FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Automatická oprava zjednodušeného relačního zápisu
Student:	Filip Machala
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Pro předmět BI-DBS je provozován nástroj pro celkovou podporu výuky dbs.fit.cvut.cz. V rámci tohoto portálu probíhá testování studentů, přičemž podstatnou částí testování je transformace konceptuálního modelu do zjednodušeného relačního zápisu. Oprava těchto transformací je pro vyučující časově velmi náročná.

Cílem této práce je navrhnout a následně realizovat vhodné rozšíření portálu o "normalizaci" jednotlivých studentských řešení do jednotné formy pro snadné opravování otázek typu transformace. Dále navrhnout a realizovat další postupy usnadňující hodnocení daného typu otázek.
Výstupem práce bude tedy ucelený návrh spolu s funkčním řešením ideálně ve formě služby na serverové straně komunikující pomocí vhodného API.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 16. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Automatická oprava zjednodušeného relačního zápisu

Filip Machala

Katedra softwarového inženýrství
Vedúci práce: Ing. Jiří Hunka

15. mája 2018

Poďakovanie

Chcel by som sa poďakovať pánovi Ing. Jiřímu Hunkovi za jeho trpezlivosť a odborné vedenie mojej práce. Ďalej by som sa chcel poďakovať mojej rodine, ktorá ma podporovala počas celého môjho štúdia. Nakoniec by som sa chcel poďakovať všetkým, ktorý ma akoukoľvek formou podporovali počas ťažkých časov doterajšieho štúdia.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k užívaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo užívať.

Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela, a za akýmkoľvek účelom (vrátane komerčného využitia). Toto oprávnenie je časovo, územne a množstevne neobmedzené. Každá osoba, ktorá využije vyššie uvedenú licenciu, sa však zaväzuje priradiť každému dielu, ktoré vznikne (čo i len čiastočne) na základe Diela, úpravou Diela, spojením Diela s iným dielom, zaradením Diela do diela súborného či zpracovaním Diela (vrátane prekladu), licenciu aspoň vo vyššie uvedenom rozsahu a zároveň sa zaväzuje sprístupniť zdrojový kód takého diela aspoň zrovnateľným spôsobom a v zrovnateľnom rozsahu ako je zprístupnený zdrojový kód Diela.

V Prahe 15. mája 2018 .....

Odkaz na túto prácu

Machala, Filip. *Automatická oprava zjednodušeného relačního zápisu.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Cieľom tejto bakalárskej práce bolo navrhnúť a implementovať riešenie pre zautomatizovanie opravy odpovedí typu transformácia konceptuálneho modelu do zjednodušeného relačného zápisu. Pre tento účel som navrhol a následne implementoval samostantnú aplikáciu, ktorá využíva možnosti jazyka Python a jeho knižníc. Aplikácia realizuje prevod odpovedí do štandardizovaného formátu JSON a ich následnú opravu. Pre komunikáciu používa rozhranie API. Táto bakalárska práca sa dotýka všetkých krokov softwarového vývoja: analýzy, návrhu, implementácie a testovania.

Klíčová slova Databázové systémy, lexikálna analýza, syntaktická analýza, parsovanie, API, PLY, Flask, zjednodušený relačný zápis, JSON

Abstract

The main purpose of this bachelor's thesis was to design and implement solution for automatization of correction answers of type transformation to simplified relational notation. I have designed and implemented application for this purpose, that use possibilities of Python programming language and it's libraries. Application also converts answers to standardized JSON format. It

communicates through API. This bachelor's thesis include all steps of software development: analysis, design, implementation and testing.

Keywords Database systems, lexical analysis, syntactic analysis, parsing, API, PLY, Flask, simplified ralational notation, JSON

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Informačné systémy sú dnes bežnou súčasťou nášho života. Sú to podporné nástroje, ktorých hlavnou úlohou je uschovávať a spracovávať dáta čím nám šetria množstvo času. Informačný systém portál dbs.fit.cvut.cz sa používa na výuku predmetu databázové systémy na Fakulte informačných technológií Českého vysokého učení technického v Praze.

Tento systém študentov prevedie celým predmetom. Študenti v ňom vytvárajú a odovzdávajú svoje semestrálne práce, píšu testy v priebehu semestra a takisto záverečnú skúšku. Súčasťou tohto predmetu, je konceptuálne modelovanie databázy a následný prevod konceptuálneho modelu do zjednodušeného relačného zápisu. Táto znalosť je od študentov vyžadovaná aj v testoch, načo je zameraná otázka typu tranformácia konceptuálneho modelu do zjednodušeného relačného zápisu. Manuálna oprava týchto otázok, je pre učiteľov časovo veľmi náročná. Preto tím, ktorého som bol v rámci predmetov Softwarový týmový projekt 1 a 2 súčasťou, začal pod vedením pána Ing. Jiřího Hunku vyvíjať nástroje pre zjednodušenie opravy. Táto téma ma veľmi zaujala a keďže výsledná verzia, ktorá vznikla v rámci týchto predmetov, nespĺňala požiadavky od nej očakávané, som sa rozhodol v tejto téme pokračovať v rámci tejto bakalárskej práce a navrhnúť nové riešenie.

V tejto práci sa najprv venujem analýze existujúceho riešenia, na základe ktorej navrhujem nové riešenie. Implementáciu navrhnutého riešenia popisujem v nasledujúcej kapitole. V závere práce sa venujem testovaniu implementovanej aplikácie a jeho ďalších možností pre rozšírenie v budúcnosti.

Kapitola 1

Analýza

1.1 Zavedenie pojmov

Pre správne pochopenie problematiky parsovania textu je potrebné zaviesť tieto základné pojmy:

Regulárna gramatika je taká gramatika, v ktorej platí, že každé pravidlo má tvar $A \rightarrow \alpha\mathtt{B}$ alebo $A \rightarrow \alpha$, kde A, B patria do množiny neterminálnych symbolov a α je terminálny symbol, alebo tvar $S \rightarrow \epsilon$, v prípade, že S sa nevyskytuje na pravej strane žiadneho pravidla. [1]

Lexikálna analýza je proces, ktorý prevádza postupnosť znakov na postupnosť symbolov tzv. tokenov. Lexikálna analýza takisto zo vstupného textu odstráni nepotrebné znaky, ako sú napríklad medzery, tabulátory a pododbne. [2] Token sa skladá z textu a identifikátora typu tokenu.

Lexikálny analyzátor alebo **lexer**, je vo svojej podstate konečný automat, ktorý realizuje lexikálnu analýzu. [2]

Syntaktická analýza alebo tiež parsovanie, je proces analýzy tokenov spracovaných lexikálnou analýzou s cieľom určiť ich gramatický význam. [2] Gramatický význam sa určuje na základe danej gramatiky. Syntaktická analýza dokáže takisto určiť správnosť vstupu vzhľadom k danej gramatike.

Softwarový proces je množina aktivít nutných k tomu, aby software vznikol, ich súslednosť a opakovanie, vstupy a výstupy jednotlivých aktivít a nároky na ich prevedenie. [3]

Normalizácia odpovede v tejto bakalárskej práci označuje transformáciu odpovede z textového formátu zadaného užívateľom do formátu JSON s presnou štruktúrou definovanou v kapitole 2.3. Pojem normalizácia sa používa aj v predmete Databázové systémy (ďalej ako DBS). Tu ale znamená pojem normalizácia postup rozkladania relácií v relačnom modely. [4]

1.2 Štruktúra zjednodušeného relačného zápisu

V otázke typu transformácia konceptuálneho modelu do zjednodušeného relačného zápisu je študentovou úlohou transformovať daný konceptuálny model zadaný formou obrázku do zjednodušeného relačného zápisu. Zjednodušený relačný zápis sa v rámci predmetu DBS vyučuje s presne definovanými pravidlami [5].

Jednotlivé tabuľky sa značia formou: „názov tabuľky“(„atribúty“). U atribútov sa rozlišuje ich povinnosť. Rozlišujú sa na primárne, povinné a obyčajné atribúty. Atribúty sa ako primárne označujú podčiarknutím, ako povinné sa označujú prefixom „*“ a za obyčajné sa považujú všetky atribúty pokiaľ niesú inak označené. Jednotlivé atribúty sa od seba oddeľujú čiarkou. Príklad takejto tabuľky: `Sklad(nazov, *rozloha, *adresa, položky)`.

Jednotlivé integritné obmedzenia sa majú následovnú štruktúru: „cieľová tabuľka“[„cieľová skupina atribútov“] $\subseteq$ „zdrojová tabuľka“[„zdrojová skupina atribútov“], kde cieľová skupina atribútov a zdrojová skupina atribútov môže byť aj jeden atribút. Pri skupine atribútov je dôležité poradie, v ktorom sú atribúty uvedené, pretože n-tý atribút z cieľovej skupiny atribútov je závislý na n-tom atribúte zo zdrojovej skupiny atribútov. Zároveň je veľmi dôležité rozlíšenie skupiny atribútov, pretože je rozdiel či zadáme jedno integritné obmedzenie s atribútmi napr. `A, B, C` alebo tri samostatné integritné obmedzenia s atribútmi `A`, `B` a `C`. V prvom prípade je závislá celá skupina súčasne, pričom v druhom sú závislé jednotlivé atribúty samostatne.

