami, ktoré server v súvislosti s týmito požiadavkami vykoná. Klient musí v každom bode komunikácie vedieť, v akom stave sa server nachádza. Každý príkaz generuje práve jednu odpoveď. SMTP odpoveď pozostáva z trojciferného kódu nasledujúceho nejakým textom.

Formálne je odpoveď definovaná ako sekvencia trojciferného kódu, <SP>, jedného riadku textu a <CRLF>. Pre viacriadkovú odpoveď platí, že každý riadok, až na posledný, začína opäť trojciferným kódom, nasledujúci pomlčkou „-“, textom a <CRLF>. Posledný riadok obsahuje namiesto pomlčky <SP>. Vo viacriadkovej odpovedi musí byť kód na každom riadku rovnaký [3]. Príklad viac riadkovej odpovede:

```
250-First line
250-Second line
250-234 Text beginning with numbers
250 The last line
```

1.2.6.1 Status kódy

Každá cifra v trojcifernom kóde má špeciálny význam. Prvá cifra indikuje, či je odpoveď kladná, záporná alebo neúplná. Druhá cifra slúži na špecifikovanie kategórie kódu. Napríklad, či ide o pripojenie alebo syntaktickú stránku príkazu. Tretia cifra poskytuje detailnejšie informácie ku každej kategórii špecifikovanej z druhej cifry [3].

Pre prvú cifru sú zadefinované štyri hodnoty:

2XX Požadovaná akcia bola úspešne dokončená a server je pripravený na ďalší príkaz.

3XX Požadovaný príkaz bol prijatý, ale server čaká na ďalšie informácie od klienta. Klient by mal poslať ďalší príkaz a doplniť tieto informácie.

4XX Požadovaný príkaz nebol serverom prijatý a žiadna akcia sa neuskutočnila. Chyba, ktorá nastala, je iba dočasná a klient by mal poslať príkaz, prípadne sériu príkazov s požadovanou akciou znova. Dočasná chyba

znamená, že pokiaľ klient zopakuje príkaz alebo sériu príkazov znova, bezo zmeny, tak akcia môže skončiť úspechom.

5XX Požadovaný príkaz nebol serverom prijatý a žiadna akcia sa neuskutočnila. Chyba, ktorá nastala je trvalá a klient by sa nemal pokúšať poslať rovnaký príkaz znova.

Pre druhú cifru sú zadefinované rovnako štyri hodnoty:

X0X Označuje syntaktickú chybu.

X1X Označuje odpovede týkajúce sa príkazov, ktoré poskytujú viac informácií, ako napríklad príkaz `HELP`.

X2X Označuje odpovede týkajúce sa spojenia medzi klientom a serverom.

X5X Označuje status poštového servera.

1.2.6.2 Rozšírené status kódy

Hlásenie chýb v SMTP protokole je značne obmedzené na neveľkú množinu kódov a popisného textu, ktorý je navyše systémovo závislý. Preto vznikla potreba pre bohatší popis stavu, chyby, takým spôsobom, ktorý by bol ľahko interpretovateľný strojmi a zároveň nebol systémovo závislý. Túto potrebu riešia rozšírené status kódy.

Každé číslo v rozšírenom kóde musí byť vyjadrené bez úvodných nulových číslic. Rozšírené status kódy pozostávajú z troch číselných polí, oddelených bodkou. Prvé číslo určuje, či pokus o doručenie bol úspešný. Druhé číslo napovedá zdroj anomálií, týkajúcich sa doručenia. Tretie číslo konkrétne špecifikuje chybu, ktorá nastala [5]. Syntax:

```
status-kód = trieda "." podtrieda "." detail
trieda = "2"/"4"/"5"
podtrieda = jedna až tri číslice
detail = jedna až tri číslice
```

1.2.7 Priebeh komunikácie

Komunikácia sa začne, keď klient nadviaže spojenie so serverom, ktorý mu odpovie s kódom 220 a pozdravom. SMTP protokol taktiež dovoľuje serveru odmietnuť spojenie. Pokiaľ server odmietne spojenie, musí odpovedať s kódom 554 a musí počkať na príkaz `QUIT` od klienta, aby mohol spojenie ukončiť. Na iný príkaz ako `QUIT` odpovedá kódom 503.

Po vytvorení spojenia medzi serverom a klientom, klient pošle príkaz `EHLO` alebo `HELO` serveru. Server na príkaz `HELO` nesmie odpovedať rozšírenou `EHLO`

odpoveďou. Ak server odpovie na `EHLO` príkaz *„command not recognized“*, klient by mal poslať príkaz `HELO`.

Transakciu e-mailu tvoria tri príkazy, ktoré klient odošle serveru. Prvý príkaz je `MAIL FROM` s e-mailovou adresou odosielateľa. Na tento príkaz server zareaguje resetovaním všetkých jeho stavov a prípravou na novú poštovú transakciu. Server akceptujúci tento príkaz odpovie kódom 250. Pri chybe server musí vrátiť odpoveď, ktorá bude klientovi indikovať, či ide o trvalú alebo dočasnú chybu. Druhým príkazom v transakcií je `RCPT TO` s e-mailovou adresou príjemcu. Ak užívateľ s poslanou e-mailovou adresou existuje na serveri, tak ten vráti kód 250. Na druhej strane, ak serveru nie je adresa známa, odpovie kódom 550. Tretím príkazom v transakcií je príkaz `DATA`. Na tento príkaz server odpovie kódom 354. Keď je koniec textu správy úspešne prijatý, server odpovie kódom 250 [3]. Ukážkovú komunikáciu je možné vidieť na obr. 1.2.

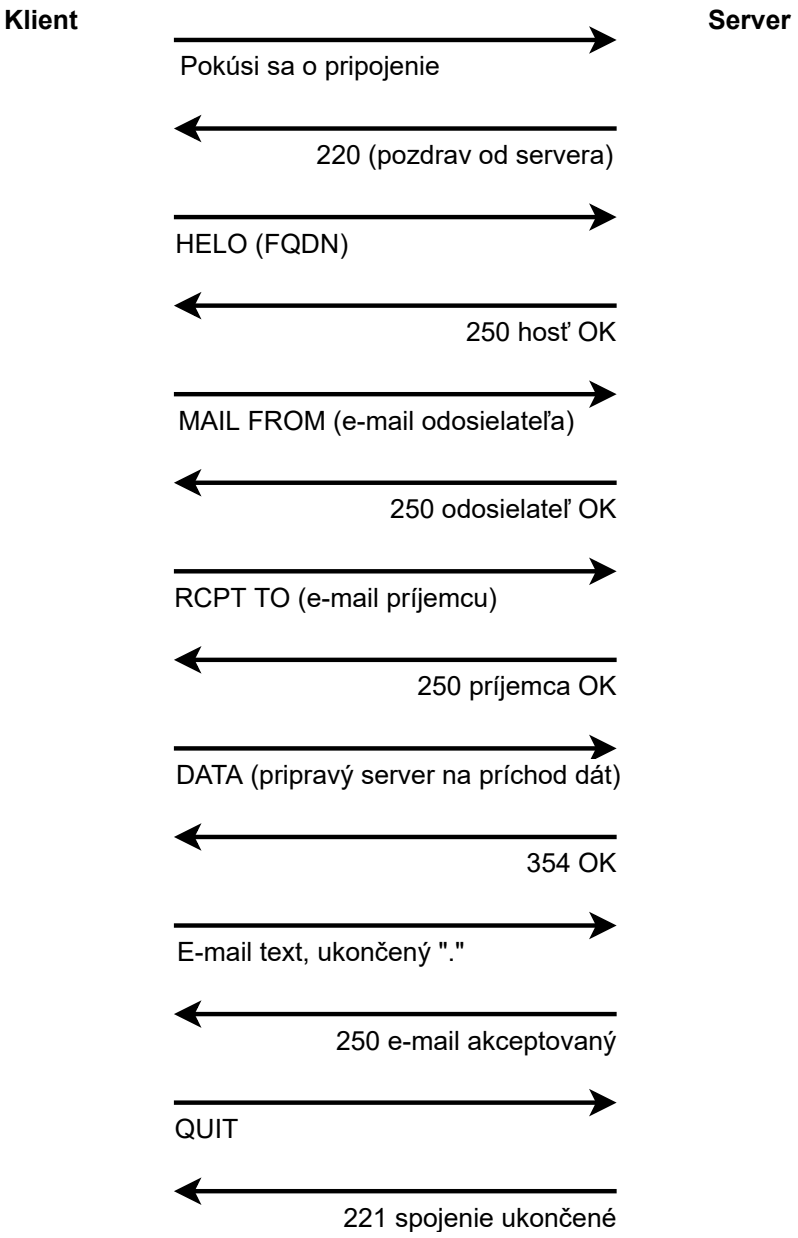
1.3 DNS

Komunikujúci hosť na internete môže byť identifikovaný mnohými spôsobmi. Jedným zo spôsobov je napríklad jeho doménové meno. Doménové mená ako *google.com* sú pre ľudí ľahko zapamätateľné. Avšak pre komunikáciu na internete poskytujú len veľmi málo informácií o tom, kde by mohol byť tento konkrétny hosť nájdený. Preto sú všetci hostia taktiež identifikovaní aj IP adresami. Všetky aplikácie, ktoré na internete zaisťujú komunikáciu medzi počítačmi, používajú na identifikáciu jednotlivých komunikujúcich uzlov IP adresu. Pre každú IP adresu je zavedené aj doménové meno. Toto doménové meno môže byť použité všade tam, kde je možné použiť IP adresu. Jedna IP adresa môže disponovať jedným, viacerými alebo žiadnym doménovým menom.

Väzba medzi doménovým menom a IP adresou hosťa je definovaná v databáze DNS. DNS je celosvetovo distribuovaná databáza, obsahujúca jednotlivé záznamy tzv. RR. Táto distribuovaná databáza je implementovaná v hierarchií DNS serverov. Jednotlivé časti databázy sú umiestnené na menných serveroch. Menné servery sa starajú o požiadavky na preklad doménového mena na IP adresu a opačne [6, 7].

1.3.1 Reverzný preklad

Ide o preklad IP adresy na doménové meno. Niektoré aplikácie potrebujú k IP adrese nájsť doménové meno, teda nájsť jej reverzný záznam. Niektoré poštové servery si týmto spôsobom overujú, či IP adresa, z ktorej požiadavka na odoslanie e-mailu pochádza, skutočne aj zodpovedá doméne, ktorá bola zadaná ako odosielateľ e-mailu [7].



Obr. 1.2: SMTP komunikácia

1.3.2 Resource Records (RR)

Ako vyplýva z [6], DNS servery, ktoré implementujú distribuovanú databázu ukladajú resource records, v rámci nich sa nachádzajú aj RR slúžiace na mapovanie IP adresy a doménového mena. Súčasťou každej DNS odpovede je jeden alebo viacero RR. Tieto záznamy obsahujú mimo iných atribútov meno, hodnotu a typ. Príklady RR:

- Typ **A** poskytuje základné mapovanie IP adresy a doménového mena.
- Typ **CNAME** je aliasom pre iné doménové meno, teda mapuje jedno doménové meno k ďalšiemu.
- Typ **MX** identifikuje poštové servery pod danou doménou a špecifikuje, že poštový server je schopný prijímať e-maily. Obsahuje dve polia, jedno určuje prioritu poštového servera a druhé obsahuje doménové meno tohto servera.
- Typ **PTR** záznam sa používa na reverzný preklad.

1.4 Editačná vzdialenosť

Editačnou operáciou na reťazci nazývame vloženie, zmazanie alebo zmenu jedného znaku.

Editačná vzdialenosť dvoch reťazcov $x = x_1...x_n$ a $y = y_1...y_m$ udáva, koľko editačných operácií je najmenej potrebných, aby z prvého reťazca vznikol druhý. Táto vzdialenosť sa označí $L(x, y)$.

Editačná vzdialenosť je niekedy nazývaná aj ako Levenshteinova vzdialenosť. Levenshteinova vzdialenosť je metrika pre meranie editačnej vzdialenosti [8].

V najkratšej postupnosti operácií sa každého znaku týka jedna editačná operácia, takže operácie je možné vždy usporiadať „zľava doprava“. Je možné si predstaviť, že reťazec x je prechádzaný od začiatku do konca a postupne je pretváraný na reťazec y a hľadá sa najmenší počet editačných operácií, ktorými to je možné dosiahuť.

Rekurzívne riešenie

Operácie, ktoré môžu nastať v optimálnej postupnosti na samotnom začiatku reťazca, sú:

- pokiaľ $x_1 = y_1$, to znamená, že prvý znak je ponechaný bez zmeny, potom $L(x, y) = L(x_2...x_n, y_2...y_m)$
- znak x_1 zmeníme na y_1, potom $L(x, y) = 1 + L(x_2...x_n, y_2...y_m)$
- znak x_1 vymažeme, potom $L(x, y) = 1 + L(x_2...x_n, y_1...y_m)$

- na začiatok vložíme y_1, potom $L(x, y) = 1 + L(x_1...x_n, y_2...y_m)$

$L(x, y)$ závisí od vzdialenosti podreťazcov x a y. Ak by boli tieto vzdialenosti známe, je jednoduché rozpoznať, ktorá z uvedených štyroch možností nastala a bola by to tá s najnižšou $L(x, y)$.

Ak sú vzdialenosti podreťazcov neznáme, vypočítajú sa rekurzívne, ako je možné vidieť v tabuľke 1.1. Rekurzia sa zastaví, ak je jeden z reťazcov už prázdny, vtedy je vzdialenosť rovná dĺžke druhého reťazca [9, 10].

$$L(x_1...x_n, y_1...y_m) = \begin{cases} n & m = 0 \\ m & n = 0 \\ L(x_2...x_n, y_2...y_m) & x_1 = y_1 \\ 1 + min \begin{Bmatrix} L(x_2...x_n, y_2...y_m), \\ L(x_2...x_n, y_1...y_m), \\ L(x_1...x_n, y_2...y_m) \end{Bmatrix} & x_1 \neq y_1 \end{cases}$$

Tabuľka 1.1: Rekurzívne vyjadrenie editačnej vzdialenosti

Iteratívne riešenie

Iteratívny spôsob spočíva v tom, že rekurzívny podproblém môže byť identifikovaný pomocou dvoch indexov a vďaka tomu je možné ukladať výsledky do tabuľky. Tabuľku T s výsledkami je možné prezentovať ako maticu, v ktorej každý prvok závisí iba od tých, ktoré ležia napravo a dole od neho. Túto tabuľku je možné preto vypĺňať po riadkoch zdola nahor, sprava doľava [9]. Ukážku algoritmu je možné vidieť v ukážke algoritmu 1.

Algoritmus 1 EditDistance

Vstup: Reťazce $A[1..n]$, $B[1..m]$
Výstup: Editačná vzdialenosť reťazcov A a B

```
1: procedure EditDistance(A[1..n], B[1..m])
2:     for i ← 1 to n + 1 do
3:         T[i, m + 1] ← n − i + 1
4:     end for
5:     for j ← 1 to m + 1 do
6:         T[n + 1, j] ← m − j + 1
7:     end for
8:     for i ← n to 1 do
9:         for j ← m to 1 do
10:            if A[i] = B[j] then
11:                cost ← 0
12:            else
13:                cost ← 1
14:            end if
15:            T[i, j] ← min(cost + T[i + 1, j + 1], 1 + T[i + 1, j], 1 + T[i, j + 1])
16:        end for
17:    end for
18:    return T[1, 1]
19: end procedure
```

Kapitola 2

Analýza

V kapitole sa v prvej časti zaoberám analýzou riešení problému validácie e-mailových adries, spolu s už existujúcimi riešeniami. V jej druhej časti sa venujem analýze riešení problému našepkávania a opravy doménovej časti e-mailovej adresy. V záverečnej časti kapitoly definujem požiadavky na základe vykonanej analýzy.

2.1 Analýza e-mail validácie

Bez toho, aby bol na ľubovolnú e-mailovú adresu odoslaný e-mail, nie je možné vždy a s istotou určiť, či daná e-mailová adresa existuje alebo nie. Predstavím niekoľko možných spôsobov, ako by bolo možné, aspoň pre väčšinu e-mailových adries určiť ich existenciu, bez toho, aby bol na ne odoslaný e-mail.

2.1.1 Validácia pomocou VRFY a EXPN

Ako najjednoduchší spôsob validácie e-mailových adries sa javia príkazy `VRFY` a `EXPN`, ktoré boli vytvorené na tento účel. Ako už bolo spomínané v teoretickej časti 1.2.5, kvôli neoprávnenému získavaniu e-mailových adries, sú tieto príkazy na väčšine serverov vypnuté. Z uvedeného dôvodu nie je možné používať tento spôsob na validáciu e-mailových adries.

