Obsah

Úvod **1**

1 Pôvodná verzia aplikácie Sprievodca FMFI **3**
1.1 Sprievodca FMFI z hľadiska používateľa 3
1.2 Sprievodca FMFI z technického hľadiska 7
1.2.1 Androidová časť aplikácie Sprievodca FMFI 8
1.2.2 Serverová časť aplikácie Sprievodca FMFI 11
1.3 Súvisiace bakalárske práce 13

2 Konverzačné rozhrania **15**
2.1 Základné požiadavky 15
2.2 Doména konverzačných rozhraní 15
2.3 Task-oriented dialogue systems 16
2.4 Chatbot dialogue systems 16
2.5 GUS 17
2.5.1 Rozoznávanie pomenovaných entít(NER) 17
2.5.2 Zisťovanie používateľského zámeru 18
2.5.3 Napĺňanie slotov 18
2.5.4 Vykonanie akcie 19
2.6 Základné rozdelenie používateľského zámeru 19

3 Vytvorené konverzačné rozhranie z používateľského hľadiska **23**

4 Vytvorené konverzačné rozhranie z technického hľadiska **31**
4.1 Manažovanie dialógu 32
4.2 Rozoznávanie pomenovaných entít 33
4.3 Rozoznávanie používateľovho zámeru 40
4.4 Napĺňanie slotov 43

4.5 Vykonanie akcie zodpovedajúcej používateľovmu zámeru . . . 48
4.6 Odpovedanie na otázky bez používateľského zámeru 49
4.7 Grafické používateľské rozhranie 50

5 Serverová časť z technického hľadiska 52
5.1 Ukladanie konverzácii 52
5.2 Odpovedanie na otázky 54
5.3 Trieda *WebServiceFacadeV1* 61

Záver 63

Úvod

Sprievodca FMFI je mobilná aplikácia, ktorá poskytuje interaktívnu mapu a zísakvanie infromácii o osobách a miestnostiach nachádzajúcich sa na fakulte matematiky fyziky a informatiky. Aplikácia Sprievodca FMFI uľahčuje mnohým študentom orientáciu v rámci fakulty matematiky fyziky a informatiky. Táto aplikácia poskytuje používateľovi možnosť vyhľadávať cesty medzi dvomi miestnosťami ale aj možnosť zobrazenia miestnosti na mape.

V niektorých prípadoch môže byť používateľovi jednoduchšie sa spýtať otázku ako sa v aplikácii preklikávať. Napríklad ak používateľ hladá cestu z A do B je jednoduchsie napisať path from A to B. Vzhľadom na rastúcu popularitu konverzačných rozhraní v komerčných aplikáciach sme sa rozhodli že konverzačné rozhranie je dobrým riešením spomenutého problému. Taktiež pre niektoré typy informácii v aplikácii nie je priestor ako napríklad koľko kreditov je potrebné za rok získať. Konverzačné rozhranie by malo byť schopné zodpovedať aj takéto otazky. Konverzačné rozhranie zároveň zapadá do kontextu samotnej aplikácie Sprievodca FMFI nakoľko predstavuje jednu osobu, ktorá používateľa sprevádza po fakulte.

V mojej práci sa venujem konverzačnému rozhraniu, ktoré bude súčasťou aplikácie Sprievodca FMFI. Konverzačné rozhranie voči používateľovi vystupuje pod menom Mia. Mia bude schopná reagovať na vety v anglickom jazyku, aby s ňou mohli komunikovať aj zahraniční študenti. Mia má dva využitia, a to odpovedanie na používateľove otázky týkajúce sa fakulty matematiky, fyziky a infromatiky a taktiež aj ovládanie už existujúcich funkcionalít. Odpovede na otázky sa Mia pokúsi čerpať z vlastnej databázy a v prípade, že na otázku nevie odpovedať sa Mia obráti na server, ktorý jej odpovie na základe informácii dostupných na stránke fakulty. Medzi funkcionality, ktoré bude používateľ schopný pomocou Mii ovládať patrí hľadanie

miestností a osôb na mape, zísakvanie informácii k osobám a miestnostiam taktiež vyhľadávanie cesty na mape, zobrazenie denného menu a zobrazenie rozvrhov pre konkrétne osoby a miestnosti. Mia je schopná rozpoznávať čo používateľ píše a taktiež aj čo danou vetou používateľ myslel. Na základe používateľovho zámeru Mia zistí potrebné informácie a používateľov zámer vykoná.

V prvej kapitole sa budeme venovať pôvodnej verzii aplikácie Sprievodca FMFI. Rozoberieme ju z používateľského hľadicka a potom aj z hľadiska technického.

V druhej kapitole sa budeme detailne venovať doméne konverzačných rozhraní. Rozoberieme rozdelenie konverzačných rozhraní, pozrieme sa na konkrétnu architektúru konverzačného rozhrania ako aj na jednotlivé podproblémy spojené s touto architektúrov a ich možné riešenia.

V tretej kapitole sa pozrieme na to ako vyzerá z používateľského hľadiska konverzačné rozhranie, ktoré som do aplikácie Sprievodca FMFI pridal. Taktiež si ukážeme jednotlivé prípady, ktoré počas používania konverzačného rozhrania môžu nastať.

V štvrtej kapitole si detailne rozoberieme technickú stránku konverzačného rozhrania. Ukážeme si ako konverzačné rozhranie spracováva používateľov vstup, extrahuje z neho potrebné informácie a potom vykonáva používateľov zámer.

V piatej kapitole sa budeme venovať mojim zmenám, ktoré som spravil na serverovej aplikácii. Popíšeme si ako sa konverzácie ukladajú, ako sa zbierajú informácie a ako sa v nich následne vyhľadáva odpoveď na kladenú otázku.

Kapitola 1

Pôvodná verzia aplikácie Sprievodca FMFI

1.1 Sprievodca FMFI z hľadiska používateľa

Sprievodca FMFI je mobilná aplikácia dostupná pre operačný systém android s verziou aspoň 4.0.3 (IceCreamSandwich). Sprievodca FMFI momentálne poskytuje funcionalitu týkajúcu sa Fakulty matematiky, fyziky a informatiky Univerzity Komenského v Bratislave. V nasledujúcej časti si prejdeme spomínanú funkcionalitu, stručne ju zadefinujeme a zároveň si ju názorne ukážeme na bežiacej aplikácii.

Funkcionalita Sprievodcu FMFI

Po otvorení aplikácie sa nám zobrazí hlavné menu(obrázok 1.1), v ktorom si môžeme vybrať z viacerých funkcionalít.
Hlavné menu obsahuje 8 možností:

- **Mapa**
  Umožňuje používateľovi pozerať si mapu fakulty.
- **Dnešné denné menu**
  Používateľ tu nájde menu školskej jedálne.
- **Vyhľadaj trasu**
  Používateľ si môže zadať miesto odkiaľ ide a kam sa chce dostať a aplikácia mu na mape vykreslí najkratšiu možnú cestu.

- **Hľadaj**
  Táto funkcionalita poskytuje používateľovi možnosť prehľadávať databázu miestností a zamestnancov, prípadne zobrazenie detailov o zvolenej entite.

- **Aktualizuj dáta**
  Po zvolení tejto možnosti sa spustí aktualizácia dát zo vzdialeného servera.

- **Nahlás chybu**
  Poskytuje možnosť nahlásenia chyby počas behu aplikácie.

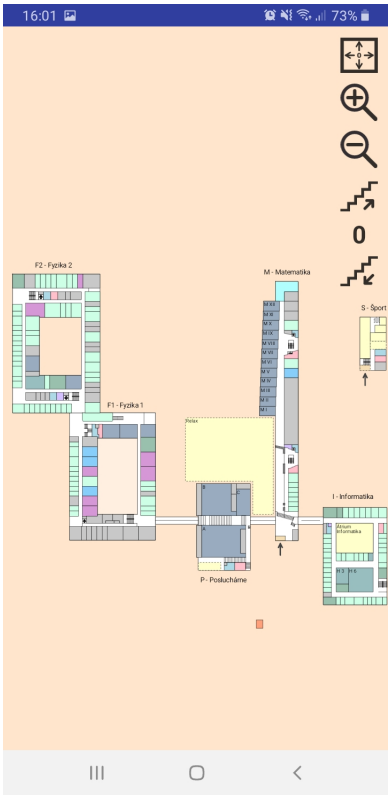
- **Nastavenia**
  Používateľ si môže v nastaveniach zvoliť či chce aby si aplikácia automaticky aktualizovala pri štarte dáta. Ďalším nastavením je či používateľ chce používať výťahy.

- **O aplikácii**
  Táto možnosť poskytuje základné informácie o aplikácii Sprevodca FMFI rovnako ako zoznam osôb, ktoré prispeli pri vývoji tejto aplikácie. Taktiež je tu možnosť sa do vývoja zapojiť v sekcii Zapoj sa.

Obr. 1.1: Hlavné menu Sprievodcu FMFI[7]

Mapa fakulty

Jednou z hlavných funkcionalít pôvodnej aplikácie Sprievodca FMFI je zobrazenie mapy fakulty (obrázok 1.2). Na mape sa používateľovi zobrazujú jednotlivé miestnosti ako aj automaty na jedlo, schodiská a výťahy.

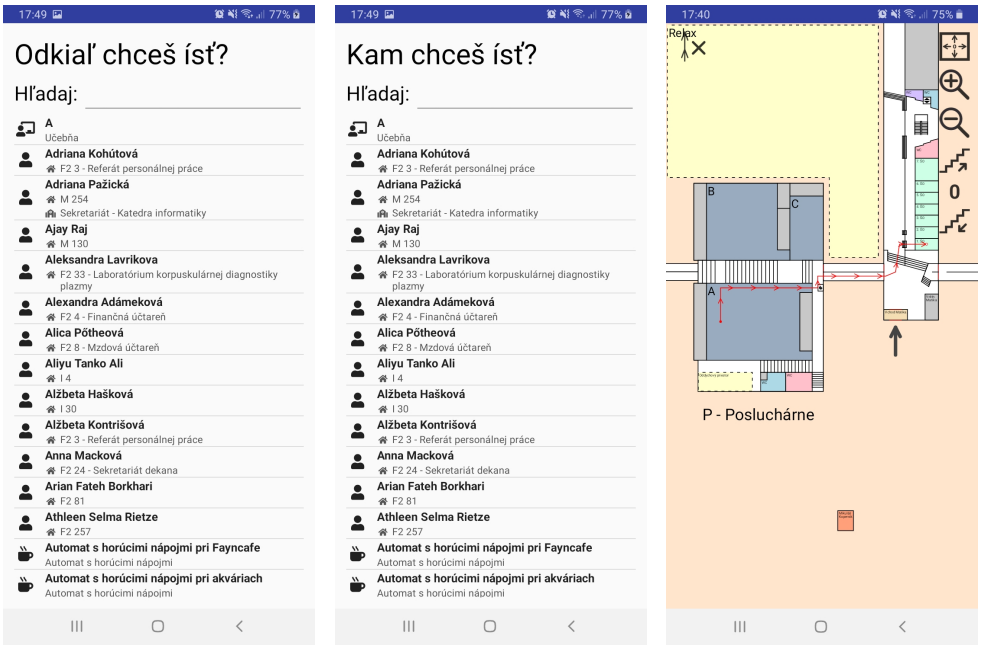


Obr. 1.2: Interaktívna mapa aplikácie Sprievodca FMFI

Na pravej strane mapy sa nachádza panel zložený z dvoch častí. Prvá časťobsahuje tri tlačidlá prvé slúži na resetovanie priblíženia, druhé tlačidlo slúži na približovanie a tretie na vzďalovanie sa v mape. Druhá časť sa skladá z tlačidla na zvýšenie poschodia, z čísla poschodia, ktoré sa aktuálne zobrazuje a z tlačidla na zníženie poschodia.

Vyhľadaj trasu

Ďaľšou dôležitou funkcionalitou, ktorú aplikácia Sprievodca FMFI poskytuje je zobrazenie najkratšej možnej cesty na mape medzi dvomi miestnosťami (obrázok 1.4). Používateľ si môže vybrať odkiaľ a kam chce ísť. Po tomto výbere sa používateľovi vykreslí samotná cesta.



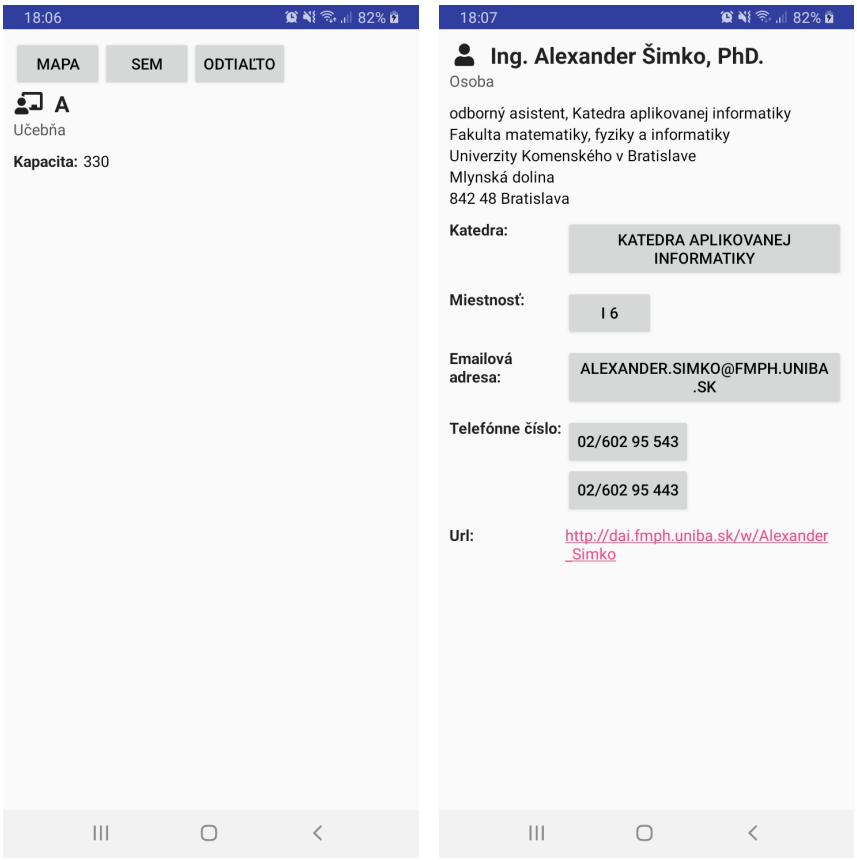
Obr. 1.3: Zobrazenue cesty z A do M I

Na obrázku 1.4 môžme vidieť, že cesta medzi prednáškovou miestnosťou A a kanceláriou M I sa zobrazuje na mape, ktorá bola spomínaná vyššie. Cesta je na mape vyznačená červenou čiarou. V ľavom hornom rohu sa nachádza tlačidlo, ktorým je možne zobrazovanú trasu zrušiť a ostane nám zobrazená iba mapa.

Zobrazenie detailov

Po kliknutí na tlačidlo hľadaj si môže používateľ vybrať miestnosť alebo osobu o ktorej si chce získať doplňujúce informácie. Keď si používateľ vyberie a klikne na niektorú z osôb alebo miestností zobrazia sa mu detaily o

kliknutej miestnosti alebo osobe.



Obr. 1.4: Zobrazenue detailu pre miestnosť

Zobrazenie detailov miestností je možné docieliť aj kliknutím na konkrétnu miestnosť na mape.

1.2 Sprievodca FMFI z technického hľadiska

V tejto sekcii sa budeme venovať pôvodnej verzii aplikácie Sprievodca FMFI z technického hľadiska. Aplikácia Sprievodca FMFI má dve časti a to androidovú a serverovú časť.

1.2.1 Androidová časť aplikácie Sprievodca FMFI

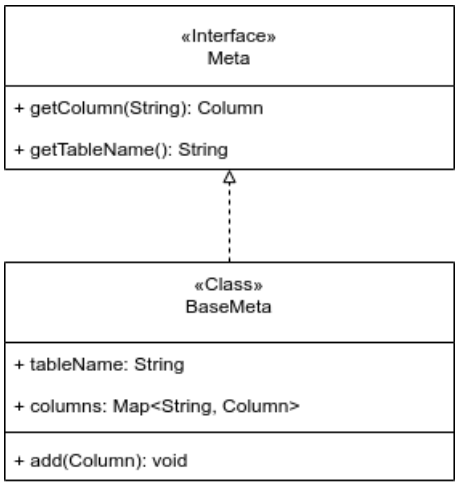
V tejto časti sa pozrieme na andoidovú aplikáciu Sprievodca FMFI z technického hľadicka. Pozrieme sa na rozdelenie kódu, prácu s databázou a používateľské rozhranie.

Rozdelenie kódu

Kód aplikácie sa nachádza v balíčku *sk.uniba.fmph.guide*. Triedy spojené s používateľským rozhraním sa nachádzajú v balíčku *ui*. Triedy určené na prácu s databázou sa nachádzajú v balíčku *db* a triedy reprezentujúce samotné databázové entity sa nachádzajú v balíčku *model*. Aplikácia taktiež obsahuje priečinok *res*, v ktorom sa nachcádzajú balíčky s xml dokumentami, podľa ktorých sa vykreslujú jednotlivé komponenty grafického používateľského rozhrania.

Práca s databázou

Každá tabuľka má svoju meta triedu, ktorá sa nachádza v balíčku *db.mappers.metas*. Každá meta trieda implementuje rozhranie *Meta*. Kvôli minimalizovaniu duplicít sa v balíčku *db.mappers.metas* nachádza aj abstraktná trieda *BaseMeta*(obrázok 1.5), z ktorej dedia jednotlivé meta triedy.

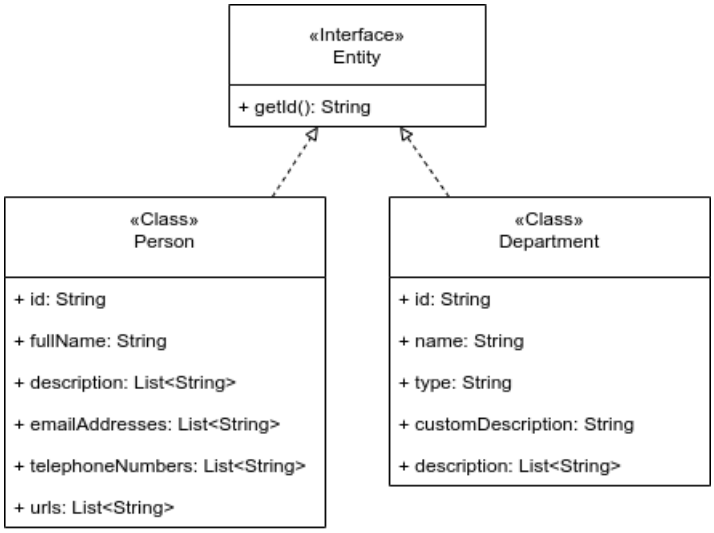


Obr. 1.5: Triedny diagram pre rozhranie *Meta*

V konštruktore si každá meta trieda pomocou metódy *add* pridá stĺpce, ktoré sa nachádzajú v tabuľke, ktorú táto meta trieda reprezentuje. Tieto stĺpce sa pridávajú do členskej premennej *columns*, ktorá sa nachádza v nadtriede *BaseMeta*.