1.3 Vývoj riešenia problematiky v rámci fakulty

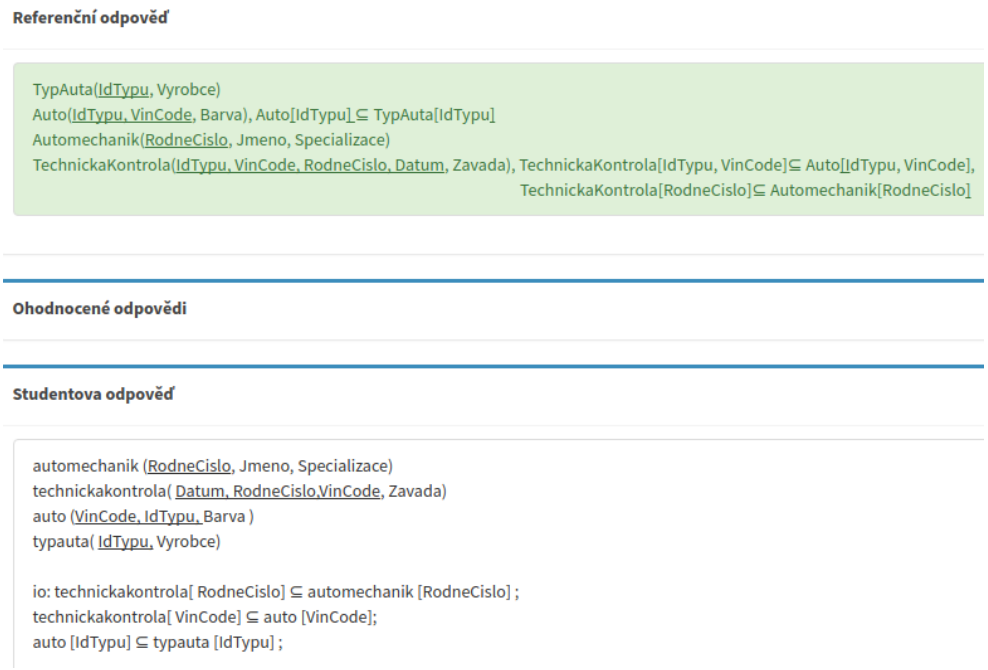
V tejto časti bakalárske práce detailne analyzujem existujúce práce na túto problematiku. Pri tejto analýze bolo cieľom odhaliť výhody a nevýhody jednotlivých riešení, pre zjednošenie návrhu môjho riešenia. Portál dbs.fit.cvut.cz prebieha neustálym vývojom a zlepšovaním, pod vedením pána Ing. Jiřího Hunku, ktorý je jeho duševným autorom. Požiadavka na opravu otázok typu transformácia konceptuálneho modelu do zjednodušeného relačného zápisu vznikla v podstate so vznikom systému a kvôli tomu, že žiadna aplikácia zameraná na opravu zjednodušeného relačného zápisu nebola dostupná, začal vývoj v rámci Fakulte informačných technológií Českého vysokého učení technického v Praze (ďalej už len ako fakulta).

1.3.1 Prvá verzia

Prvá verzia vznikla z bakalárskej práce Petra Pejša [6]. V tejto verzii sa pre odpoveď typu transformácia konceptuálneho modelu do zjednodušeného relačného zápisu používala otázka všeobecného typu text. Pri odoslaní študentovej odpovedi, sa odpoveď automaticky porovnala s referenčnou odpoveďou a všetkými ohodnotenými odpoveďami. Pri porovnaní muselo dôjsť k

absolútnej zhode. Ak sa odpoveď líšila v akomkoľvek znaku napr. študent pridal medzeru naviac, odpoveď bolo potrebné opraviť manuálne učiteľom. Pri manuálnej oprave bolo zobrazenie referenčnej odpovede a študentovej odpovede veľmi jednoduché a priamočiare. V hornej časti sa zobrazila referenčná odpoveď, pod ňou študentova. Následne musel učiteľ odpoveď manuálne opraviť. Príklad takejto opravy je zobrazený na obrázku 1.1-

Obr. 1.1: Ukážka zobrazenia opravy v prvej verzii



Táto verzia mala jasnú výhodu v tom, že jej výstup bol na 100% spoľahlivý. Odpoveď sa vyhodnotila ako správna iba pri absolútnej zhode s referenčnou odpoveďou. V prípade ak došlo k absolútnej zhode s ohodnotenou odpoveďou bola táto odpoveď ohodnotená rovnakým počtom bodov. V inom prípade ju musel učiteľ manuálne opraviť. Nevýhodou bolo množstvo manuálnych opráv, ktoré musel učiteľ vykonať.

1.3.2 Druhá a zároveň aktuálna verzia

Vývoj portálu dbs.fit.cvut.cz prebieha aj v rámci predmetov Softwarový týmový projekt 1 (BI-SP1) a Softwarový týmový projekt 2 (BI-SP2) pod vedením pána Ing. Jiřího Hunku. Takéhoto týmu som bol takisto pri absolvovaní týchto predmetov súčasťou. Náš tým mal za úlohu vytvoriť komponentu, ktorá mala zjednodušiť opravu otázky typu transformácia konceptuálneho modelu do zjednodušeného relačného zápisu. Túto úlohu som v rámci týmu riešil ja a Anna Dřevníkovská. Po konzultácii s pánom Ing. Jiřím Hunkom sme navrhli

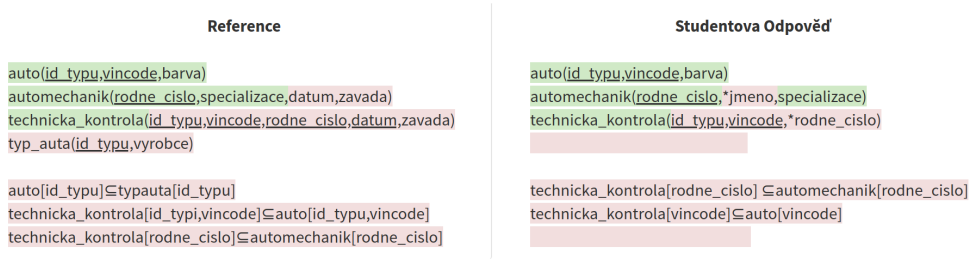
riešenie, ktoré pozostávalo z dvoch krokov. Ako prvý krok prebehne normalizácia odpovede do štandardizovaného formátu JSON. Ako druhý krok sa porovná normalizované referenčné riešenie a normalizovaná študentova odpoveď. Pri tomto porovnaní dôjde k zvýrazneniu chýb pre zjednošenie manuálnej opravy učiteľom. Ja som realizoval normalizáciu odpovedí, zatiaľ čo Anna Dřevníkovská realizovala porovnanie a zvýraznenie týchto odpovedí.

Pri normalizácii sa v prvom kroku z odpovedí odstránia pomocou regulárnych výrazov všetky nepotrebné znaky, ako sú medzery a nevyžiadané HTML tagy. Následne sa odpoveď rozdelí na jednotlivé riadky na základe príslušných HTML tagov, ktoré definujú nový riadok, kontkrétne sa jedná o tagy `<div>` a `<br>`. Jednotlivé riadky sa následne spracúvajú samostatne. Použitím regulárnych výrazov sa odlíši či sa jedná o integritné obmedzenie alebo tabuľku na základe prítomných jednoduchých alebo hranatých zátvoriek. Ako posledný bod sa v prípade tabuliek odlíšia primárne atribúty od ostatných, pretože primárne atribúty musia byť pri odpovedi podčiarknuté, čo v tomto prípade znamená prítomnosť HTML tagu `<u>`. Ako finálny krok sa odpoveď uloží vo formáte JSON, zároveň je uložená aj pôvodná odpoveď študenta pre prípad zlyhania normalizácie.

Po odovzdaní sa tak ako v predchádzajúcej verzii najprv odpoveď automaticky porovná s referenčnou. Porovnanie prebieha rovnakým spôsobom, teda je potrebná absolútna zhoda pre označenie odpovede ako správnej. V tomto prípade sa ale porovnávajú normalizované odpovede. Ak nedôjde k zhode odpoveď je potrebné manuálne opraviť. Pri manúalnej oprave sa na začiatku skontroluje, či je normalizovaná odpoveď v korektnom JSON formáte. Pri negatívnom výsledku je o tom učiteľ pri oprave informovaný a zobrazí sa mu pôvodná odpoveď. V kladom prípade dôjde k porovnaniu referenčnej a študentovej odpovede. Jednotlivé tabuľky a integritné obmedzenia sa porovnávajú samostatne a postupne sa označia podľa správnosti pre následné zobrazenie učiteľovi. Pred samostaným zobrazením, je potrebné zjednotiť počet tabuliek a integritných obmedzení v odpovediach doplnením prázdnych hodnôt. Ako posledný krok sa učiteľovi zobrazia 2 stĺpce, kde v jednom je zobrazená referenčná odpoveď a v druhom študentova s vyznačenými chybami. Príklad uvádzam na obrázku 1.2. Na tomto príklade je zároveň znázornená chyba pri oprave otázky. Integritné obmedzenie `technicka_kontrola[rodne_cislo]`$\subseteq$ `automechanik[rodne_cislo]` sa nachádza ako v referenčnej tak študentovej odpovedi, napriek tomu je označené červenou ako nesprávne.