2.1.2 Validácia založená na dátach

Sparkpost

Jeden z ďalších možných spôsobov je popísaný na stránkach služby sparkpost [11]. Ich validácia je založená na databáze „odtlačkov“ veľkého počtu e-mailových adries. Tento „odtlačok“ zahŕňa rôzne vlastnosti, spojené s danou adresou ako napríklad doručiteľnosť, syntax, koľko prostredníkov bolo potrebných k doručeniu e-mailu, ale aj informácie, či ide o dočasnú e-mailovú

adresu, výsledky DNS požiadaviek a mnoho ďalších. Na základe týchto dát, ktoré pravidelne aktualizujú, sa snažia vyhodnotiť, či je možné na danú e-mailovú adresu doručiť e-mail.

Twilio SendGrid

Na stránkach [12] služby, fungujúcej na podobnom princípe ako Sparkpost, je popísaná metóda riešenia validácie. Metóda je založená na strojovom učení, kde aplikovali konvolučnú a rekurentnú neurónovú sieť na zachytenie a zapamätanie si vzorov v e-mailových adresách. Špecificky sa zamerali na nasledujúce príznaky:

- syntax emailovej adresy – skúmajú, či adresa disponuje správnou syntaxou
- Resource Records – skúmajú, či doména e-mailovej adresy existuje a disponuje MX alebo A záznamami
- skupinový e-mail – skúmajú, či je e-mailová adresa používaná viacerými používateľmi, ako napríklad info@example.com
- dočasná e-mailová adresa – skúmajú, či ide o dočasnú e-mailovú adresu, ktorú používateľ jednorazovo použije
- preklepy – skúmajú, či zadal používateľ úmyselne alebo neúmyselne náhodnú sekvenciu znakov
- doručiteľnosť – skúmajú doručiteľnosť, resp. nedoručiteľnosť adresy

Každá e-mailová adresa je prevedená na maticu, ktorú potom neurónová sieť vyhodnotí a vráti odpoveď, s akou pravdepodobnosťou adresa neexistuje.

2.1.3 SMTP validácia

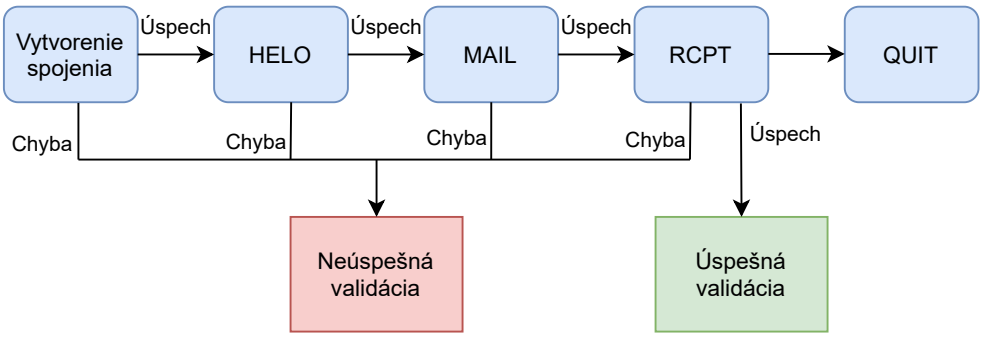
Často uvádzaným spôsobom validácie e-mailových adries je SMTP validácia. Inak nazývaná aj ako „SMTP ping“ alebo „broken SMTP handshake“. Podľa [13] a niekoľkých vlákien na rôznych fórum stránkach [14, 15, 16], je princíp tejto metódy nasledovný.

Validácia spočíva v tom, že sa vytvorí SMTP spojenie s poštovým serverom, na ktorom by mala existovať dopytujúca sa e-mailová adresa a bude sa simulovať odoslanie e-mailu na túto adresu. Po nadviazaní spojenia s poštovým serverom, klient zaháji komunikáciu `HELO` príkazom, následne pokračuje identifikácia odosielateľa `MAIL` príkazom a za tým sa použije kľúčový príkaz `RCPT TO` s parametrom e-mailovej adresy, ktorý sa kvázi pýta servera, či u neho táto adresa existuje. Po odpovedi servera na príkaz `RCPT` je validácia hotová a v tomto momente je odoslaný serveru príkaz `QUIT`, ktorým sa ukončí komunikácia so serverom. Tento spôsob by mal získať požadované informácie

a zároveň zaručiť, že na poštový server sa neodošle žiaden e-mail, pretože komunikácia je ukončená skôr, ako sa poslal `DATA` príkaz a teda obsah e-mailu samotný. Podľa odpovede na príkaz `RCPT` je možné určiť existenciu resp. neexistenciu e-mailovej adresy alebo prípadnú chybu. Postupnosť príkazov je možné vidieť na obr. 2.1.

Úspech znamená bežnú kladnú odpoveď od servera a chyba označuje bližšie nešpecifikovanú, zápornú odpoveď, kvôli ktorej server ďalej odmieta komunikovať s klientom.

Táto metóda znie síce jednoducho a priamočiaro, no prináša so sebou aj rad problémov.



Obr. 2.1: Jednotlivé kroky validácie e-mailových adries pomocou SMTP

2.1.4 Problémy SMTP validácie

Najpodstatnejším problémom, ktorý popisujú aj užívatelia v [14, 17, 18] je, že táto metóda môže byť zneužívaná spamermi, ktorí pomocou nej praktikujú istú formu slovníkového útoku. Skúšajú rôzne kombinácie užívateľských mien na danom poštovom serveri a overujú ich existenciu. Týmto spôsobom získajú valídne e-mailové adresy, ktoré môžu ďalej zneužiť na odosielanie spamu. Poštové servery s tým samozrejme počítajú a prirodzene pristupujú k opatreniam proti takémuto neoprávnenému získavaniu e-mailových adries.

Okrem iného, môžu po niekoľkých pokusoch validácie pridať na svoj blacklist IP adresu, z ktorej sa služba pripája na server.

Overenie skutočnosti IP adresy

Pri SMTP komunikácii s poštovým serverom, sa príkazu `HELO` ako parameter zadáva plne kvalifikované doménové meno servera z ktorého je `HELO` príkaz odosielaný. Veľký počet poštových serverov kontroluje reverzným DNS prekladom toto doménové meno, či skutočne patrí k IP adrese, z ktorej príkaz prichádza [19].

Greylisting

Ďalším opatrením, ktorým môže poštový server disponovať je greylisting. Ide o spôsob, ako zamedziť spamu tým, že prichádzajúci e-mail od neznámeho servera, poštový server dočasne odmietne. Toto odmietnutie trvá zvyčajne pár minút. Pokiaľ je odosielateľom e-mailu seriózny poštový server, tak sa pokúsi o opätovné doručenie e-mailu. Oproti tomu, spamer si nemôže dovoliť strácať čas opätovným odosielaním neúspešne doručeného e-mailu. Uvedená metóda je často využívaná hlavne kvôli tomu, že vygeneruje minimum práce naviac pre legitímny server, ale na druhú stranu, vygeneruje omnoho viac práce pre systém odosielajúci spam [20, 21].

Spamhouse

Mnoho poštových serverov využíva aj medzinárodnú službu *The Spamhaus Project* [22]. Ide o službu, ktorá sa snaží vystopovať spam a príbuzné kybernetické hrozby. Disponuje obrovskou databázou, ktorá blokuje valnú väčšinu spamu a malvéru na internete. Pokiaľ sa odosielateľova IP adresa objaví na spamhouse, je vysoko pravdepodobné, že poštové servery s ním nebudú vôbec komunikovať alebo pri pokuse zistiť adresáta pomocou `RCPT` príkazu budú odpovedať s chybou.

Dočasné e-mailové adresy

Problémom, ktorý nie je spojený s opatreniami poštových serverov, sú dočasné e-mailové adresy, pretože sú perfektne valídne, vzhľadom na metódu SMTP validácie. V súčastnosti existuje veľa služieb[1], ktoré poskytujú dočasné e-mailové adresy, slúžiace na prijímanie aj odosielanie e-mailov. Po krátkom používaní je táto adresa, aj celá jej poštová schránka vymazaná poskytovateľom služby. Uvedené adresy povoľujú užívateľom online registráciu, bez poskytnutia svojej skutočnej e-mailovej adresy [23].

Catch-all servery

Catch-all konfigurácia e-mailového serveru, ako už názov napovedá, je server akceptujúci všetky možné e-mailové adresy na danej doméne [24]. Pre metódu SMTP validácie to znamená, že ak sa vrámci `RCPT` príkazu služba spýta aj na neexistujúcu e-mailovú adresu, takýto catch-all server vždy odpovie kladne.

Dynamická IP adresa

Podľa [19] je odosielanie e-mailov zo zariadenia, ktorého IP adresa je dynamická veľmi nedôveryhodné, pretože to naznačuje zneužitie osobného počítača

[1] Napríklad https://temp-mail.org/

na rozosielanie spamu. Z tohoto dôvodu by malo zariadenie, na ktorom bude služba spustená, disponovať statickou IP adresou.

2.1.5 Existujúce služby

Existujúce služby na svojich stránkach jemne popisujú spôsob, akým validujú e-mail adresy. Niekoľko príkladov:

- na stránkach služby *Verify-Email* [25] sa píše: *„This tool works as SMTP Server: MX-Records are extracted from a DNS Server. Then the tool connects to SMTP servers and simulates the sending of a message.“*
- na stránkach služby *QuickEmailVerification* [26] sa píše: *„QuickEmail-Verification performs deep level SMTP verification on each email address. Each email address will be verified for mailbox existence without sending actual emails on that address.“*
- na stránkach služby *Hunter* [27] sa píše: *„The email checker does the validation without sending an email, by directly connecting to the SMTP server.“*

Na základe uvedeného usudzujem, že využívajú podobný princíp validácie, aký som popísal v časti 2.1.3 a že nejde úplne o „slepú uličku“.

2.1.6 Riešenie zabanovania

Ako sa dá teda riešiť zabanovanie IP adresy zo strany servera?

Podľa Ing. Jana Fesla, Ph.D. nie je úplne možné tomu zamedziť a aj existujúce služby musia čeliť podobnému problému, pokiaľ nie sú explicitne dohodnuté s daným poštovým serverom. Pán Fesl v e-maile [28] píše, že čiastočným riešením môže byť aj IP pool s reverznými záznamami na doménu, ktoré sú zásadné v dôveryhodnosti poštového servera, inak si nemyslí, že by sa na tom technicky dalo ešte niečo zlepšiť.

Martin Wong vo svojej odpovedi vo vlákne [29], na otázku „ako sa nedostať pri SMTP validácií na blacklist servera“, tvrdí, že je zakladateľom obdobne fungujúcej služby[2]. Riešenie naznačuje v dobrej infraštruktúre, monitoringu, inteligentnom dizajne a minimalizovaní požiadaviek na poštový server.

2.2 Analýza dopĺňania a korekcie domény

Dopĺňanie a korekcia doménovej časti e-mailu je veľmi užitočná funkcia. Nielenže užívateľom pomáha rýchlejšie vyplniť formulárové políčko, taktiež sa postará aspoň o čiastočnú správnosť vstupu. Ak užívateľ aj napriek tomu zadá doménu s preklepom, bude mu ponúknutá možná oprava. Dopĺňanie domény

[2] https://mailfloss.com/

nesmie byť pomalé, zoznam dopĺňaných domén by sa mal aktualizovať ihneď, ako užívateľ dopíše ďalší znak do formulára. V prvej časti analýzy predstavím niekoľko možných spôsobov, ako užívateľovi ponúkať dopĺňanie doménovej časti e-mailu. V druhej časti rozoberiem niekoľko spôsobov, ako ponúknuť užívateľovi opravu domény s preklepom.

2.2.1 Dopĺňanie domény

Naivný prístup

Najjednoduchším spôsobom, akým je možné implementovať dopĺňanie domén, je hrubou silou. Vždy keď užívateľ zadá ľubovoľný znak, musel by sa preiterovať celý zoznam známych domén a každá doména by sa musela porovnať, či začína na sekvenciu znakov, ktoré užívateľ zadal. Tento prístup vedie na $\mathcal{O}(n * m)$ zložitosť, kde n je počet známych domén a m je počet znakov zadaných užívateľom.

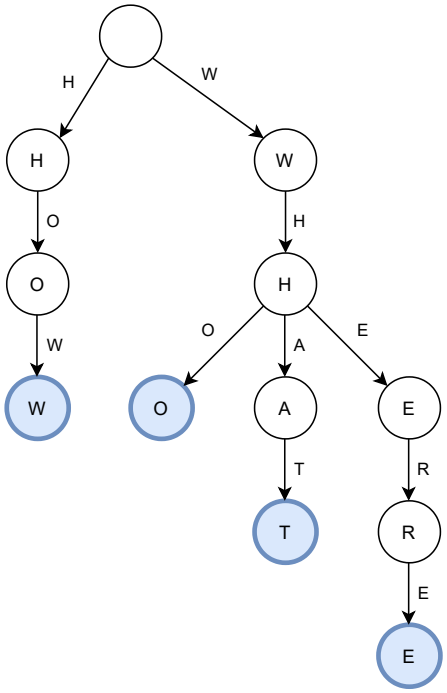
Trie

Trie alebo písmenkový strom je dátová štruktúra, ktorá slúži k uloženiu množiny dvojíc (kľúč, hodnota) a v ktorej sa kľúč neopakuje. Konkrétnejšie táto dátová štruktúra ukladá množinu slov - reťazcov, zložených zo znakov nejakej pevnej, konečnej abecedy a každému slovu môže priradiť nejakú hodnotu. Trie má tvar zakoreneného stromu. Z každého vrcholu vedú hrany označené navzájom rôznymi znakmi abecedy. V koreni vyjadrujú prvé písmeno slova, o poschodie nižšie druhé, a tak ďalej. Vrcholom môžeme priradiť reťazce tak, že prečítame všetky znaky na ceste z koreňa do daného vrcholu. Koreň zodpovedá prázdnemu reťazcu. Vrcholy, ktoré zodpovedajú slovám kľúčov, označíme a uložíme do nich hodnoty priradené kľúčom. Označené môžu byť aj vnútorné vrcholy, ak je jeden kľúč pokračovaním iného. Každý vrchol si bude pamätať pole ukazateľov na synov (ako indexy slúžia znaky abecedy), toto pole môže byť nahradené aj inou množinovou dátovou štruktúrou. Ďalej si bude pamätať značku, či ide o slovo kľúča a prípadne hodnotu priradenú tomuto slovu.

Vyhľadávanie začne v koreni a postupne nasleduje hrany podľa jednotlivých písmen hľadaného slova. Slovo sa v trie nachádza, ak budú všetky hrany existovať a vrchol do ktorého sa došlo, obsahuje značku, že ide o slovo. Ak by sa stalo, že z nejakého vrcholu ďalší znak odkazuje na neexistujúcu hranu, tak dané slovo sa v trie nenachádza. Časová zložitosť je lineárna s dĺžkou hľadaného slova, teda $\mathcal{O}(n)$, kde n je dĺžka hľadaného slova [9]. Ukážku trie so slovami how, what, where, who je vidieť na obr. 2.2.

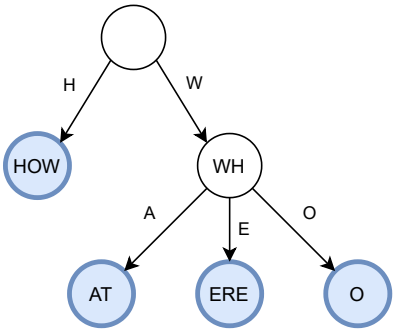
Radix Trie

Radix trie je odvodená dátová štruktúra od trie. Ide o kompaktnejšiu a pamäťovo menej náročnú dátovú štruktúru, pretože prináša jedno pravidlo oproti



Obr. 2.2: Trie pre slová how, what, where, who [30]

trie naviac. Pravidlo hovorí, že z každého vnútorného vrcholu by mali viesť aspoň dve a viac hrán [30]. Po sebe nasledujúce vrcholy, z ktorých vedie iba jedna hrana, sú kombinované do jedného vrcholu, ako je možné vidieť na obr. 2.3.



