VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

MOBILNÍ APLIKACE PRO VYHLEDÁVÁNÍ ZAJÍMAVÝCH AKCÍ A DESTINACÍ V BRNĚ

MOBILE APPLICATION FOR SEARCHING FOR INTERESTING EVENTS AND DESTINATIONS IN BRNO

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ONDŘEJ MISAŘ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ DYK,**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Misař Ondřej**
Program: Informační technologie
Název: **Mobilní aplikace pro vyhledávání zajímavých akcí a destinací v Brně**
Mobile Application for Searching for Interesting Events and Destinations in Brno
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s aktuálním stavem vývoje chatbotů a příslušných frameworků pro mobilní zařízení.
2. Navrhněte mobilní aplikaci pro vyhledávání zajímavých turistických, sportovních, kulturních akcí a míst v Brně. Pro vyhledávání bude možné využít chatbota, který uživateli navrhne akce a místa v okolí dle jeho požadavků. Naimplementujte aplikaci v programovacím jazyce Kotlin pro operační systém Android.
3. Implementujte minimalistickou použitelnou verzi řešené aplikace.
4. Otestujte aplikaci na testovacím vzorku uživatelů.
5. Analyzujte zpětnou vazbu uživatelů a realizujte iterativní úpravy řešené aplikace.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu.

Literatura:

- CAHN, Jack. CHATBOT: Architecture, design, & development. University of Pennsylvania School of Engineering and Applied Science Department of Computer and Information Science, 2017.
- SHEVAT, Amir. Designing Bots: Creating Conversational Experiences. O'Reilly Media, 2017. ISBN 978-1491974827.
- Joel Marsh. UX pro začátečníky. Zoner Press, 2019

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Dyk Tomáš, Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Cílem této bakalářské práce je vytvoření mobilní aplikace, která umožňuje využít chatbota pro vyhledávání zajímavých míst a akcí v Brně. Aplikace je vyvinuta na operační systém Android pomocí jazyka Kotlin. Nabízí uživateli možnost komunikace s chatbotem pomocí napovídajících tlačítek. Ten je implementován pomocí frameworku Rasa. Aplikace nabízí možnost procházení všech akcí nebo míst a zobrazení jejich detailních informací. Uživatel má dále možnost si zobrazit všechny položky na mapě. Data jsou uložena na serveru v databázi, kde je spravuje backend. Ten se stará o ukládání, zpracovávání a filtraci dat. Na serveru je umístěn společně s chatbotem.

Abstract

The aim of this bachelor thesis is to create a mobile application that allows user to use a chatbot to search for interesting places and events in Brno. The application is developed on Android operating system using Kotlin language. It offers the user the possibility of communicating with the chatbot using hint buttons. It is implemented using the Rasa framework. The application offers the possibility to browse all events or places and view their detailed information. The user also has the possibility to view all items on the map. The data is stored on the server in a database where it is managed by the backend. It takes care of storing, processing and filtering the data. It is placed on the server together with the chatbot.

Klíčová slova

mobilní aplikace, Kotlin, Android, Chatbot, Rasa, Ktor, databáze, pochopení přirozeného jazyka, zpracování přirozeného jazyka

Keywords

mobile application, Kotlin, Android, Chatbot, Rasa, Ktor, database, natural language understanding, natural language processing

Citace

MISAŘ, Ondřej. *Mobilní aplikace pro vyhledávání zajímavých akcí a destinací v Brně*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Dyk,

Mobilní aplikace pro vyhledávání zajímavých akcí a destinací v Brně

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Dyka. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Ondřej Misař
11. května 2022

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Tomáši Dykovi za to, že mi dal možnost navrhnutí vlastního tématu. Dále bych mu rád poděkoval za odborný přístup v podobě častých konzultací a rad směřovaných ke všem částem práce.

Obsah

Kapitola 1

Úvod

Chatbot je v dnešní době využíván pro řadu úkolů, v rámci kterých slouží k nahrazení komunikace pracovníka s uživateli. Jeho velkou výhodou je možnost komunikace s více uživateli najednou. Umožňuje zlepšit uživatelům práci se systémem.

Tato práce má za cíl vytvoření mobilní Android aplikace, která bude využívat technologii chatbot. Cílem je uživatelům umožnit vyhledávání oficiálních akcí nebo turistických míst ve městě Brně. Pro vyhledání je možné použít řadu filtrů a to podle vzdálenosti od uživatele, městské části, datumu a kategorie. Aplikace obsahuje další části v podobně listu všech položek, možnost zobrazit detailní informace, a také umožňuje zobrazení všech položek na mapě. Celý systém je rozdělen do dvou částí. První část je Android aplikace, která slouží hlavně na zobrazování dat. Druhá část, server, je rozdělena dále do dvou částí a to na chatbot pro komunikaci s uživatelem a backend pro práci s daty a filtrování akcí a míst.

Práce je strukturovaná do jednotlivých kapitol. Nejprve je vysvětlen v teoretické části způsob a princip vývoje chatbota v současné době. Dále je vytvořen průzkum současných aplikací pro vyhledávání zajímavých míst. Poté je vytvořen návrh pro celý systém a jeho příslušné části. Podle vytvořeného návrhu následně proběhne implementace definovaného systému. Bude se jednat o implementaci chatbota, Android aplikace a na konec vývoj backendu. Na konci práce bude aplikace otestována na testovacím vzorku uživatelů. Zpětná vazba bude zpracována a následně zakomponovaná do nové verze aplikace s návrhem na budoucí možné rozšíření pro aplikaci.

Kapitola 2

Chatbot

2.1 Co je Chatbot

Chatbot je program, který se snaží simulovat lidskou konverzaci. Proces komunikace může probíhat skrze psaný text nebo syntézu řeči. Je často používán pro automatizaci procesu z kategorie zákaznické podpory, finance, vzdělání nebo také komunikace pro HR. Je tak výhodný na místech, kde není dostatek pracovní síly pro komunikaci s uživatelem a to díky tomu, že je schopen obsluhovat několik uživatelů zároveň. Chatbot se poté skládá z části pro pochopení vstupních zpráv od uživatele. Zpracovaný vstup by měl být následně použit pro odpovídání uživateli a to může zahrnovat i využití externích zdrojů pro vytváření komplexní odpovědi.

2.2 Historie

Historie chatbotu začíná v roce 1850, když anglický informatik Alan Turing zveřejnil článek "Computing Machinery and Intelligence"[10]. V tomto článku si položil otázku, jestli by bylo možné pro počítač komunikovat způsobem nerozeznatelným od člověka. Proto v tomto článku vytvořil Turing test. Ten se skládá z jednotlivých testů, ve kterých se člověk snaží komunikovat s oponentem. Po komunikaci člověk rozhodne, jestli komunikoval s člověkem nebo s počítačem. Snaží se tedy testovat, jestli je chatbot na takové úrovni, že není možné poznat, zda se jedná o počítač nebo o člověka [13].

V roce 1966 byl představen jeden z prvních chatbotů, který se jmenoval Eliz a byl vytvořen profesorem Josephem Weizenbaumem. Eliz byla schopná rozpoznávat specifická slova a porovnat je vůči předem vytvořeným odpovědím. I když Eliz nebyla schopná projít turingovým testem, byly vytvořeny základní struktury chatbota, na kterých se následně stavělo. V roce 1972 byl představen další známý chatbot a to Parry, který vytvořil Kenneth Mark Colby. Tento chatbot se snažil simulovat pacienta se schizofrenií.

Další populární chatbot byl vytvořen v roce 1995 s názvem A.L.I.C.E.. Jedná se o mnohem komplexnější typ chatbota. Fungovala na principu, kdy vytvářela odpovědi s pomocí porovnávání vstupu proti párům, které jsou uloženy v znalostní databázi.

V dnešní době jsou chatboti rozšířeni na hodně místech a na vývojích se podílí velké společnosti jako Amazon, Google nebo Microsoft. Tito chatboti využívají nové moderní technologie v odvětví strojového učení [5].

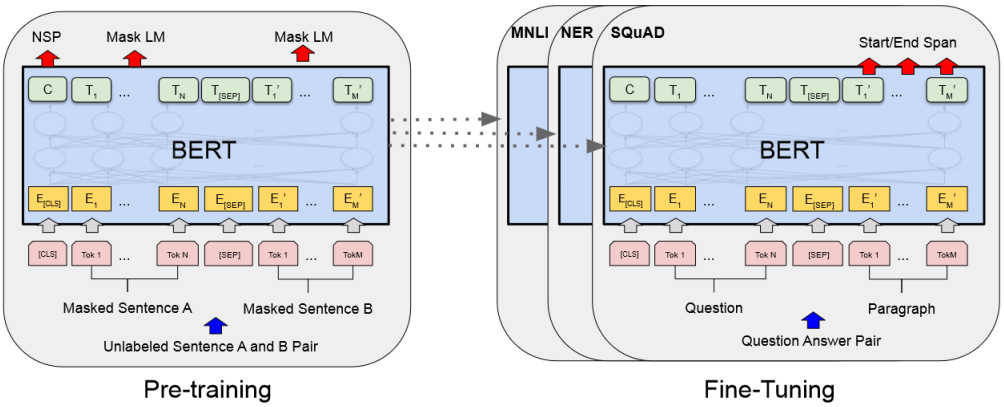
2.3 Architektury

V této kapitole jsou vysvětleny základní principy některých architektur, které jsou používány pro pochopení přirozeného jazyka (NLU) a generaci přirozeného jazyka (NLG). Tyto architektury jsou následně využívány pro tvorbu chatbotu.

2.3.1 BERT architektura

Bidirectional Encoder Representations from Transformers (BERT) je jazykový model vytvořený společností Google. Tento model používá dvě fáze, předtrénování a dolaďování. Jednou z hlavních výhod je možnost využívání stejného předtrénovaného modelu pro různé cíle pomocí dolaďování.

Samotná architektura se skládá z několika vrstev kodérových transformátorů. Důležité je, že se jedná o obousměrné transformátory [6].



Obrázek 2.1: Znázornění architektury pro předtrénování a dolaďování, obrázek převzat z [6]

Formát vstupu a výstupu

Aby byl BERT schopný zpracovat různé úkoly, vstupní formát umožňuje popisovat jednu nebo dvě věty v rámci jedné sekvence tokenů. Může se například jednat o otázku a odpověď. Samotná sekvence je následně reprezentací vstupu do modelu BERT. Ta se vždy skládá z počátečního tokenu `CLS`, který je na konci využíván pro předpovídání. Pokud poté následují dvě věty, jsou od sebe odděleny pomocí dvou kroků. Samotné věty jsou rozděleny pomocí speciálního tokenu `SEP`. Následně je do každého tokenu přidána informace, jestli patří do první věty nebo do druhé. Pro každý token je tedy přidána informace o hodnotě tokenu, k jaké větě patří a jeho pozici v sekvenci.

Předtrénování maskování

Aby bylo obousměrné trénovaní možné, je potřeba vytvořit úkol pro znemožnění, v rámci kterého by každé slovo nepřímo vidělo samo sebe. Toho jde docílit díky systému, kdy část vstupu je maskována. BERT se následně snaží předpovědět tyto tokeny. Nesnaží se předpovídat celou větu. Znamená to, že finální vektory, které odpovídají maskovanému tokenu, jsou poslány do výstupu normalizovaná exponenciální funkce přes slovní zásobu, která vybere finální slova. Následně ale nastává neshoda mezi předpovídáním a dolaďováním. A to z důvodu, že dolaďování nepoužívá předpovídání pro maskované tokeny. Aby bylo možné

tento jev zmírnit, tak náhodná slova nejsou vždy nahrazena maskovacím tokenem. Ve výsledku to tedy vypadá tak, že 15% tokenů je maskováno. Z vybraných tokenů je následně 80% nahrazeno maskovacím tokenem, 10% je nahrazeno náhodným tokenem a 10% není nahrazeno a zůstává původní token.

Před trénování předpovídání další věty

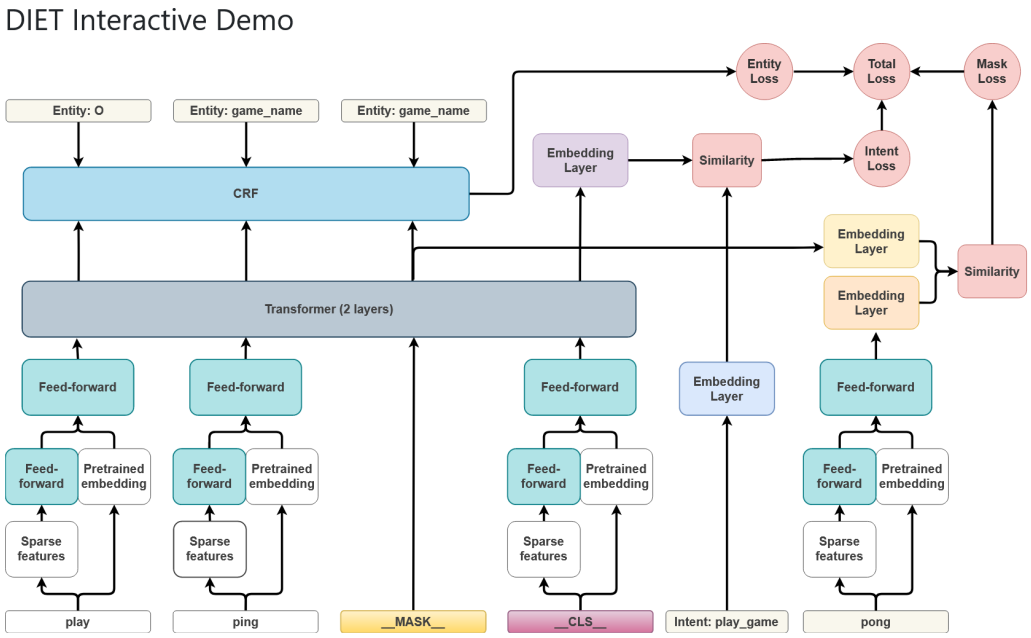
Aby bylo možné model používat pro odpovídání na otázky nebo jazykové odvozování, je potřeba zaznamenat vztah mezi dvěma větami. Pro tento účel je použito trénování pro binární předpovídání další věty. Jedná se o způsob vybírání první a druhé věty pro každý příklad. V 50 % je vybrána druhá věta, která má následovat s popiskem indikujícím, že se jedná o správnou větu, a v 50 % je vybrána náhodná druhá věta s popiskem, že daná věta není správná.