Na čítanie údajov z databázy slúžia triedy, ktoré implementujú rozhranie *Reader* nachádzajúce sa v balíčku *db.mappers.readers*. Každá tabuľka, z ktorej chceme získavať údaje má triedu triedu implementujúcu rozhranie *Reader*, ktorá slúži na čítanie údajov z danej tabuľky. Rovnako ako pri meta triedach aj tu je na predídenie duplicitnému kódu abstraktná trieda *BaseReader*, z ktorej dedia ostatné čítače pre jednotlivé tabuľky. Na manažovanie práce s databázou slúži trieda *EntityManager*, ktorá má pre každú metódu v čítačoch samostatnú metódu. Táto metóda na pozadí volá metódu samotného čítača.

Triedy reprezentujúce databázové entity sa nachádzajú v balíčku *model* a implementujú rozhranie *Entity* (obrázok 1.6). Jednotlivé triedy obsahujú členské premenné ekvivalentné stĺpcom v tabuľke, ktorú daná trieda reprezentuje.



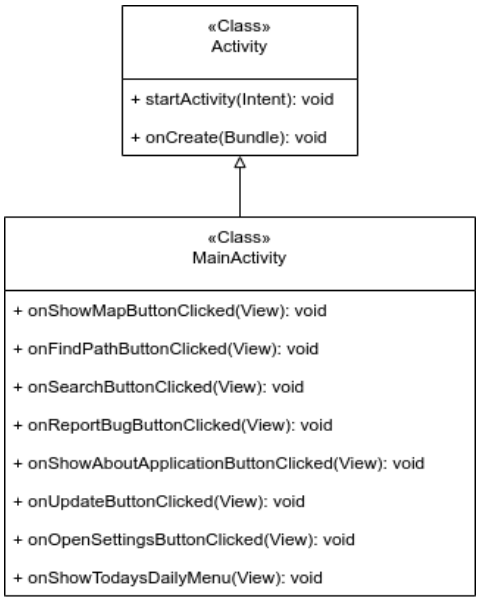
Obr. 1.6: Triedny diagram pre rozhranie *Entity*

Na obrázku 1.6 je zobrazené rozhranie *Entity*, ktoré má jedinú metódu *getId* a príklad dvoch tried, ktoré toto rozhranie implementujú. Tieto triedy predstavujú databázové entity jedna pre databázu osôb (trieda *Person*) a jedna pre databázu oddelení (trieda *Department*).

Používateľské rozhranie

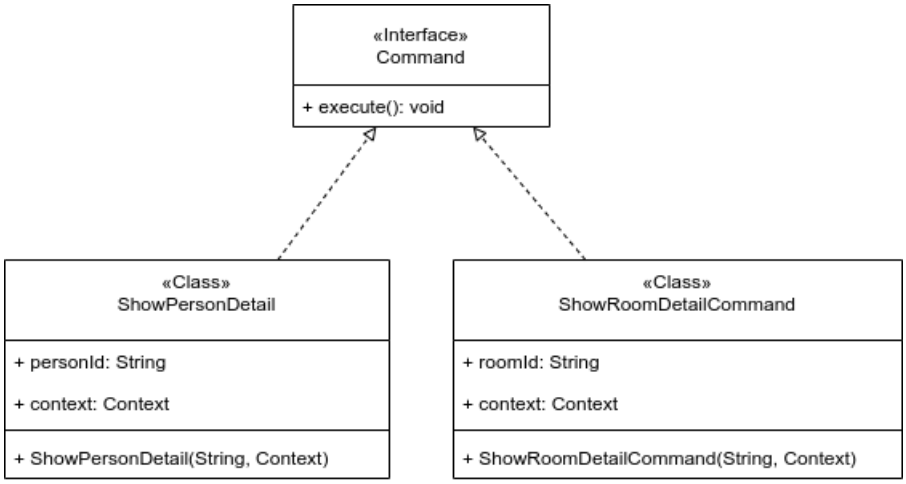
Triedy určené na komunikáciu s používateľom sa nachádzajú v balíčku *ui*.

Jednotlivé aktivity, ktoré sa používateľovi zobrazujú dedia z triedy *Activity* a nachádzajú sa v balíčku *activities*. Po spustení aplikácie Sprievodca FMFI sa vytvorí hlavná aktivita, ktorá je inštanciou triedy *MainActivity* (obrázok 1.7). Táto aktivita v metóde *onCreate* vykreslí pomocou xml dokumentu menu aplikácie s tlačidlami pre jednotlivé aktivity. Každé tlačidlo má svoju metódu, ktorá sa po stlačení vykoná a začne novú aktivitu. Na spustenie novej aktivity slúži metóda *startActivity*, ktorá je implementovaná v triede *Activity*. Táto metóda na vstupe dostane inštanciu triedy *Intent*. Na vytvorenie novej inštancie triedy *Intent* je potrebné mať kontext a aktivitu, ktorú chceme vykonávať. Ako kontext poskytujeme aktivitu, ktorá sa aktuálne vykonáva (premenná *this*).



Obr. 1.7: Triedny diagram pre triedu *Activity*

Na spúšťanie aktivít s konkrétnymi parametrami slúžia triedy implementujúce rozhranie *Command* (obrázok 1.8), ktoré sa nachádzajú v balíčku *commands*. Tieto triedy v konštruktore na vstupe dostanú parametre, ktoré sú potrebné na spustenie konkrétnej aktivity a kontext, v ktorom túto aktivitu majú spustiť. Rozhranie Command má metódu execute, ktorá spustí danú aktivitu.



Obr. 1.8: Triedny diagram pre rozhranie *Command*

Na obrázku je ukázané rozhranie *Command* a príklad dvoch tried, ktoré toto rozhranie implementujú. Trieda *ShowPersonDetail* po zavolaní metódy *execute* vytvorí novú inštanciu triedy *Intent*, ktorá ako kontext dostane členskú premennú *context* a aktivita, ktorá sa má spustiť je *PersonDetailActivity*. Pomocou metódy *putExtra*, ktorá sa nachádza v triede *Intent* sa aktivite nastaví id používateľa, ktorého informácie sa majú zobraziť. Obdobne funguje aj trieda *ShowRoomDetailCommand* až na to že namiesto aktivity *PersonDetailActivity* sa vykoná aktivita *RoomDetailActivity*.

1.2.2 Serverová časť aplikácie Sprievodca FMFI

V tejto sekcii sa budem venovať technickej stránke serverovej časti Sprievodcu FMFI. Postupne si prejdeme tri časti a to rozdelenie kódu, reagovanie na http požiadavky a prácu s databázou.

Rozdelenie kódu

Samotná aplikácia sa nachádza v balíčku *sk.uniba.fmph.guide.server*. Triedy na získavanie aktuálnych informácii z určitých webstránok sa nachádzajú v balíčku *crawlers*. Triedy slúžiace na prístup do databázy sa nachádzajú v balíčku *dao* a triedy predstavujúce databázové entity sa nachádzajú v balíčku *db*. Balíček *ws* obsahuje triedy slúžiace na spracovanie a odpovedanie na http požiadavky.

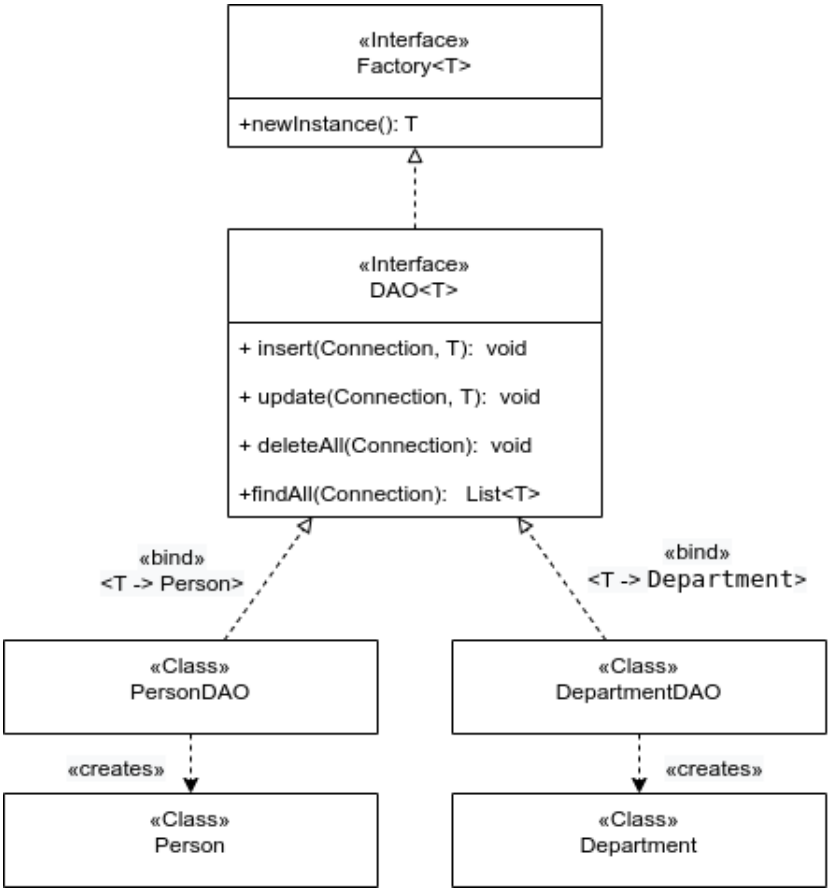
Reagovanie na požiadavky

Reagovanie na http požiadavky manažuje trieda *WebServiceFacadeV1*. Táto trieda obsahuje metódy, ktoré spracuvávajú a odpovedajú na požiadavky na jednotlivých cestách v rámci webstránky. Pomocou dekorátoru @Path sa metóde určí na akú cestu v rámci stránky má reagovať a pomocou dekorátora @POST alebo @GET sa metóde povie, na akú metódu http požiadavky má reagovať.

Prístup k databáze

Na pristupovanie k databáze serverová aplikácia využíva prístup Data Access Object(DAO)[8]. Pre každú tabuľku existuje samostatná dao trieda nachádzajúca sa v balíčku *dao*, ktorá implementuje rozhranie *DAO* (obrázok 1.9). V balíčku *db* sa pre každú tabuľku nachádza trieda reprezentujúca databázovú entitu pre danú tabuľku. Členské premenné týchto tried zodpovedajú jednotlivým stĺpcom v danej tabuľke.

Rozhranie *DAO* má metódu *insert*, ktorá vstupnú entitu vloží do databázy. Metóda *update* aktualizuje databázovú entitu podľa entity, ktorú dostane na vstupe. Metóda *deleteAll* vymaže všetky databázové entity nachádzajúce sa v danej tabuľke a metóda *findAll* vráti zoznam databázových entít nachádzajúcich sa v danej tabuľke. Rozhranie *DAO* implementuje rozhranie *Factory*, ktoré obsahuje metódu *newInstance*, ktorá vráti novú inštanciu triedy *T*. Pomocou tejto metódy jednotlivé triedy implementujúce rozhranie *DAO* vytvárajú nové inštancie entít viazaných na danú dao triedu.



Obr. 1.9: Triedny diagram pre rozhranie *DAO*

1.3 Súvisiace bakalárske práce

V tejto časti si prejdeme historický vývoj aplikácie Sprievodca FMFI.

Martin Bohumel - Mobilný sprievodca FMFI(Bakalárska práca)

V tejto práci bola vytvorená aplikácia pre mobilné zariadenie, ktorá poskytovala informácie o FMFI, možnosť prehliadať si interaktívnu mapu interiéru fakulty a možnosť vyhľadávať trasu medzi dvoma bodmi a následne ju na mape vykresliť [1].

Táto práca slúži ako základ pre súčasnú verziu aplikácie Sprievodca FMFI. V tejto verzii aplikácie boli všetky dáta napevno uložené v mobilnej aplikácii a nebola možnosť aktualizácie údajov.

Linda Jurkasová - Sprievodca FMFI(Bakalárska práca)

Táto práca vychádzala z práce Martina Bohumela. V tejto práci sa údaje presunuli z mobilnej aplikácie na server a aplikácia si od serveru získava zmeny oproti verzii dát, ktoré má aktuálne k dispozícii. Ďalej sa v tejto práci pridala podpora pre ďalšie objekty, s ktorými vie aplikácia pracovať, možnosť využívania výťahov a možnosť prezerania si denného menu[2] .

Janka Boborová - Udalosti v Sprievodcovi FMFI(Bakalárska práca)

Táto bakalárska práca sa vyvíjala paralelne s mojou prácou. Táto práca poskytuje možnosť sťahovať si údaje o prednáškach, cvičeniach, predmetoch a štúdijných krúžkoch a pristupovať k týmto údajom v mobilnej aplikáci. V detaile o miestnostiach sa zobrazuje rozvrh pre danú miestnosť a v detaile o osobe sa zobrazí rozvrh pre danú osobu. Nová verzia aplikácie taktiež poskytuje možnosť pridávať udalosti ako napríklad prednášky alebo cvičenia.

Zhrnutie

Práca, na ktorej pracujem vychádza z práce Lindy Jurkasovej a využíva aj funkcionalitu paralelne sa vyvíjajúcej práce Janky Boborovej.

Kapitola 2

Konverzačné rozhrania

Konverzačné rozhranie je časť aplikácie, ktorá umožňuje používateľovi komunikovať so systémom pomocou prirodzeného jazyka a rovnako tak dostávať odpovede v prirodzenom jazyku.

2.1 Základné požiadavky

Konverzačné rozhranie by malo obsahovať priestor, do ktorého používateľ môže písať svoje otázky. Ďalej by malo byť konverzačné rozhranie schopné porozumieť prirodzenému jazyku ľudí a vedieť ho preložiť do svojej "strojovej"reči. Taktiež by malo byť schopné klásť otázky v prípade že nejaká informácia chýba. Na záver by malo byť konverzačné rozhranie schopné poskytovať odpoveď v prirodzenom jazyku a to ako úplnú vetu.

2.2 Doména konverzačných rozhraní

Konverzačné rozhrania sa delia podľa svojho účelu na dve veľké skupiny:

- **Dialógové systémy orientované na úlohy(Task-oriented dialogue systems)**
  Využívajú konverzáciu s užívateľom za účelom vykonať špecifickú úlohu.
- **Dialógové systémy orientované na konverzáciu(Chatbot dialogue systems)**
  Systémy navrhnuté na udržiavanie konverzácie s používateľom.

2.3 Task-oriented dialogue systems

V tejto časti stručne zadefinujem problematiku Task-oriented dialógových systémov. Základnou architektúrov pre tieto systémy je Geniálny Porozumievací Systém(GUS)[4]. Táto architektúra využíva ideu dialógových systémov využívajúcich rámce . Účelom týchto systémov je získať vstup od používateľa, spracovať ho, identifikovať aký má používateľ zámer, získať informácie potrebné na uskutočnenie tohto zámeru a následne vykonať akciu príslušnú danému používateľskému zámeru. Pri architektúre GUS sú dôležité pojmy:

- **stav dialógu(dialogue state)**, ktorý hovorí o stave dialógu, v ktorom sa momentálne konverzačný agent nachádza.
- **rámec(frame)**, ktorý predstavuje používateľský zámer a obsahuje zoznam informácii (slotov), dôležitých na vykonanie tohto zámeru.
- **slot**, ktorý predstavuje jednu konkrétnu informáciu vo frame.

2.4 Chatbot dialogue systems

V tejto časti stručne zadefinujem problematiku Chatbot dialógových systémov. Tieto systémy sú najjednoduchšie v kategórii dialógových systémov. Existujú dva základné prístupy k vytváraniu chatbota a to **prístup založený na pravidlách** (Ruled-based) a **prístup založený na súbore spisov**(Corpus-based)[3].

Ruled-based

Rule-based je jeden z prvých prístupov použitých na problematiku Chatbot dialogue systémov. Základnou ideou je, že systém má sadu pravidiel a na základe nich vytvára odpovede z používateľových otázok. Jedným z najznámejších chatbotov bola ELIZA a chatbot PARRY[3].

Corpus-based

Corpus-based prístup je o niečo sofistikovanejší ako spomínaný rule-based.

Tento prístup sa spolieha na kvantum dialógov v prirodzenom jazyku, ktoré má k dispozícii. Tieto dialógy sa nazývajú korpus. Po tom ako používateľ zadá vstup, tento systém v svojom korpuse vyhľadá vetu, ktorá sa na tento vstup najviac podobá. Nakoniec v závislosti od implementácie vráti buď vetu, ktorá v korpuse nasleduje za nájdenou vetou alebo nájdenú vetu ako takú.

2.5 GUS

V tejto sekcii sa budem podrobnejšie venovať architektúre GUS systémov.

Základné rozdelenie problematiky

V tejto časti rozoberiem GUS systémy na 4 základné časti a jednotlivé časti zadefinujem a detailne rozoberiem.

- **Rozoznávanie pomenovaných entít (Named Entity Recognition)**
  Získanie zoznamu pomenovaných entít zo vstupnej vety.

- **Zisťovanie zámeru (Intent Determination)**
  Zistenie používateľovho zámeru.

- **Napĺňanie slotov (Slot Filling)**
  Naplnenie potrebných slotov na vykonanie používateľovho zámeru

- **Vykonanie akcie (Action Execution)**
  Vykonanie používateľovho zámeru s korektnými argumentami.

2.5.1 Rozoznávanie pomenovaných entít(NER)

Pomenovaná entita je dvojica slovo a označenie, kde označenie je jedným z IOB tagov podľa štandardu [3]. Vstupom pre tento proces je veta a výstupom tohto procesu je zoznam pomenovaných entít, pričom počet prvkov

zoznamu musí byť rovný počtu slov v pôvodnej vete. Existuje viacero prístupov ku NER. Medzi tri základné patria *Feature-based*, *Neural* a *Rule-based*[3].

2.5.2 Zisťovanie používateľského zámeru

V tejto časti je úlohou programu rozoznať aký je používateľov zámer. Vstupom je zoznam pomenovaných entít, ktoré sme dostali pomocou Named entity recognition a výstupom tohto procesu je používateľský zámer, na základe ktorého vie aplikácia následne rozhodnúť aký frame má začať vypĺňať. Tak ako pri named entity recognition aj tu existuje viacero prístupov. Medzi základné prístupy patrí supervised prístup, unsupervised prístup a rule-based prístup[3]. V nasledujúcej sekcii sa budem venovať iba rule-based prístupu. Hlavným princípom tohto prístupu je, že každý zámer má svoje pravidlá. Tieto pravidlá po aplikovaní na danú vetu rozhodnú, nakoľko bol používateľov zámer totožný so zámerom, ku ktorému sa aplikované pravidlá vzťahujú. Po prejdení všetkými zámermi sa vyberie ten, ktorý mal najväčšiu totožnosť s používateľovým zámerom. Takto je aplikácia schopná rozoznať zámer aj v prípade, že používateľ neposkytol všetky potrebné informácie na vykonanie požadovanej akcie.