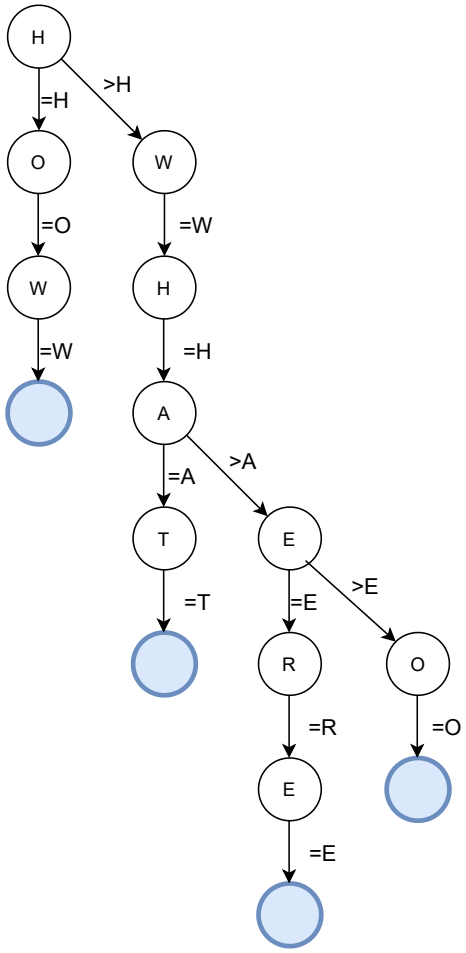
Obr. 2.3: Radix trie pre slová how, what, where, who [30]

Ternary search tree (TST)

Hlavná nevýhoda trie dátovej štruktúry je vysoká pamäťová náročnosť. TST kombinuje vlastnosti binárnych vyhľadávacích stromov a trie dátovej štruktúry. Každý vrchol TST, rovnako ako trie, ukladá jeden znak (časť kľúča) a je

pamäťovo efektívny ako binárny vyhľadávací strom, až na to, že každý vrchol má tri deti namiesto dvoch. Ľavý syn ukladá znak, ktorého hodnota je menšia než hodnota znaku v aktuálnom vrchole. Pravý syn ukladá znak, ktorého hodnota je väčšia než hodnota znaku v aktuálnom vrchole. Stredný syn ukladá ďalší znak slova. Naviac, každý vrchol ukladá značku, či ide o koniec slova. Teda každý vrchol reprezentuje predponu uloženého reťazca.

Pri vyhľadávaní sa porovnáva aktuálny znak v hľadanom reťazci so znakom uloženým vo vrchole. Ak je aktuálny znak menší ako znak vo vrchole, hľadanie prejde do ľavého syna, opačne ak je väčší, hľadanie prejde do pravého syna. Ak sa aktuálny znak rovná znaku vo vrchole, hľadanie prejde do stredného syna a pokračuje sa s ďalším znakom v hľadanom reťazci. Zložitosť hľadania reťazca o dĺžke k v TST, kde n je počet všetkých uložených reťazcov bude $\mathcal{O}(\log n + k)$ [31, 32]. Ukážku stromu pre vložené slová v poradí how, what, where, who je možné vidieť na obr. 2.4.



Obr. 2.4: Ternary search tree pre slová how, what, where, who

Zvolené riešenie

Z dôvodov, že doménová časť e-mailovej adresy má pomerne krátky rozsah a počet domén, ktoré bude nástroj dopĺňať, sa bude pohybovať maximálne v jednotkách tisícov, je najlepšia voľba použiť trie dátovú štruktúru. Je to najmä z dôvodu, že pri tak malom počte slov, ktoré budú slúžiť na dopĺňanie, nenastane žiaden pamäťový problém a nemá zmysel zvoliť radix trie. TST nie je tak efektívny, ako trie a pamäťovú efektivitu neberiem kvôli spomínaným dôvodom do úvahy.

2.2.2 Korekcia domény

Korekciu domény je možné implementovať mnohými spôsobmi. V tejto časti sa zameriam na niekoľko spôsobov využívajúcich editačnú vzdialenosť.

Naivný prístup

Najjednoduchší spôsob, akým by bolo možné navrhovať opravy domény, je vypočítať editačnú vzdialenosť pre každú uloženú doménu a vybrať tie s najnižšími hodnotami.

BK-Tree

BK-Tree alebo Burkhard Keller Tree predstavuje stromovú dátovú štruktúrú, slúžiacu na rýchle vyhľadanie podobných reťazcov. Kľúčovým pre vznik tejto dátovej štruktúry bol fakt, že Levenshteinova vzdialenosť formuje metrický priestor a nasledujúce pozorovanie.

Predpokladajú sa dva parametre, prvý je reťazec, napríklad *dotaz*, ku ktorému sa majú nájsť slová s maximálnou Levenshteinovou vzdialenosťou n a druhým parameterom je uvedená maximálna vzdialenosť n. Zoberie sa ľubovolný reťazec, napríklad *test* a porovná sa s reťazcom *dotaz*. Nech je výsledná Levenshteinova vzdialenosť d. Pretože platí trojuholníková nerovnosť, kvôli tomu, že Levenshteinova vzdialenosť je metrika, všetky výsledky musia mať najväčšiu vzdialenosť $d + n$ a najmenšiu vzdialenosť d - n.

Každý vrchol má ľubovoľný počet detí a každá hrana predstavuje číslo, zodpovedajúce Levenshteinovej vzdialenosti. Všetci potomkovia vedúci z hrany označenej n majú vzdialenosť presne n k vrcholu, z ktorého vedie daná hrana.

Vyhľadávanie v strome prebieha tak, že sa spočíta Levenshteinova vzdialenosť d hľadaného slova a koreňa stromu a hľadanie ďalej rekurzívne pokračuje s každým dieťaťom, na ktoré vedie hrana očíslovaná medzi d - n a $d + n$ vrátane. Pokiaľ vzdialenosť vrcholu od hľadaného slova je vrámci maximálnej vzdialenosti, tak vrchol bude vrátený a hľadanie pokračuje ďalej [33, 34].

Prístup Petra Norviga

Peter Norvig vo svojom blogu [35] popísal ďalší možný spôsob ako navrhnúť opravu slova. Jeho metóda funguje tak, že zo slova, ku ktorému chce nájsť možné opravy, vygeneruje všetky možné slová vrámci danej editačnej vzdialenosti. Teda pomocou operácií ako vymazanie, pridanie, nahradenie znaku alebo prehodenie dvoch znakov. Výsledné slová potom hľadá v množine všetkých známych slov.

Steve Hanovo riešenie

Vo svojom blogu [36] Steve Hanov popisuje ďalší spôsob ako navrhnúť opravu slov. Pri počítaní Levenshteinovej vzdialenosti dvoch slov, napríklad hľadaného slova *kate* a slova z množiny všetkých slov, napríklad *cat* musí algoritmus vyplniť N * M tabuľku 2.1. No pokiaľ ďalšie slovo bude napríklad *cats*, algoritmus musí znova vyplniť celú tabuľku 2.2, pritom jedinou zmenou je pridaný spodný riadok pre znak *s*. Preto je možné eliminovať veľké množstvo opakovaného výpočtu tým, že by sa slová spracovávali v istom poradí. A presne to dokáže zaručiť dátová štruktúra trie.

		k	**a**	**t**	**e**
	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Tabuľka 2.1: Výpočet editačnej vzdialenosti pre slová kate a cat

		k	**a**	**t**	**e**
	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2
s	4	4	3	2	2

Tabuľka 2.2: Výpočet editačnej vzdialenosti pre slová kate a cats

Zvolené riešenie

Ako výsledné riešenie som zvolil metódu od Steve Hanova, pretože ako uvádzam v časti 2.2.1, už budem disponovať trie dátovou štruktúrou a táto metóda vie s ňou spolupracovať. Ďalším dôvodom je tá skutočnosť, že napríklad proti BK stromu sa nemusím limitovať výberom iba Levenshteinovej vzdiale-

nosti, ale môžem zvoliť aj jej rozšírené varianty, ktoré už netvoria metrický priestor, ale pri aktuálne zvolenom riešení to ani nie je podmienkou.

2.3 Analýza požiadaviek

Dôležitým bodom analýzy je definovanie požiadaviek. V tejto časti je obsiahnutý zoznam funkčných a nefunkčných požiadaviek, ktoré som definoval na základe analýzy možných riešení popísanej vyššie v texte a na základe konzultácií s vedúcim práce, ktorý je zároveň aj zadávateľom.

2.3.1 Funkčné požiadavky

Funkčné požiadavky definujú správanie systému, inými slovami, popisujú funkcionalitu, ktorú systém musí byť schopný vykonávať [37].

- **F1: Syntaktická validácia** Služba bude e-mailovú adresu validovať po syntaktickej stránke.
- **F2: Kontrola MX záznamov** Služba dokáže zistiť existenciu MX záznamov doménovej časti e-mailovej adresy. Ďalej dokáže získať tieto záznamy a poskytnúť ich pre SMTP validáciu. V neposlednom rade si bude služba pamätať, pre danú doménu posledný použitý MX záznam a ak doména disponuje viacerými, tak poskytnúť ten ďalší a postupne ich striedať.
- **F3: SMTP validácia** Služba bude schopná pripojiť sa na poštový server, na ktorý ukazuje doménová časť e-mailovej adresy. Ďalej bude vedieť komunikovať s týmto serverom a nasimulovať odoslanie e-mailu. Vrámci tejto simulácie služba overí existenciu e-mailovej adresy.
- **F4: Rozpoznanie odpovedí** Služba bude rozpoznávať najčastejšie kódy vyskytujúce sa v odpovedi od poštového servera. Na základe odpoveďového kódu služba rozhodne o existencií e-mailovej adresy.
- **F5: Caching** Služba bude umožňovať caching výsledkov MX záznamov počas určitej doby. Caching sa bude vzťahovať aj na hash kódu e-mailovej adresy spolu s výsledkom validácie. Tie sa uložia do databázy, kvôli tomu, aby sa čo najviac minimalizoval počet validácií, ktoré budú prichádzať ako požiadavky na server.
- **F6: Doplňovanie domény** Služba bude poskytovať možné doplnenie domény na základe rozpísanej časti, vráti zoradené domény podľa počtu použitia.
- **F7: Oprava domény** Služba bude poskytovať možnú opravu domény, ak užívateľ zadá doménu s preklepom.

- **F8: Logovanie domén** Služba bude logovať doménové časti takých e-mailových adries, ktoré maju správnu syntax a doménová časť je existujúca doména s MX záznamom. Logovať sa bude do súboru, kde na každý deň sa vytvorí nový súbor, do ktorého sa bude logovať.

2.3.2 Nefunkčné požiadavky

Nefunkčné požiadavky popisujú obmedzenia kladené na systém a ďalšie jeho vlastnosti, ako napríklad: výkonnosť, udržateľnosť, škálovateľnosť, bezpečnosť, spoľahlivosť a mnoho ďalších [37].

- **N1: Jednoduchá integrácia** Službu bude možné jednoducho integrovať do existujúcej webovej aplikácie.
- **N2: Serverové požiadavky** Službu bude schopná obslúžiť viac ako 70 000 požiadaviek za deň.

Kapitola 3

Návrh

V nasledujúcej kapitole popisujem návrh výslednej služby. V jej prvej časti predstavím použité technológie a v druhej časti samotný návrh služby.

3.1 Použité technológie

Ako hlavný programovací jazyk, použitý na vývoj, som vybral Javu. V spoločnosti, pre ktorú je služba určená, používajú vývojári konkrétne Javu 12. Aby mohla byť aplikácia plynule spustiteľná a mohla sa rozširovať funkcionalita, bez potreby inštalovania ďalšej verzie a riešení problémov spojených s nekompatibilitou, ktoré by mohli nastať pri výbere odlišnej verzie Javy, som sa aj ja rozhodol pre verziu 12.

3.1.1 Spring Boot

Spring Boot je jedným z najpopulárnejších frameworkov pre tvorbu nie len webových aplikácií v Jave. Ide o nadstavbu populárneho Spring Frameworku. Spring vznikol s cieľom ponúknuť odľahčenú verziu ku JEE. Hlavná nevýhoda Springu spočíva v množstve konfigurácie, ktorá zaťažovala vývojárov. Prvotne sa aplikácie konfigurovali pomocou XML súborov. Spring 2.5 ponúkol skenovanie komponent na základe anotácií, čo výrazne znížilo množstvo potrebných konfigurácií, avšak stále veľká časť zostávala na vývojároch. Napríklad pri práci s JPA alebo web MVC sa vyžaduje množstvo osobitnej konfigurácie.

Uvedený problém rieši Spring Boot. Prináša automatickú konfiguráciu bežných scénárov pri vývoji aplikácií a umožňuje vývojárom zamerať sa priamo na kód aplikácie. Zároveň dovoľuje prepísať implicitnú konfiguráciu, ak to vývojár potrebuje. Spring Boot so sebou prináša omnoho viac komponent a konceptov na uľahčenie vývoja. Za zmienku určite stojí vstavaný aplikačný server. Nevýhodou Spring Bootu je, že kvôli automatickej konfigurácií, aplikácia disponuje veľkým počtom závislostí, ktoré nie nutne vždy používa [38].

Hlavný dôvod výberu tohoto frameworku pre jazyk Java, je jeho popularita a vysoká efektivita pri tvorbe aplikácií.

3.1.2 REST API

Webové služby sú účelovo založené webové servery, ktoré podporujú požiadavky iných aplikácií. Klientsky program používa API na komunikáciu s webovými službami. REST architektúra je bežne používaná pre návrh API moderných webových služieb. Nie je viazaná na konkrétny protokol, ale vo väčšine sa pri komunikácii používa HTTP protokol. Architektúra REST rozdeľuje menný priestor na množinu zdrojov, ktoré sú identifikované pomocou URI a k tomu nad každým zdrojom definuje štandardné HTTP operácie ako GET, POST, PUT a DELETE. Prístup k dátam je zabezpečený práve pomocou zmienených zdrojov [39, 40].

3.1.3 PostgreSQL

PostgreSQL je open source objektovo-relačná databáza [41]. Služba využíva veľmi jednoduchý databázový model a na výber databázy neboli kladené žiadne špeciálne požiadavky. PostgreSQL som zvolil z dôvodu, že spoločnosť, pre ktorú je služba určená, využíva túto databázu a vývojári nebudú mať prácu naviac s inštaláciou inej databázy.

3.1.4 Jenkins

Continuous Integration je fáza, ktorá poskytuje prvú spätnú väzbu vývojárovi. Počas tejto fázy sa stiahne kód z repozitára, spustí sa jeho kompilácia, unit testy a overí sa kvalitu kódu. Táto fáza môže disponovať mnohými ďalšími škálovateľnými krokmi. Ak ľubovoľný z týchto krokov skončí s chybou, ďalší krok v poradí sa už nevykoná a vývojár by mal vyriešiť chybu, ktorá nastala. Hlavným cieľom je nájsť chyby v kóde rýchlejšie, zlepšiť kvalitu softwaru a minimalizovať čas overenia správnosti a funkčnosti aplikácie spolu s jej dodaním.

Jenkins je open source server napísaný v Jave. Disponuje širokou škálou rozširujúcich pluginov a aktuálne je to najpopulárnejší nástroj ako implementovať Continuous Integration proces [42].

3.1.5 Docker

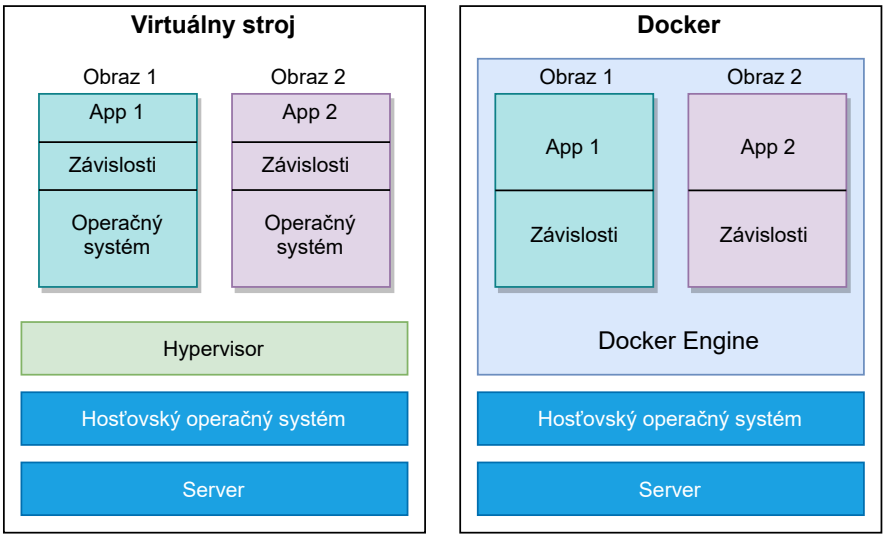
Docker je open source nástroj, ktorý uľahčuje vytváranie, nasadenie a chod aplikácií v izolovaných prostrediach, tzv. kontajneroch [42].

