VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

AUTOMATIZOVANÁ EXTRAKCE INFORMACÍ Z EMAILŮ

AUTOMATED EXTRACTION OF INFORMATION FROM EMAILS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **RASTISLAV KANDA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. FRANTIŠEK VÍDEŇSKÝ**
SUPERVISOR

BRNO 2019

Abstrakt

Táto práca má za úlohu oboznámiť sa s možnosťami extrakcie informácií z textu. Na základe získaných poznatkov navrhnúť a implementovať systém, ktorý bude schopný získať potrebné informácie z emailových správ. Navrhnutý systém má pomôcť firme Kiwi.com s.r.o. v spracovávaní emailových správ od dopravných spoločností. V momentálnej sitácii je možné tieto emailové správy spracovávať automaticky. Avšak na to aby mohli byť automaticky spracované, je nutné manuálne vytvoriť šablónu pre extrakciu dát zo správy. Zmenou v tomto prístupe je algoritmus ROBULA+, ktorý dokáže po zadaní lokátoru XPath, vygenerovať robustnejší XPath lokátor, ktorý bude odolnejší voči zmenám štruktúry v zdrojovom kóde HTML. Algoritmus ROBULA+ je použitý ako centrálny prvok pri automatizácii vytvárania šablón pre spracovávanie emailových správ. Úspešnosť implementovaného systému je možné označiť za dostačujúcu (približne 75%), čo znamená že v troch zo štyroch správach je možné úspešne získať referenciu k vytvorenej rezervácii.

Abstract

The purpose of this thesis is to familiarize oneself with methodology of information extraction from text. On the basis of acquired knowledge, propose a design and implement a system, which should be capable of gathering information from email messages. Proposed system should help Kiwi.com s.r.o. with processing of incoming email messages from travel companies. In current situation it is possible to process those email messages automatically. However, to process those messages automatically, it is necessary to manually create a template suitable for extraction. Possible alteration could be algorithm ROBULA+, which can generate more robust XPath locator from given XPath locator. These locators should be more resistant to changes in the HTML structure. ROBULA+ algorithm is a central point of automated creation of templates suitable for parsing email messages. Implemented system can be qualified with satisfactory successivity (approximately 75%). This means that system is able to find reference to created reservation in three out of four cases.

Kľúčové slová

extrakcia informácií, email, ROBULA+, automatizácia, REST API, XPath, Python

Keywords

information extraction, email, ROBULA+, automation, REST API, XPath, Python

Citácia

KANDA, Rastislav. *Automatizovaná extrakce informací z emailů*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. František Vídeňský

Automatizovaná extrakce informací z emailů

Prehlásenie

Prehlasujem, že som tuto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Františka Vídeňského. Ďalšie informácie mi poskytol Michal Cáb. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Rastislav Kanda
16. mája 2019

Poďakovanie

Rád by som poďakoval pánovi Ing. Františkovi Vídeňskému, za odborné a cielené vedenie tejto práce, pánovi Michalovi Cábovi z firmy Kiwi.com s.r.o. za poskytnuté rady a čas, ktorý mi venoval pri riešení tejto práce. Taktiež by som rád poďakoval svojej rodine a priateľom za trpezlivosť a podporu počas celého štúdia. V neposlednom rade by som sa rád poďakoval firme Kiwi.com s.r.o. za umožnenie nahliadnuť do diania firmy a možnosti nazbierať cenné skúsenosti počas vývoja tejto práce.

Obsah

Kapitola 1

Úvod

Už od nepamäti mali ľudia potrebu medzi sebou komunikovať. Existujú mnohé typy komunikácie, verbálna alebo neverbálna, priama alebo nepriama, formálna alebo neformálna. Jedna z foriem komunikácie je hlavným masovým komunikačným spôsobom po dlhé roky. Je to poštová komunikácia, ktorej použitie sa mierne uskromnilo na takmer výhradne oficiálnu komunikáciu medzi spoločnosťami, vládnymi článkami, bankami a inými útvarmi pracujúcimi s citlivými dátami. Poštová komunikácia, je tá ktorá stála na mieste ako inšpirácia, kedy s príchodom digitálnej doby a počítačov, vznikla e-mailová komunikácia. Dnes už len málokedy a ak vôbec, tak len z nostalgie alebo zo zvyku, pošlete niekomu list.

E-mailová komunikácia (skr. Email) nahradila takmer úplne všetku komunikáciu ktorá niekedy musela prebiehať pomaly a nedigitálne. V dnešnom digitalizovanom svete je samozrejmosťou, že človek má možnosť v sekunde odoslať niekomu e-mailovú správu, alebo byť s ním v spojení takmer okamžite pomocou iných digitálnych komunikačných kanálov. Medzi novodobé komunikačné kanály môžeme okrem iného zaradiť aj internetové hovory, kde spadajú video a audio hovory alebo internetový chat.

Spolu s rozvojom firemnej stratégie a neustálym expandovaním, prichádza aj potreba spracovávať emailové správy vo veľkom množstve. Tento problém má aj firma Kiwi.com s.r.o., ktorej taktiež s pribúdajúcim počtom vytvárania podpory novým leteckým spoločnostiam a s vytváraním nových dohôd medzi leteckými spoločnosťami a touto firmou, vznikol problém v podobe veľkého počtu emailových správ. Firma Kiwi.com s.r.o. je jednou z najrýchlejšie rastúcich firiem na svete, ktorej mottom je urobiť cestovanie lepším, v každom možnom smere. Medzi jej hlavné produkty patrí predaj leteniek a iných cestovných lístkov, ktorý podporuje vlastný vyhľadávací mechanizmus a taktiež non-stop podpora v 13 svetových jazykoch.

Táto práca má v tomto probléme pomôcť firme Kiwi.com. Jej cieľom je ponoriť sa do problematiky súvisiacej s hromadným spracovávaním emailových správ a taktiež vytvoriť a realizovať návrh systému, ktorý bude schopný vytvoriť šablóny potrebné pre spracovanie informácií z novoprichádzajúcich emailových správ.

V momentálnej situácii, je možné tieto emailové správy spracovať automaticky, avšak na to aby boli spracované automaticky, je nutné manuálne vytvoriť šablónu. Šablóna je špecifická pre každý typ emailovej správy, napríklad sa môže líšiť od každej prepravnej spoločnosti. So zväčšujúcim sa počtom prichádzajúcich správ, narastá aj potreba vytvárania nových šablón a taktiež neubúda potreba údržby už existujúcich.

Kapitola 2

Teoretická časť

V tejto kapitole sú definované základné pojmy, nutné k správnemu pochopeniu problematiky, alebo nutné k vyriešeniu problémy, ktoré má táto práca za úlohu vyriešiť.

2.1 Metódy spracovania jazyka a textu

V tejto sekcii som čerpal zo zdrojov [15] a [6]. Jazyk je systém, ktorý nám slúži ako základný prostriedok pre komunikáciu. Jazyk všeobecne neslúži len ľuďom, ale aj počítačom. Takýmto jazykom môžeme nadnesene povedať programovacie jazyky. Mnoho krát sa ale stretávame s bariérou, môžeme to chápať aj ako jazykovú bariéru v pravom zmysle slova, kedy počítač nedokáže porozumieť jazyku nášmu, ľudskému prirodzenému jazyku. Touto tématikou sa zaoberá odvetvie nazývané prirodzené spracovanie jazyka – *Natural Language Processing - NLP*.

2.1.1 Natural Language Processing (*NLP*)

Spracovanie prirodzeného jazyka je odvetvie počítačovej vedy a výskumu, ktoré sa rozvinulo z výskumu jazyka a výpočtovej lingvistiky[1] a spadá pod *Umelú Inteligenciu*. Cieľom NLP je vyvinúť a navrhnúť aplikácie, ktoré porozumejú ľudskej interakcii s počítačmi a inými zariadeniami pomocou použitia prirodzeného jazyka [16].
Hlavné odvetvia NLP sú:

- *Question-Answering systems (QAS)* - v dnešnej dobe čím ďalej, tým viac a viac rozšírenejšie použitie v rámci hlasových asistentov od spoločností ako Google, Microsoft alebo Apple, ktorými hlavnou úlohou je uľahčiť nám prístup k informáciám. Sú to systémy, ktoré dokážu poskytnúť odpovede na jednoduché otázky typu, „Aké je dnes počasie?“ alebo „Ako skončil zápas môjho obľúbeného týmu?“. Tieto systémy už dnes existujú a niektoré z nich dokážu odpovedať aj na zložitejšie otázky, no väčšina z nich sa opiera o kľúčové slová v texte a nespracováva ho ako celok.

- *Summarization* - táto skupina sa zameriava na spracovávanie dát z kolekcie súborov alebo emailov, z ktorých poskytuje ucelený prehľad informácií, ktoré sa v danej kolekcii nachádzajú.

- *Machine Translation* - je jednou z prvých oblastí výskumu spracovania prirodzeného jazyka. Aplikácie ako napríklad Google Translate sú deň čo deň lepšie a lepšie, čo

[1] computational linguistics

sa týka presnosti prekladu, rýchlosti prekladu a správnosti prekladu požadovaného úryvku textu.

- *Speech Recognition* - je jedným z najťažších problémov v NLP. V nedávnej dobe bol dosiahnutý postup a zlepšenie v rámci tohoto oboru. Tieto systémy, odborne nazývané *ASR – Automatic Speech recognition*, sú všade-prítomné a bežne používané v dnešnej dobe, ako napríklad hlasový asistenti v telefónoch, autách a počítačoch. Tieto systémy sú avšak stále obmedzené na užší kontext. Človek by nemal rozvádzať vety a umelecky opisovať svoj problém, ale mal by sa držať témy a vysloviť svoj problém čo najkratšie a najvýstižnejšie. V opačnom prípade tieto systému nemusia rozpoznať daný problém správne.

- *Document classification* - na strane tých úspešnejších sa môže pýšiť táto oblasť výskumu, nakoľko je to jedna z oblastí kde NLP dosahuje najlepšie výsledky. Táto oblasť sa zameriava na tzv. „škatuľkovanie“, alebo inak povedané schopnosť rozradiť jednotlivé súbory alebo informácie do správnej kategórie. Jedným z hlavných dôvodov úspechu je aj to, že na natrénovanie týchto systémov stačí relatívne jednoduchá množina správne rozdelených súborov alebo informácií.

2.1.2 Extrakcia informácií z textu

Dáta môžu mať rôznu podobu, jedna z najdôležitejších je tá kedy sú usporiadané do ľubovoľnej štruktúry. To znamená, že existuje skupina pravidiel, ktoré ak budú aplikované, tak ich výsledkom budú vždy dáta v rovnakom tvare (ak sa obsah dát nezmenil). Tým pádom je jednoduché odvodiť vzťahy a organizáciu dát pomocou predvolenej štruktúry. V tabuľke 2.1 je možné vidieť ukážku štrukturovaných dát.

Názov organizácie	Sídlo organizácie
FBI	Washington D.C.
Facebook	Menlo Park
Fedex	Memphis

Tabuľka 2.1: Ukážka štrukturovaných dát

Ak by tieto dáta boli uložené v jazyku Python ako množina `(nazov_organizacie, sidlo_organizacie)`, bolo by možné nad nimi uskutočniť niekoľko operácií. Napríklad otázka, „Ktorá organizácia sídli v meste Memphis?“, by sa dala preložiť ako

```
>>> print [org['nazov'] for org in data if org['sidlo'] == 'Memphis']
```

Výsledkom čoho by bol zoznam prvkov, obsahujúci jeden prvok s názvom spoločnosti Fedex.

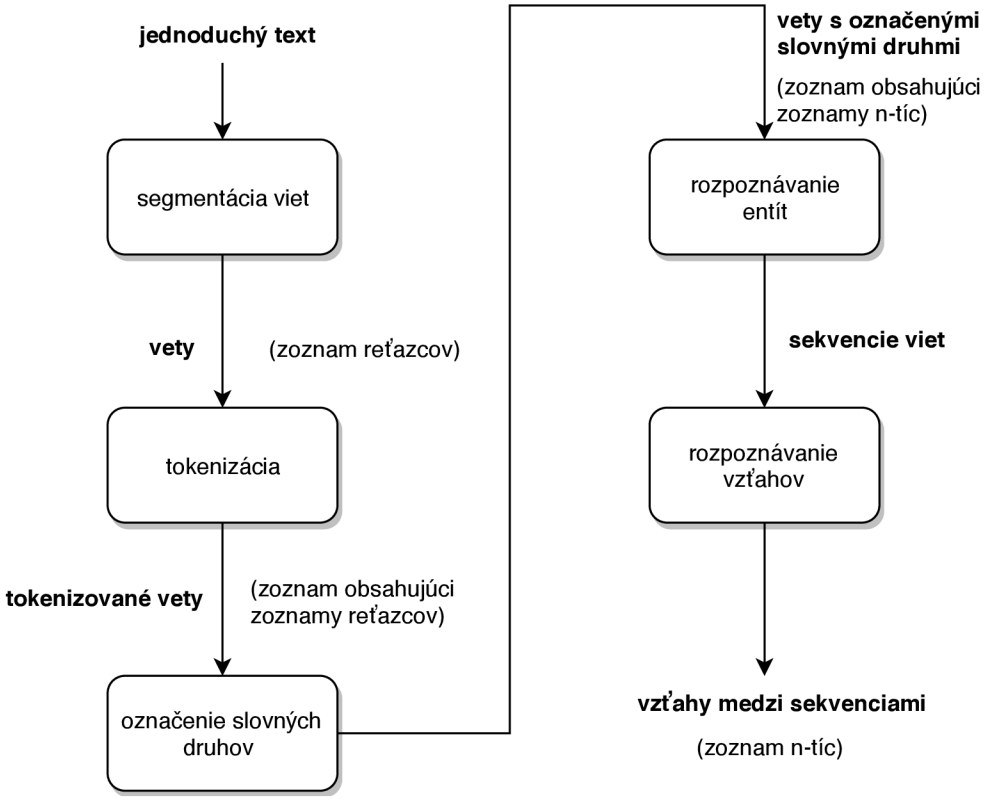
```
>>> ['Fedex']
```

Problém ale prichádza ak tieto dáta neobsahujú žiadnu viditeľnú štruktúru. Pre človeka, ktorý tieto dáta vníma, alebo tento text číta, je jasné o aký druh dát ide a akú informáciu podávajú na základe kontextu, ktorý sa dozvie z obsahu dát.

Vezmime si úryvok článku, prevzatý a preložené zo stránok spoločnosti Fedex[2]:

> „FedEx Express dokončuje zásadný projekt rozširovania prevádzky strediska na letisku Roissy–Charles de Gaulle, ktoré sa tak stáva druhým najväčším operačným sídlom po Memphise. FedEx Express predstavuje produkt FedEx International Economy® pre menej urgentné zásielky, táto služba je dostupná v 90 krajinách a územiach celého sveta.“

Človek, ktorý si tento úryvok prečíta, bude vedieť odpovedať na túto otázku. Aby počítač dokázal porozumieť tomuto úryvku, je potrebné o mnoho väčšie úsilie na dosiahnutie podobného výsledku ako v prípade človeka. O túto úlohu sa stará metóda, zvaná *Information Extraction – Extrakcia informácií*, ktorá dokáže vytvoriť z dát v neštrukturovanom formáte, štrukturovaný formát, na ktorý neskôr môžeme použiť rôzne ďalšie metódy a nástroje (SQL, Python) pre zodpovedanie našej názornej otázky.



