FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Chatbot pro produktové portfolio
Student:	Tomáš Stanovčák
Vedoucí:	Ing. David Kreuz
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Navrhněte a implementujte chatbota pro zpracování obsáhlého produktového portfolia společnosti. Chatbot bude určen pro sales tým, který potřebuje rychlejší přístup k informacím o produktech společnosti. Chatbot bude:
poskytovat informace sales týmu společnosti,
čerpat data z databáze produktového portfolia,
komunikovat prostřednictvím MS Teams,
zdokonalovat svou schopnost porozumět dotazům.

Postupujte v těchto krocích:
proveďte výzkum funkčních požadavků,
proveďte rešerši existujících NLP systémů,
analyzujte možnosti platformy RASA,
navrhněte strukturu chatbota,
implementujte chatbota na základě návrhu,
sestavte testy pro konverzaci s chatbotem,
shrňte získané zkušenosti.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 4. října 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Chatbot pro produktové portfolio

Tomáš Stanovčák

Katedra softwarového inženýrství
Vedoucí práce: Ing. David Kreuz

4. června 2020

Poděkování

Především bych rád poděkoval svému vedoucímu práce Ing. Davidu Kreuzovi za odborný přístup, cenné rady a čas, který mi věnoval po celou dobu psaní této práce. Dále děkuji své rodině a přátelům, kteří mi poskytovali podporu a motivaci při studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na jejímž základě se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ustanovení § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisu.

V Praze dne 4. června 2020 . . . . . .

Odkaz na tuto práci

Stanovčák, Tomáš. *Chatbot pro produktové portfolio*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato práce se zabývá vývojem chatbota zpřístupňujícího znalostní bázi produktového portfolia zaměstnancům společnosti. V první části čtenáři představuje obecnou problematiku chatbotů a jejich využití v praxi. Zároveň mapuje aktuální technologie zpracování přirozeného jazyka a frameworky pro tvorbu chatbotů. V další části se věnuje analýze požadavků, případů užití a návrhu chatbota, který bude postaven na frameworku Rasa. Součástí práce je i samotná implementace v jazyku Python spolu s testy, které prověřují různé konverzační scénáře.

Klíčová slova chatbot, NLP, entita, intent, znalostní báze, produktové portfolio, Rasa

Abstract

This thesis describes the development process of the chatbot that brings the knowledgebase of product portfolio to company employees. In the first part, a reader is acquainted with chatbots and their usage in general. Moreover, it gathers information about nowadays technologies of natural language processing and chatbot design frameworks. The next part of the thesis is dedicated

to the requirements and use case analysis along with the design process of the chatbot based on Rasa framework. The implementation coded in Python language accompanied with conversation scenario tests are also a part of the thesis.

Keywords chatbot, NLP, entity, intent, knowledgebase, product portfolio, Rasa

Obsah

Seznam obrázků

Seznam ukázek kódu

Seznam tabulek

Úvod

Chatboti jsou v současnosti nejen na českém trhu stále větším trendem. Mnohé firmy je nasazují pro vylepšení komunikace se zákazníky, některé pomocí nich staví reklamní kampaně a jiné automatizují nábor zaměstnanců. Tato technologie měsíc co měsíc nabírá na popularitě a to hlavně díky možnosti ušetřit drahocenný čas na činnostech, které mohou být automatizované a efektivnější.

Jak už úvodní slova napovídají, i v případě mojí práce to nebude jinak. Je nutné vyřešit současný stav neefektivního vyhledávání v produktovém portfoliu společnosti ANECT a.s. To dnes funguje na základě textového vyhledávání, které není ani zdaleka ideální. Proto je cílem zefektivnit práci s portfoliem pomocí vytvoření znalostního chatbota, který samotné portfolio použije jako podklad své báze znalostí. Bude tak moct odpovědět zaměstnancům společnosti na dotazy týkající se detailů produktů a jejich modulů.

Motivací pro výběr tohoto tématu bylo z mé strany hned několik. Už dva roky se s kolegou Janem Šafaříkem věnujeme tvorbě chatbotů pro různé oblasti jako je e-commerce a marketing v rámci našeho studentského startupu WaldoBot. Když jsme se poprvé setkali s požadavkem sestavit znalostního chatbota spolu s administrativním systémem v rámci spolupráce nad bakalářskou prací, byla to pro mě velice zajímavá nabídka. Rád propojuji své oblasti zájmu spolu se školou, a proto jsem se rozhodl pro tvorbu chatbota, který bude v praxi reálně nasazen a používán. Kolega Jan Šafařík se ve své práci věnuje tvorbě již zmiňovaného administrativního systému, který spolu s chatbotem v budoucnu vytvoří funkční celek vhodný pro nasazení do praxe. Díky této skutečnosti je možné efektivitu řešení ověřit a ladit v reálném čase. Budou tak získána cenná data a zkušenosti pro další rozvíjení projektu, který má velký potenciál budoucího rozvoje nejen ve své oblasti.

Na začátku této práce budou stanoveny primární a dílčí cíle. Teoretická část se věnuje rešerši aktuálních řešení, definování pojmů a rozboru technologií. Na ty bude pak navázána analýza a návrh samotného chatbota. V praktické části se bude práce zabývat samotnou implementací a testováním řešení.

Kapitola 1

Cíl práce

Primárním cílem této práce je navrhnout, implementovat a otestovat chatbota pro zpracování obsáhlého produktového portfolia společnosti. Dílčím cílem návrhu je zmapovat dostupné NLP systémy, které umožňují chatbotům rozumět přirozeným dotazům. Dále je důležité analyzovat funkční požadavky klienta. Mezi dílčí cíle implementace patří i vytvoření konektoru pro prostředí MS Teams, prostřednictvím kterého budou uživatelé s chatbotem komunikovat.

Kapitola 2

Teoretická část

Tato kapitola poskytne nutný teoretický úvod do problematiky chatbotů. Pojem bude definován a vysvětlen prostřednictvím praktických případů užití z mnoha odvětví. Detailněji se bude kapitola věnovat znalostním chatbotům a nástrojům, které jim umožňují porozumět dotazům od uživatelů a poskytovat tak informace. Rozborem těchto aktuálně dostupných nástrojů a platforem bude zdůvodněn výběr platformy Rasa, která poskytuje všechny potřebné prostředky pro sestavení znalostního chatbota.

2.1 Chatbot

Chatbot je program, který s uživatelem komunikuje pomocí chatu nebo hlasu. Odpovídaní ze strany chatbota je plně automatizované díky předem definovaným pravidlům nebo umělé inteligenci. Většina textových chatbotů používá ke své komunikaci jednu ze známých chatových platforem jako je například Messenger, WhatsApp nebo Slack. [1]

2.1.1 Druhy chatbotů

Na dělení chatbotů je možno se dívat z hlediska různých kritérií. Pokud bude bráno v potaz použití, tak jsou známé dvě následující kategorie:

Konverzační – Nemá za cíl o ničem konkrétním informovat, účelem je komunikace samotná. Proto jí chatbot musí aktivně rozvíjet a udržovat si tak konverzujícího uživatele. Své odpovědi staví kreativně a otázky klade tak, aby dávaly uživateli prostor pro rozvitou odpověď. [2]

Úkolově-zaměřený – Navržen pro jeden a více specifických úkolů týkajících se jednoho uzavřeného tématu. Patří sem většina chatbotů, které dnes využívají firmy pro automatizaci procesů, jako je například objednání zboží, sjednání termínu u kadeřníka nebo třeba rezervace hotelu. [2]

2.1.2 Využití

Chatboti najdou v dnešní době uplatnění napříč širokým spektrem odvětví. Jejich přínos mohou lidé pozorovat i v každodenním životě. Vhodným příkladem jsou hlasoví asistenti jako je Alexa, Siri nebo Google Assistent. Právě díky nim mohou uživatelé například pouhým hlasem ovládat domácí spotřebiče, sestavit nákupní seznam nebo ověřit jakýkoliv fakt na internetu. [3]

Mezi odvětví, ve kterých se chatboti nejvíce používají, patří ve značné míře zákaznická podpora, marketing, prodej a personalistika. V zákaznické podpoře řeší vytíženost operátorů tím, že odpovídají na často kladené dotazy a řeší požadavky zákazníků. [4] Mezi časté žádosti zákazníku patří například reklamace nebo vrácení zboží. Marketérům zase umožňují vytvářet originální kampaně, ve kterých chatbot plní většinou funkci propojování online a offline světa prostřednictvím různých interakcí. Tyto interakce se vyznačují sdílením multimediálního obsahu, jako je například posílání krátkých videí nebo vtipných hlasových nahrávek, a to přímo v chatu.

Ve sféře prodeje mají chatboti uplatnění jako virtuální asistenti prodeje. Za pomoci dat zákazníků tvoří personalizovanou nabídku, která je klíčová pro samotný prodej zboží. Navíc mohou zákazníky kontaktovat v případě akcí a slev. V personalistice je vliv chatbotů také znatelný. [4] Nejčastěji se používají pro automatizaci náboru nových zaměstnanců. Mohou tak efektivně sbírat a srovnávat informace o jednotlivých kandidátech. Šetří personalistům čas, který mohou efektivně využít na pohovorech s jednotlivými vybranými kandidáty.

Napříč všemi obory se ukazuje možnost využití specializovaných znalostních chatbotů, kteří čerpají informace z báze informací a ty pak poskytují uživatelům. Tato možnost se uplatňuje například ve sféře zákaznické podpory, kde se odpovědi na často kladené dotazy uloží do báze informací. Dále tyto postupy možno pozorovat v oblasti lidských zdrojů, kde se firemní informace zpřístupňují zaměstnancům a spolupracovníkům. Podobný případ je i předmětem této práce.

2.1.3 Znalostní chatbot

Jde o chatbota, který poskytuje informace ze znalostní báze. Znalostní báze uchovává komplexní strukturované informace z dané specifické oblasti. [5] V případě této práce bude doménu znalostí reprezentovat portfolio společnosti, které bude převedeno do formy databáze. Obvykle se pro orientaci ve znalostní bázi používají platformy pro správu znalostí. Tyto nadstavby nad bází mají vetšinou podobu online knihoven nebo jiných aplikací. [5] Úkol platformy v tomto případě zastoupí chatbot, který ve formě odpovědí na konkrétní dotazy zpřístupní dané informace.

Při srovnání chatbota s klasickými platformami pro management znalostí je možné všimnout si několika výhod:

Lepší organizace informací – Organizace informací výrazně ovlivňuje celkovou efektivitu při práci s bází. Ve standardních případech jsou informace uživatelům předávané pomocí online knihovny. Zde se mohou nacházet různé formy organizace informací, jako například stromová struktura, která data seskupuje do kategorií, podkategorií a dalších složek. Přehlednost této struktury je v přímé závislosti s prvotním návrhem, robustností a v neposlední řadě i úrovní údržby. Tyto faktory v čase výrazně ovlivňují to, jak je jednoduché se k datům dostat. Na druhé straně chatbot poskytuje přímé odpovědi s konkrétními informacemi ve formě konverzace. Uživatel musí pro získání informace vynaložit řádově menší úsilí než při prohledávání přes kategorie a soubory informací. [6]

Poskytuje jen nutné informace – Velké množství informací na jednom místě působí obecně zmatečně. Uživatelé mají většinou zájem dohledat právě jednu specifickou věc, která se skrývá pod nějakou kategorií, podkategorií nebo menší souborem dat. A zde nastává problém. Uživatel neví, kde přesně tuto informaci hledat. Časově náročné prohlížení různých podstránek má vliv na celkovou efektivitu práce a úspěšnost dohledání dané informace. Navíc musí uživatel opticky vytěsnit jiná nepotřebná fakta, která ho v danou chvíli mohou mást. Při vyhledávání zase často nastává situace zahlcení nerelevantními výsledky. Chatbot tyto problémy eliminuje jednoduchým typem odpovědí. Napíše jen to, co je potřeba a na ostatní fakta uživatelům poskytne možnost doplňujících otázek. [6]

Vyžaduje minimum školení – Pokud je platforma pro zprávu znalostí robustní, nastává problém s jejím ovládáním. Pro nové zaměstnance může být problém adaptovat se na systém, se kterým před tím nepřišli do kontaktu. Tuto situaci je možné řešit školením, které mezi zaměstnanci není příliš populární. Druhotné řešení přináší uživatelský manuál. V obou případech je ale nutné věnovat delší čas výuce práce se systémem. Chatbot tyto těžkosti nemá, jelikož klást otázky a číst odpovědi zvládne každý gramotný člověk. Při první práci s chatbotem stačí uživatele obeznámit s okruhem znalostí a s různými otázkami, na které chatbot zvládne odpovědět. To vyžaduje méně času než školení nebo studium uživatelského manuálu. [6]

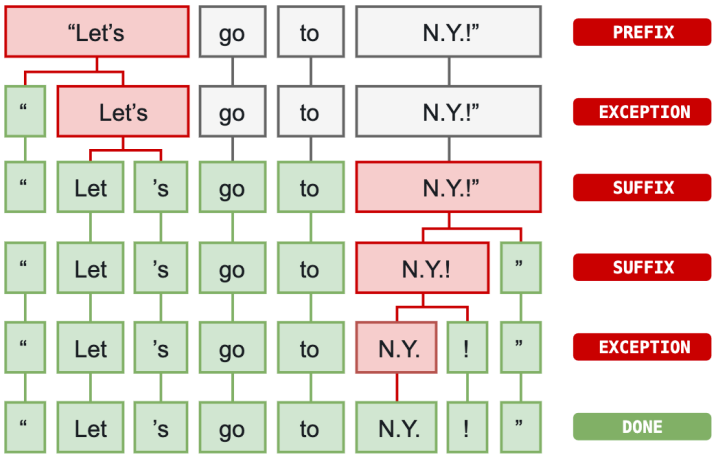
2.2 Zpracování přirozeného jazyka (NLP)

„Zpracování přirozeného jazyka (natural language processing – NLP) je analýza lingvistických dat, nejčastěji ve formě textu jako například dokumenty nebo publikace, použitím výpočetních metod. Cílem přirozeného zpracování jazyka je vytvořit takovou podobu textu, která přidává strukturu přirozenému jazyku za pomoci vhledů do lingvistiky.“ [7, překlad autora] Fakticky se jedná o část

oboru umělé inteligence, která zpracovává text za pomoci několika různých procesů a převádí ho do strukturované podoby. Mezi procesy, které se během zpracování využívají mimo jiné patří:

- Tokenizace
- Lemmatizace a stemming slov (*Word Stemming and Lemmatization*)
- Part-Of-Speech (POS) Tagging
- Chunking
- Mazání stop slov
- Rozpoznání intentů (*Intent recognition* – IR)
- Rozpoznání entit (*Named Entity Recognition* – NER)
- Porozumění přirozenému jazyku (*Natural language understanding* – NLU)
- Generování přirozeného jazyka (*Natural language generation* – NLG) [8, 9]

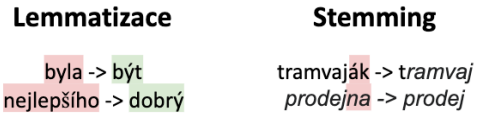
Tokenizace – Proces, u kterého se věta rozdělí na části, takzvané tokeny. Jde o sekvence znaků, které tvoří sémantickou jednotku vhodnou pro další zpracování. Musí se proto zahodit určité znaky, jako je například interpunkce, které k daným tokenům nepatří. [10] To, jak proces probíhá je znázorněno na obrázku 2.1.