2.5.3 Napĺňanie slotov

Pre pochopenie tejto časti sú dôležité dva pojmy a to frame a slot. Tieto pojmy sú stručne vysvetlené v časti 2.2. Predstavme si, že používateľský zámer je napríklad nájdenie cesty medzi dvoma miestnosťami. Frame teda bude obsahovať dva sloty a to odkiaľ používateľ ide a kam sa chce dostať. Úlohou slot fillingu je získať všetky potrebné informácie na vykonanie používateľovho zámeru a naplniť tieto informácie do korektných slotov framu. Vstupom je zoznam pomenovaných entít, ktoré sme dostali pomocou NER a výstupom je frame s naplnenými slotmi. Existuje viacero prístupov ku problematike slot fillingu. Medzi základné patrí mapovanie slov pomocou kontextuálneho vkladania pomocou systému BERT a rule-based prístup [3]. V tejto kapitole sa budem venovať opäť iba rule-based prístupu. Hlavnou ideou rule-based prístupu je , že každý frame bude mať sadu pravidiel, na základe ktorých bude vedieť jednotlivé sloty naplniť. Taktiež každý frame bude mať k dispozícii sadu otázok, ktoré budú slúžiť na získavanie doplňujúcich informácii od

používateľa. Napríklad pri otázke "How to get to A?"by bolo nevyhnutné sa spýtať odkiaľ sa tam používateľ chce dostať. Postupným kladením doplňujúcich otázok frame naplní svoje sloty.

2.5.4 Vykonanie akcie

Akonáhle má frame naplnené všetky potrebné sloty, aplikácia vie akú akciu má vykonať, rovnako ako vie aj aké argumenty má poskytnúť. V tejto časti ostáva zavolať aplikačné rozhranie ktoré vykoná akciu potrebnú na splnenie používateľovho zámeru. Výstup tejto časti bude buď text zodpovedajúci odpovedi na používateľovu otázku alebo vykonanie akcie ktorú používateľ vyžaduje.

Pochopenie prirodzeného jazyka

Dôležitou súčasťou týchto systémov je porozumenie prirodzeného jazyka. To zahŕňa rozoznanie domény, nad ktorou systém pracuje, zistenie aký má používateľ zámer a následné získavanie podstatných informácii, ktoré sú potrebné na splnenie používateľovho zámeru. Sú rôzne spôsoby ako toto porozumenie dosiahnuť. Medzi základné prístupy patrí supervised-learning, semi-supervised, unsupervised a rule-based [3]. Najbežnejším spôsobom je využitie supervised-learning algoritmu, v ktorom sa systém učí na základe veľkého množstva dát, ktoré mu vývojár poskytne. Jednoduchšou alternatívou riešenia natural language understandingu je rule-based prístup čiže rozoznávanie textu na základe vopred stanovených pravidiel. Tento prístup nevyžaduje veľké množstvo dát a aj napriek tomu dosahuje vysokú presnosť v porozumení prirodzenému jazyku.

2.6 Základné rozdelenie používateľského zámeru

Používateľský zámer, respektíve to, čo používateľ od aplikácie vyžaduje sa dá rozdeliť do dvoch kategórii:

- **Vykonanie akcie**
  Aplikácia vykoná úkon, ktorý používateľ požaduje.

- **Vyhľadávanie odpovede**
  Aplikácia vyhľadá odpoveď na používateľovu otázku.

Vykonanie akcie

Vykonanie akcie aplikácia realizuje pomocou už existujúcich aplikačných rozhraní. Vykonanie akcie sa realizuje až po naplnení rámca a teda aplikácia už vie aké parametre treba pre volané rozhranie poskytnúť.

Vyhľadávanie odpovede

V tejto sekcii sa budem venovať factoid question answeringu. Factoid question answering sa venuje otázkam, ktoré môžu byť zodpovedané jednoduchým faktom. Vyhľadávanie odpovedí na takéto otázky sa rozdeľuje do dvoch veľkých kategórii[3]:

- **Information-retrieval**
  Tento spôsob sa spolieha na veľké kvantum textu.
- **Knowledge-based**
  Tento spôsob vykonáva dopyty nad svojou databázou znalostí.

Information-retrieval

Existuje viacero prístupov k riešeniu Information-retrieval question answeringu. V tejto časti prejdem dvomi základnými.

- **Basic Answer Extraction**
  Otázku najprv preformulujeme na oznamovaciu vetu. Týmto spôsobom otázku 'Where is A' vieme preformulovať na vetu 'A is in'. Ďalším krokom je zistiť typ odpovede[3]. Typy odpovedí zväčša vyberáme z množiny typov pomenovaných entít. Potom je potrebné vybrať časť dokumentov, z tých čo máme k dispozícii, takú že bude s najväčšou pravdepodobnosťou obsahovať hľadanú odpoveď. Toto docielime procesom nazývaným passage retrieval[3]. Tento proces spočíva v tom, že má množinu pravidiel, podľa ktorých jednotlivé úseky textu ohodnotí a následne vyberie najvhodnejší úsek, teda ten s navyšším hodnotením.

Medzi spomínané pravidlá patrí napríklad počet pomenovaných entít rovnakého typu ako je typ odpovede[3]. Následne vo vybranom úseku vyhľadáme vetu, ktorá obsahuje našu preformulovanú otázku. Za touto otázkou nájdeme prvú entitu, ktorá bude mať rovnaký typ ako má byť typ odpovede. To bude naša odpoveď na používateľovu otázku.

- **Feature-based Answer Extraction**
  Tento prístup využíva regulárne výrazy, ktoré sú schopné namapovať otázku na časť odpovede. Tieto pravidlá vychádzajú z toho, že vetu je možné rozdeliť podľa spojky na dve časti, pričom jedna časť bude časťou otázky, a druhá časť bude v sebe obsahovať odpoveď na túto otázku. Ďalšou dôležitou súčasťou sú klasifikátory ako napríklad či je odpoveď správneho typu, počet kľúčových slov z otázky, ktoré odpoveď obsahuje, nakoľko odpoveď pasuje na naše regulérne výrazy, vzdialenosť medzi kľúčovými slovami a ďalšie[3].

Knowledge-based

Tento spôsob question answeringu využíva viac štruktúrované dáta ako Information-retrieval. Využíva systémy na mapovanie textu na logické formuly. Databáza, ktorú Knowledge-based systémy využívajú môže byť relačná ale kľudne aj jednoduchšia ako napríklad množina RDF trojíc. RDF trojice sú trojice obsahujúce predikát a dva argumenty. Tieto trojice zväčša vyjadrujú vlastnosť alebo tvrdenie[3]. Je viacero prístupov ako riešiť Knowledge-based question answering. Ja sa budem venovať len dvom najzákladnejším.

- **Rule-based Metóda**
  Princípom tejto metódy je sada pravidiel, na základe ktorých je systém schopný z otázky spraviť dopyt na databázu znalostí tak aby z nej získal odpoveď na používateľovu otázku. Táto metóda sa využíva najmä ak majú otázky tendenciu opakovať sa, respektíve ak rozsah domény, nad ktorou systém pracuje nie je veľký [3].

- **Supervised Metóda**
  Väčšina učiacich sa algoritmov pri question answeringu obsahuje základnú množinu pravidiel s rovnakým účelom ako pri rule-based metodách a to na transformovanie otázky na dopyt. Pri supervised metódach je ale dôležité mať aj trénovaciu množinu, ktorá v tomto prípade bude pozostávať z otázky spárovanej s jej správnou logickou formulou [3].

Na tejto množine sa potom supervised systém natrénuje a po dokončení tréningu by mal byť schopný transformovať aj otázku, ktorú jeho trénovacia množina neobsahovala. Takéto systémy by mali byť následne schopné odpovedať aj na otázky, ktoré sa netýkajú iba jednej relácie.

Kapitola 3

Vytvorené konverzačné rozhranie z používateľského hľadiska

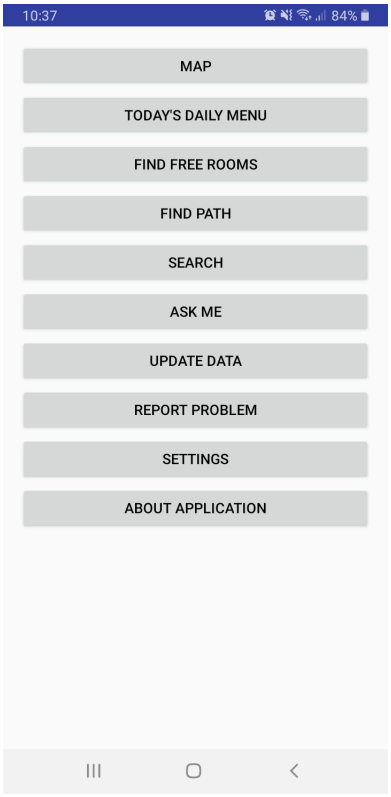
V tejto kapitole sa budeme venovať funkčnosti konverzačného rozhrania, ktoré som pridal do aplikácie Sprievodca FMFI, ako ho spustiť, ako s ním interagovať a aké výstupy podáva.

Nová verzia aplikácie Sprievodca FMFI po mojich úpravách obsahuje navyše tlačidlo ASK ME (Obr. 3.1). Po kliknutí na spomínané tlačidlo sa spustí aktivita určená pre konverzačné rozhranie. Konverzačné rozhranie k používateľovi vystupuje pod menom Mia.

S Miou je možné interagovať dvoma spôsobmi. Prvým spôsobom je písanie správ do textového políčka a následným odoslaním kliknutím na tlačidlo odoslania. Druhým spôsobom je klikanie na Miine správy.

Mia má dve základné funkcie, ktoré je schopná vykonávať:

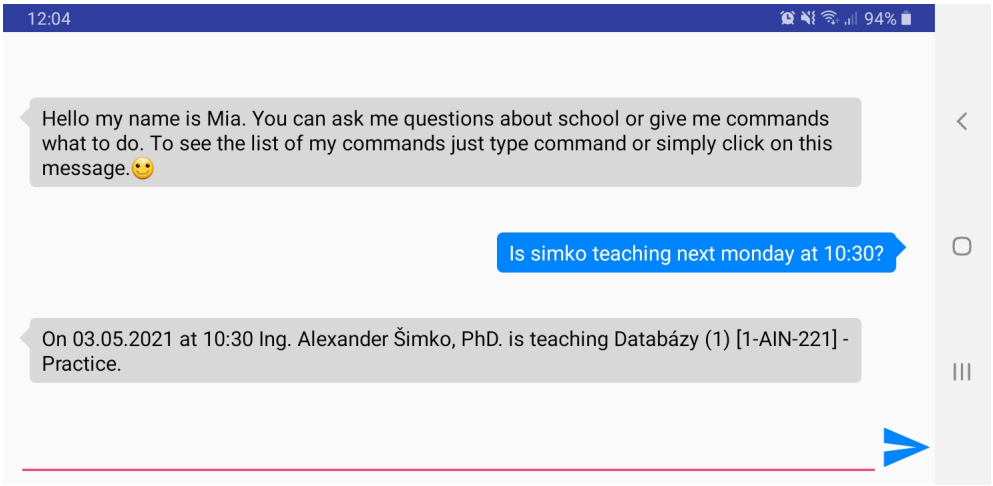
- odpovedanie na otázky týkajúce sa fakulty
- ovládanie už existujúcich funkcionalít aplikácie Sprievodca FMFI pomocou textového rozhrania



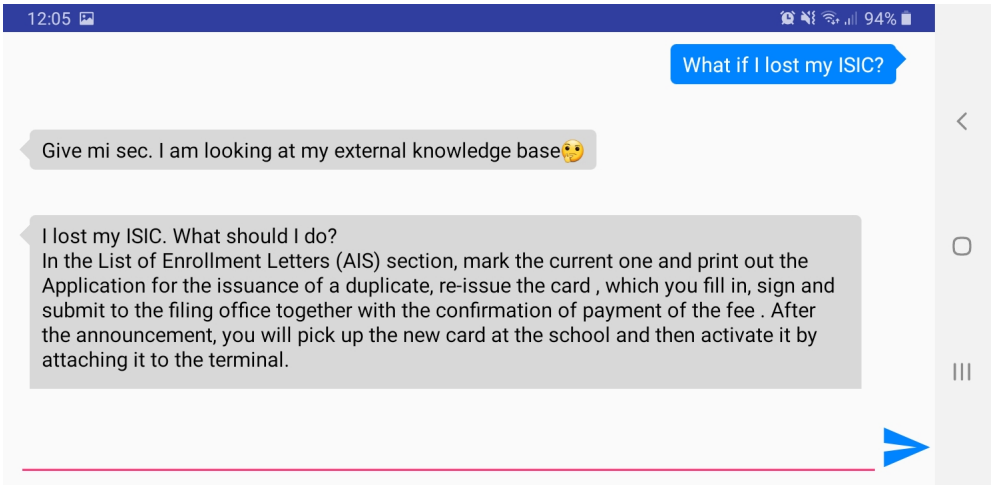
Obr. 3.1: Menu Sprievodcu FMFI po pridaní konverzačného rozhrania

Odpovedanie na otázky

V prípade otázok ohľadom fakulty používateľ môže zadať akúkoľvek otázku. Otázky nie sú nijako obmedzené. Na niektoré otázky vie Mia odpovedať na základe informácii, ktoré má k dispozícii (príklad takejto otázky je možné vidieť na obrázku 3.2).
V prípade, že Mia odpoveď na danú otázku nepozná, vypíše sa používateľovi správa Hmmm, ktorá znamená, že pýtanú otázku Mia odosiela na server, ktorý na kladenú otázku vyhľadá odpoveď a následne ju odošle späť Mii. Odpovede na otázky server vyhľadáva na stránke fakulty (https://fmph.uniba.sk/en/). Príklad otázky ako aj odpovede na ňu je ukázaný na obrázku. 3.3.



Obr. 3.2: Príklad otázok na ktoré Mia pozná odpoveď



Obr. 3.3: Príklad otázok, na ktoré Mia nepozná odpoveď - pýta sa servera

Ovládanie existujúcich funkcionalít

Druhým využitím Mii je ovládanie už existujúcich funkcionalít. Pri ovládaní existujúcich funkcionalít Mia pozná príkazy spojené s jednotlivými funkcionalitami. Príklady základných príkazov spolu s funkcionalitami, ktoré vykonávajú je v nasledujúcej tabuľke.

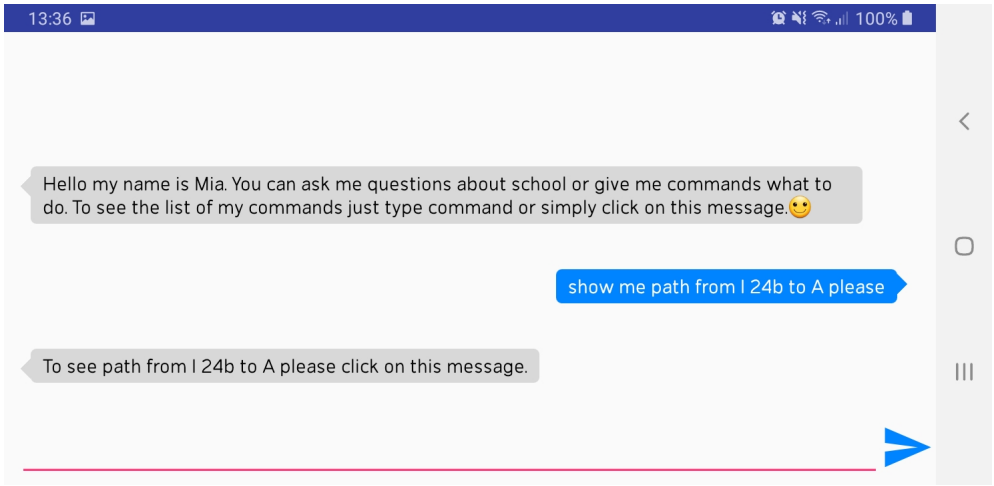
funkcionality	základné príkazy
vypísanie dátumu/času	What is the date today? What will be the time in 75 minutes?
vylistovanie príkladov príkazov	Show me commands.
vykreslenie aktuálneho menu v jedálňach	What meal can I buy? What is the menu?
vylistovanie miestností patriacich pod oddelenie	List me rooms that are _ .
nájdenie voľnej miestnosti	I am searching for free room.
zobrazenie informácii	Tell me information about _. Show me details about _ .
je osoba voľná	Is _ free in 10 minutes? Is _ teaching right now?
vylistovanie miestností s konkrétnym účelom	List me rooms that are _.
zobrazenie rozvrhu pre osobu/miestnosť	Show me schedule for _.
vykreslenie bodu na mape	Show me _ on map.
vykreslenie cesty na mape	Navigate me to _ from _ please. Show me path from _ to _.

*_ symbolizuje používateľom zadaný objekt

Pre každú funkcionalitu Mia vie aké objekty záujmu má hľadať. Teda napríklad pri hľadaní cesty Mia vie že cesta sa skladá z miesta odkiaľ používateľ ide a miesta kam sa chce používateľ dostať.
V prípade, že Mia dostane všetky objekty, ktoré potrebuje na vykonanie používateľovho príkazu a neexistuje viacero objektov s rovnakým názvom ako objekt, ktorý zadal používateľ, tak Mia vypíše správu zodpovedajúcu informáciam a príkazu od používateľa (obrázok 3.4). Po kliknutí na túto správu Mia vykoná používateľov príkaz.

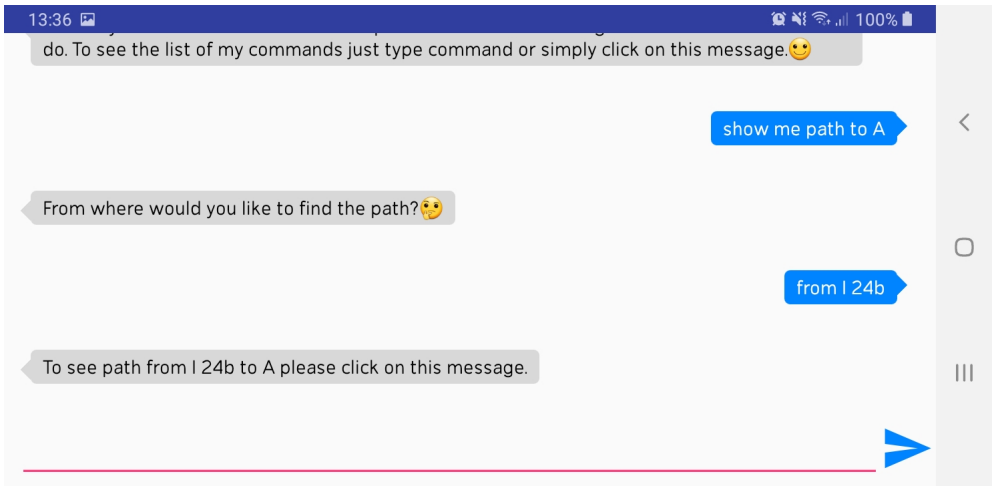
Zisťovanie doplňujúcich informácii

Mia vie reagovať na príkazy so zadanými všetkými objektami záujmu ale aj na príkaz s nekomplentým zoznamom objektov respektíve na príkaz bez akýchkoľvek objektov.



