Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Q&A Chatbot

Bc. Miloš Havránek

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Miloš Havránek

Informatika

Název práce

Q&A Chatbot

Název anglicky

Q&A Chatbot

Cíle práce

Cílem práce je návrh chatovacího robota (chatbota), který umožní odpovídat na otázky položené z kontextu předmětu Interakce člověk PC a dalších. Báze učebních dat bude získána ze zkušebních otázek v Moodle testech z daného předmětu.

Metodika

Nastudujte problematiku chat robotů.

1) Definujte vhodnou technologii pro realizaci stroje.

2) Stroj naprogramujete s využitím dostupných technologí.

3) Naučte jej odpovídat v kontextu položených otázek s přesností alespoň 50%.

4) Řešení otestujte a ověřte.

5) Práci průběžně konzultujte.

Doporučený rozsah práce

52 str.

Klíčová slova

chatbot, báze dat, architektura, kontext otázky, kontext odpovědi, programování

Doporučené zdroje informací

Soft Computing: Integrating Evolutionary, Neural, and Fuzzy Systems,Andrea Tettamanzi, Marco Tomassini,Springer Science & Business Media, 7. 9. 2001

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

Ing. Josef Pavlíček, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 31. 1. 2019

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 31. 1. 2019

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 17. 02. 2019

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Q&A Chatbot" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 26.3.2019 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Josefu Pavlíčkovi, Ph.D. za možnost pod jeho vedením zpracovat svou diplomovou práci na téma Q&A Chatbot.

Q&A Chatbot

Abstrakt

Diplomová práce je zaměřena na vývoj chatbota, jehož hlavním účelem je zodpovídat položené otázky z naučené báze dat. V diplomové práci jsou vymezeny základní pojmy z oblasti programovacího jazyka Python a souvisejících frameworků. Práce se zabývá vývojem chatbota ve funkční prototyp schopný produkčního nasazení.

Klíčová slova: chatbot, báze dat, architektura, kontext otázky, kontext odpovědi, programování

Q&A Chatbot

Abstract

Master thesis is focused on chatbot development. The main goal of the chatbot is to answer asked question from learned data base. Basic concepts of programming language Python and its frameworks are defined in this master thesis. Master thesis is about developing a chatbot as functional prototype being capable of production deployment.

Keywords: chatbot, data base, architecture, context of question, context of answer, programming

Obsah

1 Úvod 13

2 Cíl práce a metodika 14

2.1 Cíl práce 14

2.2 Metodika 14

2.2.1 Studium odborné literatury 14

2.2.2 Vývoj aplikace 14

2.2.3 Souhrn dosažených výsledků, diskuse a závěr 14

3 Teoretická východiska 15

3.1 Chatbot 15

3.1.1 Co je to chatbot 15

3.1.2 Jaké jsou typy chatbotů 15

3.1.3 Klasifikace chatbotů dle zpracování textu a odpovědi 16

3.1.4 Co od chatbota očekávat 19

3.2 Levenshteinova vzdálenost 19

3.3 Java 21

3.4 Python 21

3.4.1 Historie a kompatibilita 21

3.4.2 Virtualenv 22

3.4.3 Pipenv 22

3.5 Django 22

3.5.1 Objektový přístup k databázím 22

3.5.2 Další funkcionality 22

3.6 REST 23

3.6.1 Metody pro přístup k datům: 23

3.6.2 Metody pro přístup ke zdrojům: 23

3.7 Chatterbot 23

3.7.1 Jak Chatterbot funguje 24

3.8 Docker 25

3.8.1 Kontejnery versus virtuální stroj 26

3.8.2 Jak funguje Docker 26

3.8.3 Dockerfile 27

3.8.4 Docker compose 27

3.9 Postgresql 27

3.10 Git 28

3.11 HTML a CSS 28

3.12 Javascript 28

3.12.1 Ajax 28

4 Vlastní práce 30

4.1 Analýza aktuálních řešení na trhu 30

4.2 Analýza a rozbor zadání 33

4.2.1 Vymezení zadání dle kategorizace na trhu 33

4.2.2 Zadání 34

4.2.3 Vybrané řešení 35

4.3 Návrh a Implementace 35

4.3.1 High level architektura komponent 35

4.3.2 Struktura projektu 37

4.3.3 Django 37

4.3.4 ChatterBot 38

4.3.5 Nastavení Chatbota 39

4.3.6 Trenéři dat 39

4.3.7 Adaptéry 40

4.3.8 Levenshteinova vzdálenost 41

4.4 Web GUI 44

4.4.1 Návrh webového rozhraní 45

4.4.2 Další funkcionality a problémy 45

4.4.3 Příklad uživatelské interakce 46

4.5 REST API 47

4.5.1 Příklady volání REST API 47

4.6 Testovací data 49

4.6.1 Analýza 49

4.6.2 Zpracování dat 53

4.6.3 Transformace dat 54

4.7 Deployment 54

4.7.1 Prerekvizity 54

4.7.2 Dockerfile 55

4.7.3 Docker compose 56

4.7.4 Samotné nasazení aplikace 56

4.7.5 Aktualizace již deployované aplikace 57

4.7.6 První spuštění 57

4.7.7 Auto-migrace databáze 58

4.7.8 Správa aplikace pomocí „manage.py“ 58

4.7.9 Naučení testovacích dat 59

4.8 Otestování a verifikace 60

4.8.1 Prázdný vstup 60

4.8.2 Vstup neobsažený v datech 61

4.8.3 Vstupy obsažené v datech 61
4.8.4 Hranice (ne)rozpoznání vstupu 62

5 Výsledky a diskuse 65

6 Závěr 66

7 Seznam použitých zdrojů 67
7.1 Knižní zdroje 67
7.2 Internetové zdroje 67

8 Přílohy 69

Seznam obrázků

Obrázek 1: Klasifikační matice chatbotů 17
Obrázek 2: Strojový překlad vstupu na odpověď 18
Obrázek 3: Diagram zpracování vstupu na odpověď 25
Obrázek 4: Srovnání kontejnerizované aplikace a virtuálního stroje 26
Obrázek 5: High level architektura použitých komponent chatbota Interakt 36
Obrázek 6: Primární chatovací stránka s chatovacím oknem 45
Obrázek 7: Uživatelská interakce s chatbotem 46

Seznam tabulek

Tabulka 1: Výpočet buňky tabulky Levenshteinova algoritmu 20
Tabulka 2: Tabulka provedených operací nad výrazem A pro ekvivalenci výrazu A a B. . 42
Tabulka 3: Vizualizovaný výpočet Levenshteinovy vzdálenosti 43
Tabulka 4: Rozdíly nastavení prostředí pro deployment 54

Seznam použitých zkratek

zkratka	popis
Chatbot	Program k automatizované komunikaci s lidmi
Q&A	Questions and Answers (otázky a odpovědi)
Intent	Označení skupiny klíčových slov ve větách sdružující význam konverzace.
Deployment	Nasazení aplikace
Frontend	Část aplikace zajišťující prezentaci aplikace
Backend	Část aplikace zajišťující obchodní logiku
Pattern matching	Srovnávání textu se vzorem
API	Application Programming Interface – rozhraní pro programování aplikací
Dependency	Softwarová závislost, většinou knihovna či framework
Preprocesory	Preprocesory provádějí zpracování před samotným hlavním zpracováním.
Framework	Softwarová struktura pro podporu vývoje software. Často ve formě knihoven.
TCO	Total Cost of Ownership
Open-source	Otevřený software
Data mining	Analytická metodologie získávání netriviálních skrytých a potenciálně užitečných informací z dat.
Vendor lock-in	Proprietární uzamčení zákazníka. Znamená složitý nebo nemožný přechod na jiné řešení.
SDK	Software Development Kit
Cloud	Počítačové zdroje spravované třetí stranou, kde uživatel pouze využívá poskytnutých zdrojů výměnou za peníze.
On-premise	Lokální řešení u zákazníka
GUI	Graphical User Interface – grafické uživatelské rozhraní
NLP	Natural Language Processing – neurolingvistické programování
Closed-source	Uzavřený software
NodeJS	Serverový Javascript
SQL	Standardizovaný strukturovaný dotazovací jazyk

NO-SQL	Databázový koncept, kde je datové úložiště i zpracování dat odlišné od SQL.
Javascript	Skriptovací jazyk pro webové prohlížeče a webové stránky.
JQuery	Kdysi populární javascriptový framework
Verbose	Termín využívaný u aplikací, kdy je potřeba podrobnějšího výpisu logování.
Shell script	Soubor obsahující sérii linuxových příkazů
Git	Software pro verzování zdrojových kódů
VM	Virtuální stroj (Virtual machine)
Hypervizor	Software monitorující a spravující VM
JSON	(JavaScript Object Notation) formát souboru používaný pro přenos a ukládání dat.
XML	(Extensible Markup Language) formát souboru používaný pro přenos a ukládání dat.

1 Úvod

Chatbot je poměrně novým termínem, kterému dává vznik rozšíření a popularizace technologií strojového učení. Jedná se o počítačový program, jehož cílem je částečně nahradit potřebu člověka pro zodpovídání dotazů uživatele. Cílem uživatele bývá získat konkrétní informace. Takové informace by za normálních okolností musel uživatel získávat od druhého člověka či z rozsáhlejších dokumentů.

Tato diplomová práce se zabývá problematikou realizace autonomních beze stavových konverzačních strojů a technických řešení zpracování jazykových mutací při vyhodnocovaní uživatelského vstupu.

Konkrétně se pak práce zabývá Q&A typem chatbota, tedy mapování stochastického vstupu deterministickými metodami na deterministický výstup. Jinými slovy vzít předem neznámý vstup uživatele, ten rozpoznat a odhadnout pomocí známých metod či algoritmů, zdali se vstupní otázka nachází v naučené bázi dat. A pokud ano, tak vrátit odpovídající odpověď nebo případně uživatele upozornit na nemožnost rozpoznání vstupní otázky.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem této práce je naprogramovat funkční prototyp Q&A Chatbota, jehož účelem je odpovídat na položené otázky z kontextu naučené databáze otázek a odpovědí. Tento prototyp bude možné dál rozvíjet a využít kupříkladu jako experimentální doplněk k výuce studentů nebo jej použít jako plnohodnotnou aplikaci.

2.2 Metodika

Pro vyhotovení práce je sestaven následující postup:

2.2.1 Studium odborné literatury

Nedílnou součástí této diplomové práce je studium teoretických východisek, které objasní základní pojmy v oblasti problematiky chatbotů, programování v prostředí jazyka Python, a jeho souvisejících frameworků. Studium odborné literatury probíhá souběžně se zpracováním analytické části diplomové práce. Tato vymezení jsou stěžejní pro kvalitní zpracování celé diplomové práce. Teoretická východiska jsou zpracována zejména z literárních zdrojů a z části ze zdrojů internetových. Použité zdroje jsou vyjmenovány v kapitole „Seznam použitých zdrojů“.

2.2.2 Vývoj aplikace

Praktická část pojednává o návrhu, naprogramování, nastavení a sestavení funkčního řešení aplikací s jednoduchými příklady použití za pomoci znalostí získaných ze studia odborné literatury a praktických zkušeností z vývoje software. Samotná implementace následuje po výběru vhodných technologií pro realizaci stroje. Po implementaci by měla být aplikace otestována, kde je očekávána přesnost alespoň 50 % pro odpovídání v kontextu položených otázek.

2.2.3 Souhrn dosažených výsledků, diskuse a závěr

V kapitole Závěr je zhodnoceno naplnění kladeného cíle diplomové práce.

3 Teoretická východiska

3.1 Chatbot

3.1.1 Co je to chatbot

Chatbot je počítačový program, který interaguje s lidmi způsobem, kdy do určité míry napodobuje nebo se snaží napodobit lidské chování. Interakce s takovým strojem může nabývat různé složitosti, konkrétně od řízení chování a odpovědí pomocí klíčových slov, přes systémy vyžívající neurolingvistického programování (NLP) po systémy využívající technik umělé inteligence. Taková konverzační forma interakce člověk-počítač může sloužit jako stavební kámen pro mnoho použití řešící problémů v reálného světa. (Alphabet Inc., 2018)

3.1.2 Jaké jsou typy chatbotů

3.1.2.1 Chatboti založení na jednoduché logice

Chatboti založení na jednoduché logice existují v mnoha formách. Takový chatbot potom může komunikovat s uživatelem, za předpokladu vstupu, u kterého lze jednoznačně určit jeho funkční význam, avšak ne kontextuální. (Phillips, 2018)

Nejčastější jsou tyto formy vstupů:

- Klikání na tlačítka
- Uživateli známá příkazová logika
- Rozpoznávání klíčových slov (Pattern matching)

U tzv. klikajících chatbotů je velmi jednoduché dorozumění se s uživatelem. Uživatel vidí tlačítka, která mu dávají smysl a chatbot zná příznak, co se má stát při stisku takového tlačítka. Uživatel si většinou vybere akci z konečné množiny možností a chatbot programaticky provádí akce odpovídající danému tlačítku v daném stavu. Jako doplnění tlačítek se často používají vstupy uživatele. Dobrým příkladem je dotázání chatbota na jméno uživatele a uživatel jej vypíše do vstupu konverzačního pole a odešle. Takový vstup je pak většinou validován na integritní omezení bez validace smysluplnosti. (Phillips, 2018)

Chatboti založení na příkazové logice mají velice jednoduché rozpoznávání klíčových slov. Intent je označení skupiny klíčových slov ve větách sdružující význam konverzace, tedy za jakým účelem je konverzace vedena. Rozdílem oproti chatbotům založených na rozpoznávání klíčových slov je hlavně ten, že chatbot založený na příkazové logice od uživatele přebírá pouze klíčová slova a jejich parametry. Kdežto chatbot založený na rozpoznávání klíčových slov převezme větu obsahující mnoho slov, která jsou z pohledu rozpoznání intentu zbytečná, a navíc parametry uživatele se hůře odhadují. (Phillips, 2018)

3.1.2.2 Q&A chatboti versus konverzační chatboti

Konverzační chatboti mívají typicky tyto vlastnosti (Sanjeevi, 2018):