Dolaďování

Díky tomu, že transformátory obsahují mechanismus pro pozorování sama sebe, je samotné dolaďování rychlý proces. Proces probíhá pomocí výměny vstupní a výstupní vrstvy pro daný úkol. Následuje trénování pro dané parametry. Daný model je možné trénovat pro vstupy typu dvojice vět, jedné věty nebo otázky a odpovědi. Výstup v podobě tokenů je následně využíván pro vykonávání výsledného úkolu, například odpovídání na otázku nebo rozpoznávání věty pomocí tokenu `CLS`. Díky tomuto procesu je potřeba trénovat jenom malou část parametrů od základu. To znamená, že je rychlejší a výhodný z hlediska zdrojů.

2.3.2 DIET architektura

Dual Intent and Entity Transformer (DIET) je víceúčelová architektura pro intent a entity rozpoznávání. Jedna z hlavních výhod je možnost používat předem trénované jazykové modely a velká možnost úprav pro specifická řešení. Samotná architektura se poté skládá z následujících hlavních částí [4].



Obrázek 2.2: Znázornění všech částí DIET architektury, obrázek převzat z [12]

Featurization

Jedná se o proces, kdy vstupní věty jsou zpracovávány jako sekvence tokenů, které reprezentují slova nebo části slov. Na konci každé věty je následně přidán token `CLS`. Každý token je poté možné zpracovat přes dvě cesty. První z nich je pomocí předem trénovaného modelu, který na výstupu vrátí husté vektory. Druhá cesta naopak generuje řídké vektory, které jsou následně spojeny. Finální vektor je poté spojen pomocí feed forward sítě. Finální výsledek je spojen s hustým vektorem a opět projde přes feed forward síť. Výsledkem je list 256 desetinných čísel, neboli také vektor. Důležité je, že všechny feed forward vrstvy jsou řídké už od začátku. To znamená, že je již smazaná část napojení, v základu tedy 80 %. To napomáhá k tomu aby tyto vrstvy byly méně náročné, což je výhodnější pro využití u chatbotu. Tímto způsobem jsou zpracovávány všechny tokeny včetně `CLS` tokenu. Ten obsahuje vektory o celé větě. Při řídkých vektorech jsou sečteny pro jednotlivé vektory a při hustých vektorech jsou závislé podle typu předem trénovaného modelu, který používáme.

Transformátory

Transformátor je další část architektury, do které vstupují zpracované tokeny. Důležité je, že do transformátoru vstupuje stejný počet hodnot jako z něj vystupuje. Transformátor jako takový funguje na principu využití relativní pozornosti vůči pozici.

Rozpoznávání entity

Pro rozpoznávání entit se následně používá Conditional Random Field (CRF). Tato vrstva následuje po transformátoru, tedy na vstupu jsou posílány výsledky pro jednotlivé tokeny. V samotné vrstvě následuje feed forward vrstva, která na výstupu vrací predikci pro rozpoznání entity. Predikce mezi sebou mají ještě přechody, které napomáhají napojení predikcí mezi sebou. Tato vrstva přechodu a feed forward musí být trénovány pro správnou predikci.

Na konec jsou tyto predikce porovnány s očekávanými hodnotami a vypočítá se ztráta pro předpovídání entity, která se později bude používat pro samotnou optimalizaci.

Rozpoznávání intentu

Pro predikci intentu se bude používat zpracovaný token `CLS`, který projde přes transformátor. Tento výstup následně projde přes vrstvu, která konvertuje daný vektor do společného vektorového prostoru s intentem, jenž je očekáván. Tyto vektory jsou následně zpracovány na míře podobnosti, ze které je vypočítána ztráta pro předpovídání intentu.

Maskování

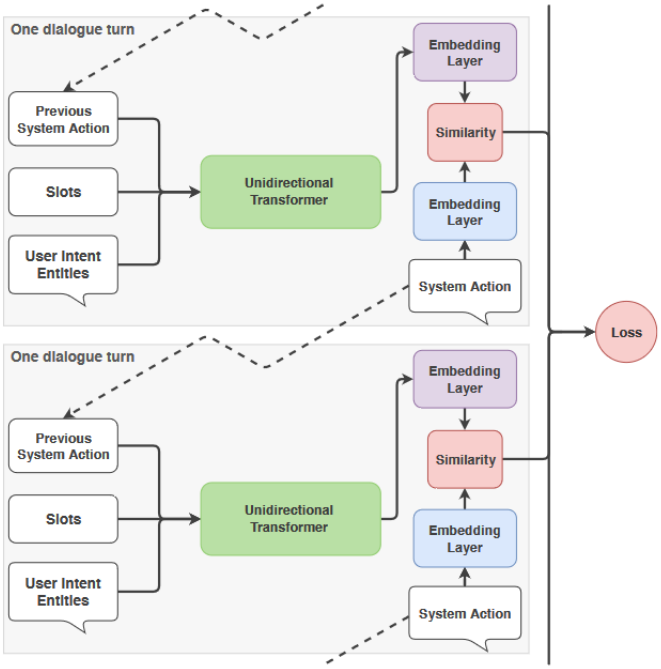
Během trénování dochází k procesu, kde náhodný token je nahrazen speciálním objektem `MASK`, který indikuje, že na tomto místě něco chybí. Původní token následně projde procesem featurizace. Objekt `MASK` tímto procesem neprochází a je posílán na vstup do transformátoru. Výstup pro původní token a objekt `MASK` projde vrstvou, která konvertuje vektory do společného vektorového prostoru a feed forward vrstvy. Tyto zpracované výstupy jsou následně porovnány na míře podobnosti a je vypočítaná ztráta pro masku. Cílem tohoto procesu je trénování modelu pro rozpoznávání obecných vlastností v textu a ne pouze vlastností specifických pro rozpoznávání.

Celková ztráta

Pro trénování samotného modelu se používá celková ztráta. Ta se skládá ze ztrát pro předpovídání entity, intentu a masky. Díky tomu, že je samotná architektura modulární, je možné nepoužívat jakoukoliv zmíněnou ztrátu a díky tomu se můžeme zaměřit na trénování pouze pro předpovídání intentu nebo entity.

2.3.3 TED architektura

Transformer Embedding Dialogue (TED) Policy je architektura zaměřena na vybírání jednotlivých kroků, které by měl chatbot vykonat. Velkým rozdílem je, že se nevyužívá klasifikátor pro vybírání akcí. Využívá se princip, kdy současný stav je společně porovnán se všemi akcemi, které chatbot může vykonat. Hlavní výhodou této architektury je, že je možné vybírat odpovědi jako reakce na konverzaci v minulosti a ne pouze jako přímou reakci [11].



Obrázek 2.3: Ukázka dvou kroků konverzace pro TED architekturu, obrázek převzat z [11]

Featurization

Pro vytváření vlastností se dále využívá vstup od uživatele, sloty, které ukládají informace v kontextu konverzace a předchozí vykonané akce. Pro zpracování vstupu od uživatele se využívá externí NLU systém, který zpracuje vstup a následně se vybraný intent a entity zakomponují v podobě vektoru. Akce jsou předpovídány z předem definovaného listu akcí. To znamená, že tyto akce jsou také zpracovány v podobě vektorů, které se budou dále používat. Tato architektura tedy nedokáže generovat nové odpovědi, ale pouze je vybírá. Nakonec jsou sloty opět zpracovány do vektoru. Je také důležité indikovat, jestli se zde slot nachází, nenachází nebo jeho hodnota není důležitá.

Transformátory

Vytvoření vlastnosti pro vstup od uživatele a akce, které může chatbot vykonat, jsou následně zpracovány pomocí transformátoru. Díky tomu, že transformátor využívá principu držení informace o pozici, je možné využívat různé části konverzace z minulosti při každém rozhodování. To umožňuje, že někdy je vstup od uživatele využit k vybrání nové akce a někdy je možné jej ignorovat a využít předchozí vstup z historie. Na výstup je poté aplikována hrubá vrstva, ta je také použita na akce, které může chatbot vykonat.

Podobnost

Tyto zpracované vrstvy jsou porovnány a pomocí StarSpace systému je vypočítaná podobnost mezi vstupem a akcemi, které může chatbot vykonat. Finální výběr je následně spojen dohromady a je poslán opět do transformátoru pro predikci s novým vstupem. To znamená, že podle konfigurace je možné využívat predikce, které již byly vykonány pro předpovídání

v kontextu delší konverzace. Výsledná ztráta z podobnosti je zakomponovaná do celkové ztráty, která je průměrem ztrát ze všech konverzací.

2.4 Framework Rasa open source

Rasa open source je framework vyvíjený společností Rasa [2]. Jedná se o framework, který poskytuje hlavní stavební bloky pro vytváření chatbotu. Hlavní části, které Rasa open source poskytuje, jsou NLU, která konvertuje text uživatele na strukturovaná data, a kontextové konverzace, které nám umožňují definovat, co by měl chatbot dělat podle kontextu celé konverzace. Poslední hlavní část je integrace chatbota. Rasa umožňuje připojení na různé platformy, ale také jako vlastní API. Dále umožňuje napojení na vlastní backend logiku pro databázi [3].

2.4.1 Trénování modelu / Konfigurace trénování

Aby se určil způsob, jakým se bude model trénovat, je potřeba definovat konfiguraci. Ta se skládá z jednotlivých komponent, které jsou definovány v konfiguračním souboru. Komponenty jsou různých typů a to podle toho, jakou funkcionalitu zprostředkovávají. Mezi ně se řadí jazykové modely, featurizéry, klasifikátory intentu, extractor entit a nakonec vlastní komponenty [9].

Konfigurace se následně skládá ze dvou částí a to pipeline a policies. Pipeline definuje, jaké komponenty se použijí na trénování modelu pro NLU rozpoznávání. Policies následně definuje komponenty, které určují, co by měl chatbot dělat během konverzace. To znamená výběr odpovědi chatbora.

2.4.2 Pochopení přirozeného jazyka

Pro pochopení přirozeného jazyka se používají postupně navazující komponenty, které následně transformují vstupní data do konečného modelu, pomocí kterého je možné rozpoznávat vstupy od uživatele.

Tokenizer

Jako první část je potřeba zpracovat vstupní text a z něho vytvořit tokeny, které se budou následně používat. Znázornění je vyobrazené na obrázku 2.4. Pro tuto operaci je možně použít několik tokenizeru. První je `WhitespaceTokenizer`, který vezme vstupní věty a oddělí jednotlivá slova do tokenu podle mezery. Tento typ vyhovuje většině jazyků, které jsou strukturovány pomocí mezer mezi slovy. U jazyku, který není takto strukturován, je možné použit `JiebaTokenizer`, který vytváří tokeny pro čínštinu. Pro další jazyky je možné použít `MitieTokenizer` a `SpacyTokenizer`, které používají externí knihovny na vytvoření tokenů [9].

Obrázek 2.4: Ukázka zpracování textu do tokenů pomocí mezer

Featurizer

Další částí pro zpracování přirozeného jazyka je vytvoření vektorů pro jednotlivé tokeny předané z první části. Ukázku je možné vidět na obrázku 2.5. Komponenty se rozdělují do dvou kategorií a to podle toho, jaké vektory generují. Dělí se na řídké vektory a husté vektory, kdy řídké vektory ukládají pouze nenulové hodnoty, zatímco husté ukládají všechny hodnoty [9].

Husté vektory

Tyto komponenty jsou typické tím, že používají buďto externí knihovny nebo předem trénované modely. Mezi tyto featurizery patří `MitieFeaturizer`, který používá `MITIE`, `SpacyFeaturizer`, který využívá `Spacy`, `ConveRTFeaturizer`, který používá `ConveRT` a `LanguageModelFeaturizer`, který používá předem natrénované jazykové modely.

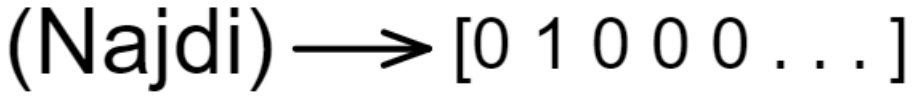
Řídké vektory

Řídké vektory vytváří komponenty, které nevyužívají externí nástroje a používají specifické algoritmy. Tyto komponenty se používají hlavně, pokud se pracuje s trénovacími daty, která nejsou specifikované a nepoužívají předem trénované modely.