Obr. 2.1: Príklad základného postupu extrakcie informácií z textu. Vytvorené podľa zdroja [2].

Na obrázku 2.1, je možné vidieť príklad jednoduchého systému, ktorý by sa dal použiť na extrakciu informácií z textu. V tomto systéme je použitých niekoľko základných metód extrakcie informácií.

- Segmentácia viet – v tejto časti je súvislý text rozdelený na menšie časti, vety. Tento proces je náročný z hľadiska správneho rozlíšenia ukončenia viet. Niektoré vety môžu

[2] http://www.fedex.com/cz/about/company-info/history.html

mať klasickú stavbu, teda obsahujú viacero slovných druhov a sú ukončené interpunkčným znamienkom. V tomto prípade je rozpoznanie konca vety jednoduché. Existujú ale aj vety, ktoré nemajú klasickú stavbu, ako napríklad veta: „Hurá!“.

- Tokenizácia – je proces rozdelenia viet na menšie zoznamy reťazcov, ako napríklad na vety a diakritiku. V tejto časti je nutné zvoliť správny prístup, nakoľko rozdelenie viet pomocou medzier, nemusí byť správne a vzniknuté tokeny (časti celku) by nemuseli korešpondovať s pôvodnou vetou.

- Označenie slovných druhov – v tejto časti systému sú označené tokeny spracované a je k nim pridaná informácia o tom, v akej časti textu sa nachádzajú, alebo inými slovami povedané, akým slovným druhom je konkrétny token.

- Rozpoznávanie entít – slovné druhy avšak neposkytnú takmer žiadnu výpovednú hodnotu o význame vety. Práve technika rozpoznávania entít, k tomu môže napomôcť. Táto technika je používaná na identifikovanie entity zadaného tokenu vo vete. Identifikovanie entity môže nadobúdať niekoľkých hodnôt, ako napríklad či zadaný token špecifikuje názov organizácie – ORGANIZATION, meno osoby – PERSON alebo názov miesta – LOCATION. Rozpoznávanie entít je možné realizovať pomocou ručne vytvorených pravidiel a množstva lingvistických dát, čo zahŕňa množstvo času expertov venujúcich sa tomuto oboru alebo pomocou strojového učenia. Tento prístup avšak vyžaduje veľké množstvo trénovacích dát opatrených anotáciami.

- Rozpoznávanie vzťahov – ako posledným, avšak nezanedbateľným krokom v tomto systéme je rozpoznávanie vzťahov. Túto techniku je možné realizovať pomocou strojového učenia alebo pomocou ručne zadaných pravidiel. V prípade strojového učenia je možné použiť natrénovaný model na rozsiahlej množine lingvistických dát (korpuse)[3], kedy strojové učenie bude skúmať vzťahy a vzory medzi slovami. Ručne zadané pravidlá môžu skúmať napríklad presné vzťahy medzi slovami označujúcimi nejakú entitu doplnených o slová, ktorými je možné definovať vzťah medzi nimi. Napríklad ORGANIZATION: Brookings Institution, „je výskumnou skupinou v“ LOCATION: Washington.

2.2 Email

Slovo *Email* je skráteným tvarom slovného spojenia *Electronic mail*, čo v preklade znamená *Elektronická pošta*. Email je jedna z najpoužívanejších služieb dnešného internetu.

Pôvodný email, ktorý čerpal námet z ARPANET–u[4], predchodcu dnešného Internetu a v protokole CPYNET[5], protokolu podobnému FTP[6], vznikol už v 80.rokoch. CPYNET umožňoval posielať súbory z jedného počítača, do umiestnenia na inom vzdialenom počítači. Prenos prebiehal tak, že pridal znak `@` a za neho adresu vzdialeného počítača, ktorý ako odpoveď na tento príkaz vzal správu a pridal ju nakoniec správneho, vopred definovaného súboru. Inými slovami povedané príkaz `SNDMSG martin@computerinotherroom` by našu správu pridal do súboru `/usr/martin/messages` zariadení `computerinotherroom`.

Ďalej sa ARPANET rozvíjal za pomoci univerzít, ktoré adaptovali a rozvíjali túto technológiu ďalej. Bez univerzít by to v tej dobe nemohlo fungovať. Neskôr v 90. rokoch bolo

[3] http://www.nltk.org/book/ch02.html#fig-text-corpus-structure
[4] http://www2.idehist.uu.se/distans/ilmh/Ren/dig-arpanet.htm
[5] https://blog.cloudflare.com/the-history-of-email/
[6] https://tools.ietf.org/html/rfc959

jeho použitie rozšírené aj mimo akademickú sféru a jedným z hlavných míľnikov, ktoré definovali systém elektronickej pošty, ako ho poznáme dnes, bola definícia protokolu SMTP (Simple Mail Transfer Protocol) pre prenos emailových správ [11] a štandard pre tvorbu emailových správ [17].

2.2.1 Formát správ elektronickej pošty

Pravidlá správneho formátu emailovej správy definuje štandard RFC 5322 [17], predtým RFC 822[7] a RFC 2822[8].

V dnešnej dobe formát emailovej správy, ktorý bol definovaný pred viac ako 35 rokmi, už ale nie je dostačujúci. Keďže obsahuje len podporu pre textové správy [4], boli zavedené rôzne rozšírenia, ktoré podporujú prenos obrazových či multimediálnych dát a taktiež emailových príloh v rôznych formátoch. Jedným z najpoužívanejších dnešných rozšírení je *MIME* (Multipurpose Internet Mail Extension), ktoré podporuje štrukturalizáciu dát tak, aby mohli správy obsahovať dáta v rôznych kódovaniach, rôzne netextové dáta, ako napríklad dáta vo zvukovom alebo video formáte, obrázky a taktiež podporu pre viacdielne správy.

2.3 Representational State Transfer (*REST*)

REST – Representational State Transfer, je skupina pravidiel, ktoré definujú štýl architektúry nutnej pre správne vytváranie webových aplikácií [19]. REST API je aplikačné rozhranie takmer každej webovej aplikácie. Táto technológia je orientovaná na takzvané *resources* alebo zdroje. Zdroje sú niečo dôležité natoľko, aby sa na to dalo odkazovať ako na vec. Ak užívatelia vedia vytvoriť odkaz, stiahnuť si ho alebo uložiť ho do vyrovnávacej pamäte, potom to môžeme nazývať ako zdroj. Vo väčšine je zdrojom niečo, čo je reprezentované vo forme toku dát, niečo čo môžeme uložiť na pevný disk, riadok v tabuľke databázy alebo výsledok algoritmu [19].

REST využíva na prístup k zdrojom takzvaný *URI* lokátor alebo adresu. URI si môžeme predstaviť ako cestu k objektu alebo zdroju popisujúcu objekt v sieti Internet. Je tu podobnosť s cestou k súboru v súborovom systéme. Taktiež cestu v súborovom systéme môžeme nazvať URI alebo *Universal Resource Identifier* a teda *všeobecný identifikátor zdroja*. S technológiou REST je nutné definovať aj protokol HTTP, ktorý je potrebný pre správne fungovanie tejto tehcnológie.

2.3.1 HyperText Transfer Protocol (*HTTP*)

Ak by sme chceli klasifikovať ľudí, mohli by sme začať tým, čo majú všetci spoločné. Napríklad, že každý z nás má spoločné základné potreby ako hlad a smäd, u každého platia rovnaké fyzikálne zákony, a každý má rovnakú štruktúru buniek. Ak by sme chceli klasifikovať webové aplikácie, tak by bolo vhodné definovať protokol HTTP, nakoľko je to to, čo má väčšina webových aplikácií spoločné.

HTTP – *HyperText Transfer Protocol* je protokol založený na dokumentoch. Pomocou tohoto protokolu klient zabalí dokument do „obálky“ a pošle ju serveru pre ďalšie spracovanie. Server túto požiadavku spracuje a odpovie na ňu taktiež pomocou dokumentu

[7] https://tools.ietf.org/html/rfc822
[8] https://tools.ietf.org/html/rfc2822

zabaleného do „obálky“. Protokol HTTP má striktné pravidlá ako táto obálka môže vyzerať, ale na druhej strane už takmer vôbec nezáleží na tom čo sa v danej obálke nachádza. V tabuľke 2.2 môžeme vidieť najpoužívanejšie stavové kódy, ktoré nám server môže vrátiť ako odpoveď. Stavové kódy jednoznačne reprezentujú typ odpovede, aby daný užívateľ vedel ako má danú odpoveď spracovať. Klient môže stavové kódy získať pomocou toho, že si prečíta prvú časť odpovede až po znak `CR+LF` a podľa toho môže zvoliť vhodnú interpretáciu metadát a samotného tela odpovede.

Stavový kód	Popis odpovede
2xx – OK	
200 *OK*	Kód popisujúci stav, kedy bol požiadavok splnený bez problémov.
201 *Created*	Kód popisujúci stav, kedy server vytvoril nový zdroj na užívateľovu požiadavku.
204 *No content*	Kód popisujúci stav, kedy bola požiadavka prijatá správne, no odpoveď neobsahuje žiadne dáta.
3xx – Presmerovanie	
301 *Moved Permanently*	Správa bola prijatá a odpoveď sa nachádza na inej adrese.
4xx – Chyba na strane užívateľa	
400 *Bad Request*	Kód popisujúci generickú chybu na strane klienta.
401 *Unauthorized*	Kód popisujúci stav, kedy sa klient snažil pristúpiť niekam, kde neposkytol dostatočnú mieru autorizácie alebo tam nemá prístup vôbec.
404 *Not Found*	Kód popisujúci stav kedy sa klient snaží pristúpiť na miesto, kde server nedokáže namapovať žiadne dáta.
406 *Not Acceptable*	Kód popisuje chybu, ktorá je vyvolaná príliš striktnými podmienkami užívateľa, ktoré server nedokáže splniť.
5xx – Chyba na strane serveru	
500 *Internal Server Error*	Kód, ktorý popisuje generickú chybu na strane servera, najčastejšie neošetrenú výnimku.
503 *Service Unavailable*	Kód popisujúci stav kedy je daná služba nedostupná, napríklad z dôvodu údržby.

Tabuľka 2.2: Dôležité stavové kódy HTTP. Zdroj:[19]

Protokol HTTP taktiež definuje niekoľko metód pre prístup k zdrojom. Veľkou výhodou je fakt, že HTTP metódy sú štandardizované a všetky servery ktoré komunikujú pomocou HTTP používajú rovnaké metódy. To zabezpečuje prehľadnosť a kompatibilitu komunikácie. Medzi najzákladnejšie patria metódy GET, POST, PUT, DELETE a HEAD [19], [7].

- GET – je hlavná metóda určená pre získanie zdrojov zo servera.
- POST – je metóda určená pre odosielanie zdrojov na server. V danom prípade môže ísť o vytváranie daného zdroja, upravovanie jeho podoby alebo o pridávanie dát k už existujúcim dátam.

- PUT – pomocou tejto metódy môžeme odoslať dáta na server, ktoré chceme buď vytvoriť alebo prepísať. Po odoslaní metódy PUT je vhodné použiť metódu GET pre overenie či nová štruktúra dát po zmene, korešponduje s chceným formátom a údajmi.
- DELETE – je metóda určená pre zmazanie refencie na zdroj na serveri. V tomto prípade zdroj, referencovaný URI identifikátorom, môže alebo nemusí byť zmazaný z úložiska serveru. Táto vlastnosť závisí na implementácii daného serveru.
- HEAD – metóda HEAD je identická s metódou GET, avšak server by v prípade HEAD metódy nemal posielať telo odpovede. To umožňuje dotazujúcemu získať metadáta správy alebo jednoducho overiť či je odoslaný URI identifikátor platný.

2.4 HyperText Markup Language (*HTML*)

Za úspechom dnešného webu stojí značkovací jazyk HTML – *HyperText Markup Language*. Každá webová stránka sa zakladá na kóde v jazyku HTML aspoň minimálne. Minimálne znamená, že obsahuje adresu na kód, ktorý je generovaný pomocou iného programovacieho jazyka, ako napríklad PHP alebo Javascript.

Jazyk HTML bol vytvorený ako derivát využívajúci značky z jazyka SGML – *Standard Generalized Markup Language*[9], v podobe stručného dokumentu popisujúceho niekoľko elementov používaných pre tvorbu webových stránok. Mnohé z týchto elementov popisovali rôzne časti webovej stránky, napríklad záhlavie, odstavce a zoznamy. S postupným zvyšovaním verzie jazyka pribúdala podpora pre nové a nové elementy, ktoré bolo možné v tomto jazyku a teda na webových stránkach definovať.#todo Najnovšou verziou tohoto značkovacieho jazyka (HTML) je verzia 5, HTML5.

Verzia HTML5 prináša so sebou radu nových vylepšení, no prevažne pozostáva z predchádzajúcich verzií tohoto jazyka.

Webové stránky dnes výrazne podliehajú trendom a vlastníci stránok, alebo jednoducho tí, ktorí chcú ísť s dobou, musia svoje stránky tomu aj prispôsobiť a pravidelne aktualizovať.

2.4.1 HTML v emailových správach

Na počiatkoch emailovej komunikácie, väčšina emailových správ, ak nie všetky, pozostávali z jednoduchého textu. Dnes sa ale v emailovej komunikácii, hlavne v tej komerčnej, kam zapadajú napríklad správy od leteckých spoločností, reklamné správy a marketingové správy, bez štruktúry v jazyku HTML nezaobídeme. Na rozdiel od webových stránok, v emailoch sa nepoužívajú žiadne rozšírenia v podobe skriptovacích jazykov, ako napríklad JavaScript vo webových stránkach. Je to z bezpečnostných dôvodov, nakoľko väčšina emailov obsahuje veľké množstvo citlivých osobných informácií. [3] Prevažuje tam teda kód HTML a kaskádové štýly, ktoré emailu dávajú správny formát a zaisťujú správne zobrazenie u každého emailového klienta.

2.5 eXtensible Markup Language (*XML*)

V tejto sekcii som čerpal zo zdrojov [23]. Jazyk XML bol vytvorený ako ďalší z derivátov využívajúci značky z jazyka SGML[10], tentokrát bol ale cielený na menšie platformy

[9] SGML – je štandardizovaná verzia jazyka GML – *General Markup Language*, skôr ako programovací jazyk, si SGML môžeme predstaviť ako pravidlá akým spôsobom sa majú dokumenty tohoto typu vytvárať.
[10] viz. 9

- webové prehliadače. Tvorcovia jazyka XML vytvorili jazyk, ktorý môžeme nazvať tzv. samo-popisným. Musí byť teda jasné, aké dáta sa nachádzajú v danom dokumente hneď pri prvom pohľade na štruktúru a názvy značiek (tagov) v dokumente.

2.5.1 XML a HTML

Dôraz jazyka XML je orientovaný špecificky na obsah dát a flexibilitu ich spracovania. Keďže v tomto prípade si autor môže špecifikovať svoje prvky a štruktúru dokumentu XML, je tento jazyk vo svojej podobe nevhodný pre webové stránky. Práve táto flexibilita má na svedomí to, že webové prehliadače alebo užívateľskí agenti nemajú v žiadnom prípade šancu, zobraziť dáta tak, ako to autor zamýšľal. Naopak, XML sa hodí na uloženie aplikačných dát. Jazyk HTML má naopak za úlohu postarať sa o to aby dáta boli správne a čitateľne zobrazené, alebo inak povedané jazyk HTML dáva dôraz hlavne na prezentáciu dát.