Obrázek 2.1: Vizualizace procesu tokenizace

[11]

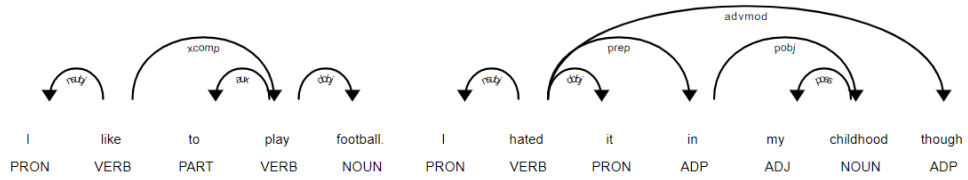
Lemmatizace a stemming slov – Stemming je proces, u kterého dochází k odstraňování odvozených tvarů slov pomocí odstraňování částí, jako jsou například předpony a přípony. Na druhé straně během lemmatizace se z vyskloňovaných tvarů slov prostřednictvím slovníkové a morfologické analýzy získává základ slov, neboli lemma. [10] Praktická ukázka těchto procesů je na obrázku 2.2.



Obrázek 2.2: Vizualizace lemmatizace a stemmingu

[12]

Part-Of-Speech (POS) Tagging – Postup, který slovům (na úrovni tokenů) ve větě přiřazuje jejich odpovídající slovní druhy. [13] Názorně je tento proces přiblížen na obrázku 2.3.



Obrázek 2.3: Vizualizace POS Taggingu

[14]

Chunking – Proces, který z textu extrahuje fráze místo samostatně stojících tokenů, které v některých případech nenesou skutečný význam. Příkladem je sdružené pojmenování „Jižní Afrika“, které by ve formě jednotlivě stojících slov „Jižní“ a „Afrika“ ztratilo svůj původní význam. [15]

Mazání stop slov – Stop slova jsou pojmy, které mají velmi malou hodnou z hlediska významu textu. V textech se vyskytují často, a proto je jejich rozeznání založeno na seřazení dle počtu použití v daném dokumentu. V češtině mají stop slova velké zastoupení u předložek a spojek. [10, 16]

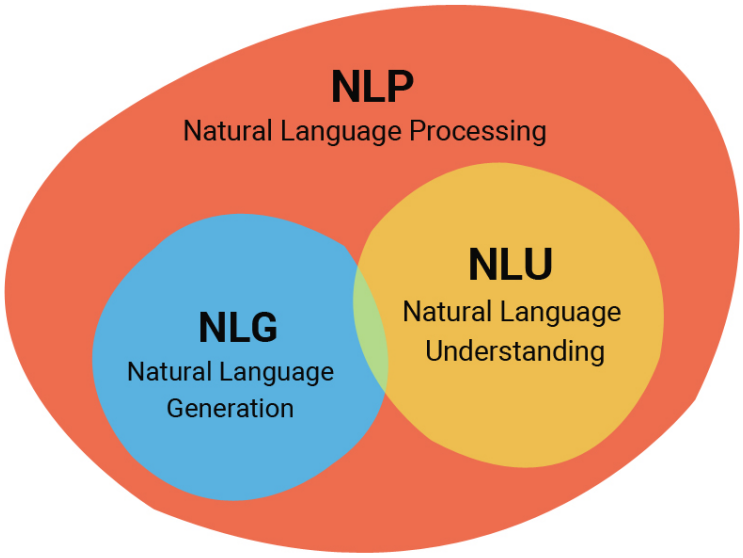
Intent – Intent vypovídá o záměru uživatelského textového vstupu. Například intentem dotazu „Kolik je teď v Praze stupňů?“ je zjištění počasí. [17]

Entita – Entita je informace získaná z uživatelského textového vstupu. Má za cíl doplnit intent o konkrétní informace, které uživatele v rámci daného účelu zajímají. Entitami v předešlém příkladu „Kolik je teď v Praze stupňů?“ jsou „teď“ představující časový údaj a „Praze“, která nese informaci o místě. [18]

2.2.1 Porozumění přirozenému jazyku (NLU)

Porozumění přirozenému jazyku umožňuje počítačům chápat textový vstup. Rozeznává jednotlivé odlišnosti ve formulaci vět a zaměřuje se na zjištění významu daného textu. Pro lepší pochopení problematiky je uveden příklad – věty „Jak se máš?“ a „Jak se vede?“ se významově ptají na tu stejnou věc, jen jinou formou. Hlavním cílem NLU je rozpoznat význam textu bez ohledu na jeho formu.

V různých publikacích jsou pojmy NLP a NLU často zaměňovány. I když je NLU klíčovou částí NLP, je možné mezi nimi najít důležitý rozdíl. NLP se v zásadě dívá na text z pohledu obsahu a zaměřuje na strukturu jazyka. Na rozdíl od NLU nám neřekne nic o významu textu. Obecně je možné v přirozeném jazyce vyjádřit stejné myšlenky různými způsoby. Proto je pro vývoj chatbota, který je schopen porozumět konverzaci s člověkem, nutné použít právě i NLU. Bez ní by chatbot nebyl schopen porozumět záměrům dotazů uživatelů. Pro shrnutí – použití NLU ovlivní míru porozumění psaného textu chatbotem. [8] Vizualizace prostřednictvím množin na diagramu 2.4 napomáhá ujasnit vztahy mezi NLP, NLU a NLG.



Obrázek 2.4: Členění NLP

[19]

2.2.2 Generovaní přirozeného jazyka (NLG)

Generování přirozeného jazyka umožňuje počítačům měnit strukturovaná data na text. Obecně existuje několik forem, jak toho dosáhnout. Mezi ty nejstarší patří vyplňování šablon, neboli vět s jasně danou strukturou. Součástí této struktury jsou data, která jsou během procesu nahrazeny hodnotami z datových zdrojů. [20] Pro lepší porozumění je uveden příklad – „V {město} je {teplota} stupňů“, kde město a teplota budou nahrazeny konkrétními hodnotami.

Tento způsob byl později rozšířen o možnost využití skriptů, které NLG obohacovalo o podmínky, cykly a přístupy k různým knihovnám. I když byl tento krok výrazným zlepšením v porovnání s pouhým vyplňováním šablon, stále zde chyběly možnosti lingvistiky a schopnost generovat rozsáhlejší text. Logickou nadstavbou tohoto principu bylo přidání gramatických funkcí na úroveň slov, které se vypořádají morfologií i neočekávanými výjimkami. Vyšší formu NLG tvoří tzv. dynamické NLG, které vytváří věty (i dokumenty) na základě lingvistické struktury. Tento přístup umožňuje vývojářům upustit od psaní exaktních šablon a skriptů pro jejich vyplňovaní. [20]

2.3 NLP nástroje

Obecně se NLP nástroje, které zde budou zmíněny, zaměřují na část porozumění přirozenému jazyku (NLU). Neposkytují možnost vytvářet konverzační scénáře, které by chatbotům poskytovaly možnost udržování kontextu nebo tvorbu odpovědí. O tuto část se při použití těchto nástrojů musí starat samotný vývojář chatbota.

2.3.1 Wit.ai

Wit.ai je open-source NLP nástroj od společnosti Facebook vyvinutý v roce 2013. Jeho použití pro osobní i komerční účely je zdarma, což je velkou výhodou pro menší i středně velké projekty. Komunita této platformy překročila hranici 100 tisíc členů. Na rychlém vývoji se podílí primárně tato početná skupina fanoušků. Mezi základní funkce patří rozpoznávání intentů a extrakce entit. Pro vybrané jazyky jsou dostupné předdefinované entity jako čas, datum nebo e-mail. O jejich extrakci se stará knihovna Duckling. Pro trénování rozpoznávání vlastních intentů a entity je potřeba je naplnit testovacími daty – větami, které spadají pod dané téma. Této potřebě napomáhá dostupný soubor zpráv, který zaznamenává všechny zpracovávané zprávy. Ty pak mohou být použité ke zlepšování přesnosti odhadů jednotlivých intentů. Celý trénink, specifikaci intentů a další práci s nástrojem je možné vykonávat pomocí dostupné webové aplikace s přehledným uživatelským prostředím. Pro automatizaci úkonů a propojení s chatbotem je poskytnuto i rozsáhlé API. Je

potřeba vyzdvihnout i velkou jazykovou podporu. Dostupných je více jak 130 jazyků. [21, 22, 23]

2.3.2 LUIS

Language Understanding Inteligent Service (LUIS) je služba od společnosti Microsoft, která využívá strojové učení k odhadu významu textu v přirozeném jazyce. Výstupem jsou relevantní a detailní informace o daném textu. Služba běží v cloudovém prostředí a vývojářům poskytuje API rozhraní. Obsahuje předpřipravené modely zaměřené na specifickou doménu, jakou je například rezervace místa v restauraci nebo ovládání chytrých zařízení v domácnosti. LUIS nabízí možnost vytvářet i vlastní modely, které fungují samostatně i v kombinace s předpřipravenými modely. [24]

2.3.3 Dialogflow

Dialogflow je NLP nástroj od společnosti Google. Je dobře škálovatelný, protože běží na Google Cloud Platform. Podporuje více jak 20 jazyků, ale čeština mezi nimi není. Mezi jeho klíčové výhody, kterými se odlišuje od konkurence, patří například vestavěná analytika, sentimentální analýza a autokorekce. Analytika poskytuje náhled do konverzací a nabízí tak možnost optimalizace rozeznání jednotlivých intentů. Skóre získané sentimentální analýzou může být použito jako rozhodující element k přesměrování nespokojených zákazníků na živé operátory. [25]

Dialogflow je dostupný ve dvou verzích – standardní a firemní. Standardní edice je zdarma a vážou se k ní omezení počtu volání jednotlivých služeb. Firemní balíčky jsou zpoplatněny, přičemž poskytují větší škálu funkcí a štědřejší až neomezené limity pro jednotlivé funkce. [25]

2.4 Nástroje pro tvorbu konverzací

V porovnání s již zmíněnými NLP nástroji, umožňují nástroje pro tvorbu konverzací vytvářet jednotlivé scénáře průběhů komunikace chatbota s uživatelem. Zde zmíněné platformy propojují vlastní NLP nástroje s frameworkem pro vytváření chatbotů. Poskytují tak vývojářům téměř komplexní balíček pro návrh a implementaci. V práci bude použit nástroj Rasa, který byl již autorem práce několikrát použit, zejména kvůli své spolehlivosti. Další důvody této volby budou zmíněny v samostatné sekci. Pro porovnání bude zmíněna i jiná podobná platforma.

2.4.1 Botpress

Botpress je open-source platforma, která v sobě spojuje potřebné nástroje pro tvorbu chatbotů. Mezi základní součásti patří:

- NLU
- administrační rozhraní
- vizuální editor chování
- emulátor chatu
- podporu chatových kanálů

Mezi její hlavní výhody se řadí možnost provozu na vlastní serverové infrastruktuře a nulová externí závislost. Nevýhodou je chybějící podpora českého jazyka v rámci vestavěného NLU. [26]

2.4.2 Rasa

Rasa je nástroj pro tvorbu a vylepšování chatovacích a hlasových asistentů založený na umělé inteligenci. Skládá se z Rasa Open source a Rasa X. První ze jmenovaných má na starost porozumění psanému textu, tvorbu dialogů založenou na strojovém učení a integraci pro známé chatovací kanály. [27] Rasa X je administrační rozhraní, neboli webová aplikace, která slouží k trénování modelů strojového učení za pomoci anotování konverzací. Pro tyto účely uživatelům zpřístupňuje možnosti vytváření jednotlivých intentů, entit, průběhů konverzací a dalších nutností pro vylepšování modelů. [28] Od svého představení v roce 2016 Rasa získala více jak 2 miliony stáhnutí a víc jak 400 přispěvatelů, kteří se aktivně podílí na vývoji. Mezi její hlavní přednosti patří: [29]

Spojení s výzkumem – Díky investicím do výzkumu získává přistup k posledním studiím věnujícím se tvorbě konverzační umělé inteligence, které umožňují i vývojářům bez jakéhokoliv výzkumného týmu stavět chatovací asistenty. [30]

Špičkové NLU – Technologie, která mimo klasické pochopení zpráv a zjišťování intentů poskytuje vývojářům možnost získávat informace o kontextu, zpracovávat naráz více intentů a používat jak předtrénované, tak speciálně vytvořené entity. [30] Navíc podporuje mnohé jazyky, mezi kterými je při specifickém nastavení pipelines a dostatku testovacích vstupů i čeština, jelikož v tomto případě se nepoužívá model založený na jazyku. [31]

Pokročilá správa dialogů – Dialogy, které dokážou chatboti postavení na této platformě vést, jsou založeny na strojovén učení z reálných konverzací. Navíc si umí udržovat kontext a díky tomu vést smysluplnou konverzaci. [30]

Deployment – Je nabízeno několik možností hostingu, včetně běhu na vlastních serverech nebo na cloudu od poskytovatelů třetích stran. [30]

Integrace – Nabízí možnost propojení již existujících systémů a bází znalostí prostřednictvím API. [30]

Chatové kanály – Dostupná je široká škála konektorů pro standardní chatovací aplikace, ale též integrace pro weby a vlastní kanály. [30]

Právě kvůli těmto benefitům a předchozím zkušenostem byla zvolena tato platforma. Z osobní praxe bych rád zdůraznil pozitivní a promptní přístup k návrhům na zlepšení, opravám chyb a dotazům na GitHubu. Je vidět, že jak open-source komunita, tak samotní vývojáři v Rasa věnují dostatek času a úsilí pro kontinuální rozvoj této platformy. Dále je nutno podotknout, že v době zvýšené pozornosti na bezpečnost a ochranu osobních údajů je výrazným benefitem dříve zmiňovaná možnost vlastního hostingu.