Výhoda aktuálnej verzie spočíva hlavne vo vylepšení automatickej opravy, pretože z normalizovaných odpovedí sú odstránené neviditeľné znaky ako sú napr. medzery. Nevýhodou je nedostatočná kontrola vstupu. Množstvo odpovedí, ktoré nemajú správny tvar sú označené ako validné. Ďalšou nevýhodou je zobrazovanie chýb pri manuálnej oprave učiteľom z dôvodu jeho neúplnej funkčnosti. V určitých situáciách sú zobrazené správne časti odpovede ako nesprávne čím sa systém stáva nedôveryhodný v očiach učiteľov. Učiteľ si v takomto prípade pre korektnú opravu musí zobraziť pôvodnú odpoveď a

Obr. 1.2: Ukážka zobrazenia opravy v druhej verzii



opraviť túto otázku ako v prípade prvej verzie. Toto riešenie danej problematiky teda nenaplnilo svoj účel, zjednošiť manuálnu opravu učiteľom a tým im ušetriť drahocenný čas, naopak pridalo nedôveru a nespokojnosť.

Kapitola 2

Návrh

Pri návrhu riešenia bolo mojím cieľom zachovať všetky výhody predchádzajúcich riešení a zároveň eliminovať všetky ich nevýhody. Moje ciele sú teda následovné:

1. Zabezpečiť bezchybné zobrazenie odpovedí pri manuálnej oprave učiteľom. Ak je to možné označiť chyby v tejto odpovedi v opačnom prípade zobraziť pôvodnú odpoveď.

2. Minimalizovať množstvo pôvodných odpovedí, ktoré budú zobrazené učiteľom pri manuálnej oprave.

3. Vylepšiť zobrazovanie chýb pre čo najprehľadnejšie zobrazenie pri manuálnej oprave.

Rozhodol som sa nepokračovať v súčasnom riešení, pretože ako jeho autor viem, že ani jeho ďalšími úpravami nenaplním stanovené ciele. Jednoduché použitie regulárnych výrazov nieje dostačujúce pre riešenie tejto problematiky. Po konzultácii s pánom Ing. Jiřím Hunkom som navrhol riešenie, ktoré všetky tieto kritéria spĺňa.

Riešenie, ktoré som navrhol je založené na použití lexikálnej a syntaktickej analýzy. Jedná sa o samostatnú aplikáciu, ktorá komunikuje s ostatnými službami cez rozhranie REST API. Túto aplikáciu ďalej nazývam Tralex.

2.1 Komunikácia s portálom dbs.fit.cvut.cz

Aplikácia bude s portálom dbs.fit.cvut.cz komunikovať cez rozhranie REST API. Tralex bude poskytovať dva prístupové body a to `transform`, ktorý bude slúžiť pre normalizáciu odpovede a `correct`, ktorý bude slúžiť na opravu odpovede. Tieto prístupové body budú na vstupe príjmať dáta a ich odpoveďou budú požadované spracované dáta.

2.2 Architektúra

Rozhodol som sa pre použitie monolitickej architektúry [7]. Tralex nebude používať vlastnú databázu. Všetky dáta potrebné pre spracovanie, bude Tralex príjmať ako obsah volaní na jednotlivé prístupové body a spracované dáta vráti ako odpoveď. Portál dbs.fit.cvut.cz používa vlastnú databázu. Ak by k nej aplikácia Tralex chcela pristupovať, bolo by potrebné zabezpečiť bezpečnosť tohto prístupu. Zároveň by musela používať rovnaký formát dát ako portál dbs.fit.cvut.cz. To by znamenalo ošetrovanie bezpečnosti a formátu dát, ktoré už ošetruje samotný portál. Naopak, vďaka neexistencie databázy, nieje možné aplikáciu Tralex zneužiť, pretože ak by sa niekomu podarilo zavolať prístupový bod pre opravu potrebuje vedieť správnu odpoveď podľa, ktorej sa má oprava vykonať.

2.3 Normalizácia odpovede

Pri normalizácii odpovede bude na vstupe originálna odpoveď. V prvom kroku potrebné previesť orignálnu odpoveď študenta na jednotlivé tokeny pomocou lexeru. Podľa štruktúry zjednodušeného relačného zápisu budem rozoznávať nasledujúce tokeny:

- WORD - Slovo, identifikuje názov tabuľky alebo atribútu.
- PRIMARY_START - HTML tag (<u>), identifikuje začiatok primárnych atribútov.
- PRIMARY_FINISH - HTML tag (</u>), identifikuje koniec primárnych atribútov.
- LEFT_PAREN - Ľavá zátvorka.
- RIGHT_PAREN - Pravá zátvorka.
- LEFT_BRACKET - Ľavá hranatá zátvorka.
- RIGHT_BRACKET - Pravá hranatá zátvorka.
- COMMA - Čiarka.
- REQUIRED - Hviezdička (*) označuje povinné atribúty.
- CONDITION - Znak ($\subseteq$) pre určenie závislosti v prípade integritných obmedzení.

Tieto tokeny budú následne slúžiť ako vstup pre syntaktický analyzátor, ktorý ich spracuje.

S problematikou syntaktickej analýzy som sa zoznámil v rámci predmetu Automaty a Gramatiky (ďalej ako BI-AAG). Najdôležitejší krok pri návrhu

syntaktického analyzátora je popis daného jazyka vhodnou bezkontextovou gramatikou. V tomto prípade som potreboval zostrojiť gramatiku pre zjednodušený relačný zápis, ktorého detaily popisujem v kapitole 1.2.

Pri návrhu gramatiky som si ako prvé definoval množinu terminálnych symbolov, čo je vlastne abeceda nad ktorou je definovaný jazyk zjednodušeného relačného zápisu. Množina terminálnych symbolov sa skladá z nasledujúcich prvkov: `WORD`, `LEFT_PAREN`, `RIGHT_PAREN`, `LEFT_BRACKET`, `RIGHT_BRACKET`, `PRIMARY_START`, `PRIMARY_FINISH`, `CONDITION`, `COMMA`, `REQUIRED`. Ich význam je rovnaký ako v prípade rozpoznateľných znakov zavedených pre lexikálny analyzátor. Vďaka znalostiam nadobudnutých v predmete BI-AAG sa mi podarilo vytvoriť regulárnu gramatiku popisujúcu tento jazyk. Táto gramatika obsahuje 27 prechodocých pravidiel a je uvedená vo formáte BNF.

```
<NAME> ::= "WORD"<CHOOSE>
<CHOOSE> ::= "LEFT_PAREN"<ATTRIBUTES>
| "LEFT_BRACKET" <LEFT_ATTRIBUTES>
<ATTRIBUTES> ::= "COMMA" <ATTRIBUTES>
| "WORD" <ATTRIBUTES>
| "PRIMARY_START" <PRIMARY_ATTRIBUTES>
| "REQUIRED" <REQUIRED_ATTRIBUTES>
| "RIGHT_PAREN" <NAME>
| "RIGHT_PAREN"
<REQUIRED_ATTRIBUTES> ::= "WORD" <ATTRIBTUES>
<PRIMARY_ATTRIBUES> ::= "WORD" <PRIMARY_ATTRIBUTES>
| "COMMA" <PRIMARY_ATTRIBUTES>
| "REQUIRED" <PRIMARY_ATTRIBUTES>
| "PRIMARY_FINISH" <ATTRIBUTES>
| "RIGHT_PAREN" <PRIMARY_END_SWITCH>
<PRIMARY_END_SWITCH> ::= "PRIMARY_FINISH"
| "PRIMARY_FINISH" <NAME>
<LEFT_ATTRIBUTES> ::= "WORD" <LEFT_ATTRIBUTES>
| "COMMA" <LEFT_ATTRIBUTES>
| "RIGHT_BRACKET" <CONDITION>
<CONDITION> ::= "CONDITION" <RIGHT_NAME>
<RIGHT_NAME> ::= "WORD" <LEFT_BRACKET_SECOND>
<LEFT_BRACKET_SECOND> ::= "LEFT_BRACKET" <RIGHT_ATTRIBUTES>
<RIGHT_ATTRIBUTES> ::= "COMMA" <RIGHT_ATTRIBUTES>
| "WORD" <RIGHT_ATTRIBUTES>
| "RIGHT_BRACKET" <NAME>
| "RIGHT_BRACKET"
```

Vďaka zostrojeniu regulárnej gramatiky, môžem navrhnúť konečný automat, ktorý bude realizovať syntaktickú analýzu. Znalosť prevodu medzi regulárnou gramatikou a konečným automatom [8] som takisto nadobudol

v rámci predmetu BI-AAG. Výsledný konečný automat je znázornený na obrázku 2.1.

Výstupom normalizácie bude normalizovaná odpoveď. Tá obsahuje identifikátor, toho či bola otázka správne normalizovaná s názvom `valid` a pri jeho zápornej hodnote aj pole chýb s názvom `errors`. V prípade správnej normalizácie môže obsahovať pole správ, ktoré inforrmujú o nesprávnej syntaxi v odpoveď, ktorá ale neovplivní správnosť validácie. Jednotlivé tabuľky sa nachádzajú v asociatívnym poli s názvom `entities`. Ako kľúce sa v používa názov tabuľky, pretože 2 tabuľky s rovnakým názvom nemôžu existovať. Tabuľka potom obsahuje svoj názov, pole primárnych atribútov, pole obyčajných atribútov a pole povinných atribútov. Jednotlivé integritné obmedzenia sa nachádzajú v asociatívnym poli `conditins`. Každe integritné obmedzenie je identifikované názvom cieľovej tabuľky a jej atribútmi. Táto kombinácia je unikátna pretože prípad, že by pre rovnaké atribúty z tabuľky existovali rôzne integritné obmedzenia nemôže nastať. Integritné obmedzenie obsahuje názov a atribúty zdrojovej a cieľovej tabuľky. Prikladám konkrétny príklad popísanej štruktúry 2.2.