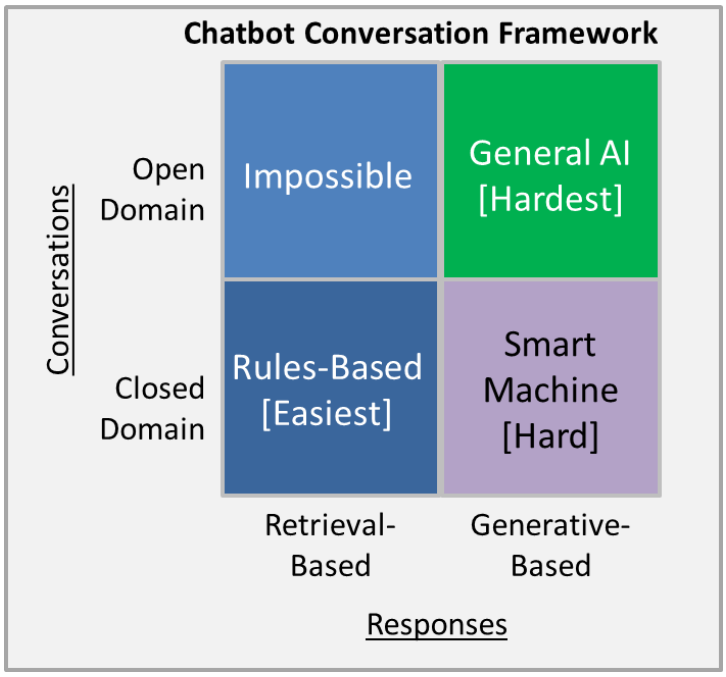
- intent (cíl uživatele v konverzaci: zjistit aktuální počasí)
- proměnné (datum: 1.1.2019, místo: Praha, jméno uživatele: Petr)
- personifikovaná odpověď uživateli („Dobrý den Petře, 1.1.2019 bude v Praze oblačno až zataženo a místy s občasným sněžením. Nejvyšší teploty -5 až -1 °C, na horách -10 až -6 °C.“)

Konverzační chatboti umí být i sofistikovanější a je možné se s nimi bavit s omezením dle dané konfigurace a implementace. Q&A chatboti se oproti tomu soustředí na poskytování znalostí ze znalostní báze dat nebo internetových zdrojů. Q&A může být vraceno přímo nebo generováno na základě předchozích učení. Konverzační chatboti jsou však mnohem vyspělejší než Q&A chatboti. (Sanjeevi, 2018)

3.1.3 Klasifikace chatbotů dle zpracování textu a odpovědi

U chatbotů je důležité rozlišovat rozsah jejich zaměření. Některé chatboty je možné realizovat jednoduše a jiné velmi složitě za nynějších metodik. Obecně je možné klasifikaci chatbotů rozdělit na matici 2x2, kde se řeší způsoby získání odpovědi pro uživatele a rozsah zaměření konverzace chatbota. Obrázek popisuje obtížnosti realizace frameworku pro chatbota za různých kombinací daných klasifikačních kritérií.

Obrázek 1: Klasifikační matice chatbotů



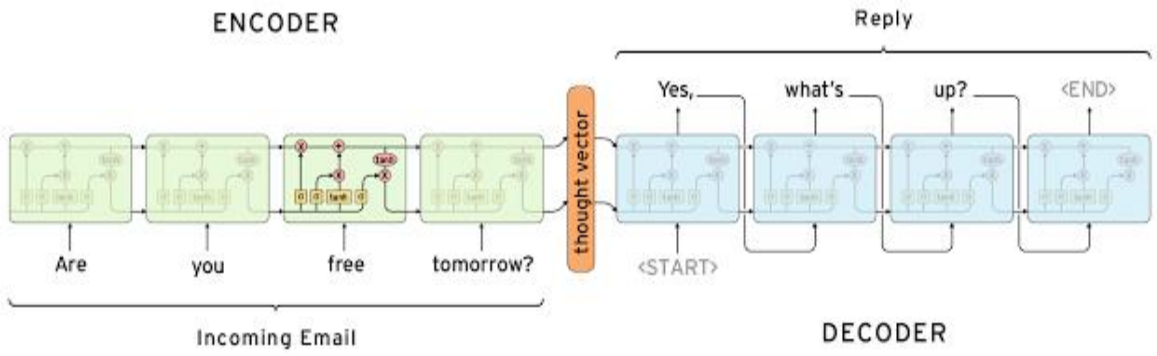
Zdroj: Kojouharov, 2016

3.1.3.1 Odpovědi na základě vyhledávání versus generativní modely

Odpovědi na základě vyhledávání (snazší) využívají bázi dat s předdefinovanými odpověďmi. Odpověď je následně vybrána pomocí heuristického algoritmu na základě kontextu otázky vstupu uživatele. Heuristický algoritmus může být z hlediska složitosti implementován od jednoduchých algoritmu jako například příkazová logika, pattern matching až po složitější soubory klasifikátorů strojového učení. Takové systémy potom odpověď získají ze známé báze dat a žádné nové odpovědi již negenerují. (Kojouharov, 2016)

Generativní modely (obtížnější) nepoužívají bázi dat s předdefinovanými odpověďmi, ale generují nové odpovědi. Generativní modely se většinou zakládají na technikách strojového překladu. Rozdílem od klasického překladu z jazyka do jazyka je ale překlad ze vstupu na odpověď uživateli. (Kojouharov, 2016)

Obrázek 2: Strojový překlad vstupu na odpověď



Zdroj: Kojouharov, 2016

Pokud srovnáme oba uvedené přístupy je zřejmé, že oba své výhody a nevýhody. Odpovědi založené na vyhledávání z báze dat jsou přímé a nebude v nich udělána gramatická chyba způsobená chatbotem. Nevýhodou však může být malý počet předdefinovaných odpovědí nebo žádná vhodná pro odpověď uživateli na jeho vstup. Také není možné odpovědi uživateli personalizovat na základě předešlého kontextu konverzace. Generativní modely jsou „chytřejší“. Vzhledem k jejich práci s kontextem v konverzaci mohou působit jako kdyby uživatel komunikoval přímo s člověkem. Naučit takový model je ale těžké a jsou k tomu potřeba vhodná data, kterých musí být velké množství. Generativní modely mají problém se skloňováním, proto také často taková řešení bývají vázána na konkrétní jazyk. (Kojouharov, 2016)

Svoji roli v komunikaci s uživatelem hraje také délka samotné konverzace. Konverzace s kratší délkou bude pochopitelně snazší na automatizaci, kdežto delší konverzace bude obtížnější. U Q&A chatbotů se kupříkladu odpovídá vždy jen na samotný vstup a konverzace je tedy nejkratší možná, protože se skládá ze vstupu uživatele a odpovědi chatbota. U delších konverzací, které jsou typické pro konverzační chatboty, je naopak složitější konverzaci automatizovat a každý další konverzační cyklus je složitější. Takový systém si musí být vědom konverzace jako celku z pohledu kontextu či intentu. (Kojouharov, 2016)

3.1.3.2 Otevřená versus uzavřená doména konverzace

V **otevřené doméně konverzace (obtížnější)** může uživatel svými vstupy zavést konverzaci kamkoliv. Takový přístup nemá žádný jasně definovaný cíl či záměr. Konverzace tohoto typu se vyskytují většinou na sociálních sítích, kde se může konverzace vyvinout jakýmkoliv

směrem. Nekonečné množství možných témat a potřebné znalosti o světě, které má každý člověk, z daného přístupu činí velice složitý problém na uchopení a realizaci. (Kojouharov, 2016)

Uzavřená doména konverzace (snazší) počítá s tím, že hranice daných témat je jasně daná. Takový chatbot potom je schopen odpovídat na omezený počet témat v předem definovaných situacích. Chatbot má definovaný účel, kolem kterého je doména konverzace postavena. Může se jednat například o pomoc s nákupem, získání jakýkoliv informací o firmě nebo technická podpora. Uživatel také počítá s tím, že takový chatbot je schopen se bavit s uživatelem pouze účelově nebo mu to chatbot dá sám najevo. (Kojouharov, 2016)

3.1.4 Co od chatbota očekávat

I přesto, že chatbot může do jisté míry nahradit člověka, nelze očekávat, že je chatbot univerzální řešení, jak nahradit člověka a automatizovat konverzace s uživatelem. Na chatboty lze pohlížet minimálně jako na systémy pro usnadnění uživatelského zážitku.

3.1.4.1 Možné použití v praxi v business oblasti

Každý chatbot byl vytvořen za nějakým konkrétním účelem. Pro firmy takový chatbot může znamenat při správné aplikaci např. minimalizaci nákladů, kde lze v některých situacích člověka nahradit. Uživatel tím také může získat přístup k informacím mimo pracovní dobu a firmy mohou mít méně lidských operátorů obsluhujících uživatele.

Reálné aplikace mohou být na příklad:

- technická podpora
- zodpovězení nejčastěji pokládaných otázek
- informace o firmě
- podpora prodeje zboží a služeb

3.2 Levenshteinova vzdálenost

Levenshteinova vzdálenost je vzdálenost dvou řetězců definovaná, jako minimální počet operací, po kterých budou zadané řetězce totožné. (algoritmy, 2019)

Operace (algoritmy, 2019):

- vkládání znaku
- smazání znaku
- nahrazení znaku

Algoritmus funguje následovně (Kleiweg 2019):

1. Algoritmus začíná levým horním rohem dvourozměrného pole, které je vyplněno indexem v řádcích znaky zdrojového výrazu a ve sloupcích znaky výrazu cílového.
2. Vyplní se zbytek pole při nalezení všech vzdáleností mezi každou počáteční předponou zdrojového výrazu na jedné straně a každou počáteční předponou výrazu cílového.
3. Každá buňka [i, j] představuje (minimální) vzdálenost mezi prvními znaky v „i“ zdrojového výrazu a prvními znaky „j“ výrazu cílového.
4. Hodnotu buňky je možné vyplnit pouze v případě, že jsou vyplněny hodnoty všech sousedních buněk nalevo a nahoře.
5. Počáteční hodnota diagonály je vždy menší nebo rovno 0.
6. První řádek je vždy inkrementální číselná řada „ceny vložení znaku“.
7. První sloupec je vždy inkrementální číselná řada „ceny smazání znaku“.
8. Hodnoty jednotlivých cen je možné měnit, ale většinou nabývají hodnot 1.

Výpočet buňky tabulky Levenshteinova algoritmu:

Tabulka 1: Výpočet buňky tabulky Levenshteinova algoritmu

diagonála	horní
vlevo	minimum (horní + smazání, diagonála + nahrazení, levé + vložení)

Zdroj: Kleiweg, 2019

Jakmile jsou buňky nahoře a vlevo vyplněny, je možné vyplnit sousední buňku. Také je nutné spočítat všechny tři hodnoty, které se tam mohou objevit. (Kleiweg 2019)

- Výsledek vymazání znaku indexujícího tento sloupec (přidaného k hodnotě přímo nad)
- Výsledek vložení znaku indexujícího řádek (přidaného k hodnotě vlevo)

- Výsledek nahrazení znaku sloupce písmenem řádku (přidáno k hodnotě diagonálně, horní a vlevo).

Zbývající buňky jsou vyplněny pomocí volby minimálního čísla z předchozích výpočtů tří vzorců. (Kleiweg 2019)

3.3 Java

Java je objektově orientovaný programovací jazyk a patří mezi jeden z nejpopulárnějších jazyků dnešní doby. Důvodem popularity je pestrá nabídka API a knihoven třetích stran, které jsou pro samotnou platformu jazyka dostupné. (Paul, 2019)

Je obecně známo, že programovací jazyk Java má všestranné použití. Dříve tomu opravdu tak bylo a Javu bylo možné nalézt na mobilních zařízeních, webu, desktopu a serverech. Nyní však použití Javy ustupuje pouze jako backend webových aplikací a serverových řešení. (Paul, 2019)

3.4 Python

Python je dynamicky interpretovaný programovací jazyk. Většinou je řazen mezi skriptovací jazyky. Lze v něm vyvíjet jednoduché skripty, ale i objektově orientované programování či funkcionální programování. (Paul, 2019)

Python je současně velice oblíbený výukový programovací jazyk na univerzitách, kde na některých místech nahradil jazyk Java. Jeho využití je však více univerzální. V pythonu je možné vyvíjet jednoduché automatizační skripty. Na popularitě nabyl díky využití u strojovému učení, psaní webových aplikací a dalších. (Paul, 2019)

3.4.1 Historie a kompatibilita

Existují dvě hlavní verze Pythonu. Původní verze Python 2 je starší verze a Python 3 je nově vyvíjená verze. Obě verze vzájemně nejsou kompatibilní, nicméně lze vyvíjet aplikace tak, aby byly kompatibilní pro Python 2 a zároveň byly i funkční na Python 3. Záleží však na použitých funkcionalitách a dalších náležitostech. (Wichmann, 2017)

3.4.2 Virtualenv

Virtualenv je nástroj, který vytváří izolované Python runtime prostředí. Izolované prostředí obsahuje vlastní binární soubory dané verze pythonu a veškeré knihovny, na kterých je aplikace závislá. Díky tomu může být spuštěná aplikaci nezávisle na globálně instalovaných knihovnách či verzi samotného Pythonu. (Bicking, 2018)

3.4.3 Pipenv

Pipenv je dependency management nástroj (nástroj pro správu závislostí), který cílí na snadnou práci s externími knihovnami a závislostech na nich. Pipenv automaticky vytváří a spravuje virtualenv. Pipenv umí při prvním spuštění příkazu „install“ převzít dependency ze souboru „requirements.txt“, pokud takový soubor existuje. Pro uložení verzí nebo rozsahu verzí knihoven, které jsou vyžadovány zdrojovými kódy aplikace slouží soubor „Pipfile“. Soubor „Pipfile.lock“ je pak generovaný soubor obsahující přesně instalované verze knihoven a jejich kontrolní součty. (Reitz, 2018)

3.5 Django

Django je vyspělý webový framework programovacího jazyka Python. Napomáhá k rychlejšímu a ucelenějšímu vývoji. Soustředí se na maximální produktivitu a znovu použitelnost samotného vývoje. Nejpoužívanější architektonickým vzorem frameworku Django je Model-View-Controller. V Djangu je interpretován jako MTV (Model-Template-View). (Dvořák, 2009)

3.5.1 Objektový přístup k databázím

V Djangu je preferován ORM (Object-Relational Mapping) přístup. Výsledkem toho je práce s objekty namísto samotných SQL dotazů. Python ORM operace jsou pak frameworkem Django automaticky převedeny na SQL dotazy, se kterými framework na nižší úrovni pracuje. (Dvořák, 2009)

3.5.2 Další funkcionality

Django nabízí také mnoho dalších vestavěných funkcionalit. Každá webová aplikace má integrované administrační rozhraní, ve kterém je možné procházet data v databázi a

spravovat tak celou aplikaci. Samozřejmostí je také tzv. „šablonovací“ systém, který slouží k samotné prezentaci dat na webu. (Dvořák, 2009)

3.6 REST

REST (Representational State Transfer) je architektura rozhraní, která je navržená pro distribuované prostředí, je datově orientovaný a jeho hlavním úkolem je jakým způsobem se přistupuje k datům. (Malý, 2009)

3.6.1 Metody pro přístup k datům:

- Přístup reprezentovaný zkratkou CRUD: (Malý, 2009)
    - Create (C) – Vytvoření dat
    - Retrieve (R) – Získání požadovaných dat
    - Update (U) – Změna, aktualizaci dat
    - Delete (D) – Smazání dat

Uvedené metody jsou implementovány pomocí metod HTTP protokolu.