Obr. 3.4: Správa zodpovedajúca informáciam a príkazu od používateľa

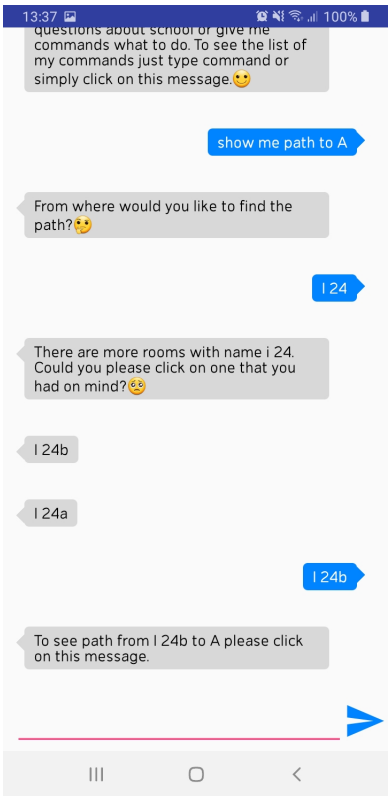
V prípade, že používateľ nezadá všetky informácie potrebné na vykonanie príkazu, respektíve neuvedie žiadne informácie, Mia sa na chýbajúce informácie postupne pýta a v následných používateľových odpovediach na jej otázky hľadá objekty, ktoré by mohla použiť pri vykonaní používateľovho príkazu (obrázok 3.5).



Obr. 3.5: Kladenie otázok na chýbajúce informácie

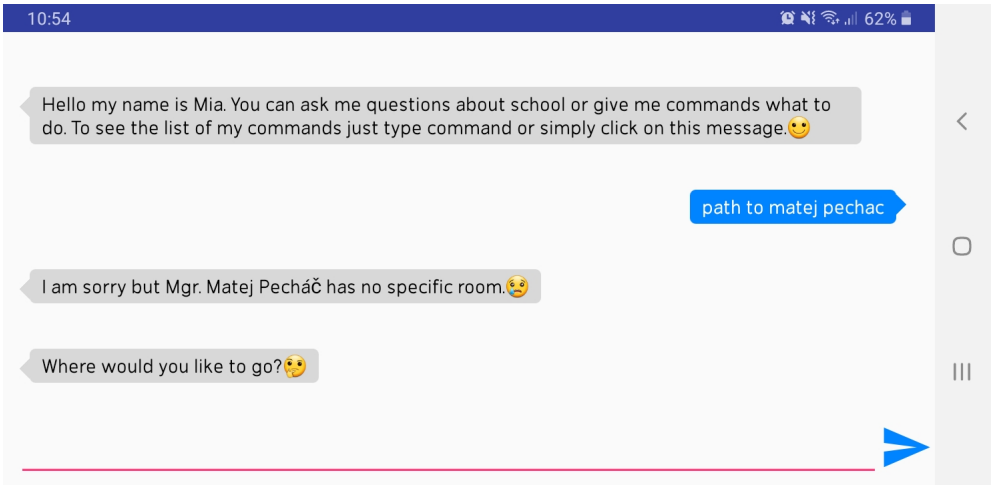
Ak nastane situácia, že existuje viacero objektov s rovnakým názvom ako ob-

jekt ktorý zadal používateľ, Mia vypíše zoznam týchto objektov a dá tak používateľovi možnosť si vybrať konkrétny objekt, ktorý mal na mysli(obrázok 3.6). Na tento zoznam je taktiež možné kliknúť, čo spôsobí, že správa, na ktorú používateľ klikol sa odošle ako používateľova správa, čím používateľ potvrdí svoj výber.



Obr. 3.6: Vyberanie nejednoznačného objektu

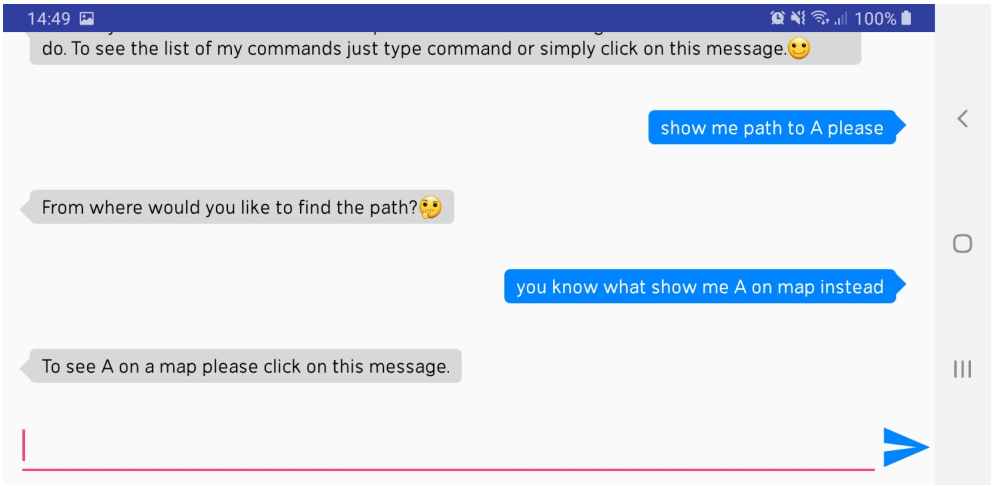
Mia sa snaží objekty od používateľa pretransformovať na objekty potrebné pre vykonanie používateľovho príkazu. Napríklad ak používateľ zadá, že chce na mape zobraziť osobu, nezobrazí sa osoba ako taká ale miestnosť, ktorú má daná osoba pridelenú. Rovnako tak sa Mia snaží pretransformovať aj vyučovacie predmety, oddelenia a jednotlivé účely miestností. Ak používateľom zvolená osoba alebo udalosť nemá konkrétnu miestnosť Mia o tom používateľa informuje a očakáva nový objekt, ktorý nahradí objekt bez miestnosti(obrázok 3.7).



Obr. 3.7: Osoba bez konkrétnej prikazBezMiestnosti

Zmena používateľovho zámeru

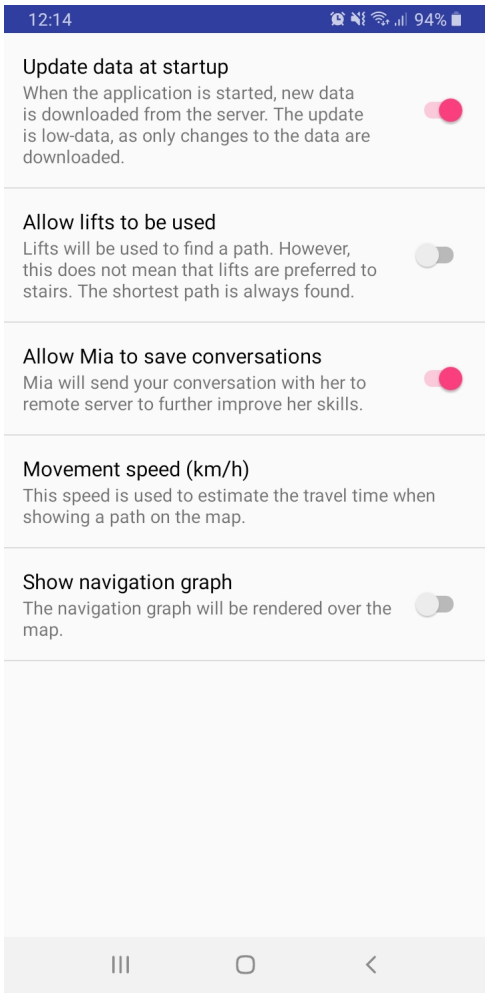
V priebehu konverzácie je Mia schopná určiť či sa používateľ nerozhodol zmeniť svoj pôvodný zámer. V takom prípade sa Mia už nesnaží zistiť informácie k pôvodnému príkazu ale k novo určenému (obrázok 3.8).



Obr. 3.8: Zmena používateľovho zámeru

Ukladanie konverzácii

Z dôvodu budúceho rozvoja je Mia schopná konverzácie s používateľom odosielať na server, kde sa následne uložia do databázy. Takto uložené konverzácie sa následne v budúcnosti môžu využiť na zlepšovanie reakcií Mii. Používateľ toto odosielanie môže povoliť respektíve zakázať v nastaveniach stlačením prepínača 'Allow Mia to save conversations'(obrázok 3.9).

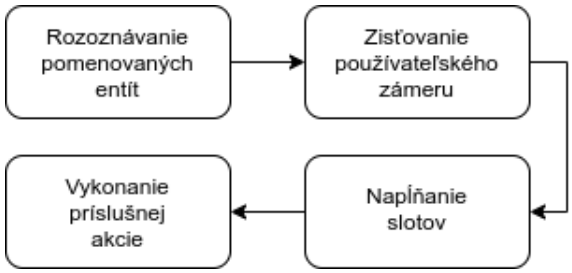


Obr. 3.9: Nastavenie na ukladanie konverzácii

Kapitola 4

Vytvorené konverzačné rozhranie z technického hľadiska

Tak ako bolo vo východiskovej kapitole naznačené, problematika dialógových systémov zameraných na vykonávanie používateľových príkazov sa skladá zo 4 hlavných podproblémov (obrázok 4.1). V tejto kapitole sa budem venovať mojej implementácii jednotlivých podproblémov.

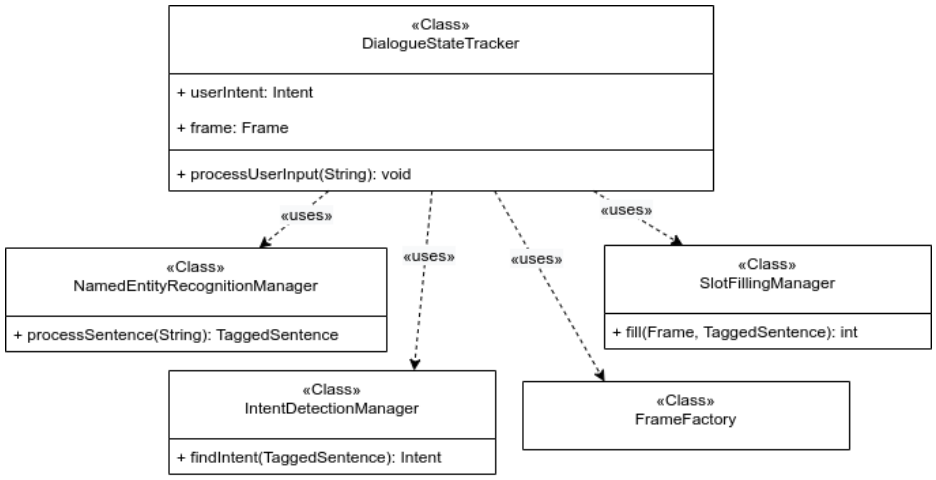


Obr. 4.1: Blokový diagram zobrazujúci náveznosť 4 hlavných podproblémov

Pre pripomenutie v tejto časti budem používať pojem pomenovaná entita, ktorou sa myslí postupnosť slov, ktorú napíše používateľ a databázová entita, ktorou sa myslí riadok v databáze, ktorý obsahuje dáta o nejakom objekte.

4.1 Manažovanie dialógu

Na to aby bola Mia schopná fungovať treba spomínané 4 hlavné podproblémy (obrázok 4.1) manažovať a vzájomne prepojiť. Na účely manažovania týchto podproblémov slúži trieda *DialogueStateTracker* (obrázok 4.2). Pre každý používateľov vstup sa zavolá metóda *processUserInput*, ktorá najprv z používateľovho vstupu získa zoznam pomenovaných entít pomocou triedy, ktorá je zodpovedná za proces rozoznávania pomenovaných entít (*NamedEntityRecognitionManager*). Pomocou získaného zoznamu pomenovaných entít Mia následne určí používateľov zámer pomocou triedy *IntentDetectionManager*. Zistený používateľov zámer sa ukladá v členskej premennej *userIntent*. Ak *DialogueStateTracker* nemá uložený žiaden rámec vytvorí si nový rámec zodpovedajúci danému používateľskému zámeru pomocou factory metód nachádzajúcich sa v triede *FrameFactory* a ten si uloží do členskej premennej *frame*. V prípade, že v členskej premennej *frame* už bol uložený nejaký rámec ale zmenil sa používateľov zámer, uložený rámec sa zahodí a uloží sa novy rámec zodpovedajúci danému zámeru. Keď máme určený používateľský zámer aj vytvorený rámec môžme začať s procesom napĺňania slotov. Tento proces prebieha v triede *SlotFillingManager* a sloty sa napĺňajú na základe pomenovaných entít z používateľovho vstupu. Ak sa všetky sloty naplnia Mia vykoná akciu zodpovedajúcu používateľovmu zámeru. V prípade, že niektorý zo slotov ostane prázdny Mia používateľovi položí doplňujúcu otázku a tento proces sa zopakuje odznova počínajúc metódov *processUserInput*.



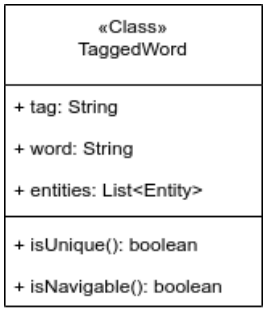
Obr. 4.2: Triedny diagram zobrazujúci triedu *DialogueStateTracker*

4.2 Rozoznávanie pomenovaných entít

Ako prvú časť som v aplikácii navrhoval a implementoval rozoznávanie pomenovaných entít nakoľko od tejto časti sa odvíjajú aj ostatné súčasti konverzačného rozhrania.
Na rozoznávanie pomenovaných entít som zvolil kombináciu dvoch rôznych prístupov. Na rozoznávanie pomenovaných entít spojených s objektami viazanými na fakultu som využil databázu aplikácie obsahujúcu osoby, miestnosti, účely miestností, oddelenia, rozvrhy a predmety. Rozvrhy a predmety sú súčasťou paralelne sa vyvíjajúcej bakalárskej práce(odkaz na ňu). Na pomenované entity ako dátum, čas alebo interval som využil prístup zameraný na pravidlá, ktorý som realizoval pomocou regulárnych výrazov. Týmto dvom spôsobom sa budem bližšie venovať v nasledujúcich podkapitolách.
Implementácia rozoznávania pomenovaných entít sa nachádza v balíčku *namedEntityRecognition*.

Pomenované entity

Samotná pomenovaná entita je reprezentovaná pomocou triedy *TaggedWord* (obrázok 4.3).



Obr. 4.3: Triedny diagram zobrazujúci triedu *TaggedWord*

Táto trieda obsahuje označenie daného objektu a samotný objekt zadaný od používateľa. Označenie objektu je string určujúci, čo daný objekt predstavuje, napríklad pri osobe je označenie person, pri miestnosti room a pri dátume date. Zoznam týchto pomenovaní ako aj ich význam je v tabuľke číslo 4.1.

Trieda *TaggedWord* taktiež obsahuje aj list databázových entít, ktoré majú podobný názov. Metóda *isUnique* vracia *true* v prípade, že neexis-

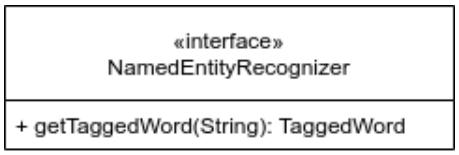
pomenovanie	význam
date	pomenovaná entita predstavujúca dátum
time	pomenovaná entita predstavujúca čas
interval	pomenovaná entita predstavujúca interval
greeting	pomenovaná entita predstavujúca pozdrav
thanks	pomenovaná entita predstavujúca poďakovanie
ack	pomenovaná entita predstavujúca potvrdenie
number	pomenovaná entita predstavujúca číslo
purpose	pomenovaná entita predstavujúca účel miestnosti
department	pomenovaná entita predstavujúca oddelenie
room	pomenovaná entita predstavujúca miestnosť
person	pomenovaná entita predstavujúca osobu
abstractEvent	pomenovaná entita predstavujúca vyučovací predmet
event	pomenovaná entita predstavujúca udalosť
out	entita, ktorá nespadá do žiadnej konkrétnej kategórie

Tabuľka 4.1: Tabuľka pomenovaní a ich významov

tuje viac ako jedna entita s podobným názvom, teda dĺžka listu *entities* je rovná jednej, inak táto funkcia vráti *false*. Metóda *isNavigable* vracia *true* v prípade, že je táto pomenovaná entita unikátna (výsledok metódy *isUnique* je *true*) musí byť taktiež inštanciou triedy *NavigableEntity* a zároveň musí mať *vertexId* rôzny od *null*. Inak táto funkcia vráti *false*.

Rozoznávače pomenovaných entít

Každé označenie má svoj špecifický rozoznávač, ktorého úlohou je určiť či entita patrí do danej množiny pomenovaných entít. Všetky rozoznávače implementujú rozhranie *NamedEntityRecognizer*(obrázok 4.4), ktoré sa nachádza v balíčku *namedEntities*.