Napriek veľkej podobnosti jazyka XML s jazykom HTML a ďalšími značkovacími jazykmi, existujú ale pravidlá, ktoré špecifikujú XML. V tejto časti budem porovnávať jazyk XML s jazykom HTML.

Vlastnosti XML:

- pri názvoch prvkov a atribútov **záleží** na veľkosti písmen
- všetky prvky musia byť správne ukončené
- všetky vlastnosti musia mať špecifikovanú hodnotu
- všetky hodnoty vlastností musia byť v úvodzovkách

Vlastnosti HTML:

- prvky a atribúty v HTML sú tzv. *case-insensitive* a teda u nich **nezáleží** na veľkosti písmen[11]
- v niektorých verziách HTML je povolené definovať vlastnosti, ktoré nemajú priradenú žiadnu hodnotu
- jazyk HTML má podporu pre nepárové prvky, tzv. tie ktoré nemajú byť ukončené

2.6 XPath

Teória v tejto časti bola čerpaná z [27], [20] a [19]. Syntax XPath bola vytvorená pre popis jednotlivých častí XML dokumentu, viac o XML v sekcii 2.5.

Systém XPath bol vytvorený a zmýšľaný na použitie a adresovanie v jednotlivých atribútoch v XML dokumente. Táto technológia je štandardizovaná a zastrešovaná organizáciou W3C[12]. Syntax je zmiešaním základného programovacieho jazyka, napríklad `$x*6` a ciest v systémoch podobným systému *Unix – Unix-like systems* (napríklad `/cesta/do/sveta/unixu`). S XPath-om sa môžete odkazovať na prvý `<elem>` element, na každý `<elem>` element, ktorý obsahuje text `„Eric"` alebo na všetky elementy `<tr>` v celom dokumente. V tejto sekcii si bližšie špecifikujeme túto technológiu.

[11] http://w3c.github.io/html-reference/documents.html#syntax-document-xml
[12] https://www.w3.org/

2.6.1 Dátový model XPath

XPath verzie 1.0 vníma XML dokument ako strom uzlov. Tento strom je veľmi podobný štruktúre stromu DOM, ktorý bude uvedený v sekcii 2.7.

XPath verzie 2.0 vníma prijímané dáta ako postupnosť. Postupnosť, ktorá môže obsahovať rôzne typy uzlov, tak ako aj atomické hodnoty; atomická hodnota je hodnota, ktorá už nemôže byť rozdelená na menšie časti.

2.7 Document object model (*DOM*)

DOM – *Document object model*, je programové rozhranie pre HTML a XML dokumenty [14]. Webová stránka môže byť špecifikovaná dvomi spôsobmi, pomocou zdrojového kódu (kódu v HTML alebo XML) alebo pomocou objektu DOM. V oboch prípadoch ide o úplne rovnakú stránku, no s rozdielom, že DOM objekt nám umožňuje manipuláciu s danou stránkou. Je to spôsob akým komunikujú programovacie jazyky s webovou stránkou. DOM reprezentuje objektovo-orientovanú štruktúru stránky, ktorá môže byť upravovaná a manipulovaná pomocou programovacích jazykov, ako napríklad JavaScript.

Výpis 2.1: Porovnanie štruktúry HTML a štruktúry DOM reprezentovanou v tvare stromu elementov

```
<!DOCTYPE html>
<html class=e>
 <head>
   <title>Aliens?</title>
 </head>
 <body>Why yes.</body>
</html>
```

```
└─ Document
   ├─ Doctype: html
   └─ Element: html class=e
      ├─ Element: head
      │  └─ Element: title
      │     └─ Text: Aliens?
      ├─ Text: ⏎
      └─ Element: body
         └─ Text: Why yes.⏎
```

Na výpise 2.1, je možné vidieť štruktúru, akou DOM rozhranie reprezentuje zadanú stránku. DOM vytvára objekty z každého elementu v zdrojovom kóde a tie následne usporiada do štruktúry stromu. Strom je možné si predstaviť ako postupnosť elementov, kde prvky majú svoj rodičovský prvok a nula alebo viac dcérskych prvkov. Počiatočný prvok, ktorý je iba jeden, obsahuje iba dcérske prvky. Prvky ktoré nemajú dcérske prvky sa nazývajú listy stromu (nodes).

Kapitola 3

ROBULA+

V tejto časti sa pozrieme na použitie technológie XPath v praxi, keďže algoritmus s názvom **ROBULA+** je založený práve na tejto technológii. Algoritmus ROBULA+ vznikol v roku 2016 ako aktualizovaná verzia algoritmu *ROBULA* z roku 2014 [13]. Algoritmus sa zameriava na zlepšenie robustnosti lokátorov XPath, ktoré su v dnešnej dobe stavebným kameňom automatizovaného testovania webových aplikácií.

3.1 Manuálne testovanie webových aplikácií

Pred tým ako si bližšie priblížime automatizáciu testovacieho postupu, je vhodné najprv pripomenúť prečo tieto postupy vznikli a čo stojí za úspechom ich používania. Webová aplikácia, ako každá iná aplikácia v praxi potrebuje spĺňať niekoľko podmienok a teda je potrebné aby prešla niekoľkými krokmi verifikácie a validácie či spĺňa všetky požiadavky zákazníka a taktiež či implementácia súhlasí so špecifikáciou. Tento postup je pracný a pomerne jednotvárny. Človek musí overiť či aplikácia spĺňa všetky požiadavky manuálne a opakovane. Ručné a opakované testovanie, priam okamžite u niektorých osôb vedie k úspešnému unudeniu a neochoty daného človeka pracovať na danej úlohe. To bolo aj hlavnou príčinou prečo sa pozornosť začala upriamovať na možnosti automatizácie. Taktiež k tomu musíme prirátať ľudskú chybovosť a tým pádom tento druh testovania nie je natoľko spoľahlivý. Tento druh testovania je taktiež časovo náročný, človek nezvládne v jednej minúte spustiť niekoľko desiatok alebo stoviek testov, narozdiel od počítača dokáže spustiť práve len niekoľko jednotiek.

3.2 Automatizované testovanie webových aplikácií

Testovanie webovej aplikácie so sebou prináša niekoľko úskalí. V bežnom živote človek len málokedy postrehne, že nejaká webová stránka sa zmenila. Opak je ale pravdou, každý deň sa totiž nejaká webová stránka (aplikácia) pozmení čo i len o malý kúsok, čo v niektorých prípadoch môže znamenať až katastrofálne následky vo výsledkoch testov. Webové stránky sú totiž generované pomocou značkovacieho jazyka HTML a taktiež je možné použiť aj skriptovací jazyk JavaScript alebo jazyk PHP. Jazyk HTML je bližšie popísaný v sekcii 2.4.

Automatizované testovanie webových aplikácií je postavené na testoch, ktoré sa majú vykonať v rôzne špecifikovanom poradí. Napríklad ak chceme porovnávať výsledky rôznych internetových vyhľadávačov, musíme najprv zadať požiadavku na vyhľadanie s uvedenými kľúčovými slovami. Testy sú zostavené zo sekvencie algoritmizovaných príkazov. Väčšina

príkazov sa odkazuje na nejaký element na webovej stránke, s ktorým interaguje, čo znamená, že buď sa snaží prečítať jeho hodnotu, alebo sa ju snaží pozmeniť. Pod zmenou elementu si môžeme predstaviť vyplnenie hodnoty do textového poľa, ako napríklad vyplnenie kľúčových slov na vyhľadávanie alebo zaškrtnutie výberového tlačítka (checkboxu), ako napríklad povolenie rozšíreného vyhľadávacieho režimu vo vyhľadávači.

3.2.1 Vyhľadávanie elementov na stránke

Pre vyhľadávanie elementov na stránke, inak možno nazvať ako lokalizovanie, nám slúžia CSS selektory alebo XPath selektory. Technológia XPath je bližšie popísaná v sekcii 2.6, CSS selektory nie sú bližšie v tejto práci špecifikované, avšak je možné sa o nich bližšie dočítať v [9].

Vytváranie týchto elementov môže znieť ako jednoduchá činnosť, ako napríklad zvoliť tretí element v tabuľke alebo štvrtý obrázok v poradí. To má ale svoje úskalia. V prípade ak sa na nejakej stránke, ktorá môže byť napríklad prehľad členov tímu v nejakej spoločnosti, zmení počet osôb tímu a na prvé miesto pribudne nový člen. Tým pádom obrázky členov tímu na stránke zmenia svoje poradie a naša aplikácia bude priraďovať fotografiu niekoho úplne iného. Lokátory teda musia byť odolné voči zmene a v ideálnom prípade musia odkazovať na konkrétny jedinečný prvok na stránke.

3.2.2 Momentálne dostupné nástroje

V dobe písania je dostupných niekoľko nástrojov, pomocou ktorých je možné automatizované testovanie aplikácií vykonávať aj pomocou minimálnej znalosti programovania. Niektoré z nástrojov majú vlastné grafické rozhranie a umožňujú funkciu nahrávania sedenia (angl. *session*), kde si zaznamenajú všetky svoje úkony na danej stránke. Vytváranie testovacích šablón bez nutnosti programovania má svoje výhody aj nevýhody, ako napríklad:

⊕ rýchlosť vytvárania testov

⊕ nižšie nároky na zaškolenie testerov (ľudí ktorí vytvárajú testovacie šablóny)

⊕ nižšie nároky na samotných ľudí pri prijímaní na pozíciu

⊖ rýchlosť vytvárania testov závisí na možnostiach daného softvéru

⊖ robustnosť testov a ich odolnosť voči zmenám závisí na testovacom softvéri, ktorý vytvára prepojenia medzi klikom testera na element a medzi výstupom v podobe šablóny

Najpoužívanejšie dostupné nástroje:

- Selenium Webdriver: nástroj považovaný za nepísaný štandard v obore automatizovaného testovania grafického rozhrania webových aplikácií. Takmer deväť z desiatich testerov používa pri svojej práci práve Selenium podľa prieskumu z Mája 2018[1][1]. Tento framework[2] podporuje niekoľko programovacích jazykov, vrátane jazyku `Java`,

[1] https://d1h3p5fzmizjvp.cloudfront.net/wp-content/uploads/2018/06/05101901/The-Most-Striking-Problemns-in-Test-Automation-A-Survey.pdf

[2] Podporný softvér, ktorý obsahuje základnú funkcionalitu pre uľahčenie práce v špecifickom prostredí alebo pri špecifickom riešení problému. [26]

v ktorom je tento framework napísaný, alebo iné ako napríklad `C#, Perl, Python, JavaScript, Ruby` alebo `PHP`. Tento framework je voľne dostupný k používaniu a podporuje najpoužívanejšie platformy operačných systémov ako napríklad `Microsoft Windows, GNU/Linux` a `Apple macOS`. Selenium primárne neobsahuje grafické užívateľské rozhranie (GUI) a je potrebná znalosť napríklad jedného z vyššie uvedených programovacích jazykov.

- Katalon Studio: na rozdiel od Selenia, tento nástroj podporuje testovanie API, webového rozhrania a taktiež mobilných aplikácií. Katalon studio podporuje nahrávanie sedenia (angl. *session*) a taktiež obsahuje mnoho nástrojov na uľahčenie a urýchlenie vytvárania testov [1]. Katalon Studio obsahuje komplexnú množinu rozšírení a súčastí, ktoré spája do jednotného grafického rozhrania v podobe IDE[3]. Aj keď tester nemusí poznať programovací jazyk, je tu potrebná skúsenosť s daným vývojovým prostredím. Tento nástroj môže byť veľmi užitočný a je možné s ním vytvoriť veľké množstvo testov v relatívne krátkom časovom období[4]. Pri nedostatku skúseností sa v ňom človek dokáže aj stratiť, nakoľko je to komplexné riešenie.

- Cypress.io: Ďalším z nástrojov pre automatizované testovanie je *Cypress.io*. Tento nástroj funguje na podobnom princípe ako Selenium, avšak nemá takú podporu programovacích jazykov. Je to nástroj, ktorý využíva jazyk JavaScript a beží priamo v prehliadači narozdiel od Selenia [5]. Možnosť použitia (oficiálne) len jedného programovacieho jazyka so sebou prináša výhody aj nevýhody. Výhodou je jednotnosť komunity a oveľa vyššia šanca, že človek nájde alebo sa stretne s problémom, ktorý aktuálne rieši, vďaka tomu, že už niekto pred ním tento problém riešil alebo dokonca vyriešil. Nevýhodou je, že človek neznalý jazyka Javascript je nútený sa tento jazyk naučiť, čo značne spomaľuje mieru vývoja na počiatku adaptácie nového nástroja.

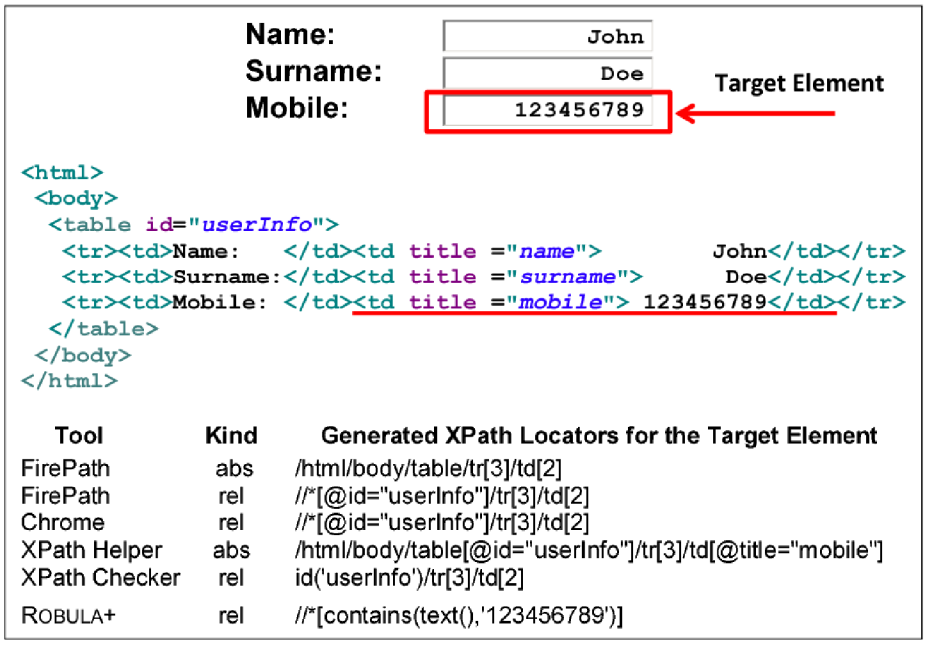
- Puppeteer: Posledným z tejto množiny dostupných a používaných nástrojov, je relatívne mladý nástroj od firmy Google [8]. Tento nástroj pracuje na podobnom princípe ako Cypress.io, taktiež využíva jazyk Javascript, avšak jeho proces nebeží natívne v prehliadači ale beží v samostatnom procese. Aj napriek tomuto faktu Puppeteer ponúka vynikajúcu integráciu s prehliadačom Google Chrome a tam aj končí. Puppeteer totiž podporuje iba prehliadač Google Chrome v plnom rozsahu. Ďalším prehliadačom je Mozilla Firefox, ktorá má základnú podporu, avšak niektoré funkcie a vymoženosti postráda. Puppeteer je vhodným kandidátom, ak niekto testuje aplikácie len v prehliadači Chrome.