2.4 Oprava odpovedí

Pri oprave odpovedí bude na vstupe normalizovaná odpoveď študenta a normalizovaná referenčná odpoveď. Následne sa zistia rozdieli v týchto odpovediach na základe čoho vznikne opravená odpoveď so zoznamom chýb, správnymi tabuľkami a integritnými obmedzeniami vo formáte JSON. Chyby, ktoré bude oprava identifikovať, popisujem v nasledujúcej tabuľke 2.1.

Tabuľka 2.1: Popis chýb pri oprave

Kľúč vo formáte JSON	Typ chyby	Popis
missing	Chýbajúca hodnota	Táto chyba znamená, že v odpovedi niečo chýba, môže sa jednať o tabuľku, integritné obmedzenie poprípade atribút
not_match	Prebytočná hodnota	Táto chyba znamená, že v odpovedi je niečo naviac, môže sa jednať o tabuľku, integritné obmedzenie poprípade atribút
not_marked	Neoznačená hodnota	Jedná sa o špecifickú chybu. Táto chyba znamená, že atribút nieje označený ako povinný.

Formát JSON opravenej odpovede rozširuje formát normalizovanej odpovede o zoznam chýb. Tieto sa nachádzajú v asociatívnym poli `mistakes`, ktorý pribudol v jednotlivých tabuľkách, integritných obmedzeniach ale aj úplne samostatne. Zároveň pribudne aj identifikátor, určujúci správnosť odpovede s názvom `correct`. Príklad opravenej odpovedi vo formáte JSON 2.3.

2.5 Zobrazenie odpovede pri oprave v portáli dbs.fit.cvut.cz

Zobrazenie odpovede pri oprave je potrebné zjednodušiť, pretože v súčasnej verzii je neprehľadné. Namiesto zobrazovania odpovedí v dvoch stĺpcoch, zobrazím odpoveď iba jednu. V tejto odpovedi budú farebne odlíšené chýbajúce a prebytočné tabuľky, integritné obmedzenia a jednotlivé atribúty, pričom červená farba bude reprezentovať chýbajúcu hodnotu a modrá prebytočnú hodnotu. Zároveň vypíšem kompletný zoznam chýb v osobitnej zozname, zobrazený nad samotnou odpoveďou. V prípade, že sa pri normalizácií vyskytne chyba zobrazí sa originálna odpoveď a zároveň informácia o tom, že sa pri normalizázií vyskytla chyba.

2.6 Zvolené technológie

Ako vhodnú technológiu som pre Tralex vybral použitie jazyku Python vo verzii 3.x a knižnice PLY. Python vo verzii 3.x natívne podporuje unicode kódovanie, [9] čo uľahčuje prácu s textom. Knižnica PLY [10] je parsovací nástroj implementovaný v jazyku Python. Tento výber sa ukázal na riešenie problému parsovania textu ako veľmi vhodný, čo ukazuje napr. bakalárska práca Martina Kubiša[11]. Pre komunikáciu aplikáciu Tralex cez REST API použijem framework[1] Flask [12]. Tento framework sa vyznačuje jednoduchosťou nasadenia aj samotného používania. Pre zobrazovanie odpovede v portále DBS pri oprave použijem framework Nette [13], na ktorom je celý portál DBS postavený. S touto technológiou som sa zoznámil už počas absolvovania predmetov BI-SP1 a BI-SP2.

2.7 Postupy pri vývoji

Pri vývoji Tralexu chcem aplikovať viaceré metodiky pri vývoji aplikácií.

Pri vývoji budem používať iteratívny model softwarového procesu [3]. Charakteristikov tohto modelu je vývoj verzií v iteráciach, pričom vývoj jednotlivých verzií prebieha vodopádovým modelom. Oproti vodopádu je pri

[1] framework - aplikáčný rámec

tomto modely výhoda v rýchlejšom vytvorení fungujúcej verzie, podľa ktorej je možné lepšie smerovať nasledujúci vývoj. To má za následok väčšiu spokojnosť s hotovou aplikáciou.

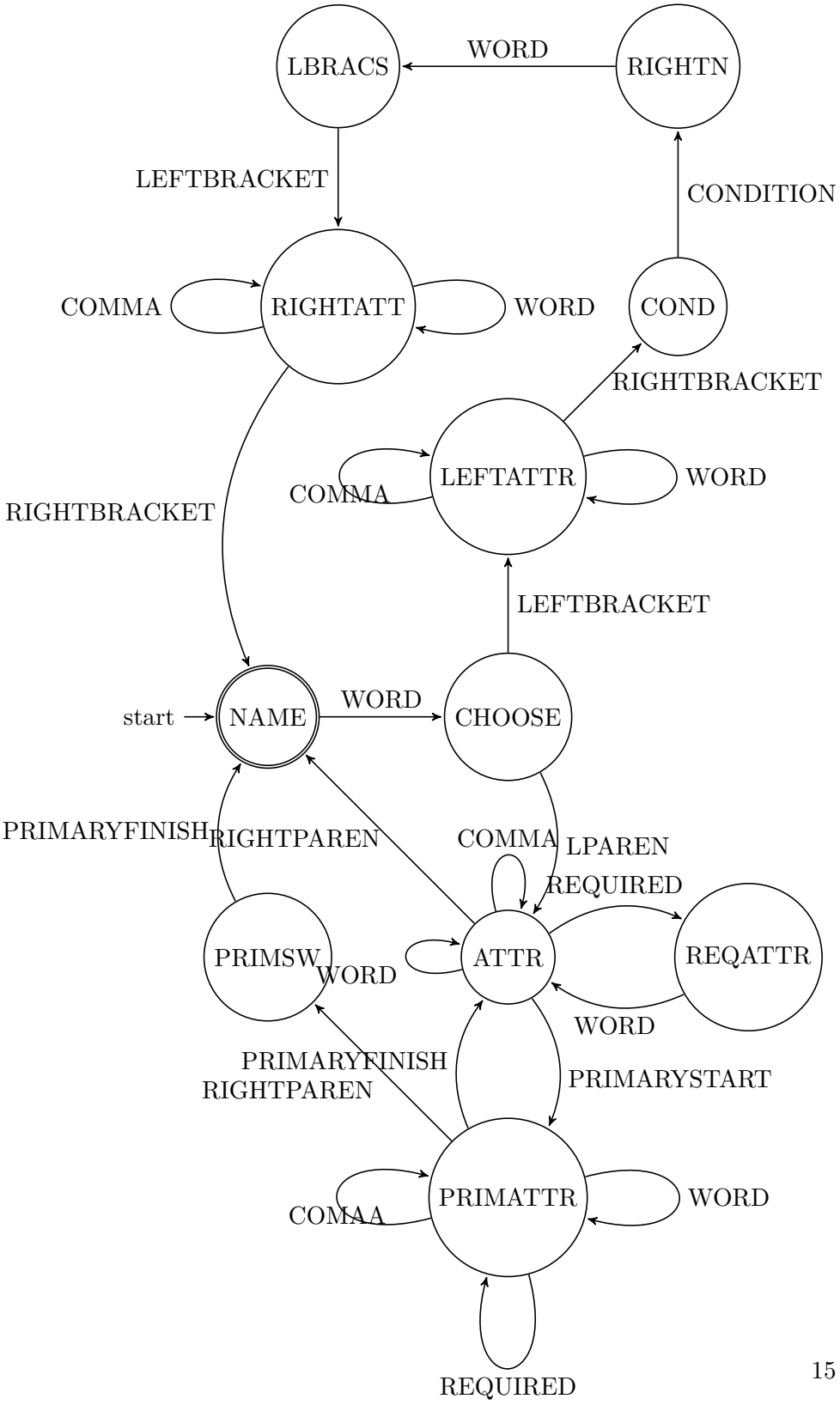
Zber požiadaviek je proces, ktorý zahŕňa všetky aktivity ohľadom zberu, dokumentovania a udržiavania požiadaviek pre aplikácie. [14] „Špatně definované požadavky způsobují neúspěch projektů.“ [15]

Pri vývoji budem pre verzovania kódu používať verzovací systém GIT [16]. S týmto systémom som sa zoznámil pri absolvovaní predmetu BI-GIT a následne ho aktívne používal aj v rámci predmetov BI-SP1 a BI-SP2. GIT sa vyznačuje možnosťami, ktoré ponúka pri verzovaní kódu. Za najväčšiu výhodu tohto systému považujem dokonalú históriu, ku ktorej sa dá jednoducho kedykoľvek vrátiť. V rámci fakulty máme zdarma k dispozícii systém gitlab.fit.cvut.cz, ktorý slúži ako GIT server[17].

Pre zabezpečenie funkčnosti systému budem používať techniky testovania [18]. Mojím cieľom je zamerať sa hlavne na automatické testovanie. Tieto testy poslúžia pre overenie funkčnosti ako pri vývoji novej ale hlavne úprave existujúcej funkcionality. Takéto automatické testy by mali vzniknúť ako pre bežné prípady použitia, tak aj hraničné prípady a v neposlednom rade pre všetky nájdené chyby pre ich trvalé odstránenie.

Pri písaní kódu sa budem držať metódy samo-dokumentujúceho kódu [19]. Vďaka tomu by sa mala zvýšiť prehľadnosť a čitateľnosť kódu pri ďalšom vývoji.