Kontajnerizácia vs virtualizácia

Izolácia aplikácií, ako aj ďalšie benefity, môžu byť dosiahnuté pomocou virtuálnych strojov. Virtuálny stroj napodobňuje počítačovú architektúru a poskytuje funkcionalitu fyzického počítača. Kompletnú izoláciu aplikácií je možné dosiahnuť, ak je každá z nich spustená ako separátny obraz. Obraz je šablóna, obsahujúca softwarovú konfiguráciu, ktorej súčasťou býva aj operačný systém a používa sa na vytváranie nových instancií. Tieto obrazy spúšťa tzv. hypervisor, ktorý napodobuje fyzický počítač. Nevýhody tohoto prístupu sú:

- nízky výkon – tým, že virtuálny stroj napodobuje celú počítačovú architektúru, je s tým spojená aj vysoká réžia každej operácie
- vysoká spotreba zdrojov – napodobovanie počítačovej architektúry musí fungovať separátne pre každú aplikáciu
- veľkosť obrazov – každá aplikácia je doručovaná s operačným systémom, čo má za následok vysokú veľkosť obrazu

Kontajnerizácia na druhej strane funguje odlišne. Každá aplikácia je spustená v separátnom kontajneri, je doručená so závislosťami, ktoré potrebuje, ale už bez operačného systému. Kontajnery zdieľajú operačný systém a sú spustené ako izolované procesy na tomto operačnom systéme. To ma za následok, že tu nie je žiadna medzi vrstva, neplytvá sa tak výkonnostnými zdrojmi a aj jednotlivé obrazy majú výrazne menšiu veľkosť [42]. Porovnanie je možné vidieť na obr. 3.1.



Obr. 3.1: Porovnanie virtuálneho stroja s dockerom [42]

Docker compose

Docker compose je nástroj na definovanie, spustenie a manažovanie viac kontajnerových doker aplikácií. Jednotlivé služby sú definované v konfiguračnom súbore a môžu byť vytvorené a spustené pomocou jediného príkazu [42].

3.1.6 Vue.js

Vue.js je open source pokrokový JavaSript framework pre tvorbu užívateľského rozhrania a jednostránkových aplikácií. Jadro celého frameworku je zamerané najmä na prezentačnú vrstvu aplikácie [43].

Uvedený framework som vybral hlavne kvôli tomu, že je jednoduchý na pochopenie a vďaka tomu ponúka veľmi rýchly štart pri tvorbe front-end aplikácií.

3.2 Návrhové vzory

Návrhový vzor predstavuje všeobecné riešenie bežne vyskytujúceho sa problému v softwarovom dizajne.

3.2.1 Visitor

Hierarchia tried so spoločným predkom, deklarujúcim metódu, ktorá je v každej podtriede implementovaná osobitne, nemusí byť vždy najlepším riešením. Výhodou takéhoto riešenia je, že po pridaní novej triedy kompilátor donúti programátora implementovať špecifický kód aj pre túto novú triedu. Na druhej strane tohoto špecifického kódu môže byť veľké množstvo alebo sa svojím charakterom nehodí do danej triedy. Tento problém rieši *visitor* návrhový vzor. Kód zodpovedný za prídavnú funkcionalitu pre každú podtriedu, sa presunie do osobitnej triedy, zvanej *Visitor*. V tejto novej triede vznikne metóda pre každú podtriedu, ktorej parametrom bude práve instancia tejto podtriedy. Uvedený parameter je dôležitý, pretože poskytuje potrebné dáta na realizáciu akcie, ktorá by bola inak implementovaná priamo v danej podtriede. Pre volanie uvedených metód *Visitor* používa techniku zvanú *Double Dispatch*, ktorá pomáha zavolať správnu metódu bez ťažkopádnych podmienok. Výber správnej metódy sa deleguje každej podtriede, ktorá bude v metóde prijímať ako argument *Visitor* triedu. Každá podtrieda už bude vedieť, ktorú konkrétnu metódu *Visitor* triedy má zavolať. Danú metódu zavolá a vloží tam seba ako argument [44].

3.2.2 Proxy

Existujú situácie, v ktorých je potrebné istý objekt nahradiť zástupným objektom. Zástupný objekt sa nazýva *Proxy* a navonok poskytuje úplne rovnaké rozhranie ako objekt ktorý nahrádza. Hlavným dôvodom je, aby sa mohlo

pozmeniť správanie objektu a zároveň, aby sa ďalšie objekty závislé od nahradzujúceho objektu nemuseli prispôsobiť novému rozhraniu. *Proxy* objekt kontroluje prístup do originálneho objektu a povolí vykonávať operácie pred alebo potom ako sa požiadavka dostane do pôvodného objektu [44].

3.3 Návrh služby

Služba pozostáva z dvoch podprojektov. Prvý projekt riešiť samotnú validáciu e-mailových adries a druhý dopĺňanie a opravu domén. Prvý projekt obsahuje závislosť na druhom a disponuje spoločnou API, ako pre validáciu e-mailov, tak aj pre dopĺňanie a opravu domén. Súčasťou služby je aj jednoduchá javascriptová knižnica, ktorá slúži na integráciu klienta so službou.

3.3.1 Validácia e-mailových adries

V tejto časti je predstavený návrh tried súvisiacich s validáciou e-mailových adries. Diagram tried na obr. 3.2 ukazuje štruktúru kľúčových tried a vzťahov medzi nimi. Nasleduje stručná charakteristika jednotlivých tried. Interface `EmailValidation` spolu s `EmalValidationService`, `EmailValidationProxy`, tak isto aj `MXValidation, MXValidationService, MXValidationProxy` implementujú *Proxy* 3.2.2 návrhový vzor.

`Controller` Predstavuje triedu poskytujúcu endpointy, ako pre validáciu e-mailových adries, tak aj pre dopĺňanie a opravu doménových častí.

`DomainCompletionService` Trieda poskytujuca metódy, zabezpečujúce dopĺňanie a opravu doménových častí e-mailových adries.

`EmailValidation` Rozhranie poskytujúce metódy pre validáciu e-mailových adries. Metóda `validateEmail` slúži na validáciu e-mailovej adresy a vracia výsledok validácie. Metóda `validateEmailWithFileLog` uskutoční validáciu e-mailovej adresy a zároveň zapisuje do súboru priebeh komunikácie s poštovým serverom. Metóda slúži na testovacie účely.

`EmalValidationService` Servisná trieda implemnetujúca logiku validácie e-mailových adries. Validácia pozostáva z troch častí. Ako prvá sa overí syntaktická stránka e-mailovej adresy, ak bude v poriadku, prejde sa na druhú časť. V druhom kroku sa vyhľadajú doménové mená pre poštové servery danej domény. Tretím krokom je samotná SMTP validácia.

`EmailValidationProxy` Trieda predstavuje prostredníka medzi `Controller` triedou a `EmailValidationService` triedou. Zabezpečuje caching už zvalidovaných e-mailových adries.

`SyntaxValidation` Trieda slúži na validáciu syntaktickej stránky e-mailovej adresy.

`SMTPValidation` Trieda vykonáva SMTP validáciu e-mailových adries.

`MXValidation` Rozhranie poskytuje metódu na vyhľadanie doménových mien poštových serverov zadanej domény a súčasne overuje existenciu domény.

`MXValidationService` Servisná trieda, ktorá implementuje logiku vyhľadávania doménových mien poštových serverov zadanej domény, ktorým validuje existenciu domény a zároveň poskytuje nájdené výsledky pre SMTP validáciu.

`MXValidationProxy` Trieda slúži ako prostredník, zabezpečuje caching výsledkov DNS požiadaviek.

SMTP validácia

Triedy zodpovedné za SMTP validáciu implementujú *Visitor* 3.2.1 návrhový vzor. Diagram tried je možné vidieť na obr. 3.3. Trieda `SMTPValidation` je zodpovedná za správne posielanie jednotlivých príkazov na poštový server. Rovnako je zodpovedná aj za prijímanie odpovedí od servera a rozhodovanie o stave na základe odpovede.

3.3.2 Dopĺňanie a oprava domén

Štruktúru tried je možné vidieť na obr. 3.4. Nasleduje stručný popis funkcionalít jednotlivých tried.

`DomainCompletionService` Hlavná trieda poskytujúca metódu na dopĺňanie domén a metódu na opravu domén. Trieda využíva `TrieMap` dátovú štruktúru, pri inicializácii triedy sa vyberú z databázy všetky domény, ktorými sa `TrieMap` inicializuje.

`DomainCompletionDao` Trieda slúži na prístup k databáze.

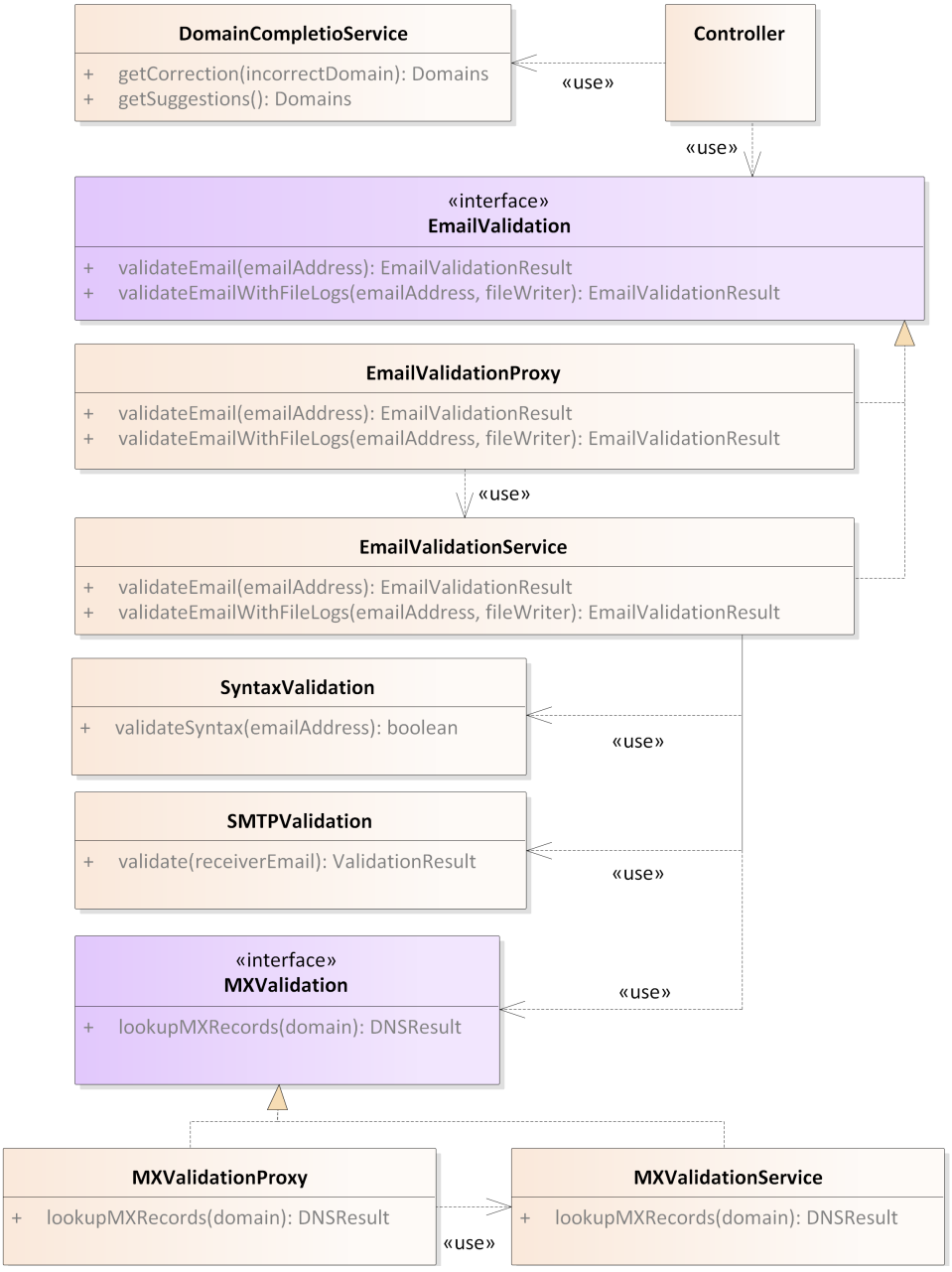
`TrieMap` Trieda implementuje Trie dátovú štruktúru.

`EditDistance` Rozhranie poskytujúce metódy na výpočet editačnej vzdialenosti dvoch slov.

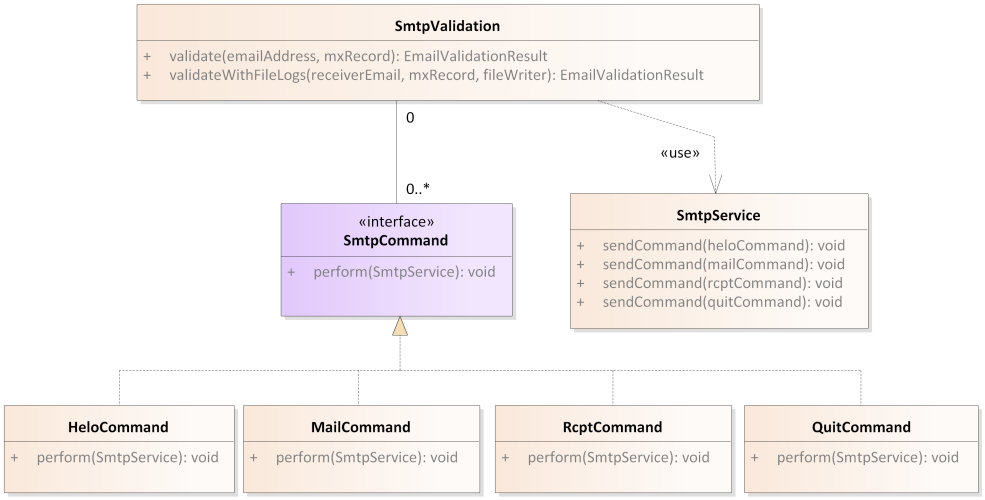
`OSADistance` Trieda reprezentuje jeden z možných spôsobov implementácie editačnej vzdialenosti.

`LettersComparator` Rozhranie, ktoré vystavuje metódu slúžiacu na porovnanie dvoch znakov. Čím sa znaky viac podobajú, tým nižšie číslo vracia.

`KeyboardCloseLettersComp` Implementácia `LettersComparator`, ktorá porovnáva dva znaky. Pokiaľ sú znaky blízko seba na klávesnici, tak vracia nižšiu hodnotu ako keď sú znaky vzdialené viac ako o jednu klávesu.



Obr. 3.2: Štruktúra kľúčových tried, z ktorých je služba zostavená



Obr. 3.3: Štruktúra tried zodpovedných za SMTP validáciu e-mailovej adresy

3.3.3 Databázový model

Databázový model je veľmi jednoduchý. Pre potreby výslednej služby existujú v databáze, ktorá je na obr. 3.5, dve tabuľky. Tabuľka `email_validation` slúži na ukladanie už zvalidovaných e-mailových adries počas istej doby. Konkrétne sa ukladá iba hash kód danej adresy, výsledok validácie a čas uloženia. V druhej tabuľke `domains` sú uložené známe domény s ich prioritami. Pre rýchlejšie vyhľadávanie už zvalidovaných adries je vytvorený index nad `email_address_hash` stĺpcom v tabuľke `email_validation`.

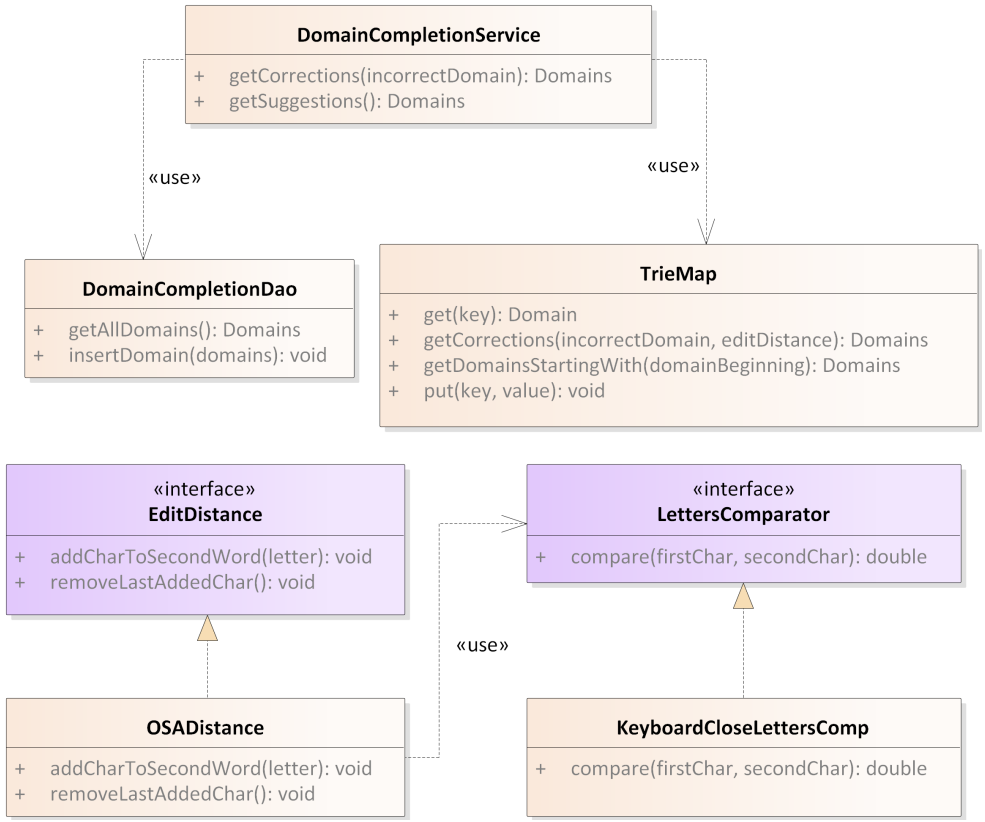
3.3.4 Diagram aktivít

Celý proces od našepkávania e-mailovej adresy až po jej validáciu je zaznamenaný na obr. 3.6.