2.4.2.1 Rasa NLU

Rasa NLU je část věnující se rozpoznávání psaného textu. Používá k tomu celou řadu komponent, počínaje vektorizovanými slovníky (jen v některých jazycích), přes tokenizery až po klasifikátory intentů a extraktory entit. Tyto jednotlivé komponenty jsou spojeny do tzv. „pipeline“. Jde o posloupnost jednotlivých modulů, pomocí kterých z testovacích dat vzniká model sloužící ke klasifikaci intentů a extrakci entit. Trénovací data jsou zapsána ve formátu yaml (ukázka 2.1). Jejich specifikace je složena z několika částí: [32]

Příklady – Různé modifikace vět týkající se stejného intentu. V rámci těchto vět je možné označovat i jednotlivé entity, které později také napomáhají predikci intentů.

Synonyma – Přiřazují extrahované entity ke stejnému klíči, jelikož různé modifikace slov mohou odkazovat na stejnou věc.

Regulární výrazy – Napomáhají detekci intentů a extrakci entit v případě, že mají deterministickou strukturu (e-mail, telefonní číslo, PSČ...)

Vyhledávací tabulka – Jde o seznam příkladů jedné entity, který společně vytváří regulární výraz hledající přesnou shodu v trénovacích datech. [32]

2.4.2.2 Rasa Core

Rasa Core používá několik různých nástrojů pro řízení dialogů. Nosná část je postavena na „Stories“, které reprezentují jednotlivé průchody konverzací a slouží k natrénování modelu rozhodujícím o řízení dialogů. Konkrétně se jedná o strukturovaný zápis konverzace sestávající z odkazů na jednotlivé intenty, entity a akce. Tyto scénáře jsou zapsané ve formátu yaml (ukázka 2.2). Výhodou tohoto způsobu řízení konverzace je odstranění potřeby popsat každý

možný průchod dialogu vázaný na konkretní zprávu. Díky výstupu z NLU je průchod vázán na extrahované entity a intenty, na základě kterých se predikují odpovědi a žádané akce. [33]

Akcemi se rozumí kroky, které chatbot vykoná jako reakci na určitou zprávu. Jsou známé 2 typy akcí – výrokové (*utterance*) a vlastní. Výrokové akce jsou předpřipravené textové zprávy, které chatbot může odesílat bez nutnosti dalšího programování. Na druhou stranu vlastní akce odkazují na kód, který je při jejich zavolání spuštěn. [33] Zde je prostor pro implementaci komunikace s API, databází nebo pokročilejší zpracování vstupu.

```
## intent:check_balance
- Jaký je můj zůstatek?
- Kolik mám na [spoření](source_account)
- Kolik peněz mám na [spořícím učtu](source_account:savings)
- Můžu platit v~[eurech](currency)?

## synonym:savings
- prasátko

## regex:zipcode
- [0-9]{5}

## lookup:additional_currencies
path/to/currencies.txt
```

[32]

Ukázka kódu 2.1: Zápis trénovacích dat

```
## Story 01
* greet
    - utter_greet
* ask{"topic": "payment"}
    - action_payment_methods
    - utter_offer_other_help
* deny
    - utter_thankyou
    - utter_goodbye
```

[33]

Ukázka kódu 2.2: Zápis story

Kapitola 3

Analýza

Tato kapitola se zabývá analýzou požadavků a uživatelských případů použití. Jedná se o důležitou fázi ve vývoji softwaru, při které je zapotřebí klást důraz na důslednost zpracování. Díky tomu je možné předcházet chybám, které jsou v pozdějších etapách vývoje časově náročnější na opravu. Výsledky analýzy poslouží jako nosná struktura pro návrh samotného programu.

3.1 Požadavky

Požadavky se pro potřeby práce dělí na funkční a nefunkční. Formulace konkrétních bodů vznikla na základě několika konzultací se společností, která vytvoření chatbota poptává. Jednotlivé funkční požadavky zohledňují nároky uživatelů na výsledné chování řešení. Na druhou stranu nefunkční požadavky definují kvalitu a vlastnosti daného systému.

3.1.1 Funkční požadavky

F1: Chatbot bude přijímat textový vstup
Chatbot bude připraven pro příjem výhradně textového vstupu. Není potřeba zpracovávat multimediální vstupy jako například obrázky a videa.

F2: Uživatel bude moci chatbotovi položit dotaz
Uživatel bude moci chatbotovi kdykoliv položit dotaz týkající se portfolia společnosti. Chatbot z dotazu extrahuje informace jako jsou entita zájmu, její typ, atributy zájmu a intent dotazu. Ty strukturovaně uloží pro pozdější tvorbu vyhledávacího dotazu.

F3: Chatbot bude udržovat během konverzace kontext
Chatbot si bude po zodpovězení dotazu nadále udržovat kontext konverzace v podobě entity zájmu, pro případ navazujícího dotazu. Uživatel následovně může položit dotaz bez zbytečného opětovného zpřesňování.

F4: Chatbot bude získávat informace z databáze
Při sestavování databázového dotazu chatbot použije uložené údaje. Odpověď z databáze zpracuje a připraví pro tvorbu textové odpovědi pro uživatele.

F5: Chatbot bude vytvářet textové odpovědi
Chatbot připravenou šablonu odpovědi doplní o zpracované informace získané z databáze. Vznikne tak věcná odpověď doprovázená vhodným textem.

F6: Chatbot bude reagovat na neúplný dotaz
Chatbot bude na chybějící údaje reagovat dotazem poptávajícím jejich doplnění ze strany uživatele.

F7: Chatbot bude vytvářet návrhy na dotazy
Chatbot bude schopen uživateli odeslat nabídku ve formě tlačítek, které budou obsahovat návrhy na další rozvoj konverzace.

F8: Chatbot umožní přeformulovat dotaz
Chatbot po neúspěšném pokusu rozeznat intent dotazu, nabídne uživateli možnost dotaz kompletně přeformulovat.

F9: Uživatel se bude moci odkazovat na vypsané informace
Pokud uživatel obdrží odpověď ve formě seznamu, může se při dalším dotazu odkazovat na jednotlivé položky pomocí číslovek. Chatbot tak pozná, o kterou entitu se jedná.

F10: Uživatel bude moci manuálně resetovat konverzaci
Uživatel může v případě nouze konverzaci manuálně restartovat. Tato možnost vzniká pro krajní případ, kdy chatbot nebude moct opakovaně rozpoznat typ nebo téma dotazu.

F11: Chatbot bude rozlišovat mezi zaměstnancem a externistou
Chatbot bude rozlišovat uživatele na základě druhu spolupráce s firmou. Zaměstnancům zpřístupní informace v celém rozsahu, zatímco externistům zpřístupní jen externí verzi portfolia.

F12: Chatbot bude reagovat na neoprávněný dotaz
Pokud se externista zeptá na informace, které v rámci své spolupráce s firmou nemá zpřístupněné, chatbot bude reagovat vhodnou textovou zprávou.

3.1.2 Nefunkční požadavky

N1: Rozhraní
Jako rozhraní komunikace bude použita chatovací aplikace pro firmy – Microsoft Teams. Chatbot bude vystupovat jako samostatný uživatel, kterému bude moci kdokoliv, kdo má přístup k firemnímu prostředí, napsat. Přidávání do skupinových a týmových chatů bude zakázáno. Chat bude fungovat jen v režimu osobní (one-to-one) komunikace.

N2: Rozšiřitelnost
Znalostní databáze, na kterou bude chatbot napojen, by měla být v budoucnu rozšiřitelná o další záznamy, entity a atributy bez výrazného narušení funkčnosti a struktury chatbota.

N3: Nezávislost
Chatbot by měl fungovat nezávisle na výstupním kanálu (chatovací platformě).

N4: Bezpečnost dat
Přístup do znalostní databáze bude zabezpečen.

3.2 Případy užití

Případy užití ověřují funkční a nefunkční požadavky na daný software. Popisují možné scénáře toho, jak může uživatel a systém vzájemně interagovat v určitých podmínkách. Jednotlivé případy sestávají z posloupností kroků, které mají v závěru přinést určitý cíl. Je nutno zohlednit všechny procesy systému, které mohou uživatele nějakým způsobem ovlivnit. [34]

3.2.1 Aktéři

Uživatel
Je abstraktní neboli obecný aktér, který může s chatbotem komunikovat. Jelikož jde o abstraktní třídu, nejsou zde definována práva pro přístup k informacím z jednotlivých sekcí portfolia.

Externista
Je aktér, který má přístup k externí verzi portfolia. Může se ptát na produktové pilíře, produkty a produktové moduly společnosti.

Zaměstnanec
Je aktér, který má zpřístupněné informace z portfolia v celém interním rozsahu.

3.2.2 Seznam případů užití

Vizualizaci jednotlivých vztahů je možno pozorovat na diagramu případu užití 3.1.

UC1: Vypsat pilíře

1. Uživatel do chatu napíše dotaz, který poptává výpis pilířů, například: „Ukaž mi produktové pilíře“.
2. Chatbot vyhledá v databázi informace.

3. Chatbot uživateli odešle strukturovanou textovou odpověď ve formátu číslovaného seznamu.

UC2: Vypsat produkty

1. Uživatel do chatu napíše dotaz, který bude poptávat výpis produktů patřící danému pilíři, například: „Které produkty máme v nabídce?“.
2. Chatbot položí doplňující dotaz spolu s výpisem pilířů, aby zjistil, kterého pilíře se produktový dotaz týká.
3. Uživatel se odkáže svou odpovědí na jeden z nich, například: „Je to jednička.“
4. Chatbot vyhledá v databázi informace.
5. Chatbot uživateli odešle strukturovanou textovou odpověď ve formátu číslovaného seznamu.

UC3: Vypsat produktové moduly daného produktu

1. Uživatel v chatu vidí vypsaný číslovaný seznam produktů.
2. Aby uživatel nemusel název produktu při dalším dotazu kopírovat, může se na něj odkázat pomocí číslovky, například: „Které produktové moduly nabízíme v rámci prvního produktu?“.
3. Chatbot číslovku nahradí odpovídajícím produktem.
4. Chatbot vyhledá v databázi informace.
5. Chatbot uživateli odešle strukturovanou textovou odpověď ve formátu číslovaného seznamu.

UC4: Získat informace o lidech v oddělení Consulting

1. Uživatel do chatu napíše dotaz na pracovníky, který se váže k předtím diskutované technologii, například: „Kdo tam pracuje v oddělení Consulting?“
2. Chatbot vyhledá v databázi informace.
3. Chatbot uživateli odešle strukturovanou textovou odpověď ve formátu číslovaného seznamu.

UC5: Získat informace o kapacitě na Slovensku

1. Uživatel do chatu napíše dotaz na kapacitu, který se váže k předtím diskutovanému produktovému modulu, například: „Kolik lidí zde na Slovensku pracuje?“.
2. Chatbot položí doplňující dotaz spolu s výpisem slovenských technologických dodavatelů, aby zjistil, ke kterému se dotaz váže.
3. Uživatel se odkáže na jednu z technologií.
4. Chatbot vyhledá v databázi informace.

5. Chatbot uživateli odešle strukturovanou textovou odpověď ve formátu číslovaného seznamu.

UC6: Získat podrobnosti o produktovém modulu

1. Uživatel do chatu napíše dotaz, který bude poptávat detaily modulu, například: „Chci vidět detaily produktového modulu X“.
2. Chatbot vyhledá v databázi informace.
3. Chatbot uživateli odešle strukturovanou textovou odpověď se všemi dostupnými detaily modulu.
4. Chatbot nabídne další možnosti rozvoje komunikace pomocí tlačítek.

UC7: Získat kontakt na konkrétního zaměstnance

1. Uživatel při vypsaném seznamu pracovníků položí dotaz, který poptává informace kontaktu, například: „Jak mu mohu napsat pánovi X?“
2. Chatbot vyhledá v databázi informace.
3. Chatbot uživateli odešle strukturovanou textovou odpověď se všemi dostupnými detaily daného pracovníka (včetně kontaktu).

UC8: Položit neočekávaný dotaz

1. Uživatel položí neočekávaný textový dotaz, například: „Jaké je venku počasí?“.
2. Chatbot není připraven reagovat na dotazy, které se netýkají produktového portfolia. Vytvoří proto textovou odpověď, která uživatele navede k přeformulování položeného dotazu, například: „Bohužel dotazu nerozumím. Prosím zkuste ho přeformulovat.“
3. Pokud chatbot disponuje alespoň nepřesnými odhady intentu dotazu, odešle je formou tlačítek jako nabídku možností.

UC9: Resetovat konverzaci

1. Uživatel do chatu pošle text s intentem restartovat konverzaci. Tento text může mít různou formu, například: „Prosím znovu“, „Restartuj konverzaci“ a podobně.
2. Chatbot tento příkaz zpracuje tak, že odstraní všechny uložené strukturované informace.
3. Chatbot uživateli odešle uvítací zprávu.