3.3 Stavba robustnejších lokátorov XPath

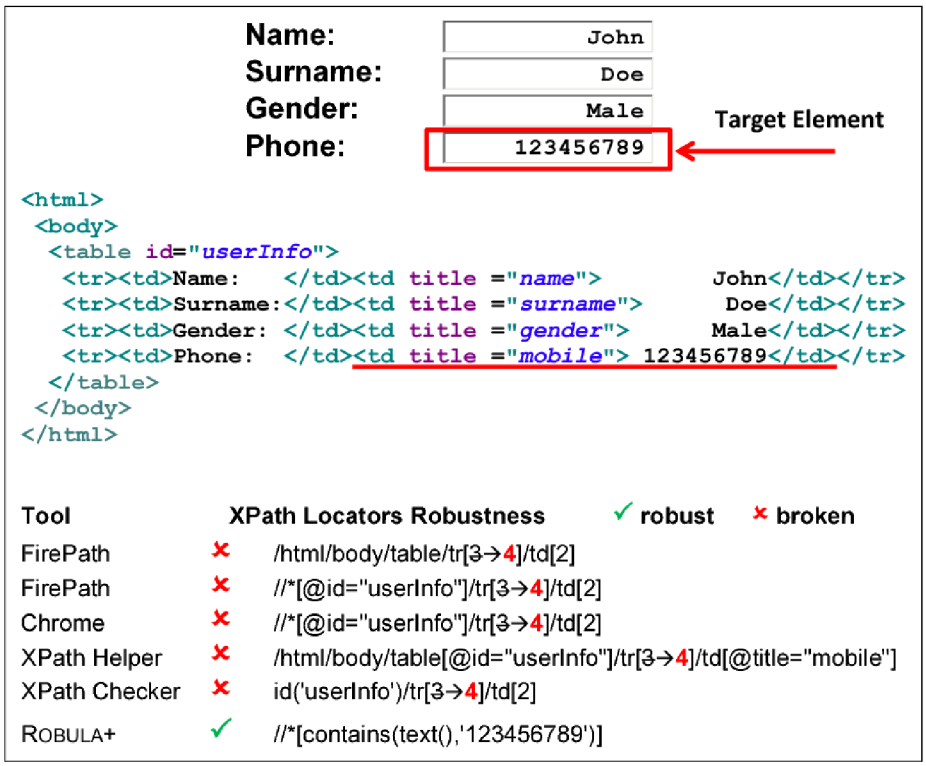
Autori algoritmu *ROBULA+* tvrdia zlepšenie robustnosti oproti lokátoru s absolútnou cestou k elementu o *90%* a voči lokátorom vygenerovaným iným algoritmom pre tvorbu relatívnych XPath lokátorov (algoritmus technológie Selenium WebDriver, viac o Seleniu v 3.2.2) o 63%. Na obrázkoch 3.1 a 3.2 je možné názorne vidieť fragilitu a zložitosť vytvárania XPath lokátorov.

[3]IDE – integrated development enviroment - integrované vývojové prostredie, je množina základných nástrojov potrebných pre prácu a testovanie softvéru. Vyznačuje sa plnohodnotným grafickým rozhraním a taktiež ponúka možnosť grafickej reprezentácie dát. [22]

[4]https://docs.katalon.com/katalon-studio/videos/create_basic_automation_test_case_katalon_studio.html



Obr. 3.1: Príklad XPath lokátorov na HTML stránke vytvorenými rôznymi algoritmami. Zdroj:[13]



Obr. 3.2: Príklad XPath lokátorov po zmene stránky. Zdroj:[13]

Algoritmus využíva niekoľko krokov, ktorými zabezpečí, že výsledný lokátor bude v konečnom dôsledku naozaj odolný voči drobným a častokrát aj väčším či rozsiahlym zmenám webovej aplikácie.

3.3.1 Špecializácia lokátorov

Jedným z hlavných bodov vytvárania robustných lokátorov je špecializácia lokátorov. V tomto bode je na daný lokátor uplatňovaných niekoľko operácií, v správnom poradí. V tomto bode si opíšeme jednotlivé operácie algoritmu, ktorých použitie neskôr opíšeme v kapitole 5. Zoznam operácií algoritmu ROBULA+:

`xp` – XPath výraz potrebný na špecializáciu, napríklad `//td`

`N` – dĺžka výrazu XPath `xp`, napríklad `//td` ⇒ `N=1`, `//table/tr/td` ⇒ `N=3`

`L` – zoznam nasledovníkov skúmaného elementu `e` v konkrétnom strome DOM (napríklad webová stránka), počnúc a vrátane e
– napríklad, nech `L` je zoznam nasledovníkov elementu e
v strome DOM `[td,tr,table,body,html]` zavolaním `L.get(2)` dostaneme element `tr`

- transfConvertStar
  - predpoklad: *výraz XPath* ***xp*** *začína* ***//****
  - činnosť: : nahradiť počiatočnú * názvom značky skúmaného DOM elementu L.get(N)
  - príklad: `//*/td a L.get(2).getTagName() = tr` **výstup**: `//tr/td`
- transfAddID
  - predpoklad: N-tá úroveň výrazu neobsahuje takýto typ predikátu
  - činnosť: pridať predikát založený na atribúte id (ak je dostupný) elementu DOM L.get(N) na vyššiu úroveň xp
  - príklad: `xp = //td and L.get(1).getID() = 'name'` **výstup**: `//td[@id='name']`
- transfAddText
  - predpoklad: : N–tá úroveň výrazu xp neobsahuje žiaden predikát s textom alebo žiaden predikát s pozíciou
  - činnosť: pridať predikát obsahujúci text (ak je to možné) v DOM elemente L.get(N) na vyššiu úroveň výrazu xp
  - príklad: `xp = //td and L.get(1).getText() = 'John'`
    výstup: `//td[contains(text(),'John')]`
- transfAddAttribute
  - predpoklad: N–tá úroveň výrazu xp neobsahuje žiaden typ predikátu
  - činnosť: pre každý dostupný pár hodnoty typu `atribút -- hodnota` elementu DOM L.get(N), vygeneruj možný lokátor pridaním predikátu založenom na danej hodnote pre vyššiu úroveň výrazu xp

- príklad: `xp = //tr/td` a
  `L.get(2).getAttributes() = {name='data', class='table-row'}`
  výstup: `//tr[@name='data']/td, and //tr[@class='table-row']/td`

- transfAddAttributeSet
  - predpoklad: N–tá úroveň výrazu xp neobsahuje žiaden typ predikátu
  - činnosť: pre každý element (s mohutnosťou > 1) potenčnej množiny vygenerovanej z množiny všetkých párov hodnôt typu `atribút -- hodnota` elementu DOM `L.get(N)`, vygeneruj možný lokátor pridaním predikátu založenom na danej hodnote pre vyššiu úroveň výrazu xp
  - príklad: `xp = //tr/td` a
    `L.get(2).getAttributes() = {name='data', class='table-row'}`
    výstup: `//tr[@name='data' and @class='table-row']/td`

- transfAddPosition
  - predpoklad: N–tá úroveň výrazu xp neobsahuje žiaden typ predikátu
  - činnosť: pridaj pozíciu elementu `L.get(N)` pre vyššiu úroveň výrazu xp
  - príklad: `xp = //tr/td and L.get(2).getPosition() = {if tag-name=2, if '*'=3}`[5] **výstup**: `//tr[2]/td`

- transfAddLevel
  - predpoklad: `N < L.lenght()`
  - činnosť: pridaj `//*` na vrchol elementu xp
  - príklad: `xp = //tr/td` **výstup**: `//*/tr/td`

Ďalšou z vecí, ktorá prispieva k špecializácii lokátorov generovaných pomocou algoritmu ROBULA+ je metóda prioritizácie a vyňatia atribútov. Algoritmus ROBULA+ obsahuje podporu pre definovanie atribútov v elementoch, ktoré nie sú žiadúce pre vytváranie XPath lokátorov. Z teoretického hľadiska sa jedná o zoznam názvov atribútov, ktoré majú byť prioritizované a v ideálnom prípade použité len tie. A o druhý zoznam atribútov, ktoré sú zakázané a nemali by v žiadnom prípade byť použité k vytváraniu lokátorov. Jedným z najprioritnejších atribútov môže byť napríklad atribút `id`, ktorý špecifikuje unikátny výskyt elementu v HTML kóde. Ako príklad pre atribút vyňatý z algoritmu je možné uviesť atribút `width`, ktorý môže byť pri každom otvorení stránky iný alebo sa môže meniť dynamicky podľa šírky okna alebo šírky elementu.

[5] transformácia ráta s tým, že pozícia elementu `L.get(N)` ktorá je pridaná v potencionálnom elemente XPath **xp**, sa môže zmeniť ak takýto element používa * alebo aktuálne meno prvku. Toto môže nastať, ak element má predchádzajúcich súrodencov iných typov.

Kapitola 4

Zhrnutie existujúcich riešení a návrh vlastného riešenia

V tejto kapitole sú zhrnuté vlastnosti a možnosti využiteľnosti kandidátov pre riešenie problému, ktorému sa táto práca venuje. Súčasťou tejto kapitoly je taktiež návrh vlastného riešenia, vychádzajúci z poznatkov získaných štúdiom existujúcich riešení.

4.1 Existujúce riešenia

V najbližších sekciách sa detailnejšie pozrieme na existujúce riešenia a nástroje, ktoré v istej miere spĺňajú podmienky pre použitie v tejto problematike. V týchto existujúcich riešeniach bola braná inšpirácia pri návrhu riešenia tejto práce.

4.1.1 Diplomová práca – Algoritmy pro rozpoznávání pojmenovaných entit

V tejto diplomovej práci [29] sa jej autor zaoberal podobným problémom aký je riešený v tejto bakalárskej práci. Autor sa pokúsil extrahovať informácie z textu emailových správ pomocou jednotlivých metód pre rozpoznávanie pomenovaných entít a pomocou modulu postavenom na základe grafového modulu CRF a porovnanie ich úspešnosti na riešení daného problému.

NER

NER — Named Entity Recognition — rozpoznávanie pomenovaných entít je jedna z metód používaných pri spracovávaní prirodzeného jazyka, medzinárodne známe aj ako NLP – Natural language processing, bližšie opísané v kapitole 2.1.1.

Cieľom NER je opísať všetky slovné výskyty *pomenovaných entít* v texte – osoby, spoločnosti, časové údaje a podobne. Viz 2.1.2.

Operácie s textom

V tejto práci autor vykonáva niekoľko potrebných operácií s testovacími aj trénovacími dátami. Pred prvým spustením je potrebné dáta predspracovať. Autor využíva regulárne výrazy na predspracovanie textu, v ktorom označí jednotlivé entity.
Následne dáta prevedie tokenizérom a teda rozdelí súvislý text na menšie časti alebo celky,

nazvané tokeny. Tieto tokeny slúžia ako vstup pre ďalšie moduly a časti výsledného analyzátora.
Jednými zo vstupov sú už vyššie spomenuté nástroje, prvý je implementácia neurónovej siete a druhý je postavený na základe grafového modelu CRF.
Tieto nástroje nebudeme bližšie špecifikovať, nakoľko sa to netýka zamerania tejto bakalárskej práce.

Výsledky spracovania

Po rôznych operáciách s textom, rozdelením na menšie časti a následne klasifikovaním neurónovou sieťou trénovanou na testovacích dátach, autor dosiahol úspešnosti 51.9%. V praxi to znamená, že takmer polovicu všetkých prichádzajúcich emailových správ, bol schopný úspešne zaradiť do kategórie a úspešne extrahovať informácie obsiahnuté v danej správe. Toto číslo je v celku prijateľné, avšak s takouto úspešnosťou nemožno hovoriť o úplnej spoľahlivosti.

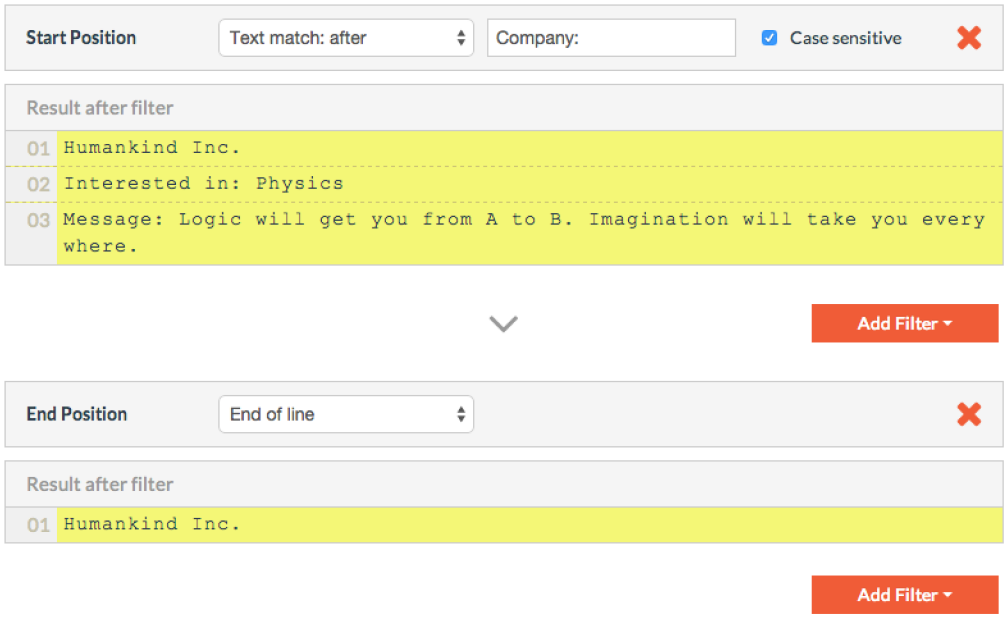
4.1.2 Mailparser.io

Jedným z možných riešení nášho problému by bolo využiť produkt spoločnosti *Mailparser.io*[1].
Hlavnou funkciou produktu spoločnosti Mailparser.io, ako už z názvu môže vyplývať — Mail – Parser, *[Pošta – Analyzátor]* — je analyzátor, pomocou ktorého môžete automaticky extrahovať potrebné dáta z jednotlivých emailových správ. Tento analyzátor má už od základov podporu pre niekoľko realitných kancelárií a donáškových služieb. Okrem samotného trénovania služba ponúka aj možnosti integrácie s niekoľkými, dnes používanými službami. Napríklad je tu možnosť exportovania dát do formátu `CSV`, `JSON` alebo `XML` a taktiež v prípade záujmu možnosť odoslať dáta na vzdialený server. Služba taktiež podporuje emailové správy s prílohami a analýzu týchto príloh, nakoľko prílohy sú bežnou súčasťou emailových správ. Je tu avšak nutnosť interakcie od užívateľa, ktorý je nútený pre správne fungovanie tohto systému, vytvoriť šablónu pre každý druh emailovej správy.

Tréning analyzátora

V prípade nášho problému, by sa jednalo o manuálny *„tréning“* pre každý email od každej leteckej spoločnosti pomocou tohoto analyzátor. To zahŕňa definíciu pravidiel v predpripravenom grafickom rozhraní, čo prináša ďalšie limitácie v podobe použitia len obmedzeného preddefinovaného počtu použiteľných operácií, pre každú hodnotu, ktorú by sme chceli extrahovať z daného dokumentu. Na obrázku 4.1 je možné vidieť príklad aplikovania pravidla pre vzorovú hodnotu.