3.6.2 Metody pro přístup ke zdrojům:

REST implementuje čtyři základní metody, známé pod označením **CRUD**. Tyto metody jsou implementovány pomocí odpovídajících metod HTTP protokolu. Jednotlivá písmena představují zkratky níže uvedených slov: (Malý, 2009)

- GET (Retrieve)
- POST (Create)
- PUT (Update)
- DELETE

3.7 Chatterbot

Chatterbot je knihovna programovacího jazyka Python, která usnadňuje vytvoření chatbota a generování automatizovaných odpovědí na uživatelské vstupy. Chatterbot využívá výběru algoritmů strojového učení k produkování různých odpovědí. (Cox, 2019)

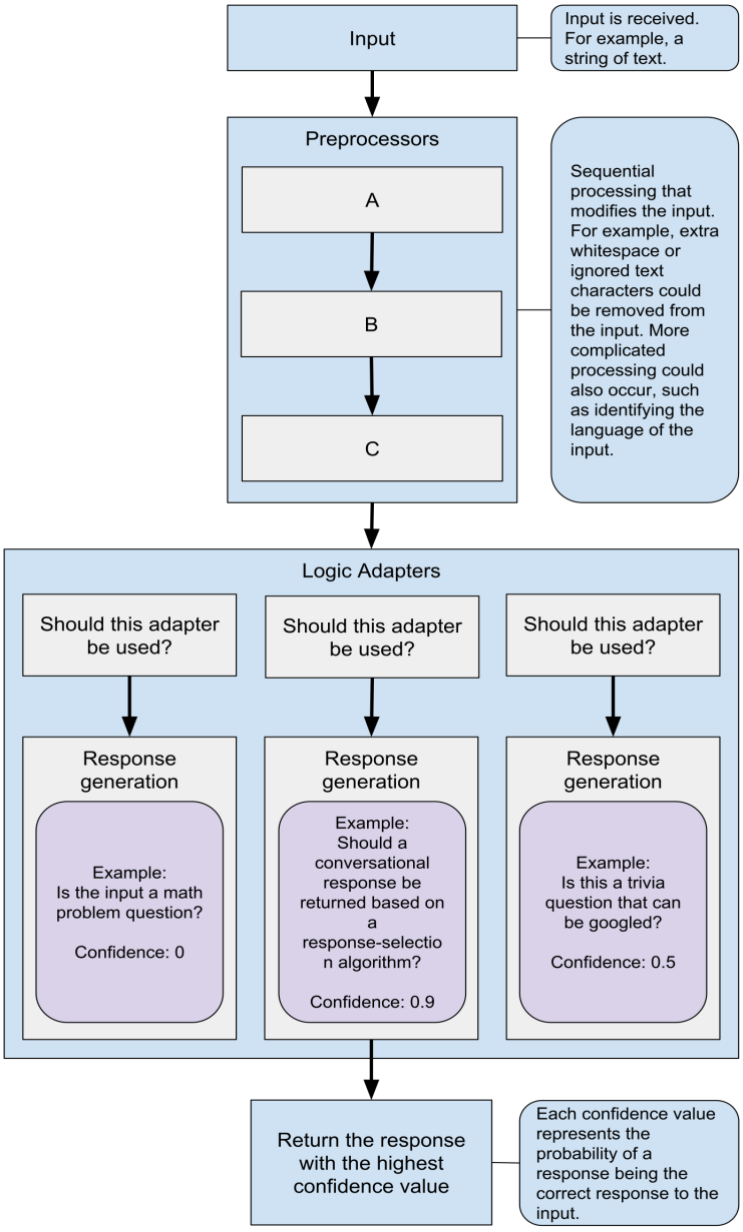
3.7.1 Jak Chatterbot funguje

Čistá instance chatbota využívajícího knihovny Chatterbot nemá žádnou znalost o tom, jak komunikovat. Vždy, když uživatel odešle svoji větu na vstupu, tak ji framework uloží i s návazností na předchozí odpověď, a tak se z toho dokáže učit. Samozřejmě je možné chatbota učit i uměle. S rostoucím počtem naučených vstupů a odpovědí roste schopnost chatbota odpovídat uživateli. (Cox, 2019)

Program na základě vstupu uživatele zvolí nejbližší možnou shodu a tu vrátí uživateli jako odpověď. Tato shoda je určována podle naučené báze dat. (Cox, 2019)

Níže uvedený obrázek popisuje zpracování vstupu a vrácení odpovědi uživateli. Po získání vstupu nadchází jeho zpracování. Před zpracováním může být vstup transformován preprocesory, které mohou například odstranit nepodporované znaky nebo odstranit nežádoucí diakritiku. Framework obsahuje rozhraní logických adaptérů, které vyhodnocují vstup, vrací odpověď a určují jistotu svého zpracování, že je odpověď správná. Odpověď je z množiny možných odpovědí nakonec vybrána podle nejvyšší procentuální jistoty. Následně je odpověď vrácena uživateli.

Obrázek 3: Diagram zpracování vstupu na odpověď



Zdroj: Cox

3.8 Docker

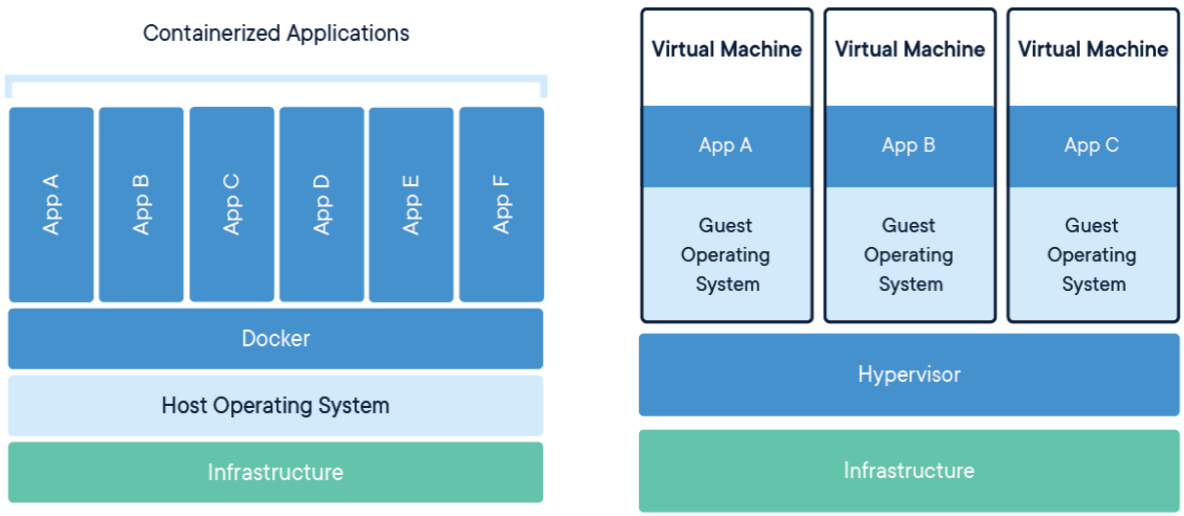
Docker je open source projekt pro automatizaci nasazení aplikací ve formě přenosných a izolovaných kontejnerů. Aplikace běží ve formě kontejnerů, kde každá aplikace by měla být jeden kontejner. Kontejnery mohou běžet nativně na operačních systémech Linuxu a Windows. Windows kontejnery mohou běžet pouze ve Windows prostředí a Linuxové kontejnery mohou běžet na Linuxovém prostředí anebo ve Windows ve virtualizovaném Linux prostředí. Prostředí, na kterém může Docker běžet může být dedikované nebo

virtualizované. Docker je využíván také mezi vývojáři aplikací. Vývojáři pak mohou vyvíjet nezávisle na prostředí a jeho nainstalovaných součástech. (Microsoft, 2019)

3.8.1 Kontejnery versus virtuální stroj

Rozdíl mezi kontejnerizací aplikace a její virtualizaci je možné vidět na obrázku níže. Hlavním rozdílem je, že virtuální stroj (VM) vyžaduje hypervizor, který rozděluje prostředky mezi jednotlivé VM a každé VM má svůj operační systém, který je emulován. Kontejnery oproti tomu běží přímo na hostujícím operačním systému. Výhodou je, že takový přístup značně šetří prostředky, jelikož není potřeba dalšího přepínání kontextu navíc mezi hypervizorem a VM. Kontejnery jsou izolované a využívají kernel hostujícího operačního systému (na Linuxu, na Windows a MacOS je nutné virtualizovat). Ve výsledku se pak kontejner na první pohled může tvářit podobně jako by to byl virtuální stroj, ale odpadá nutnost emulace dalšího operačního systému. (Microsoft, 2019)

Obrázek 4: Srovnání kontejnerizované aplikace a virtuálního stroje



Zdroj: Docker

3.8.2 Jak funguje Docker

Aplikace je spuštěna ve formě kontejneru nebo více kontejnerů. Kontejner je instancí obrazu (image), kde obrazem je myšleno binární soubor ver formátu OVF (Open virtualization format), který se skládá z vrstev a je možné jej verzovat. Obrazy je možné vytvářet na základě jiných obrazů, takový obraz má pak vrstvy obrazu předchozího a vrstvy své. Obraz je pouze ke čtení, samotný kontejner si myslí, že je zapisovatelný a výsledné chování je, že zápis je možný pouze do vrchní vrstvy obrazu. Taková vrstva se pak může stát součástí nové

verze obrazu. Kontejnery je možné mezi sebou propojit virtuální sítí anebo vystavit kontejner samostatně na sít hostujícího počítače. (Microsoft, 2019)

Každý kontejner má různé vlastnosti, které se dělí na (Microsoft, 2019):

- Otevřené porty
- Environmentální proměnné
- Úložiště dat kontejneru (volume či mount point hostujícího počítače)
- Hesla a jiné zabezpečené informace
- Virtuální sítě

3.8.3 Dockerfile

Dockerfile je šablona pro automatizaci vytvoření nového obrazu. Dockerfile má svoji vlastní syntaxi a definuje nastavení obrazu. Jednotlivé kroky jsou potom ekvivalentní vrstvám obrazu. Krok může být samotné nastavení, operace se soubory či příkaz příkazové řádky, který je proveden na dané vrstvě v průběhu tvorby obrazu. Soubor „.dockerignore“ slouží k vyloučení souborů pro build obrazu. (Docker, 2019)

3.8.4 Docker compose

Docker compose je nástroj zajišťující automatizaci běhu více kontejnerových aplikací. Příkladem použití může být kontejnerizace aplikace pomocí Dockerfile, přičemž aplikace vyžaduje přístup do databáze. Docker compose potom definuje společné parametry, jak jsou přístupové údaje do databáze a jednotlivé kontejnery, z nichž jeden je aplikace a druhý je databázový server. Celé nasazení je pak izolováno pro komunikaci mezi kontejnery vlastní virtuální sítí a ven je vystaven pouze port pro veřejný přístup k aplikaci. Docker compose také dokáže převzít environmentální proměnné ze souboru „.env“. (Docker, 2019)

3.9 Postgresql

Postgresql je plnohodnotný objektově-relační databázový systém (ORDBMS). Jedná se o vysoce spolehlivý systém pro správu velkých datových objemů. Systém je open source a váže se k němu velká komunitní podpora. (OVH, 2019)

3.10 Git

Git je distribuovaný verzovací systém pro správu verzí souborů. Vznikl jako nástroj pro vývoj Linuxové jádra. V současnosti oproti jiným řešením má projekt bohatou podporu komerčních nástrojů a je velice využívaným nástrojem při vývoji software. (git, 2019)

Verzování funguje na bázi rozdílů jednotlivých řádků textových souborů. Díky čemuž je možné zachovat celou historii souboru a zároveň být šetrný na diskový prostor. U verzování binárních souborů není možné porovnávat na textové bázi a je tedy vždy jako změna uložen celý soubor znovu. V důsledku verzování velkých binárních souborů mít repositář signifikantní dopad na diskový prostor. Všechny soubory jsou verzované samostatně, ale skupina změn se sdružuje (commit). Commity jsou sdruženy do větví a větve do repositáře. (git, 2019)

3.11 HTML a CSS

HTML neboli HyperText (odkaz) Markup Language je značkovací jazyk pomocí kterého jsou tvořené webové stránky. HTML se používá k vytvoření struktury a definování obsahu webové stránky a CSS (kaskádové styly) slouží k definování stylů. Ve výsledku je tedy pomocí HTML napsána struktura webové stránky a pomocí CSS je nadále upravován vzhled samotné struktury a obsahu. (w3c, 2019)

3.12 Javascript

Javascript je používan jako skriptovací jazyk sloužící k implementaci dynamicky aktualizovaného obsahu na webové stránce. Výsledný kód je obsažen v HTML kódu webové stránky a je interpretován přímo u klienta webovým prohlížečem. Javascript se v průběhu let vyvinul v programovací jazyk s použitím na serveru jako backend pomocí NodeJS interpreteru. Dnes je javascript populární zejména díky možnosti programovat frontend a backend jedním programovacím jazykem. (Acellere, 2018)

3.12.1 Ajax

Ajax (Asynchronous Javascript and XML) je asynchronní zpracování javascriptového kódu. Využití Ajaxu bývá po načtení celé webové stránky, kde uživatel při práci se stránkou

potřebuje obnovit obsah části webové stránky. Při použití Ajaxu je načten jen obsah stránky, který je potřeba. Bez jeho použití by uživateli byla obnovena stránka celá. Ve výsledku se pak jedná o schopnost Javascriptu zavolat na serveru nějaký dotaz API či skript bez nutnosti čekání na odpověď. Uživatel pracuje se stránkou bez čekání na odpověď a stránka může v průběhu před obdržením odpovědi provádět další operace. Při obdržení odpovědi je provádění hlavního kontextu stránky ve vhodné chvíli pozastaveno a je zavolán callback, který provádí definovanou operaci. (itnetwork, 2015)

4 Vlastní práce

Realizace diplomové práce vyžaduje pochopení základních principů vývoje software. Velice nutné je vymezit samotné zadání a hranice daného systému, který bude vyvíjen.