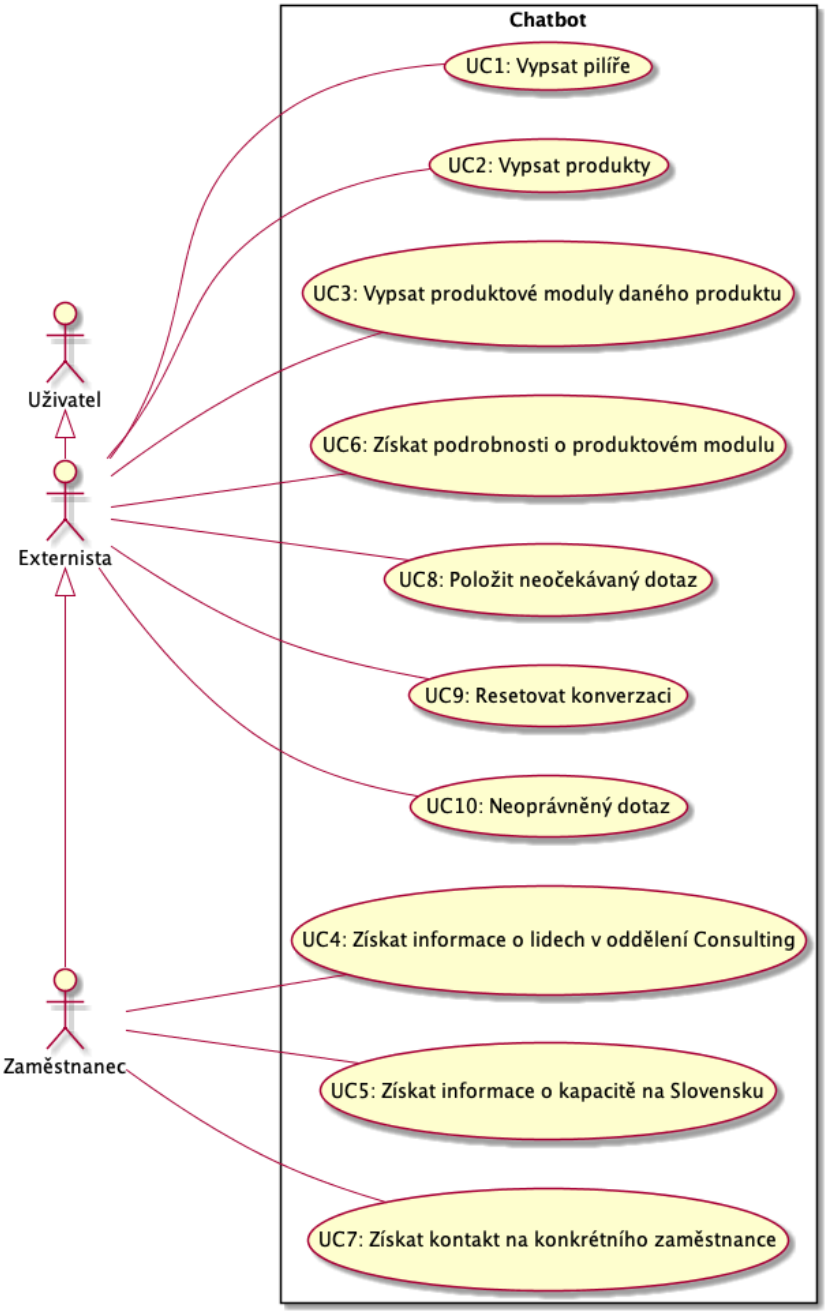
UC10: Neoprávněný dotaz

1. Externista do chatu položí dotaz, který se týká interního portfolia, například: „Kolik máme zkušeností s modulem X?“
2. Chatbot zjistí, že externista pro tyto informace nemá oprávnění.
3. Chatbot uživateli odešle informační zprávu o chybějících právech, například: „Omlouvám se, ale k těmto informacím nemáte oprávnění.“

3.2.3 Tabulka pokrytí

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
UC1	✓	✓		✓	✓						✓	
UC2	✓	✓	✓	✓	✓	✓			✓		✓	
UC3	✓	✓	✓	✓	✓				✓		✓	
UC4	✓	✓	✓	✓	✓						✓	
UC5	✓	✓	✓	✓	✓	✓					✓	
UC6	✓	✓	✓	✓	✓		✓				✓	
UC7	✓	✓	✓	✓	✓		✓				✓	
UC8	✓	✓			✓		✓	✓				
UC9	✓	✓			✓					✓		
UC10	✓	✓			✓						✓	✓

Tabulka 3.1: Tabulka pokrytí případů užití



Obrázek 3.1: Diagram případů užití

Kapitola 4

Návrh

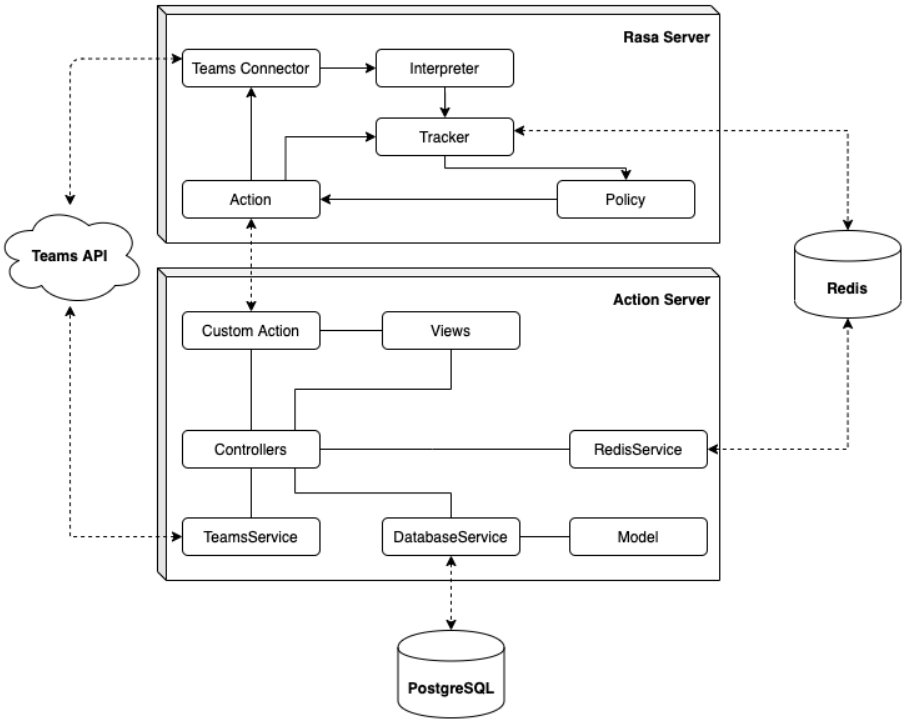
Tato kapitola bude věnována návrhu systému. Jde o proces, který přetváří zpracovanou analýzu do konkrétní realizovatelné podoby. Jeho součástí je popis architektury systému, segmentace na moduly nebo menší komponenty a modelování toku dat systémem. [35] Při zadefinování jednotlivých modulů budou zmíněny i konkrétní technologie, které budou v implementační části použity.

4.1 Architektura

Architektura systému je vnímaná jako nejvyšší možná abstrakce, která systém rozděluje na moduly. Ty detailně popisuje na základě jejich funkčnosti. Specifikuje také možné vztahy a vzájemné interakce, a tak dotváří obraz o celkovém chodu systému. Zrcadlí v sobě požadavky na software s přihlédnutím na všechny atributy kvality. Jejím hlavním cílem je zohlednit požadavky, které mají vliv na strukturu aplikace. Správně položený základ architektury následně minimalizuje riziko problému při technickém vyhotovení. [36] Je to důležitý krok, který v praxi rozhoduje o dalších možnostech rozšiřování aplikace o nové funkcionality. V případě této práce je konkrétní podoba architektury ovlivněna architekturou Rasa Frameworku. Po jeho vzoru bude chatbot sestávat ze čtyř částí: Rasa Server, Action Server, PostgreSQL databáze a Redis. Tento návrh je v souladu s nefunkčními požadavky, které žádají bezpečnost dat, možnosti modifikace databáze a nezávislost na chatovací platformě.

4.1.1 Rasa Server

Rasa Server se stará o zprávu od jejího přijetí až po odeslání odpovědi uživateli. Používá při tom několik důležitých součástí, které zprávu v jednotlivých fázích zpracovávají. Součástí tohoto serveru jsou nástroje pro NLP a to konkrétně Stories a NLU soubor. To, jak tyto nástroje vypadají a fungují již bylo



Obrázek 4.1: Diagram architektury chatbota a komunikace mezi jeho částmi [37]

zmíněno v teoretické části. Zde je nutné jen připomenout, že soubory se Stories a NLU budou právě součástí Rasa serveru.

Pro vysvětlení toho, jak se zpráva zpracovává v jednotlivých krocích je připraven diagram 4.1 vizualizující architekturu chatbota a mezimodulovou komunikaci včetně toku zprávy.

1. Zpráva přijata na vlastní konektor pro MS Teams bude převedena do obecného formátu, který je používán napříč Rasa Serverem.
2. Tento obecný formát zprávy je postoupený části zvané Interpreter. Zde se za pomoci natrénovaného NLU modelu převede na objekt obsahující originální text, intent a také extrahované entity.
3. Tento objekt putuje do Trackeru. Ten sleduje aktuální stav konverzace, proto si uloží informaci o příchozí zprávě do externího Redis Tracker Storu (existuje také možnost interního úložiště, ale v praxi je kvůli robustnosti dat preferována právě externí varianta).
4. Policy získá aktuální stav z Trackeru a vykoná predikci další akce.

5. Predikovaná akce se zapíše do Trackeru.
6. Pokud jde o vlastní akci, Rasa Server požádá o její exekuci Action Server.
7. Po vykonání akce je odpověď odeslána přes vlastní konektor pro MS Teams zpátky uživateli prostřednictvím Teams API. [37]

Většina těchto procesů (až na primární procesy a odesílání zpráv pomocí vlastního konektoru) je vykonávána bez zásahů vývojáře. S tím souvisí výhoda nulové nutnosti dané činnosti spravovat nebo modifikovat. Na druhé straně v některých krajních případech může být tato skutečnost i nevýhodou. Během dalšího rozšiřování řešení se může vyskytnout situace, která bude vyžadovat modifikaci těchto procesů. Je pak na místě prostor k zamyšlení, jestli je požadovaný zásah nezbytně nutný a zda neexistuje jiná možnost řešení.

4.1.2 Action Server

Action Server zastává funkci exekuce vlastních programovatelných akcí. Rasa umožňuje tento server postavit na jakékoliv technologii od Node.js, přes PHP až po Python. Této možnosti dosáhli díky otevřenému HTTP API, které vývojářům dovoluje spouštět vlastní akce pomocí POST requestu odeslaného z Rasa Serveru. [38] Obsahem requestu je jméno další predikované akce, ID odesílatele, Tracker objekt reprezentující aktuální stav chatbota a Dispatcher, který obsahuje konfiguraci a parametry Rasa Serveru. Odpovědí ze strany Action Serveru jsou modifikace Trackeru a reakce na uživatelskou zprávu. [39]

Aby Rasa tyto náležitosti vývojářům ulehčila, vytvořila Rasa SDK obsahující nástroje v jazyku Python pro psaní vlastních akcí. Tento způsob je preferován, jelikož SDK je připraveno na všechny typy akcí, modifikace Trackeru a též na odesílaní reakcí uživateli. [40] Po zohlednění autorových zkušeností byl použit jazyk Python, kterého kód je jednoduše čitelný a dobře udržovatelný.

Jak diagram 4.1 naznačuje, Action server bude postaven na modifikované MVC architektuře. Při zavolání POST requestu z Rasa Serveru se začne vykonávat požadovaná akce. Ta zavolá Controller, který použije DatabaseService k získání dat z databáze. Ty převede na konkrétní Modely, které budou reprezentovat databázové entity. Tato získaná a předpřipravená data poskytne View, zařídí správné využití textů, formátování a konečnou podobu odpovědi pro uživatele. Dané předpřipravené zprávy se pomocí Dispatcheru uvnitř vlastní funkce odešlou zpátky na Rasa Server. Action server disponuje také přístupem na Teams API pomoci TeamsService, která se stará o zjišťování uživatelových detailů, které slouží k následné identifikaci druhu spolupráce (zaměstnanec/externista). Tato data ukládá pomocí RedisService na úložiště Redis z důvodu úspory transferu dat.

4.1.3 PostgreSQL databáze

Pro uložení informaci z portfolia byla zvolena databáze PostgreSQL. Při výběru technologie byly zohledněny požadavky na práci s daty – převážná většina

odpovědí na portfoliové dotazy jsou založeny na propojovaní jednotlivých tabulek za pomocí relací. Proto byla relační databáze preferovanou volbou. Je na místě zdůraznit i kompatibilitu s ORM nástrojem SQLAlchemy, který byl vybrán pro připojování a práci s databází. Jde o soubor funkcí s podporou objektového relačního mapování, který vývojářům ulehčuje práci z databází bez nutnosti používat SQL skripty. Nabízí kolekci známých perzistentních vzorů, které byly navrženy pro rychlý a efektivní přístup k databázi pomocí jazyku Python. [41]

4.2 Modely

Modely jsou v softwarovém inženýrství chápány jako jistá forma opisu, neboli abstrakce nad důležitými prvky systému. Měly by z různých pohledů pozvednout důležitost jednotlivých položek, opomenout zbytečné detaily a pomoct vývojářům vypořádat se s mohutností daného modelovaného problému. Obecně mohou modely zachycovat rozličné perspektivy na jeden a ten samý systém – popisovat vlastnosti problémové domény, organizaci kódu nebo databáze. [42]

V případě této práce je vhodné kromě obecné architektury rozebrat dva pohledy na systém: doménový a databázový.

4.2.1 Doménový model

Doménový model znázorňuje vztahy mezi jednotlivými položkami portfolia. Na jeho základu bude navržen relační databázový model. Tyto objekty zrcadlí i reálné použití objektů při implementaci Action Serveru, které budou rozšířeny o další potřebné metody.

Jak je v modelu 4.2 znázorněno, pilíře obsahují skupiny produktů. Stejný scénář je možno dál pozorovat i u produktů a produktových modulů. Produkt může mít svého vlastníka – vetšinou se jedná o osobu. Jednotlivé produktové moduly mohou být provozovány na různých typech distribucí. Samotné moduly jsou postaveny na technologii (službě), kterou poskytuje určitý dodavatel. Jeden dodavatel může poskytovat více technologií pod různými značkami. U služeb je dále evidována míra zkušeností, krajina ve které se jí věnují a fakt, jestli je daná služba poskytována s pomocí externích zdrojů. Na službách se dále podílí jistá pracovní síla – lidé nebo celé týmy. Každý má svoje jméno, kontaktní údaj v podobě e-mailu a příslušnost k oddělení. Lidi a týmy se od sebe liší na základě kapacity.

4.2.2 Relační databázový model

Relační databázový model definuje datové struktury a operace nad nimi. V databázi jsou data a relace organizovány do tabulek, které představují soubory záznamů se stejným formátem. [43]

Databázový model 4.3 byl vytvořen na základě předchozího doménového modelu, pomocí dekompozice „many-to-many“ vztahů. Vznikla tak tabulka distribuce, která propojuje typ s produktovým modelem. Stejnou metodou vznikla i tabulka spolupráce, která vznikla dekompozicí vztahu služby a pracovní síly. Model neobsahuje vůči doménovému modelu další výrazné změny. Je ale nutno zmínit přítomnost sloupců typu Enum, konkrétně *departmentenum* a *countryenum*. Tato data mohou nabývat hodnoty jen z předem připravené skupiny – (cz, sk) pro *countryenum* a (consulting, operations, external) pro *departmentenum*.