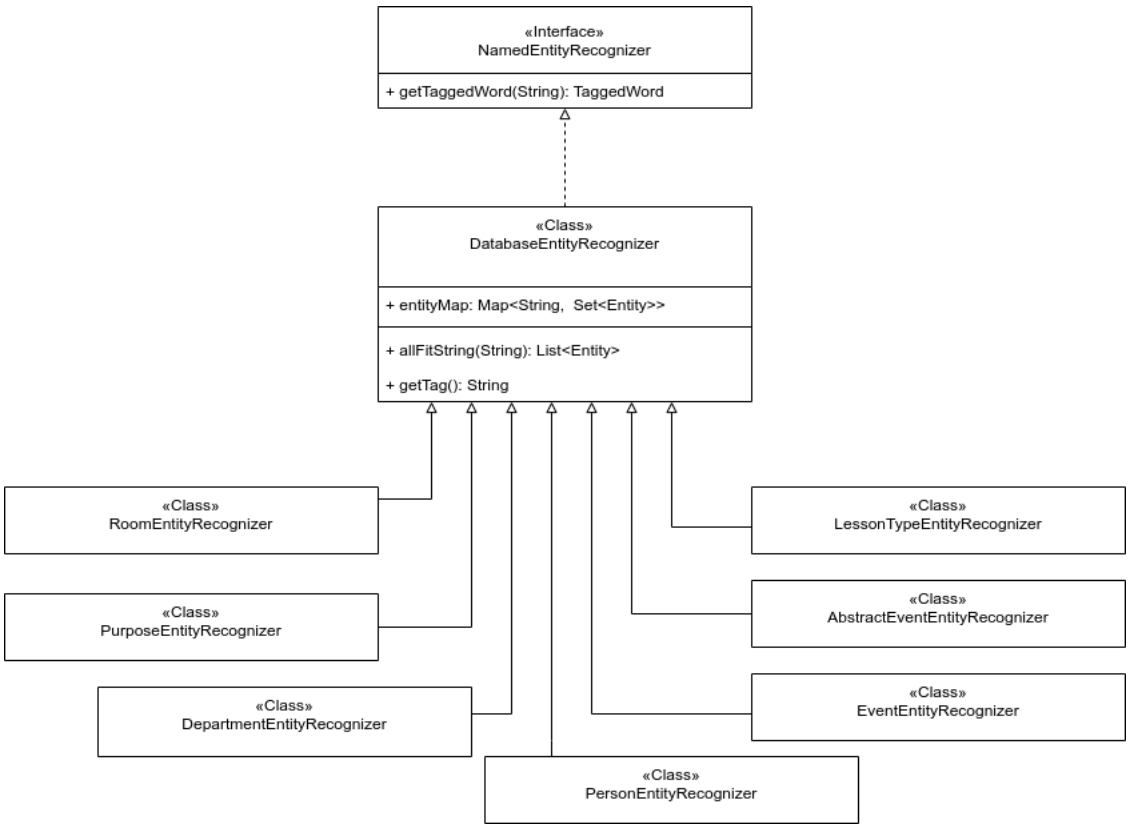


Obr. 4.4: Triedny diagram zobrazujúci interface *NamedEntityRecognizer*

Rozhranie *NamedEntityRecognizer* slúži ako základ pre všetky rozoznávace. Metóda *getTaggedWord* vráti inštanciu triedy *TaggedWord* v prípade, že vstupná entita patrí do množiny pomenovaných entít, ktorú daný rozoznávač predstavuje. V prípade, že vstupná entita do tejto množiny nepatrí tak táto metóda vráti *null*. Rozhranie *NamedEntityRecognizer* implementujú dve abstraktné triedy a to *DatabaseEntityRecognizer* a *PatternEntityRecognizer*. Triedy dediace z abstraktnej triedy *DatabaseEntityRecognizer* rozoznávajú pomenované entity na základe databázy, ktorú má aplikácia k dispozícii. Triedy dediace z abstraktnej triedy *PatternEntityRecognizer* rozoznávajú pomenované entity na základe predpísaného vzoru. Všetky triedy dediace z týchto dvoch abstraktných tried slúžia na rozoznávanie konkrétnych pomenovaných entít.

Trieda DatabaseEntityRecognizer

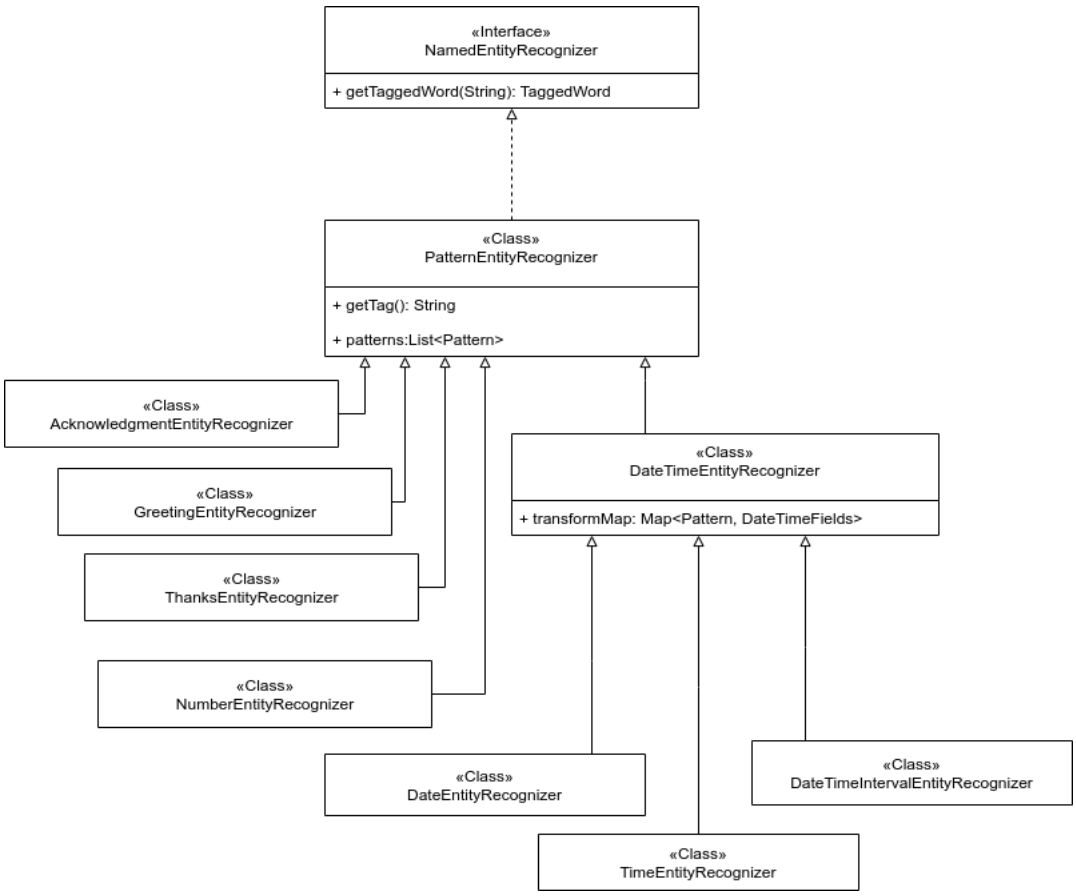
Obrázok 4.5 zobrazuje triedu *DatabaseEntityRecognizer* ako aj všetky triedy dediace z tejto abstraktnej triedy.
Triedy, ktoré dedia z abstraktnej triedy *DatabaseEntityRecognizer* využívajú na rozoznanie pomenovaných entít údaje získané z databázy. V konštruktore, každá z týchto tried získa z databázy množinu databázových entít, ktoré je schopná rozoznať (napríklad *PersonEntityRecognizer* získa množinu všetkých osôb) a následne si túto množinu uloží do premennej entityMap. Ako klúč sa použijú podstringy z poľa stringov získaného rozdelením výstupu metódy *getDisplayString* podľa medzery. Metódu *getDisplayString* som v svojom riešení pridával ja a má ju každá trieda implementujúca rozhranie Entity. Metóda *allFitString* vracia zoznam inštancií rozhrania *Entity*, ktoré majú názov podobný vstupnému stringu. Vyhľadávanie takýchto inštancii sa vykonáva pomocou premennej *entityMap*, kde vstup pre metódu *allFitString* slúži ako kľúč do premennej *entityMap*.
V prípade, že dĺžka zoznamu, ktorý vráti metóda *allFitString* je rôzna od nuly, metóda *getTaggedWord* vráti novú inštanciu triedy TaggedWord, ktorá bude mať označenie na základe metódy *getTag* a pole entít sa nastavý na pole, ktoré dostaneme volaním metódy *allFitString*. V prípade, že metóda *isUnique* vráti false Mia používateľovi poskytne výber inštancii rozhrania *Entity*, na základe poľa entities nachádzajúceho sa v triede *TaggedWord*.



Obr. 4.5: Triedny diagram pre abstraktnú triedu *DatabaseEntityRecognizer*.

Trieda PatternEntityRecognizer

Na obrázku 4.6 je zobrazené trieda PatternEntityRecognizer ako aj všetky triedy dediace z tejto abstraktnej triedy.
Abstraktná trieda *PatternEntityRecognizer* si v premennej *patterns* uchováva inštancie triedy *Pattern*, ktoré predstavujú ručne písané vzory, voči ktorým sa vstupná pomenovaná entita kontroluje. Tieto vzory majú formu regulárnych výrazov. Recognizer rozpozná entitu, práve vtedy ak entita vyhovuje aspoň jednému zo vzorov nachádzajúcich sa v premennej *patterns*. Ak sa nájde vzor, ktorému vstupný string vyhovuje tak metóda *getTaggedWord* vráti novú inštanciu triedy *TaggedWord* obsahujúcu označenie na základe metódy *getTag*. Keďže všetky pomenované entity, ktoré sa rozoznávajú pomocou pravidiel nie sú priamo spojené s FMFI, inštanciám triedy *TaggedWord PatternEntityRecognizer* automaticky nastavuje *navigable* na *false* a *unique* na *true*.

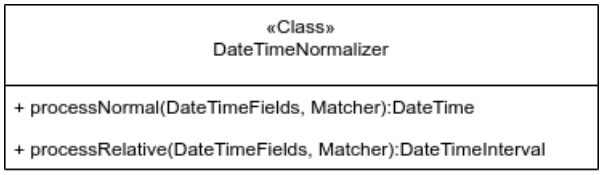


Obr. 4.6: Triedny diagram pre abstraktnú triedu *PatternEntityRecognizer*.

DateTimeNormalization

Pre spracovanie používateľovho vstupu je dôležité nielen rozoznať dátumy, časy a intervaly ale aj ich znormalizovať. Normalizácia je dôležitá kvôli následnej práci so vstupnými dátami. V prípade, že používateľ chce vedieť či konkrétna osoba učí budúci pondelok je dôležité zistiť, že vstup 'budúci pondelok' je dátum a následne určiť aký konkrétny dátum to je. Pri normalizácii potrebujeme v pomenovanej entite identifikovať jednotlivé zložky dátumu a času. Napríklad pre dátum tvaru rok-mesiac-deň vyzerá vzor nasledovne: "(\\d{4})-(0?[1-9]|1[012])-(0?[1-9]|[12][0-9]|3[01])". Tento vzor má tri zložky a

to zložka (\\d{4}), ktorá symbolizuje rok zložka (0?[1-9]|1[012]), ktorá symbolizuje mesiac a zložka (0?[1-9]|[12][0-9]|3[01]), ktorá symbolizuje deň v používateľovom vstupe. Vzorov je viac a tak sa ukladajú v členskej premennej *transformMap*, kde kľúčom je vzor a hodnotou je inštancia triedy *DateTimeFields*, ktorá slúži ako wrapper na triedu *Map*. Na rozoznanie a normalizáciu dátumu, času a intervalu slúžia triedy dediace z abstraktnej triedy *DateTimeEntityRecognizer*. Hodnoty členskej premennej *transformMap* využíva trieda DateTimeNormalizer (obrázok 4.7), ktorá pomocou metódy *processNormal* spracuje dátum a čas a pomocou metódy *processRelative* spracuje vstupné intervaly.



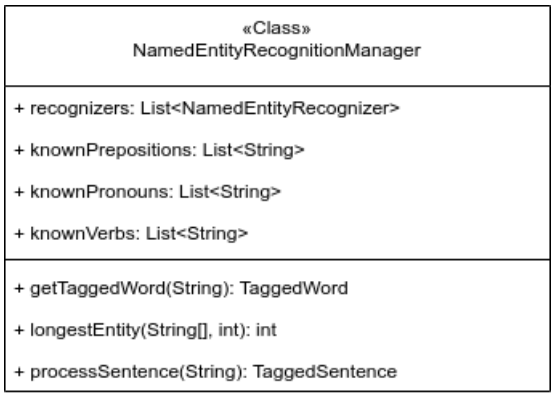
Obr. 4.7: Triedny diagram pre triedu DateTimeNormalizer.

Obe tieto metódy dostanú na vstupe inštanciu triedy *DateTimeFields* a inštanciu triedy Matcher, ktorá predstavuje pravidlo spojené s používateľovým vstupom. Metóda *processNormal* vracia inštanciu triedy *DateTime* a metóda *processRelative* vracia inštanciu triedy *DateTimeInterval*. Obe tieto triedy sa nachádzajú v balíčku *dateTimeNormalization*.
Trieda DateTime predstavuje wrapper na triedu GregorianCalendar. Na tieto účely by bolo jednoduchšie využiť triedu LocalDate avšak vzhľadom na to, že aplikácia Sprievodca FMFI poskytuje podporu aj starším zariadeniam využitie tejto triedy nebolo možné.
Trieda DateTimeInterval obsahuje inštanciu triedy DateTime, ktorá predstavuje jej počiatočný dátum a čas, ktorý je defaultne nastavený na aktuálny dátum a čas. Ďalej táto trieda obsahuje mapu stringov na integery, kde kľúčom je konkrétna časť dátumu a času ako napríklad rok, deň alebo hodina a hodnotou je koľko daných častí treba pripočítať. Napríklad kľúč year a hodnota 10 znamená, že k počiatočnému dátumu sa pripočíta 10 rokov.

Trieda NamedEntityRecognitionManager

Trieda *NamedEntityRecognitionManager* (obrázok 4.8) slúži na vykonávanie procesu rozoznávania pomenovaných entít vo vstupnej vete. Táto trieda obsa-

huje usporiadaný zoznam inštancií tried predstavujúcich jednotlivé rozoznávače. Taktiež obsahuje zoznam predložiek, zámen a slovies, ktoré sa pre lepšie rozoznávanie pomenovaných entít ignorujú, nakoľko veľa pomenovaných entít v sebe prvky týchto zoznamov obsahujú avšak tie prvky nepredstavujú dôležitú súčasť tejto entity.



Obr. 4.8: Triedny diagram pre triedu *NamedEntityRecognitionManager*.

Metóda *getTaggedWord* prechádza zoznam rozoznávačov pomenovaných entít a v prípade, že niektorý z nich vrátil inštanciu triedy TaggedWord a nie null tak táto metóda získanú inštanciu vráti ako výstup. Ak všetky rozoznávače vrátia null táto metóda vráti inštanciu triedy TaggedWord s označením out, čo reprezentuje, že táto entita nemá žiadne konkrétne pomenovanie.
Pri rozoznávaní pomenovaných entít chceme nájsť čo najväčšiu možnú entitu aby sme mali čo najpresnejšie pomenované entiy. Vstup Meno Priezvisko chceme označiť ako jednu veľkú entitu obsahujúcu Meno aj Priezvisko a nie ako dve malé entity, jednu obsahujúcu Meno a druhú obsahujúcu Priezvisko. Metóda *longestEntity* teda má za úlohu prechádzať vstupný zoznam slov a nájsť index, na ktorom končí najväčšia možná entita začínajúca na vstupnom indexe i.

Algoritmus ako toto vyhľadávanie prebieha je zobrazený a popísaný nižšie (algoritmus 1).
Metóda *longestEntity* má nastavený limit na najväčšiu možnú entitu. Vyhľadávanie najväčšej entity prebieha spojením podpoľa začínajúceho na indexe i a končiaceho na indexe k, pričom index k sa postupne zmenšuje. Takto vytvorený string sa následne odošle metóde *getTaggedWord*. Ak táto metóda vráti entitu označenú ako out, tak sa k zmenší o 1 a tento proces sa opakuje

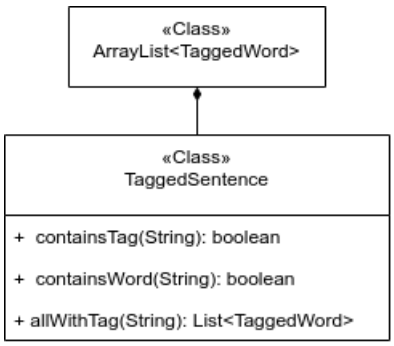
Algorithm 1: Ako funguje longestEntity

```
Result: Index, na ktorom končí najväčšia možná entita
i = start index;
k = min(length of sentence, i + limit);
result = null;
while k ≥ i do
    subSentence = sentence from index i to index k;
    taggedWord = getTaggedWord(subSentence);
    if result == null or 'out' == result.getTag() then
        result = taggedWord;
    end
    if 'out' != result.getTag() or k == i then
        break;
    end
    k = k - 1;
end
taggedSentence.add(result);
return k;
```

pokým metóda *getTaggedWord* nevráti entitu označenú inak ako out alebo pokým k nie je rovné i. Keď tento proces skončí inštancia triedy *TaggedWord*, ktorá sa nachádza v premennej *result* sa pridá do zoznamu pomenovaných entít a metóda vráti k. Metóda *processSentence* ako argument dostane používateľov vstup, ktorý si podľa medzier rozdelí na jednotlivé slová a pomocou metódy *longestEntity* takto rozdelený zoznam slov označí a získa tak zoznam pomenovaných entít. Tento výsledný zoznam je reprezentovaný ako inštancia triedy TaggedSentence(obrázok 4.9), ktorá je kompozíciou triedy ArrayList<TaggedWord>.

4.3 Rozoznávanie používateľovho zámeru

V tejto sekcii sa budem venovať mojej implementácií rozoznávania používateľovho zámeru. Prístup, ktorý som zvolil spadá do kategórie prístupov zameraných na rozoznávanie používateľovho zámeru na základe predpísaných pravidiel. Tento prístup som zvolil z dôvodu, že na akékoľvek prístupy založené na machine learningu je potrebné veľké množstvo dát, ktoré na za-

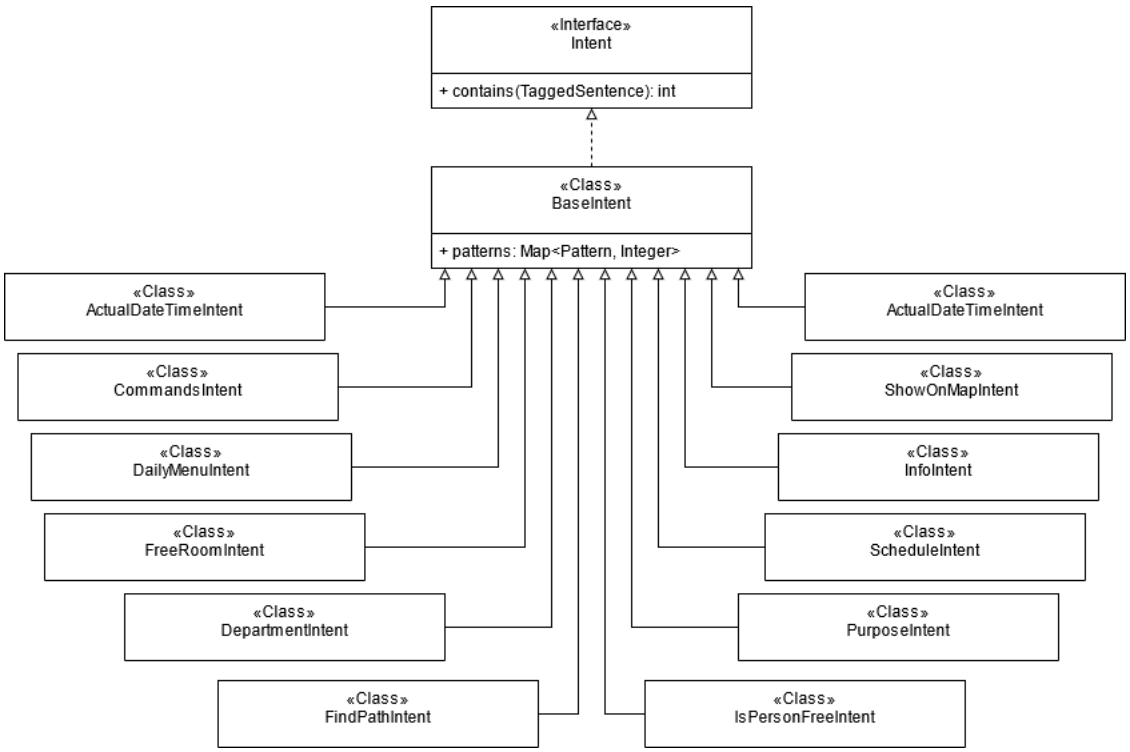


Obr. 4.9: Triedny diagram pre triedu TaggedSentence.