Pro zjištění možných cest kudy se ve vývoji ubrat na cestě k implementaci aplikace je vhodné udělat analýzu aktuálních řešení na trhu, která by měla kromě možností poskytnout i slepá místa, kterým bude nutné či vhodné se vyhnout.

Samotná analýza aktuálních řešení na trhu by měla pomoci k pochopení současných řešení dané problematiky a zdali je vůbec schůdné aplikaci řešící problematiku chatbotů vyvíjet „od zelené louky“ nebo použít již hotové řešení či framework.

Pro každou volbu je nutné uvést subjektivní i objektivní důvody zvoleného řešení spolu s výhodami, nevýhodami a problémy, které zvolené řešení přináší nebo mohou přinést.

Architektura aplikace by měla být na high-level bázi popsána pro snadné pochopení použitých komponent či jejich funkcích v aplikaci.

Kód aplikace je vyvíjen mimo diplomovou práci a samotná diplomová práce by měla obsahovat pouze zajímavé problematiky při implementaci či útržky kódu nezbytné pro pochopení postupu při vypracování analýzy a implementaci.

4.1 Analýza aktuálních řešení na trhu

Na vývoj aplikace je nutné pohlížet z pohledu zadavatele i dodavatele. Na trhu existuje celá řada řešení různých typů a různých TCO (celkové náklady spojené s vlastnictvím). Z množiny řešení je potřeba separovat řešení přípustná a ta potom zvažovat pro finální zadání a implementaci.

Cílem diplomové práce je vyvinout funkční řešení Q&A chatbota dle zadání práce. Samotné cíle však nestačí pro implementaci, a proto se po provedení analýzy bude možné rozhodnout jakými způsoby daného cíle dosáhnout.

Cíl diplomové práce spolu s analýzou by měly poskytnout dostatečné podklady pro výběr technologií, pomocí kterých bude aplikace vyvinuta.

4.1.1.1 Kategorizace řešení na trhu

Řešení na trhu je vhodné kategorizovat do různých kategorií podle iniciálních požadavků, složitosti a rozmanitosti řešení.

Dělit lze tyto řešení dle kategorií:

- Provoz
    - On-premise (u zákazníka)
    - On-premise cloud-bound (u zákazníka, ale vázané na cloudové řešení)
    - Cloud (v cloudu)
- Otevřenost zdrojových kódů
    - Open-source
    - SDK open-source (server closed-source)
    - Closed-source
- TCO, typ závazku
    - Zdarma, bez poplatků
    - Vendor lock-in s měsíčními poplatky
- Typ chatbota
    - Command-driven (řízený příkazová logika)
    - Konverzační
    - Q&A (otázky a odpovědi)
- Typ použité technologie chatbota
    - NLP (Neurolingvistické programování)
    - Neurální sítě / Strojové učení
    - Programatický
- Vývojová platforma
    - GUI / Web GUI
    - Python
    - NodeJS
    - C#
    - Java

4.1.1.2 Nejčastější řešení na trhu

V dávných dobách před popularizací chatbotů byly dostupné akorát tzv. Programatické chatboty, které byly většinou tvořeny nadšenci či se snažili konverzaci jen předstírat.

Nynější stav na trhu je již jiný. Metodiky a nástroje pro vytvoření funkčního řešení se rozvinuly díky komunitním projektům.

Komunitní řešení

Komunitní řešení jsou často výtvorem nějaké zájmové skupiny a je jich mnoho. Většinou se jedná o frameworky a hlavní motivací je nadšení, rozvinout portfolio a získat prestiž. Některá řešení jsou dále rozvíjena a popularizují se, jiná však upadnou v zapomnění. Všechny bývají zdarma a open-source. Je zde možné nalézt všechny možné typy chatbotů, dokonce i ty nezmíněné.

Problémem u takového řešení bývá najít slibný komunitní projekt, který má dostatečnou podporu. Dále takové řešení bývá implementováno pouze v jednom programovacím jazyce a je tedy obtížné vybrat dobré řešení, pokud je vývojář omezen vývojovou platformou. Nejčastěji bývají tyto projekty psány v programovacím jazyce Python, který je známý především v komunitě lidí zabývající se data miningem a strojovým učením.
Zde je nutné mít štěstí na vyhovující, dobře podporovanou platformu a schopnosti chatbota vyvinout.

Komerční řešení

Komerční řešení se snaží nabídnout zákazníkům funkční řešení většinou jako konverzační typy chatbotů. Hlavní motivace je v tomto případě zisk a vendor lock-in. Řešení bývají dobře dokumentována, a dokonce jejich SDK je dostupné pro různé vývojové platformy. Nasazení bývá možné v cloudu i on-premise, ale většinou je on-premise řešení stejně závislé na cloudu, kde na pozadí jsou dotazy zpracovány a zákazník si vyvíjí pouze klienta. Open-source je pouze část s SDK a ukázkovými příklady. Zákazníkovi je toto nabídnuto jako služba, kde okamžitě spadá do vendor lock-inu. Samotná implementace pak je pomocí zmíněného SDK nebo v GUI někde na webu. Chatboti bývají řešeni kombinací NLP a strojového učení.

Takovému řešení je ale nejlepší se vyhnout, pokud ho sami neprodáváte.

4.2 Analýza a rozbor zadání

4.2.1 Vymezení zadání dle kategorizace na trhu

Díky analýze kategorizace na trhu bylo možné zjistit okolnosti a podmínky prostředí. Dalším postupem je ze zjištěných poznatků vymezit rozhodnutí požadavků, ze kterých potom bude vycházet samotný vývoj.

4.2.1.1 Provoz

U provozu aplikace nezáleží, kde chatbot bude provozován, ale je nutné mít možnost výběru. On-premise řešení je vždy možné nasadit do cloudu. Také je důležité mít kontrolu nad daty. Dále znát polohu země, kde jsou data uložena a mít možnost je kdykoliv nenávratně smazat, exportovat či zálohovat.

4.2.1.2 Otevřenost zdrojových kódů

Je nutné nespoléhat se na closed-source řešení a využít řešení založené na open-source nebo vyvinout vlastní.

4.2.1.3 TCO, typ závazku

Samozřejmostí je, že každý by chtěl TCO co nejnižší či nulové. V případě této diplomové práce nepřipadají placená řešení s vendor lock-inem v úvahu.

4.2.1.4 Typ použité technologie chatbota

Q&A se soustředí na pokládání otázek a jejich zodpovězení. V případě použití programatického chatbota by musela být otázka položena přesně nebo vybrána z výčtu možností zobrazených uživateli.

Ideální by bylo, kdyby uživatel znalý okruhů či témat otázek položil danou otázku v kontextu známé otázky. U takové otázky je nutné najít shodu se známou otázkou. Shoda může být nalezena pomocí NLP či NLP s kombinací programatického přístupu pro usnadnění a ulehčení nutnosti znalosti okruhů či témat otázek.

Samozřejmě je možné použít i strojové učení, tam je pouze nutné vybrat, zdali se pro danou problematiku více hodí NLP či strojové učení. Takové rozhodnutí se může odvíjet od dostupných řešení či subjektivního dojmu, případně i rozsahu složitosti realizace takového řešení.

4.2.1.5 Vývojová platforma

Plnohodnotného chatbota je možné vytvořit v GUI pomocí logických operací, které poskytuje dané řešení. Jednalo by se o programování, ale ne o použití programovacího jazyka. Takové řešení bude mít jen omezené možnosti při neplaceném použití a uvrhne uživatele do vendor lock-inu.

Co se vývojových platforem z pohledu programovacích jazyků týče, tak autor má znalost programovacího jazyka Java a technologií s ním spojených. Zadavatel (universita) by tento jazyk také preferoval.

Komunita a její open-source projekty jsou však nyní převážně tvořeny výhradně na platformě programovacího jazyka Python a někdy také NodeJS. Strojové učení a data mining jsou silná stránka Pythonu a technologií s ním spojených. Právě pro tyto výhody je tento jazyk často používán.

4.2.2 Zadání

Chatbot by měl být schopen odpovídat na položené otázky z naučené báze dat. Dále musí být nezávislý na použitém jazyku. Většina platforem konverzačních chatbotů má s multi-jazyčností problém. Chatbot bude typu Q&A, kde je mimo jiné jednodušší docílit nezávislosti na daném jazyce.

Uživatel zadá otázku v kontextu naučených otázek a chatbot by měl být schopen k dané otázce vrátit uživateli správnou odpověď. Tato přesnost bude stanovena na 50 %, nicméně ji lze jednoduše přenastavit v případě potřeby.

Uživateli by měl být poskytnut snadný přístup k rozhraní aplikace, třeba přes webovou stránku. Dalšími typy přístupů bude stránka, kterou je možné vložit na externí webové stránky a přímá komunikace přes REST API.

4.2.3 Vybrané řešení

Na internetu existuje celá řada použitelných typů řešení, které byly probrány v této kapitole. Je však nutné vybrat takové řešení, na kterém bude reálně možné chatbota postavit. Řešení bude on-premise v možností přesunutí do cloudu. TCO bude samozřejmě nulové a snaha bude najít kvalitní open-source framework, který má dobrou komunitní podporu.

Chatbot bude typu Q&A, ale pro tyto účely je možné použít konverzační framework, který je multi-jazyčný či nezávislý na jazyku. Konkrétní forma zpracování bude NLP. Pro tyto účely byl vybrán konverzační framework Chatterbot.

Za zmínku stojí také řešení, která nebyla vybrána pro nadcházející implementaci. Konkrétní výběr byl mezi frameworky Chatterbot, Rasa, Tensorflow. Rasa framework je řešení připravené k použití, kde je však potřeba programovat chatbota speciální syntaxí. Tensorflow je světoznámá knihovna používaná pro strojové učení, ale samotné zhotovení takové aplikace by bylo složitější s nejistým výsledkem.

4.3 Návrh a Implementace

Pro chatbota byl vybrán programovací jazyk Python verze 3.7. Bylo by možné vyvíjet i ve starší verzi pythonu 2 pro lepší kompatibilitu s různým runtime prostředím, nicméně vzhledem k použití některých python 3 specifik tuto zpětnou kompatibilitu nelze použít.

Jako dependency management byl použit pipenv s využitím projektového virtualenv.

4.3.1 High level architektura komponent

Uživateli by mělo být poskytnuto rozhraní pro interakci s chatbotem. Jako nejvhodnější se jeví jako webové rozhraní v podobě jednoduché stránky, kde bude uživatel moci odesílat a přijímat zprávy, tedy chatovací okno.

Webové stránky budou sestávat ze statického a dynamického obsahu. Statickým obsahem je myšleno čisté HTML, CSS a JS soubory. Dynamickým obsahem pak jsou sestavy obsahu webových stránek složeného z obsahu šablon frameworku Django.

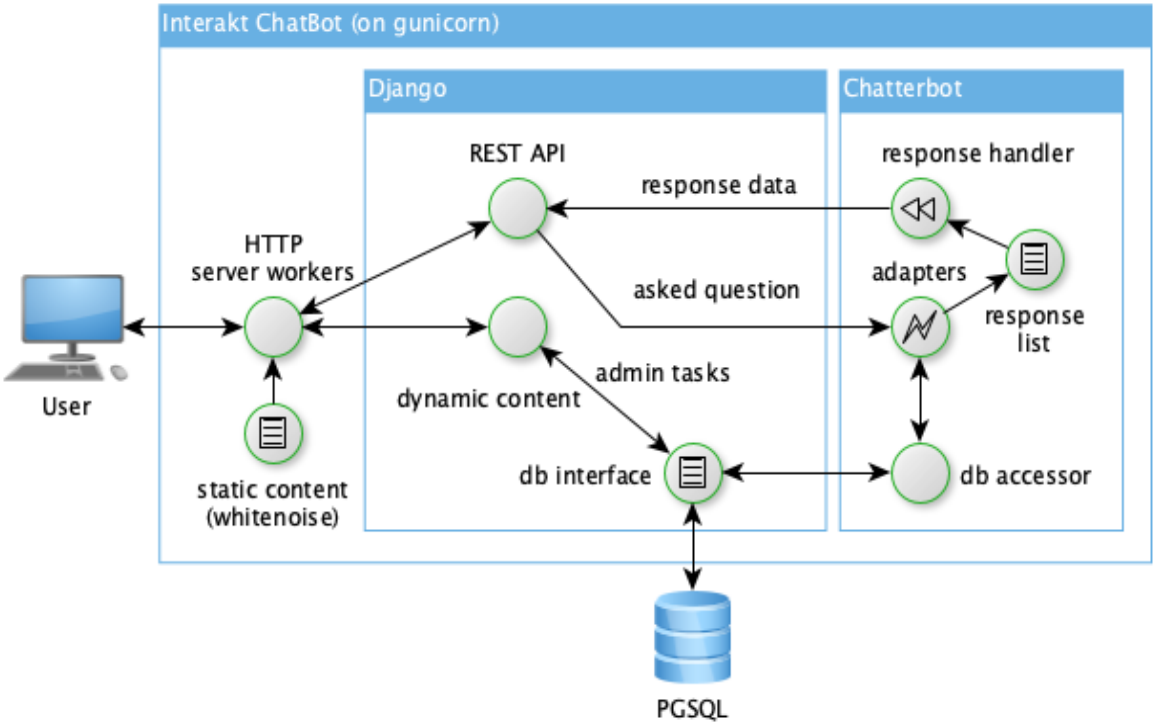
Framework Django bude také použit na přístup k databázi, vystavění REST API a administraci aplikace.

Jako databázový server lze použít podporované databáze NO-SQL či SQL. Pro potřeby této aplikace bude vybrán PostgreSQL.

Aplikace Interakt chatbot bude stavět na chatbot frameworku ChatterBot, kde bude využita stávající architektura a na níž bude stavěno.