První taková komponenta je `RegexFeaturizer`. Ta funguje na principu předem definovaných regulárních výrazů a pokud předávaná zpráva v tokenu obsahuje nějaký z těchto výrazů, tak to bude zaznamenáno do vektoru. Hodnoty z tohoto featurizeru dokážou používat pouze některé klasifikátory.

Jako další je možné použít `CountVectorsFeaturizer`. Tato komponenta slouží pro vytvoření gramatiky ze zpráv, intentu a odpovědí, které jsou definovány v trénovacích datech. Je možné použít pouze slova nebo také části slov až po jednotlivé znaky.

Poslední komponenta je `LexicalSyntacticFeaturizer`. Ta se pohybuje pomocí posuvného okna přes tokeny a vytvoří vlastnosti podle konfigurace. Je možné tedy konfigurovat, aby se vlastnosti vytvářely podle nastavení, zda je token na začátku nebo na konci věty, jestli je napsán velkými nebo malými písmeny, zda obsahuje číslo a nakonec podle části slova, a to podle prefixu nebo sufixu.



Obrázek 2.5: Ukázka zpracování tokenu do vlastností

Intent klasifikátory

Po zpracování vstupních dat následuje část, kde pomocí vytvořených vektorů jsou trénována spojení mezí vstupními zprávami a intentem, ke kterému patří. Tyto komponenty se dají rozdělit podle toho, jaký typ vektoru přijímají, tedy řídké nebo husté [9].

`MitieIntentClassifier` je klasifikátor, který používá MITIE pro klasifikaci.

Další klasifikátor je `SklearnIntentClassifier`. Ten vyžaduje husté vektory a trénuje pomocí lineární SVM. Následně seřadí intenty, které nebyly vybrány.

Jako jednoduchý klasifikátor je možné použít `KeywordIntentClassifier`. Pro klasifikaci nepotřebuje žádné vlastnosti, používá totiž jednoduché vyhledávání celého slova ve zprávě vůči definovaným intentům. To znamená, že je vhodný pouze na základní rozpoznávání.

Významným klasifikátorem je následně `DIETClassifier`. Jedná se o komponentu, která je vytvořena na zpracování všech typů vlastní. Funguje na architektuře DIET. Tato architektura umožňuje použít komponentu jako intent klasifikátor a entity rozpoznávač. Samotná komponenta následně neposkytuje předem trénované jazykové modely. Je ale schopná je využívat a je možné je definovat v pipeline, pak budou následně využity.

Jako poslední komponentu lze použít `FallbackClassifier`, která vyžaduje, aby již některý z předchozích klasifikátorů vybral intent. To znamená, že pokud předchozí klasifikátor nebyl schopný rozpoznat větu nebo pokud jistota není nad určitou hranicí, tak bude následovat tato komponenta. Ta přidá nový intent značící, že nebylo schopné rozpoznat danou větu.

Entity Extractors

Zároveň s vybíráním intentu souvisí vybírání entit. Tyto komponenty extrahují z tokenů entity, které jsou definovány v trénovacích datech [8].

Jako první komponenta se dá použít již zmiňovaný `DIETClassifier`.

Další komponenta je `MitieEntityExtractor`, která potřebuje Mitie tokeny a používá Mitie extractor.

Na stejném principu funguje `SoacyEntityExtractor`, která potřebuje Spacy tokeny a používá externí Spacy extractor.

Komponenta `CRFEntityExtractor` vykonává entity rozpoznávání na stejném principu jako DIETClassifier. To znamená, že používá CRF architekturu pro rozpoznávaní entit.

`DucklingEntityExtractor` je komponenta, která slouží pro rozpoznávání číselných entit jako je datum, vzdálenost a peníze.

Dále jde použít `RegexEntityExtractor`, která využívá definované regulární výrazy pro rozpoznávaní entit.

Jako poslední se dá použít `EntitySynonymMapper`. Tato komponenta se používá po již definované entity extractoru. V případě, že jsou definovaná některá synonyma v trénovacích datech, tak budou podle této definice namapovány.

Selectors

Jedná se o komponenty, které predikují odpovědi chatbota podle předem definovaných postupů pro konverzaci v trénovacích datech.

Pro tuto sekci pipeliny existuje pouze jedna komponenta a to `ResponseSelector`. Ta zpracovává oba typy vektorů, pomocí kterých se snaží následně sestavit odpovědi používané v kontextové konverzaci.

2.4.3 Vybírání odpovědí

Pro výběr odpovědi se definují komponenty, podle kterých chatbot rozhoduje, jaká akce by měla nastat v jednotlivých krocích konverzace. Každý typ komponenty při jednotlivém kroku v konverzaci předpovídá následující akci, kde je následně vybrána akce s největší důvěrou. Pokud ale nastane případ, kdy dvě a více komponent předpoví další akci se stejnou důvěrou, rozhoduje se výběr podle priority jednotlivých komponent. Největší prioritu má RulePolicy, poté následuje MemoizationPolicy společně s AugmentedMemoizationPolicy, další v pořadí je UnexpecTEDIntentPolicy a poslední je TEDPolicy [9].

TED Policy

TED Policy je komponenta, která využívá TED architekturu pro předpovídání jakou další akci by měl chatbot vykonat. Díky této architektuře dokáže chatbot využívat historii konverzace pro vybírání odpovědi. Samotná komponenta ale nedokáže generovat nové odpovědi, ale pouze vybírá z definovaných akcí, které může chatbot vykonat. Komponenta tedy dovoluje změnu jak moc velká historie konverzace se bude využívat při vybírání odpovědi.

UnexpecTED Intent Policy

`UnexpecTEDIntentPolicy` je pomocná komponenta, kterou je potřeba využívat s další komponentou pro předpovídání odpovědi. A to hlavně z důvodu, že tato komponenta předpovídá pouze jednu akci. Využívá se tedy pro to, aby chatbot byl schopný reagovat na nepravděpodobné kroky od uživatele. Tato komponenta funguje opět na principu TED architektury. Hlavní rozdíl je v tom, že nepředpovídá jakou další akci by měl chatbot udělat. Naopak se učí, jaká sada intentu je nejvíce pravděpodobná, k jakému vstupu od uživatele v kontextu trénovacích konverzací může dojít. Poté tedy během konverzace využije natrénované informace tak, že při NLU predikci zkontroluje, jestli daný intent je pravděpodobný. Pokud v kontextu konverzace je pravděpodobné, že k němu dojde, UnexpecTED Intent Policy nebude předpovídat žádnou akci. Pokud je ale nepravděpodobné, že k němu dojde, tak je předpověděna akce pro nepravděpodobný intent s maximální důvěrou.

Memoization Policy

`MemoizationPolicy` je komponenta, která využívá příběhy z definovaných trénovacích dat. Funguje na principu, že se podívá, jestli současná konverzace odpovídá některému příběhu z trénovacích dat. Jestli tak tomu je, další akce je předpověděna s maximální důvěrou, pokud tomu tak není, komponenta nic nepředpoví. Kvůli lepšímu předpovídání z trénovacích dat se využívá možnost sledování několika předchozích akcí.

Augmented Memoization Policy

`AugmentedMemoizationPolicy` funguje na podobném principu jako Memoization Policy. To znamená, že využívá příklady z trénovacích dat s danou historií. Také ale využívá mechanismus pro zapomínání části kroků v historii. Snaží se tedy najít další akci z předem definovaných příběhů s redukovanou historií.

Rule Policy

`RulePolicy` je komponenta, které předpovídá další akce pro části konverzace, které mají pevně definované, jak by se měly chovat. Proto tedy využívá pro předpověď pravidla, která jsou předem definována v trénovacích datech.

2.4.4 Struktura dat pro trénování

Pro trénování chatbota se používá několik různých typů trénovacích dat. Tyto data jsou strukturovány v souborech typu YAML, pomocí korespondujících klíčů pro daný typ trénovacích dat [9].

NLU trénovací data

Jedná se o typ trénovacích dat, které se využívají pro trénování NLU rozpoznávání. Data se skládají z jednotlivých kategorií a to v podobě intentu. Tyto příklady následně mohou obsahovat definici entit, kde entity jsou strukturované části informací, které obsahuje vstupní zpráva. Může se tedy jednat o text obsahující informace o specifikaci daného intentu. Dále je možné definovat synonyma, která se využijí na mapovaní hodnoty entity po tom, co je rozpoznána. Dají se tedy využít, pokud je možné entitu vyjádřit různými slovy, Ale cílem je, aby se používala pro všechny stejná hodnota. Pro zlepšení klasifikace intentu a vybírání entity je možné využit definice regulárního výrazu. Při využití regulárního výrazu pro přidání informací pomocí `RegexFeaturizer` se daný výraz nenamapuje přímo na definovaný intent nebo entitu. To znamená, že se vytváří asociace mezi daným intentem nebo entitou pro konkrétní vstupní zprávu. Regulérní výrazy je možné použít i pro samotné vybírání entit, kde se vytváří přímá asociace výrazu s danou entitou. Regulérní výrazy je také možné definovat jako list slov, ze kterých je následně vygenerován výraz, který bude použit stejným způsobem jako předchozí varianta. Tento zápis může být například vhodný pro rozpoznávání listu názvů.

Konverzační trénovací data

Aby byl chatbot schopný předpovídat jaké akce by měly být volány, je potřeba definovat konverzační trénovací data. Ty definují, jaké kroky by měl chatbot v dané konverzaci vykonávat.

Jako první je možné definovat příběhy. Jedná se o typ konverzačních trénovacích dat, která napomáhají chatbotu předpovídat předem nedefinované cesty v konverzaci. Tyto trénovací data jsou definována ve formátu uživatelského vstupu a to v podobě intentu nebo entit. Pokud se využívá zapamatování daných informací v podobě slotů, je potřeba je také definovat. Odpověď chatbota je následně definována pomocí akcí. Pro vytváření komplexnějších příběhů je možné využít definování Kontrolních bodů. Ty jsou využívány pro modularizaci příběhů. Při jejich používání je ale potřeba jich nevyužívat velké množství, protože zpomalují trénování. Další možností je využití Nebo výrazu, pomocí kterého je možné definovat, aby některé vstupní intenty uživatele byly využívány stejným způsobem. Při velkém používání je opět jejich nevýhodou zpomalování trénování modelu.

Dále je možné definovat pravidla. Využívají se krátké části konverzace pro definování cesty, která by měla být vždy stejná. Pro definici pravidel je potřeba určit, který vstupní intent toto pravidlo vyvolá. To znamená, že pravidla by měla být využívána pouze pro definici velmi malých částí konverzací, jež uživatel vyvolá specifickým intentem a chatbot zareaguje pokaždé stejnou akcí. Pro pravidla je také možné definovat podmínky, které by měly být splněny pro vyvolání dané akce. Tyto podmínky mohou být definovány pomocí slotů.

2.4.5 Doména

Pro definici, jak má samotný chatbot fungovat, je potřeba použít doménový soubor. Ten slouží pro definici intentů, entit, slotů, odpovědí a akcí. Také se zde definuje konfigurace, kterou bude chatbot využívat, a to informace o expiračním čase a jestli by sloty měly být přeneseny do nové konverzace [9].

Jako první je potřeba definovat, jaké intenty chatbot využívá. Pro jednotlivé intenty je možné definovat, jaké entity používá nebo nepoužívá. Díky tomu je možné se vyhnout

rozpoznávání entity, kdy ji nepotřebujeme. Dále je možné definovat, jaké entity je schopen rozpoznávat.

V doménovém souboru je možné definovat sloty. Ty se využívají pro ukládání informací, které následně může chatbot využít pro vybírání dalších kroků. Pro sloty je možné určit jestli jejich nastavení následně ovlivní konverzaci a to v podobě vybírání dalšího kroku. Sloty je možné vytvořit v několika typech, které určují jaký druh informace bude uložen. Jedná se tedy o text, bool, kategorie, číselný, list, žádný nebo vlastní typ. Pro každý slot je také potřeba nastavit, jakým způsobem bude mapován. Jedná se o mapování ze specifické entity, kde se uloží její hodnota. Poté je možné ukládat text, který byl poslán na vstup pro daný intent nebo je možné definovat, jaká hodnota by se měla uložit, pokud byl rozpoznán daný intent. Pro tato mapování je možné nastavit podmínky, které musí být splněny. Pokud se využije vlastní typ mapování, je možné nastavit akci, která následně naplní daný slot.

Dále je možné definovat, jaké odpovědi bude chatbot používat. Jedná se tedy o informace, které nevyužívají napojení na vlastní server. Jde hlavně o odpovědi v textu, ale je možné také definovat obrázek nebo tlačítka, které uživatel může využít k přeskočení rozpoznávání. Nakonec je možné definovat i vlastní typ odpovědi.

Aby chatbot byl schopný odpovídat, je potřeba definovat akce, které může vykonávat. Může se jednat o odpovědi, volání externího serveru nebo získávání informací z databáze. Jako speciální typ akce je možné definovat formulář, který se využívá k postupnému získávání informací pomocí mapování slotů. Definují se sloty, které je potřeba mapovat a pro které budou postupně volány otázky za účelem získaní těchto informací.

2.4.6 Externí server

Aby bylo možné použít vlastních akcí, je potřeba definovat externí server. Pokud chatbot vyhodnotí, že by měla následovat vlastní akce, zašle požadavek typu POST na definovaný externí server. Tento požadavek bude obsahovat všechny informace o aktuálním stavu konverzace. Externí server poté následně odpovídá ve formě odpovědí, které má chatbot využít. Zároveň s odpověďmi je možné posílat události, které mohou ovlivnit konverzaci, a to v podobě slotů, připomínek, nastavením následují akce nebo ovládáním konverzace v podobě pozastavení, obnovení a restartování [9].