Proces začína napísaním prvého znaku za znakom „@“, v tej chvíli systém užívateľovi navrhne možné domény. Užívateľ má možnosť pokračovať v písaní a systém mu bude po každom novom znaku ponúkať možné dokončenia alebo môže vybrať navrhovanú doménu. Ak užívateľ vybral navrhovanú doménu, záleží, na akej akcií má nastavené navrhovanie opráv a validáciu e-mailovej adresy, ale ak vykoná ľubovoľnú z niektorých jeho nadefinovaných akcií, navrhne sa užívateľovi oprava domény, resp. sa začne validácia adresy.

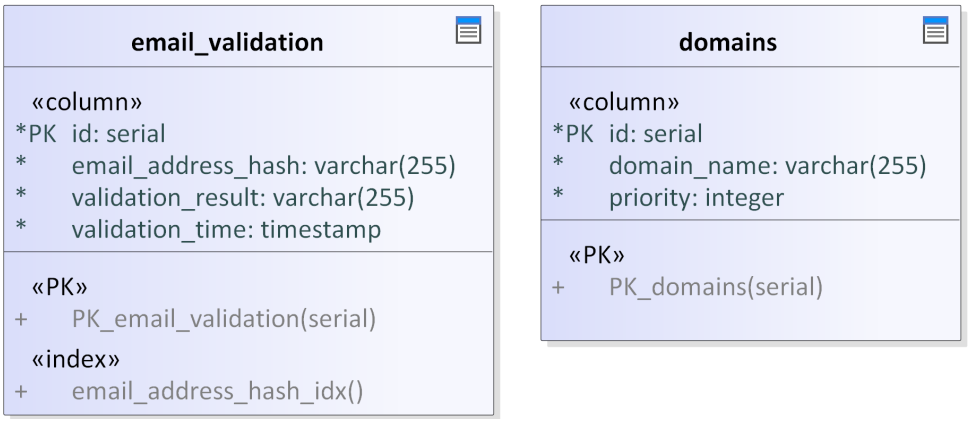
Proces validácie e-mailovej adresy

Diagram 3.7 zachytáva proces validácie e-mailovej adresy.



Obr. 3.4: Štruktúra tried zodpovedných za dopĺňanie a opravu doménovej časti e-mailovej adresy

Proces začína kontrolou, či sa e-mailová adresa na vstupe nachádza v cache pamäti a či je stále tento záznam aktuálny. Ak sú obe podmienky splnené, výsledok validácie sa vyberie z cache pamäte a vráti. Ak sa e-mailová adresa v cache nenachádza alebo už nie je naďalej valídna, tak sa ako prvá skontroluje syntaktická stránka adresy. Pokiaľ je syntax adresy v poriadku, pokračuje ďalší krok, a tým je získanie doménových mien poštových serverov. Pri nevalídnej syntaxi sa ďalšie validačné kroky preskočia a rovno sa vráti výsledok. Po dokončení DNS požiadavku sa výsledok uloží do cache pre výsledky DNS hľadaní. Ak boli nájdené nejaké doménové mená potencionálnych poštových serverov, pokračuje sa SMTP validáciou adresy. Ak žiadne doménové mená nájdené neboli, tak sa hneď vráti výsledok validácie. Po SMTP validácií dôjde k uloženiu výsledku validácie do cache a zalogovaniu doménovej časti e-mailovej adresy. Posledným krokom je vrátenie výsledku validácie.



Obr. 3.5: Databázový model

3.3.5 Návrh API

Komunikácia s webovou službou bude zabezpečená pomocou REST API. Táto sekcia popisuje návrh endpointov pre našepkávanie a opravu doménových častí e-mailových adries a tak isto aj pre validáciu e-mailových adries.

GET `/domain/getSuggestions/{typedInput}` Endpoint prijíma rozpísanú časť domény e-mailovej adresy a vracia možné domény, ktoré začínajú na parameter `typedInput`. Vrátené domény sú zoradené podľa priority, ktorou každá doména disponuje. Priorita vyjadruje koľko krát bola za určité časové obdobie validovaná e-mailová adresa s touto doménou.

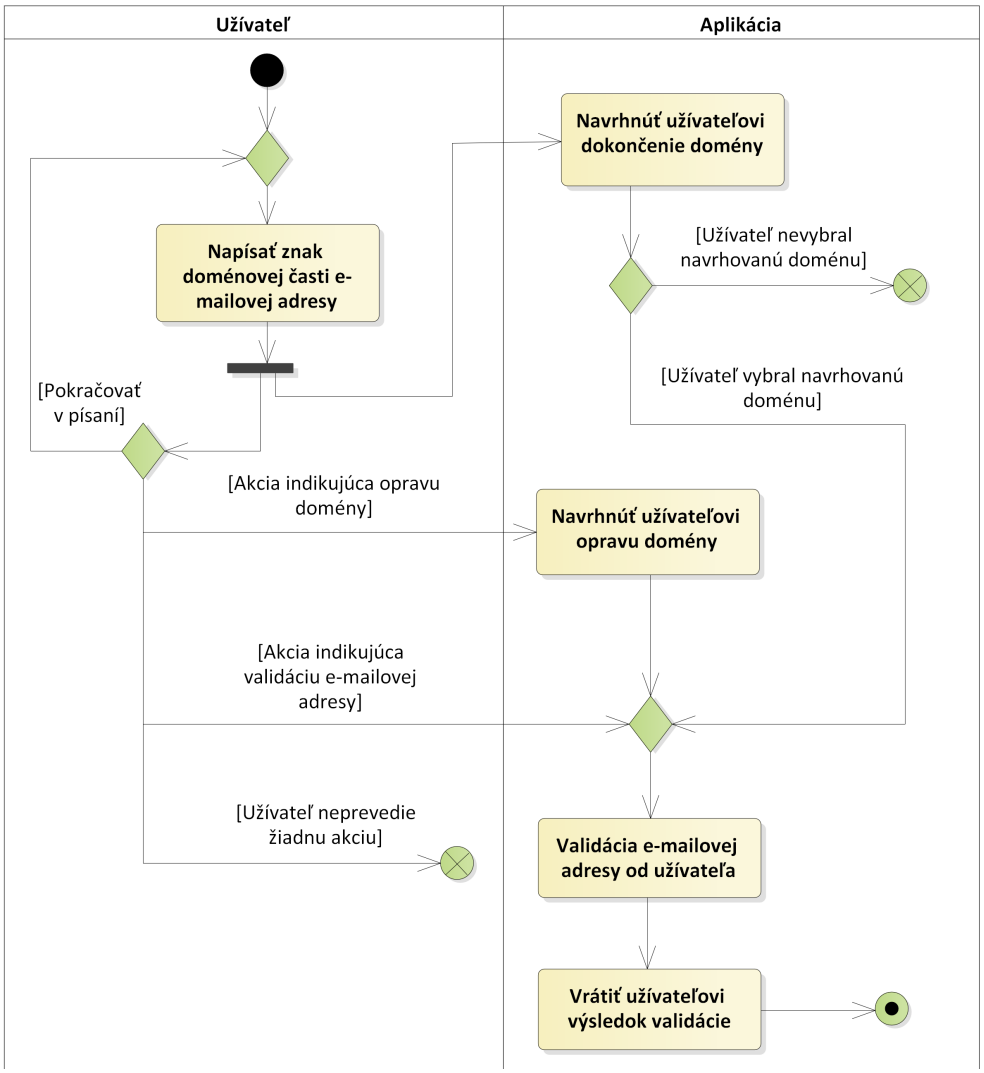
GET `domain/getCorrections/{typedInput}` Endpoint prijíma doménovú časť e-mailovej adresy s preklepom a vracia domény, ktoré majú editačnú vzdialenosť od `typedInput` menšiu ako určitá hranica. Domény vracia zoradené podľa priority a sekundárne podľa editačnej vzdialenosti.

POST `/validation/validateEmail` Endpoint, ktorého úlohou je validácia e-mailových adries. Vracia výsledok validácie.

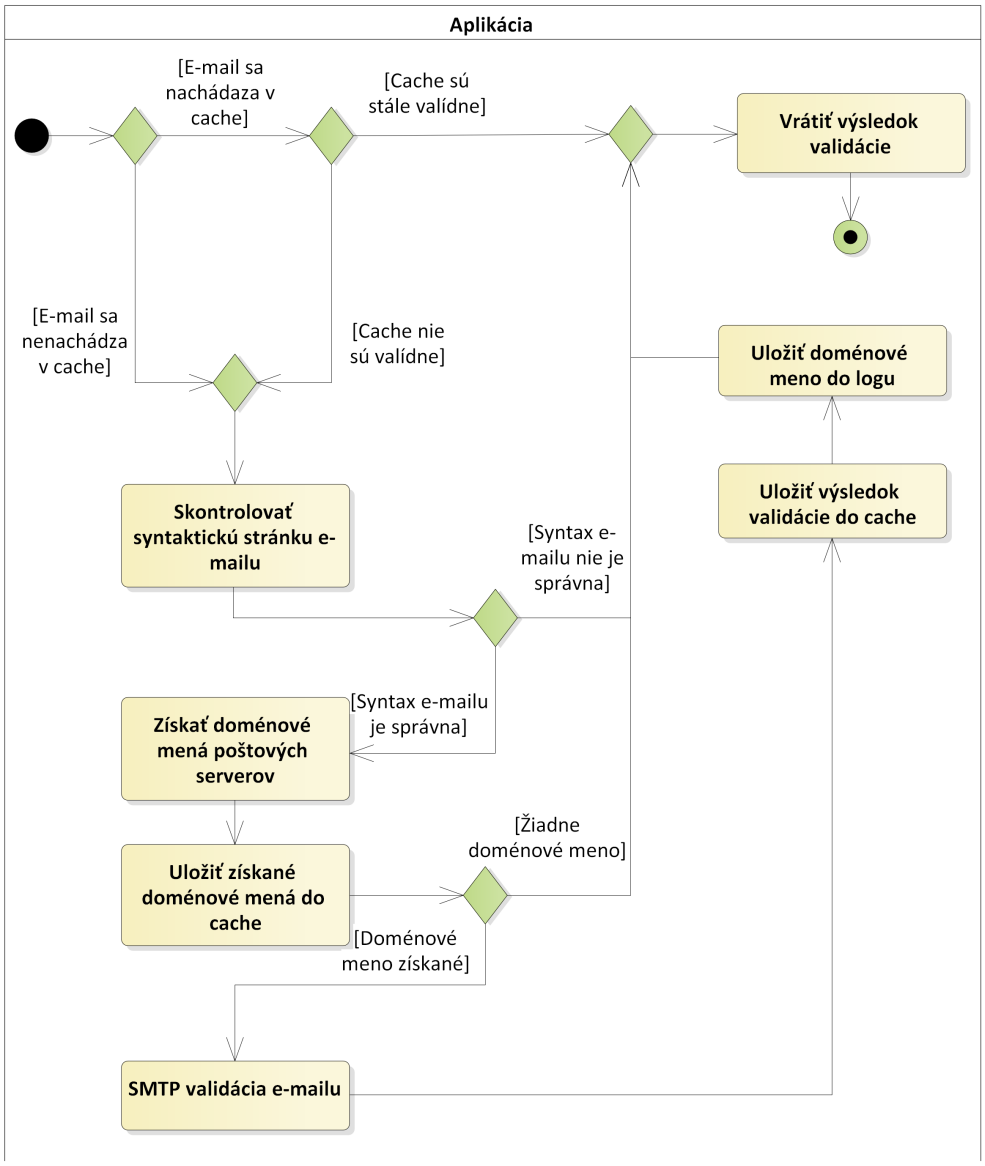
GET `/validation/getMxRecordsExistance/{domain}` Endpoint pre validáciu domén. Vracia true ak doména disponuje MX záznamom, inak vracia false.

3.3.6 Prepojenie klienta so serverom

Pre čo najjednoduchšiu integráciu uvedenej služby do webovej aplikácie, som vytvoril aj javascriptovú knižnicu. Tá má za úlohu prepojiť vstupné políčko



Obr. 3.6: Diagram aktivít celého procesu od užívateľského vstupu, až po validáciu e-mailovej adresy



Obr. 3.7: Diagram aktivít procesu validácie e-mailovej adresy

pre e-mailové adresy na webovej aplikácii klienta so službou a poskytovať dáta pre klienta, ktorý má nasledujúce možnosti.

1. Klient môže knižnici poskytnúť callback a nadefinovať, pre akú akciu sa má vykonať vrátenie návrhov, oprav domén, či výsledok validácie adresy. Pri tomto spôsobe sa potom po vykonaní danej akcie zavolá klientov callback s výsledkami zodpovedajúcimi vykonanej akcii.

2. Klient môže priamo volať metódy, ktoré komunikujú so službou a takýmto spôsobom získavať potrebné dáta.

3. Klient môže iba vytvoriť instanciu triedy, ktorú knižnica poskytuje a nechať našepkávanie, opravu aj validáciu adries na implicitných nastaveniach. Dodatočne si môže pomocou CSS upraviť našepkávanie.

3.3.7 Front-end

Na prezentáciu funkčnosti služby, som vytvoril jednoduchú front-endovú časť reprezentujúcu klienta. Front-endová časť obsahuje jednu obrazovku s formulárom prijímajúcim e-mailové adresy, v ktorom sa prezentujú jednotlivé funkčnosti služby.

Našepkáva sa iba prvá navrhnutá doména. Oprava domény sa navrhne iba v prípade, že výsledok validácie bude nevalídna doména. Výsledok validácie sa zobrazí ako existujúca adresa, iba ak validácia vráti, že adresa skutočne existuje. V opačných prípadoch sa výsledok bude prezentovať ako neexistujúca e-mailová adresa.

Je vytvorená pomocou javascriptového frameworku Vue.js. Pre získavanie dát zo služby využíva spomínanú javascriptovú knižnicu.

Kapitola 4

Implementácia

Obsahom nasledujúcej kapitoly je podrobnejší rozbor implementácie služby spolu s jej nasadením. Najskôr popisujem implementáciu jednotlivých častí validácie e-mailových adries. V druhej časti sa venujem implementácií trie dátovej štruktúry, spolu s editačnou vzdialenosťou a spôsobu ich spolupráce. Posledná časť je zameraná na nasadenie výslednej služby.

4.1 Implementácia validácie e-mailových adries

Validácia e-mailovej adresy pozostáva z troch krokov. Validuje sa syntax adresy, následne existencia domény, pričom musí existovať aspoň jeden poštový server pod touto overovanou doménou. Nakoniec sa zisťuje existencia konkrétnej poštovej schránky na získanom poštovom serveri.

4.1.1 Validácia syntaxe

Písať vlastný syntax validátor e-mailových adries tak, aby fungoval správne a boli korektne ošetrené aj všetky hraničné prípady, môže byť veľmi prácne. Preto som sa rozhodol využiť existujúce riešenie od *Apache Commons* [45], ktoré poskytuje, okrem iných aj validátor syntaxe e-mailových adries.

4.1.2 Validácia existencie domény

Druhým krokom validácie e-mailovej adresy je zistenie existencie domény a získanie MX záznamov, ktoré slúžia SMTP validácií ako server, na ktorom sa overuje existencia poštovej schránky.

Aktuálne služba vyžaduje prítomnosť MX záznamu a to najmä kvôli tomu, že poštové servery, ktoré nedisponujú MX záznamom a majú len A záznam, sú zriedkavé a tým, že vylúčim e-mailové adresy bez MX záznamov, vylúčim aj množstvo nevalídnych e-mailových adries, ktoré by sa inak validovali. Častá SMTP validácia nevalídnych e-mailových adries by mala za následok zníženie

reputácie IP adresy, až zápis IP adresy na blacklist, akým disponuje napríklad Spamhaus.

Pre prácu s DNS v jave sa ponúka JNDI. Bližšia charakteristika používania JNDI v spojení s DNS sa uvádza v oficiálnej dokumentácií [46]. Ďalšia možnosť je využiť externú knižnicu. Najpopulárnejšia voľba je *dnsjava* [47]. V nasledujúcej kapitole som popísal testovanie oboch prístupov. Keďže výsledky testov boli pre oba spôsoby takmer totožné, rozhodol som sa pre JNDI spôsob a vyhol sa tým zbytočne ďalšej závislosti na externej knižnici v kóde.