Obr. 2.1: Výsledný konečný automat



Obr. 2.2: Normalizovaná odpoveď vo formáte JSON

```
{
  "conditions": {
    "vozik_sklad": {
    "left_attributes": ["sklad"],
    "left_name": "vozik",
    "right_attributes": ["nazov"],
    "right_name": "sklad"
    }
  },
"entities": {
  "sklad": {
    "name": "sklad",
    "primary": [["adresa", "nazov"]],
    "required": ["rozloha"],
    "secondary": ["farba"]
  },
  "vozik": {
    "name": "vozik",
    "primary": [["spz"]],
    "required": ["nosnos", "sklad"],
    "secondary": []
    }
  },
"valid": true
}
```

Obr. 2.3: Opravená odpoveď vo formáte JSON

```
{
  "conditions": {
    "schovano_left_attr1_left_attr2_left_attr3": {
      "left_attributes": ["left_attr1", "left_attr2",
         "left_attr3"],
      "left_name": "schovano",
      "mistakes": {
        "right_attributes": {
          "missing": ["right_attr1"],
          "not_match": ["right_attr"]
          }
        },
      "right_attributes": ["right_attr2", "right_attr3
         "],
      "right_name": "vajicko"
    }
  }
  "correct": false,
  "entities": {},
  "mistakes": {
    "entities": {
      "missing": {
        "sklad": {
        "name": "sklad",
        "primary": [["id"]],
        "required": [],
        "secondary": ["second_id", "some_attribute"]
        }
      }
    }
  },
  "valid": true
}
```

Kapitola 3

Implementácia

Implementácia aplikácie Tralex je hlavnou časťou tejto bakalárske práce. Implementácia je rozdelená do implementácie samotnej aplikácie Tralex a jej nasadenia a implementácie komununikácie a správneho zobrazenia v portály DBS.

3.1 Knižnica PLY

Pri implementácii lexikálnej analýzy som použil knižnicu PLY [10]. Autorom tejto knižnice je David Beazley. Táto knižnica sa skladá z dvoch hlavných komponent a to lex.py a yacc.py. Komponenta lex.py je lexikálny analyzátor, ktorý v aplikácii Tralex používam pre spracovanie vstupného textu. Komponentra yacc.py je syntaktický analyzátor, ktorý využíva techniky LR-parsovania [20]. Vďaka návrhu konečného automatu realizujúceho syntaktickú analýzu som tento modul vo svojej práci nepoužil. Táto knižnica má veľmi dobrú dokumentáciu, vďaka čomu sa velmi dobre používa.

3.2 API

Aplikácia tralex komunikuje cez rozhranie REST API. Pre tento účel som použil micro framework Flask. Tento framework slúži k tvorbe webových aplikácii v jazyku Python. Flask má veľmi dobrú dokumentáciu a takisto tutoriál čo zjednodušuje jeho používanie. Pre inštaláciu som použil správcu balíkov pre jazyk Python menom pip [21]. Inštalácia je veľmi jednoduchá, stačí do príkazového riadku zadať príkaz:

```
pip install Flask
```

a počkať kým pip stiahne a nainštaluje všetky potrebné súbory. V základnej konfigurácii je aplikácia po spustení dostupná na lokálnom servery na porte

Zdrojový kód 3.1: API pre normalizáciu

```
@app.route("/transform/", methods=['POST'])
def transform():
  try:
    answer = request.form.get('answer')
    parse = Parser.Parser(answer)
    output = parse.parse()
    return output
  except BaseException as ex:
    return 400
```

5000, teda na adrese `http//127.0.0.1:5000/`. Na tejto adrese sa na serveri portálu dbs.fit.cvut.cz už používa prekladač relačnej algebry, vytvorený v rámci bakalárskej práce Martina Kubiša [11]. Adresu, na ktorej je Tralex dostupný som preto zmenil port 7000, teda na adresu `http//127.0.0.1:7000/`. Tieto zmeny popisujem v kapitole ohľadom nasadenia aplikácie 3.6. Flask podporuje všetky základné typy API, to znamená GET, POST, PUT a DELETE. Tralex komunikuje cez 3 endpointy[2] dostupné iba z lokálneho servera. Napriek bezpečnému návrhu aplikácie Tralex, ktorý znemožnuje získanie dát, predstavuje jeho dostupnosť riziko vyťaženia serveru, čo by mohlo v najhoršom prípade viesť k pádu portálu dbs.fit.cvut.cz.

Testovací typu GET s adresou `http//127.0.0.1:7000/test/`, ktorý v prípade úspechu vráti text „Working!“. Slúži iba pre testovacie účely, ako overenie, že server funguje v poriadku a spracováva naše volania.

Endpoint pre normalizáciu odpovede je dostupný na adrese `http//127.0.0.1:7000/transform/`. V tomto prípade sa jedná o API typu POST. Toto API príjma odpoveď v originálnom formáte. Následne prevedie normalizáciu a vráti odpoveď vo formáte JSON. Kód tohto API 3.1.

Endpoint pre opravu odpovede je dostupný na adrese `http//127.0.0.1:7000/correct/`. Jedná sa takysto o API typu POST. Toto API príjma na vstupe normalizovanú refenrenčnú a študentovu odpoveď vo formáte JSON. Následne prevedie opravu a vráti opravenú odpoveď vo formáte JSON. Kód tohto API 3.2.

3.3 Tranformácia odpovede

3.3.1 Lexikálna analýza

Ako prvý krok pri normalizácií odpovede je potrebné vstupný text previesť na postupnosť tokenov, ktoré budeme následne spracovávať. V knižnici PLY

[2] endpoint-prístupový bod

Zdrojový kód 3.2: API pre normalizáciu

```
@app.route("/correct/", methods=['POST'])
def correct():
  try:
    teacher_answer = request.form.get("reference")
    student_answer = request.form.get("answer")
    output = Parser.Parser.correct_answer(str(
       teacher_answer), str(student_answer))
    return output
  except BaseException as ex:
    return 400
```

slúži na tento účel slúži modul lex.py. Pri jeho volaní dostaneme ako návratovú hodnotu instanciu objektu LexToken, ktorý má 4 atribúty:

- type - Popisuje o aký typ tokenu sa jedná.
- value - Text, ktorý obsahuje daný token.
- lineno - Číslo riadku na ktorom sa daný token vyskytuje.
- linepos - Číslo pozície na ktorej daný token začína.

Tento modul potrebuje pre správne rozpoznávanie tokenov definíciu jednotlivých tokenov vo forme regulárnych výrazov. To realizuje trieda `Tokenizer`. V vstupnom texte sa môžu nachádzať znaky, ktoré je potrebné ignorovať. Jedná sa o všetky HTML tagy okrem tagu <u>, ktorý odlišuje primárne atribúty a tagov
, , ktoré definujú nový riadok. Rozpoznateľné znaky sú definované v kapitole 2.3. Zoznam použitých tokenov:

WORD, PRIMARY_START, PRIMARY_FINISH, LEFT_PAREN, RIGHT_PAREN, LEFT_BRACKET, RIGHT_BRACKET, COMMA, REQUIRED, CONDITION.

Príklad kódu fuknkcií definujúcich regulárne výrazy podľa, ktorých sa realizuje lexikálna analýza 3.3.

Z príkladu funkcií 3.3 vyplýva, že niektoré funkcie obsahujú okrem definície aj úpravu premennej `position`. Kvôli HTML tagom, ktoré sa pri normálnom zobrazení nezobrazia bolo potrebné triede `Tokenizer` pridať túto premennú, v ktorej je uložená aktuálna pozícia tokenu ale iba od začiatku aktuálneho riadku. Hodnota tejto pozície sa na začiatku každého riadku vynuluje. Modul `Lex.py` v sebe obsahuje premennú `lineno`, ktorá označuje aktuálny riadok. Pre korektné počítanie riadkov je potrebné definovať oddeľovač nového riadku. Podľa dokumentácie [10], je potrebné implementovať funkciu `t_newline`. Ako

Zdrojový kód 3.3: Definícia pravidiel pre tokeny

```
def t_WORD(self, t):
    r'[a-zA-Z0-9_-]+'
    self.position += len(t.value)
    return t

def t_CONDITION(self, t):
    r'(&sube;|\u2286)'
    t.value = '\u2286'
    self.position += 1
    return t

def t_PRIMARY_START(self, t):
    r'&lt;u((?!&gt;).)*&gt;|<u[^>]*>'
    t.value = '<u>'
    return t

def t_PRIMARY_FINISH(self, t):
    r'&lt;/u((?!&gt;).)*&gt;|</u[^>]*>'
    t.value = '</u>'
    return t
```

oddeľovač používam `<br>` a `<div>` HTML tagy. Táto funkcia zároveň vynuluje premennú `position`. Kód tejto funkcie 3.4.

Zdrojový kód 3.4: Funkcia pre počítanie riadkov

```
def t_newline(self, t):
    r'<div[^>]*>|<br>|<br\/>'
    t.lexer.lineno += 1
    self.position = 0
```

Funkcia `t_space` slúži pre správnu kalkuláciu aktuálnej pozície uloženej v premennej `position`. Jej zdrojový kód 3.5

Zdrojový kód 3.5: Funkcia pre rozpoznie medzery

```
def t_space(self):
    r'\ '
    self.position += 1
```

Pre odstránenie HTML tagov definujem následovné pravidlá, ktoré majú

zabezpečiť ich ignorovanie, kvôli ich počtu a podobnosti uvádzam iba krátky príklad 3.6.