Kapitola 3

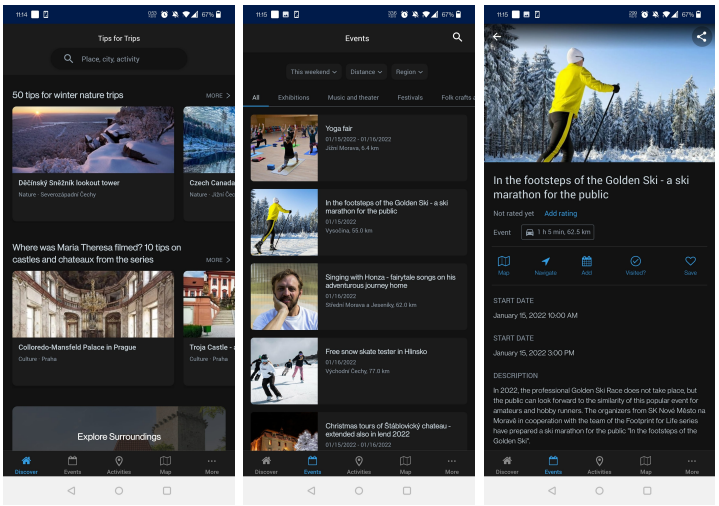
Návrh

3.1 Průzkum současných řešení

V současné době existuje několik aplikací, které nabízí možnost vyhledávání zajímavých míst a akcí. Neexistuje ale varianta s chatbotem, která by pomáhala uživateli s vyhledáváním. Mezi aplikace s vyhledáváním míst a akcí v České republice patří hlavně projekt Kudy z nudy nebo také Mapy. V zahraničí je oblíbená aplikace Meetup.

3.1.1 Kudy z nudy

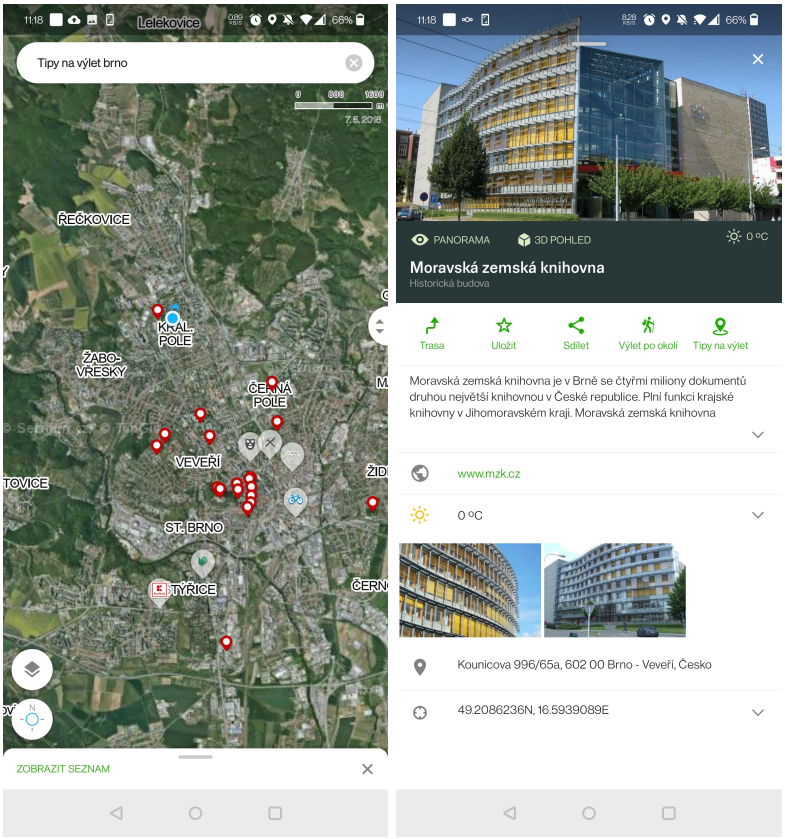
Kudy z nudy je projekt, který je součástí České centrály cestovního ruchu neboli CzechTourism. Hlavním cílem tohoto projektu je podpora domácího cestovního ruchu. Díky této podpoře má aplikace Kudy z nudy velmi rozšířenou databázi míst a akcí v České republice. Zároveň také aplikace a web dosahují vysokých návštěv a to až 3,5 milionů.
Jednou z největších nedostatků pro uživatele je vyhledávání samotných akcí a míst. Samotná aplikace nemá extenzivní filtrování a nenabízí žádnou možnost komunikace pomocí chatu. Zároveň projekt není vázán na aplikaci Kudy z nudy. Protože se jedná o projekt podporovaný CzechTourism je možné požádat o přístup k API, přes které jde komunikovat s celou databází akcí a míst.



Obrázek 3.1: Snímky obrazovky z aplikace Kudy z nudy

3.1.2 Mapy

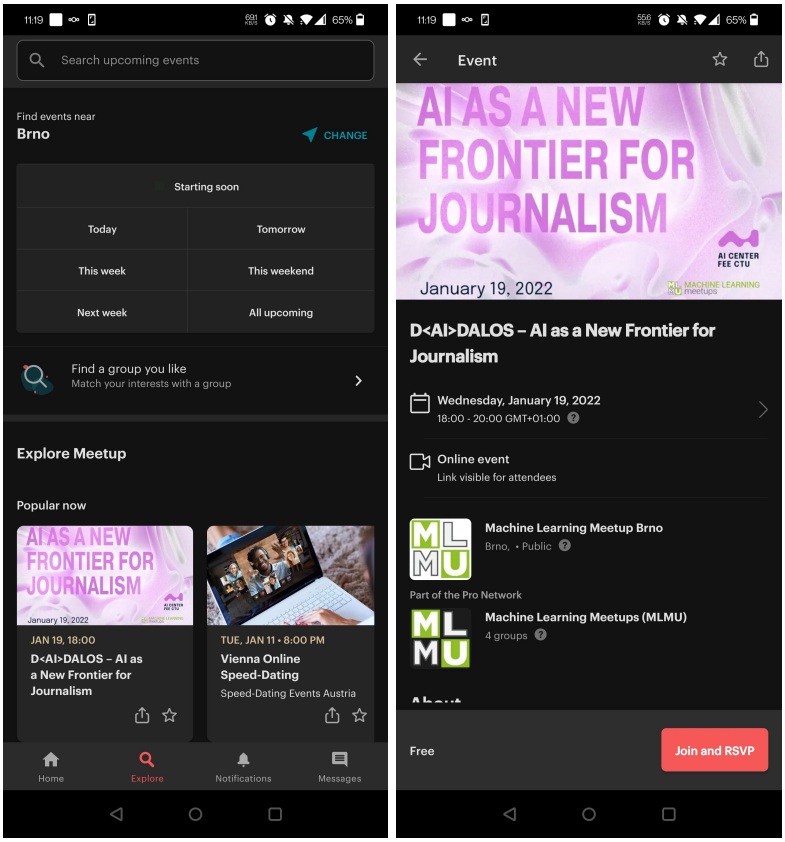
Mapy cz je aplikace a web, který je vyvíjen společností Seznam. Samotná aplikace necílí čistě na uživatele, kteří by pouze chtěli vyhledávat zajímavé akce a místa. Z toho důvodu nenabízí možnost vyhledávat konkrétní akce. V čem ale mapy mají výhodu, je velmi dobré vyhledávání turistických míst.
Samotná aplikace ale nijak nepomáhá uživateli ve hledání zajímavých míst, kam by se mohl vydat. Jedná se tedy spíše o vyhledávání, kde již uživatel ví o místo, kam by chtěl cestovat. Mapy cz nabízí také api přístup, díky tomu by bylo možné je použít pro mapové vyhledávání míst nebo zobrazování akcí.



Obrázek 3.2: Snímky obrazovky z aplikace mapy

3.1.3 Meetup

Meetup je zástupce pro vytváření a vyhledávání akcí po světě. V dnešní době patří mezi jedny z největších platforem nabízející pořádání a možnost navštívení akcí. V současné době se v aplikaci nachází každý měsíc miliony akcí.
Aplikace nabízí pouze vytváření nebo vyhledávání akcí. Z pohledu vyhledávání se jedná o velmi rozšířené filtrování. Může se jednat o virtuální nebo lokální akce. V současné době není příliš rozšířena v České republice. Zároveň nepodporuje vyhledávání míst. Jedná se tedy o platformu, která nabízí velmi dobrou podporu pro akce, zároveň ale nijak nepodporuje konkrétní místa.



Obrázek 3.3: Snímky obrazovky z aplikace Meetup

3.2 Architektura celého systému

Celý systém je rozdělen do dvou částí. První část je klientská část neboli samotná aplikace. Další je serverová část, která se dělí na backend a chatbot. Klient komunikuje se serverem pomocí API rozhraní.

Tuto architekturu jsem zvolil, protože nabízí možnost flexibilního a dynamického vývoje. Díky tomu bylo možné samostatně navrhnout a iterativně implementovat chatbota a jeho napojení na backend, což má vliv na větší kontrolu nad používanými daty. Také umožňuje odlehčení zátěže chatbota na zdroje mobilní aplikace. V pozdějších částech vývoje existuje možnost změny konverzací, které chatbot podporuje bez nutnosti nového vydání aplikace. Aplikace tedy funguje hlavně na principu zobrazování dat, které zpracovává server.

3.3 Návrh Chatbota

Tato kapitola je zaměřena na bližší popsání návrhu chatbota. Jedná se o vstup od uživatele, který by měl být schopný rozpoznávat. Dále využívá odpovědi a také samotné struktury jednotlivých konverzací s potřebnými vstupními daty. Dále se tato kapitola zaměřuje na způsob komunikace pro získávání informací z databáze a nakonec také na způsoby pro ulehčení komunikace uživatele s chatbotem.

3.3.1 Uživatelský vstup

Pro komunikaci uživatele s chatbotem je potřeba definovat, jaký typ vstupu je třeba zpracovávat a klasifikovat. To zahrnuje i samotná trénovací data, která jsou použita pro trénování jazykového modelu. Aby bylo možné používat uživatelský vstup, je nutné definovat, jaké intenty a entity je možné rozpoznat.

Hlavním cílem chatbota je umožnit uživatelům filtrovat akce a místa, proto jsou definovány následující intenty a entity:

- intent pro inicializaci vyhledávání,
- intent společně s place a event entity pro specifikaci, co se bude vyhledávat,
- intent a entity pro specifikaci data,
- intent a entity pro definici vzdálenosti,
- intent a entity pro specifikaci městské části,
- intent a entity pro zadání kategorie,
- intent pro potvrzování,
- intent pro zamítání,
- entity definici pozice uživatele.

3.3.2 Odpovědi chatbota

Aby chatbot mohl uživateli odpovídat, je potřeba definovat jednotlivé akce, které je možné vykonat. Akce mohou obsahovat přímou textovou odpověď nebo dostanou odpověď od externího serveru. Jsou tedy navrženy akce, které chatbot používá pro komunikaci o filtrování. Zároveň s odpověďmi je nutné navrhnout nápovědy a to tlačítka, která napomáhají pro přeskočení NLU rozpoznávání intentu a entity. Jedná se o následující typy akcí:

- inicializaci konverzace s možností výběru z nápovědy pro vyhledávání místa nebo akce,
- otázky pro jednotlivé filtry a to vzdálenost, městská část, datum a kategorie s příslušnými nápovědami,
- otázky, zda-li uživatel chce vyhledávat podle specifikovaných filtrů,
- akce a odpovědi pro samotné vyhledávané akce a místa.

3.3.3 Informační sloty

Aby bylo možné pro uživatele mít kontextovou konverzaci, kde si chatbot ukládá data, která budou následně využita, je potřeba navrhnout datové sloty. Zde se jedná o jednotlivé informace, které je možné využít k filtrování. Každý slot tak bude korespondovat k příslušným entitám, které již byly definovány. Jedná se o sloty pro vzdálenost, pozici uživatele, městskou část, datum a kategorie.

3.3.4 Konverzace

Důležitou součástí návrhu chatbota jsou samotné konverzace, které můžou probíhat s uživatelem. Jelikož se jedná o konverzace, které se mohou dělit do několik cest, rozhodl jsem se rozdělit konverzace tak, že se v jednotlivých bodech dělí podle toho, jak chce uživatel filtrovat. To znamená, že konverzace je složena z jednotlivých modulů, které je možné za sebou postupně navazovat a vytvářet tak kompletní konverzace. Zde se jedná o dvě hlavní cesty, a to vyhledávání akcí nebo míst. Každá z cest se skládá z počátečního modulu, jednotlivých filtrovacích modulů a modulu pro konečné vyhledání.

Začátek konverzace

Na začátku je nutné, aby uživatel nebo samotný chatbot začal konverzaci pro vyhledávání. Následným výsledkem by měla být informace, co se má vyhledávat, zda místo nebo akce. Tento modul je tedy potřeba využít hned na začátku konverzace, aby uživatel mohl vyhledávat.