[1] https://mailparser.io/



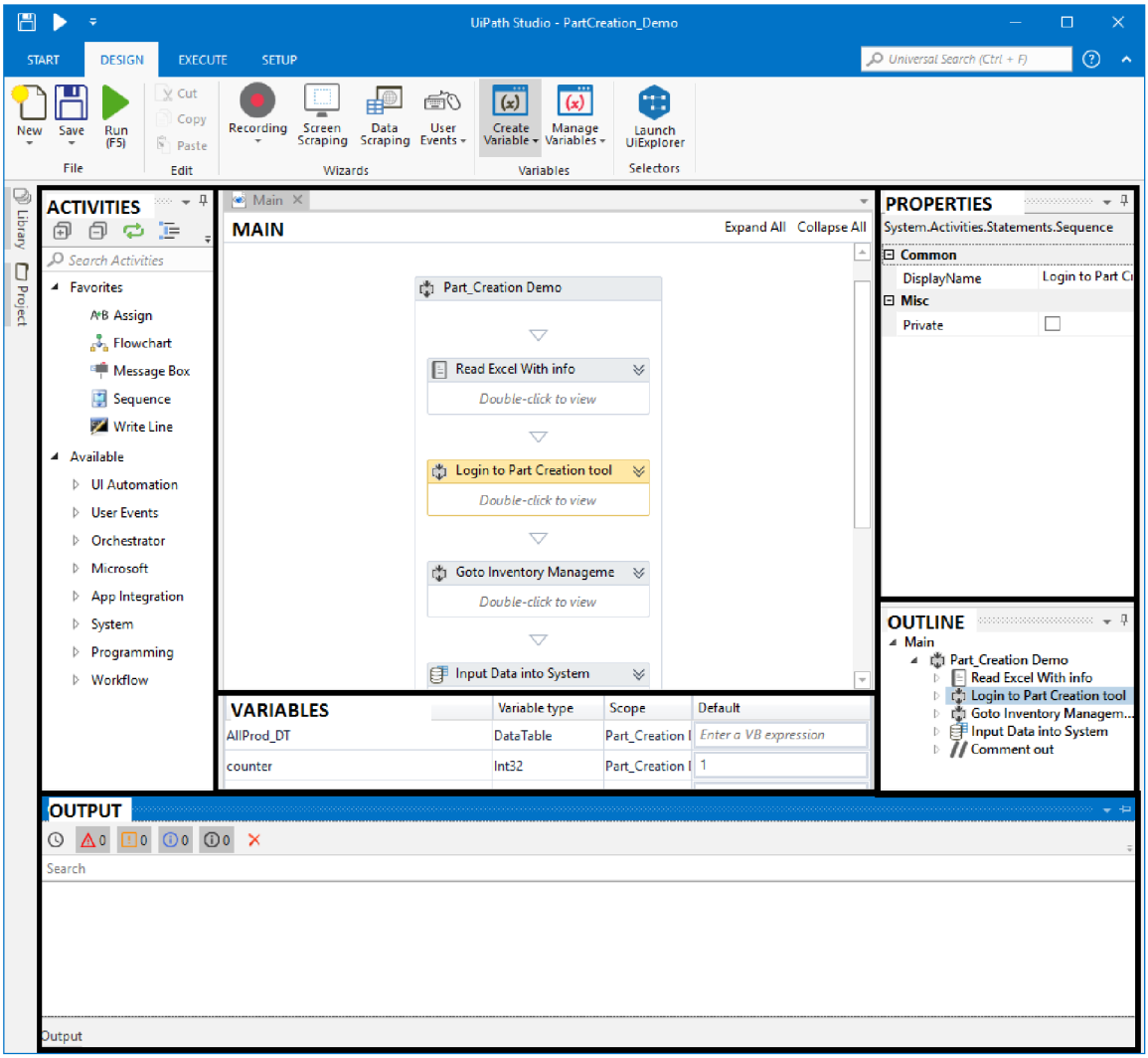
Obr. 4.1: Ukážka aplikovania textového filtra pre potrebu získania správnych dát z emailovej správy pomocou služby Mailparser.io.
Zdroj: https://mailparser.io/blog/email-parser-software/

Tieto pravidlá by ale platili len do momentu kedy by daná letecká spoločnosť ponechala formát emailovej správy v rovnakom tvare po dlhú dobu. Pri každej zmene formátu emailu by sme museli vytvoriť znova novú množinu pravidiel.

Táto skutočnosť predstavuje výraznú limitáciu v použiteľnosti tohoto nástroja. Na základe uvedeného môžeme konštatovať, že nespĺňa základnú myšlienku tejto práce, čím je automatizované vytváranie šablón pre spracovávanie emailových správ.

4.1.3 RPA – *Robotic Process Automation*

Robotic Process Automation v preklade automatizované vykonávanie procesu pomocou robota, je technológia umožňujúca vykonanie činnosti, ktorú by za normálnych okolností vykonával človek, avšak v tomto prípade je daný úkon, vykonávaný robotom [28]. Jedným z nástrojov podporujúci túto technológiu je *UiPath*. Táto spoločnosť poskytuje komplexné riešenia automatizácie procesov, ako napríklad grafické užívateľské rozhranie pre konfiguráciu a tréning automatizovaných procesov. Najbližšie ku tejto tématike má však samotná automatizácia procesov pomocou robota. Tento robot dokáže spracovať obraz o tom, ako vyzerá užívateľské rozhranie, ktoré ponúka možnosti užívateľovi pri obrazovke monitora. Robot taktiež dokáže manipulovať s objektmi ako reálny užívateľ, odosielať signály a požiadavky iným systémom alebo súčastiam systému, ale lepšie. Lepšie z dôvodu, že tento robot nepotrebuje spánok, nerobí chyby a jeho prevádzka stojí menej ako zamestnávanie reálneho človeka. To robí z tejto technológie výrazného favorita v rámci akéhokoľvek spracovania informácií, ktoré je nutné vykonávať manuálne a je repetetívne. Tento úkon je ale potreba natrénovať.



Obr. 4.2: UiPath Studio: grafické rozhranie systému určené na podporu vytvárania procesov automatizácie.
Zdroj: http://prorpa.com/index.php/2018/03/09/3-key-elements-of-uipath/

Tréning automatizovaného procesu

Tento proces by mohol uspieť, v prípade, že je potrebné spracovať veľké množstvo dokumentov, ako napríklad vyplnenie údajov z faktúr. Ak má väčšina faktúr rovnaký formát, faktúry odosiela dodávateľ, s ktorým máme uzatvorenú zmlúvu a väčšina faktúr teda vyzerá v celku rovnako, percento úspešnosti bude vysoké. Nakoľko automatizovanie tohoto príkladu, by zahŕňalo vytvorenie a natrénovanie automatizovaného procesu, pomocou manuálnych vstupov od užívateľa. Výsledkom toho by mohlo byť množstvo vyextrahovaných informácií z dokumentov v ktoromkoľvek štrukturovanom formáte. Systém dokonca umožňuje ich zadanie priamo do systému.

V našom prípade je tu však veľká prekážka. Aj napriek zmlúvnej dohode, letecké spoločnosti si sami určujú aký formát zvolia pre svoje emailové správy a kedy sa ho rozhodnú obnoviť.

4.2 Zhrnutie použiteľnosti existujúcich riešení

Z textu vyššie vyplýva, že každý zo spomenutých nástrojov, má rôzne možnosti a obmedzenia. Tieto obmedzenia sú ale natoľko závažné, že nám nedovoľujú žiadne z týchto riešení použiť. V prípade riešenia pomocou služby *Mailparser.io* je hlavným nedostatkom počiatočné a priebežné udržiavanie aktuálnosti šablóny pre prichádzajúce emailové správy.

Týmto riešením by problém v podobe podpory emailových správ od doposiaľ nepodporovaných leteckých spoločností a taktiež správ od podporovaných spoločností, no s upraveným alebo úplne novým formátom, zostal takmer nezmenený. Spomenutým riešením by sa neeliminovali žiadne problémy, naopak mohli by sa objaviť nové, doposiaľ neobjavené, nakoľko by sa jednalo o použitie novej služby.

V prípade riešenia pomocou *UiPath*, by bolo možné problém riešiť obdobným spôsobom, ako v prípade služby *Mailparser.io*, s tým rozdielom, že UiPath sa nešpecializuje iba na emailové správy. Nástroj podporuje taktiež možnosti konfigurácie a vytvorenia postupností pre získavanie dát z webových stránok (Web automation alebo Web scraping) alebo automatizáciu procesov v operačnom systéme, ako napríklad hromadné presúvanie súborov. Na obrázku 4.2, je možné vidieť užívateľské rozhranie určené na vytváranie definícií automatizácie procesov. V konkrétnom prípade je možné vidieť definíciu sekvencie príkazov pre načítanie hodnôt zo súboru programu Excel.

Všetky z vyššie uvedených riešení majú ale spoločnú vlastnosť a tou je miera interakcie s užívateľom voči miere spracovaných výsledkov. Od každého je potreba prísun ľudskej interakcie aj napriek faktu, že vyššie spomenuté riešenia sú práve tými, ktoré používajú umelú inteligenciu.

V prípade riešenia podľa autora diplomovej práce [29] je hlavným nedostatkom fakt, že NLP – *Natural Language Processing* pracuje s holým textom a teda by nebolo možné použiť žiadnu z technológií, ktorá sa dnes využíva v prípade štrukturovaných dokumentov ako napríklad XPath, CSS selektory, selektory atribútov. Tento fakt výrazne zužuje možnosti výberu technológie z oboru automatizácie. Ďalším podporným faktom je taktiež skutočnosť výslednej percentuálnej úspešnosti riešenia, ktorá dosiahla výšku 51.9%.

V tomto momente je už na zváženie čo bude figurovať ako lepšia investícia prostriedkov do budúcnosti, či vytváranie šablón na spracovanie novo-prichádzajúcich emailov manuálne, ako tomu je doposiaľ, alebo nutnosť zdokonaľovania danej aplikácie. V konečnom dôsledku výsledky spracovania nie sú presvedčivé natoľko aby spoločnosť Kiwi.com rozvíjala toto riešenie s ohľadom na náročnosť vývoja tohoto softvéru.

Hlavným nedostatkom budeme považovať časovú náročnosť a manuálnu prácu v prípade riešenia *Mailparser.io* a *UiPath* a nízku úspešnosť a náročnosť problematiky v prípade riešenia podľa autora diplomovej práce [29].

4.3 Návrh systému

Návrh riešenia tohoto problému vychádza z algoritmu *ROBULA+* definovanom v kapitole číslo 3. Algoritmus ROBULA+ pracuje s lokátormi XPath a z tohoto dôvodu je navrhnutý systém závislý na dokumentoch a ich štruktúre v jazyku HTML alebo XML. V sekcii 2.5.1 boli definované rozdiely medzi HTML a XML. Z uvedených vlastností vyplýva, že jediným vhodným jazykom teda zostáva HTML. Ako bolo definované v sekcii 2.4, HTML je jazyk štrukturovaný a preto je nelogické tento faktor nevyužiť. Táto práca pracuje s emailovými správami, v pôvodnej verzii v akej ich letecká spoločnosť odoslala, bez žiadnych ďalších úprav. Tento fakt uľahčuje prácu so vstupnými dátami a taktiež zvyšuje mieru schopnosti

rozšírenia a podpory nových leteckých spoločností v budúcnosti. Napriek faktom, výhodám a použiteľnosti metód spracovania prirodzeného textu, ako boli definované v sekcii 2.1.1, v tejto práci využívané nebudú. Práca sa zameriava na fakt, že emailové správy, ktoré sú k dispozícií od firmy Kiwi.com s.r.o., sú v štrukturovanom formáte. Bolo by nadnesene povedané, nerentabilné tento formát ignorovať a preto budú spracovávané v ich originálnom znení.

4.4 Požiadavky na systém

Náplňou systému má byť pomoc pri rozširovaní podpory súčasnej základne spracovávania emailových správ spoločnosti Kiwi.com s.r.o.. To možno docieliť pomocou automatizácie procesov, ktoré v dnešnej dobe je potrebné realizovať ručne. Ide o procesy vytvárania šablón pre spracovávanie prichádzajúcich emailov. S rastúcim počtov leteckých spoločností na trhu a taktiež s rastúcim počtom podporovaných leteckých spoločností firmou Kiwi.com s.r.o. takmer s istotou pribudnú požiadavky na vytváranie podpory pre nové letecké spoločnosti. Ako už bolo spomenuté, to sa dá čiastočne eliminovať pomocou automatizácie tohoto procesu.

Systém by mal byť vyvíjaný ako webová aplikácia z dôvodu kompatibility s doterajšími a budúcimi aplikáciami, s ktorými má komunikovať. Vývoj tohoto systému s architektúrou webovej aplikácie a rozhraním REST, nám zaručí kompatibilitu do budúcnosti, nakoľko webové aplikácie musia spĺňať jediné technické požiadavky a to tie, ktoré sú definované v protokole HTTP.

Systém obsahuje niekoľko závislostí z princípu procesu.

Týmito závislosťami sú:

- závislosť na dátach o doteraz vytvorených rezerváciách
- závislosť na množine emailových správ, z ktorých budú dáta potrebné k vytváraniu šablón extrahované
- potreba emailových správ v štrukturovanom formáte jazyka HTML, alebo jeho derivátu

4.5 Vstupné dáta

Vstupné dáta systému sú špecifikované ako emailové správy od leteckých spoločností. V tejto práci budeme cielene spracovávať iba správy, ktoré letecká spoločnosť odoslala ako potvrdenie o vytvorení rezervácie u danej spoločnosti alebo jednej z jej partnerských spoločností. Tieto emailové správy sú vo formáte HTML, na ktorý sa dá uplatniť hlavne technológia XPath, o XPath viac v 2.6. Taktiež je možné pracovať s jednotlivými časťami emailových správ na objektovej úrovni, čo výrazne uľahčuje prácu a náročnosť vďaka technológii DOM. Viac o DOM v 2.7.

Samotné emailové správy nám ale nepostačujú ako zbierka vstupných dát. K emailovým správam je potrebné pripojiť aj údaje, ktoré budú v daných správach vyhľadávané. Tieto údaje sú zabezpečené pomocou internej aplikácie firmy Kiwi.com, ktorá je abstrakciou nad dátami uloženými v relačnej databáze a je možné k nim pomocou tejto abstrakcie pristupovať. Tieto dáta sú následne upravené do podoby, kedy obsahujú pre našu aplikáciu, len nevyhnutné dáta. Ako príklad možno uviesť číslo rezervácie, meno pasažierov, číslo

letu alebo kód leteckej spoločnosti. Najpodstatnejším údajom je ale číslo rezervácie, nakoľko pomocou tohoto čísla je možné unikátne špecifikovať rezerváciu u konkrétnej leteckej spoločnosti.

Tieto údaje sú následne doplnené o ďalšie údaje, ktoré sú v explicitne určenom formáte. Ako príklad je možné uviesť napríklad údaj `status`, u ktorého sú jasne stanovené hodnoty. Napríklad `cancelled` alebo `confirmed`.

4.6 Spracovanie emailových správ

Ako už bolo spomenuté v sekcii 4.5 a na začiatku tejto kapitoly, je vhodné ponechať dáta v štrukturovanom formáte, aby sa dali ďalej jednoduchšie spracovať. Na túto úlohu bolo zvolených niekoľko knižníc, z čoho najväčšie zastúpenie má knižnica Beautiful Soup 4.