Zdrojový kód 3.6: Príklad funkcií pre ignorovanie HTML tagov

```
t_ignore_a = r'<\/?a[^>]*>|&lt;\/?a((?!&gt;).)*&gt;'
t_ignore_abbr = r'<\/?abbr[^>]*>|&lt;\/?abbr((?!&gt;)
    .)*&gt;'
t_ignore_area = r'<\/?area[^>]*>|&lt;\/?area((?!&gt;)
    .)*&gt;'
...
```

Najdôležitejšia funkcia triedy `Tokenizer` je funkcia `tokenize` 3.7. Tá používa funkciu `token` z modulu `Lex.py`, ktorá vráti nasledujúci token vo forme instancie objektu `LexToken`, popísanú na začiatku tejto kapitoly. K jeho základným atribútom pridá táto funkcia atribút `position` na základe premennej `position`. V prípade, že nastane koniec vstupu, vráti táto funkcia `None`.

Zdrojový kód 3.7: Funkcia získanie ďalšieho tokenu

```
# Return next matched token, in case of no more input
    return None
def tokenize(self):
  tok = self.lexer.token()
  if not tok:
    return None      # No more input
  tok.position = self.position
  return tok
```

Funkcie, ktoré sa používajú pri získavaní pozície, na ktorej nastala chyba sa výstižne volajú `get_error_position` a `get_error_line`. Na vstupe majú jeden argument typu LexToken. Tieto funkcie sú potrebné z dôvodu ošetrenia konca vstupu, to znamená, že na vstupe dostanú None. V tom prípade vrátia aktuálnu pozíciu a riadok posledného tokenu.

3.3.2 Syntaktická analýza

Syntaktickú analýzu realizuje konečný automat popísaný v kapitole 2.3. Implementáciu konečného automatu realizuje trieda `Automaton`. Jednotlivé prechodové pravidlá definuje číselník `StatesChange` v súbore `Enums.py`. Táto trieda obsahuje jednu instančnú premennú `state`, ktorá označuje aktuálny stav automatu. V konštruktore tejto triedy túto premennú nastavím na začiatočný stav, teda `NAME`.

Funkcia `next_step` na základe vstupného tokenu a definície prechodov zmení aktuálny stav automatu. V prípade, že prechodové pravidlo pre tento token neexistuje zavolá sa funkcia `handle_error`, ktorá spracuje nesprávny vstup. Kód funkcie `next_state` 3.8.

Zdrojový kód 3.8: Funkcia pre prechod medzi stavmi v konečnom automate

```
def next_state(self, current_token):
  if current_token.type in Enums.StatesChange[self.
     actual_state].\
    value and isinstance(Enums.StatesChange[self.
       actual_state].\
    value[current_token.type], str):
    self.actual_state = Enums.StatesChange[self.
       actual_state].\
    value[current_token.type]
    return self.actual_state
  else:
    self.handle_error()
```

Funkcia `handle_error` slúži na identifikáciu chyby v prípade nesprávneho vstupu. Na tento účel sú v súbore `_exceptions.py` definované konkrétne výnimky s príslušnou správou. Všetky konkrétne výnimky sú potomkom triedy `BadInputException` a obsahujú popis chyby z triedy `Error` nachádzajúcej sa v pomocnom súbore `Enums.py`. Definujem vlastný typ výnimky `BadInputException`, ktorá je potomkom triedy `Exception`. Jednotlivé stavy automatu majú definované vlastné výnimky, ktoré v týchto stavoch môžu nastať. Mojím cieľom je pokryť všetky chybné vstupy, ktoré sa bežne vyskytujú a informovať o tom užívateľa. V tabuľke 3.1, uvádzam typy výnimiek v jednotlivých stavoch a v tabuľke 3.2, popisujem význam týchto výnimiek.

Tabuľka 3.1: Výnimky v jednotlivých stavoch

Stav	Typ výnimky
NAME	NameMissingException
CHOOSE	BracketParenMissing
ATTRIBUTES	ParenException
REQUIRED_ATTRIBUTE	MoreRequiredSignsException
LEFT_ATTRIBUTES	BracketException
CONDITION	ConditionSignException
RIGHT_NAME	ConditionNameException
LEFT_BRACKET_SECOND	BracketException
RIGHT_ATTRIBUTES	BracketException

Tabuľka 3.2: Popis jednotlivých výnimiek

Typ výnimky	Popis
NameMissingException	Chýbajúci názov tabuľky.
BracketParenMissing	Chýbajúca obyčajná a hranatá zátvorka.
ParenException	Chybné ukončenie zátvorky.
MoreRequiredSignsException	Atribút je označený viacerými „*“ ako povinný.
BracketException	Chybné ukončené hranaté zátvorky, poprípade úplna absencia na vyžadovanom mieste
ConditionSignException	Chýbajúci znak „∈“ medzi integritnými obmedzeniami.
ConditionNameException	Chýbajúci názov zdrojovej tabuľky integritného obmedzenia.

3.3.3 Reprezentácie odpovede

Výsledkom normalizácie je odpoveď vo formáte JSON. Pri samotnom normalizovaní odpovede, sa snažím čo najviac držať princípov objektového programovania a preto používam objekty, ktoré až úplne nakoniec kovertujem do formátu JSON.

Pri návrhu triedy reprezentujúcu odpoveď, bolo mojím hlavným cieľom, ju navrhnúť tak, aby prípadné budúce úpravy nepredstavovali žiadny problém. Pri iteratívnom spôsobe vývoja je to kritická vlastnosť, pretože sa funkčnosť postupne pridáva v jednotlivých iteráciách a v prípade nesprávneho návrhu, by to mohlo predstavovať rozsiahle úpravy pôvodného návrhu. Mojim ďalším cieľom bolo túto triedu navrhnúť tak, aby práca s ňou bola intuitívna a jednoduchá. Výsledok môjho návrhu je trieda príznačne pomenovaná `Answer`. Pre zabezpečenie všetkých operácií potrebných pri normalizácií a dosiahnutie stanovených cieľov bolo potrebné navrhnúť triedy, ktoré budú reprezentovať menšie celky, ktoré bude trieda `Answer` obsahovať. Pre tento účel som navrhol triedy `Entity` a `Condition`. Trieda `Entity` reprezentuje jednu tabuľku a trieda `Condition` reprezentuje jedno integritné obmedzenie. V prvých verziách sa jednalo o samostatné triedy ale neskôr som si uvedomil, že tieto triedy majú viacero rovnakých atribútov a metód, preto som vytvoril abstraktnú triedu `GeneraLine`. Triedy `Entity` a `Condition` sú teraz potomkovia tejto triedy. Detailne sú tieto triedy popísané v diagrame tried 3.1. V tomto diagrame je pre úplnosť znázornená aj trieda `CorrectionAnswer`, ktorá sa používa pre opravu odpovedí. Jej funkčnosť opisujem v kapitole 3.4.

Trieda `Parser` implementuje syntaktický analyzátor. Metóda `parse` realizuje samotnú normalizáciu odpovede. Požíva pri tom konečný automat implementovaný v triede `Automaton`. Ako prvý krok normalizácie sa text prevedie na malé písmená. Následne prebieha spracovanie vstupu. Postupne sa na

základe aktuálneho stavu automatu vytvára normalizovanú odpoveď. Keďže tabuľka a integritné obmedzenie začína rovnako na meno tabuľky to sa uloží. V ďalšom kroku sa podľa príslušnej zátvorky rozhodne či sa jedná o tabuľku alebo integritné obmedzenie a zavolá sa funkcia `parse_one_entity` v prípade tabuľky, resp. `parse_one_condition` v prípade integritného obmedzenia. Funkcia `parse_one_entity` spracuje jednu tabuľku a vráti ju uloženú v instancii triedy `Entity`. Tá, je následovne pridaná do premennej `entitites`, čo je asociatívne pole tabuliek, kde je je ako kľúc použitý názov tabuľky. To zabezpečí kontrolu duplicít v tabuľkách. Funkcia `parse_one_condition` obdobne spracuje jedno integritné obmedzenie a vráti ho uložené v instancii triedy `Condition`. Tá je pridaná do premennej `conditions`, čo je asociatívne pole integritných obmedzení, kde je ako kľúč použitá kombinácia názvu a atribútov cieľovej tabuľky.

V prípade, že sa na vstupe vyskytuje chyba, trieda `Automaton` vyhodí výnimku, ktorá je následne spracovaná. Normalizácia ihneď skončí a vráti odpoveď s príslušnou chybou. Ak nastane neočakavána výnimka, je takisto spracovaná a pridaná do zoznamu chýb. Ukážka kódu so spracovaním výnimiek:

```
def parse(self):
    ...
    except _exceptions.BadInputException as ex:
    self.answer.set_error(ex.args[0], self.lexer.
        get_error_position(self.current_token), self.
        lexer.get_error_line(self.current_token))
    break
    except Exception as ex:
    self.answer.set_error(ex.args[0], self.lexer.
        get_error_position(self.current_token), self.
        lexer.get_error_line(self.current_token))
    break
    ...
```

Po úspešnom spracovaní jednotlivých tabuliek a integritných obmedzení, ostáva odpoveď konvertovať do formátu JSON. Pre zachovanie jednoduchej použiteľnosti, chcem túto konverziu realizovať štandardnou funkciu v jazyku Python `json.dumps`. Podľa dokumentácie [22], je potrebné implementovať vlastnú triedu ako potomka `JSONEncoder` s funkciou `default`. Na tento účel slúži trieda `AnswerEncoder`. Pri použití `json.dumps` využijem jej vlastnosť, že dokáže výstup pri konverzii aj zoradiť. Ukážka konvertovania vo funkcii `parse`:

```
def parse(self):
    ...
    return json.dumps(self.answer, sort_keys=True, cls=
        Answer.AnswerEncoder)
```

3.4 Oprava odpovede

API pre opravu odpovedí som popísal v kapitole 3.2. Samotné API otázky neopravuje, to iba zavolá statickú funkciu `correct_answer` triedy `CorrectionAnswer` s referenčnou a študentovou odpoveďou. Trieda `CorrectionAnswer` je navrhnutá pre reprezentáciu opravenej odpovede. Je potomkom triedy `Answer` a túto triedu rozširuje o funkciu `rate_answer` a statickú funkciu `correct_answer` ako je znázornéné na diagrame tried 3.1.