Filtrování

Konverzační moduly pro filtrování mají všechny stejnou strukturu, při čemž se mění samotný filtr, odpovědi a akce. Uživatel tedy využívá konverzačních modulů pro filtrování pomocí vzdálenosti od jeho aktuální polohy, městské části, datumu a kategorie. Struktura modulu následně obsahuje otázku, zda-li uživatel chce daným způsobem filtrovat. Při negativní odpovědi končí tento modul a pokračuje v konverzaci. Při pozitivní odpovědi je následně uživatel dotázán na specifikaci hodnoty, podle které chce dále filtrovat. Uživatel zde dostane doporučené hodnoty pro daný filtr, pak tato cesta končí a konverzace pokračuje dále.

Vyhledání podle zadaných filtrů

Zde se jedná o konverzační modul, který je použit na konci konverzace s cílem, aby chatbot vyhledal a zobrazil výsledky pro specifikované filtrování. Moduly se dělí podle toho, co je vyhledáváno, zda místo nebo akce. Po následném zobrazení výsledků je konverzace ukončena opět počáteční otázkou, aby uživatel mohl ihned začít další vyhledávání.

3.3.5 Napojení na databázi

Aby mohl chatbot využívat informace z databáze, je navrhnuto napojení na externí server. S ním chatbot komunikuje pomocí API rozhraní. Externí server poté musí být schopný zpracovat vstup a podle zvolené akce vykonává daný úkol. Ten by měl zahrnovat vyhledání dat v databázi a následnou odpověď chatbota společně s typem odpovědi, kterou by měl chatbot použít. Externí server se tedy používá pouze pro akce na vyhledávání akcí a míst a zpětnou odpověď ve vlastním formátu.

3.4 Návrh android aplikace

Tato kapitola je zaměřena na popsání návrhu android aplikace. Obsahuje komunikaci s chatbotem a grafický návrh samotné aplikace.

3.4.1 Komunikace s chatbotem

Android aplikace by měla mít možnost, aby uživatel mohl komunikovat s chatbotem. Samotná komunikace probíhá přes API rozhraní, které běží na serveru s chatbotem. Vstupní zprávy od uživatele jsou z aplikace posílány na rozhraní chatbotu. Ten jej zpracuje a následně pošle odpověď. Ta může obsahovat několik typů dat. A to jednoduchou textovou odpověď, dále také vlastní formát, který obsahuje informace o vyhledaných místech jako název a jejich identifikátor a také data pro pomocná tlačítka, která obsahují název a text pro přeskočení klasifikace intentu.

3.4.2 Návrh GUI

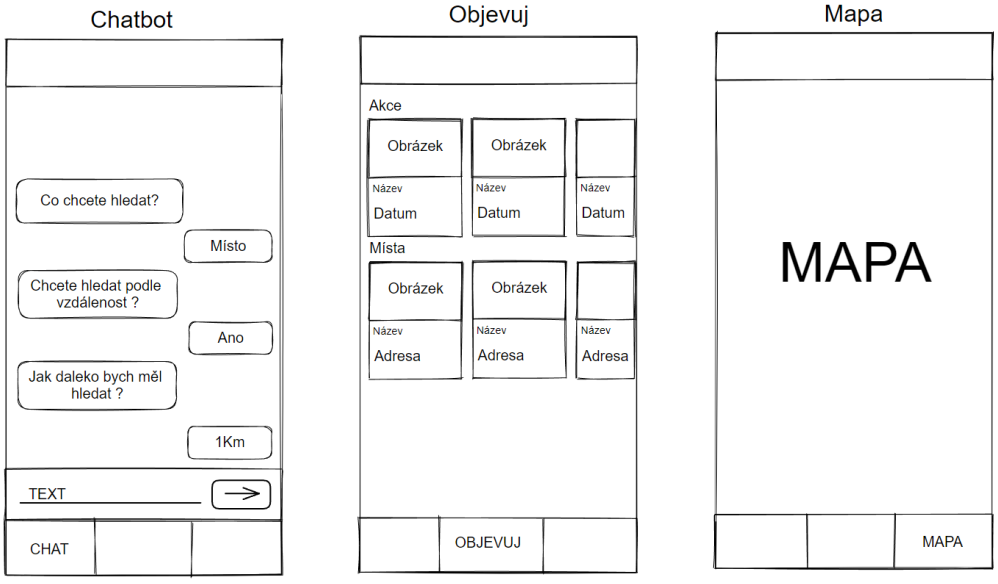
Pro návrh aplikace jsem vytvořil základní vzhled uživatelského rozhraní, které bylo implementováno. Hlavním cílem byl návrh, který splňuje základní podmínky a bude možné jej rozšířit.

Aplikace je rozdělena do tří hlavních sekcí, mezi kterými je možné navigovat pomocí spodního menu. Jedná se o sekce chatbot, objevuj a mapa.

Chatbot sekce je zobrazena na obrázku 3.4a. Slouží pro samotnou komunikaci uživatele s chatbotem. Zajišťuje tedy zobrazení konverzace, posílání zpráv a využívání pomocných konverzačních tlačítek. Uživatel si také může zobrazit detailní informace vyhledaných akcí nebo míst.

V sekci objevuj, která je vyobrazena na obrázku 3.4b. Má uživatel možnost vyhledat si různé akce a místa. Dále je také možné si zobrazit jejich detail a podívat se na všechny akce a místa nacházející se v databázi.

Sekce mapy, kterou je možné vidět na obrázku 3.4c. Slouží pro zobrazení všech akcí a míst na základě jejich polohy na mapě. Uživatel si může také zobrazit samotný detail akcí nebo míst.

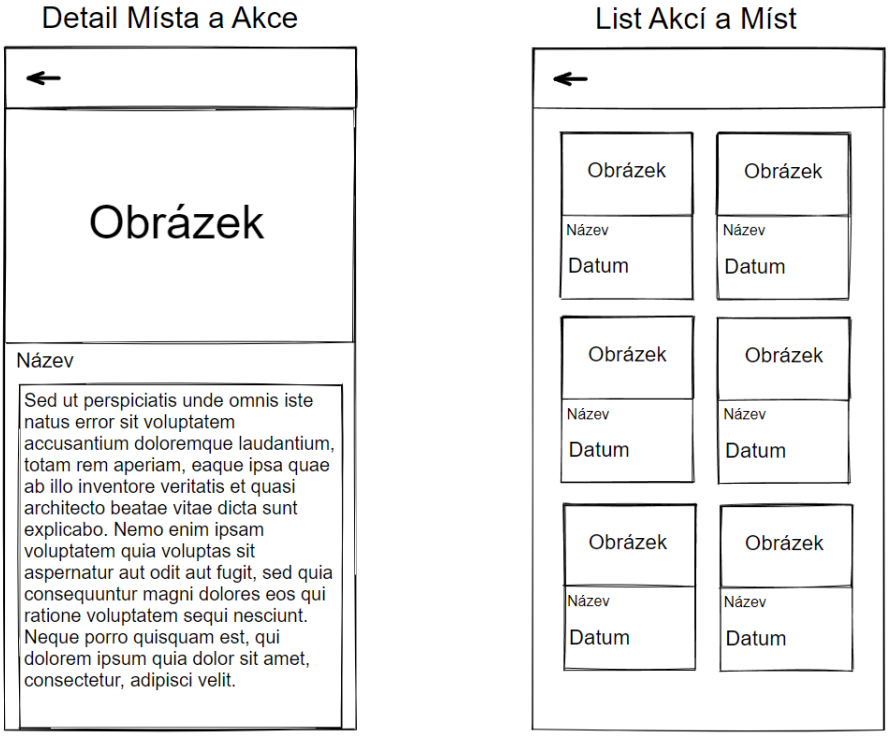


(a) Sekce chatbot (b) Sekce objevuj (c) Sekce mapa

Obrázek 3.4: Grafický návrh pro sekce chatbot, objevuj a mapa

Pro zobrazení všech akcí a míst existuje samostatná obrazovka. Je možné se na ní dostat skrze sekci objevuj. Dále je možné si zobrazit detail akce nebo místa. Ukázka návrhu je možné vidět na obrázku 3.5b.

Akce a místa mají podobný styl detailu. Kde se zobrazují všechny důležité informace. Na detail je možné se dostat několika cesty a to přes sekci objevuj a mapy nebo skrze list všech akcí a míst. Návrh je znázorněn na obrázku 3.5a.



(a) Detail položky (b) List položek

Obrázek 3.5: Grafický návrh pro detail a list akcí a míst

3.5 Návrh Back-endu

V této kapitole je přiblížen návrh backendu. Ten je využíván chatbotem a android aplikací. To znamená, že velká část funkcionalit pro aplikaci a chatbot jsou zastřešeny na backendu, kde s ním komunikují pomocí API rozhraní. To je společné pro aplikaci i chatbota. Návrh se zaměřuje na databázi, zpracování dat a poté následně algoritmy, které jsou používány pro filtrování.

3.5.1 Databáze

Z důvodu kompletní kontroly nad daty, která se zobrazují v aplikaci a chatbotu, je navrhnuta databáze sloužící pro ukládání dat týkajících se akcí a míst. Tyto struktury jsou si podobné, ale nejsou stejné. O akcích se ukládají následující informace: identifikátor, jméno, popis, informace o vstupenkách, odkaz na vstupenky, obrázek, webový odkaz na akci, kategorie, pozice na mapě, datum pořádání a městská část. O místech se ukládají následující

údaje: identifikátor, jméno, popis, obrázek, webový odkaz na místo, adresa, webový odkaz na kontaktní informace, pozice na mapě a městská část.

3.5.2 Zpracování dat

Data, která se ukládají do databáze, jsou získána z oficiálních otevřených dat města Brna. Data pro akce a místa jsou provozovány společností TIC Brno. Informace o městských částech jsou následně vytvořeny z dat od ČÚZK. Pro jejich uložení do databáze jsou staženy přes API rozhraní a následně zpracovány do definovaných struktur. Využívají se tři API rozhraní a to pro akce, turistická místa a hranice městských části. Všechny tato rozhraní se využívají pouze pro získávání dat. Data pro zobrazení jsou získávána z již definované databáze.

3.5.3 Algoritmus pro vyhledání podle vzdálenosti

Pro využívání filtru podle vzdálenosti od uživatele, je potřeba definovat algoritmus pro výpočet vzdálenosti akce a místa od pozice uživatele. Ten se následně používá pro vyhledávání v databázi. Tento algoritmus je implementován podle [1].

Pseudokód algoritmu vypadá následně:

```
FUNCTION distance(lat1, lon1, lat2, lon2)
    R = 6371
    lat1Radians = lat1 * PI/180;
    lat2Radians = lat2 * PI/180;
    latRadians = (lat2-lat1) * PI/180;
    lonRadians = (lon2-lon1) * PI/180;

    a = sin(latRadians/2) * sin(latRadians/2) +
        cos(lat1Radians) * cos(lat2Radians) *
        sin(lonRadians/2) * sin(lonRadians/2)
    c = 2 * atan2(sqrt(a), sqrt(1-a));

    RETURN R * c

ENDFUNCTION
```

Algoritmus využívá Haversinův vzorec. Funguje tedy na principu určení vzdálenosti dvou bodů na kouli vzhledem k jejich zeměpisné délce a šířce. Algoritmus tedy prvně definuje potřebné hodnoty a to poloměr Země v kilometrech. Poté jsou zeměpisné polohy převedeny ze stupňů do radiánů, kde se pomocí archaversine vypočítá samotná vzdálenost, která je následně vrácena v kilometrech

3.5.4 Algoritmus pro mapování městské části

Jako jeden z dalších filtrů je možnost použít městskou část. Pro použití městské části je potřeba pro jednotlivé akce a místa namapovat danou městskou část vůči její pozici na mapě. Využijí se souřadnice hranic městské části a poté proběhne kontrola, jestli se souřadnice akce nebo místa nachází v definovaném polygonu. Tento algoritmus je implementován podle [7].

Pseudokód algoritmu pro mapováni vypadá následovně:

```
FUNCTION isInPoligon(lat, lon, polygon)
    minX = polygon[0].X
    maxX = polygon[0].X
    minY = polygon[0].Y
    maxY = polygon[0].Y
    FOR 1 TO polygon.Length
        q = polygon[i];
        minX = min( q.X, minX )
        maxX = max( q.X, maxX )
        minY = min( q.Y, minY )
        maxY = max( q.Y, maxY )
    ENDFOR

    IF lat < minX || lat > maxX || lon < minY || lon > max
        RETURN false
    ENDIF

    inside = false
    FOR i = 0 AND j = polygon.Length TO i < polygon.Length WHERE j = i++
        IF ((polygon[i].Y > lon ) != ( polygon[j].Y > lon) &&
             lat < ( polygon[j].X - polygon[i].X ) * ( lon - polygon[i].Y ) /
             ( polygon[j].Y - polygon[i].Y ) + polygon[i].X)
            inside = NOT inside
        ENDIF
    ENDFOR

    RETURN inside

ENDFUNCTION
```

Algoritmus se skládá ze dvou částí. První je kontrola, jestli se může nacházet v hranicích polygonu. Poté se kontroluje pomocí systému, kdy je fixní souřadnice Y a posouvající souřadnice X. Následně probíhá kontrola jestli je souřadnice v hranici Y pro polygon a zároveň jestli souřadnice je v levé části hranice. Děje se tedy, že parametr `inside` je přepínán mezi true a false pokaždé, když je překročena hrana. To slouží pro získání informace, jestli bylo překročeno sudé nebo liché číslo hran.

Kapitola 4

Implementace

V této kapitole budu podrobněji popisovat implementaci jednotlivých částí systému. Jedná se o implementaci chatbota, android aplikaci a backend. U všech částí bude popsáno, jaké technologie byly použity a budou popsány důležité části pro danou implementaci a jejich význam.