4.6.1 Beautiful Soup

Beautiful Soup je knižnica napísaná v jazyku Python umožňujúca extrahovanie dát z webových stránok, organizáciu dát pomocou objektovo orientovanej technológie podobnej technológii DOM a taktiež reprezentáciu dát v dátových štruktúrach [18]. Knižnica verzie 4 a teda Beautiful Soup 4, podporuje všetky verzie programovacieho jazyka Python od verzie 2.7 a vyššej. Existuje verzia knižnice pod názvom Beautiful Soup 3, ale tá nie je naďalej vyvíjaná a pre nové aplikácie je odporúčané použiť novšiu verziu aplikácie. Tým, že je knižnica napísaná výhradne v jazyku Python so sebou prináša isté úskalia v podobe dlhšej doby behu programu, avšak na druhej strane prináša takzvaný „Pythonic“ spôsob manipulácie s dátami, čo je v komunite jazyka Python takmer nepísané pravidlo týkajúce sa formy a štýlu písania kódu. Ponúka jednoduché rozhranie pre spracovanie vstupného HTML kódu do podoby stromu, čo prináša jednoduchší spôsob úpravy a traverzovania skrz uzly stromu.

4.7 Stavba webovej aplikácie

Byť webovou aplikáciou, znamená dokázať zvládnuť niekoľko požiadavkov v jeden moment. Ani jedna požiadavka zo strany klienta sa nesmie stratiť niekde v pamäti a byť zabudnutá. O to aby k niečomu takémuto nedošlo sa stará asynchrónny plánovač.

4.8 Asynchrónny plánovač

Tento nástroj sa stará o uloženie prichádzajúcich úloh do fronty, odkiaľ si ich „vezme“ a spracuje takzvaný *worker*. Workerom môžeme nazvať proces, ktorý je spustený pre potrebu vykonania nejakého druhu úkonu, tento úkon vykoná a výsledok uloží na miesto, kde si to tvorca tejto úlohy žiada. Pre tento komponent bude v aplikácii použitý nástroj s názvom *Celery*.

4.8.1 Celery

Celery je nástroj, ktorý nám umožňuje spracovávanie úloh asynchrónne, čo znamená, že nám nezáleží na tom v akom poradí a kedy sa daná operácia vykoná, ale zaujíma nás výsledok tejto operácie. Či už je to odoslanie emailu, alebo je to výpočet určitého desatinného miesta iracionálneho čísla, tento nástroj zabezpečí, že tieto operácie budú vykonané [24]. Využíva na to systém frontu, kde sa ukladajú úlohy, ktoré majú byť vykonané, ku ktorým

pristupujú už vyššie spomenutý *workery* (procesy) a tieto úlohy vykonávajú. Na predávanie informácií o fronte a vykonávaní uskladňovania týchto operácií do a z frontu slúži takzvaný sprostredkovateľ alebo *broker*. Celery má podporu pre rôzne technológie na mieste brokera ako napríklad *RabbitMQ*, *Redis*, *Amazon SQS* a *Zookeeper*. V našej aplikácii bude použitá pre tento účel technológia **Redis**.

4.8.2 Redis

Redis je úložisko vyvíjané spôsobom *open-source*, teda softvér s voľne dostupným zdrojovým kódom [12]. Redis je úložiskom, ktoré je takzvané *in-memory*. To znamená, že všetky dáta, s ktorými pracuje sa snaží držať v pamäti RAM, aby boli rýchlo prístupné. Dáta, ktoré už nemôžu byť uložené v pamäti RAM, kvôli limitu operačného systému pre každú aplikáciu, alebo kvôli nedostatku pamäti RAM na zariadení, nie sú prijaté na uloženie do Redisu. To môže byť v niektorých prípadoch zradné, no Redis nebol vyvinutý na ukladanie veľkých súborov. Redis je vyvinutý v jazyku C, čo zaisťuje veľmi rýchle spracovanie dát. Redis očakáva, že do ňho budeme vkladať dáta, ktoré chceme mať okamžite prístupné, dáta ktoré chceme čítať mnohokrát po sebe, alebo dáta, ktoré majú krátku trvanlivosť a často ich budeme prepisovať. Dá sa to teda nazvať softvérovou realizáciou pre rýchlu vyrovnávaciu pamäť.

4.9 Flask

„Microframework“ je oficiálny názov akým autori prezentujú túto knižnicu s podporou vytvárania webových aplikácií [21]. „*Micro*“ v tomto slovíčku neznamená, že celá aplikácia sa musí vojsť do jedného súboru. Aj keď tomu naozaj tak je, znamená to, že jadro tejto knižnice je udržiavané tak, aby nikomu neurčovalo, akú databázovú technológiu má použiť, alebo akú štruktúru šablón a usporiadanie priečinkov musí dodržiavať. Táto knižnica bola postavená s dôrazom na jednoduchosť a voľnosť použitia vo všetkých smeroch.

Pomocou tejto technológie bude realizovaný vývoj REST API danej aplikácie, nakoľko táto technológia ponúka excelentný pomer miery práce voči miere funkčnosti, a teda dokážeme v relatívne krátkom čase naprogramovať relatívne rozsiahlu štruktúru aplikačného rozhrania.

4.10 Abstraktná databázová vrstva

Ako vrstva umožňujúca prístup k dátam uloženým v databáze, bude v tejto práci použitá interná aplikácia firmy *Kiwi.com*, ktorá úplne zapuzdruje akúkoľvek prácu s dátami o vytvorených rezerváciách uložených v databáze, k čomu by bolo potrebné použitie jazyka *SQL*[2]. Aplikácia je webová a teda komunikácia s ňou prebieha pomocou rozhrania REST API.

[2]SQL – *Structured Query Language*, jazyk ktorý je používaný na komunikáciu s databázou. Je štandardom pre komunikáciu s relačnými databázami a ich spravovaním. Viac o SQL v [25].

Kapitola 5

Implementácia a overenie funkčnosti

Pre túto prácu bol zvolený jazyk Python vo verzii 3.6, nakoľko tento jazyk má výraznú podporu čo sa týka oblasti pre prácu s dátami a taktiež pre prácu s webovými aplikáciami.

V tejto kapitole je opísaný postup vývoja systému do štádia, v ktorom je možné previesť overenie funkčnosti pomocou testovacích skriptov. Sú pri tom využité poznatky z teoretického hľadiska definované v kapitole číslo 2. Systém využíva operácie algoritmu ROBULA+ ako sú definované v sekcii 3.3.1.

5.1 Spracovanie vstupných dát

Formát vstupných dát bol opísaný v sekcii 4.5. V tejto sekcii je opísaný postup spracovania vstupných dát, ktoré sú následne spracované do štruktúry, ktorá obsahuje všetky potrebné informácie k správnemu spracovaniu pomocou algoritmu *ROBULA+* a k všeobecnému fungovaniu systému.

5.1.1 Spracovanie požiadavky

O spracovanie prichádzajúceho požiadavku sa stará rozhranie typu *REST API*, ktoré spracuje prichádzajúcu požiadavku a vytvorí úlohu vo fronte úloh. Túto časť zabezpečuje technológia *Celery* spoločne s technológiou *Redis*, ktoré boli dôslednejšie opísané v sekcii 4.8. Toto rozhranie ponúka niekoľko možných operácií, ktoré je možné vyvolať.

- spracovať doposiaľ nespracovanú emailovú správu pomocou predvytvorenej šablóny
- zadať požiadavku pre vytvorenie šablóny
- zobraziť si všetky existujúce šablóny
- vymazať šablónu pre zadanú leteckú spoločnosť

Špecifikácia REST API navrhnutého systému		
časť URI	HTTP metóda	možnosti odpovede
`/email`	GET – získať výsledky spracovania	200 OK
		204 No content
	POST – vytvoriť požiadavku pre spracovanie	201 Created
		400 Bad Request
`/template`	GET – získať vytvorenú šablónu	200 OK
		204 No content
	POST – uložiť vytvorenú šablónu	201 Created
		400 Bad Request
	DELETE – požiadať o invalidáciu šablóny	202 Accepted
		406 Not Acceptable

Tabuľka 5.1: Schéma rozhrania REST API s popisom metód a stavových kódov odpovedí

5.1.2 Získanie emailových správ

V prípade požiadavky na spracovanie emailu, ktorý ešte nebol spracovaný a nie je jasné k akej rezervácii patrí, je daná nasledujúca postupnosť úkonov. V prvom rade, je špecifikovaný email vyhľadaný a získaný zo servera. Pod pojmom email je myslené vyhľadanie požadovaného riadku v tabuľke databázy s dopredu špecifikovaným identifikátorom alebo primárnym kľúčom, pomocou jednoduchého príkazu *SELECT*. V tejto databáze sa nachádzajú základné údaje o doručenej emailovej správe, ako napríklad z akej adresy bola emailová správa doručená, kto bol jej odosielateľom, na akú adresu bola doručená, kedy bola doručená a čo je predmetom danej správy. Po nájdení tohoto záznamu je vyhľadané telo tohoto emailu, potrebné pre ďalšie spracovanie. Tieto údaje sú uložené oddelene z dôvodov zachovania výkonu databázy.

5.1.3 Získanie potrebných dát o rezerváciách

Po dokončení získavania tela emailových správ je potrebné získať dáta, ktoré budú určovať aké údaje budeme v danej správe vyhľadávať. Tieto údaje sa skladajú z konkrétnych údajov o potvrdených rezerváciách. Najpodstatnejším údajom v týchto dátach je takzvané *číslo rezervácie*, anglicky nazývané *reservation number* alebo v skratke **PNR** – *passenger name record*[1] Tieto dáta sú uložené v databáze leteckej spoločnosti a teda by bolo nutné v prípade nutnosti získať nejaké z dát o rezervácií, dotazovať samotnú leteckú spoločnosť pri každom prístupe k dátam. Z tohoto dôvodu sú v databáze rezervácií uložené aj iné dáta ako meno pasažierov, ktorích počet môže byť väčší ako jedna v jednej rezervácií, údaje o letoch, ktoré sú spojené s danou rezerváciou a taktiež nutné informácie o doplnkových službách (angl. *ancillaries*) zviazaných s rezerváciou. To môže byť napríklad počet kusov batožiny, ktorú si zákazník môže vziať na palubu lietadla alebo číslo priradenej sedačky.

[1]PNR záznam je nosičom dát a je to medzinárodný formát, ktorý špecifikuje dáta o vytvorenej rezervácii u leteckej spoločnosti. Byť medzinárodným formátom ale ešte neznamená taktiež zastrešovanie správy týchto dát a teda je nutné aby si každá letecká spoločnosť uchovávala vo svojom systéme až do doby kedy si pasažier vyžiada takzvaný *check-in* a teda agent leteckej spoločnosti (túto časť dnes vo väčšine zastupuje počítač) priradí miesto v lietadle, s konkrétnym číslom sedadla a taktiež preverí či daný pasažier má povolené vstúpiť na palubu tohoto lietadla [10].

5.2 Systém vyhľadávania informácií v emailových správach

V momente kedy máme všetky potrebné informácie spojené s vytvorenou rezerváciou, je možné teda započať vyhľadávanie týchto informácií. Tento navrhnutý systém funguje na základe nájdenia daných hodnôt v celom texte emailovej správy vo formáte HTML vrátane zdrojového kódu. Algoritmus vyhľadávania informácií v emailových správach je opísaný v nasledujúcej časti.

5.2.1 Vyhľadávanie hodnôt v emailových správach

Algoritmus je založený na základe vytvárania štruktúry pomocou knižnice Beautiful Soup 4, ktorá dokáže spracovať HTML dokument do štruktúry podobnej štruktúre DOM, a teda je možné s ňou pracovať ako s objektom a využívať rôzne metódy a funkcie narozdiel od manipulácie s čistým textom. Pre optimalizáciu výsledkov bol zvolený prístup overenia či sa daná hodnota v texte naozaj nachádza. To je realizované pomocou jednoduchej metódy v jazyku Python bez použitia ďalších rozšírení. Následne ak vieme, že sa daná hodnota nachádza v dokumente, môžeme pokračovať so spracovávaním. Ďalším krokom je vytvoriť už spomínanú štruktúru pomocou knižnice Beautiful Soup 4 a danú nájdenú hodnotu lokalizovať v tejto štruktúre. Výsledkom tohoto úkonu je lokalizátor XPath, ktorý odkazuje na konkrétny element v štruktúre, avšak je v tvare s absolútnou cestou. To znamená, že cesta k danému elementu, obsahuje názov každého elementu, cez ktorý táto cesta prechádza. Príklad absolútneho XPath lokátora ako aj XPath lokátora s relatívnou cestou je možné nájsť v sekcii 2.6.

5.2.2 Vytváranie robustných lokátorov XPath

V momente kedy je email pretvorený do tvaru zoznamu obsahujúci množstvo lokátorov s absolútnou cestou pre každú nájdenú hodnotu v emailovej správe, je potrebné tieto lokátory premeniť do relatívneho tvaru, kedy každý z nich bude odkazovať len na jediný element v celom dokumente. Oporným bodom by mal byť fakt, že `id` v dokumente HTML ako aj v dokumente CSS by mal byť unikátny. Toto avšak nie je vždy pravda, nakoľko každú webovú stránku vytvára iný tím vývojárov, s inou skúsenosťou a znalosťou jazyka HTML a CSS. A teda niekedy môžeme na danej stránke objaviť 2 alebo viac elementov, ktoré majú rovnakú hodnotu v atribúte `id`, nanajvýš ak sa jedná o stránku, ktorej HTML zdrojový kód je generovaný dynamicky.

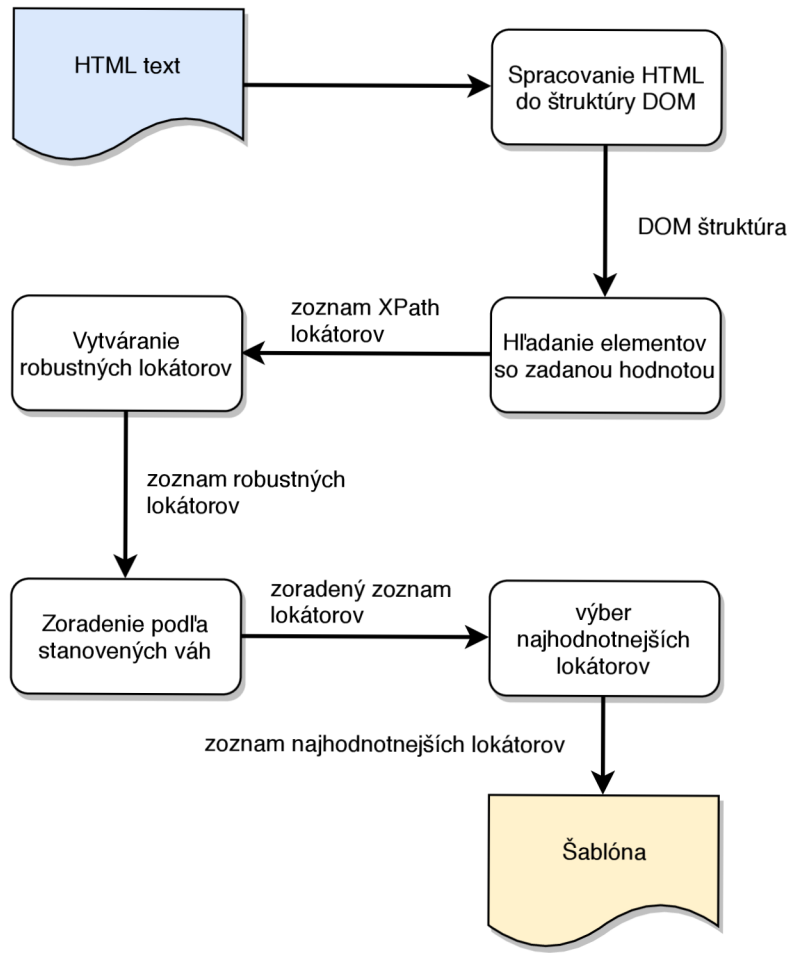
Túto funkciu zastrešuje algoritmus ROBULA+, ktorého návrh a špecifikáciu je možné nájsť v kapitole 3. Implementácia tohoto algoritmu je popísaná v nasledujúcej sekcii.