čiatku ešte nemáme k dispozícii.
Rozoznávanie používateľovho zámeru je realizované v balíčku *intentDetection*. Tento balíček obsahuje triedu *IntentDetectionManager* a triedy implementujúce rozhranie Intent, ktoré predstavujú jednotlivé používateľské zámery.

Rozhranie Intent

Triedy implementujúce rozhranie *Intent* predstavujú jednotlivé používateľske zámery (obrázok 4.10). Pre každý používateľský zámer je jedna trieda. Každá takáto trieda je zodpovedná za určenie či vstupný text spadá do daného používateľského zámeru. Na to aby sa kód neopakoval som spoločné časti dal do abstraktnej triedy *BaseIntent*.
Abstraktná trieda *BaseIntent* obsahuje zoznam vzorov, ktorý využíva na určovanie používateľského zámeru. Tieto vzory sú uložené v členskej premennej *patterns*, ktorá je typu *Map*. Kľúčom v tejto mape sú samotné vzory, realizované prostredníctvom triedy Pattern. Každá z tried dediacich z triedy *BaseIntent* si pri konštruktore do mapy patterns vloží svoje pravidlá. Jednotlivé vzory sú reprezentáciou rôznych možných používateľových vstupov, pre daný zámer.
V pravidlách sa nepíšu priamo pomenované entity ale ich označenia v tvare: \u2402označenie\u2403. Unikódový znak \u2402 znamená začiatok daného označenia a \u2403 znamená koniec tohto označenia. Znak \u2402 je špeciálny unikódový znak, ktorý znamená začiatok textu a znak \u2403 znamené koniec textu [6]. Označenia sú v pravidlách písané takto aby sa označenie nezamieňalo za používateľov vstup.



Obr. 4.10: Triedny diagram pre interface *Intent*, abstraktnú triedu *BaseIntent* a triedy dediace z tejto abstraktnej triedy.

Vzor môže vyzerať napríklad takto: 'path from \u2402 room \u2403 to \u2402 room \u2403' . Tento vzor patrí do množiny vzorov pre triedu *FindPathIntent* a rozoznáva vstup pýtajúci sa na cestu medzi akýmikoľvek dvoma miestnosťami.
Každý vzor má priradené svoje skóre. Toto skóre je nastavené podľa toho nakoľko dané pravidlo určuje daný zámer.
Metóda *contains* slúži na určenie, nakoľko daná veta zodpovedá konkrétnemu používateľskému zámeru. Keďže vstupom pre túto metódu je inštancia triedy TaggedSentence a reguárne výrazy je možné aplikovať iba na string tento vstup si musíme pretransformovať. Táto transformácia prebieha tak, že každú pomenovanú entitu (inštancia triedy TaggedWord) nahradíme nasledovne:

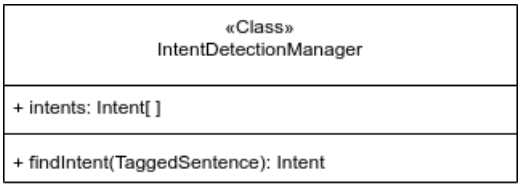
- ak má označenie 'out', teda nespadá do žiadnej konkrétnej množiny pomenovaných entít, tak sa nahradí samotným slovom

- ak má iné označenie ako 'out' tak sa nahradí \u2402označenie\u2403

S takto pretransformovaným vstupom následne prechádzame všetky vzory uložené v členskej premennej *patterns*. V prípade že niektorému zo vzorov vyhovuje pretransformovaný string k výslednému skóre sa pripočíta skóre, ktoré je priradené k tomuto vzoru. Výsledná suma je následne výstupom funkcie *contains*.

Trieda IntentDetectionManager

Trieda *IntentDetectionManager*(obrázok 4.11) slúži na manažovanie rozoznávania používateľovho zámeru.



Obr. 4.11: Triedny diagram pre triedu *IntentDetectionManager*.

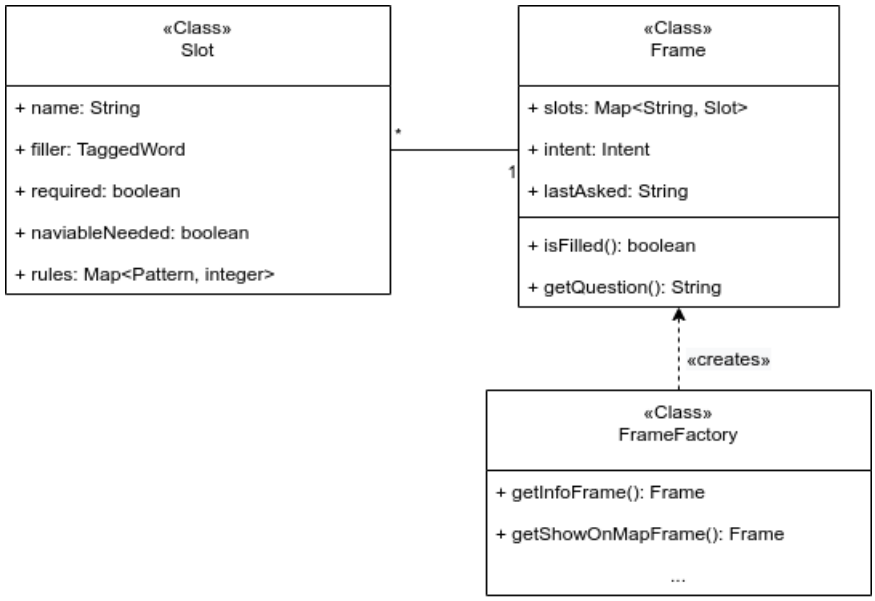
Táto trieda má členskú premennú *intents*, ktorá je pole inštancií všetkých tried implementujúcich rozhranie *Intent*, pričom každá trieda sa tam nachádza iba raz. Metóda *findIntent* dostane na vstupe inštanciu triedy *TaggedSentence*. Následne potom prechádza pole *intents* a pre každý zámer zavolá funkciu *contains*. Na základe výsledkov tejto funkcie pre jednotlivé zámery metóda nájde najlepší zámer teda taký, čo má najväčšie skóre. Tento zámer bude výstupom tejto metódy.

4.4 Napĺňanie slotov

V tejto sekcii sa budem venovať svojej implementácii procesu napĺňaniu slotov. Pre pripomenutie každý používateľský zámer má pridelený svoj vlastný rámec (frame), ktorý pozostáva zo slotov. Každý slot reprezentuje informáciu potrebnú pre vykonanie používateľovho zámeru. Triedy súvisiace s procesom napĺňania slotov sa nachádzajú v balíčku *slotFilling*.

Rámce a sloty

V tejto podsekcii sa budem venovať rámcom a slotom, ich definovaniu a implementácii.
Pre rámce aj sloty si potrebujeme uchovávať dve veci a to ich definíciu a potom samotné hodnoty, ktoré obsahujú. Rámce sú reprezentované pomocou triedy *Frame* a sloty pomocou triedy *Slot* (obrázok 4.12).Po zistení používateľského zámeru rámec získavame volaním factory metód nachádzajúcich sa v triede *FrameFactory*. Táto trieda obsahuje metódu pre kažý používateľský zámer, ktorá vracia novú inštanciu rámca zodpovedajúceho danému zámeru. Napríklad ramec pre nájdenie cesty obsahuje dva sloty a to od kiaľ sa má cesta hľadať a kam sa používateľ chce dostať. Zísaný rámec následne napĺňame pomenovanými entitami.



Obr. 4.12: Triedny diagram pre triedu Slot, Frame a FrameFactory.

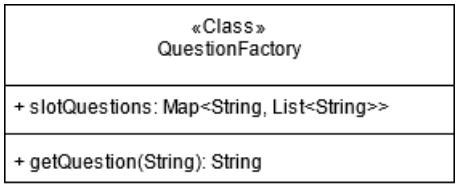
Trieda *Slot* obsahuje definíciu daného slotu a informácie o tom aká pomenovaná entita sa v danom slote nachádza. Informácie definujúce slot sú názov daného slotu (členská premenná *name*), informácie o tom či je daný slot povinný (členská premenná *required*), či je pre daný slot povinná pomenovaná entita, ktorá sa dá nájsť na mape (členská premenná *navigableNeeded*) a člen-

ská premenná *rules*. Premenná *rules* predstavuje pravidlá určujúce nakoľko pomenovaná entita patrí do daného slotu. Táto premenná je mapa, ktorá ako kľúč obsahuje spomínané pravidlá a ako hodnotu táto mapa obsahuje skóre, ktoré určuje nakoľko do tohto slotu daná pomenovaná entita patrí.

Trieda *Frame* obsahuje definíciu daného rámca a informácie o napĺňaní daného rámca. Definicia rámca pozostáva z členskej premennej *slots* a z členskej premennej *intent*, ktorá reprezentuje používateľský zámer, na ktorý je daný rámec viazaný. Sloty sú uložené v mape *slots*, kde kľúčom je názov daného slotu.

Rámec vie určiť, či má všetky potrebné informácie na vykonanie používateľovho zámeru alebo je ešte potrebné klásť doplňujúce otázky. Na tento účel slúži metóda *isFilled*, ktorá prechádza všetky povinné sloty (tie čo majú členskú premennú *required* nastavenú na *true*) a od každého zistí, či obsahuje nejakú pomenovanú entitu.

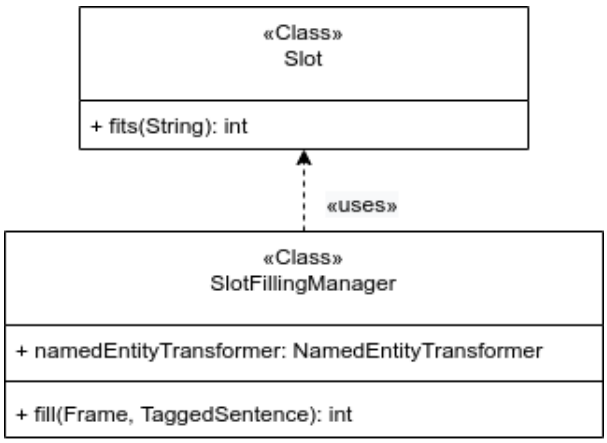
V prípade, že je potrebné klásť doplňujúce otázky, volá sa metóda *getQuestion*. Výstupom metódy *getQuestion* je znenie otázky na konkrétny slot v danom rámci. Názov slotu, na ktorý sa kládla otázka sa následne zapamätá do premennej *lastAsked*. Slot na ktorý sa bude klásť otázka si metóda *getQuestion* vyberá sama. V prípade, že sme na nejaký slot kládli otázku ale ešte stále nie je naplnený, na tento slot sa spýtame znovu. Ak nie tak sa spýtame na prvý nenaplnený slot ktorý sa v rámci nachádza. Metóda *getQuestion* v triede *Frame* kladenie otázky následne deleguje na triedu *QuestionFactory* (obrázok 4.13), ktorá má pre každý slot zoznam otázok. Pri zavolaní funkcie *getQuestion*, ktorá ako argument dostane názov slotu, *QuestionFactory* vyberie náhodnú otázku zo zoznamu otázok priradených danému slotu. Účelom vyberania náhodnej otázky je simulácia ľudskej konverzácie a prirodzeného chodu dialógu.



Obr. 4.13: Triedny diagram pre triedu *QuestionFactory*.

Napĺňanie slotov

Na to aby bola Mia schopná vykonať používateľov zámer potrebuje vyextrahovať všetky potrebné informácie z používateľovho vstupu a následne ich naplňiť do slotov. Na získavanie relevantných informácii a ich následné napĺňanie do slotov slúži trieda *SlotFillingManager*(obrázok 4.14).



Obr. 4.14: Triedny diagram pre triedu *SlotFillingManager*.

Niektoré pomenované entity je potrebné pretransformovať. Táto transformácia sa týka pomenovaných entít, ktoré chceme vložiť do rámca na miesto navigovateľnej entity avšak táto pomenovaná entita ako taká navigovateľná nie je. Napríklad keď sa používateľ spýta ako sa dostať na Databázy(1) tento vstup patrí medzi používateľský zámer nájdenia cesty. Na nájdenie cesty ale treba pomenované entity ktoré sa dajú nájsť na mape, a keďže Databázy(1) sú vyučovací predmet na mape sa nenachádzajú. Je preto potrebné Databázy(1) pretransformovať na udalosť. Výstupom tejto transformácie je nová inštancia triedy *TaggedWord*, ktorá ako slovo obsahuje to isté slovo ako vstupná inštancia, avšak bude mať nové označenie a aj nový zoznma entít (členská premenná entities). Napríklad pre Databázy(1) bude pretransformovaná pomenovaná entita obsahovať zoznam udalosí viazaných na tento predmet. Keďže Databázy(1) majú viacero udalostí (prednáška, cvičenia) novo vzniknutá inštancia triedy *TaggedWord* nie je unikátna a teda používateľ dostane výber ktorú z týchto udalotí mal na mysli. Na účely tejto transformácie slúži trieda *NamedEntityTransformer*, ktorá pomocou metódy *transformEntity* pretransformuje vstupnú inštanciu triedy *TaggedWord* a vráti novú pre-

transformovanú inštanciu tejto triedy.

Samotné napĺňaňie slotov prebieha v dvoch fázach. V prvej fáze si pre každú dvojicu slotu a pomenovanej entity získame skóre, ktoré určuje nakoľko daná pomenovaná entita patrí do daného slotu. Toto skóre získame pomocou metódy *fits*, ktorá je implementovaná v triede *Slot*. Metóda *fits* na vstue dostane string a postupne prechádza zoznam pravidiel uložených v slote. Jej výstupom je maximálna hodnota z pravidiel, ktorým vstupný string vyhovoval. Keďže chceme pre každý slot získať najlepšiu možnú pomenovanú entitu takto vytvorenú maticu si vložíme do triedy *EntitySlotQueue*. Táto trieda je na základe tejto matice schopná pre každý slot vrátiť najlepšiu možnú entitu (metóda *getBestEntity*) a pre každú entitu najlepší možný slot (metóda *getBestSlot*).

V druhej fáze prechádzame tento front a jednotlivým slotom hľadáme najlepšie možné entity. Toto vyhľadávanie prebieha spôsobom, ktorý je zobrazený a popísaný nižšie(algoritmus 2).

Algorithm 2: Fáza 2 napĺňanie slotov

```
queue = nová inštancia triedy EntitySlotQueue vytvorená z matice
 slotov a pomenovaných entít;
changed = true;
while queue nie je prázdny a changed je true do
    changed = false;
    for slot in allSlots do
        bestEntity = queue.getBestEntity(slot);
        bestSlot = queue.getBestSlot(bestEntity);
        if slot == bestSlot then
            changed = true;
            slot.setValue(bestEntity);
            reserved.add(bestEntity);
            queue.removeEntity(bestEntity);
            queue.removeSlot(slot);
        end
    end
end
```

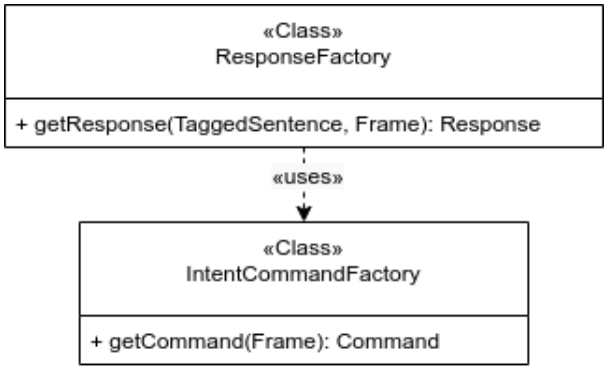
Pre každý slot získame najlepšiu možnú pomenovanú entitu. Pre túto entitu následne získame najlepší možný slot. Ak je získaný slot rovnaký ako aktuálny tak to znamená že daná entita najlepšie vyhovuje danému slotu. V takom prípade sa daná entita do tohto slotu vloží. Táto pomenovaná entita sa následne odstráni z frontu a pridá sa medzi rezervované entity. V prípade, že je náš front prázdny alebo sa žiaden zo slotov nezmenil, teda sme nenašli žiadnu pomenovanú entitu, ktorá by týmto slotom vyhovovala tento proces sa zastaví. Toto napĺňanie sa vykonáva v metóde *fill*, výstupom ktorej je počet slotov, ktoré sa úspešne naplnili.
V prípade, že nie všetky povinné sloty sú naplnené, Mia používateľovi kladie doplňujúce otázky na chýbajúce informácie a tento proces sa opakuje od časti Rozoznávanie pomenovaných entít. Ak je rámec naplnený, Mia prechádza do konečného štádia a to vykonávanie používateľského zámeru.

4.5 Vykonanie akcie zodpovedajúcej používateľovmu zámeru

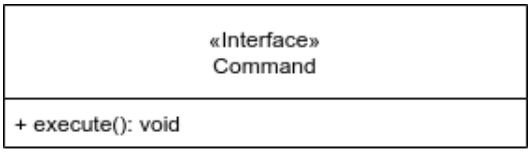
V tejto podkapitole sa budem venovať spôsobu ako Mia po naplnení rámca vykonáva používateľov zámer.

Ak má Mia k dispozícii všetky informácie potrebné na vykonanie používateľovho zámeru vygeneruje odpoveď pre daný používateľský zámer. Generovanie odpovedí prebieha v triede *ResponseFactory*(orbázok 4.15), ktorá pomocou metódy *getResponse* vygeneruje pre vstupný používateľský zámer vhodnú odpoveď. Výstupom tejto metódy je inštancia triedy *Response*. Táto trieda obsahuje text správy aj príkaz, ktorý sa po kliknutí na túto správu má vykonať.