4.3 Uživatelské prostředí

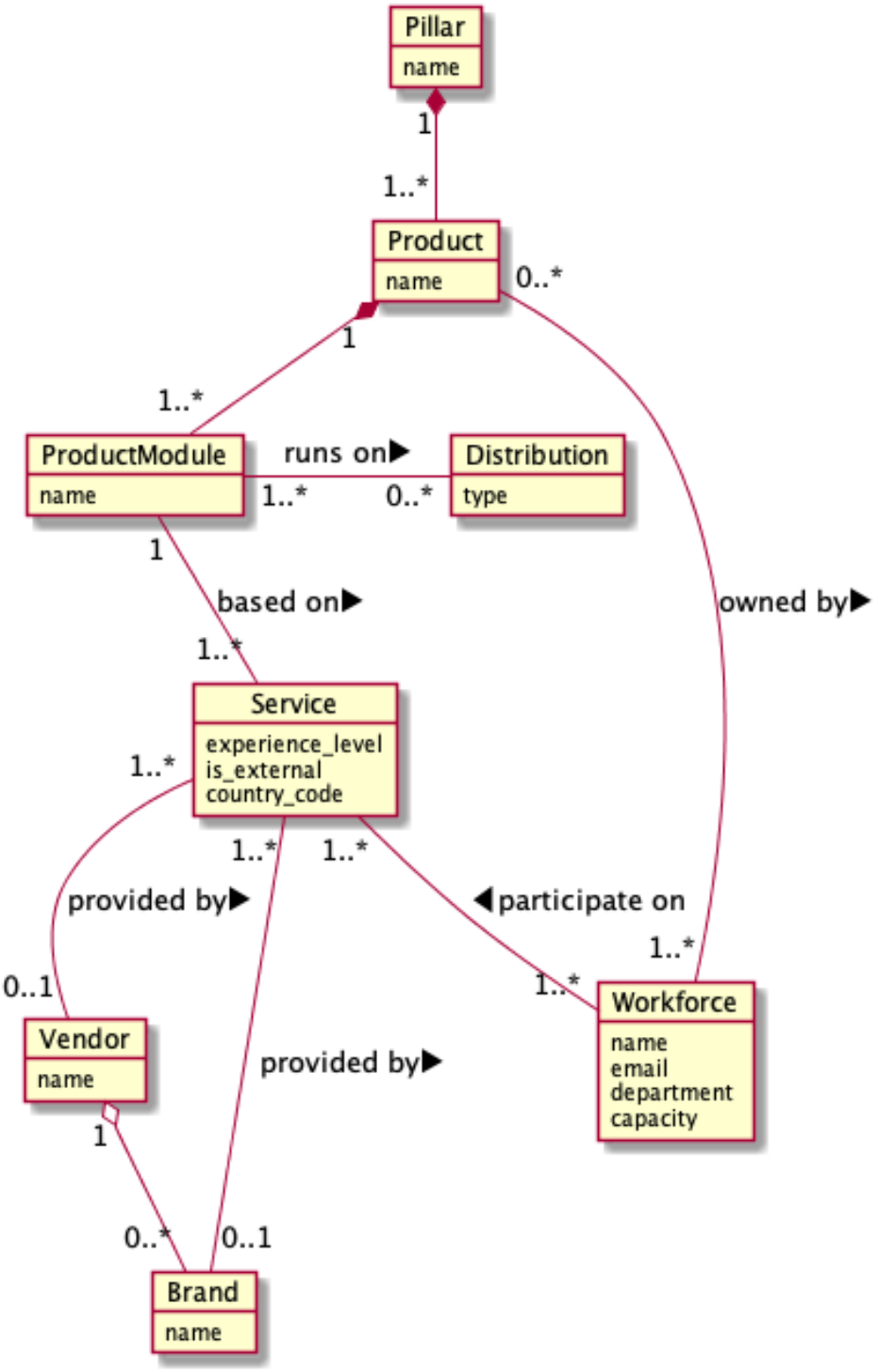
Uživatelské prostředí v případě chatbota tvoří platforma, pomocí které uživatelé komunikují. V tomto případě jde o již zmiňovaný MS Teams. Od této platformy se odvíjí i vzhled textových zpráv a dalších prvků chatbota jako jsou například tlačítka. Platforma Teams má vlastní nativní i webové aplikace, které dovolují uživatelům komunikovat s chatbotem na jakékoliv platformě. Tyto rozdílné platformy ale výrazně neovlivňují vzhled uživatelského prostředí – rozdíly jsou viditelné jen při uzpůsobení jednotlivých ovládacích prvků pro rozdílné velikosti displejů (možno pozorovat na obrázcích 4.4 a 4.5).

Teams podporuje odesílání čtyř typů komunikačních prvků směrem k uživateli. Patří mezi ně:

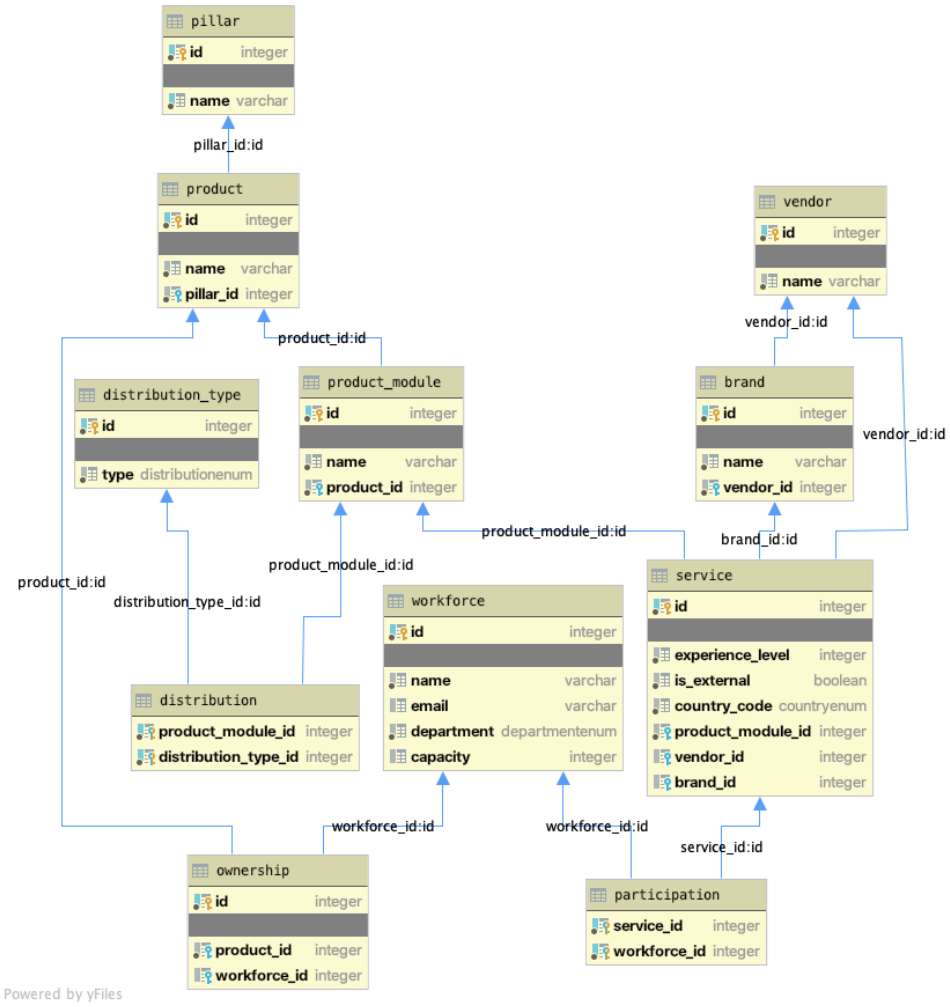
- textová zpráva
- obrázek
- karta
- emoji [44]

Textová zpráva navíc přináší možnosti formátování. Mezi podporované typy patří tučné písmo, kurzíva, hypertextové odkazy a mnohé další. I takto formátované zprávy mohou být součástí již zmiňovaných karet. [45]

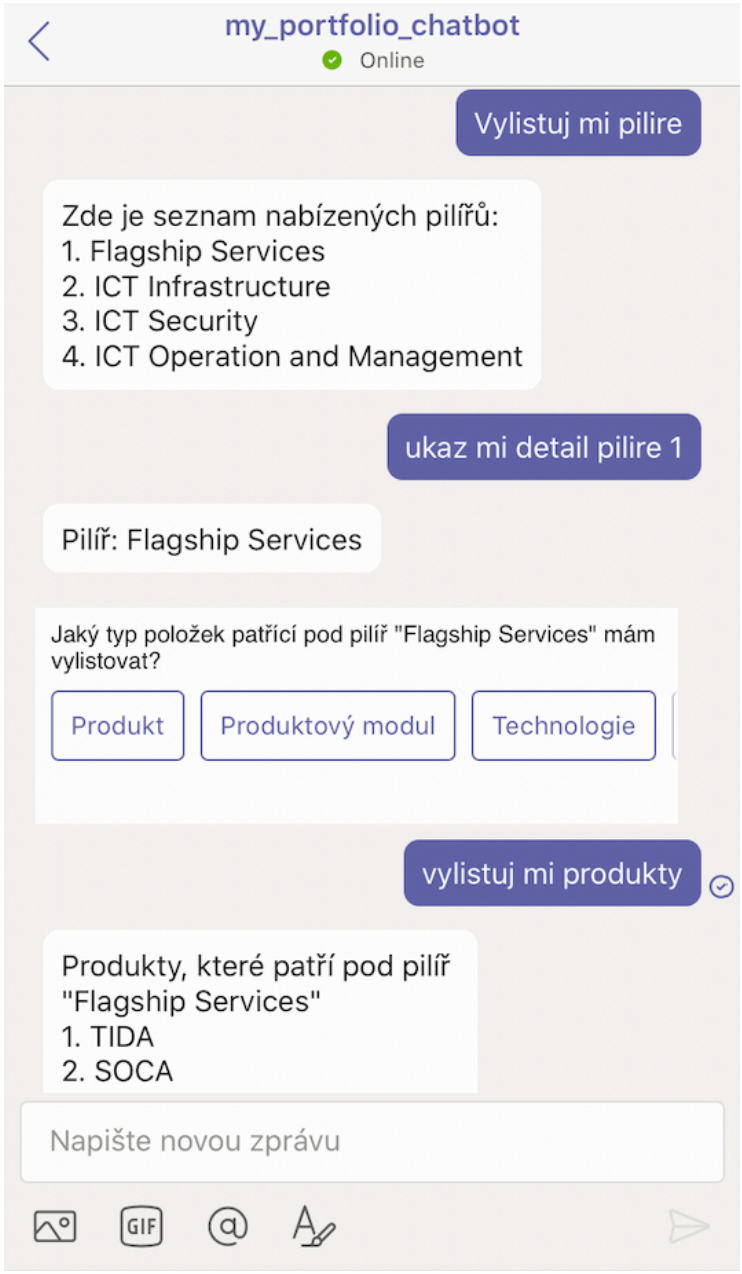
Kartu lze čtenáři přiblížit jako prvek v chatu, který v sobě kombinuje komponenty jako je text, tlačítka, obrázky a další. Pro účely práce budou použity především tlačítka, které mohou uživatele nasměrovat při výběru odpovědi. [46] Pro ukázku některých zmiňovaných prvků přímo v chatu jsou připraveny obrázky. Příklad uživatelské konverzace za použití textových prvků a tlačítek na mobilních platformách reprezentuje obrázek 4.4. Pro přiblížení vzhledu chatu v desktopové aplikaci je dostupný obrázek 4.5.



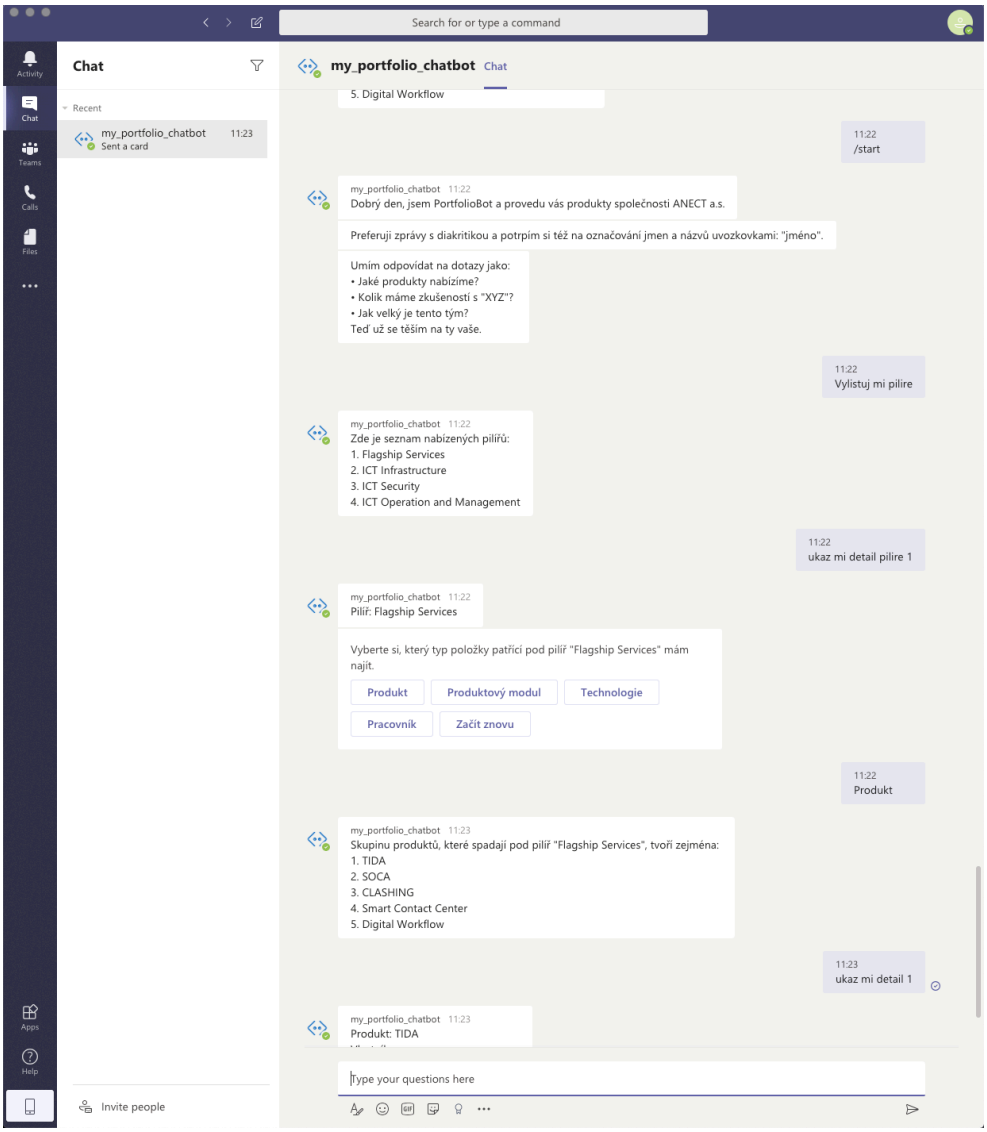
Obrázek 4.2: Doménový model



Obrázek 4.3: Relační databázový model



Obrázek 4.4: Ukázka konverzace v Teams na mobilu



Obrázek 4.5: Ukázka konverzace v Teams na počítači

Kapitola 5

Implementace

Tato kapitola čtenáři přiblíží fungování jednotlivých komponent uvnitř kódu. Její dělení kopíruje rozdělení aplikace, což přispívá k lepšímu pochopení problematiky. Budou zde primárně popsány mechanizmy chodu těch tříd, které se starají o správnou funkčnost chatbota.