Na obrázku 5.1 je možné vidieť zjednodušený prehľad, akým je algoritmus ROBULA+ zakomponovaný do systému a postup pri vytváraní šablóny pre určitý druh emailových správ.

5.3 Trieda `RobulaPlus`

Táto trieda zastrešuje implementáciu algoritmu s rovnakým názvom. V tejto triede je implementovaná logika operácií definovaných v sekcii 3.3.1.

Táto trieda vykonáva priechod celou štruktúrou HTML súboru. Na vstupe očakáva lokálizátor XPath, ktorý analyzuje a v prípade ak splňuje všetky potrebné podmienky, tento prvok lokalizuje. Hlavnou podmienkou je odkazovanie sa iba na jediný prvok v štruktúre



Obr. 5.1: Princíp práce algoritmu ROBULA+ a pomocných častí, ktoré su potrebné pre vytvorenie šablóny

súboru. Následne sa bude pre daný prvok snažiť vytvoriť čo najrobustnejší lokalizátor XPath v relatívnom tvare.

Tento lokalizátor je formovaný postupne, pridávaním rôznych prvkov, ako napríklad názov elementu, atribútov alebo vyhľadávaním konkrétnej textovej hodnoty pomocou výrazu vyhľadávania vo výraze XPath.

Rôznym príkladom lokalizátora môže byť napríklad:

- `//span/strong` – element s názvom značky `<strong>`, ktorému predchádza element `<span>`
- `//*[@data-model="specific_model"]` – všetky elementy ktoré obsahujú atribút `data-model` a jeho hodnota je `specific_model`
- `//td[contains(@class, "selected")]` - element `<td>`, ktorý obsahuje hodnotu `selected` v atribúte triedy `class`

Pseudo-algoritmus algoritmu ROBULA+, ktorý je zjednodušeným obrazom o implementácii môžeme nájsť v Algoritme 1.

Algoritmus 1: ROBULA+

Input: $(xpath, tree)$
Output: *xpath_list*

```
1:  Element node = find_nodes(tree, xpath);
2:  List<XPath> xpList = ["//*"];
3:  while true do
4:      XPath xp = xpList.popFirst();
5:      List<XPath> tmp = List [];
6:      tmp.append(transfConvertStar(xp));
7:      tmp.append(transfAddID(xp));
8:      tmp.append(transfAddText(xp));
9:      tmp.append(transfAddAttribute(xp));
10:     tmp.append(transfAddAttributeSet(xp));
11:     tmp.append(transfAddPosition(xp));
12:     tmp.append(transfAddLevel(xp));
13:     for XPath x in tmp do
14:         if uniquely_eval(x, node, tree) then
15:             return x;
16:         else
17:             xpList.append(x);
18:         end if
19:     end for
20: end while
21: return xpath_list

22: List<XPath> find_nodes(Document tree, XPath xpath):
23:   return the element in tree selected by the XPath locator xpath

24: Boolean uniquely_eval(XPath x, Element e, Document tree):
25:   return TRUE if find_ondes(tree, xpath) returns only e
```

Ako je možné v Algoritme 1 vidieť, metódy sú teda aplikované postupne jedna po druhej, a po každej operácií je skúmaný výsledok danej operácie. V tejto práci bolo potreba

vynechať operáciu *transfAddText(xp)*, nakoľko táto operácia pridáva do lokátora časť s vyhľadávaným textom. Z tohoto dôvodu teda túto operáciu nemôžeme použiť, nakoľko by to vnieslo určitú nežiadúcu konkretizáciu do týchto lokátorov. Lokátory by sa teda stali závislými na konkrétnych hodnotách, ktoré obsahuje daný email, ako napríklad konkrétny čas odletu lietadla, konkrétne číslo rezervácie alebo na konkrétneho pasažiera. Ak by sme chceli využívať túto operáciu, existuje možnosť kedy by sme po aplikovaní tejto operácie, vyhľadávali či vyhľadávaná hodnota nie je súčasťou lokátora a v tom prípade by sme danú hodnotu nahradili za hodnotu generickú. Avšak táto možnosť tiež nebola vyhovujúca z dôvodu nutnosti úpravy tohoto lokátora pred samotným použitím.

Ďalšou odchýlkou od tohoto návrhu je aj fakt akým pracuje metóda `RobulaPlus().t_add_attribute(xpath, elem, use_whitelist)`.
V tomto prípade metóda zastrešuje operácie *transfAddID(xp)*, *transfAddAttribute(xp)* a *transfAddAttributeSet(xp)*. Vstupný atribút tejto metódy `use_whitelist` slúži ako prepínač, ktorý určuje či sa majú v danom momente používať zvýhodnené atribúty uložené v triednej premennej `attribute_whitelist` aby boli dostupné vo všetkých operáciách, ktoré by potrebovali tento zoznam atribútov využiť.

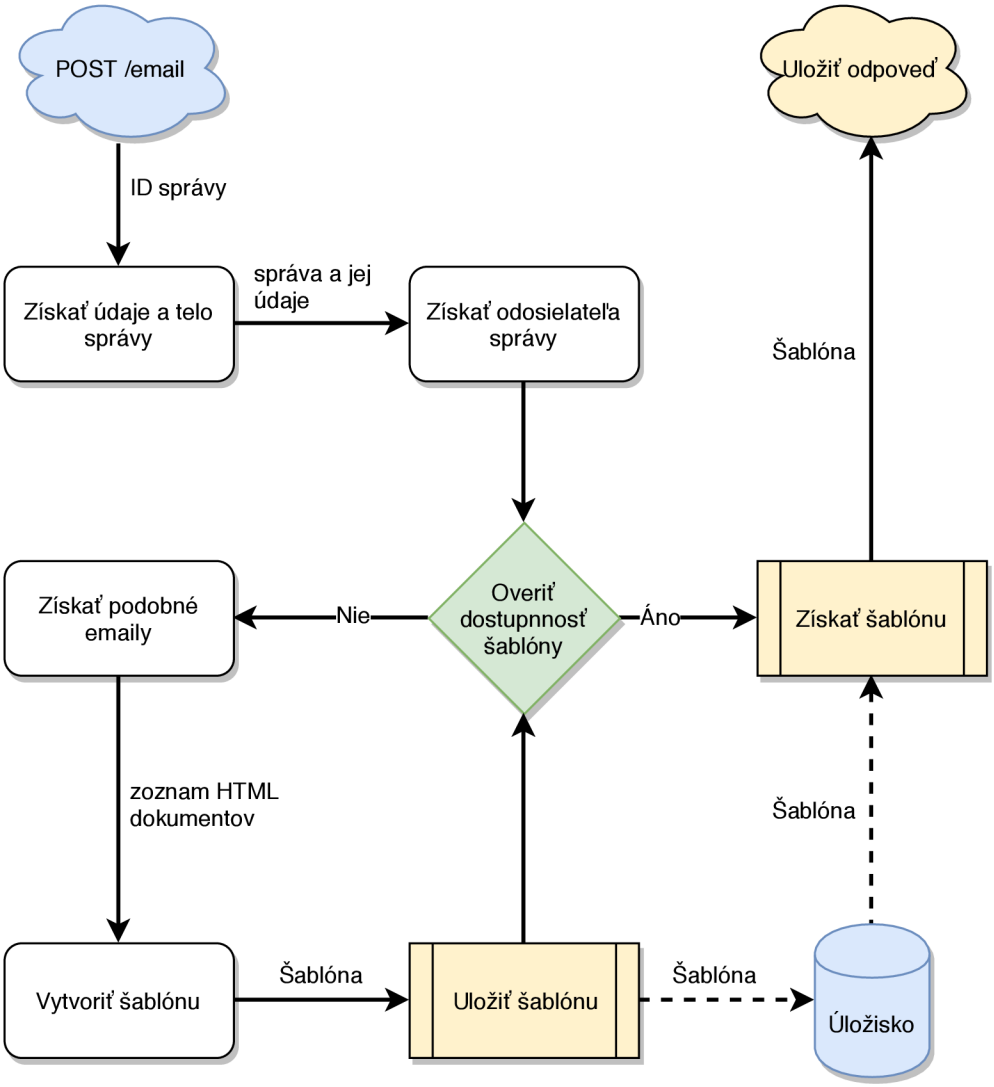
5.3.1 Triedne atribúty

Jedným z atribútov tejto triedy je už vyššie spomenutý zoznam zvýhodnených alebo žiadúcich názvov atribútov `attribute_whitelist`. Ďalším je `attribute_blacklist`, ktorý obsahuje elementy, ktorým sa chceme pri vytváraní lokátorov vyhnúť. Pre príklad je dobré v tomto prípade spomenúť atribúty `onclick`, `onload` alebo `width` a `height`, ktoré závisia na výrazne premenlivých elementoch stránky a teda šírka alebo výška elementu, ktorá môže byť generovaná podľa šírky zobrazovaného okna a z tohoto dôvodu je dobré sa im vyhnúť. Ďalším príkladom môžu byť taktiež atribúty `src`, `href` a `alt`, ktoré slúžia pre odkazovanie sa na externé alebo interné zdroje. Tieto odkazy sa môžu meniť podľa potreby stránky a teda taktiež je vhodné sa takýmto lokátorom vyhnúť.

5.3.2 Vytváranie šablón

Po aplikovaní všetkých potrebných operácií nutných pre spracovanie emailových správ, čoho výsledkom sú dáta vo formáte `kľúč:hodnota`, je nutné tieto dáta ďalej spracovať do formy výslednej šablóny. Na obrázku 5.2 je možné vidieť postupnosť úkonov pri spracovaní požiadavku na spracovanie emailovej správy. Na tomto obrázku je možné vidieť použitie metód opísaných v sekcii 5.1 a taktiež aj metódy popisované v tejto sekcii. Obrázok slúži ako nutná abstrakcia pre jednoduchšie pochopenie implementácie návrhu systému.

Pred samotným spustením algoritmu a aj celého systému, nie je vopred jasné, ktoré hodnoty sú obsiahnuté v daných emailových správach, nakoľko každá množina emailových správ je jednoznačná, a vytvára ju samotná letecká spoločnosť pre svoje účely. Niektoré obsahujú len potvrdenie rezervácie pomocou záznamu PNR a mena pasažiera, čo sú nevyhnutné údaje potrebné pre prístup k rezervácii v rezervačnom systéme každej leteckej spoločnosti. Iné avšak obsahujú aj údaje o všetkých pasažieroch, zoznam letov spoločne so zoznamom segmentov daných letov alebo údaje o platbe. Hlavným cieľom vytvárania týchto šablón je získať potrebné údaje pre prístup k danej rezervácii. V tomto prípade je teda nutné získať daný záznam PNR a meno pasažiera figurujúceho ako kontaktná osoba pre danú rezerváciu.



Obr. 5.2: Znázornenie spracovania požiadavku o vyhodnotenie emailu

5.4 Uloženie šablón a ďalšia podpora

V momente kedy systém dokončí vytváranie šablóny, uloží ju na predom špecifikované miesto, odkiaľ je možné ju získať a ďalej s ňou pracovať. Toto miesto je špecifikované ako databáza v Redise (viz. 4.8). V tomto prípade nejde o relačnú databázu, nakoľko Redis nie je úložiskom, ktoré by túto technológiu podporovalo. V tomto momente je postačujúce tieto dáta uložiť na krátku dobu, nakoľko sa šablóna môže zmeniť zo dňa na deň. Šablóny sú uložené pod kľúčom takzvanej „pre–template“ a teda dočasnej šablóny v podobe spojenia spoločných atribútov emailových správ ako predmet, meno odosielateľa alebo adresa odosielateľa. To sa deje z dôvodu, kedy z rôznych emailových adries a od rôznych odosielateľov, prichádzajú iné emailové správy, a teda je pochopiteľné, že tieto šablóny budú previazané s týmito údajmi.

Vopred vytvorené šablóny sú uložené pre neskoršie používanie a definované pomocou niekoľkých atribútov vyplývajúcich z obsahu a možnosti použitia šablóny. Dôvodom prečo dané šablóny nie sú špecificky pomenované alebo prečo nie sú pomenované napríklad podľa konkrétnej leteckej spoločnosti, je fakt, že v prípade jednej leteckej spoločnosti, alebo jedného odosielateľa, môže byť rôzny počet emailových správ, ktoré sú odosielané od jednej leteckej spoločnosti. Avšak v prípade, že jedna letecká spoločnosť alebo napríklad organizácia, ktorá zastrešuje komunikáciu viacerých leteckých spoločností so zákazníkmi odosiela viacero emailov z tej istej adresy, v tomto momente ale nebudú iné emailové správy obsahovať ten istý predmet správy. Ak by sa tak dialo, tento systém túto možnosť nepodporuje a tak by mohlo dôjsť ku skresleným výsledkom. Tento prípad hodnotím ako veľmi vzdialený od reálnych podmienok, preto v tejto práci nie je aplikovaná žiadna podpora tohoto nežiadúceho stavu.

5.5 Overenie funkčnosti

Pre overenie funkčnosti boli zvolené 2 testovaciu sady emailových správ s počtom 37 a 20. V prípade sady č.1 ide o emailové správy, ktoré vybrané náhodne z množiny emailových správ. Pri výbere bolo cielené vybrať čo najväčší počet leteckých spoločností, nakoľko šablóna emailovej správy sa mení vo všeobecnosti zo spoločnosti na spoločnosť. Niekedy je možné, že jedna letecká spoločnosť bude používať viacero formátov pre správu o potvrdení rezervácie, avšak tento jav je málo pravdepodobný a tieto správy boli vybrané od rôznych leteckých spoločností.

V prípade sady č.2 boli zvolené správy od leteckých spoločností, ktoré patria do rebríčka 20 najväčších leteckých spoločností na svete. Veľkosť bola usudzovaná podľa súčtu sedadiel vo všetkých lietadlách, ktoré letecká spoločnosť má k dispozícii[2].

Testovacia sada emailových správ bola zostavená z emailových správ, ktoré boli anonymizované pre zachovanie súkromia osôb, pre ktoré tieto správy boli určené, nakoľko údaj o potvrdení rezervácie (PNR), je hlavnou referenciou rezervácie a môže teda obsahovať údaje o identite pasažiera (viz. 5.3.2). Taktiež spolu s údajmi o lete, na ktorý je tento údaj vytváraný, sa môže k týmto údajom jednoducho dostať nepovolaná osoba a môže nastať nežiadúci únik informácií[3].

Proces samotného overenia funkčnosti spočíval v spracovaní emailovej správy do formátu, kedy je možné s ňou pracovať. To bolo docielené pomocou knižnice `lxml`, tej istej knižnice, aká bola použitá pri spracovaní správ pomocou knižnice *Beautiful Soup 4* (viz.

[2] https://www.businessinsider.com/biggest-airlines-world-oag-2019-3
[3] https://www.easypnr.com/

4.6.1) a následné spustenie samotného algoritmu pomocou triedy `RobulaPlus`. Tieto emailové správy boli uložené a sú súčasťou priečinku obsahujúceho skripty jednotkových testov, z dôvodu anonymizácie, ako už bolo vyššie spomenuté, ale taktiež z dôvodu, že pre prístup k týmto údajom je potrebný prístup do internej siete Kiwi.com s.r.o., či už pomocou priameho pripojenia alebo pomocou virtuálnej súkromnej siete VPN (*Virtual Private Network*). Vďaka tomu, že tieto správy sú uložené a obsiahnuté v priečinku spolu s testovacími skriptami, je možné toto overenie previesť mimo sieť spoločnosti Kiwi.com s.r.o.