Na níže zobrazeném obrázku lze vidět propojení jednotlivých komponent. Schéma znázorňuje, která komponenta zajišťuje jednotlivý krok při toku uživatelského requestu až po jeho response. Schéma však neřeší, jakým způsobem komponenta danou problematiku řeší.

Obrázek 5: High level architektura použitých komponent chatbota Interakt



Zdroj: Vlastní zpracování

4.3.2 Struktura projektu

Projekt má jednoduchý dependency management skládající se z jedno kořenového modulu. Není tedy potřeba dependency (závislosti) dělit mezi jednotlivé moduly, jako by tomu bylo třeba u většího projektu.

Závislé balíčky z pypi.org:

- django
    - Hlavní knihovna frameworku Django
- chatterbot
    - Hlavní knihovna frameworku ChatterBot
- psycopg2-binary
    - Podpůrná knihovna pro Django a ChatterBot řešící podporu pro PostgreSQL
- gunicorn
    - Python webový server
- whitenoise
    - Provider statických souborů webové služby
- python-levenstein
    - Knihovna pro podporu algoritmu Levensteinovy vzdálenosti

4.3.3 Django

Pro výběr frameworku webových aplikací v Python byl vybrán framework Django. Ač je viditelně složitější pro neznalého programátora, tak nabízí komplexní robustnost, která po osvojení potřebných know-how přináší jisté výhody.

Django ve svém základu řeší klasické problematiky dnešních webových aplikací vyvíjených v jazyce Python. Aplikace potřebuje načítat a ukládat data, většinou do databáze či databázového souboru. Aplikace také potřebuje komunikační rozhraní mezi frontendem a backendem, či jen pro přístup třetích stran. Dále framework řeší routování webových URL na backend a nabízí šablonovací systém na frontendovou část aplikace.

4.3.3.1 Nastavení Django frameworku

Základní nastavení je provedeno v souboru „settings.py“. Oproti jiným platformách lze tedy nastavení provádět programaticky, protože se jedná o soubor formátu zdrojového kódu Python. Lze tak snadno řešit problémy, se kterými se každá aplikace při vývoji a deploymentu na různá prostředí může setkat.

V případě chatbota Interakt se takto řeší hlavně problematika statických souborů. Dále zda je aplikace deployována na vývoj či produkční prostředí, ale také zda je aplikace izolována v Docker containeru.

4.3.3.2 PostgreSQL

Django a ChatterBot frameworky defaultně využívají NOSQL databáze MongoDB, avšak byla zvolena SQL databáze PostgreSQL navzdory popularitě MongoDB mezi vývojáři. Tato volba byla subjektivní, ale i tak se spíše najde více administrátorů, kteří si spíše poradí s SQL databázemi a jejich správou. Na nepoužité databáze nebyla zkoumána podpora u použitých frameworků a databázové platformy byly vyřazeny z různých důvodů i před zkoumáním kompatibility. Většinou tomu tak bylo těžkopádností daného databázového řešení spolu s potřebnými licencemi.

4.3.4 ChatterBot

ChatterBot je Python knihovna, která usnadňuje generovat automatizované odpovědi na uživatelský vstup. ChatterBot je možné použít jako framework pro implementaci řešení Q&A chatbota Interakt.

Jedná se o dobře dokumentovaný framework, který má komunitní podporu. Původně byl vyvíjen jako projekt studenta zahraniční univerzity. Samozřejmě je tomu tak i nyní, avšak za větší podpory komunity.

Tato knihovna se specializuje na konverzační chatboty a je velice jednoduché pomocí pár řádků kódu vytvořit základního chatbota, který bude funkční dle implementovaných případů užití. Vzhledem k tomu, že cílem diplomové práce je vyvinout Q&A chatbota, tak je možné namítnout, že se framework nehodí pro implementaci, protože je to typově konverzační chatbot.

Pravda je ovšem taková, že framework má vymyšlený systém vrstev, na které lze pohlížet jako na interface k řešení procesu A-Z. Tato skutečnost z něj činí mocný framework, nad kterým lze implementovat vlastní vrstvy. Tyto vrstvy budou implementovat dané rozhraní.

Minimálně je tedy možné využít rozhraní jednotlivých vrstev. Nabízí se však mnohem širší sady hotových funkcí a vrstev, které je možné použít pro řešení vyvíjené na míru. Jmenovitě se určitě jedná o databázové rozhraní a systém rozhodování.

Framework dále nabízí implementace různých algoritmů používaných ve strojovém učení, což skutečně bude znamenat usnadnění a zkrácení doby vývoje.

4.3.5 Nastavení Chatbota

Chatterbot framework bylo potřeba nastavit a upravit pro potřeby aplikace Interakt Q&A chatbot. Úpravami je myšleno využití stávající architektury a rozhraní a dle potřeby implementovat kupříkladu vlastní logické adaptéry, které řeší rozpoznání vstupu a vrácení odpovědi uživateli. Dále také bylo potřeba navrhnout a implementovat vlastní trenéry dat, které zajišťují převod vstupních dat a jejich uložení do databáze, ve které je naučená báze dat.

4.3.6 Trenéři dat

Základní problém rozdílnosti frameworku Chatterbot je už v samotném učení dat. Vestavěný trenér dat je uzpůsoben na učení konverzací. To v důsledku znamená logické propojení, které ústí ve spojový seznam. Data jsou propojena stylem, kde aktuální věta má vazbu na větu předcházející. V konverzačním modelu je toto vhodné pro znalost průběhu konverzace.

Vizualizace vazeb seznamového konverzačního trenéra:

- u konverzačních dat:
  A → B → C → D

- u Q&A dat:
  Q → A → Q → A

Použijeme-li takového trenéra k učení Q&A, tak každá otázka bude mít vazbu na předcházející odpověď. To je v důsledku špatně, protože daná vazba nemá opodstatnění a následně vede ke špatným odpovědím uživateli.

- Vizualizace vazeb seznamového Q&A trenéra:
  Q → A
  Q → A

4.3.6.1 Q&A List Trainer

Algoritmicky lze „Q&A List Trainer“ navrhnout poměrně jednoduchým způsobem. Je potřeba vždy zachovat vazbu mezi otázkou a odpovědí tak, aby při položení otázky a jejím následným rozpoznáním bylo možné vyhledat na ni odpověď a tu vrátit uživateli.

Formát vstupního souboru Q&A dědí tento trenér od konverzačního seznamového trenéra. Tedy co řádek to věta. Rozdílem u Q&A seznamového trenéra je však ten, že v souboru je rozlišováno párově, zda se jedná o otázku či odpověď. Základní rozlišení počítá s tím, že na prvním řádku je otázka a za ní následuje odpověď na danou otázku. Následně se pořadí opakuje až do konce souboru.

Samotné rozpoznání je algoritmicky řešeno čítačem přečtených vět, kde každá sudá věta (otázka) nemá vazbu na předchozí větu a každá lichá věta (odpověď) má vazbu na předchozí větu. To ve výsledku znamená to, že je dodržena vazba mezi otázkou a odpovědí. Pro zjištění sudé a liché věty je použito modulo.

```
conversation_count % 2 == 0
```

4.3.7 Adaptéry

Interakt Q&A chatbot vyžaduje ke své funkčnosti celkem tři speciální logické adaptéry:

- No data adapter
- No input adapter
- Best match with low confidence adapter

4.3.7.1 No data adapter

V iniciálním stavu, kdy chatbot nemá žádná data se chová iracionálně, tedy jinak, než se od něj očekává. Kupříkladu když uživatel odešle vstup, tak je mu zpátky navrácen. Za stávajících podmínek, pokud by byl implementován chatbot bez frameworku, tak by se tento speciální stav dal řešit úpravou kódu, nicméně framework pro podobné případy přímo nabízí rozhraní logických adaptérů, kde je možné danou funkcionalitu aplikovat.

Racionalizace chatbota je řešena adaptérem, který zjišťuje, zda má chatbot k dispozici naučená nějaká data. Uživateli je v případě žádných naučených dat vrácena odpověď: „Chatbot nemá k dispozici žádné data v DB.“.

4.3.7.2 No input adapter

Další iracionální stav nastává, když uživatel zadá prázdný vstup. Toto by bylo možné ošetřit například javascriptem na frontendové části aplikace, nicméně to neřeší problém, když je prázdný dotaz odeslán přes REST API. Je tedy vhodné řešit tento problém na backendové části aplikace pomocí adaptéru. V případě, že uživatel zadá prázdný vstup, tak je mu následovně navrácena odpověď: „Položte prosím otázku.“.

4.3.7.3 Best match with low confidence adapter

Hlavní adaptér řešící zpracování vstupů a vrácení odpovědi uživateli funguje na principu nejlepší shody. Adaptér má nastavitelnou procentuální hranici, od které považuje nalezené shody za důvěryhodné. Každé shodě je přiřazena váha v oboru hodnot <0, 1>. Po zpracování je vybrána shoda s nejvyšší váhou, kterou pak přímo představuje procentuální pravděpodobnost shody.

Tímto způsobem je jednoduše ke vstupu uživatele nalezena otázka v databázi a následně je uživateli vrácena odpověď na tuto otázku. Pokud adaptér nepovažuje žádnou ze shod za důvěryhodnou, tak vrací defaultní odpověď: „Nebylo možné správně determinovat položenou otázku. Zkuste ji položit jiným způsobem.“.

4.3.8 Levenshteinova vzdálenost

O samotné rozpoznávání shod vstupu s otázkami v bázi dat se stará algoritmus „Levenshteinova vzdálenost“. Implementace tohoto algoritmu je importována pomocí

knihovny z balíčkovacího systému pypi: „python-levenshtein“. Implementace daného algoritmu pak nabízí python rozhraní pro práci s nativní knihovnou jazyka C (cpython). Algoritmus „Levenshteinova vzdálenost“ byl vybrán pro jeho přímou podporu frameworkem Chatterbot a také pro jeho použití vyhledávačem Google.

Pokud by se pro porovnávání výrazů používala přímá shoda slov, tak by bylo velice obtížné nalézt shodu. Přímá shoda slov není připravena na jiné velikosti písmen, změnu skloňování a diakritiky. Z tohoto důvodu existují algoritmy, jako je Levenshteinova vzdálenost.

4.3.8.1 Příklad výpočtu Levenshteinovy vzdálenosti

Mějme dva výrazy:

Výraz A: „červená karkulka“

- délka výrazu: 16 znaků

Výraz B: „červ a kulka“

1. délka výrazu: 12 znaků

Výsledná hodnota Levenshteinovy vzdálenosti odpovídá minimálnímu počtu operací potřebných pro změnu výrazu A, tak aby byl ekvivalentní výrazu B. Tabulka níže demonstruje potřebné operace pro ekvivalenci obou výrazů a průběh transformace výrazu A na výraz B. Z tabulky je také možné odvodit Levenshteinovu vzdálenost, která je pro dané výrazy a minimální provedených počet operací rovna 5.

Tabulka je pro lepší čitelnost barevně rozložena. Modrá barva znaku výsledného výrazu po operaci značí aktuální ukazatel, na kterém byla provedena operace. Pokud je znak smazán, tak ukazatel zůstává na původním umístění, jelikož se v další operaci liší následující znak a původní, na kterém by jinak byl ukazatel byl smazán.

Tabulka 2: Tabulka provedených operací nad výrazem A pro ekvivalenci výrazu A a B.

Pozice znaku	Znak	Operace nad znakem	Výsledný výraz po operaci
1	č	beze změn	červená karkulka
2	e	beze změn	červená karkulka
3	r	beze změn	červená karkulka
4	v	beze změn	červená karkulka

5	e	**smazat**	červná karkulka
6	n	**smazat**	červá karkulka
7	á	**smazat**	červ karkulka
8	„ “	beze změn	červ karkulka
9	k	**smazat**	červ arkulka
10	a	beze změn	červ arkulka
11	r	**nahradit znakem „ “**	červ a kulka
12	k	beze změn	červ a kulka
13	u	beze změn	červ a kulka
14	l	beze změn	červ a kulka
15	k	beze změn	červ a kulka
16	a	beze změn	červ a kulka

Zdroj: Vlastní zpracování

Levenshteinovu vzdálenost je možné lépe vizualizovat pomocí tabulkového porovnání výrazu A a B s transformací výrazu A na výraz B pomocí pravidel algoritmu popsaných v teoretické části práce. V tabulce níže je proveden výpočet algoritmem s výsledkem Levenshteinovy vzdálenosti 5.

Tabulka 3: Vizualizovaný výpočet Levenshteinovy vzdálenosti

		č	**e**	**r**	**v**		**a**		**k**	**u**	**l**	**k**	**a**
	0	1	2	3	4	5	6	7	8	9	10	11	12
č	1	0	1	2	3	4	5	6	7	8	9	10	11
e	2	1	0	1	2	3	4	5	6	7	8	9	10
r	3	2	1	0	1	2	3	4	5	6	7	8	9
v	4	3	2	1	0	1	2	3	4	5	6	7	8
e	5	4	3	2	1	1	2	3	4	5	6	7	8
n	6	5	4	3	2	2	2	3	4	5	6	7	8
á	7	6	5	4	3	3	3	3	4	5	6	7	8
	8	7	6	5	4	3	4	3	4	5	6	7	8
k	9	8	7	6	5	4	4	4	3	4	5	6	7
a	10	9	8	7	6	5	4	5	4	4	5	6	6
r	11	10	9	8	7	6	5	5	5	5	5	6	7
k	12	11	10	9	8	7	6	6	5	6	6	5	6
u	13	12	11	10	9	8	7	7	6	5	6	6	6
l	14	13	12	11	10	9	8	8	7	6	5	6	7
k	15	14	13	12	11	10	9	9	8	7	6	5	6
a	16	15	14	13	12	11	10	10	9	8	7	6	5

Zdroj: Vlastní zpracování

4.3.8.2 Převod výpočtu Levenshteinovy vzdálenosti na procentuální vyjádření

Levenshteinovu vzdálenost je možné jednoduchým výpočtem převést na procentuální vyjádření.

Výraz A má délku 16 znaků, výraz B 12 znaků a Levenshteinova vzdálenost je rovna 5.