5.1 Model

Modely v aplikaci zastupují funkci databázových tabulek. Jsou na nich namapované pomocí již zmiňovaného ORM nástroje SQLAlchemy. Pro tento účel byla vytvořena rodičovská třída pro databázové objekty s názvem *KnowledgeBaseModel*. Tato třída poskytuje svým potomkům potřebné rozhraní v podobě metod a atributů. Přispívá to ke generičnosti řešení, které je klíčové pro chod aplikace a její rozšiřování do budoucna.

5.1.1 KnowledgeBaseModel

Jelikož budou všechny databázové modely identifikovány pomocí primárního klíče v podobě *id*, bylo potřeba tento atribut zařadit pod rodičovský model, jak je v kódu 5.1 znázorněno. Navíc každý model dle dokumentace SQLAlchemy potřebuje svůj *__tablename__* pro jednoznačnou identifikaci. Ten se bude generovat pomocí konverze názvu třídy, která je zapsaná v tzv. *CamelCase* do formátu *SnakeCase* – standard pro jména tabulek. Odstraní to povinnost repetitivního opisování názvů. Navenek je zpřístupněn pomocí veřejné statické metody. Dále je třída rozšířena o metody *get_list_text* a *get_object_text*, které sestavují odpovídající texty při zasílání zpráv. Přidaný atribut s názvem *filter_attribute* může sloužit k filtrování při vyhledávání v databázi. Bude obsahovat jméno sloupce databáze, pomocí kterého se filtrovací podmínka bude sestavovat. Další atribut s jménem *attributes* obsahuje seznam objektů, na které se uživatel v následující otázce může zeptat.

```python
class KnowledgeBaseModel(object):

  id = Column(Integer, primary_key=True)

  @declared_attr
  def __tablename__(cls) -> Text:
    return re.sub(r'(?<!^)(?=[A-Z])', '_', cls.__name__).lower()

  filter_attribute = None
  attributes = []

  @classmethod
  def get_tablename(cls) -> Text:
    return cls.__tablename__

  def get_list_text(self) -> Text:
    return f'{self.get_tablename()}'

  def get_object_text(self) -> Text:
    return f'\n{self.get_tablename()}'
```

Ukázka kódu 5.1: Třída KnowledgeBaseModel

5.2 Controller

Controllery jsou využívány k práci se *Services*. Pomocí nich získají a upravují data do vhodného formátu, který požaduje daná *View* nebo *Action*. Jedná se o třídy, které si v inicializaci vezmou jako atribut aktuální stav uložený v objektu typu *Tracker*. Pomocí něj pak získávají potřebné informace z extrahovaných entit a slotů.

5.2.1 KnowledgeBaseController

Tento controller slouží k získávání a propojování informací z aktuálně položeného dotazu a dotazu předešlého.

Kód na obrázku 5.2 je ukázkou, jak chatbot získává poslední vypsaný objekt. Pomocí trackeru se pokusí ze slotů extrahovat jeho *type* a *id*. Když tyto údaje existují, použije je databázová *PostgresService* k získání záznamu z tabulky se jménem *type*. Záznam se v případě existence namapuje na přiřazený model, který je podtypem *KnowledgeBaseModel*. Téměř totožně funguje získávání posledních vypsaných objektů. Identifikátor *id* je nahrazen seznamem identifikátorů a výsledkem úspěšného hledání je seznam položek místo jednoho modelu.

Další skupinu tvoří metody, které získávají objekt podle zmíněného jména nebo číslovky z posledního vypsaného seznamu. Řeší problém odkazováním se na položky, což je jednou z klíčových funkcí chatbota. V kódu 5.3 je metoda používající k extrakci jméno. Na začátku se z trackeru získají všechna extra-

hovaná jména. Pokud je nějaké jméno nalezeno, odstraní se z něho úvozovky, které ho označovaly, a zažádá se o poslední vypsaný list objektů. Tyto parametry jsou odevzdány databázové *PostgresService*, která objekty ze seznamu vyfiltruje podle jména. Pokud se v posledním listu nenachází žádné požadované položky, vyzkouší se vyhledávání i v záznamech tabulek *Pilar*, *Product* a *ProductModule*. Odkazování pomocí číslovek funguje velice podobně. Číslovka se převede na index v poli posledních objektů a pokud tento objekt existuje, vrátí se jako výsledek.

Důležitým prvkem pro správnou funkčnost je i metoda z kódu 5.4, která extrahuje typ entity zájmu – typ tabulky, o kterou má uživatel zájem. Pokud se v metodě nachází parametr *required_object* je vrácen jeho typ. Pokud ne, metoda pokračuje extrakcí entit z trackeru, které následně ověří vůči dostupným typům tabulek. Když je i po ověření sesbírán více než jeden typ, vrací se nejvíc zanořený z nich. Tento postup vychází z formy pokládaných otázek, kde se často vyskytuje další typ tabulky asociován se jménem, na které se odkazuje, například „Ukaž mi modul patřící pod produkt X“.

```
def get_last_object(self) -> Optional[KnowledgeBaseModel]:
  object_id = self.tracker.get_slot(KnowledgeBase.General.LAST_OBJECT_ID)
  object_type = self.tracker.get_slot(KnowledgeBase.General.LAST_OBJECT_TYPE)
  if object_id is None or object_type is None:
    return None

  try:
    return postgresService.get_object(object_type, object_id)
  except ValueError:
    return None
```

Ukázka kódu 5.2: Získání posledního vypsaného objektu

```
def get_name_objects(self, object_type: Optional[Text] = None):
  object_name_cons = KnowledgeBase.General.OBJECT_NAME
  name_values = list(self.tracker.get_latest_entity_values(object_name_cons))
  if not name_values:
    return []

  object_name = name_values[0].replace('"', '')
  last_list = self.get_last_list()
  objects = postgresService.find_name_objects_in_list(object_name, last_list)

  return objects or postgresService.find_name_objects(object_name,
                                                      object_type)
```

Ukázka kódu 5.3: Získání objektů podle jména

```
def get_object_type(self,
                    required_object: Optional[KnowledgeBaseModel] = None):
  if required_object is not None:
    return required_object.get_tablename()

  object_type_cons = KnowledgeBase.General.OBJECT_TYPE
  all_objects = KnowledgeBase.Object.all_objects

  type_values = list(self.tracker.get_latest_entity_values(object_type_cons))
  object_types = [entity_value for entity_value
                               in object_type_entity_values
                               if entity_value in all_objects]

  if len(object_types) == 1:
    return object_types[0]
  elif len(object_types):
    object_type_index = KnowledgeBase.Object.all_objects.index(object_type)
    indexes = [object_type_index for object_type in object_types]
    return KnowledgeBase.Object.all_objects[max(indexes)]
```

Ukázka kódu 5.4: Získání typu entity zájmu

5.2.2 UserController

Tato třída aktuálně slouží jen pro zajištění interního nebo externího statusu uživatele chatbota, jak naznačuje kód 5.5. Využívá *TeamsService* pro získání e-mailu uživatele, který slouží jako identifikátor daného stavu. Aby nedocházelo k plýtvání žádostmi na Teams API, e-maily uživatelů jsou ukládány spolu identifikátory do *RedisCache*, do které controller přistupuje pomocí *RedisService*. Doména získaného e-mailu se v závěru porovná s doménou, kterou disponují jen interní zaměstnanci. Výsledek srovnání se vrací jako status uživatele.

V budoucím vývoji bude možno tento controller rozšířit o další uživatelské metody, které mohou přispět k vyšší personalizaci i lepšímu uživatelskému zážitku. Vhodným příkladem jsou proaktivní zprávy, které Teams jako platforma nabízí. Jedná se o typ zpráv, které chatbot může posílat jako reakci na určitý typ uživatelské interakce. Tyto reakční zprávy mohou zvýšit komfort a celkový dojem konverzace s chatbotem.

5.3 Service

Skupina těchto tříd slouží k získávání dat z určitého datového zdroje. Jak již bylo zmíněno, *RedisService* čerpá informace z lokálního úložiště *RedisCache*. U *TeamsService* je zdrojem Teams API, které poskytuje informace na základě requestů. Jelikož mají tyto services v práci marginální funkčnost, tato sekce bude věnována databázové *PostgresService* a jejím metodám.

```
async def is_user_internal(self) -> Optional[bool]:
  try:
    user_id = self.tracker.sender_id
    email = await redis_cache.hash_get('users', user_id)
    if not email:
      metadata = self.tracker.get_last_event_for('user')['metadata']
      conversation_url = metadata['conversation_url']
      conversation_id = metadata['conversation_id']

      email = await teams_api_service.get_user_email(conversation_url,
                                                     conversation_id,
                                                     user_id)
      if email is None:
        Exception('Can not obtain user email')
      await redis_cache.hash_save('users', user_id, email)

      email_domain = email.split("@")[-1]
      return email_domain == Domains.INTERNAL_EMAIL
  except Exception as error:
      logger.error(f'Exception "{repr(error)}" during obtaining'
       'user internal/external status.')
```

Ukázka kódu 5.5: Získání statusu uživatele

5.3.1 PostgresService

Při inicializaci této třídy 5.6 je pomocí URL knihovně SQLAlchemy předána informace o databázi, jménu, heslu a též adrese. Knihovna tyto parametry použije při sestavovaní *session*, která se následně využívá pro kompozici *query* dotazů. Pomocí nich lze získávat z databáze záznamy příslušných tabulek.

```
class PostgresService(object):

  def __init__(self):
    url = f'postgresql+psycopg2://{postgres_username}:{postgres_password}@' +
          f'{postgres_host}/bachelor'
    self.engine = create_engine(url)
    session = sessionmaker(bind=self.engine)
    self.session = session()
```

Ukázka kódu 5.6: Inicializace PostgresService

Mezi nejpoužívanější metody patří trojice v kódu 5.7. První dvě plní funkci obalu nad *query* dotazy, kde doplňují místo jména poptávané tabulky konkrétní třídu, na kterou se mají výsledky mapovat. Metoda *get_relation_list* získává záznamy, které jsou s danou tabulkou provázány pomocí „one-to-many“ vztahu. Tyto namapované objekty dále dle podmínek v parametru *filters* profiltruje a výsledek použije jako návratovou hodnotu metody.

```
def get_object(self,
               tablename: Text,
               object_id: int) -> Optional[KnowledgeBaseModel]:
  model_class = self._class_by_table_name(tablename)
  return self.session.query(model_class).get(object_id)

def get_list(self, tablename: Text):
  model_class = self._class_by_table_name(tablename)
  return self.session.query(model_class).all()

def get_relation_list(self,
                      tablename: Text,
                      object_id: int,
                      relation_attribute: Text,
                      filters: List[Text] = None):
  model_class = self.get_object(tablename, object_id)
  objects = getattr(model_class, f'{relation_attribute}_relationship', [])

  relation_model_class = self._class_by_table_name(relation_attribute)
  objects = self._filter_list(filters, relation_model_class, objects)
  return objects
```

Ukázka kódu 5.7: Metody využívající query

5.4 View

Jak již bylo v kapitole návrhu naznačeno, *view* neplní v případě chatbota standardní roli, která je známá z webových stránek nebo aplikací. Plní funkci souboru grafických prvků (konkretně textů a tlačítek), které chatbot odesílá uživateli jako odpověď. Jelikož je logika vytváření textů a tlačítek poměrně přímočará, generování všech zmiňovaných prvků shrnuje třída *ResponseCreator*. Ta při inicializaci dostane v parametru referenci na aktuální dispatcher, pomocí které následně zasílá vygenerované *view* pro konkrétní situaci. Metody této třídy se dle funkcionality dělí na dvě skupiny. Jedna zmiňované grafické prvky připravuje a druhá je odesílá. Oba typy metod se nachází v kódu 5.8. Jak název *_recommendation_buttons* napovídá, tato metoda vytváří tlačítka usnadňující pokládaní navazujících dotazů. Tato tlačítka se využívají u druhé metody *show_detail*, která uživateli odesílá detail poptávaného objektu. Detail objektu a doporučovací tlačítka doprovází i text vybízející k položení navazujícího dotazu.