Príkazy, ktoré obsahuje trieda *Response* sú inštancie tried implementujúcich interface *Command* (obrázok 4.16). Interface *Command* má jednu metódu a to *execute*, ktorá vykoná daný príkaz. Tieto príkazy sa generujú v triede *IntentCommandFactory* pomocou metódy *getCommand*, ktorá na vstupe dostane naplnený rámec a jej výstupom je inštancia triedy *Command* zodpovedajúca danému používateľskému zámeru.
V prípade, že používateľským zámerom je získať odpoveď na kladenú otázku, Mia jednoducho vypíše odpoveď vygenerovanú triedou *ResponseFactory* a po



Obr. 4.15: Triedny diagram pre triedu *ResponseFactory*.



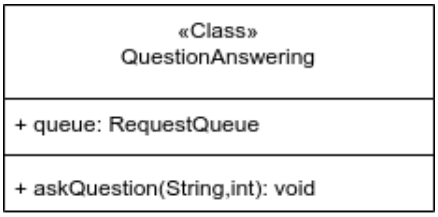
Obr. 4.16: Triedny diagram pre interface *Command*.

kliknutí na túto správu sa nemá vykonať žiadna ďalšia akcia.
Ak bol používateľov zámer ovládať existujúce funkcionality Sprievodcu FMFI Mia si najprv získa príkaz zodpovedajúci konkrétnemu zámeru (pomocou triedy *IntentCommandFactory*) a až tak vypíše odpoveď. Vypísaná odpoveď má priradený daný príkaz a po kliknutí na ňu sa daný príkaz vykoná.

4.6 Odpovedanie na otázky bez používateľského zámeru

V prípade, že sa Mii nepodarí určiť používateľov zámer používateľovu správu odošle na server. Odosielanie používateľovej správy na server a následné prijatie odpovedi prebieha v triede *QuestionAnswering* (obrázok 4.17).

Trieda *QuestionAnswering* si pri inicializácii vytvorý novú inštanciu triedy *RequestQueue*, ktorú vloží do členskej premennej *queue*. Premenná *queue* slúži ako fronta na posielanie žiadostí na vzdialený server. Vytvorenie žiadosti a následné vloženie do fronty sa vykonáva v metóde *askQuestion*. Táto metóda ako vstup dostane samotný text používateľovej správy a taktiež aj

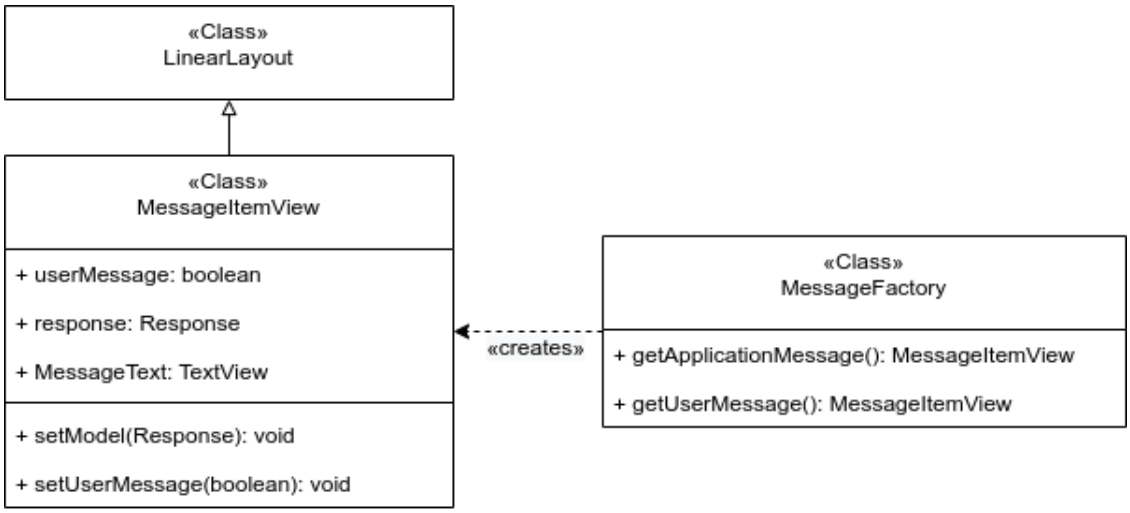


Obr. 4.17: Triedny diagram pre interface *QuestionAnswering*.

poradové číslo tejto správy. Účelom poradového čísla je že v prípade, že používateľ naísal inú správu skôr ako stihol server odpovedať Mia nebude dodatočne písať odpoveď od serveru. Na základe používateľovho vstupu Mia vytvorí požiadavku na server, ktorú následne vloží do fronty požiadaviek (členská premenná *queue*). Ak nastala chyba v komunikácii so serverom Mia používateľovi vypíše správu že server nie je k dispozícii. V opačnom prípade Mia čaká na odpoveď zo serveru. Keď dostane odpoveď skontroluje či nie je odpoveď prázdny reťazec (teda ani server nepozná odpoveď) a v takom prípade Mia odpovie, že na tento vstup nepozná odpoveď. V opačnom prípade Mia používateľovi vypíše odpoveď, ktorú od serveru získala.

4.7 Grafické používateľské rozhranie

V tejto sekcii sa budem venovať detailom spojeným s androidom. Po kliknutí na tlačidlo ASK ME sa spustí nová aktivita. Táto aktivita je reprezentovaná triedou *ConversationActivity*. Rozloženie tejto aktivity je uložené v xml dokumente. *ConversationActivity* obsahuje 3 časti a to vstupný riadok, tlačidlo na odoslanie správy a zoznam správ v konverzácii. Každá poslaná správa je inštancia triedy *MessageItemView* (obrázok 4.18), ktorá má informácie o tom, či bola daná správa poslaná aplikáciou alebo používateľom, ďalej obsahuje samotný text správy a prípadne aj príkaz, ktorý sa po kliknutí na správu má vykonať. Na vytváranie inštancii triedy MessageItemView slúži MessageFactory, ktorá ma metódy na vytvorenie správy od používateľa a správy od aplikácie. Po získaní inštancie triedy *MessageItemView* Mia nastaví samotný text a prípadne aj príkaz, ktorý sa má vykonať. Toto nastavovanie prebieha pomocou metódy *setModel*.



Obr. 4.18: Triedny diagram pre triedu *MessageItemView*.

Kapitola 5

Serverová časť z technického hľadiska

Server, s ktorým aplikácia Sprievodca FMFI komunikuje využíva Mia na dva rôzne účely. Mia na tento server v prípade potvrdenia používateľom odosiela konverzácie za účelom budúceho rozvoja a zdokonalovania. Taktiež na tento server Mia kladie otázky, na ktoré nevie reagovať. V nasledujúcich podkapitolách sa budem venovať jednotlivým funkcionalitám, ktoré som pridal.

5.1 Ukladanie konverzácii

V tejto sekcii sa budem venovať štruktúre, v akej sa konverzácie odosielajú a následnému ukladaniu do databázy.

Štruktúra konverzácii

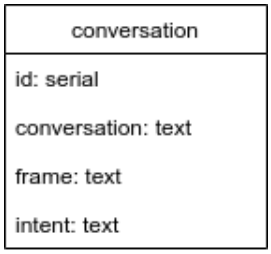
Samotnú konverzáciu Mia odosiela ako parameter conversation v POST požiadavke. V tomto parametri sú správy v takom poradí v akom boli poslané a sú oddelené unikódovým znakom \u241E, ktorý slúži na odelovanie záznamov [6]. Taktiež sú jednotlivé správy označené, podľa toho či ich poslal používateľ alebo aplikácia. Keďže som nenašiel unikódový znak určený na označoanie používateľských správ alebo správ od aplikácie pre používateľove správy som využil unikódový znak \u2406, ktorý predstavuje potvrdenie (ACK)[6] a pre správy od aplikácie som využil unikódový znak \u2415, ktorý predstavuje negatívne potvrdenie (NAK) [6]. Konverzácia teda bude

mať nasledovnú štruktúru:
\u2406How do I get to A?\u241E\u2415From where would you like to find the path?\u241E\u2406From B\u241E\u2415To see path from B to A please click on this message.\u241E

Ako ďalší parameter POST požiadavky je používateľov zámer, ktorý Mia zdetekovala, v prípade vyššie by to bolo FindPathIntent. Posledným parametrom je naplnený rámec obsahujúci názov slotu a hodnotu, ktorá mu bola priradená. V príklade konverzácie vyššie by to bolo src->B;dst->A. Všetky tieto informácie sú dôležité prípade, že by sa v budúcnosti určovanie používateľského zámeru riešilo pomocou strojového učenia.

Ukladanie konverzácii do databázy

Ked dostane server požiadavku na uloženie konverzácie jednotlivé parametre tejto požiadavky uloží do databázy. Pre ukladanie konverzácii som zvolil jednoduchý dátový model pozostávajúci iba z jednej tabuľky (obrázok 5.1).

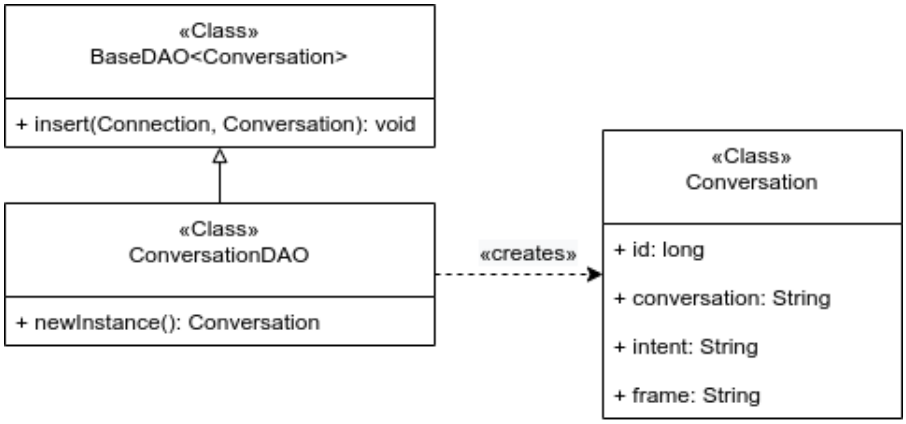


Obr. 5.1: Diagram pre tabulku *conversation*.

Stĺpec conversation reprezentuje samotnú konverzáciu medzi používateľom a Miou, stĺpec frame je textová reprezentácia vyplneného rámca, ktorá obsahuje názvy jednotlivých slotov a hodnoty, ktoré do nich boli vyplnené. Stĺpec intent predstavuje používateľský zámer, ktorý v tejto konverzácii Mia detekovala.

Na pristupovanie do databázy používa server prístup Data Access Object (DAO)[8]. V tomto prístupe je každý riadok reprezentovaný samostatnou triedou a každá tabuľka má svoju DAO triedu, ktorá pracuje s jednotlivými

riadkami. Pri ukladaní konverzácii je riadok reprezentovaný triedou *Conversation* a DAO trieda sa volá *ConversationDAO*(obrázok 5.2)



Obr. 5.2: Diagram pre tabulku *ConversationDAO*.

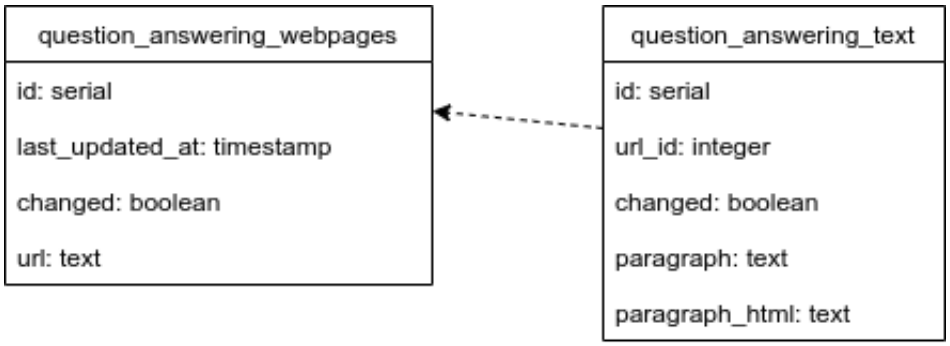
Najprv sa pomocou metódy *newInstance* vytvorý nová inštancia triedy *Conversation*. Tejto inštancii následne nastavíme hodnoty členských premenných conversation, intent a frame pomocou hodnôt získaných z požiadvky na server. Následne zavoláme metódu *insert*, ktorej ako argument dáme prístup do databázy a inštanciu triedy *Conversation*, ktorú sme vytvorili. Metóda *insert* vstupnú inštanciu vloží do databázy a tým je proces ukladania konverzácie ukončený.

5.2 Odpovedanie na otázky

V tejto časti sa budem venovať spôsobu akým server vyhľadáva odpovede na kladené otázky. Vyhľadávanie odpovedí na otázky prebieha pomocou prístupu zameraného na hľadanie odpovedí v texte (information retrieval). Aby sme vedeli tento prístup aplikovať potrebujeme v prvom rade text, v ktorom budeme odpovede vyhľadávať. Na získavanie tohto textu slúži trieda *FmphCrawler*, ktorá prechádza všetky stránky začínajúce prefixom *fmph.uniba.sk/en/*, vyberá z nich jednotlivé odstavce a vkladá ich do databázy. V takto získaných odstavcoch vieme následne vyhľadávať odpovede na kladené otázky.

Dátový model

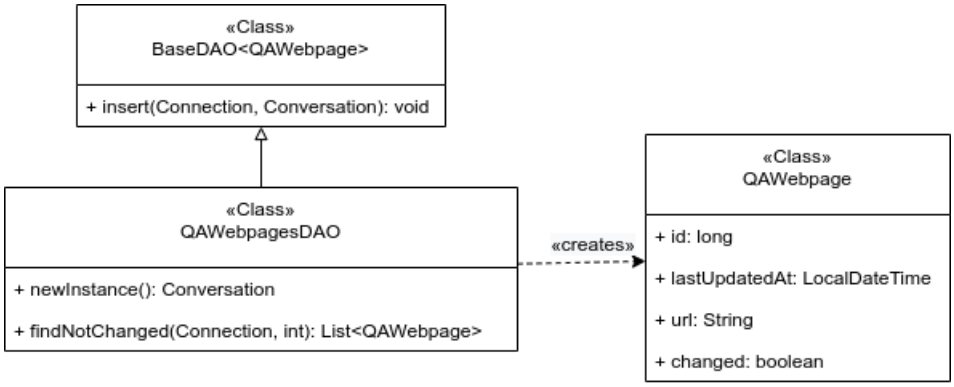
Na tento problém som zvolil dátový model zložený z dvoch tabuliek a to *question_answering_webpages* a *question_answering_text* (obrázok 5.3)



Obr. 5.3: Dátový model pre zbieranie infromácii.

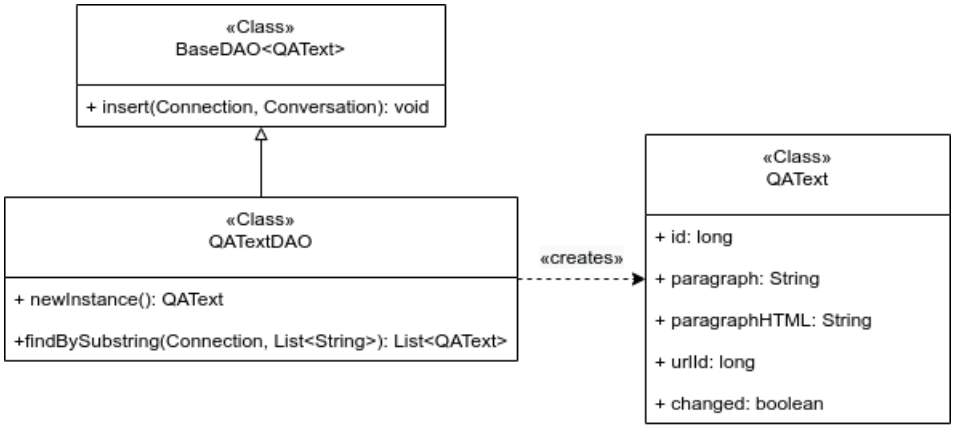
Tabuľka *question_answering_webpages* obsahuje jednotlivé stránky, ktoré *FmphCrawler* navštívil alebo má v pláne navštíviť. Stĺpec *changed* uchováva informáciu, či danú stránku crawler navštívil. Stĺpec *last_updated_at* má informáciu kedy naposledy bola táto stránka navštívená. Stĺpce *changed* a *last_updated_at* slúžia na detekovanie neaktívnych stránok, ktoré sa následne z databázy odstránia. Tabuľka *question_answering_text* obsahuje odstavce, ktoré boli na konkrétnej stránke nájdené. Stĺpec *paragraph* obsahuje čistý text tohto odstavca a *paragraph_html* obsahuje pôvodný odstavec aj so všetkými jeho html tagmi. Odpoveď na kladenú otázku sa vyhľadáva v stĺpci *paragraph* a ako odpoveď sa následne odošle jeho html reprezentácia (*paragraph_html*), kvôli zachovaniu pôvodnej štruktúry odstavca. Rovnako ako pri ukladaní konverzácii aj tu sa na pristupovanie do databázy využíva prístup DAO[8]. Na prístup ku stránkam sa využíva DAO trieda *QAWebpagesDAO* (obrázok 5.4). Jednotlivé riadky sú reprezentované triedou *QAWebpage*, ktorej členské premenné sú rovnaké ako stĺpce v tabuľke.

Aby sme vložili novú webstránku najprv si získame inštanciu triedy *QAWebpage* pomocou metódy *newInstance*. Takto vytvorenej inštancii následne nastavíme jej url. Dátum poslednej zmeny nastavíme na null a changed nastavíme na false. Túto inštanciu spolu s databázovým pripojením následne pošleme ako argumenty metóde *insert*. Výstupom metódy *findNotChanged* je zoznam webstránok, ktoré ešte neboli navštívené, teda hodnota *changed* je *false*.



Obr. 5.4: Triedny diagram pre triedu QAWebpagesDAO.

Na prácu s odstavcami (tabuľka *question_answering_text*) slúži DAO trieda *QATextDAO* (obrázok 5.5). Jednotlivé riadky sú reprezentované triedou *QAText*, ktorej členské premenné sú rovnaké ako stĺpce v tabuľke.

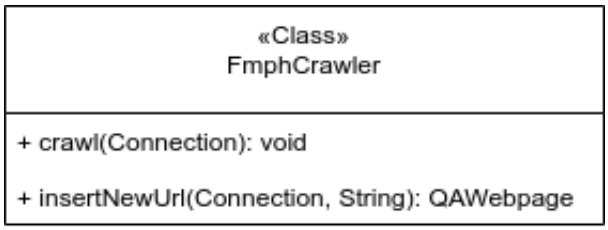


Obr. 5.5: Triedny diagram pre triedu QATextDAO.

Vytváranie a vkladanie nového odstavca do databázy prebieha obdobne ako pri vytváraní novej stránky. Metóda *findBySubstring* dostane na vstupe pripojenie na databázu a zoznam podreťazcov. Výstupom tejto metódy je zoznam odstavcov, ktoré ako podreťazec obsahovali aspon jeden reťazec zo vstupného zoznamu reťazcov, pričom na poradí podreťazcov nezáleží.