3.4.1 Funkia correct_answer

Táto funkcia implementuje opravu odpovede. Na vstupe príjma študentovu odpoveď a referenčnú odpoveď vo formáte JSON, podľa ktorej prevedie opravu. Na jej výstupe je opravená odpoveď vo formáte JSON. Ako prvý krok sa zo vstupných odpovedí vytvoria objekty typu `Answer`, ktoré reprezentujú jednotlivé odpovede. Vytváranie týchto objektov zabezpečuje statická funkcia `create_from_dict` triedy `Answer`. Tá používa funkciu rovnakého názvu avšak z triedy `Entity`, resp. `Condition` pre vytvorenie jednotlivých instancií týchto tried pre reprezentáciu tabuliek, resp. integritných obmedzení. Tieto funkcie sú implementačne veľmi podobné. Obidve prejdú v cykle vstupné asociatívne pole a na základe kľúčov skopírujú ich hodnotu do príslušnej premennej, líšia sa len v návratovej hodnote, ktorá je buď instancia objektu `Entity` alebo `Condition`. V prípade hodnoty premennej `valid` je potrebné previesť konverziu, pretože môžu nastať prípady, kedy sa v normalizovanej odpovedi nachádza hodnota ako text namiesto typu `Boolean`. To zabezpečuje statická funkcia `convert_to_bool_from_string`, ktorá porovná vstupnú hodnotu s textovou reprezentáciou týchto premenných a v prípade zhody vráti `Boolean` hodnotu, v opačnom prípade vráti pôvodnú hodnotu na vstupe.

Po konvertovaní je potrebné zkontrolovať validitu obidvoch odpovedí. V prípade, že je jedna z odpovedí nevalidná automatickú opravu nechcem previesť, pretože neviem zaručiť, že oprava bude relevantná a jeden z mojich cieľov v tejto práci je dosiahnuť úplnu spoľahlivosť pri oprave. Ak táto kontrola prebehne v poriadku zavolá sa funkcia `rate_answer` na novo vytvorenej instancii triedy `CorrectionAnswer` reprezentujúcu opravenú odpoveď. Táto funkcia uloží všetky správne časti aj chyby do instancie triedy. Na záver sa instancia triedy `CorrectionAnswer` reprezentujúca opravenú odpoveď konvertuje do

formátu JSON rovnako ako v prípade triedy `Answer`, tentokrát ale použitím triedy `AnswerCorrectionEncoder` ako argumentu vo funkcii `json.dumps`.

3.4.2 Funkcia rate_answer

V tomto odstavci detailne opisujem funkciu `rate_answer`. Táto funkcia má na vstupe dve argumenty typu `Answer`. Jedná sa o študentovu a učiteľovu odpoveď. Jedná sa o triednu funkciu, to znamená, že všetky údaje ukladá v rámci instancie triedy a keďže pri nej nemôže nastať žiadna chyba vďaka ošetreniu vstupu funkciou `correct_answer` nemá žiadnu návratovú hodnotu. Pri oprave rozlišujem chyby popísané v tabuľke 2.1.

Oprava tabuliek a integritných obmedzení prebieha samostatne, ale rovnakým princípom. V prvom kroku sa v cykle prejdú všetky tabuľky referenčnej odpovede. Skontroluje sa, či sa rovnaká tabuľka nachádza aj v študentovej odpovedi. Pri zápornom výsledku sa táto tabuľka pridá do zoznamu chýb opravenej odpovedi ako chýbajúca. Pri kladnom výsledku je potrebné skontrolovať jednotlivé atribúty tabuľky. Nato slúži funkcia `rate` triedy `Entity`. Táto v prvom kroku prejde v cykle skupiny primárnych atribútov referenčnej tabuľke pričom zisťuje existenciu týchto skupín v študentovej tabuľke. V prípade, že sa tam daná skupina atribútov nenachádza, je pridaná do asociatívneho poľa chýb danej tabuľky ako chýbajúca. Ak sa tam nachádza, je pridaná do poľa primárnych atribútov opravenej tabuľky a zároveň sa táto skupina odstráni z študentovej tabuľky. Následne sa v cykle prejdú všetky povinné atribúty referenčnej odpovede obdobne ako primárne s malým rozdielom. Ak sa daný atribút nenájde v povinných atribútoch študentovej odpovede otestuje sa jeho existencia v poli obyčajných atribútoch. Ak sa tam nachádza pridá sa tento atribút do asociatívneho poľa chýb ako neoznačený a zároveň sa odstráni z poľa obyčajných atribútov v študentovej tabuľke. V prípade, že sa nenachádza tento atribút ani v poli obyčajných atribútov, pridá sa do asociatívneho poľa chýb ako chýbajúci. Ak nastane zhoda už v prvej podmienke pri porovnávaní s poľom povinných atribútov študentovej odpovede, odstráni sa z poľa povinných atribútov študentovej odpovede a pridá sa sa do poľa primárnych atribútov opravenej odpovede. Rovnako ako pole primárnych atribútov je následne spracované pole obyčajných atribútov. Ako posledný krok pri oprave tabuľky sa postupne prejdú v cykle zostávajúce primárne, povinné aj obyčajné atribúty v študentovej tabuľke a pridajú sa do asociatívneho poľa chýb ako prebytočné. Tento krok môžme spraviť na základe predošlého odstraňovania zhodných atribútov. Na konci funkcia `rate` vráti opravenú tabuľku, ktorú funkcia `rate_answer` pridá do asociatívneho poľa tabuliek. Potom skontroluje či sa v tejto tabuľke nachádzajú chyby a v kladnom prípade označí odpoveď ako nesprávnu. Následne je zo asociatívneho poľa tabuliek študentovej odpovede táto tabuľka odstránená.

Nasleduje oprava integritných obmedzení. V tomto prípade sa takisto v cykle prejdú všetky integritné obmedzenia referenčnej odpovede. Integritné

obmedzenie je v asociatívnom poli identifikované kľúčom, ktorý sa skladá z názvu cieľovej tabuľky a jej atribútov. Zistí sa ich existencia v štundetovej odpovedi a v zápornom prípade je toto integritné obmedzenie pridané do asociatívneho poľa chýb ako chýbajúce. Ak sa tam nachádza je potrebné opraviť jednotlivé časti tohto obmedzenia. Nato slúži funkcia `rate` triedy `Condition`. Vďaka identifikácii integritného obmedzenia názvom ľavej tabuľky a jej atribútov sa tieto hodnoty iba skopírujú do opraveného integritného obmedzenia. Následne sa porovná názov pravej tabuľky. V prípade, že sa nezhoduje je celá pravá tabuľka z referenčnej odpovede pridaná do asociatívneho poľa chýb ako chýbajúca a pravá tabuľka z študentovej odpovede ako nezhodujúca. Ak sa názov pravej tabuľky zhoduje, v cykle sa prejdú všetky atribúty pravej referenčnej odpovede. Tieto atribúty sa postupne porovnávajú s atribútmi študentovej odpovede. Ak sa zistí zhoda je tento atribút pridaný aj do výsledného integritného obmedzenia a zároveň je odstránený z študentovej odpovede. Ak sa nezhoduje je pridaný do asociatívneho poľa chýb ako chýbajúci. Na konci funkcie `rate` sa všetky zostávajúce atribúty pravej tabuľky študentovej odpovede pridajú do asociatívneho poľa chýb ako nezhodujúce.

Ako posledný krok funkcie `rate_answer` sa v cykle prejdú všetky tabuľky a integritné obmedzenia študentovej odpovede a pridajú sa do asociatívneho poľa chýb ako chýbajúce.

3.5 Úpravy v portále dbs.fit.cvut.cz

Po nasadení aplikácie Tralex je potrebné upraviť portál dbs.fit.cvut.cz pre jeho používanie. Pre prehľadnosť je vhodné triedu, ktorá bude zaobstarávať komunikáciu s aplikáciou Tralex pomenovať rovnako Tralex. Trieda `Tralex` už v portáli existovala z predchádzajúcej implementácie. Túto triedu som sa rozhodol premenovať na `OldTralex` a označiť ju ako deprecated[3]. Pre úplné odstránenie som sa nerozhodol, pretože som si myslel, že ešte môže nastať prípad, keď ju budem potrebovať použiť. Ako sa neskôr pri konvertovaní dát ukázalo, tento krok bol správny.