4.1.3 SMTP Validácia

SMTP validácia funguje na princípe simulovania odosielania e-mailu na práve validujúcu sa adresu. Komunikácia s poštovým serverom je implementovaná pomocou javovských Socketov. Trieda `SmtpValidation` disponuje hlavnou metódou pre validáciu, ktorú je možné vidieť na obr. 4.1. Každá validácia začína pripojením sa k poštovému serveru, ďalej sa postupne posielajú na server jednotlivé príkazy, v metóde `checkUserExistance` a po každom sa kontroluje odpoveď od servera. Vyhodnotenie odpovedí rieši trieda `ResultValidation`. Tú reprezentuje na výpise 4.1 `validator` premenná. Kolekcia, prezentovaná premennou `checkUserCommands` je pri vytváraní `SmtpValidation` triedy inicializovaná *Command* triedami: `HeloCommand, MailCommand, RcptCommand`, ktorým sa postupne predáva tzv. *visitor* trieda `SmtpService`. Tá implementuje jednotlivú logiku pre každý príkaz. V tejto fáze, sa podľa odpovede, primárne na príkaz `RCPT`, rozhodne o existencii e-mailovej adresy. Najčastejšie odpovede na príkaz `RCPT` sú:

- Kód 250, znamená, že užívateľ s veľkou pravdepodobnosťou existuje na poštovom serveri.
- Kód 550, pri tomto kóde sa rozlišujú ešte dva najčastejšie používané rozšírené kódy a to 5.1.1, ktorý znamená, že užívateľ neexistuje na poštovom serveri a 5.7.1, ktorý znamená, že nastala situácia, kvôli ktorej server informáciu o užívateľovi nezdelí. Toto môže nastať, ak blacklist služby Spamhause obsahuje odosielateľovu IP adresu.

4.1.4 Cache

Aby sa počet validácií na poštových serveroch čo najviac minimalizoval, zaviedol som caching e-mailových adries. Podstatné je, že sa neukladá e-mailová adresa, ale ukladá sa len hash kód adresy spolu s výsledkom validácie a časom uloženia. Konkrétne je použitá hashovacia funkcia *sha256*. Funkcionalitu cachovania zabezpečuje proxy trieda `EmailValidationProxy`, ako je možné vidieť na obr. 3.2. E-mailová adresa sa najskôr dohľadá v cache pamäti a ak sa nájde a je aktuálna, tak sa rovno vráti výsledok jej validácie. Ak by sa

Výpis kódu 4.1 Hlavné metódy SMTP validácie

```
public EmailValidationResult
       validate( EmailAddress receiverEA, MXRecord rec ) {
    EmailAddress senEm =
                 new EmailAddress( "uss", senderDomain, senderEA );
    SmtpService smtpService =
                new SmtpService( parser, senEm, receiverEA, rec );

    EmailValidationResult result = EmailValidationResult.UNKNOWN;

    //connecting to the mail server
    smtpService.connect();

    //validating response from mail server
    if ( !validator.validateResponse( smtpService.getResult() ) )
        return result;

    //sending commands to mail server
    result = checkUserExistance( smtpService, validator );

    if ( !endCommunication( smtpService, validator ) )
        log.warn( "Communication does not end properly");

    smtpService.closeConnection();
    return result;
}

private EmailValidationResult
        checkUserExistance(SmtpService s, ResultValidation rv){

    for(SmtpCommand command : checkUserCommands) {
        command.perform(s);
        SmtpCommandResult res = s.getResult()
        if(!resultValidation.validateResponse(res))
            return resultValidation.determineResult(res);
    }
    // in case everything is okay user exists on server
    return EmailValidationResult.SERVER_HAS_USER;
}
```

nenašla alebo už nie je naďalej aktuálna, prebehne validácia a hash kód spolu s výsledkom sa uloží do cache. Dáta sú ukladané do databázy. A samotné ukladanie prebieha asynchrónne.

```
CompletableFuture.supplyAsync( () -> cache.storeToCache(hash, value));
```

Pod caching spadajú aj doménové mená poštových serverov, ale už nie na úrovni databázy, ale len za behu programu. To zaručuje zrýchlenie validácie.

4.1.5 Obmedzenie vlákien

Počas testovania som zistil, že poštový server môže dočasne zablokovať IP adresu, z ktorej prebieha SMTP validácia, ak prebieha súčasne vo viacerých vláknach. Preto som pomocou triedy `ThreadPoolTaskExecutor` obmedzil maximálny počet vlákien, ktoré môžu vykonávať v jednej chvíli SMTP validáciu na dve.

4.2 Implementácia dopĺňania a opravy domén

Dopĺňanie domén

Jadrom našepkávania domén je trie dátová štruktúra. Trie implementáciu som mierne pozmenil. Každý vrchol obsahuje namiesto poľa ukazateľov na synov, hash mapu `children`, kde kľúčom je znak a hodnotou je ďalší vrchol s totožným znakom. Funkcie slúžiace na navrhovanie možných domén na základe rozpísanej časti domény, je možné vidieť na výpise 4.2.

Výpis kódu 4.2 Metódy pre návrh doménovej časti

```
public TrieNode getNodeForLastChar( String prefix ) {
    if ( prefix.isEmpty() )
        return this;

    char processedChar = prefix.charAt( 0 );
    if ( children.containsKey( processedChar ) ) {
        TrieNode child = children.get( processedChar );
        return child.getNodeForLastChar( prefix.substring( 1 ) );
    }
    else
        return null;
}

public void getDomains( List<Domain> outputWords, int alreadyIn ) {
    if ( createsWord )
        outputWords.add( domain );

    if ( ( outputWords.size() + alreadyIn ) >= SEARCH_LIMIT )
        return;

    for ( TrieNode child : children.values() )
        child.getDomains( outputWords, alreadyIn );
}
```

Samotný výber vhodných domén funguje nasledovne. V trie sa nájde vrchol zodpovedajúci poslednému znaku rozpísanej domény a výsledkom sú všetky možné slová, ktoré existujú v trie od tohoto vrcholu hlbšie. V ukážkovej im-

plementácii výpisu 4.2, premenná `createsWord` je typu boolean a označuje vrcholy, ktoré tvoria slovo. Premenná `alreadyIn` označuje množstvo nájdených domén.

Oprava domén

Navrhnutie opravy domény prebieha v spolupráci trie dátovej štruktúry spolu s editačnou vzdialenosťou. Konkrétna vzdialenosť, ktorú som implementoval, nesie názov *Optimal String Alignment Distance*. Od Levensteinovej vzdialenosti sa odlišuje tým, že rozširuje možné editačné operácie o vzájomnú výmenu dvoch po sebe idúcich znakov. Na vstupe algoritmus dostane doménu, ku ktorej bude hľadať také domény, ktoré od nej nemajú editačnú vzdialenosť väčšiu, ako je určené hranicou. Samotné hľadanie je možné vidieť na výpise 4.3. Metóda `getCorrections` berie ako parametre výstupné pole a instanciu `EditDistance`. Táto instancia je už inicializovaná doménou, ku ktorej sa hľadajú domény s určitou editačnou vzdialenosťou a slúži na postupnú tvorbu tabuľky s výsledkami vzdialeností. Tabuľka sa tvorí postupne, vždy nový riadok pre novo pridaný znak a vďaka tomu, že sa slová prechádzajú systematicky, tak ako sú uložené v trie, tak pre každý nový znak sa nový riadok tabuľky vypočíta práve jeden krát.

Výpis kódu 4.3 Metóda pre získanie domén s určitou edit. vzdialenosťou k zadanej doméne

```
public void
    getCorrections( List<DomainAndDistance> out, EditDistance dist ) {

    if ( out.size() >= SEARCH_LIMIT )
        return;

    double cmpDist = dist.getDistance();
    if ( cmpDist <= dist.getMaxDistance() ) {
        if ( createsWord )
            out.add( new DomainAndDistance( domain, cmpDist ) );
    }
    if ( !dist.existsMinimum() )
        return;
    for ( Map.Entry<Character, TrieNode> entry : children.entrySet()){
        dist.addCharToSecondWord( entry.getKey() );
        entry.getValue().getCorrections( out, dist );
        dist.removeLastAddedChar();
    }
}
```

4.3 Implementácia javascriptovej knižnice

Javascriptová knižnica slúži na prepojenie klienta s API služby. Vstupné políčko, ktoré sa použije na validáciu e-mailových adries a našepkávanie, musí obsahovať určitý class atribút. Pri integrácií si klient túto knižnicu stiahne a vytvorí instanciu hlavnej triedy, pomocou statických metód.

Prvou metódou je `createDefaultEmailApi(evenListenerConfig)`, ktorá ponúka automatické našepkávanie a validáciu e-mailových adries, podľa zadanej konfigurácie.

Druhou metódou je `createEmailApi(callback, evenListenerConfig)`, ktorej užívateľ predá ako parametre callback a konfiguráciu jednotlivých akcií, pri ktorých sa má vykonať dopĺňanie, oprava domén a validácia e-mailových adries. Pri jednotlivých akciách je potom volaný klientov callback, ktorému sú odovzdané dáta zo služby. Bližší popis je možné vidieť v dokumentácii, ktorá sa nachádza v prílohe práce. Samotná knižnica využíva pri komunikácii so službou fetch API.

4.4 Nasadenie

Pre nasadenie a spustenie služby som využil nástroje Docker a Jenkins. Služba, ktorá je nasadená na serveri, je rozdelená na niekoľko kontajnerov.

- postgreSQL databáza s menom `email-validation-db`
- služba vysavujúca API, pomenovaná ako `email-validation-app`
- front-end pomenovaný ako `email-validation-app-frontend`

Kontajner databázy a služby je popísaný v `docker-compose.yml` súbore, uloženom na serveri v `/home/ansible/docker/email_validation`. Tento súbor je možné vidieť vo výpise 4.4. Pre front-endovú časť je použitý kontajner, obsahujúci nginx server. Po nasadení zostáva už len zariadiť presmerovanie doménových mien na jednotlivé kontajnery. To sa odohrá v už existujúcom kontajneri, ktorý tak isto obsahuje nginx server. Ten slúži primárne len ako proxy server.

Pre samotné nasadenie som vytvoril niekoľko jobov na Jenkinse. Prvý job si stiahne adresár projektu z verzovacieho systému, prevedie zostavenie aplikácie a uloží výsledný jar súbor. Ďalší job slúžiaci pre nasadenie služby, si skopíruje najaktuálnejší jar z predchádzajúceho jobu, pripojí sa na server, na ktorý chce službu nasadiť a prevedie nasledujúce príkazy.

```
mv -f EmailValidation.jar /opt/email_validation/data/bin/
cd /home/ansible/docker/email_validation/
docker restart email-validation-app
```

Výpis kódu 4.4 Definovanie kontajnerov pre databázu a e-mail validáciu

```
version: '3.2'
services:
  email-validation-db:
    image: postgres:9-alpine
    container_name: email-validation-db
    restart: always
    environment:
      TZ: Europe/Prague
      LANG: cs_CZ.UTF-8
      LC_ALL: cs_CZ.UTF-8
      POSTGRES_DB: postgres
      POSTGRES_PASSWORD: ${PG_PASSWORD}
    volumes:
      - /opt/email_validation/db:/var/lib/postgresql/data
    ports:
      - "192.168.42.120:5444:5432"
    networks:
      - network
      - frontend

  email-validation-app:
    image: openjdk:12-jdk-alpine
    container_name: email-validation-app
    restart: always
    command: java -jar bin/EV.jar --spring.profiles.active=production
    working_dir: /opt/email_validation/data
    environment:
      TZ: Europe/Prague
      LANG: cs_CZ.UTF-8
      LC_ALL: cs_CZ.UTF-8
      PG_USERNAME: postgres
      PG_PASSWORD: ${PG_PASSWORD}
    volumes:
       - /opt/email_validation/data:/opt/email_validation/data
    depends_on:
      - email-validation-db
    ports:
      - "192.168.42.120:8048:8080"
    networks:
      - network
      - frontend

networks:
  network:
      driver: bridge
      driver_opts:
          com.docker.network.enable_ipv6: "false"
  frontend:
      external: true
```

Job skopíruje jar súbor na server a reštartuje už bežiaci kontajner. Týmto sa nasadí nová verzia služby. Pre nasadenie prvej verzie som vytvoril separátny job, v ktorom by posledný príkaz používal docker-compose nástroj. Spustenie služby s databázou by teda vyzeralo nasledovne `docker-compose up`.

Kapitola 5

Testovanie

V úvode kapitoly predstavím spôsob testovania samotnej služby a v ďalšej časti popíšem testovanie smtp validácie, priblížim problémy, s ktorými som sa potýkal a predstavím dosiahnuté výsledky.

5.1 Unit testy

Služba pri vývoji bola primárne testovaná unit testami. Unit testy izolovane testujú jednotlivé časti. Cieľom unit testov je, čo najrýchlejšie odhaliť možné chyby v kóde, pretože tieto testy majú nízke nároky na spustenie a môžu byť spúšťané veľmi často. Počas vývoja služby bola aplikácia pravidelne zostavovaná na jenkinse. Spolu so zostavením boli spúšťané aj unit testy. Pri neúspešnom zostavení alebo padajúcom teste bol odoslaný e-mail s upozornením.

5.2 Vyhľadávanie MX záznamov

Pri implementácii časti, ktorej úlohou bolo nájsť možné MX záznamy, som sa rozhodoval medzi použitím *JNDI* a externou knižnicou *dnsjava*. Oba spôsoby som implementoval a rozhodol sa ich porovnať oproti linuxovému príkazu `dig`, ktorý tak isto dokáže nájsť MX záznamy pre danú doménu. Dáta, teda domény použité na testovanie, som získal z [48]. Testovacie dáta som si pripravil pomocou bash skriptu, ktorý postupne iteruje všetkými doménami, na ktorých volá príkaz `dig`. Z výstupu tohoto príkazu odfiltruje všetko okrem výsledkov MX záznamov a to je priorita a doménové meno poštového servera. Tento výsledok uloží do súboru v tvare `skúmaná doména, priorita, doména poštového servera, ...`

Testovanie prebiehalo tak, že sa ako prvé uložili spracované výsledky z `dig` príkazu do mapy `validationMap`, kde kľúčom bola skúmaná doména a hodnotou množina MX záznamov. Obe metódy začali iterovať kľúče mapy a po-

rovnávali sa hodnoty získané týmito metódami s hodnotami v mape. Ukážku kódu je možné vidieť vo výpise 5.1. Kód je napísaný paralelne, aby mohli byť spustené hneď obe metódy a znížil sa čas testovania. Pri chybnom porovnaní sa doména pomocou `writer` instancie zapíše do súboru, pre ďalšie možné skúmanie.

Výpis kódu 5.1 Testovanie získavania MX záznamov

```
for ( String domain : validationMap.keySet() ) {
    DNSResult result = mxValidation.lookupMXRecords( domain, 999 );

    if ( !compare( result.mxRecords, validationMap.get( domain ))){
        log.info( "Domain: {} does not match", domain );
        try {
            writer.write( domain + "\n" );
        }
        catch ( IOException e ) {
            throw new RuntimeException( "Exception occured " );
        }
    }
}
```

Výsledky sú porovnateľné, pri 47433 doménach bolo pri validácii pomocou knižnice *dnsjava* 73 nesprávne a pomocou *JNDI* 75 nesprávne. Rozhodol som sa pre *JNDI* spôsob, najmä z dôvodu, že obe metódy sú porovnateľné, ale vybraným spôsobom nezanesiem ďalšiu zbytočnú závislosť do kódu.

5.3 Validácia e-mailových adries

Prvotne som mal zámer testovať SMTP validáciu e-mailových adries na AWS serveri. Toto rozhodnutie vzniklo z vyššie spomínaného dôvodu v časti 2.1.4, a to, že najmä pri testovaní by poštové servery mohli zabanovať IP adresu, z ktorej sa validácia vykonávala. Testovanie na AWS serveri nakoniec nebolo možné, pretože AWS implicitne blokuje odchádzajúcu komunikáciu na port 25, aby zabránil v rozposielaní spamu.

Blokovaná odchádzajúca komunikácia na port 25 bola aj v mojej domácej sieti. Blokovanie je bežné u väčšiny internetových poskytovateľov. Je to kvôli tomu, aby sa zamedzilo prípadom, keď by bol klientsky počítač v sieti napadnutý vírusom, ktorý by bol vytvorený za účelom rozosielania spamu. Po kontaktovaní internetového poskytovateľa sa podarilo zariadiť odblokovanie komunikácie na port 25.

Testovanie validácie prebiehalo z viacerých IP adries. Bola to IP adresa mojej domácej siete, ďalej som testoval z IP adresy mobilného operátora a na záver testovanie prebiehalo aj na serveri so statickou IP adresou, doménou a PTR záznamom v DNS.