4.1 Implementace Chatbota

Pro implementaci chatbota jsem vycházel z definovaného návrhu 3.3. Implementace je rozdělena do sekcí přibližující konfiguraci, doménu a trénovací data. V každé z nich bude popsáno, jaké byly použity komponenty a nastavení pro dosažení definovaných požadavků.

4.1.1 Použité technologie

Pro implementaci chatbota byl použit framework Rasa Open Source[1], který je popsán v sekci 2.4. Tento framework používá programovací jazyk Python. Jednotlivé konfigurace jsou poté napsány v YAML formátu. Pro zpracování datumů se používá knihovna dateparser.

4.1.2 Konfigurace

Konfigurace slouží pro definování komponent, které budou použity pro trénování rozpoznávání textu a vybírání odpovědí. Pro tyto účely je konfigurace rozdělena do dvou částí.

Pipeline

Tato konfigurace slouží pro trénování rozpoznávání textu. Skládá se z jednotlivých částí, které jsou potřeba k rozpoznávání textu. Mězi ně patří tokenizace, vytváření vektorů, klasifikace intentu a vybírání entit. Konfigurace bude implementována z následujících komponent s příslušným nastavením.

`WhitespaceTokenizer` pro vytváření tokenů podle mezer s nastavením, kde speciální znaky jsou nahrazeny mezerou.

`RegexFeaturizer` pro vytvoření vlastností s regulárními výrazy s nastavením, kde rozeznává velká a malá písmena.

`LexicalSyntacticFeaturizer` pro vytvoření vlastností s nastavením pro tři případy. Token před aktuálním podle, zda je text malým nebo velkým písmenem a jestli je první

[1] https://rasa.com/open-source/

písmeno velké. Aktuální token podle toho, jestli je text malým nebo velkým písmem, zda je první písmeno velkým a zda je token na začátku nebo na konci věty a či je číslem. A token po aktuálním, který má stejné nastavení jako před aktuálním tokenem.

`CountVectorsFeaturizer` pro vytvoření gramatiky ze vstupních zpráv s nastavením vytváření vlastností pro celá slova. Další `CountVectorsFeaturizer` slouží k vytvoření ještě specifičtější gramatiky pro jednotlivá slova. Obsahuje nastavení pro vytváření vlastností pro části slov od jednoho až po čtyři písmena.

`DIETClassifier` slouží jako klasifikátor intentu a rozpoznávač entit. V rámci nastavení zahrnuje 100 epoch pro trénování a dále omezení podobnosti pro lepší obecnost modelu. Dále jsou definovány dvě vrstvy transformátoru s velikostí 256. Velikost vkládací vrstvy je 20. Důvěra modelu je nastavena na softmax, která zpracuje důvěru intentu tak, aby se dohromady rovnala jedné. Důvěra bude vždy mezi nulou a jedničkou.

`DateparserEntityExtractor` jedná se o upravenou vlastní komponentu[2], která slouží pro rozpoznání a zpracování datumu. Využívá knihovnu dateparser. Postupně jsou procházeny všechny zprávy, nad kterými je zavolána funkce `search_dates` s nastavením pro český jazyk a vyhledávání do budoucnosti. Tato funkce najde datum a zpracuje ho do specifického formátu. Následně jsou vyhledané entity přidány s maximální důvěrou do listu entit pro chatbota.

`EntitySynonymMapper` rozpoznané entity definované jako synonyma namapuje na stejnou hodnotu.

`FallbackClassifier` slouží pro klasifikaci intentu jako `nlu_fallback`, pokud předchozí klasifikátor s dostatečnou důvěrou. Nastavená hranice důvěry je 0,3 nebo rozdíl mezi prvními dvěma klasifikacemi musí být menší než 0,1.

Policy

Tato konfigurace slouží pro trénování vybírání odpovědí. Její implementace zahrnuje komponenty z jednotlivých kategorií. Skládá se z níže uvedených komponent s příslušným nastavením.

`MemoizationPolicy` slouží k vybírání odpovědi v případě, že současná konverzace odpovídá na trénovací konverzace. Uvedená komponenta je nastavena tak, že se dívá tří odpovědi a vstupní zprávy do historie konverzace.

`RulePolicy` vybere odpověď, pokud vstupní zpráva odpovídá některému definovanému pravidlu. S nastavením, že před trénováním je zkontrolováno, jestli si některé pravidla neprotiřečí a pravidla jsou omezena pouze na jedno kolo vstupu a odpovědí.

`UnexpecTEDIntentPolicy` zapříčiní, aby chatbot mohl reagovat na nepravděpodobné vstupní zprávy. Je nastaven se 100 epochy pro trénování a dívá se tří odpovědi a vstupní zprávy do historie konverzace. Dále je nastavena jedna vrstva transformátoru pro vstupní zprávu a jedna pro konverzaci s velikostí 128. Velikost vkládací vrstvy je 20.

`TEDPolicy` predikuje jaká další odpověď by měla nastat. Je nastaveno 100 epoch pro trénování a dívá se tří odpovědi a vstupní zprávy do historie konverzace. Dále je omezena podobnost pro lepší obecnost modelu. Poté je nastavena jedna vrstva transformátoru pro vstupní zprávu, jedna pro odpovědi a jedna pro konverzaci s velikostí 128. Velikost vkládací vrstvy je 20. Důvěra modelu je nastavena na `softmax`, který zpracuje důvěru odpovědí tak aby se dohromady rovnaly jedné. Jednotlivá důvěra tedy bude mezi nulou a jedničkou.

[2] https://rasahq.github.io/rasa-nlu-examples/docs/extractors/dateparser/

4.1.3 Doména

Doménový soubor slouží pro definici prostředí, ve kterém bude chatbot pracovat. Definuje intent, entity, sloty, odpovědi, akce a konfiguraci práce chatbota. V této sekci bych tedy chtěl popsat definici těchto parametrů a jejich účel.

Intent a entity

Definice intentu a entity v doméně slouží pro nastavení, která se budou používat z trénovacích dat. Dále zde přiblížím, jakým způsobem se budou využívat. Jsou definovány následující intenty:

- `search` pro vyhledávání,
- `search_type` intent pro specifikaci co je vyhledáváno,
- `date` pro zadávání datumu,
- `distance` pro zadávání vzdálenosti,
- `start_conversation` pro inicializaci konverzace,
- `location` pro zadávání současné pozice,
- `affirm` pro potvrzení, ve kterém je nastaveno, že nemůže obsahovat `datetime_reference` entitu,
- `deny` pro zamítnutí, ve kterém je nastaveno, že nemůže obsahovat `datetime_reference` entitu,
- `cityPart` pro zadávání městské části,
- `categories` pro zadávání kategorie.

Poté jsou k definovány entity, které se mohou vyskytnou ve větách podle trénovacích dat a to jsou:

- `place` pro specifikaci, že se vyhledává místo,
- `event` pro specifikaci, že se vyhledává akce,
- `distance` pro definici vzdálenosti,
- `lat` pro definici zeměpisné šířky,
- `lon` pro definici zeměpisné délky,
- `datetime_reference` pro definici datumu,
- `cityPart` pro definici městské části,
- `categories` pro definici kategorie.

Sloty

Sloty slouží k definici menších dat, která se budou v rámci konverzace ukládat. Využívané tedy jsou pro specifikaci filtrovacích parametrů, které používá backend. Kopírují entity, které jsou používány pro specifikaci dat a pomocí mapovací implementace z nich automaticky dostávají hodnoty. Všechny sloty jsou typu text a neovlivňují konverzaci. Specificky se jedná o sloty `distance`, `date`, `lat`, `lon`, `cityPart`, `categories`.

Odpovědi a akce

Odpovědi definují, co může chatbot využívat v konverzaci. Implementovány jsou dva typy odpovědí. První definují pouze text odpovědi. Druhý typ odpovědí navíc definují tlačítka pro nápovědu uživateli. Implementace odpovědi, která obsahuje oboje, vypadá následovně.

```
utter_search_question:
  - text: Co by jsi chtl vyhledat?
    buttons:
      - payload: /search_type{{"place":""}}
        title: msto
      - payload: /search_type{{"event":""}}
        title: akce
```

Tyto odpovědi jsou poté přidány do listu akcí. Zde jsou definovány všechny akce, které může chatbot vykonat. To také zahrnuje vlastní akce, pomocí kterých je komunikováno s externím serverem. Pro ně se, ale definuje pouze jejich název.

4.1.4 Trénovací data

Všechna trénovací data jsou definována jako vlastní. Nepoužívá se předem vytvořená sada dat. Všechna trénovací data se dají rozdělit do tří kategorií a to pro rozpoznávání, konverzace a pravidla. Trénovací data byla vytvořena pomocí interaktivního vývoje. To znamená komunikaci se základní definicí chatbota a přidávání dat podle toho, co se uživatel snaží napsat a co si představoval za odpověď. Z toho důvodu stačí mít definovaný doménový soubor.

Trénovací data pro rozpoznávání

Pro určení co by měl být chatbot schopný rozpoznávat, jsou definována trénovací data pro rozpoznávání. Zde jsou implementovány všechny intenty a většina entit z domény. Jediná entita, která není obsažena v trénovacích datech, je `datetime_reference`, protože je rozpoznávána pomocí vlastní komponenty.

Při výběru dat jsem se zaměřil hlavně na variabilitu dat potřebných pro daný intent nebo entitu, nikoliv na samotné množství dat. Pro český jazyk je důležité definovat data také ve variantě bez diakritiky a ve variantě s velkými písmeny pro názvy. Pro eliminaci pravopisných chyb pomáhá definovaná konfigurace pro trénování, kde se přidávají vlastnosti i pro části slova. Příklad kódu pro definici trénovacích dat vypadá následně:

```
- intent: search_type
  examples: |
    - [msta](place)
    - [mista](place)
```

```
    - [msto](place)
    - [misto](place)
    - [akce](event)
    - [akci](event)
```

V rámci zlepšení klasifikace intentu pro datum, je přidán regulární výraz, který odpovídá číslicovým datumům. Je uveden ve tvaru 12.2. pro den a měsíc. Není ale vytvořeno přímé napojení na intent nebo entity. To je vytvořeno pomocí již definovaných trénovacích dat, kde jsou vyhledána ta, která nejvíce sedí danému výrazu. Definice regulérního výrazu vypadá následovně:

```
- regex: date_regex
  examples: |
    - ^[0-9]{1,2}.[0-9]{1,2}$
```

Konverzace

Aby chatbot mohl odpovídat uživateli, je potřeba definovat trénovací konverzace. Proto je využita funkcionalita `checkpoint`. Ta nám umožňuje napojovat jednotlivé části, kdy filtrové moduly mají na začátku napojení na předchozí modul a na konci na následující modul.

Protože se jedná o dvě hlavní cesty pro konverzace, jsou implementovány dva soubory, jeden pro místa a druhý pro akce. Pro každý filtrovací modul jsou definovány dvě varianty. Jedna varianta pro případ, kdy uživatel chce použít tento filtr, a druhá pro případ, kdy ho využít nechce. Implementace modulu pro filtrování pomocí vzdálenosti vypadá následovně:

```
- story: search_event_filter_distance
  steps:
    - checkpoint: filter_event_distance
    - action: utter_ask_distance_search_event
    - intent: affirm
    - action: utter_ask_event_form_distance
    - intent: distance
      entities:
        - distance: 1km
    - slot_was_set:
        - distance: 1km
    - checkpoint: filter_event_date
```

Uživatel je prvně dotázán, jestli chce filtr použít a pokud odpoví kladně, je dotázán na specifikaci vzdálenosti. Tato hodnota je následně uložena do slotu a uživatel je přesměrován na další `checkpoint`.

Varianta při nepoužití pro stejný filtr vypadá následovně:

```
- story: search_event_filter_distance_deny
  steps:
    - checkpoint: filter_event_distance
    - action: utter_ask_distance_search_event
    - intent: deny
    - checkpoint: filter_event_date
```

Uživatel je dotázán na to, jestli chce použít filtr a při záporné odpovědi je rovnou přesměrována na další filtr.

Pravidla

Pro situace, kdy je od chatbota vždy požadována stejná odpověď, jsou definována pravidla. Z toho důvodu jsou použita na implementaci, aby chatbot inicializoval konverzaci s uživatelem. Jedná se o pravidlo, kdy je poslán startovní intent a je odpovězen s otázkou, která navazuje na definované trénovací konverzace. Implementace tohoto pravidla vypadá následovně:

```
- rule: Start conversation
  steps:
    - intent: start_conversation
    - action: utter_search_question
```

Další pravidlo které bylo definováno, je pro uložení současné polohy uživatele. Pravidlo v tomto případě ale nemá definovanou odpověď. Cílem je, aby byly pouze zpracovány entity do slotů a nebylo žádným způsobem komunikováno s uživatelem.

4.2 Implementace Android aplikace

Implementace android aplikace vychází z vytvořeného návrhu v sekci 3.4. V této sekci je popsána implementace pro navigaci mezi obrazovkami, systém implementace MVVM architektury, všechna navržená uživatelská rozhraní s navigací a nakonec ukládání malých dat.