Trieda *FmphCrawler*

Trieda *FmphCrawler* (obrázok 5.6) na získavanie informácii využíva stránky začínajúce prefixom *fmph.uniba.sk/en/*. Získavanie infromácii prebieha pomocou metódy *crawl*, ktorá na vstupe dostane pripojenie na databázu. V prípade, že je databáza webstránok prázdna vloží sa do databázy koreňová stránka fmph.uniba.sk/en/.

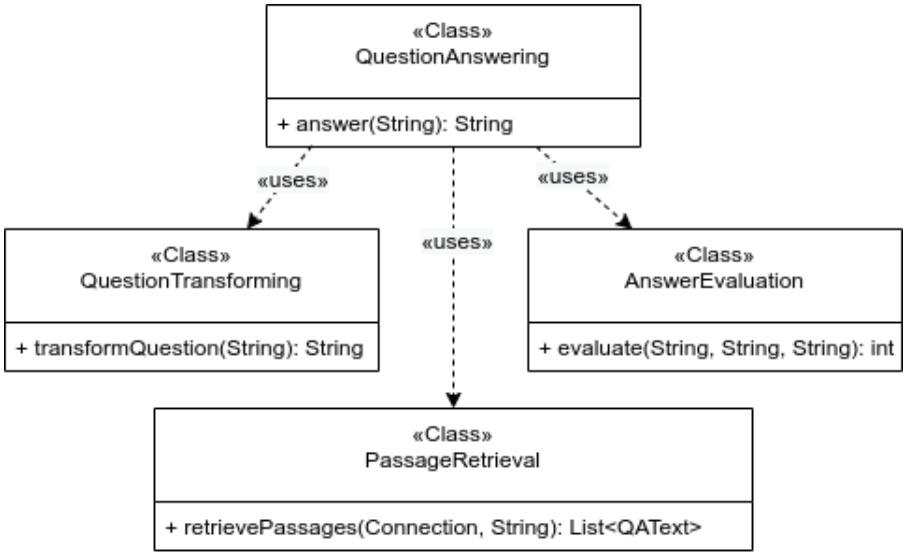


Obr. 5.6: Triedny digram pre triedu *FmphCrawler*.

Pred začatím prechádzania si metóda *crawl* vytvorí pomocnú množinu, do ktorej bude vkladať url adresy, ktoré na prechádzaných webstránkach nájde. Prechádzanie začína vybratím prvých n webstránok z databázy, ktoré ešte neboli prejdené pomocou metódy *findNotChanged*. Pre každú webstránku sa získa jej obsah, z ktorého sa validné url pridajú do pomocnej množin a zároveň sa do databázy uložia všetky odstavce nachádzajúce sa na tejto stránke. Po tom ako sa táto stránka spracuje sa jej v databáze stĺpec *changed* nastavý na true a stĺpec *last_ updated_ at* sa nastaví na aktuálny dátum a čas. Po prejdení všetkých n webstránok sa tento proces skončí. Pomocnú množinu webstránok následne prechádzame a pomocou metódy *insertNewUrl* danú webstránku vložíme do databázy. Metóda *insertNewUrl* najprv skontroluje, či sa v databáze nenachádza webstránka s rovnakou url adresou. Ak nie tak si metóda získa novú inštanciu triedy *QAWebpage*, ktorej nastaví url a pomocou QAWebpagesDAO túto inštanciu vloží do databázy. Tento proces sa opakuje každých 20 minút.

Hľadanie odpovedí na kladené otázky

Samotné vyhľadávanie odpovedí na kladené otázky prebieha v triede *QuestionAnswering* (obrázok 5.7)



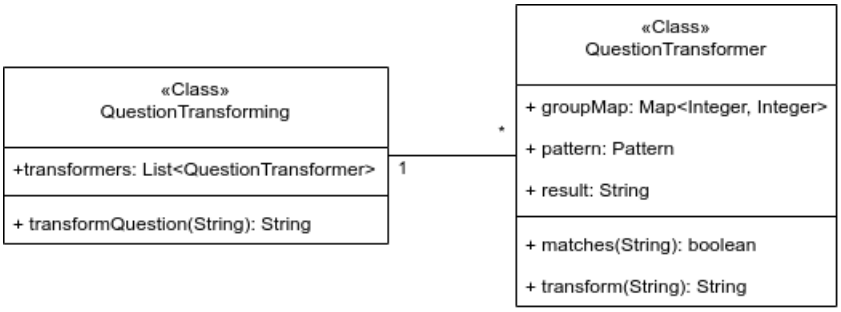
Obr. 5.7: Triedny diagram pre triedu *QuestionAnswering*.

Trieda *QuestionAnswering* má jedinú metódu a to *answer*, ktorá na vstupnú otázku vráti najlepšiu možnú odpoveď. Vyhľadávanie odpovede na kladenú otázku prebieha v 4 krokoch:

- Kladená otázka sa pretransformuje na oznamovaciu vetu pomocou triedy *QuestionTransforming*
- Pomocou triedy *PassageRetrieval* sa nájdu všetky odstavce, ktoré obsahujú ako podreťazec nejakú časť pretransformovanej vety.
- Po získaní odstavcov sa jednotlivé odstavce ohodnotia pomocou triedy *AnswerEvaluation*.
- Na záver sa vyberie odstavec s najvyšším ohodnotením a vráti sa jeho html reprezentácia.

Trieda *QuestionTransforming*

Trieda *QuestionTransforming* pretransformuje vstupnú otázku na oznamovaciu vetu, ktorej slová sa následne vyhľadávajú v jednotlivých odstavcoch.



Obr. 5.8: Triedny diagram pre triedu *QuestionTransforming*.

Transformovanie vstupnej otázky je realizované pomocou zoznamu inštancii triedy *QuestionTransformer*. Každá inštancia triedy *QuestionTransformer* obsahuje vzor (členská premenná *pattern*), pomocou ktorého je schopná rozoznať, či vie vstupný reťazec pretransformovať (metóda *matches*). členská premenná *groupMap* obsahuje dvojicu čísel, kde prvé číslo predstavuje číslo skupiny vo vzore a druhé číslo predstavuje na aké miesto sa má táto skupina vložiť vo výstupnom reťazci(členská premenná *result*). V prípade, že je daná inštancia triedy *QuestionTransformer* schopná vsupnú otázku pretransformovať zavolá sa jej metóda *transform*, ktorá na základe členskej premennej *groupMap*, *pattern* a *result* pretransformuje vstupnú otázku.

Trieda *PassageRetrieval*

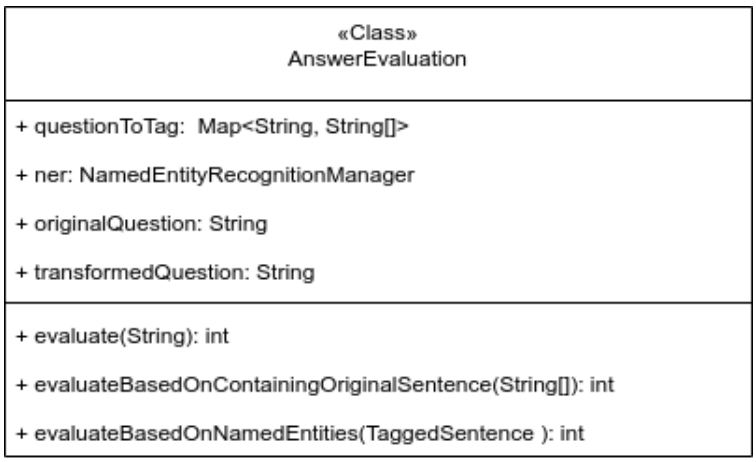
Trieda *PassageRetrieval* slúži na získanie všetkých odstavcov, ktoré pravdepodobne obsahujú odpoveď na kladenú otázku.

Trieda *PassageRetrieval* obsahuje jedinú metódu a to *retrievePassages*, ktorá na vstupe dostane pripojenie na databázu a pretransformovaný používateľov vstup. Pretransformovaný vstup si rozdelí podĺa medzier a odstráni z neho všetky predložky, slovesá a zámená. Takto získaný zoznam slov následne pošle spolu s pripojením na databázu ako argument metóde *findBySubstring*, ktorá sa nachádza v triede *QATextDAO*. Výstup tejto metódy je taktiež aj

výstupom metódy *retrievePassages*, čiže zoznam odstavcov, ktoré obsahujú aspoň nejaké slovo nachádzajúce sa v kladenej otázke s výnimkou slovies, zámen a predložiek.

Trieda *AnswerEvaluation*

Trieda *AnswerEvaluation*(obrázok 5.9) slúži na ohodnotenie jednotlivých odstavcov získaných pomocou triedy *PassageRetrieval*.



Obr. 5.9: Triedny diagram pre triedu *AnswerEvaluation*.

Trieda *AnswerEvaluation* pomocou metódy *evaluate* ohodnotí konkrétny odstavec. Jednotlivé odstavce sa ohodnocujú na základe 2 kritérii:

- Prvým kritériom je koľko slov z pôvodnej vety tento odstavec obsahuje. Toto ohodnocovanie prebieha v metóde *evaluateBasedOnContainingOriginalSentence*, ktorá na vstupe dostane zoznam slov z odstavcu. Metóda postupne prechádza slová pretransformovanej otázky (členská premenná *transformedQuestion*) a v prípade, že sa toto slovo nachádzalo v odstavci tak sa do výstupu pripočíta čiastkové skóre. Čiastkové skóre je vyrátané z podielu maximálneho skóre za slová z pôvodnej vety a dĺžky pôvodnej vety. Výstupom tejto metódy je skóre, ktoré je menšie nanajvýš rovné maximálnemu skóre za slová z pôvodnej vety.

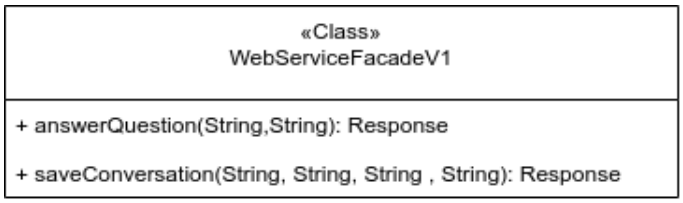
- Druhým kritériom je počet pomenovaných entít, na ktoré sa používateľ pýta. V prípade, že sa používateľ pýtal otázku začínajúcu slovom who,

odstavce obsahujúce osoby majú vačšiu váhu ako tie, ktoré žiadne osoby neobsahujú. To, ku ktorému opytovaciemu zámenu patrí ktoré označenie pomenovanej entity sa nachádza v mape *questionToTag*, kde kľúčom je opytovacie zámeno a hodnoutou je zoznam označení, ktoré vyhovujú danému zámenu. Ohodnocovanie podľa druhého kritéria prebieha v metóde *evaluateBasedOnNamedEntities*, ktorá na vstupe dostane inštanciu triedy TaggedSentence, ktorá predstavuje zoznam pomenovaných entít nachádzajúcich sa v odstavci, ktorý momentálne ohodnocujeme. Získavanie zoznamu pomenovaných entít zo vstupného reťazca prebieha pomocou triedy *NamedEntityRecognitionManager*, ktorá má rovnakú funkcionalitu ako trieda *NamedEntityRecognitionManager*, ktorú som spomínal vyššie (sekcia 4.2). Zoznam pomenovaných entít pre jednotlivé odstavce nie je možné mať uložené v databáze nakoľko pomenované entity sa priebežne pridávajú a menia. Podľa mapy *questionToTag* následne určíme aké pomenované entity by mala odpoveď obsahovať. V prípade, že vstupná otázka obsahuje slovo who odpoveď by mala obsahovať pomenovanú entitu s označením person. Ak vstupná otázka obsahuje slovo when odpoveď by mala obsahovať pomenovanú entitu s označením date alebo time alebo interval.

Na záver sa spraví suma ohodnotení za jednotlivé kritéria. Ak táto suma neprekračuje určitý prah tak metóda *evaluate* vráti 0, inak vráti túto sumu.

5.3 Trieda *WebServiceFacadeV1*

Samotné spracovávanie HTML požiadaviek na server manažuje trieda *WebServiceFacadeV1* (obrázok 5.10).



Obr. 5.10: Triedny diagram pre triedu *WebServiceFacadeV1*.

Metóda *answerQuestion* dostane na vstupe parametre, ktoré boli odoslané v POST požiadavke. Táto metóda následne zavolá metódu *answer* v triede *QuestionAnswering*, a ako vstup jej pošle reťazec, ktorý získala z POST požiadavky. Metóda *answerQuestion* vráti inštanciu triedy Response, ktorá ako správu bude obsahovať výstup metódy *answer*.

Metóda *saveConversation* dostane na vstupe parametre, ktoré boli odoslané v POST požiadavke. Táto metóda následne zavolá uloží vstupnú konverzáciu spôsobom, ktorý som popísal vyššie (sekcia 5.1). Táto metóda následne vráti inštanciu triedy Request s prázdnym telom správy.

Záver

V mojej práci som sa venoval rozširovaniu existujúcej aplikácie Sprievodca FMFI o konverzačné rozhranie. Aplikácia Sprievodca FMFI je mobilná aplikácia, ktorá umožnuje používateľovi zobrazovať si na mape miestnosti, osoby alebo cestu medzi miestnosťami alebo osobami. Taktiež umoznuje používateľovi zobraziť si informácie o miestnostiach a osobách pôsobiachich na fakulte matematiky, fyziky a informatiky.

Na využívanie aplikácie Sprievodca FMFI je potrebné sa v menu aplikácie preklikávať. Niektorým používateľom však môže byť jednoduchšie napísať čo chcú nájsť namiesto toho aby sa preklikávali. Na tieto účeli sme sa rozhodli do aplikácie Sprievodca FMFI pridať konverzačné rozhranie, do ktorého môže používateľ svoje požiadavky jednoducho napísať.

Novo pridané konverzačné rozhranie sa dá využiť nie len na ovládanie už existujúcich funkcionalít ale aj na zisťovanie odpovedí na konkrétne používateľské otázky. Konverzačné rozhranie voči používateľovi vystupuje pod menom Mia. Na to aby Mia fungovala potrebuje 4 funkčné podčasti a to rozoznávanie pomenovaných entít, určovanie používateľského zámeru, napĺňanie slotov a vykonanie akcie zodpovedajúcej používteľskému zámeru.

Pomenované entity si vieme rozdeliť na tie, ktoré súvisia s fakultou matematiky, fyziky a informatiky a tie, ktoré s fakultou nesúvisia. Pomenované entity, ktoré súvisia s fakultou sú napríklad osoby, mietnosti, účely miestností a vyučovacie predmety. Tieto pomenované entity sa vyhľadávajú na základe databázy, ktorú má Mia k dispozícii. Pomenované entity nesúvisiace s fakultou ako napríklad dátum, čas alebo interval sa rozoznávajú pomocou ručne písaných pravidiel. Rozoznávanie pomenovaných entít je spravené modulárne. Pre každú pomenovanú entitu je samostatný rozoznávač, ktorý vie určiť či vstupná entita spadá do danej kategórie pomenovaných entít. Vďaka

tomu je jednoduché pridať novú skupinu pomenovaných entít.

Z používateľovho vstupu a zoznamu pomenovaných entít je Mia schopná určiť čo od nej používateľ požaduje - používateľov zámer. Rozoznávanie používateľského zámeru je realizované pomocou ručne písaných pravidiel a skóre, ktoré majú jednotlivé pravidlá priradené. Podobne ako rozoznávanie pomenovaných entít aj určovanie používateľského zámeru je naprogramované modulárne. Každý používateľský zámer, ku ktorému Mia pozná akciu, ktorú je potrebné vykonať má samostatný rozoznávač. Ako výsledok je pridanie nového používateľského zámeru relatívne jednoduché.

Vďaka tomu, že je Mia schopná určiť používateľov zámer vie následne zistiť aké informácie od používateľa na vykonanie tohto zámeru potrebuje. Každý používateľský zámer má priradený svoj rámec, ktorý pozostáva zo slotov. Kažý slot predstavuje informáciu, ktorá je potrebná na vykonanie daného zámeru. Sloty sa napĺňajú na základe používateľovho vstupu. Ak Mia nie je schopná získať všetky potrebné informácie z používateľovho vstupu tak na chýbajúce informácie používateľovi kladie doplňujúce otázky a tento proces sa opakuje od rozoznávania pomenovaných entít.

V prípade, že má Mia všetky informácie, ktoré potrebuje na vykonanie používateľovho zámeru, vypíše používateľovi finálnu správu. Po kliknutí na túto správu Mia vykoná používateľov zámer a používateľovi sa zobrazí výsledok. Ak Mia nevie čo s používateľovým vstupom robiť odošle ho na server, z ktorého následne dostane odpoveď na používateľov vstup.

Na serverovú aplikáciu Sprievodcu FMFI sme pridali možnosť ukladať konverzácie, ktoré prebehli medzi používateľom a Miou. Taktiež sme pridali možnosť odpovedať na používateľove otázky týkajúce sa fakulty matematiky, fyziky a informatiky. Odpoveď na tieto otázky sa vyhľadáva v databáze odstavcov získaných na stránke fmph.uniba.sk/en/. Na získanie odpovede na používateľve otázky sa najprv používateľova otázka pretransformuje na oznamovaciu vetu pomocou ručne písaných pravidiel. Z takto pretransformovanej otázky sa následne odstránia všetky predložky spojky a slovesá pre presnejšie vyhľadávanie. Následne sa v databáze vyhľadajú všetky odstavce obsahujúce ako podreťazec niektoré zo slov, ktoré zostali v pretransformovanej vete. Tieto odstavce sa následne ohodnotia na základe niekoľkých kritérii a server ako odpoveď vráti odstavec s najvyšším skóre.

Literatúra

[1] Martin Bohumel, Mobilný Sprievodca FMFI, Univerzita Komenského, 2017.

[2] Linda Jurkasová, Sprievodca FMFI, Univerzita Komenského, 2020.

[3] Daniel Jurafsky, James H. Martin, Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistic, and Speech Recognition Third Edition draft, 2019.

[4] Daniel G. Bobrow, Ronald M. Kaplan, Martin Kay, Donald A. Norman, Henry Thompson and Terry Winograd, GUS, A Frame-Driven Dialog System, dostupné na https://nlp.stanford.edu/acvogel/gus.pdf, citované v 2020.

[5] Amazon, Amazon Lex: Developer Guide, 2020.

[6] Unicode Symbol - Control Pictures[online], dostupné na: www.unicode-symbol.com/block/Control_Pictures.html, citované v 2020.

[7] Alexander Šimko et al. Zdrojové kódy aplikácie Sprievodca FMFI. 2020.

[8] Oracle. Core J2EE Patterns - Data Access Object. Dostupné online: www.oracle.com/java/technologies/dataaccessobject.html , citované v 2021.