5.5 FormAction

FormAction je speciální třída poskytovaná frameworkem Rasa, která modifikuje klasickou *Action* na situace, kde je pro obdržení výsledku nutno zaslat více parametrů. Získávání těchto parametrů probíhá za definovaných pravi-

```
def _recommendation_buttons(object_for_recommendation: KnowledgeBaseModel):
  buttons = []
  for attribute in object_for_recommendation.attributes:
    buttons.append({
      'title': Localization.objects[attribute].capitalize(),
      'payload': f'/listing{{"object_type":"{attribute}"}}'
    })
  buttons.append(ResponseCreator._restart_button())
  return buttons

def show_detail(self, referenced_object: KnowledgeBaseModel):
  detail_text = referenced_object.get_object_text()
  object_type = referenced_object.get_tablename()
  object_name = referenced_object.get_list_text()
  buttons = self._recommendation_buttons(referenced_object)

  self.dispatcher.utter_message(template='utter_object_detail',
                                detail=detail_text)
  if len(buttons) == 1:
    self.dispatcher.utter_message(buttons=buttons)
  else:
    self.dispatcher.utter_message(template='utter_recommended_buttons_text',
                                  object_type=Localization.objects[object_type],
                                  object_name=object_name,
                                  buttons=buttons)
```

Ukázka kódu 5.8: Metody pro přípravu a odeslání textů/tlačítek

del z textů, intentů nebo entit. Extrahované části je možné dále validovat a následně použít při získávání výsledku.

Tato třída byla vhodným začátkem pro implementaci generické logiky, která se bude starat o získávání všech potřebných prvků pro sestavování *query* dotazů. Jelikož jsou známé dva obecné scénáře zjišťování informací, byly vytvořeny třídy *ListingForm* a *DetailForm*. Jak již názvy naznačují, první z nich zajišťuje zasílání seznamu určitých položek. *DetailForm* slouží k výpisu podrobností u diskutované položky. Exekuce obou akcí probíhá až na pár odlišností velice podobně. Při inicializace jsou vytvořeny atributy, do kterých se ze slotů a entit pomocí zmiňovaného *KnowledgeBaseController* přiřadí všechny známé položky. Pokud některá z položek chybí a je klíčová pro exekuci, daná akce se ukončí spolu se zprávou o chybě pro uživatele. V kladném případě se následně přistoupí ke vygenerování seznamů prvků, které je potřeba naplnit pro sestavení *query*. Tyto prvky se od uživatele sbírají pomocí doplňujících dotazů, které chatbot pokládá. Po sesbíraní všech dat chatbot sestaví databázový dotaz a výsledek odešle uživateli ve formě textové zprávy. Rozhraní *FormAction* dovoluje řešit různé nepředvídatelné situace, například odbočování od tématu během vyplňování potřebných prvků. I tato situace je v práci vyřešena zprávou, která uživatele informuje o návratu ke sběru dat.

5.6 Teams konektor

Konektor pro MS Teams byl jeden z klíčových požadavků práce. Vytvořený konektor je postaven na základě tříd, které pro jednotlivé chatovací platformy poskytuje Rasa. Pro vstupní zprávy byla vytvořena třída *TeamsInputChannel*, která své vlastnosti dědí od *InputChannel*. Odcházející zprávy spravuje třída *TeamsOutputChannel*, která je potomkem *OutputChannel*.

5.6.1 Přijímaní zpráv

Pro přijímaní zpráv bylo potřeba vystavit *webhook*, který akceptuje POST i GET metody. Chatbot musí těmto zprávám validovat jwt token, který se nachází v hlavičce requestu. Validace probíhá v 4 krocích:

1. Získání JWT tokenu z hlavičky
2. Získání OpenID dokumentu
3. Získání jwks_uri
4. Verifikace jwt tokenu [47]

```
async def _verify_jwt_token(self, headers: Dict, service_url: Text) -> bool:
  ...
  public_keys = {}
  for jwk in jwks:
    kid = jwk['kid']
    public_keys[kid] = jwt.algorithms.RSAAlgorithm.from_jwk(json.dumps(jwk))
  kid = jwt.get_unverified_header(token)['kid']
  key = public_keys[kid]
  payload = jwt.decode(token, key=key, algorithms=algorithm,
                       audience=self.microsoft_app_id, issuer=issuer)
  return payload['serviceurl'] == service_url
  ...
```

Ukázka kódu 5.9: Verifikace JWT tokenu

Z implementačního hlediska byla čtvrtá část (znázorněna v kódu 5.9) náročnější pro pochopení. Veřejný klíč, kterým se na konci dekódoval jwt token byl vytvořen pomocí RSA algoritmu. Při dekódování se porovnaly všechny potřebné části, které obsahovala původní hlavička. Když srovnání dopadlo kladně, výsledek verifikace byl validní.

Mimo verifikace se při přijetí zprávy ověřuje formát a obsah těla requestu. Extrahované informace se jako argumenty posílají do tříd, které spravují odesílaní odpovědí, jak je vidět v kódu 5.10. Navíc se tady ověřuje i typ zprávy. Pro účely chatbota jsou rozlíšeny dva typy zpráv: klasická textová a aktualizace konverzace. Aktualizace konverzace se váže s prvním začátkem chatu

s chatbotem. Při této příležitosti je uživateli nutno poslat úvodní zprávu. To zajišťuje řádek 6 v kód 5.10, který do textu zprávy přidává vyvolání intentu pro start konverzace.

```
1  @teams_webhook.route("/webhook", methods=["POST", 'GET'])
2  async def webhook(request: Request) -> HTTPResponse:
3    ...
4    action_type = json_data['type']
5    if action_type == 'conversationUpdate' and json_data.get('membersAdded'):
6      json_data['text'] = f'/{self.starting_intent}'
7    elif action_type != 'message':
8      raise Exception(f'Action type "{action_type}" was '
9                      f'received instead of "message".')
10
11   conversation = json_data['conversation']
12   recipient = json_data['recipient']
13   sender_id = json_data['from']['id']
14   message_text = json_data['text']
15   metadata = {
16     'conversation_url': TeamsOutputChannel.create_base_conversation_url(url),
17     'conversation_id': conversation['id']
18   }
19
20   teams_output_channel = TeamsOutputChannel(self.microsoft_app_id,
21                                             self.microsoft_app_password,
22                                             recipient,
23                                             conversation,
24                                             url)
25   await teams_output_channel.send_typing(sender_id)
26   message = UserMessage(message_text,
27                         teams_output_channel,
28                         sender_id,
29                         input_channel=self.name(),
30                         metadata=metadata)
31   await on_new_message(message)
32   ...
```

Ukázka kódu 5.10: Příjímání zprávy od uživatele

5.6.2 Odesílaní zpráv

Každá odeslaná zpráva musí v hlavičce obsahovat validní autorizační token. Ten je možno získat po odeslání žádosti na API. Její tělo je potřebné vyplnit zejména údaji, které vývojář získá pomocí Azure portálu – konkrétně *app_id* a *app_password*. Po úspěšném získaní odpovědi je token spolu s údajem o jeho expiraci uložen pro další použití, jak ukazuje kód 5.11.

Odesílání zpráv pak probíhá velice přímočaře. Pomocí předpřipravených metod, které jsou uvnitř třídy přepsány na vlastní implementaci 5.12, jsou do požadovaných formátů uspořádány tlačítka i texty. Uvnitř textů je nahra-

zen klasický symbol pro novou řádku nahrazen jiným, který dokáže Teams správně zobrazit. Na závěr se pomocí metody *create_message* přidají do těla zprávy důležité údaje pro identifikaci konverzace, přijímatele a odesílatele. Tata zpráva se posléze odešle metodou *send* jako POST request na API.

```
ync def get_token(self) -> Optional[Text]:
if (TeamsOutputChannel.access_token and
    TeamsOutputChannel.expires > time.time()):
  return TeamsOutputChannel.access_token
TeamsOutputChannel.access_token = None

url = 'https://login.microsoftonline.com/botframework.com/oauth2/v2.0/token'
headers = {'Content-Type': 'application/x-www-form-urlencoded'}
body = {
  "grant_type": 'client_credentials',
  "client_id": self.app_id,
  "client_secret": self.app_password,
  "scope": 'https://api.botframework.com/.default'
}

async with ClientSession(headers=headers) as token_session:
  async with token_session.post(url, data=body) as resp:
    try:
      token_data = await resp.json()
      resp.raise_for_status()
      TeamsOutputChannel.access_token = token_data['access_token']
      TeamsOutputChannel.expires = token_data["expires_in"] + time.time()
...
```

Ukázka kódu 5.11: Žádost pro získání tokenu

```
async def send_text_message(self,
                            recipient_id: Text,
                            text: Text,
                            **kwargs: Any) -> None:
  for message_part in text.split("\n\n"):
    text_message = {'text': message_part.replace('\n', '<br/>')}
    message = self.create_message(recipient_id, text_message)
    await self.send(message)

async def send_text_with_buttons(self,
                                 recipient_id: Text,
                                 text: Text,
                                 buttons: List[Dict[Text, Any]],
                                 **kwargs: Any) -> None:
  try:
    teams_buttons = [{
      'type': 'messageBack',
      'title': button['title'],
      "displayText": button['title'],
      'text': button['payload'],
    } for button in buttons]
  except KeyError as err:
    logger.error(f'Buttons do not have required fields: {repr(err)}')
    return

  buttons_message = {
    'attachments': [{
      'contentType': 'application/vnd.microsoft.card.hero',
      'content': {
        'text': text,
        'buttons': teams_buttons
      }
    }]
  }
  message = self.create_message(recipient_id, buttons_message)
  await self.send(message)
```

Ukázka kódu 5.12: Odesílání různých typů zpráv

Kapitola 6

Testování

Kapitola testování se věnuje testům, které ověřují jednotlivé konverzační scénáře s chatbotem. Tyto testy napomáhají udržovat funkčnost chatbota během různých zásahů do business logiky v kódu. Jsou sestaveny tak, aby jejich průchodnost ověřovala nutný základ, na který chatbot musí umět odpovědět.

6.1 Technologie

K testování je používán framework *pytest*, který je určen k sestavování menších testů, které je možné škálovat i do komplexního testování. Výhodou jsou přehledné reporty a statistiky o průběhu, a též podpora více než 300 externích pluginů. [48] Tyto pluginy rozšiřují možnosti testování, například o asynchronicitu nebo mocking. Mocking je možno přeložit jako „falšování“ ve smyslu nahrazování určitých proměnných nebo návratových hodnot funkcí, za jiné hodnoty. Tato vlastnost ulehčuje testování funkcí, které se za běžných okolností dotazují na externí API. Jejich výsledky jsou jednoduše nahrazeny. Právě tyto možnosti budou v testech často využívány.

6.2 Testy

Testy jsou rozděleny do tří tříd (kategorií) podle testovaných akcí. Patří sem *TestListingAction*, *TestDetailAction* a *TestAttributeAction*. Každá z těchto tříd obsahuje metody, které pokrývají základní scénáře dané akce. Aby bylo možno tyto scénáře otestovat, bylo potřeba udělat mock statusu uživatele. O tuto činnost se stará funkce 6.1, která se spouští před každým testem.

Ze všech testovacích metod je nutno zmínit dva přístupy k sestavování scénářů. První přístup zobrazen v kódu 6.2 je velice přímočarý. Dotaz obsahuje všechny potřebné položky pro odpověď. Jak je v kódu vidět, akce je potřeba nasimulovat ve formě rozpoznaného intentu i extrahované entity. Takto předpřipravená zpráva se odešle do trackeru a zažádá se o vykonání správné akce.

Po vykonání se porovná počet zpráv i přítomnost konkrétní šablony, která se používá pro výpis informací.

Druhý přístup, zobrazen v kódu 6.3, je sestaven ze dvou akcí. Nejprve je potřeba pilíře vypsat jako v kódu 6.2. Návratovou hodnotu v podobě *events*, kterou akce vrátí, se zkonvertuje na sloty. Ty se použijí při volání další akce, kvůli simulaci běžného chodu chatbota. Výsledek, v podobě zpráv se ověří stejně jako u prvního typu testu. Přibyla jen šablona, která obaluje odesílání tlačítek.

```
ytest.fixture(autouse=True)
f run_around_tests(mocker):
  # Called before each test
  mocker.patch(
      "bachelor_action_server.actions.UserController.is_user_internal",
      side_effect=[async_wrapper(True),
                   async_wrapper(True),
                   async_wrapper(True)],
  )
  yield
  # Called after each test
```

Ukázka kódu 6.1: Mock statusu uživatele

```
async def test_plain_listing(self):
  latest_message = {
      'intent': create_intent('listing'),
      'entities': [create_entity('object_type', 'pillar')],
      'text': 'Vylistuj mi seznam pilířů'
  }

  events, messages = await execute_action(ListingForm(), latest_message)
  assert len(messages) == 1
  assert messages[0].get("template") == 'utter_list_of_pillar'
```

Ukázka kódu 6.2: Testování dotazu pro seznam pilířů

```
ytest.mark.asyncio
ync def test_mention_detail(self):
latest_message = {
    'intent': create_intent('listing'),
    'entities': [create_entity('object_type', 'pillar')],
    'text': 'Vylistuj mi seznam pilířů'
}

events, _ = await execute_action(ListingForm(), latest_message)

latest_message = {
    'intent': create_intent('detail'),
    'entities': [create_entity('mention_index', '2'),
                 create_entity('object_type', 'pillar')],
    'text': 'Ukaž mi detail pilíře číslo 2'
}

slots = slots_from_events(events)
slots['mention_index'] = '2'

events, messages = await execute_action(DetailListingForm(),
                                        latest_message,
                                        slots)
assert len(messages) == 2
assert messages[0].get("template") == 'utter_object_detail'
assert messages[1].get("template") == 'utter_recommended_buttons_text'
```

Ukázka kódu 6.3: Testování dotazu na detail pilíře

Závěr

Cílem této práce bylo navrhnout a implementovat chatbota pro zpracování obsáhlého produktového portfolia společnosti ANECT a.s. Bylo potřeba analyzovat funkční a nefunkční požadavky, na kterých návrh a architektura vznikala. Aby chatbota mohl sales tým používat, bylo součástí zadání i vytvoření konektoru pro prostředí MS Teams, prostřednictvím kterého komunikace s chatbotem probíhá.

Všechny primární i dílčí cíle se v práci podařilo splnit. Byl kladen důraz na implementaci samotného řešení, konkrétně na různé typy dotazů. Vypořádal jsem se se všemi typy odkazování na položky i dotazy, které mají složitou jazykovou strukturu obsahující vícero informací. Tyto případy se postupně objevovaly až během několika iterací implementace, co mělo za důsledek velké změny v samotné logice fungování.

Je ale nutno podotknout, že způsob, kterým uživatel klade dotazy je velice subjektivní záležitost. Jazykový model, který chatbot používá je vytvořen na základě setu otázek, které jsem během testování vytvořil. Pokud s nimi uživatel není obeznámen, může nastat situace, kdy chatbot daný dotaz nerozpozná. Pozitivem do budoucnosti je ale fakt, že různé typy dotazů se mohou do jazykového modelu jednoduše přidat, co zlepší schopnost porozumění.