Hlavnú metódu spúšťajúcu testovanie, je možné vidieť vo výpise 5.2. Ako argumenty sa berú počet vlákien, meno súboru s e-mailovými adresami a meno výstupného súboru s výsledkami. Metóda uloží e-mailové adresy zo súboru do kontajnera `ConcurrentLinkedQueue`. Ide o *thread-safe* kontajner pre prácu s viacerými vláknami. Ďalej sa vytvorí `ExecutorService` a inicializujú sa `Callable` objekty, ktoré sa spustia metódou `invokeAll`. Pri spúšťaní týchto objektov ako samostatné vlákna sa volá metóda `call`, ktorá postupne vyberá e-mailové adresy z kontajnera a spúšťa na nich validáciu.

Testovacie dáta

Testovacie dáta mi poskytol vedúci práce. Išlo o 36000 e-mailových adries. Z nich 21000 tvorili e-mailové adresy s doménou seznam.cz alebo s doménou, ktorá využíva poštové servery domény seznam.cz. Druhý najvyšší počet 7000 tvorili e-mailové adresy s doménou gmail.com alebo s doménou využívajúcou poštové servery domény gmail.com. Na základe dát a populárnosti uvedených poštových klientov usudzujem, že väčšina užívateľov bude využívať poštových klientov od seznam.cz alebo gmail.com. Preto som tieto dve domény testoval aj samostatne.

Testovacie dáta som využil aj na inicializáciu domén, ktoré sa následne používajú pre dopĺňanie domén, či návrh opravy. Je to dočasné riešenie a tieto domény s ich prioritami budú nahradené neskôr produkčnými dátami z logovania domén, zadaných užívateľmi.

5.4 Testovanie

Testovanie validácie e-mailových adries som začal na IP adrese mobilného operátora. Ta má *Policy Blocklist* záznam v službe Spamhause. Poskytovatelia internetu majú istý rozsah IP adries, napr. pre zákazníkov s telefonickým pripojením implicitne na blackliste služby Spamhaus, aby sa vyvarovali nežiadúceho správania zo strany užívateľov. Testovanie sa realizovalo niekoľko krát a s rôznym počtom vlákien od 1 až po 6. Validovalo sa všetkých 36000 e-mailových adries. Pri testovaní validácie z IP adresy mobilného operátora sa vyskytli nasledujúce problémy:

- odmietanie komunikácie zo strany poštového servera kompletne alebo server odpovedal s chybou na príkaz `RCPT`, s odôvodnením, že IP adresa odosielateľa má záznam v službe Spamhaus
- odmietanie komunikácie zo strany poštového servera, kvôli dynamickej IP adrese
- odmietanie komunikácie zo strany poštového servera, kvôli nedostatočnej autentizácií

- dočasná neschopnosť pripojiť sa na poštový server
- odpoveď s dočasnou chybou na `RCPT` príkaz, z dôvodu greylistingu
- odpoveď s chybou na `RCPT` príkaz, z dôvodu anti-spamového filtra
- odpoveď s chybou na príkaz `MAIL`, z dôvodu žiadneho MX záznamu pre doménu z parametra
- odpoveď s chybou na príkaz `MAIL`, z dôvodu nemožnosti pripojenia sa na poštový server, ktorý by sa mal skrývať pod doménou e-mailovej adresy z parametra príkazu `MAIL`

Pri testovaní validácie z IP adresy mojej domácej siete zmizol problém s komunikáciou kvôli službe Spamhaus, nakoľko tam IP adresa nemala žiadny záznam. Ostatné problémy stále pretrvávali.

Testovanie zo statickej IP adresy s doménovým menom a PTR záznamom už nebolo sprevádzané vyššie uvedenými problémami, až na problém s nemožnosťou pripojiť sa na poštový server a absenciou MX záznamu. Tento problém bol však veľmi minoritný a vyskytol sa len pri jednej e-mailovej adrese. Poštové servery zväčša nerobia takúto hĺbkovú kontrolu. Ďalší pretrvávajúci problém je greylisting a prípadná nedostatočná autentizácia.

Porovnanie výsledkov testov e-mailových adries s unikátnymi doménami pre jednotlivé IP adresy je možné vidieť v tabuľke 5.1. Validácie s chybou označujú také validácie, pri ktorých nastal niektorý z vyššie spomínaných problémov.

IP adresa	**Celkový počet validácií**	**Počet validácií s chybou**
Mobilná IP	2074	835
Domáca IP	2074	431
Statická IP	2074	214

Tabuľka 5.1: Výsledky testovania validácie e-mailových adries s unikátnymi doménami

5.5 Gmail a Seznam

Servery gmail.com a seznam.cz sú prístupnejšie, čo sa týka striktnosti prijímania e-mailov a komunikovali bez problémov so službou zo všetkých troch testovaných IP adries. Pri testovaní týchto dvoch poštových serverov som sa zameral na otestovanie nasledujúcich aspektov.

- Maximálny počet vlákien, ktorými je možné validovať e-mailové adresy na danom poštovom serveri z rovnakej IP adresy. Výsledky testu je možné vidieť v tabuľke 5.2.

- Doba počas ktorej je možné neustále, bez prestávky validovať e-mailové adresy na danom poštovom serveri. Výsledky testu je možné vidieť v tabuľke 5.3.

- Pomer existujúcich a neexistujúcich e-mailových adries, ktoré je možné validovať bez zabanovania zo strany servera. Výsledky testu je možné vidieť v tabuľke 5.4.

V nasledujúcich tabuľkách výsledok testu *úspech* označuje štandardnú komunikáciu so serverom počas celého testovania. Tým je myslené, že odpovede na jednotlivé príkazy neobsahujú žiadnu chybu a server poskytoval legitímne odpovede. Výsledok *neúspech* znamená, že server odpovedal na nejaký príkaz neštandardne s chybou. To znamená napríklad, že IP adresu, z ktorej prebieha validácia dočasne zabanoval alebo sa niekoľko krát nepodarilo na server pripojiť. Doba testovania je vyjadrená v hodinách.

Testovaná doména	**Doba testovania (h)**	**Počet vlákien**	**Výsledok**
gmail.com	2	1	úspech
	2	2	úspech
	2	3	úspech
	2	4	úspech
seznam.cz	2	1	úspech
	2	2	úspech
	0,1	3	neúspech

Tabuľka 5.2: Výsledky testovania maximálneho počtu vlákien, pri ktorom sa neustále validujú e-mailové adresy a server štandardne odpovedá

Testovaná doména	**Doba testovania (h)**	**Počet vlákien**	**Výsledok**
gmail.com	48	1	úspech
	24	2	úspech
seznam.cz	48	1	úspech
	24	2	úspech

Tabuľka 5.3: Výsledky testovania dĺžky doby, počas ktorej sa neustále validujú e-mailové adresy a server štandardne odpovedá

Testovaná doména	Doba testovania (h)	Pomer neexistujúcich adries	Výsledok
gmail.com	2	0,25	úspech
	2	0,5	úspech
	2	0,75	úspech
	2	1	úspech
seznam.cz	2	0,25	úspech
	0,15	0,5	neúspech

Tabuľka 5.4: Výsledky testovania pomeru neexistujúcich e-mailových adries, pri ktorom sa neustále validujú e-mailové adresy a server štandardne odpovedá

5.6 Doba odpovede

Pri testovaní som zaznamenával aj dobu odpovede poštových serverov na jednotlivé príkazy.

Zo získaných dát som pozoroval, či so stúpajúcim počtom zvalidovaných e-mailových adries nestúpa aj doba odpovede od poštových serverov. To by mohlo slúžiť ako obranný mechanizmus voči neoprávnenému získavaniu e-mailových adries. Toto pozorovanie sa nepotvrdilo a doba odpovede od serverov aj napriek stúpajúcemu počtu zvalidovaných e-mailových adries bola približne rovnaká.

Druhé pozorovanie, na ktoré som sa zameral, malo za cieľ zistiť, či sa líši doba odpovede od poštových serverov počas dňa a noci a to najmä z dôvodu, že služba by mohla využívať napríklad rýchlejšiu nočnú validáciu vo svoj prospech. Toto pozorovanie sa taktiež nepotvrdilo a doba odpovede od serverov bola počas dňa aj počas nočných hodín približne rovnaká

5.7 Výsledky

Výsledky testovania vnímam ako pozitívne a to najmä preto, že služba je funkčná pre dva najpoužívanejšie poštové servery v Čechách a na Slovensku, kde sa bude primárne využívať.

Dôležitým zistením bolo, že služba dokáže aj počas dlhšej časovej doby, napríklad dva dni, neustále validovať e-mailové adresy.

Pri doméne seznam.cz je síce limitácia maximálne dvoch vlákien a 50% pomeru neexistujúcich emailov, ale na účely, na aké sa bude služba využívať by tento pomer nemal vystúpiť nad 25%. Ďalej testovanie ukázalo, že na produkčnej, statickej IP adrese je služba funkčná na približne 1860 poštových serverov z 2074. Služba preto bude mať praktické využitie.

Výpis kódu 5.2 Testovanie SMTP validácie

```
public void
getStatistics( int threadCount, String emailsFile, String outputCsvFile ) {
    try {
        List<String> emailAddresses = loadEmailAddresses( emailsFile );
        Queue<String> allEmailAddresses;
        allEmailAddresses = new ConcurrentLinkedQueue<>( emailAddresses );
        AtomicInteger count = new AtomicInteger( 0 );

        CSVWriter csvWriter = new CSVWriter( new FileWriter( outputCsvFile ) );
        writeHead( csvWriter );

        ExecutorService service = Executors.newFixedThreadPool( threadCount );
        List<EmailValidationCallable> threads = new ArrayList<>();
        for ( int i = 0; i < threadCount; i++ )
            threads.add( new EmailValidationCallable( emailValidationService,
            csvWriter, allEmailAddresses, count, allEmailAddresses.size() ) );

        List<Future<EmailValidationStats>> validationResult;
        validationResult = service.invokeAll( threads );

        EmailValidationStats mergedStats = new EmailValidationStats( 0 );
        for ( Future<EmailValidationStats> res : validationResult ) {
            EmailValidationStats actualRes = res.get();
            mergedStats.addAll( actualRes.getMap() );
        }

        csvWriter.close();
        log.info( mergedStats.getStatistics() );
    }
    catch ( IOException | InterruptedException | ExecutionException e ) {
        log.error( "Exception was thrown {}", e.toString() );
        throw new RuntimeException( "Error while validating emails", e );
    }
}
```

Kapitola 6

Možné rozšírenia

Rozloženie záťaže

Testovanie 5 na poštových serveroch domén gmail.com a seznam.cz v predchádzajúcej kapitole ukázalo, že zatiaľ čo gmail zvládne validáciu aj zo štyroch vlákien v rámci jednej IP adresy, tak seznam ich zvládne maximálne dve. Momentálne je služba obmedzená na dve bežiace vlákna zároveň.

Rozšírenie môže pre individuálne domény nastaviť počet bežiacich vlákien. Napríklad pre gmail môžu bežať súčasne maximálne štyri vlákna, pre seznam dve vlákna a pre ostatné domény len jedno vlákno. Rozširujúce riešenie musí brať ohľad na maximálny celkový počet vlákien, ktoré môžu bežať súčasne a tieto vlákna rozdistribuovať medzi jednotlivé domény, podľa toho, koľko vlákien majú jednotlivé domény maximálne povolené. Tým sa docieli zrýchlenie služby.

IP pool

Ďalšie rozšírenie môže riešiť striedanie IP adries. Tým sa zvýši možný počet validácií a zníži sa šanca na chybné odpovede zo strany serveru.

Čiastočné riešenie greylistingu

Pokiaľ poštový server využíva metódu greylisting, tak pri opätovnej validácií po istom čase už poštový server štandardne odpovie. Pri odpovedi s kódom pre greylisting sa daná e-mailová adresa dočasne uloží a prevaliduje sa o nejaký čas. Výsledok druhej validácie sa uloží do cache pre budúce použitie.

Detekcia catch-all serverov

Detekcia catch-all serverov môže fungovať tak, že sa vygenerujú aspoň dva náhodné reťazce, ktoré predstavujú lokálnu časť e-mailovej adresy. Na testovaný poštový server sa potom pokúsia zvalidovať tieto náhodne vygenerované

e-mailové adresy. Ak je výsledkom existencia oboch týchto adries, tak server je s najvyššou pravdepodobnosťou catch-all. Pri tomto rozšírení sa musí ale zobrať do úvahy zvýšený počet validácií neexistujúcich e-mailových adries.

Detekcia dočasných e-mailových adries

Služba aktuálne nie je schopná detekovať dočasné e-mailové adresy. Riešením môže byť scraping niektorých vybraných stránok, kde sa objavujú domény týchto dočasných e-mailových adries a udržiavanie databázy týchto domén.

Záver

Cieľom práce bolo vytvoriť službu, ktorá by užívateľom pomáhala vo webových formulároch doplniť e-mailové adresy a po vyplnení overiť ich existenciu. Prínosom uvedenej služby je zabezpečenie, že vložené e-mailové adresy skutočne existujú. Z toho by benefitovali najmä majitelia webových aplikácií.

Pre vytvorenie požadovanej služby som ako prvé spracoval analýzu dostupných riešení a zvolil jedno vhodné riešenie, ktoré som následne implementoval. Zvolené riešenie využíva SMTP protokol, preto ďalším krokom bolo naštudovanie tohoto protokolu. Okrem analýzy validácie e-mailových adries som analyzoval aj možné riešenia dopĺňania a opráv doménových častí. Na základe nových poznatkov som vytvoril návrh služby, ktorý sa stal podkladom pre implementáciu.

Po implementačnej časti nasledovalo niekoľko fáz testovania. Pri testovaní sa objavili očakávané, ale aj neočakávané problémy ako napríklad nedostačujúca autentizácia, greylisting, či dočasné blokovanie komunikácie. Väčšinu problémov sa podarilo vyriešiť, ale istá časť stále pretrváva. Ide napríklad o greylisting a niektoré ďalšie, ktoré vzhľadom na povahu veci, nie je jednoduché vyriešiť.

Posledným krokom bolo nasadenie služby na server. Záverom môžem konštatovať, že služba je vo väčšine prípadov, čím myslím populárne poštové servery, funkčná a bude mať praktické využitie.

Bibliografia

1. P. RESNICK, Ed. *Internet Message Format* [online]. RFC Editor, 2008-10 [cit. 2021-03-01]. RFC, 5322. RFC Editor. ISSN 2070-1721. Dostupné z: `https://tools.ietf.org/html/rfc5322`.
2. WOOD, David. *Programming Internet Email mastering Internet messaging systems*. 101 Morris Street, Sebastopol: O'Reilly, 1999. ISBN 1-56592-479-7.
3. KLENSIN, J. *Simple Mail Transfer Protocol* [online]. RFC Editor, 2008-10 [cit. 2021-03-01]. RFC, 5321. RFC Editor. ISSN 2070-1721. Dostupné z: `https://tools.ietf.org/html/rfc5321`.
4. HUGHES, Lawrence. In: *Internet E-mail: protocols, standards, and implementation*. 685 Canton Street Norwood, MA 02062: Artech House, 1998, s. 239–240. ISBN 0-89006-939-5.
5. VAUDREUIL, G. *Enhanced Mail System Status Codes* [online]. RFC Editor, 2003-01 [cit. 2021-03-01]. RFC, 3463. RFC Editor. ISSN 2070-1721. Dostupné z: `https://tools.ietf.org/html/rfc3463`.
6. KUROSE, James F.; ROSS, Keith W. *Computer networking: a top-down approach*. 7. vyd. Edinburgh Gate, Harlow, Essex CM20 2JE, England: Pearson, 2017. ISBN 978-1-292-15359-9.
7. KABELOVÁ, Alena; DOSTÁLEK, Libor. In: *Velký průvodce protokoly TCP/IP a systémem DNS*. 5. vyd. Brno: Computer Press, 2008, s. 255–258. ISBN 978-80-251-2236-5.
8. YUJIAN, L.; BO, L. A Normalized Levenshtein Distance Metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 2007, roč. 29, č. 6, s. 1091–1095 [cit. 2021-04-06]. ISSN 1939-3539. Dostupné z DOI: `10.1109/TPAMI.2007.1078`.
9. MAREŠ, Martin; VALLA, Tomáš. In: *Průvodce labyrintem algoritmů*. Praha: CZ.NIC, z.s.p.o., 2017, s. 91–92. ISBN 978-80-88168-22-5.

10. MALÍK; SUCHÝ; TVRDÍK; VALLA. Dynamické programování. In: *Algoritmy a grafy 1 (BI-AG1), Přednáška č. 10* [online]. 2020–2021 [cit. 2021-04-05]. Dostupné z: `https://courses.fit.cvut.cz/BI-AG1/media/lectures/bi-ag1-p10-handout.pdf`. [Súbor prístupný po prihlásení do siete ČVUT – kópia súboru uložená na priloženej SD karte].