A=16
B=12
levDis=5

levDisPct = (max(A,B) - levDis) / max(A,B)

levDisPct = (16 – 5) / 16 = 11 / 16 = 0.6875

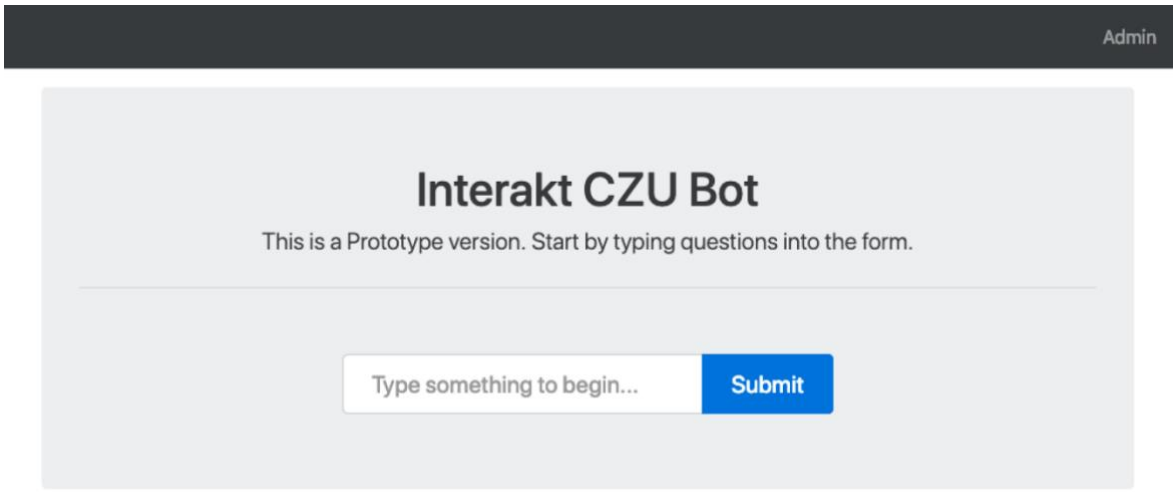
Výsledek je v oboru hodnot <0,1> a pro získání klasického procentuálního vyjádření stačí výsledek vynásobit 100. Procentuální vyjádření je tedy 68,75 %. Toto procentuální vyjádření je využito chatbotem pro porovnávání shody vstupu uživatele s jednotlivými záznamy báze dat.

4.4 Web GUI

Základní webové rozhraní je postaveno na ukázkových příkladech frameworku ChatterBot. Stavba stránky se na první pohled změnila minimálně. Pro následné integrování chatbota do stránek je žádoucí vytvořit stránku vlastní, třeba jako visuální interaktivní panel (widget).

Primární webová stránka s chatovacím oknem:

Obrázek 6: Primární chatovací stránka s chatovacím oknem



Zdroj: Vlastní zpracování

4.4.1 Návrh webového rozhraní

Webové rozhraní je postavené na frameworku Bootstrap 4 a JQuery. Pro takto jednoduché webové rozhraní je to dostatečné.

Grafické rozraní obsahuje textové pole pro uživatelský vstup a tlačítko pro odeslání uživatelského vstupu. Uživatelským vstupem je myšlena otázka, která bude chatbotovi položena.

4.4.2 Další funkcionality a problémy

Při použití frameworku ChatterBot verze 0.8.7 vše fungovalo plynule. Po přechodu na novější verzi 1.0.4 trvá první odezva déle kvůli inicializaci interní knihovny NTLK (Natural Language ToolKit), která by měla zajistit lepší preciznost při zpracování uživatelského vstupu a jeho rozpoznávání.

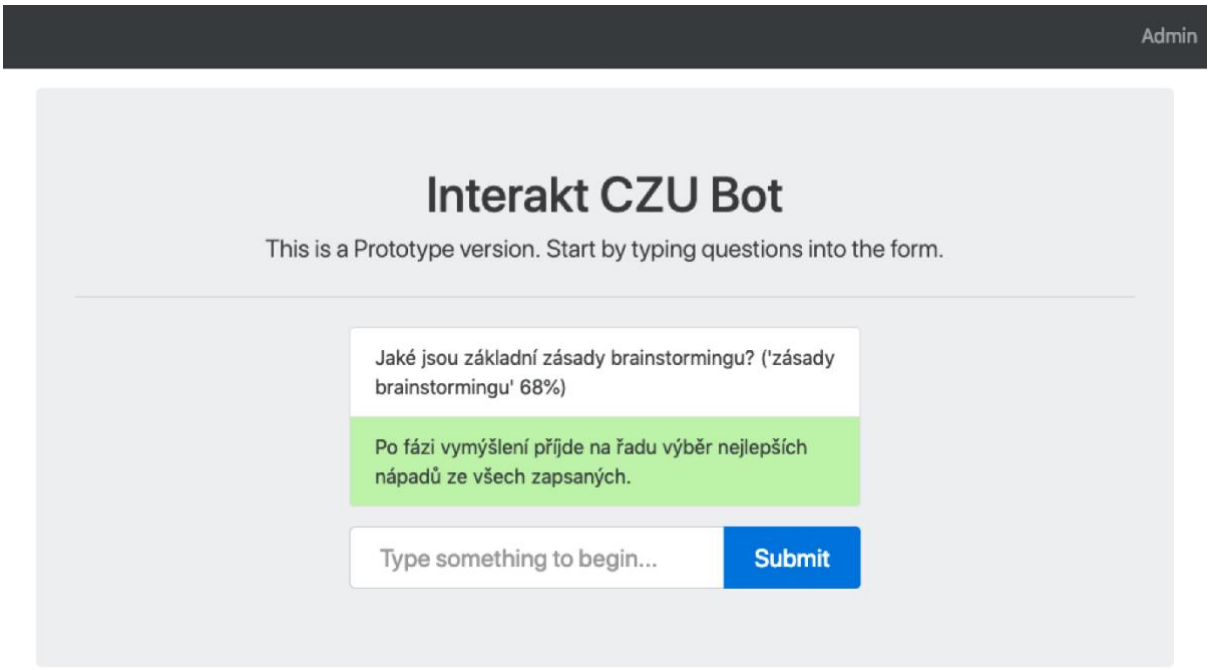
Do grafického rozhraní byla tedy implementována brzda na odesílání požadavků a zároveň indikace odeslané zprávy s čekáním na odezvu. Jakmile je požadavek odeslán, tlačítko je zablokováno a změní barvu a tím dá uživateli najevo, že byla zpráva odeslána a zároveň, že požadavek ještě nebyl zpracován. Request je odeslán přes javascript ajaxové volání metod JQuery. Pokud by odeslání selhalo je tlačítko odemknuto uživateli. V případě zpracování

requestu uživatele a obdržení response je tlačítko také odblokováno a je uživateli zobrazena odpověď.

4.4.3 Příklad uživatelské interakce

Na obrázku níže je příklad uživatelské interakce s chatbotem:

Obrázek 7: Uživatelská interakce s chatbotem



Zdroj: Vlastní zpracování

Na obrázku je možné pozorovat uživatelskou interakci s chatbotem. Uživatel zadal dotaz: „zásady brainstormingu“. Chatbot tento dotaz zpracoval a s 68 % pravděpodobností našel odpovídající dotaz v bázi dat. Nalezena byla otázka: “Jaké jsou základní zásady brainstormingu?“ a uživateli vrácena odpověď na tuto otázku: “Po fázi vymýšlení přijde na řadu výběr nejlepších nápadů ze všech zapsaných.“

Vzhledem k tomu, že JSON data získaná z volání jsou strukturalizovaná, je možné stránku upravit i pro výpis bez dodatečných informací.

4.5 REST API

Interakt chatbot vystavuje REST API jako rozhraní jak pro následnou možnou samostatnou komunikaci, tak i pro vestavěné Web GUI.

4.5.1 Příklady volání REST API

4.5.1.1 Informace o chatbotovi

Informace o chatbotovi je možné získat HTTP voláním metodou GET na url cestu:“/api/chatterbot/“. Volání nevyžaduje žádné zvláštní HTTP headery, ani request body.

Informace o chatbotovi request:

```
GET /api/chatterbot/
cache-control: no-cache
Postman-Token: 8b441f27-7299-4c04-ac4e-62e4182ac07f
User-Agent: PostmanRuntime/7.6.1
Accept: */*
Host: 0.0.0.0:8990
accept-encoding: gzip, deflate
```

Informace o chatbotovi response:

```
HTTP/1.1 200
status: 200
Server: gunicorn/19.9.0
Date: Fri, 15 Mar 2019 14:37:38 GMT
Connection: close
Content-Type: application/json
X-Frame-Options: SAMEORIGIN
Content-Length: 62

response body:
{
   "name": "Interakt",
   "qas_count": 28,
   "read_only": "no_learn"
}
```

Response vrací JSON se jménem chatbota a počtem párů otázek a odpovědí v databázi. Klíč „read_only“ pak indikuje, zda se chatbot může učit na základě nových konverzací s uživateli.

4.5.1.2 Položení otázky

Otázku chatbotovi lze položit odesláním HTTP volání metody POST na url cestu:“/api/chatterbot/“. Volání vyžaduje „Content-type“:“application/json“ v HTTP headeru, a JSON v body requestu.

Samotný JSON v body requestu, pak vyžaduje value s klíčem:“text“, kde value je otázka, která bude chatbotovi položena.

Položení otázky request:

```
POST /api/chatterbot/
Content-Type: application/json
cache-control: no-cache
Postman-Token: 3517af78-9e79-4196-987d-d9d526a193b5
User-Agent: PostmanRuntime/7.6.1
Accept: */*
Host: 0.0.0.0:8990
accept-encoding: gzip, deflate
content-length: 36

response body:
{
        "text":"zásady brainstormingu"
}
```

Položení otázky response:

```
HTTP/1.1 200
status: 200
Server: gunicorn/19.9.0
Date: Fri, 15 Mar 2019 14:40:39 GMT
Connection: close
Content-Type: application/json
X-Frame-Options: SAMEORIGIN
Content-Length: 358

{
  "id": null,
  "text": "Mluvit by měl v jednom okamžiku pouze jeden.",
  "search_text": "zásady brainstormingu",
  "created_at": "2019-03-15T14:40:39.793393",
  "conversation": "",
  "in_response_to": "Jaké jsou základní zásady brainstormingu?",
  "search_in_response_to": "",
  "persona": "bot:Interakt",
  "tags": [],
  "confidence": 0.68
}
```

V response jsou důležité hodnoty:

- text
    - Odpověď na otázku.
- search_text
    - Původní otázka uživatele.
- in_response_to
    - Otázka v bázi dat, která byla rozpoznána chatbotem na původní otázku uživatele.
- confidence
    - Pravděpodobnost v oboru hodnot <0,1>, že je daná otázka rozpoznána správně.

4.6 Testovací data

Testovací data byla použita ze školního software Moodle, předmětu Interakce člověk počítač. Bylo potřeba přijít na nejvhodnější způsob získání testovacích dat a jejich případnou transformaci.

4.6.1 Analýza

4.6.1.1 Moodle

U Moodle je možné data získat přímo z databáze, avšak vzhledem k nepřístupnosti k databázi a neznalosti datového modelu je tento přístup nežádoucí i z pohledu množství obsažených a citlivých dat.

4.6.1.2 HTML

Jako další možnost je možné použít HTML extrakt testové zkoušky daného předmětu. Bohužel vzhledem ke struktuře dat je patrné, že není možné rozpoznat váhy správných odpovědí vůči špatným. A tak by v tomto případě osoba provádějící extrakci ji mohla rovnou dělat ručně nebo by bylo nutné ručně tyto vyextrahované otázky projít a vyřadit špatné odpovědi.

Příklad extraktu z Moodle v HTML:

```
<!-- question: 1234987  name: Otázka č. 3 -->
<div class="question">
   <h3>Recept na vařenou vodu</h3>
   <p class="questiontext">
   <p>Co je potřeba k uvaření vody?></p>
   <ul class="multichoice">
      <li><input name="quest_1234987" type="checkbox" value="&lt;p&gt;Voda.&lt;/p&gt;"/>
         <p>Voda.</p></li>
      <li><input name="quest_1234987" type="checkbox" value="&lt;p&gt;Nevím.&lt;/p&gt;"/>
         <p>Nevím.</p></li>
      <li><input name="quest_1234987" type="checkbox"
               value="&lt;p&gt;10 programátorů a jedna cvičená opice.&lt;/p&gt;"/>
         <p>10 programátorů a jedna cvičená opice.</p></li>
      <li><input name="quest_1234987" type="checkbox" value="&lt;p&gt;Sůl a pepř.&lt;/p&gt;"/>
         <p>Sůl a pepř.</p></li>
      <li><input name="quest_1234987" type="checkbox" value="&lt;p&gt;Přivést vodu k
varu.&lt;/p&gt;"/>
         <p>Přivést vodu k varu.</p></li>
   </ul>
</div>
```

Otázka zkouškového testu opravdu v extrakci HTML neobsahuje váhy správných/špatných odpovědí.

4.6.1.3 XML

Poslední autorovi známá možnost je XML extrakt, který je i velmi vhodný a jednoduchý na programatické extrakce dat.

Příklad XML extraktu:

```
<!-- question: 1676973  -->
<question type="multichoice">
   <name>
      <text>Recept na vařenou vodu</text>
   </name>
   <questiontext format="html">
      <text>
         <![CDATA[<p>Co je potřeba k uvaření vody?</p>]]></text>
   </questiontext>
   <generalfeedback format="html">
      <text></text>
   </generalfeedback>
   <defaultgrade>1.0000000</defaultgrade>
   <penalty>0.3333333</penalty>
   <hidden>0</hidden>
   <single>true</single>
   <shuffleanswers>true</shuffleanswers>
   <answernumbering>abc</answernumbering>
   <correctfeedback format="html">
      <text><![CDATA[<p>Vaše odpověď je správná.</p>]]></text>
   </correctfeedback>
```

```
<partiallycorrectfeedback format="html">
  <text><![CDATA[<p>Vaše odpověď je částečně správná.</p>]]></text>
</partiallycorrectfeedback>
<incorrectfeedback format="html">
  <text><![CDATA[<p>Vaše odpověď je chybná.</p>]]></text>
</incorrectfeedback>
<shownumcorrect/>
<answer fraction="50" format="html">
  <text><![CDATA[<p>Voda.</p>]]></text>
  <feedback format="html">
    <text></text>
  </feedback>
</answer>
<answer fraction="-33.33333" format="html">
  <text><![CDATA[<p>Nevím.</p>]]></text>
  <feedback format="html">
    <text></text>
  </feedback>
</answer>
<answer fraction="-33.33333" format="html">
  <text><![CDATA[<p>10 programátorů a jedna cvičená opice.</p>]]></text>
  <feedback format="html">
    <text></text>
  </feedback>
</answer>
<answer fraction="-33.33333" format="html">
  <text><![CDATA[<p>Sůl a pepř.</p>]]></text>
  <feedback format="html">
    <text></text>
  </feedback>
</answer>
<answer fraction="50" format="html">
  <text><![CDATA[<p>Přivést vodu k varu.</p>]]></text>
  <feedback format="html">
    <text></text>
  </feedback>
</answer>
</question>
```

Každá odpověď obsahuje atribut fraction, který slouží jako indikátor správnosti odpovědí a zároveň se podle něj pravděpodobně určuje i zda se má bodové hodnocení odečíst / přičíst dle správnosti odpovědí.