4.2.1 Použité technologie

Aplikace byla implementována pomocí jazyka kotlin s využitím Java API framewok. Dále je použita řada základních knihoven pro možnost vývoje podle doporučených postupů. Jedná se o knihovny pro implementaci základních funkcionalit, uživatelského rozhraní `compose`[3], definování stylů `material design` a ukládání malých dat `SharedPreferences`. Pro komunikaci přes API rozhraní je použít `retrofit`. Mapa je implementována pomocí `google maps`. Pro DI je využit `koin`, pro načítání obrázků `coil` a pro řadu vylepšení `accompanist`.

4.2.2 MVVM architektura

Všechny mobilní obrazovky jsou implementovány na principu Model-view-viewmodel (MVVM) architektury. View je implementováno pomocí compose funkcí, viewmodel poté s lifecycle viewmodel a model je zastřešen za doménovou vrstvou. Důležité je dodržení způsobu komunikace mezi vrstvami. Uživatelské rozhraní komunikuje s viewmodel vrstvou, která získává data z modelové vrstvy. Nikdy ale spodnější vrstva nemá odkaz ne svého rodiče. Pro předávání informací se používá princip pozorovatel a vydavatel. Pro tento účel se používají `coroutines`, které jsou součástí jazyka kotlin. Díky této architektuře jsou jednotlivé části jasně odděleny. To pomáhá i testování, protože se testují menší části.

4.2.3 Navigace

Pro navigaci mezi jednotlivými obrazovkami se používá `compose` navigační komponenta. Ta využívá navigační ovladač, který spravuje celkový navigační graf. Přesměrovávání následně probíhá pomocí ovladače a definované cesty. Je potřeba ovladač předávat mezi jednotlivými

[3] https://developer.android.com/jetpack/compose

`compose` funkcemi. Protože ovladač obsahuje navigační graf, je možné jich mít několik pro jednotlivé funkcionality. Pro spodní navigační menu je vlastní ovladač a pro globální navigaci je jiný. Každý ovladač také musí mít nastavenou počáteční obrazovku, na kterou bude přesměrován.

4.2.4 Uživatelské rozhraní

V této sekci je popsána implementace pro všechny obrazovky v mobilní aplikaci. Všechny jsou implementovány pomocí technologie `compose`, která funguje na principu vytváření `compose` funkcí. Ty je možné vnořovat do sebe. Základem je, hlavní aktivita. Ta neobsahuje přímo grafické prvky, ale pouze odkaz na základní `compose` funkci. Zároveň je v hlavní aktivitě nastaven vzhled, který bude aplikace používat.

Chatbot

Na této obrazovce má uživatel možnost komunikovat s chatbotem. Jedná se o obrazovku, která se zobrazí jako první. Je součástí spodního navigačního menu. Implementovanou obrazovku je možné vidět na obrázku 4.1a.

Skládá se ze dvou hlavních částí. První je okno pro zobrazování zpráv, to je implementováno jako list pomocí komponenty `LazyColumn`. Ta umožňuje vytvářet nekonečný list, se kterým jde pohybovat. Aby poslední zpráva byla vždy na konci a posouval se list, je nastaveno obrácené zobrazení. To znamená, že list je vytvářen odspodu nahoru. Každá zpráva je poté dvou typů, a to jestli pochází od uživatele nebo chatbota. Podle typu se poté použije jiný styl. Zprávy od chatbota jsou zobrazeny jinou barvou a jsou umístěny vlevo. Součástí chatbot zpráv jsou ještě pomocná tlačítka, která se zobrazují vpravo s jinou barvou pro indikaci, že je možné na ní kliknout. Zprávy od uživatele mají světlejší barvu a jsou vpravo.

Druhá část obrazovky je vstupní lišta, která se skládá ze psacího pole a tlačítka pro odeslání zprávy. Při stisknutí na psací pole se zobrazí virtuální klávesnice a posune se celá obrazovka nahoru včetně spodního navigačního menu. Pro klávesnici je nastaven typ s potvrzovacím tlačítkem pomocí `KeyboardOptions`. Pro odeslání zprávy je možné použít tlačítko na klávesnici nebo na obrazovce. Pro psací pole je udržován stav textu pomocí funkce `remember { mutableStateOf("") }`, která drží hodnotu a při odeslání je poslána do `viewmodelu`.

Objevuj

Do sekce objevuj je možné se přesunout pomocí spodního navigačního menu. Obrazovka se skládá ze dvou horizontálních listů. První je list akcí a druhý list míst. Ten je implementován pomocí `LazyRow`. Jednotlivé položky jsou ze základu komponenta `Card`. Obrázek je zobrazen pomocí `AsyncImage`, který asynchronně zobrazí obrázek ze vstupní url. To znamená, že se položky zobrazí i bez obrázku a ten se postupně načte. Na položku je možné kliknout a pomocí navigačního ovladače je uživatel přesměrován na detail akce nebo místa. Pro zjednodušení navigace je zde předáván identifikátor položky, který poté využije detail. Definovaná cesta je implementována následovně `EventDetail.route + "/{id}"`. Z objevuj sekce je možné se ještě dostat na list všech akcí nebo míst a to přes klikatelný text v rohu listu. Kde hotovou implementaci je možné vidět na obrázku 4.1b.

Mapa

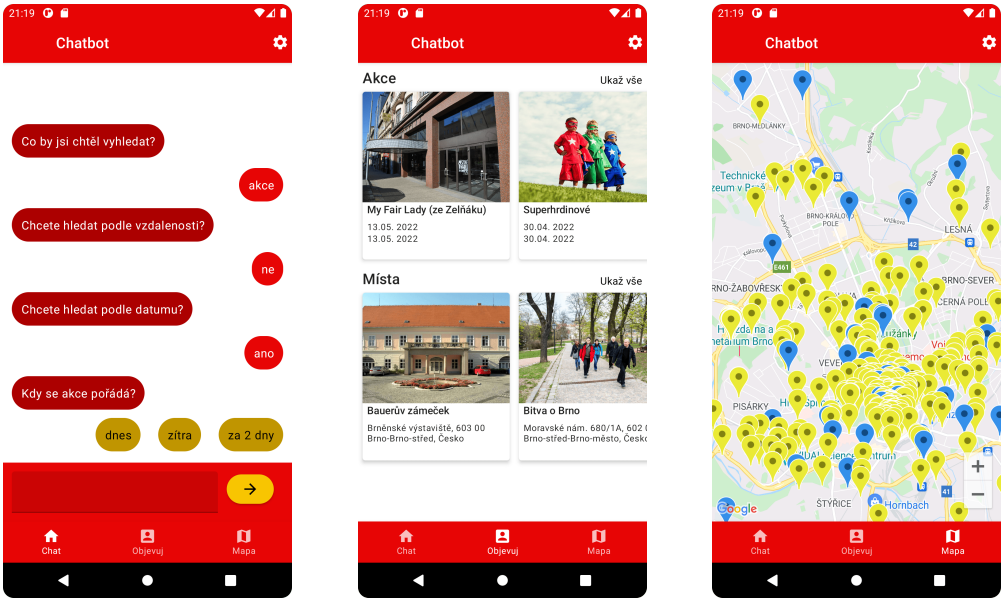
Poslední sekce, do které je možné se dostat skrze spodní navigační menu, je mapa. Ta je implementována pomocí `GoogleMap` komponenty. Kamera je nastavena, aby se zobrazila nad městem Brno pomocí `rememberCameraPositionState`, který obsahuje informace o pozici kamery. Pro všechny akce a místa je vytvořena ikona na mapě. Ty se liší v barvě a jejich implementace je pomocí `Marker` komponenty. Na jednotlivé ikony je možné kliknout a pomocí navigačního ovladače je zobrazena obrazovka detailu. Následnou podobu je možné vidět na obrázku 4.1c.

Detail místa a akce

Obrazovky pro detail místa a akce, jsou implementovány jako dvě komponenty. A to z důvodu, že se liší v informacích, které obsahují. Implementovaná podoba je vyobrazena na obrázku 4.1d. Princip implementace je ale stejný. Jedná se o obrazovku, která neobsahuje spodní navigační menu. V horním menu je nastavena ikona pro možnost vrátit se zpět. Přesměrováni probíhá pomocí uvolnění zásobníku obrazovek, přes navigační ovladač, kde implementace vypadá následně `navController.popBackStack()`. Pro zobrazování dat se využívají základní komponenty pro zobrazování textu a obrázku. Odlišné je zobrazení textu pro přesměrování na webovou stránku. Ten se vytvoří pomocí `InformationAnnotatedTextCol` komponenty, pro kterou je nastaven vlastní styl a url, kam má navigovat.

List míst a akcí

Uživatelské rozhraní pro list akcí a míst jsou stejné a liší se pouze v zobrazovaných datech. Jejich implementace je tedy téměř stejná. Položky jsou zde zobrazovány jako mřížkový list a to pomocí komponenty `LazyVerticalGrid`, kde je nastaveno, že se budou zobrazovat dvě položky vedle sebe. Jednotlivé položky jsou poté implementovány stejným principem jako v sekci objevuj. Horní navigační menu obsahuje ikonku pro navigaci zpět, podle stejného principu jako bylo vysvětleno v sekci pro detail místa a akce. Ukázku obrazovky je možné vidět na obrázku 4.1e.



(a) Sekce chatbot (b) Sekce objevuj (c) Sekce mapa

(d) Detail položky (e) List všech položek

Obrázek 4.1: Snímky obrazovky první verze android aplikace

4.2.5 Ukládání malých dat

Pro ukládání malých dat od uživatele je implementován repositář, který využívá `SharedPreferences`. Tímto způsobem jsou ukládány data o použitém stylu, jazyku a o současné poloze uživatele. Ukládání a čtení těchto dat je implementováno asynchronně pomocí `coroutines`. Jsou implementovány metody pro ukládání jednotlivých dat. Zápis probíhá nejprve editací uložených `SharedPreferences` a následnou změnou hodnoty `MutableStateFlow`, který drží hodnotu pro asynchronní komunikaci. Pro čtení dat je použit příslušný `StateFlow`, který je naslouchán.

4.3 Implementace Backendu

Backend implementace vychází z návrhu 3.5. V této sekci je popsána implementace API rozhraní, databáze a zpracování dat.

4.3.1 Použité technologie

Backend byl implementován v jazyce kotlin s použitím `ktor`[4] a `netty` frameworku. K němu bylo použito několik pomocných knihoven, které `ktor` nabízí. Mezi ně se řadí `jakcson` pro konvertování dat do a z JSON formátu a `exposed` pro práci s databází. Pro DI je použita knihovna `koin`.

4.3.2 API rozhraní

API rozhraní je implementováno jako sada rozšiřujících funkcí. Ty jsou rozděleny do tří modulů. Těmi jsou ovladače pro chatbota, akce a místa. Komunikace s ovladači probíhá skrze HTTP požadavky. V rozšiřujících funkcích jsou implementovány dílčí funkce, které zpracovávají specifické HTTP požadavky. V těchto funkcích jsou následně používány poskytovatelé, kteří zastřešují práci s databází. Ty jsou inicializovány pomocí DI.

Ovladače pro místa a akce implementují následující metody, které jsou všechny požadavky typu `GET`. Jedná se o získání všech položek, získání položky podle identifikátoru a stažení všech nových dat a následné uložení do databáze.

Ovladač pro chatbota implementuje jednu metodu a to pro požadavek typu `POST`, na který chatbot posílá požadavky na vykonání specifické akce. Vstupní data jsou zpracovány z formátu JSON do objektů. Následně se použije parametr `next_action` pro zjištění, která akce by měla proběhnout. Implementovány jsou dvě akce a to pro vyhledání akcí a míst. Vyhledané položky jsou následně konvertovány do formátu odpovědí a událostí, ve kterém jsou odeslány jako odpověď. Události jsou posílány pro smazání hodnot ze slotů.

4.3.3 Stahování a zpracování dat

Aplikace a chatbot pracují s daty, které jsou již zpracované a jsou hledány v databázi. Backend data stahuje z externích API rozhraní, následně je zpracovává a ukládá do databáze. Komunikace je implementována pomocí `ktor` klienta, který je zastřešen ve `HTTPClient`. Stahování probíhá na vrstvě `RemoteDataSource`. Ty jsou implementovány pro akce a místa. Společně s nimi je na stejném rozhraní implementován `LocalDataSource` pro komunikaci s databází. Díky tomu je možné mít přepínač mezi lokálními a externími data pro stejné metody. Pro získání nových dat jsou nejprve stažena data z externího zdroje, společně jsou staženy městské části. Následně jsou pomocí algoritmu, který je popsán v sekci 3.5.4, namapovány položky na jejich příslušnou městskou část. Zpracovaná data jsou poté poslána do lokální datové vrstvy na uložení do databáze.

4.3.4 Databáze

Jako databáze je použit PostgreSQL systém. Pro definici tabulek a práci s databází byla použita knihovna `exposed`, pomocí které je možné pracovat s databází v jazyce kotlin. Napojení na databázi a vytvoření tabulek je zastřešeno v `DatabaseManager`. Metoda pro vytvoření tabulek je implementována tak, že všechny smaže a následně je vytvoří. Knihovna

[4] https://ktor.io/

`exposed` funguje pomocí rozšiřujících funkcí, proto je potřeba se připojit na databázi jednou, a poté už je s databází možné pracovat globálně.