Ďalším z krokov overenia funkčnosti, po samotnom získaní a spracovaní tela emailovej správy, je nutné spracovať informácie potrebné k úspešnému fungovaniu algoritmu ROBULA+. Teda okrem samotného tela správy je potrebná hodnota, ktorá bude vyhľadávaná v emailovej správe.

Následné vyhodnotenie tohoto procesu, záviselo na poslednej uvedenej hodnote v tomto nastavení. Touto informáciou je ručne vybraný XPath lokátor elementu čísla rezervácie a teda po následnom dokončení behu algoritmu, výsledkom čoho je taktiež XPath lokátor, boli porovnané elementy, ktoré boli získané z jednotlivých lokátorov.

Výsledky overenia funkčnosti – sada č.1				
číslo behu	beh č.1	beh č.2	beh č.3	beh č.4
celkový počet súborov	37	37	37	37
počet úspešne spracovaných súborov	28	29	29	28
počet neúspešne spracovaných súborov	9	8	8	9
percentuálna úspešnosť	75.67%	78.38%	78.38%	75.67%

Tabuľka 5.2: Výsledky overenia funkčnosti pomocou testovacieho skriptu

Výsledky overenia funkčnosti – sada č.2				
číslo behu	beh č.1	beh č.2	beh č.3	beh č.4
celkový počet súborov	20	20	20	20
počet úspešne spracovaných súborov	15	16	14	15
počet neúspešne spracovaných súborov	5	4	6	5
percentuálna úspešnosť	75.00%	80.00%	70.00%	75.00%

Tabuľka 5.3: Výsledky overenia funkčnosti pomocou testovacieho skriptu na sade č.2

V overení funkčnosti bol dosiahnutý priemerný výsledok 77.02% na sade č.1 a 75.00% na sade č.2, ktorých výsledky je možné vidieť v tabuľke 5.2 a tabuľke 5.3. Tieto hodnoty symbolizujú fakt, kedy v troch zo štyroch náhodne poskytnutých emailových správach je možné lokalizovať element, ktorý slúži ako hlavná referencia rezervácie. Výška tohoto výsledku spočíva vo fakte, že medzi vybranými správami sa nachádzajú aj správy, ktoré neobsahujú HTML štruktúru vôbec, neobsahujú referenciu rezervácie (PNR) v správe a poskytujú len odkaz pre prihlásenie sa do spravovania rezervácie na webovej stránke leteckej spoločnosti (systém MMB – *Manage My Booking*) alebo štruktúra samotnej stránky to nedovoľuje. V tabuľkách 5.2 a 5.3 je možné vidieť zmenu hodnôt výsledkov pri jednotlivých behoch.

Tento jav je zapríčinený spôsobom vytvárania šablón. Pri vytváraní šablón sú výsledky z algoritmu ROBULA+ sú zoradené podľa váh a následne vybrané tie s najvyššou hodnotou. V prípade že dôjde k tomu, že dva alebo viacero lokátorov majú rovnakú váhu, je výber ponechaný na jazyku Python a jeho vnútornej konfigurácii. V tomto prípade teda môže dojsť k zámene niektorých lokátorov za iné, rovnako robustné. Tento jav je zapríčinený návrhom overenia funkčnosti algoritmu ale pri reálnom vytváraní šablón tejto jav neprináša žiadnu negatívnu funkcionalitu.

Aj keď výsledok je možné ohodnotiť ako dobrý z hľadiska úspešnosti v celku, stále však existuje priestor pre zlepšenie. Ako jedným zo zlepšení by mohol byť výsledný formát aký tento algoritmus produkuje. Týmto je myslený fakt, že momentálne je výsledkom algoritmu lokátor, avšak bez doplňujúcej informácie, kde sa hľadaný údaj nachádza a teda či je to v texte elementu, alebo v konkrétnom atribúte. V budúcnosti by sa dal daný systém upraviť alebo rozšíriť, aby podporoval možnosť poskytnúť informáciu o tom, koľko letov sa nachádza v emailovej správe, koľko pasažierov je obsiahnutých v potvrdení a podobne.

V momentálnom stave, je komplikované overiť ku akej leteckej spoločnosti emailová správa patrí. Ďalším z navrhovaných rozšírení je možné rozšírenie, ktoré by dokázalo určiť, podľa niektorých špecifických vlastností a atribútov správy, ku akej leteckej spoločnosti správa patrí. Toto rozšírenie by pomohlo z hľadiska optimalizácie programu, nakoľko momentálne algoritmus vyhľadáva hodnoty, ktoré by sa v prijatej správe nemali nachádzať, nakoľko súvisia s rezerváciou u inej leteckej spoločnosti. Proti tomuto faktu bola naimplementovaná vlastnosť programu overiť duplikáty v zozname hodnôt, ktoré budú súčasťou vyhľadávacieho procesu.

Navzdory všetkým nedostatkom a obmedzeniam je možné povedať, že program spĺňa základné požiadavky na tento systém (viz. 4.4) a teda, že dokáže poskytnúť po aplikovaní všetkých potrebných úkonov informáciu o tom, ku akej rezervácii, daná emailová správa patrí. To uľahčí mnoho práce zamestnancom, ktorí spravujú tieto rezervácie v internom systéme, ktorí už nebudú musieť čakať na výsledok procesu, kedy vývojár musí vytvoriť šablónu, ktorá úspešne spracuje prichádzajúce správy od leteckej spoločnosti, ale tento systém poskytne túto šablónu po sprístupnení minimálne jednej, vopred vytvorenej rezervácie v internom systéme firmy Kiwi.com s.r.o.

Kapitola 6

Záver

Správy o potvrdení rezervácie sú neoddeliteľnou súčasťou procesu rezervácie v akomkoľvek prípade. Úlohou tejto práce bolo vytvoriť systém, ktorý bude figurovať ako chýbajúci diel rozsiahleho systému, ktorý poskytuje možnosť spravovať rezervácie vytvorené vo firme Kiwi.com ich zákazníkmi, ako aj spoločnosťou u leteckých spoločností.

V tejto práci boli opísané metódy a možnosti extrakcie informácií z textu, avšak nakoľko väčšina emailových správ obsahuje dáta v štrukturovanom formáte, tieto metódy využité neboli. Táto práca závisí na algoritme ROBULA+, ktorý umožňuje vytvárať robustné XPath lokátory, z ktorých sú ďalej vytvárané šablóny. Tieto šablóny je možné ďalej použiť pre spracovanie nových emailových správ a získať dáta potrebné k priradeniu danej správy ku správnej rezervácii.

V rámci tejto práce bola nadviazaná spolupráca s firmou Kiwi.com, vďaka ktorej táto práca nadobudla význam a potrebné dáta pre správny vývoj tejto práce. Firma poskytla možnosť vývoja a nahliadnutia do ich interného systému, čo bolo nápomocné pri výbere použitých technológií. Technológie Redis, Celery alebo Beautiful Soup firma využíva na dennej báze a vďaka tomu dokázala poskytnúť potrebnú podporu v nutnom prípade.

Návrh systému bol realizovaný na základe ustálených noriem a zaužívaných pravidiel v tejto firme a teda vytvorený systém je kompatibilný s aktuálne pracujúcou infraštruktúrou. Systém bol implementovaný v jazyku Python a bol overený na reálnych správach z databázy správ, ktoré boli podrobené anonymizáciou z bezpečnostných dôvodov.

V rámci rozširovania funkcionality tohoto systému existuje niekoľko možností zlepšenia. Jednou je rozšírenie o systém adaptujúci technológiu počítačového videnia, ktorý by bol schopný toto riešenie posunúť na vyšší stupeň autonómnosti. Vyšší stupeň autonómnosti znamená, že systém by bol sám schopný rozoznať od akej leteckej spoločnosti daný email prišiel, koľko údajov o pasažierovi obsahuje a koľko letov sa nachádza v správe. Nakoľko to sú hlavné nedostatky tohoto systému.

Funkcionalitu a úspešnosť tejto práce je možné ohodnotiť ako dostačujúcu (približne 75%), pre základné spracovanie emailovej správy u ktorej nemáme žiadne informácie o obsahu tejto správy. Pomocou vyextrahovaných informácií je možné získať referenciu k rezervácii vytvorenej v systéme dopravnej spoločnosti, nakoľko táto práca nie je obmedzená iba na letecké spoločnosti aj keď letecké spoločnosti sú jej hlavným zameraním.

Literatúra

[1] Anderson, B.: Best Automation Testing Tools for 2019 (Top 10 reviews). Oct 2017, [Online; navštívené 28.4.2019].
URL https://medium.com/@briananderson2209/best-automation-testing-tools-for-2018-top-10-reviews-8a4a19f664d2

[2] Bird, S.; Klein, E.; Loper, E.: *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. Sebastopol, CA: O'Reilly Media Inc, 2009, ISBN 0596516495.

[3] Chapman, S.: *JavaScript and Emails*. Marec 2019, [Online; navštívené 14.4.2019].
URL https://www.thoughtco.com/javascript-and-emails-2037682

[4] Crocker, D.: *STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES*. STD 11, RFC Editor, August 1982.
URL http://www.rfc-editor.org/rfc/rfc822.txt

[5] Cypress.io: *End to End Testing Framework*. May 2019, [Online; navštívené 2.5.2019].
URL https://www.cypress.io/how-it-works/

[6] Cermak, F.: *Jazyk a jazykoveda: Prehled (Czech Edition)*. Prazska imaginace, 1994, ISBN 8071101494.

[7] Fielding, R. T.; Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Technická Správa 7231, Jún 2014, doi:10.17487/RFC7231.
URL https://rfc-editor.org/rfc/rfc7231.txt

[8] Google: Puppeteer – Tools for Web Developers. [Online; navštívené 28.4.2019].
URL https://developers.google.com/web/tools/puppeteer/

[9] Hauser, M.: *HTML a CSS : velka kniha reseni*. Brno: Computer Press, 2006, ISBN 8025111172.

[10] IATA: *Facilitation and Passenger Data*. [Online; navštívené 1.5.2019].
URL https://www.iata.org/whatwedo/passenger/Pages/passenger-data.aspx

[11] Klensin, J.: *Simple Mail Transfer Protocol*. RFC 5321, RFC Editor, October 2008.
URL http://www.rfc-editor.org/rfc/rfc5321.txt

[12] Labs, R.: *Why Redis*. [Online; navštívené 25.4.2019].
URL https://redislabs.com/why-redis/

[13] Leotta, M.; Stocco, A.; Ricca, F.; aj.: *ROBULA+: an algorithm for generating robust XPath locators for web testing. Journal of Software: Evolution and Process*, ročník 28, č. 3, 2016: s. 177–204, doi:10.1002/smr.1771.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1771

[14] MDN: *Introduction to the DOM.*
URL https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

[15] Namratha, S.; Srivatsa, R.; Veena, B.: *Textual Emotion Detection Using Emotion Detector Algorithm*, ročník 6. IJIRSET, jul 2017, [Online; navštívené 1.5.2019].
URL http://www.ijirset.com/upload/2017/iccstar/10_I012_n.pdf

[16] Pustejovsky, J.; Stubbs, A.: *Natural language annotation for machine learning.* Sebastopol, CA: O'Reilly Media Inc, 2013, ISBN 978-1-449-30666-3.

[17] Resnick, P. W.: *Internet Message Format.* RFC 5322, RFC Editor, October 2008.
URL http://www.rfc-editor.org/rfc/rfc5322.txt

[18] Richardson, L.: *Beautiful Soup Documentation.* [Online; navštívené 25.4.2019].
URL https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[19] Richardson, L.; Ruby, S.: *RESTful web services.* Sebastopol, CA: O'Reilly Media Inc, 2007, ISBN 978-0-596-52926-0.

[20] Robie, J.; Dyck, M.; Spiegel, J.: *XML Path Language (XPath) 3.1.* [Online; navštívené 14.4.2019].
URL https://www.w3.org/TR/2017/REC-xpath-31-20170321/

[21] Ronacher, A.: *Flask.* [Online; navštívené 25.4.2019].
URL http://flask.pocoo.org/

[22] Rouse, M.: *What is integrated development environment (IDE).* [Online; navštívené 28.4.2019].
URL https://searchsoftwarequality.techtarget.com/definition/integrated-development-environment

[23] Schafer, S. M.: *HTML, XHTML a CSS.* Grada, 2009, ISBN 978-80-247-2850-6.

[24] Solem, A.: *Celery – Distributed Task Queue.* [Online; navštívené 25.4.2019].
URL http://docs.celeryproject.org/en/latest/index.html

[25] SQLCourse: *What is SQL?* Aug 2000, [Online; navštívené 28.4.2019].
URL http://www.sqlcourse.com/intro.html

[26] Techopedia.com: *What is Application Framework?* [Online; navštívené 28.4.2019].
URL https://www.techopedia.com/definition/6005/application-framework

[27] Tidwell, D.: *XSLT: Mastering XML Transformations.* Sebastopol, CA: O'Reilly Media Inc, druhé vydanie, 2008, ISBN 978-0-596-52721-1.

[28] UiPath: What is Robotic Process Automation? - RPA Software. [Online; navštívené 28.4.2019].
URL https://www.uipath.com/rpa/robotic-process-automation

[29] Winter, L.: *Algoritmy pro rozpoznávání pojmenovaných entit*. 2017, Vysoké učení technické v Brně. Fakulta strojního inženýrství.

Prílohy

Zoznam príloh

Príloha A

Manuál

Systém je možné spustiť v plnej funkčnosti, iba v internej sieti firmy Kiwi.com, nakoľko táto aplikácia vyhodnocuje a pracuje s citlivými dátami o zákazníkoch tejto firmy. K spusteniu sú pripravené testovacie skripty z ktorých časť nie je nutné spúšťať v internej sieti Kiwi.com.

Pred prvým spustením je nutné sa uistiť že sú nainštalované všetky prerekvizity tohoto systému.

Technické požiadavky systému:

- Python vo verzii 3.6
- Docker daemon (nie je nutné mimo siete Kiwi.com)

Po nainštalovaní jazyka Python, je nutné nainštalovať všetky závislosti tohoto systému. To je možné urobiť pomocou príkazu:
`pip install -r requirements/requirements_public.txt`.
Súbor `requirements_public.txt` obsahuje závislosti nutné k spusteniu testov.

Testy je následne možné spustiť v zdrojovom adresári príkazom:
`pytest test/unit/test_parse.py`.

Tento testovací súbor obsahuje definíciu testovacieho skriptu, kedy sú využité reálne HTML správy uložené v testovacom priečinku. Na týchto testovacích správach je možné pomocou vyššie spomenutých testov previesť základnú analýzu pomocou jadra implementovaného systému.

V prípade pripojenia na internú sieť je možné predviesť plnú funkcionalitu, pomocou predpripravených súborov `Dockerfile` a `docker-compose.yml`, ktoré obsahujú popis častí systému a ich konfiguráciu. Pomocou príkazu `docker-compose build` je možné vytvoriť obraz systému a následne pomocou príkazu `docker-compose up` je možné tento systém uviesť do prevádzky. Pri spustení tento systém počúva na adrese `localhost:8000`, kam je možné posielať požiadavky.