V ukázkovém případu lze vidět, že otázka má 5 odpovědí a z toho 2 správné a 3 nesprávné. Jednotka atributu fraction je procentuální vyjádření [100 % / počet správných či negativních odpovědí] nebo i možná individuální. Pro 2 správné odpovědi je fraction 2x 50 % a pro 3 špatné odpovědi 3x -33.33333 %. Z toho lze usoudit, že je možné získat za otázku mínusové body.

4.6.1.4 Akceptovaný formát dat pro chatbota

Vzhledem k tomu, že testovací data byla ve formátu XML bylo potřeba tato data transformovat pro chatbotem akceptovaný formát.

Akceptovaný formát je textový soubor obsahující vždy pár Otázka a správná odpověď ve tvaru:

Q
A
Q
A

Na zpracování dat autor naprogramoval jednoduchý program v programovacím jazyce Java, tento jazyk byl zvolen oproti primárně použitému Pythonu díky autorovým znalostem nástrojů, které jsou pro Javu k dispozici.

4.6.1.5 XSD

Díky absenci XSD bylo potřeba jej ručně vygeneroval z poskytnutého XML. V praxi by tento přístup nebyl přípustný, pokud by XSD bylo potřeba třeba pro SOAP komunikaci. V případě, kdy je potřeba získat aproximaci možné podoby schématu toto nevadí. Tuto aproximaci lze použít pro následné vytvoření finálního schématu, když je schéma tvořeno pro nějaký účel z ukázkových XML nebo pro tuto jednorázovou extrakci dat.
Schéma bylo vygenerováno pomocí funkcionality v Java IDE „Intellij IDEA“ z XML extraktu.

4.6.1.6 Vygenerování modelu

Způsobů, jak vygenerovat modelové Java třídy z XSD je hodně, avšak vzhledem k použití nástroje Maven, stačilo přidat maven-jaxb2-plugin a nastavit jej.

Nastavení v pom.xml:

```
<plugin>
  <groupId>org.jvnet.jaxb2.maven2</groupId>
  <artifactId>maven-jaxb2-plugin</artifactId>
  <version>0.13.2</version>
  <executions>
    <execution>
```

```
        <goals>
          <goal>generate</goal>
        </goals>
        <configuration>
        <generatePackage>cz.havranek.milos.interakt.moodletools.model</generatePackage>
          <schemaDirectory>src/main/resources/</schemaDirectory>
          <schemaIncludes>
            <include>**/*.xsd</include>
            <include>*.xsd</include>
          </schemaIncludes>
          <encoding>UTF-8</encoding>
        </configuration>
      </execution>
    </executions>
</plugin>
```

4.6.2 Zpracování dat

Pro získání správných odpovědí byla určena hodnota, kde nelze určit, zda je odpověď správná či špatná [fraction = 0], to znamená, že odpovědi s [fraction > 0] jsou považovány za správné a [fraction < 0] za špatné.

Při získání párů otázek a odpovědí bylo nutné:

1. Vyfiltrovat pouze správné odpovědi.
2. Vytvořit umělou redundanci otázek tak, aby každá odpověď mohla mít svoji otázku do páru.
3. Odstranit HTML elementy a jiné nedostatky, aby zůstal čistý text.

Filtrování správných odpovědí:

```
private List<Answer> mineAnswers(List<Answer> answers) {
  return answers.stream()
        .filter(answer -> answer.getFraction().doubleValue() > 0)
        .collect(Collectors.toList());
}
```

Odstranění HTML elementů:

```
private String removeUnwantedChars(String s) {
  return s.replace("\n", " ")
        .replace("\r", "")
        .trim()
        .replaceAll("<\\s*?[^>]+\\s*?>", "");
}
```

4.6.3 Transformace dat

Samotná transformace dat se skládá z několika kroků:

1. Načtení XML
2. Transformace dat z XML na Java objekty s XSD validací
3. Filtrování správných odpovědí
4. Filtrování nechtěných znaků (XML a HTML elementy)
5. Otázka je spárována se správnými odpověďmi
6. Uložení výsledných Q&A do textového souboru

Ukázkový výstupní soubor qas.txt:

Co je potřeba k uvaření vody?
Vodu.
Co je potřeba k uvaření vody?
Přivést vodu k varu.

4.7 Deployment

4.7.1 Prerekvizity

Aplikaci lze rozdělit do několika forem deploymentu, při kterém se liší chování aplikace dle potřeb a podmínek prostředí.

Aplikace bude nasazována na neprodukční a produkční prostředí. Neprodukční prostředí plní účelnost vývoje a testů, kde je potřeba podrobnějšího logování, zatímco produkční vyžaduje stabilitu a rychlost.

4.7.1.1 Deployment prostředí

V tabulce níže je rozdělení deployment prostředí a speciálního případu, kdy je aplikace deployována v Dockeru. Docker deployment je možné použít jak pro produkční, tak neprodukční prostředí. Pro snažší vývoj je neprodukční prostředí nastaveno tak, aby vyhovovalo potřebám vývoje a testu.

Tabulka 4: Rozdíly nastavení prostředí pro deployment

Vlastnost	Neprodukční	Produkční	Docker

Debug	Ano	Ne	*
Databáze	SQLite	PostgreSQL	PostgreSQL
Server	Django debug	Gunicorn	Gunicorn
Statické soubory poskytuje	Django debug	Whitenoise	Whitenoise
Generování statických souborů	Ne	Ano	Ano, ale při sestavení image

Zdroj: Vlastní zpracování

4.7.1.2 Nginx vs. Whitenoise

Vzhledem k tomu, že aplikační server gunicorn poskytuje pouze dynamické soubory aplikace (soubory generované serverem), tak je nutné zařídit poskytování těchto souborů jiným webovým serverem nebo dodatečnou knihovnou v aplikaci. Je možné statické soubory poskytovat pomocí webového serveru Nginx, ale to vyžaduje další aplikaci, která běží vedle hlavní aplikace. Lepším řešením je Python knihovna Whitenoise, který je možné použít v kombinaci s gunicorn aplikačním serverem přímo.

4.7.2 Dockerfile

Dockerfile obsahuje instrukce pro sestavení Docker image (obraz), který je nezávislý na platformě. Lze tedy říci, že je tímto způsobem možné mít pod kontrolou prostředí, na kterém aplikace běží. Díky tomu je možné hotový image přenést na jiný počítač s jinou OS platformou a nemuset řešit, jaký software je nainstalován. A také především odpadají rozdílné procedury a prerekvizity samotné instalace programu. Samozřejmě je také možné daný image sestavit až na cílovém systému.

Docker image aplikace Interakt je přizpůsoben pro běh Pythonu verze 3 s podporou pipenv pro instalaci dependencies. Vzhledem k tomu, že není python potřeba kompilovat, tak na takovémto základu je potřeba překopírovat soubory do nově vytvářeného image. A nastavit execute oprávnění pro spouštěcí skript aplikace, jelikož se jedná o shell script.

Aplikační server vyžaduje také provést systémový management příkaz „collectstatic“, který statické soubory překopíruje do finálního umístění, kde jej bude knihovna whitenoise při startu aplikačního serveru gunicorn očekávat.

V neposlední řadě je potřeba po instalaci dependencies, tedy závislých balíčků nastavit takzvaný entrypoint aplikace. Entrypoint aplikace se odkazuje na spouštěcí skript „interakt.sh“, který je pak při tvorbě containeru a jeho spuštění defaultně spuštěn.

4.7.3 Docker compose

Docker compose je nadstavba dockeru, která v případě Interakt chatbot aplikace umožňuje jednorázového nastavení deploymentu a propojení závislých containerů.

Docker compose projekt se skládá ze dvou containerů. Ze samotné aplikace a databázového serveru. Nastavení aplikace pak probíhá přes environmentální proměnné. K rozdílnému nastavení je jen potřeba tyto environmentální proměnné nastavit v souboru „env“, který si Docker compose načte při deploymentu.

Je nutné zmínit i fakt, že ačkoliv pro nasazení v Dockeru je defaultně použita defaultní kombinace přihlašovacích údajů do databáze „postgres:admin“, tak není potřeba jej měnit, pokud se nejedná třeba o produkční provoz aplikace. V rámci bezpečnosti dat by sice mělo být myšleno na změnu takových přihlašovacích údajů, ale je nutné mít na paměti, že s danou databází bude komunikovat jen aplikace a jediný, kdo bude mít možné se k databázi připojit je také aplikace Interakt. Proto je i toto zabezpečení pro vývoj a test dostačující.

Izolované připojení aplikaci k databázi zajišťuje separátní virtuální podsíť, kde je jejím členem jen aplikace a databázový server.

4.7.4 Samotné nasazení aplikace

K provozu aplikace je potřeba nainstalovaný Git, Docker a Docker Compose. Následně pomocí několika kroků je možné aplikaci nainstalovat na server.

Stažení kódu aplikace z gitu:

```
cd ~
```

```
git clone https://github.com/ExampleUserName/interakt.git
cd ./interakt
```

Dále je potřeba nastavit environmentální proměnné pro sestavení. Je možné použít již připravený defaultní soubor:

```
cp .env.default .env
```

```
cat .env
        COMPOSE_PROJECT_NAME=interakt
        INTERAKT_PRODUCTION=1
        INTERAKT_DOCKER=1
        DJANGO_SECRET_KEY=random-secret-string
        POSTGRES_USER=postgres
        POSTGRES_PASSWORD=admin
        POSTGRES_HOST=db
        POSTGRES_PORT=5432
```

Tento soubor „env“ je možné upravit dle potřeb.

Následná první instalace a build containeru:

```
sudo docker-compose up -d
```

4.7.5 Aktualizace již deployované aplikace

Pokud aplikace běží je nutné ji zastavit:

```
sudo docker-compose stop
```

Ve složce s kódem provedeme update repositáře:

```
cd ~/interakt
git pull
```

Opětovné sestavení containeru a zapnutí aplikace:

```
sudo docker-compose build
sudo docker-compose up -d
```

4.7.6 První spuštění

Je-li aplikace prvně spuštěna neobsahuje databáze žádná data. Je potřeba vytvořit databázové struktury.

Soubor „interakt.sh“, který je defaultně zavolán jako Docker entrypoint obsahuje tyto potřebné procedury.

První spuštění zajistí:

- Vytvoření databázových struktur
- Vytvoření defaultního administrátorského Django účtu (admin:admin)
- Naučení defaultní báze dat ze souboru:“qas_clean.txt“

Po úspěšném prvním spuštění se vytvoří placeholder soubor indikující stav nainstalované aplikace: „interakt-installed“.

4.7.7 Auto-migrace databáze

Při každém spuštění aplikace probíhají auto-migrace databáze stejným způsobem, jakým je tomu při prvním spuštění. Rozdíl je ovšem ten, že při existujících datech v databázi se provedou pouze migrace, u již vytvořené struktury.

Databáze si tedy zachová svá data a pokud se z nějakého důvodu aktualizovala struktura databáze, tak pomocí migračních skriptů jsou provedeny migrace.

Aplikace je tedy schopna fungovat i při aktualizaci komponent a kódové báze.

Komponenty, které obsahují migrační skripty:

- Django
- ChatterBot

4.7.8 Správa aplikace pomocí „manage.py“

Django framework poskytuje ke každé aplikaci soubor „manage.py“, pomocí kterého je možné provádět základní předpřipravené programatické operace a také vlastnoručně napsané skripty. Pro administrátora aplikace je vhodné základní příkazy používat. Je totiž pomocí nich možné například zálohovat celou databázi.

Exekuce „manage.py“ v Docker containeru „interakt_web_1“:

```
docker exec -it interakt_web_1 python manage.py
```

Očekávaný výstup (zkrácený na vlastní skripty aplikace Interakt):

```
[interakt]
  asktest
```

count
createsuperuserdefault
createsuperusernoprompt
purge
traindefault
trainlistfile

Speciálně pro aplikace Interakt Q&A Chatbot byly vytvořeny některé management příkazy skriptu „manage.py“ k usnadnění administrace aplikace.
Příkazy:

- asktest
    - Položí testovací otázku chatbotovi. Slouží k debugování stavu databáze.
- count
    - Vypíše počet naučených Q&A naučených v databázi.
- createsuperuserdefault
    - Vytvoří defaultní administrátorský účet (admin:admin) pro Django administraci
- createsuperusernoprompt
    - Vytvoří administrátorský účet pro Django dle zadaných parametrů.
- purge
    - Vymaže naučenou bázi dat Q&A z databáze.
- traindefault
    - Naučí chatbota data z defaultního souboru „qas_clean.txt“.
- trainlistfile
    - Naučí chatbota data ze souboru.

4.7.9 Naučení testovacích dat

Jak bylo již zmíněno, tak defaultní báze dat je naučena při prvním spuštění aplikace. Pro testovací účely tedy není potřeba provádět další nebo jiné naučení báze dat. Pro úplnost je zde uveden proces učení pro administrátory aplikace.

Neběží-li aplikace v Dockeru, není potřeba zvláštním způsobem řešit, jak dostat data do containeru a jak spouštět v Dockeru skripty. Vzhledem k použití Dockeru jako výchozího deploymentu je postup uveden. Pokud by administrátor Docker nepoužil, stačí podobným způsobem provést ty samé operace bez použití zaobalujících Docker příkazů.

Zkopírování souboru Q&A do Docker containeru:

sudo docker cp ./qas_clean.txt interakt_web_1:/app/

Chatbota je možné naučit postupně z více souborů inkrementálně unikátní otázky. Je dobré se však vyhnout duplicitám. Pokud je tedy potřeba naučit soubor obsahující již existující otázky je nutné nejdříve promazat bázi dat.