Jsem přesvědčen, že produkční nasazení chatbota zlepší aktuální přístup k informacím z portfolia. Ostrý provoz také ukáže, které dotazy jsou běžnou praxí při komunikaci o prodeji. Prostřednictvím administračního rozhraní, kterého základ vytvářel ve své práci kolega Jan Šafařík, budeme tyto dotazy analyzovat a zlepšovat tak stávající řešení. Po úspěšném ověření konceptu plánujeme na práci navázat i v jiných segmentech než je produktové portfolio. Chatbot by měl jako platforma sloužit pro jakoukoliv znalostí bázi.

Literatura

[1] Schlicht, M.: The Complete Beginner's Guide To Chatbots. [online], 2016, [cit. 2019-12-10]. Dostupné z: https://chatbotsmagazine.com/the-complete-beginner-s-guide-to-chatbots-8280b7b906ca

[2] Ștefania Budulan: Chatbot Categories and Their Limitations. [online], 2018, [cit. 2019-12-10]. Dostupné z: https://dzone.com/articles/chatbots-categories-and-their-limitations-1

[3] Rawes, E.; Wetzel, K.: What exactly is Alexa? Where does she come from? How does she work? [online], 2019, [cit. 2019-12-10]. Dostupné z: https://www.digitaltrends.com/home/what-is-amazons-alexa-and-what-can-it-do/

[4] Top 20 Chatbot Usecases / Applications in Business in 2019. [online], 2019, [cit. 2019-12-10]. Dostupné z: https://blog.aimultiple.com/business-chatbot/

[5] What's a knowledge base and why you need it. [online], [cit. 2019-12-11]. Dostupné z: https://www.atlassian.com/it-unplugged/knowledge-management/what-is-a-knowledge-base

[6] Wade, M.: 5 ways chatbots are revolutionizing knowledge management. [online], 2018, [cit. 2019-12-11]. Dostupné z: https://blog.getbizzy.io/5-ways-chatbots-are-revolutionizing-knowledge-management-bdf925db66e9

[7] Verspoor, K.; Cohen, K.: *Natural Language Processing*. 01 2013, ISBN 978-1-4419-9862-0, s. 1495–1498, doi:10.1007/978-1-4419-9863-7_158.

[8] NLP vs. NLU: What's the Difference and Why Does it Matter? [online], 2019, [cit. 2019-12-10]. Dostupné z: https://blog.rasa.com/nlp-vs-nlu-whats-the-difference/

[9] Liyanapathirana, L.: NLP Chronicles: Introduction to Natural Language Processing with NLTK. [online], 2018, [cit. 2019-12-10]. Dostupné z: https://heartbeat.fritz.ai/nlp-chronicles-intro-to-nlp-with-nltk-b2c369fbb9a7

[10] *Introduction to information retrieval*. New York: Cambridge University Press, 2009 vydání, 2008, ISBN 05-218-6571-9.

[11] Tokenization. [online], [cit. 2020-02-27]. Dostupné z: https://spacy.io/tokenization-57e618bd79d933c4ccd308b5739062d6.svg

[12] Stemming and lemmatization. [online], [cit. 2020-02-27]. Dostupné z: https://qph.fs.quoracdn.net/main-qimg-cd7f4bafaa42639deb999b1580bea69f

[13] Python for NLP: Parts of Speech Tagging and Named Entity Recognition. [online], 2019, [cit. 2020-01-29]. Dostupné z: https://www.elasticfeed.com/808954966b535fbd550bccc7b565bec9/

[14] POS Tagging. [online], [cit. 2020-02-27]. Dostupné z: https://www.elasticfeed.com/wp-content/uploads/1ee8b35289a283e15b04562ca721defe.jpg

[15] D'Souza, J.: Learning POS Tagging & Chunking in NLP. [online], [cit. 2020-01-29]. Dostupné z: https://medium.com/greyatom/learning-pos-tagging-chunking-in-nlp-85f7f811a8cb

[16] Janik, M.: Stop slova. [online], 2009, [cit. 2020-01-29]. Dostupné z: https://www.michaljanik.cz/oblibene/stop-slova

[17] Intents in your LUIS app. [online], 2019, [cit. 2019-12-15]. Dostupné z: https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-concept-intent

[18] Entities and their purpose in LUIS. [online], 2019, [cit. 2019-12-15]. Dostupné z: https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-concept-entity-types

[19] NLP. [online], [cit. 2020-02-28]. Dostupné z: https://www.aismartz.com/blog/wp-content/uploads/2019/09/differences-between-NLP-NLG-NLU.jpg

[20] A Comprehensive Guide to Natural Language Generation. [online], 2019, [cit. 2020-06-01]. Dostupné z: https://medium.com/sciforce/a-comprehensive-guide-to-natural-language-generation-dd63a4b6e548

[21] Giulia: Wit.ai and how to use it. [online], 2017, [cit. 2019-12-12]. Dostupné z: https://medium.com/@Giuul/wit-ai-and-how-to-use-it-72372b07d98b

[22] Team, T. W.: Supporting 100,000 developers and beyond…. [online], 2017, [cit. 2019-12-12]. Dostupné z: https://medium.com/wit-ai/supporting-100-000-developers-and-beyond-e6e46751ad31

[23] Frequently Asked Questions. [online], [cit. 2019-12-12]. Dostupné z: https://wit.ai/faq

[24] What is Language Understanding (LUIS)? [online], 2019, [cit. 2019-12-14]. Dostupné z: https://docs.microsoft.com/en-us/azure/cognitive-services/luis/what-is-luis

[25] Dialogflow. [online], [cit. 2019-12-14]. Dostupné z: https://cloud.google.com/dialogflow/

[26] Introduction. [online], [cit. 2019-12-16]. Dostupné z: https://botpress.io/docs/introduction

[27] Getting Started with Rasa. [online], [cit. 2019-12-16]. Dostupné z: https://rasa.com/docs/getting-started/

[28] Building Bots with the Rasa Framework. [online], 2019, [cit. 2019-12-16]. Dostupné z: https://discover.bot/bot-talk/guide-to-bot-frameworks/rasa/

[29] Rasa. [online], 2019, [cit. 2020-02-03]. Dostupné z: https://rasa.com/

[30] Why Rasa. [online], 2019, [cit. 2020-02-03]. Dostupné z: https://rasa.com/product/why-rasa/

[31] Choosing a Pipeline. [online], 2020, [cit. 2020-02-03]. Dostupné z: https://rasa.com/docs/rasa/nlu/choosing-a-pipeline/

[32] Training Data Format. [online], [cit. 2020-02-27]. Dostupné z: https://rasa.com/docs/rasa/nlu/training-data-format/

[33] Stories. [online], [cit. 2020-02-27]. Dostupné z: https://rasa.com/docs/rasa/core/stories/

[34] Rouse, M.: Definition use case. [online], 2007, [cit. 2020-02-03]. Dostupné z: https://searchsoftwarequality.techtarget.com/definition/use-case

[35] Odhiambo, D.: System Design in Software Development. [online], [cit. 2020-02-06]. Dostupné z: https://medium.com/the-andela-way/system-design-in-software-development-f360ce6fcbb9

[36] Software Architecture & Design Introduction. [online], 2020, [cit. 2020-05-14]. Dostupné z: https://www.tutorialspoint.com/software_architecture_design/introduction.htm

[37] Architecture. [online], 20202, [cit. 2020-05-14]. Dostupné z: https://rasa.com/docs/rasa/user-guide/architecture/

[38] Actions. [online], 2020, [cit. 2020-05-14]. Dostupné z: https://rasa.com/docs/rasa/core/actions/

[39] Action Server. [online], 2020, [cit. 2020-05-14]. Dostupné z: https://rasa.com/docs/rasa/api/action-server/#action-server

[40] Rasa SDK. [online], 2020, [cit. 2020-05-14]. Dostupné z: https://rasa.com/docs/rasa/api/rasa-sdk/

[41] SQLAlchemy. [online], 2020, [cit. 2020-05-14]. Dostupné z: https://www.sqlalchemy.org/

[42] Modelling and the UML. [online], 2020, [cit. 2020-05-15]. Dostupné z: https://www.open.edu/openlearn/science-maths-technology/introduction-software-development/content-section-6

[43] Relational Database Model. [online], 2020, [cit. 2020-05-15]. Dostupné z: https://databasemanagement.fandom.com/wiki/Relational_Database_Model

[44] Conversation basics. [online], 2020, [cit. 2020-03-05]. Dostupné z: https://docs.microsoft.com/en-us/microsoftteams/platform/bots/how-to/conversations/conversation-basics?view=msteams-client-js-latest&tabs=dotnet

[45] Format your bot messages. [online], 2020, [cit. 2020-05-16]. Dostupné z: https://docs.microsoft.com/en-us/microsoftteams/platform/bots/how-to/format-your-bot-messages

[46] Cards Reference. [online], 2020, [cit. 2020-03-05]. Dostupné z: https://docs.microsoft.com/en-us/microsoftteams/platform/task-modules-and-cards/cards/cards-reference

[47] Authenticate requests from the Bot Connector service to your bot. [online], 2020, [cit. 2020-05-22]. Dostupné z: https://docs.microsoft.com/en-us/azure/bot-service/rest-api/bot-framework-rest-connector-authentication?view=azure-bot-service-4.0

[48] Krekel, H.: Pytest: helps you write better programs. [online], 2020, [cit. 2020-05-21]. Dostupné z: https://docs.pytest.org/en/latest/

Příloha A

Uživatelský manuál

A.1 Podporované dotazy

Tato témata dokáže chatbot rozpoznat během konverzace. Ke každému z nich bylo přiřazeno několik ukázkových dotazů, pomocí kterých byl vytrénován použitý jazykový model. Některé z nich je možno využít jen ve správném kontextu konverzace – tato skutečnost bude znázorněna trojtečkou.

Vylistování položek

- Které produkty ANECT nabízí?
- Jaké produktové moduly patří pod produkt "TIDA"?
- Zajímají mě české technologie pro model "IPS"

Detail položky

- Řekni mi něco o modulu "Virtualization"
- ...Rád bych viděl detail jedničky.
- ...Ukaž mi detail druhého produktu

Označování položek během doplňkových otázek

- ...Dvojka
- ...Jde o 1
- ...Ten třetí

Zdvořilostní fráze a reakce

- Dobrý den
- Skvělé
- Škoda
- Mockrát děkuji

- Měj se hezky

Možnosti konverzace

- Na co všechno mi umíš odpovědět?
- Které dotazy zvládneš?
- Jak se tě mám správně ptát?

Lidé

- Na koho se mohu obrátit kvůli "Audits"?
- ...Kdo se tomu věnuje v oddělení consulting?
- ...Kteří lidé pracují na dvojce?

Kontakt

- Jaký email má pan "Petr"?
- Jak mohu kontaktovat zaměstnance "Jiří"?
- ...Jak mohu napsat prvnímu člověku?

Distribuce

- Může modul "Run" běžet i v cloudu?
- Jak je možné poskytovat "Load Balancing"?
- Na jaké distribuci běží "Fault Monitoring"?

Zkušenosti

- Máme nějakou zkušenost s "Aducid"?
- ...Kolik zkušeností s tím máme?
- Jakou zkušenost máme s modulem "Aducid"?

Vlastníci

- Kdo produkt "TIDA" zastřešuje?
- Kdo je vlastníkem "SOCA"?
- Kdo se stará o produkt "Collab"?

Konkrétní člověk

- Pracuje u nás i pan "David"?
- Na kterém oddělení pracuje "Libor"?
- Kdo je to "Adam"?

Kapacita

- ...Kolik lidí zde pracuje v consultingu?

- ...Jak velký je tým u jedničky?
- Jak jsme na tom s kapacitou u "Aducid"?

Restart konverzace (zahození kontextu)

- Restart
- Tak znova

A.2 Jak se správně ptát

Dotazy na určitá témata musí respektovat postupné zanořování. Je známo pět stupňů zanoření:

1. Pilíř
2. Produkt
3. Produktový modul
4. Technologie
5. Lidé/týmy

Pokud uživatel sestavuje dotaz, musí dbát na zanořování směrem dolů – chatbot umí vyhledat technologie pro určitý produkt, ale neumí dohledat produkt pro určité technologie. Toto omezení vzniká na základě struktury portfolia. Další omezení se váže k otázkám, které uživatel vytváří bez předešlého kontextu. V tomto případě je možné položit dotaz, který v sobě bude obsahovat název pilíře, produktu, modulu nebo pracovníka. Není možnost se bez kontextu odkázat na název technologie, protože jejich názvy nejsou unikátní.

A.3 Modelová konverzace

Uživatel: Dobrý den

Chatbot: Hezký den přeji

Uživatel: Které produkty ANECT nabízí?

Chatbot: Pro upřesnění bych rád věděl, o kterou položku typu pilíř jde? + {*seznam položek*}

Uživatel: Ten třetí

Chatbot: Produkt: Process Security + {*nabídka tlačítek*}

Uživatel: {*tlačítko Produktový modul*}

Chatbot: Pod Process Security patří tyto prodávané moduly: + {*seznam položek*}

Uživatel: Máme nějakou zkušenost s "Audits"?

Chatbot: {*detail technologie*}

Uživatel: Kdo se tomu věnuje v oddělení Consulting?

Chatbot: Tito lidé (týmy) pracují na technologii...+ {*seznam položek*}

Uživatel: Kdo je to "{*jméno*}"?

Chatbot: {*detail pracovníka s vybraným jménem*}

Uživatel: Mockrát děkuji

Chatbot: Rádo se stalo

Uživatel: Měj se hezky

Chatbot: Mějte se hezky. Rád jsem si s vámi promluvil.

Příloha B

Seznam použitých zkratek

NLP Natural language processing

POS Part-Of-Speech

IR Intent recognition

NER Named entity recognition

NLU Natural language understanding

NLG Natural language generation

API Application programming interface

SDK Software development kit

MVC Model-View-Controller

ORM Object-relational mapping

URL Uniform Resource Locator

MS Microsoft

Příloha C

Obsah přiložené SD karty

- `readme.txt` ..... stručný popis obsahu SD karty
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Stanovcak_Tomas_2020.pdf` ..... text práce ve formátu PDF