Práce s databází probíhá na `LocalDataSource` vrstvě. Pro ukládání dat do databáze je použita metoda `batchInsert`, která umožňuje hromadné uložení dat do databáze. Pro vyhledávání položek v databázi je prvně vytvořen základní dotaz pro všechny, poté je postupně kontrolováno, jestli jsou některé filtrovací parametry nastaveny. V případě že ano, jsou položky pomocí daných parametrů vyfiltrovány. Pro filtrování pomocí textových parametrů je text předělán do malého písma a následně je nahrazena všechna diakritika. Poté probíhá porovnání, jestli se filtrovací parametr rovná parametru položky. Pro filtrování podle datumu, je prvně filtrovací parametr konvertován do formátu timestamp. Data uložena v databázi jsou s tímto formátem uloženy. Datum položky je následně porovnáno, jestli spadá do vyhledávaného datumu. Pro filtrování pomocí vzdálenosti je porovnána vzdálenost pozice uživatele vůči poloze položky pomocí algoritmu definovaném v sekci 3.5.3. Výsledná vzdálenost je poté porovnána vůči požadované vzdálenosti z filtru. Po aplikaci všech filtrů je následně vrácen finální list položek.

Kapitola 5

Testování aplikace

V této kapitole je popsáno, jakým způsobem probíhalo testování s pomocí uživatelů. Hlavním cílem bylo zjistit, jak uživatelé hodnotí existující funkcionality, jejich dostupnost v rámci uživatelského rozhraní a možná vylepšení. Testování probíhalo na telefonech uživatelů. Pro instalaci aplikace byla vydána pro interní testování na Google Play, kde uživatelé měli možnost si ji stáhnout po potvrzení formuláře. Dohromady testování proběhlo s 10 uživateli, s částí uživatelů jsem byl osobně přítomen a část proběhla vzdáleně. Pro lepší analýzu zpětné vazby jsem vytvořil seznam stručných otázek, které jsem během testování pokládal uživatelům a zaznamenával jejich odpovědi. Zároveň bylo testování výhodné pro odzkoušení funkčnosti různých android zařízení. Výsledkem testování bylo nalezení chyb a nových úprav pro funkcionality, které byly následně zakomponovány do aplikace.

5.1 Průběh testování

Testování bylo rozděleno na dvě hlavní části. První část bylo samotné testování a zpětná vazba ke všem sekcím kromě chatbota. Druhá část byla zaměřena na chatbota a jeho funkcionality. Na začátku byl uživatelům v krátkosti vysvětlen pouze účel aplikace. Nejprve byly uživateli zadány úkoly, které měl zkusit udělat. To sloužilo pro zjištění intuitivnosti uživatelského rozhraní. Následně měli uživatelé možnost vysvětlit, co jim nebylo jasné nebo co by si představovali jinak.
První část se skládala z následujících úkolů:

- zobrazení sekce objevuj,
- přesměrování na detail akce a místa,
- přesměrování na list všech položek,
- zobrazení externí webové stránky s bližšími informacemi,
- zobrazení sekce mapa,
- přesměrování na detail ze sekce mapa.

Druhá část se skládala z úkolů komunikace s chatbotem a využití filtrů. Testovalo se tedy vyhledávání akcí a míst zvlášť. Úkoly vypadaly následovně:

- začít vyhledávání místa nebo akce,

- použít komunikaci pomocí tlačítek s nápovědou,
- použít komunikaci pomocí odeslání zprávy,
- použití jakýchkoliv nabízených filtrů,
- přesměrování na detail místa nebo akce.

Po dokončení zadaných úkolů byli uživatelé dotázáni na otázky k daným obrazovkám, které prošli. Pro získání zpětné vazby ohledně funkcionalit, uživatelé dostali návrhy různých změn a vylepšení, které hodnotili podle jejich přínosu. Nakonec měli uživatelé možnost navrhnou vlastní funkcionality, které by si představovali.

5.2 Zpětná vazba uživatelů

Od uživatelů bylo získáno velké množství zpětné vazby. Ta se převážně skládala z nových nebo upravených funkcionalit. A byly nalezeny chyby, které se vyskytovaly v aplikaci. Většina uživatelů hodnotila aplikaci pozitivně a byli by otevřeni zkusit využívat aplikaci dlouhodoběji.

Po zpracování všech zodpovězených otázek jsem vybral část odpovědí, které se objevovaly u většiny uživatelů. Pro vytvoření nové verze aplikace se zakomponovanou zpětnou vazbou by bylo nutné zpracovat následné úpravy:

- není jednoznačné, že je možné kliknout na text pro zobrazení všech položek v listu,
- v sekci mapa není možné rozeznat, co je akce a co místo,
- pro vyhledávání pomocí chatbota je potřeba změnit pořadí filtrů, podle větší priority,
- není jednoznačné, že je možné kliknout na vyhledané položky pro zobrazení detailu,
- pro filtrování pomocí datumu je potřeba používat rozmezí.

5.3 Realizace úprav

V této sekci je popsána implementace úprav, které byly definovány z výsledku testování. Úpravy jsou seskupeny do sekcí podle toho, k jaké části aplikace patří.

5.3.1 Sekce objevuj

V rámci sekce objevuj se jednalo pouze o úpravy pro uživatelské rozhraní. Bylo nutné změnit pouze mobilní aplikaci. Pro zlepšení přehlednosti, že je možné kliknout na text pro zobrazení všech položek, byl text předělán na tlačítko 5.1c.

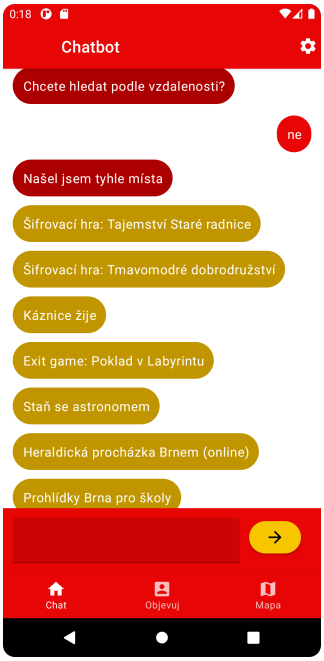
5.3.2 Sekce mapa

Úpravy pro sekci mapa byly pouze pro uživatelské rozhraní. Měnila se mobilní aplikace, aby bylo možné lepší rozpoznání mezi akcemi a místy na mapě byly přidány rozdílné ikony 5.1d.

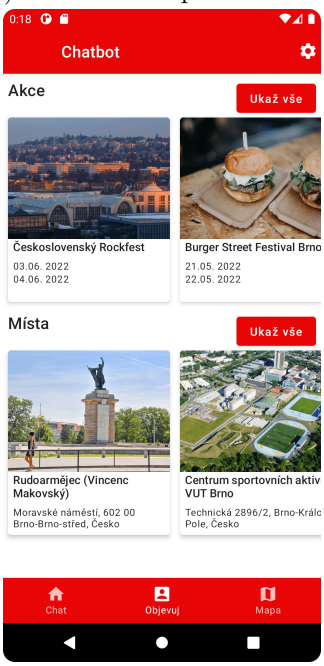
5.3.3 Sekce chatbot

Sekce chatbot vyžadovala změny na úrovni uživatelského rozhraní a funkcionality. Bylo potřeba změnit jak mobilní aplikaci, tak chatbota. Změnu pořadí bylo možné udělat pomocí přeskládání filtrovacích modulů. Podobně byla implementována změna pro filtrování pomocí datumu a to přidáním nových otázek a slotů pro zadání rozmezí 5.1a. Změny pro aplikaci zahrnovaly barvu vyhledaných položek a to konkrétně barvu pro tlačítka 5.1b.

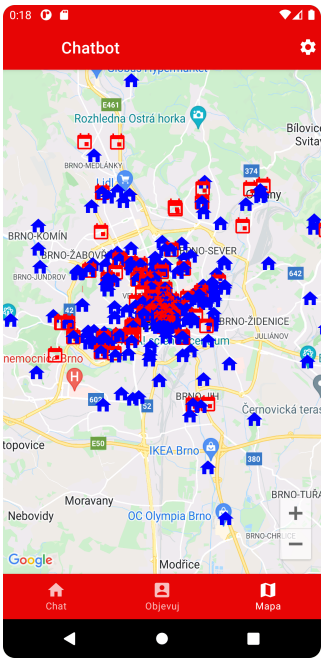
(a) Použití filtru podle datumu



(b) Sekce chatbot po vyhledání akcí



(c) Sekce objevuj s novým tlačítkem



(d) Sekce mapa s ikonami

Obrázek 5.1: Snímky obrazovky po zakomponování zpětné vazby

Kapitola 6

Závěr

Cílem práce bylo vytvořit mobilní android aplikaci, která umožní uživateli vyhledávat zajímavé akce a místa v rámci města Brna. Pro vytvoření aplikace bylo nastudováno několik architektur, které se používají pro vývoj chatbota. Pro jeho implementaci byl použit framework Rasa open source. V rámci zastřešení dat a funkcionality pro filtrování byl implementován backend, se kterým komunikuje aplikace a chatbot. Aplikace umožňuje uživatelům komunikovat s chatbotem pomocí textové konverzace, kde mohou používat řadu filtrů. Konkrétně se jedná o vzdálenost, městskou část, datum a kategorie. Dále má uživatel možnost si procházet všechny položky a zobrazit si jejich detailní informace. Nakonec je možné si zobrazit mapu všech akcí a míst. Aplikace byla otestovaná řadou uživatelů, kteří testovali jednotlivé části od uživatelského rozhraní po funkcionality, které aplikace a chatbot nabízí. Podle zpětné vazby byla vytvořena další verze aplikace s řadou změn a oprav.

K vývoji aplikace jsem se snažil použít nejnovější technologie, které prostor android aplikací nabízí. Jednalo se hlavně o nový způsob vývoje uživatelského rozhraní, využití asynchronní komunikace a použití stejného programovacího jazyka pro vývoj aplikace a backendu. To všechno znamenalo, že jsem někdy při vývoji narážel na sekce, které ještě nemají přímo definovaný standart. Zároveň jsem měl ale možnost získat řadu nových zkušeností, kterým směrem se posouvá vývoj android aplikací.

Aplikace mě zaujala a plánuji v budoucnosti pokračovat v jejím vývoji. Cílem by bylo následně vydání aplikace na Google Play. Díky testování s uživateli a jejich zpětné vazbě, se mi podařilo vytvořit plán dalších funkcionalit, které by mohla aplikace nabízet. Mezi ně patří přidání dalších typů položek, které by bylo možné vyhledávat. Nabízí se třeba restaurace, hospody nebo kavárny. Dále byl zájem o možnosti vytvoření osobního účtu, pomocí kterého by bylo možné přidat řadu funkcionalit jako oblíbená místa a kategorie. Dále by se nabízela možnost vytvářet vlastní akce a pozvat jiné uživatele. Všechny tyto funkcionality by bylo možné zakomponovat do chatbota. Ten by mohl podporovat vyhledávání pomocí celého textu k zrychlení konverzace pro uživatele, kteří jsou již v ovládání chatbota zdatní.

Literatura

[1] *Calculate distance, bearing and more between Latitude/Longitude points* [online]. [cit. 2022-04-20]. Dostupné z: http://www.movable-type.co.uk/scripts/latlong.html.

[2] *THE RASA MASTERCLASS HANDBOOK* [online]. 2020. Dostupné z: https://cdn2.hubspot.net/hubfs/6711345/ebook-v3.pdf?utm_referrer=https%3A%2F%2Finfo.rasa.com%2Fmasterclass-ebook.

[3] Bocklisch, T., Faulkner, J., Pawlowski, N. a Nichol, A. *Rasa: Open Source Language Understanding and Dialogue Management*. arXiv, 2017. DOI: 10.48550/ARXIV.1712.05181. Dostupné z: https://arxiv.org/abs/1712.05181.

[4] Bunk, T., Varshneya, D., Vlasov, V. a Nichol, A. *DIET: Lightweight Language Understanding for Dialogue Systems*. arXiv, 2020. DOI: 10.48550/ARXIV.2004.09936. Dostupné z: https://arxiv.org/abs/2004.09936.

[5] Cahn, J. *CHATBOT: Architecture, Design, Development*. 2017.

[6] Devlin, J., Chang, M.-W., Lee, K. a Toutanova, K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv, 2018. DOI: 10.48550/ARXIV.1810.04805. Dostupné z: https://arxiv.org/abs/1810.04805.

[7] Franklin, W. R. *PNPOLY - Point Inclusion in Polygon Test* [online]. [cit. 2022-04-20]. Dostupné z: https://wrf.ecse.rpi.edu/Research/Short_Notes/pnpoly.html.

[8] Jiao, A. An Intelligent Chatbot System Based on Entity Extraction Using RASA NLU and Neural Network. *Journal of Physics: Conference Series*. IOP Publishing. mar 2020, sv. 1487, č. 1, s. 012014. DOI: 10.1088/1742-6596/1487/1/012014. Dostupné z: https://doi.org/10.1088/1742-6596/1487/1/012014.

[9] Kong, X. *Conversational AI with Rasa*. 2021.

[10] Turing, A. Computing Machinery and Intelligence. *Mind*. Leden 1950, LIX, s. 433–460.

[11] Vlasov, V., Mosig, J. E. M. a Nichol, A. *Dialogue Transformers*. arXiv, 2019. DOI: 10.48550/ARXIV.1910.00486. Dostupné z: https://arxiv.org/abs/1910.00486.

[12] Warmerdam, V. *DIET architecture interactive demo* [online]. 2020. Dostupné z: http://bl.ocks.org/koaning/raw/f40ca790612a03067caca2bde81e7aaf/.

[13] Zemčík, T. *A Brief History of Chatbots*. 2019.