Promazání báze dat Q&A v databázi:

sudo docker exec -it interakt_web_1 python manage.py purge

Očekávaný výstup:

Purge Training of Interakt ChatBot has started.
Purge Training of Interakt ChatBot has finished.

Naučení Q&A ze zkopírovaného souboru:

sudo docker exec -it interakt_web_1 python manage.py trainlistfile qas_clean.txt

Očekáváný výstup:

Training of Interakt ChatBot has started.
QA List Trainer: [####################] 100%
Training of Interakt ChatBot has finished.

4.8 Otestování a verifikace

Pro otestování výstupů chatbota byla zvolena data z předmětu Interakce člověk počítač. Tato data jsou distribuována s aplikací.

Typy testů:

- Prázdný vstup
- Vstup neobsažený v datech
- Vstupy obsažené v datech
- Hranice (ne)rozpoznání vstupu

4.8.1 Prázdný vstup

Níže uvedený test slouží k otestování prázdného vstupu uživatele.

Vstup: „“

Očekáváný výstup: „Položte prosím otázku.“

Výstup: „Položte prosím otázku.“

Jistota správné shody: 100 %

4.8.2 Vstup neobsažený v datech

Tento test demonstruje chování chatbota při vložení vstupu, který není uložen v bázi dat.

Vstup: „ahoj“

Očekáváný výstup: „Nebylo možné správně determinovat položenou otázku. Zkuste ji položit jiným způsobem.“

Výstup: „Nebylo možné správně determinovat položenou otázku. Zkuste ji položit jiným způsobem.“

Jistota správné shody: 100 %

4.8.3 Vstupy obsažené v datech

Testy vstupů obsažených v datech demonstrují funkčnost při zadání vstupu uživatele, kde je docíleno takové shody, aby byl vrácen správný vstup.

4.8.3.1 Test 1

Vstup: „zásady brainstormingu“

Nalezen shodný vstup: „Jaké jsou základní zásady brainstormingu?“
Očekáváný výstup [jeden z množiny]:

- Před započetím problém ještě jednou zopakovat.
- Mluvit by měl v jednom okamžiku pouze jeden.
- Po fázi vymýšlení přijde na řadu výběr nejlepších nápadů ze všech zapsaných.

Výstup: „Před započetím problém ještě jednou zopakovat.“

Jistota správné shody: 68 %

4.8.3.2 Test 2

Vstup: „jak definujeme grafický vzhled obrazovek“

Nalezen shodný vstup: „Co definuje grafický vzhled obrazovek?“

Očekáváný výstup [jeden z množiny]:

- Grafický vzhled obrazovek definuje finální grafickou formu ovládacích prvků (tvary, velikosti, barvy atd.).
- Logický vzhled obrazovek definuje rozmístění jednotlivých komponent a ovládacích prvků.

Výstup: „Grafický vzhled obrazovek definuje finální grafickou formu ovládacích prvků (tvary, velikosti, barvy atd.).“

Jistota správné shody: 90 %

4.8.3.3 Test 3

Vstup: „formulace myšlenkového modelu uživatele“

Nalezen shodný vstup: „Co formuluje myšlenkový model uživatele?“

Očekáváný výstup: „Myšlenkový model uživatele formuluje vnímání člověka směrem k pochopení a využití stroje.“

Výstup: „Myšlenkový model uživatele formuluje vnímání člověka směrem k pochopení a využití stroje.“

Jistota správné shody: 84 %

4.8.4 Hranice (ne)rozpoznání vstupu

Chatbot dokáže dobře rozpoznávat vstupy, kde v samotné bázi dat jsou dané otázky od sebe odlišné. Pokud jsou otázky velmi podobné (zhruba 80 % a více), tak je těžké daný vstup najít v bázi dat.

Příklad Q&A párů:

- Do které části obrazovky by se měly umísťovat nejdůležitější navigační a ovládací prvky (pro Střední a Východní Evropu)?
    - Do levého horního rohu.
- Do které části obrazovky by se měly umísťovat nejdůležitější navigační a ovládací prvky (pro Střední a Západní Evropu)?
    - Do levého horního rohu.
- Do které části obrazovky by se měly umísťovat nejdůležitější navigační a ovládací prvky (pro arabské země)?
    - Do pravého horního rohu.

Na ukázce výše je možné vidět příklad podobných otázek, které mohou mít různé odpovědi. Pro správnost přiřazení vstupu k otázce a následného vrácení správné odpovědi je potřeba, aby uživatel specifikoval ve vstupu detaily, které pomohou k následnému rozlišení mezi otázkami.

4.8.4.1 Při obecném zadání vstupu:

Vstup: „kam umístit navigační prvky“
Nalezen shodný vstup: „Do které části obrazovky by se měly umísťovat nejdůležitější navigační a ovládací prvky (pro arabské země)?“
Výstup: „Do pravého horního rohu.“
Jistota správné shody: 40 %

Jistota shody je v tomto případě pouze 40 %. Pro otázku: „Do které části obrazovky by se měly umísťovat nejdůležitější navigační a ovládací prvky (pro Střední a Východní Evropu)?“, pak jen 37 %. Otázkou je pak i jestli v takovém případě uživatel vůbec chtěl odpověď pro „Arabské země“, „Střední a Východní Evropu“ anebo pro úplně jinou otázku.

4.8.4.2 Při specifikovaném zadání vstupu:

Vstup: „kam umístit navigační prvky pro střední Evropu“

Debug:

INFO:chatterbot.chatterbot:Similar text found: Do které části obrazovky by se měly umísťovat nejdůležitější navigační a ovládací prvky (pro Střední a Východní Evropu)? 0.55
INFO:chatterbot.chatterbot:Similar text found: Do které části obrazovky by se měly umísťovat nejdůležitější navigační a ovládací prvky (pro Střední a Západní Evropu)? 0.56

Zde je možné vidět, že byly nalezeny dvě shody, které mají sice ve výsledku stejnou odpověď, ale jistota shody je jen 55 % a 56 %.

Po zadání vstupu: „kam umístit navigační prvky pro střední a východní Evropu“

Debug:

INFO:chatterbot.chatterbot:Similar text found: Do které části obrazovky by se měly umísťovat nejdůležitější navigační a ovládací prvky (pro Střední a Východní Evropu)? 0.62

Zde je již vidět, že je vstup uživatele dostatečně specifikován pro nalezení unikátní odpovědi s jistotou shody 62 %.

5 Výsledky a diskuse

Na trhu existuje mnoho řešení zabývající se různými problémy. Každý chatbot je svým způsobem unikátní, i když metodiky a algoritmy jsou známé již řadu let. Unikátnost každého chatbota spočívá v řešení dané problematiky, kterou má chatbot svou existencí řešit. Co však každého chatbota odlišuje je ve výsledku báze dat a okruhy informací, se kterými je chatbot schopný pracovat a dále schopnost umět s těmito informacemi pracovat.

Vize autora této práce byla uchopit úlohu, která by se zabývala řešením konkrétního problému. Na tento problém najít řešení a řešení následně implementovat na aktuálních technologiích za pomoci aktuálních metodik a algoritmů. Tato práce se zabývala Q&A chatbotem, který může být nasazen do produkce a pomáhat uživatelům zodpovědět otázky z uzavřeného okruhu témat. Řešení stačí integrovat do libovolného systému a naučit chatbota konkrétní data.

Jako pokračování či navázání na tuto práci by mohlo být samotná analýza dat pro chatbota k naučení. Ne každá data se hodí pro takovýto systém a je nutné data minimálně vyčistit od nevhodných či je transformovat do jiné podoby při zachování stejného významu. Také by bylo celkem zajímavé poskytnout uživateli možnost mezi otázkami vyhledávat. Toho by bylo možné dosáhnout vytvořením dalšího adaptéru, který by nebyl Q&A, ale byl by založen na příkazové logice.

6 Závěr

Autorovi práce se podařilo splnit zadané cíle práce, kde postupoval dle stanovené metodiky za využití odborné literatury, dokumentací a odborných online publikací. Interakt Q&A chatbot je funkční řešení, které je schopné zodpovídat naučené otázky. Aplikace vystavuje REST API a webové rozhraní přes které je možné s chatbotem komunikovat.

V teoretické části práce byl popsán jazyk Python a použité frameworky pro vývoj aplikace. Dále jsou popsány základní typy chatbotů a principy jejich fungování. Teoretická část práce se především zabývá metodickými východisky, které je pak možné použít v praktické části práce jako potřebné znalosti pro zhotovení samotné aplikace.

Pro vývoj aplikace bylo použito předchozích praktických znalostí z oboru vývoje software a teoretických východisek získanými z odborné literatury. Vlastní práce začíná zjištěním aktuálních řešení na trhu vůči zadaným cílům práce, kde následuje analýza samotného zadaní pro zhotovení aplikace. Cílem analýzy aktuálních řešení na trhu bylo získat přehled o dostupných řešeních a technologiích, které by bylo možné použít pro následnou implementaci. Analýza zadaní pak čerpá z aktuálních řešení na trhu a vymezuje vhodná a použitelná řešení pro vývoj chatbota dle zadaných cílů diplomové práce.

Samotná praktická část sestává z funkční aplikace Interakt Q&A chatbot, která je naprogramována v jazyce Python. Praktická část práce obsahuje programový kód, testovací bázi dat a deployment skripty. Jako dílčí programový kód je doplněn „Q&A Extractor“, který transformuje XML Moodle extrakty do chatbotem podporovaného formátu pro následné naučení. Praktická část práce je zakončena otestováním aplikace a verifikací funkcionality.

7 Seznam použitých zdrojů

7.1 Knižní zdroje

TETTAMANZI, Andrea G. B. a Marco TOMASSINI. *Soft Computing Integrating Evolutionary, Neural, and Fuzzy Systems*. Springer, 2001. ISBN 978-3-662-04335-6.

7.2 Internetové zdroje

Alphabet Inc. *Hangouts chat API.* [online]. Google Developers, 2018 [cit. 2019-1-25]. Dostupné z: https://developers.google.com/hangouts/chat/concepts/bots

BICKING, IAN. *Virtualenv.* [online]. 2018 [cit. 2019-2-2]. Dostupné z: https://virtualenv.pypa.io/en/latest/#

COX, Gunther. *Chatterbot*. [online]. 2019 [cit. 2019-2-8]. Dostupné z: https://chatterbot.readthedocs.io/en/stable/

DOCKER. *What is a container?* [online]. Docker Inc., 2019 [cit. 2019-1-8]. Dostupné z: https://www.docker.com/resources/what-container

DVOŘÁK, Pavel. *Django: Prezentace dat.* [online]. Zdroják.cz, 2009. [cit. 2019-1-9]. Dostupné z: https://www.zdrojak.cz/clanky/django-prezentace-dat/

KOJOUHAROV, Stefan. *Chatbots Life*. [online]. Medium, 2016 [cit. 2019-1-20]. Dostupné z: https://chatbotslife.com/ultimate-guide-to-leveraging-nlp-machine-learning-for-you-chatbot-531ff2dd870c

SANJEEVI, Madhu. *Deep math machine learning AI*. [online]. Medium, 2018 [cit. 2019-1-20]. Dostupné z: https://medium.com/deep-math-machine-learning-ai/chapter-11-chatbots-to-question-answer-systems-e06c648ac22a

MALÝ, Martin. *REST: architektura pro webové API.* [online]. Zdroják.cz, 2009. [cit. 2018-1-19]. Dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

PAUL, Javin. *Javarevisited.* [online]. 2019 [cit. 2019-1-25]. Dostupné z:

https://javarevisited.blogspot.com/2018/07/10-reasons-to-learn-java-programming.html

PHILLIPS, Casey. *Chatbots Magazine.* [online]. OCTANE AI, 2018 [cit. 2019-1-10]. Dostupné z: https://chatbotsmagazine.com/the-3-types-of-chatbots-how-to-determine-the-right-one-for-your-needs-a4df8c69ec4c

REITZ, Kenneth. *Pipenv: Python Dev Workflow for Humans*. [online]. 2018 [cit. 2019-2-8]. Dostupné z: https://pipenv.readthedocs.io/en/latest/

WICHMANN, Mats. *Python*. [online]. 2017 [cit. 2019-2-9]. Dostupné z: https://wiki.python.org/moin/Python2orPython3

MICROSOFT. *What is Docker?* [online]. 2019 [cit. 2019-2-9]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/containerized-lifecycle-architecture/what-is-docker

DOCKER. *Dockerfile reference.* [online]. Docker Inc., 2019 [cit. 2019-2-9]. Dostupné z: https://docs.docker.com/engine/reference/builder/

DOCKER. *Overview of Docker compose.* [online]. Docker Inc., 2019 [cit. 2019-2-9]. Dostupné z: https://docs.docker.com/compose/overview/

OVH. *Začíname s PostgreSQL*. [online]. OVH.cz s.r.o., 2019 [cit. 2019-2-9]. Dostupné z: https://docs.ovh.com/cz/cs/clouddb/zaciname-s-postgresql/

GIT-SCM. *Základy systému GIT.* [online]. git-scm.com, 2019 [cit. 2019-2-9]. Dostupné z: https://git-scm.com/book/cs/v1/%C3%9Avod-Z%C3%A1klady-syst%C3%A9mu-Git

W3C. *HTML & CSS.* [online]. World Wide Web Consortium, 2019. [cit. 2019-2-9]. Dostupné z: https://www.w3.org/standards/webdesign/htmlcss

ACELLERE. *Is JavaScript a programming language?*. [online]. Acellere GmbH, 2018 [cit. 2019-2-10]. Dostupné z: https://medium.com/acellere/is-javascript-a-programming-language-b068b6eda749

ITNETWORK. *Úvod do AJAXu.* [online]. Unicorn College, 2015 [cit. 2019-2-10]. Dostupné z: https://www.itnetwork.cz/javascript/ajax/uvod-do-ajaxu

ALGORITMY.NET. *Levenshteinova vzdálenost.* [online]. algoritmy.net, 2019 [cit. 2019-2-15]. Dostupné z: https://www.algoritmy.net/article/1699/Levenshteinova-vzdalenost

KLEIWEG, Peter. *Levenshtein.* [online] 2019 [cit. 2019-2-15]. Dostupné z: http://www.let.rug.nl/kleiweg/lev/

8 Přílohy

Příloha 1: Zdrojové kódy aplikací …….……..CD