11. KIM, Isaac. Peeking into Email Validation Techniques. In: *Sparkpost* [online]. 2019 [cit. 2021-03-05]. Dostupné z: `https://www.sparkpost.com/blog/peeking-email-validation-techniques/`.

12. SPRINGGATE, Madison. Validating Email Addresses With Machine Learning. In: *SendGrid* [online]. 2019 [cit. 2021-03-05]. Dostupné z: `https://sendgrid.com/blog/validating-email-addresses-with-machine-learning/`.

13. How to check if an email address exists without sending an email? In: *Web Development* [online]. 2009 [cit. 2021-03-08]. Dostupné z: `https://www.webdigi.co.uk/blog/2009/how-to-check-if-an-email-address-exists-without-sending-an-email/`.

14. How to check if an email address exists without sending an email? In: *Stack Overflow* [online]. 2009 [cit. 2021-03-08]. Dostupné z: `https://stackoverflow.com/questions/565504/how-to-check-if-an-email-address-exists-without-sending-an-email`.

15. How to check if an email address is real. In: *Stack Overflow* [online]. 2013 [cit. 2021-03-08]. Dostupné z: `https://stackoverflow.com/questions/19261987/how-to-check-if-an-email-address-is-real-or-valid-using-php/19263515#19263515`.

16. How does email validation really work? In: *Quora* [online]. 2008 [cit. 2021-03-12]. Dostupné z: `https://www.quora.com/How-does-email-validation-really-work`.

17. How to handle SMTP email verification at scale without getting IPs blocked or SMTP requests rejected? In: *Stack Overflow* [online]. 2015 [cit. 2021-03-08]. Dostupné z: `https://stackoverflow.com/questions/33426503/how-to-handle-smtp-email-verification-at-scale-without-getting-ips-blocked-or-sm/44990019#44990019`.

18. Email SMTP validator. In: *Stack Overflow* [online]. 2015 [cit. 2021-03-12]. Dostupné z: `https://stackoverflow.com/questions/27474/email-smtp-validator`.

19. XIE, Yinglian; YU, Fang; ACHAN, Kannan; GILLUM, Eliot; GOLDSZMIDT, Moises; WOBBER, Ted. How Dynamic Are IP Addresses? *SIGCOMM Comput. Commun. Rev.* [Online]. 2007, roč. 37, č. 4, s. 301–312 [cit. 2021-03-08]. ISSN 0146-4833. Dostupné z DOI: `10.1145/1282427.1282415`.

20. JAJODIA, Sushil; ZHOU, Jianying. In: *Security and privacy in communication networks: 6th Iternational sic ICST conference, SecureComm 2010, Singapore, September 2010*. Berlin: Springer, 2010, s. 36–36. ISBN 978-3-642-16160-5.

21. HARRIS, Evan. The Next Step in the Spam Control War: Greylisting. In: *Greylisting: Whitepaper* [online]. 2003 [cit. 2021-03-13]. Dostupné z: `http://projects.puremagic.com/greylisting/whitepaper.html`.

22. *About The Spamhaus Project* [online] [cit. 2021-03-13]. Dostupné z: `https://www.spamhaus.org/organization/`.

23. HU, H.; PENG, P.; WANG, G. Characterizing Pixel Tracking through the Lens of Disposable Email Services. In: *2019 IEEE Symposium on Security and Privacy (SP)* [online]. 2019, s. 365–379 [cit. 2021-04-09]. Dostupné z DOI: `10.1109/SP.2019.00033`.

24. WEI, Chun; SPRAGUE, Alan; WARNER, Gary; SKJELLUM, Anthony. Mining Spam Email to Identify Common Origins for Forensic Application. In: *Proceedings of the 2008 ACM Symposium on Applied Computing* [online]. Fortaleza, Ceara, Brazil: Association for Computing Machinery, 2008, s. 1433–1437 [cit. 2021-04-09]. SAC '08. ISBN 9781595937537. Dostupné z DOI: `10.1145/1363686.1364019`.

25. *Free Email Verifier FAQ* [online] [cit. 2021-03-13]. Dostupné z: `https://verify-email.org/faq.html`.

26. *Feature-rich email verification service* [online] [cit. 2021-03-13]. Dostupné z: `https://quickemailverification.com/features`.

27. *Email Verifier - Verify any email address Hunter* [online]. 2021 [cit. 2021-03-13]. Dostupné z: `https://hunter.io/email-verifier`.

28. FESL, Jan. *Informácie ku SMTP validácií e-amilových adries* [elektronická pošta]. 2021 [cit. 2021-03-13]. Príjemca správy: strbaben@cvut.cz. 13. 1. 2021 23:12.

29. How those email validation services connect to SMTP servers and not getting blacklisted. In: *Quora* [online]. 2015 [cit. 2021-03-08]. Dostupné z: `https://www.quora.com/How-those-email-validation-services-like-https-emailchecker-io-http-quickemailverification-com-http-kickbox-io-connect-to-SMTP-servers-and-not-getting-blacklisted-by-those-servers`.

30. KOPPIKAR, U.; RAJUR, A.; JAYALAXMI, G. N. Efficient Word Processing Applications Using Radix Tree. In: *2019 4th International Conference on Recent Trends on Electronics, Information, Communication Technology* [online]. 2019, s. 1041–1046 [cit. 2021-04-06]. ISBN 978-1-7281-0630-4. Dostupné z DOI: `10.1109/RTEICT46194.2019.9016862`.

31. KARWOWSKI, Waldemar. Towards the data structure for effective word search. *Information Systems in Management* [online]. 2018, roč. 7, č. 4, s. 227–236 [cit. 2021-04-06]. Dostupné z: https://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-38e1a66d-42a7-4e9a-8229-4ac846f683fa.

32. JON BENTLEY, Jon; SEDGEWICK, Bob. Ternary Search Trees. *Dr. Dobbs* [online]. 1998 [cit. 2021-04-06]. Dostupné z: https://www.drdobbs.com/database/ternary-search-trees/184410528.

33. BAEZA-YATES, R.; NAVARRO, G. Fast approximate string matching in a dictionary. In: *Proceedings. String Processing and Information Retrieval: A South American Symposium (Cat. No.98EX207)* [online]. 1998, s. 14–22 [cit. 2021-04-06]. ISBN 0-8186-8664-2. Dostupné z DOI: 10.1109/SPIRE.1998.712978.

34. JOHNSON, Nick. Damn Cool Algorithms, Part 1: BK-Trees. In: *Nick's Blog* [online]. 2007 [cit. 2021-04-05]. Dostupné z: http://blog.notdot.net/2007/4/Damn-Cool-Algorithms-Part-1-BK-Trees.

35. NORVIG, Peter. How to Write a Spelling Corrector. In: *Peter Norvig* [online]. 2007 [cit. 2021-04-05]. Dostupné z: http://norvig.com/spell-correct.html.

36. HANOV, Steve. Fast and Easy Levenshtein distance using a Trie. In: *Steve Hanov's Blog* [online]. 2011 [cit. 2021-04-05]. Dostupné z: http://stevehanov.ca/blog/?id=114.

37. ARLOW, Jim; NEUSTADT, Ila. *UML and the unified process practical object-oriented analysis and design*. Great Britain: TPB, 2002. ISBN 0-201-77060-1.

38. WALLS, Craig. In: *Spring Boot in action*. Shelter Island: Manning, 2016, s. 2–7. ISBN 9781617292545.

39. H., Massé Mark. In: *REST API design rulebook*. Sebastopol, CA: O'Reilly, 2012, s. 5–5. ISBN 978-1-449-31050-9.

40. JACOBSON, Daniel; WOODS, Dan; BRAIL, Greg. In: *APIs: a strategy guide*. Sebastopol, CA: O'Reilly, 2012, s. 60–61. ISBN 978-1-449-30892-6.

41. *What is PostgreSQL?* [Online]. 2021 [cit. 2021-03-16]. Dostupné z: https://www.postgresql.org/about/.

42. LESZKO, Rafał. In: *Continuous Delivery with Docker and Jenkins*. Birmingham: PACKT PUBLISHING LIMITED, 2017. ISBN 978-1-78712-523-0.

43. *Introduction - Vue.js* [online]. 2021 [cit. 2021-03-25]. Dostupné z: https://vuejs.org/v2/guide/.

44. *The Catalog of Design Patterns* [online]. 2021 [cit. 2021-03-20]. Dostupné z: https://refactoring.guru/design-patterns.

45. Class EmailValidator. In: *EmailValidator (Apache Commons Validator 1.7 API)* [online]. 2020 [cit. 2021-03-25]. Dostupné z: https://commons.apache.org/proper/commons-validator/apidocs/org/apache/commons/validator/routines/EmailValidator.html.

46. *DNS Service Provider for the Java Naming Directory Interface (JNDI)* [online]. 2020 [cit. 2021-03-25]. Dostupné z: https://docs.oracle.com/javase/7/docs/technotes/guides/jndi/jndi-dns.html.

47. *dnsjava* [online]. 2021 [cit. 2021-03-25]. Dostupné z: https://github.com/dnsjava/dnsjava.

48. *Domains Project* [online]. 2021 [cit. 2021-04-28]. Dostupné z: https://github.com/tb0hdan/domains.

Dodatok A

Zoznam použitých skratiek

A Address Record

API Application Programming Interfaces

AWS Amazon Web Services

CNAME Canonical Name Record

CRLF Carriage Return, Line Feed

CSS Cascading Style Sheets

DNS Domain Name System

ESMTP Extended Simple Mail Transfer Protocol

FQDN Fully Qualified Domain Name

HTTP Hypertext Transfer Protocol

IP Internet Protocol

JEE Java Enterprise Edition

JNDI Java Naming Directory Interface

JPA Java Persistence API

MDA Mail Delivery Agent

MSA Mail Submission Agent

MTA Mail Transfer Agent

MUA Mail User Agent

MVC Model View Controller

MX Mail Exchanger Record

PTR Pointer Record

REST Representation State Transfer

RR Resource Records

SMTP Simple Mail Transfer Protocol

SP Space

TCP Transmission Control Protocol

URI Uniform Resource Identifier

XML Extensible Markup Language

Dodatok B

Slovník pojmov

blacklist – zoznam IP adries, z ktorých nie je vhodné prijímať e-maily

cache – pamäť, do ktorej sa dočasne ukladajú nejaké dáta

caching – metóda, fungujúca na princípe ukladania dát a dovoľuje opätovné použitie už raz získaných dát

callback – funkcia predaná do inej funkcie ako argument, ktorá je neskôr v tejto funkcií zavolaná

dynamická IP adresa – adresa, ktorá je priradená zariadeniu iba dočasne a po istom čase je zmenená za inú

endpoint – bod vstupu v komunikačnom kanáli pri komunikácii dvoch systémov, predstavuje URL nejakého serveru alebo služby

event – objekt, ktorý je vyvolaný užívateľom alebo inou príčinou a má za úlohu avizovať

framework – software vytvorený pre používanie vývojármi, na tvorbu ďalších aplikácií

instancia – konkrétne vytvorený objekt z nejakej triedy

IP adresa – jednoznačný identifikátor priradený jednotlivým staniciam

jenkins job – konkrétna úloha v Continuous Integration procese

scraping – vyťažovanie dát z webových stránok

spam – pomenovanie pre všetky nežiadúce správy, ktoré nám boli elektronicky doručené

spamer – pojem charakterizuje osobu rozosielajúcu spam

statická IP adresa – adresa, ktorá je zariadeniu priradená a nemení sa

validácia e-mailovej adresy – slovné spojenie znamená overenie existencie e-mailovej adresy

zabanovanie – v spojení s IP adresou, dostať sa na blacklist servera

Dodatok C

Dokumentácia integrácie služby

Popis hlavnej triedy

Užívateľ si stiahne javascriptový súbor, ktorý obsahuje potrebný kód ku komunikácií so službou.

Vstupné políčko, ktoré má slúžiť na validáciu e-mailových adries musí mať class atribút `email-input`.

Hlavnou triedou pre použitie užívateľom je `EmailApi` a jej rozhranie je nasledovné:

- `static createDefaultEmailApi(eventListenerConfig)` Slúži na vytvorenie instancie, ktorej sa predá eventListenerConfig argument. Táto instancia automaticky zaregistruje vstupné políčko s class atribútom `email-input` a na uvedené eventy v predanom parametre eventListenerConfig začne vykonávať našepkávanie, opravu alebo validáciu. Hlavný `div` tag pre našepkávanie a opravu disponuje `suggestions-div` class atribútom, jednotlivé `div` tagy v ňom disponujú `suggestion-div` class atribútom. Je teda možné tieto tagy upraviť vo vlastnom CSS súbore. Validácia e-mailových adries v tejto instancií nastavuje pri úspešnej validácií pozadie vstupného políčka na zeleno, pri neúspešnej na červeno.

- `static createEmailApi(callback, eventListenerConfig)` Slúži na vytvorenie instancie, ktorej sa predá užívateľova callback metóda. Vytvorená instancia pri uvedených eventoch volá užívateľov callback.

- `start()` Metóda slúžiaca na zaregistrovanie vložených eventov.

- `getSuggestions(input)` Metóda, ktorá získa možné doplnenia doménovej časti e-mailovej adresy a zavolá užívateľov callback. Túto metódu je možné volať, pokiaľ sa užívateľ rozhodol nezaregistrovať pre dopĺňanie žiadny event alebo potrebuje získať možné doplnenia nezávisle na tom, či sa event vykoná.

- `getCorrections(input)` Metóda, ktorá získa možne opravy doménovej časti e-mailovej adresy a zavolá užívateľov callback. Túto metódu je možné volať, pokiaľ sa užívateľ rozhodol nezaregistrovať pre opravu žiadny event alebo potrebuje získať možné opravy nezávisle od toho, či sa event vykoná.

- `validateEmail(emailAddress)` Metóda, ktorá priamo spustí validáciu e-mailovej adresy a zavolá užívateľov callback s výsledkom. Túto metódu je možné volať, pokiaľ sa užívateľ rozhodol nezaregistrovať pre validáciu e-mailových adries žiadny event alebo potrebuje vykonať validáciu nezávisle od toho, či sa event vykoná.

Parameter eventListenerConfig

Parameter eventListenerConfig je objekt s nasledujúcou štruktúrou

```
{
   suggestions : "eventType",
   corrections : "eventType",
   validation : "eventType"
}
```

- `suggestion` nastavuje event pre dopĺňanie doménovej časti adries

- `corrections` nastavuje event pre opravu doménovej časti adries

- `validation` nastavuje event validáciu e-mailových adries

Ak užívateľ nechce registrovať žiadny event pre danú operáciu vloží typ eventu `NO_EVENT`

Callback štruktúra

Callback od užívateľa má dva parametre:

- `status` parameter označuje akciu, ktorá sa vykonala a možné statusy sú `SUGGESTIONS, CORRECTIONS, EMAIL_VALIDATION`

- `value` parameter obsahuje výsledok zrealizovanej akcie a pre vrátené statusy `SUGGESTIONS, CORRECTIONS` bude value predstavovať pole s možnými výsledkami a pre status validácie `EMAIL_VALIDATION` to bude textový výsledok validácie

Jednotlivé statusy pre callback

- `SUGGESTIONS` status znamená, že vo `value` parametri sa nachádza pole s výsledkami doplnenia domény
- `CORRECTIONS` status znamená, že vo `value` parametri sa nachádza pole s výsledkami opravy domény
- `EMAIL_VALIDATION` status znamená, že vo `value` parametri sa nachádza textová výsledok validácie e-mailovej adresy

Výsledky validácie e-mailových adries

- `SERVER_HAS_USER` e-mailová adresa existuje
- `SERVER_DOES_NOT_HAVE_USER` e-mailová adresa neexistuje
- `INVALID_SYNTAX` syntaktická stránka e-mailovej adresy nie je správna
- `INVALID_DOMAIN` doména neexistuje alebo nedisponuje MX záznamom
- `FULL_INBOX` e-mailová adresa má plnú schránku
- `UNKNOWN` existencia resp. neexistencia e-mailovej adresy nie je známa

Dodatok D

Obsah priloženej SD karty

- `readme.txt` ..... stručný popis obsahu SD karty
- `src`
  - `EmailService` ..... zdrojové kódy implementácie služby
  - `FrontEnd` ..... zdrojové kódy front-endovej stránky
  - `JsLib` ..... zdrojové kódy javascriptovej knižnice
- `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF
- `materials` ..... ďalšie materiály využité v texte práce
  - `bi-ag1-p10-handout.pdf` ..... citovaná prednáška
  - `Fesl-Jan-email-communication.txt` ... e-mailová komunikácia s Ing. Jan Fesl, Ph.D.