3.5.1 Komunikácia s aplikáciou Tralex

Pre komunikáciu s aplikáciou Tralex som vytvoril triedu s rovnankým názvom `Tralex`. V konštruktore sa vyplní premenná `url` na základe konfiguračného súboru `config.neon`, kde definujem adresu na ktorej je Tralex dostupný. Táto trieda obsahuje funkciu `transform` a `correct`. Pre komunikáciu tieto funkcie používajú Guzzle HTTP klienta [23]. Tento klient sa v portáli už používa

[3]deprecated - označenie používané v programovacích jazykoch pre zastaralé funkcie a triedy, ktoré budú v budúcnosti odstránené.

a vďaka jeho dobrej dokumentácií a jednoduchému použitiu som ho zvolil takisto.

Funkcia `transform` má na vstupe odpoveď študenta, ktorú pošle na API aplikácie Tralex pre tansformáciu a vráti objekt typu `TralexResponse`.

Funkcia `correct` má na vstupe 2 normalizované odpovede, ktoré pošle na API aplikácie Tralex pre opravu a vráti objekt typu `TralexResponse`.

3.5.2 Reprezentácia normalizovanej odpovede

Pre normalizovanú odpoveď vo formáte JSON, bolo potrebné vytvoriť vhodnú reprezentáciu na strane portálu. Pre tento účel som navrhol triedu `TralexResponse`. Táto trieda kopíruje štruktúru normalizovanej odpovede vo formáte JSON. Táto trieda je vytvorená priamo z normalizovanej odpovede vo formáte JSON. Pri tomto procese bolo nutné ošetriť kovertovanie `Boolean` hodnôt. Táto trieda musí podporovať konvertovanie do formátu JSON. Aj v tomto prípade som chcel docieliť čo najjednoduchšie použitie, ideálne aby sa na konvertovanie dala použiť štandardná funkcia v jazyku PHP `json_encode`. Kvôli tomu táto trieda implementuje rozhranie `JsonSerializable` a povinnú funkciu `jsonSerialize`, ktorá sa pri konvertovaní použije. Pre jednoduché zobrazenie normalizovanej odpovede slúžia funkcie `getEntitiesForLatte` a `getConditionsForLatte)`. Účelom týchto funkcií je pripraviť jednotlivé tabuľky, resp. integritné obmedzenia pre zobrazenie. Ich výstupom je text obsahujúci, všetky HTML elementy pre správne zobrazenie. V latte šablóne potom tieto tabuľky, resp. integritné obmedzenia iba jednoducho zobrazím.

Pre použitie novej triedy `Tralex` pri spracovaní odpovedí, bolo potrebné upraviť všetky triedy, ktoré realizujú prácu s odpoveďami. Jedná sa o triedu zabezpečujúcu vytváranie odpovedí `TransformationCreate` a úpravy odpovedí `TransformationEdit` pri správe odpovedí učiteľom a o triedu zabezpečujúcu spracovanie odpovedí `Transformation` pri písaní testu študentom. Všetky tieto triedy majú rovnakú štruktúru. Obsahujú funkciu `render`, ktorá sa stará o zobrazenie a funkcie `transformAnswer` pre prevedenie normalizácie a funkciu starajúcu sa o uloženie. Vo všetkých prípadoch bolo potrebné upraviť tieto funkcie tak, aby sa odpovede správne zobrazovali a uložili do databázi. Zároveň bolo potrebné upraviť latte šablóny, ktoré vykresľujú normalizovanú odpoveď. Pri normalizácií môže nastať množstvo syntaktických chýb. O ich správny preklad sa stará funkcia `getErrorsAndWarnings`, ktorá do latte šablóny pošle preloženú chybu a pozíciu na ktorej nastala.

Pre použitie triedy `Tralex` pri oprave odpovedí učiteľom bolo potrebné upraviť triedu `StudentTransformationShow`. Táto trieda sa používa pri oprave ako jednotlivých otázok, tak pri hromadnej oprave otázok. V tomto prípade bolo potrebné väčšie zmeny ako v prípade spracovaní odpovedí. Pre zobrazenie opravy som sa rozhodol použiť rozdielne zobrazenie oproti existujúcemu. Namiesto zobrazovania odpovedí v dvoch stĺpcoch, zobrazím odpoveď iba jednu, kde budú farebne odlíšené chýbajúce a prebytočné tabuľky, integritné

obmedzenia a jednotlivé atribúty. Zároveň vypíšem stručný zoznam chýb. Pre získanie zoznamu chýb slúži v triede `TralexResponse` funkcia `countMistakes`. Táto funkcia vráti pole chýb vyskytujúci sa v odpovedi. V latte šablóne sa použije pre tieto chyby požadovaný preklad. Pre zobrazenie odpovede s vyznačenými chybami slúži funkcia `convertCorrectedForLatte`, ktorá vráti odpoveď ako text obsahujúci HTML tagy pre správne zobrazenie, ktorú následne latte šablóna iba zobrazí. Táto funkcia postupne vytvára text a v prípade chyby ihneď označuje jednotlivé oblasti pridaním HTML tagov. Medzi jednotlivé tabuľky a integritné obmedzenia zároveň pridáva odriadkovanie pre čo najprehľadnejšie zobrazenie. Príklad takéhoto zobrazenia uvádzam na obrázku 3.2.

Pri vykreslení odpovede latte šablónou sa kontroluje jej validita a správnosť. V prípade, že odpoveď nieje valídna (pri normalizácií nastala chyba), je o tom učiteľ informovaný hláškou a zobrazí sa pôvodná odpoveď študenta bez farebného označenia.

3.5.3 Konvertovanie existujúcich odpovedí

Existujúce odpovede pre otázku typu transformácia do zjednodušeného relačného zápisu z aktálnej verzie boli uložené vo formáte JSON s odlišnou štruktúrou, preto ich bolo potrebné previesť na novšiu štrukúru. Odpovede sú uložené v databázi v tabuľke `test_student _answer`, pričom v stĺpci `answer` je uložená normalizovaná odpoveď a v tabuľke `answer_backup` pôvodná študentova odpoveď. Po detailnej analýze som zistil, že dáta sú v databázi nekonzistetné. Pri konvertovaní bolo teda potrebné zároveň dáta zjednotiť. Pri analýze som zistil, že odpovede môžem rozdeliť do 3 odlíšiteľných kategórií a podľa toho realizovať prevod.

1. V stĺpci `answer` sa nachádza originálna odpoveď a stĺpec `answer_backup` je prázdny. Tieto odpovede pochádzajú z úplne prvej verzie opravy. Stĺpec `answer_backup` bol pridaný až s druhou verziou opravy. V tomto prípade bolo potrebné originálnu odpoveď presunúť do `answer_backup` a zároveň túto odpoveď normalizovať a výsledok do uložiť do stĺpca `answer`.

2. V stĺpci `answer_backup` je korektne uložená originálna odpoveď. Túto odpoveď stačilo znovu normalizovať a následne uložiť výsledok do stĺpcu `answer`.

3. Pri vývoji v rámci predmetu Softwarový projekt 2, došlo ku chybe a pri niektorých odpovediach, sa originálna odpoveď študenta nenachádza v stĺpci `answer_backup`. V prípade týchto odpovedí je pôvodná odpoveď nenávratne stratená. V tomto prípade som sa rozhodol využiť triedu `OldTralex`, pomocou ktorej som normalizovanú odpoveď previedol na pôvodnú textovú odpoveď a následne znovu normalizoval pre získanie

aktuálneho formátu a uložil do stĺpca `answer` a pôvodnú odpoveď do `answer_backup`.

Podobne ako odpovede študentov bolo potrebné konvertovať aj referenčné odpovede k otázkam. Tieto sú uložené v tabuľke `test_answer` v stĺpcoch `answer`, kde je uložená originálna odpoveď a v stĺpci `tralex`, kde je uložená jej normalizovaná podoba. V tejto tabuľke sa takisto nachádzala nekonzistencia v dátach. Po analýze som dáta rozdelil na dve skupiny, ktoré boli následovne kovnvertované.

1. Normalizovaná odpoveď bola uložená v stĺpci `answer` a stĺpec `tralex` bol prázdny. V tomto prípade bola opäť originálna odpoveď nenávratne stratená. Preto som použil triedu `OldTralex` pre vytvorenie pôvodnej odpovedi. Táto bola následne normalizovaná a výsledok uložený do stĺpca `tralex` a pôvodná odpoveď do sĺpca `answer`.

2. Odpoveď je správne uložená a v stĺpci `answer` sa nachádza pôvodná odpoveď a v stĺpci `tralex` jej normalizovaná podoba. Pri tejto možnosti je potrebné iba aktualizovať normalizovanú podobu odpovede, normalizovaním pôvodnej odpovede a uložiť ju do stĺpca `tralex`.

Túto koverziu realizuje funkcia `transformFromOldTralex` triedy `Tralex`. Napriek tomu, že by sa táto funkcia mohla zdať po použití nepotrebná nieje tomu tak. Ak bude v budúcnosti prebiehať ďalší vývoj aplikácie Tralex a dôjde k zmene, ktorá spôsobí nekompatibilitu so staršími dátami, použitím tejto funkcie je možné dáta aktualizovať rýchlo a korektne.

3.6 Nasadenie aplikácie

Portál dbs.fit.cvut.cz používa pre nasadenie nástroj pre automatizáciu Ansible [24]. Ansible dokáže konfigurovať systémy, nasadzovať aplikácie a riadiť zložitejšie IT úlohy. Hlavné cieľe nástroja sú jednoduchosť a jednoduché používanie. Takisto kladie veľký dôraz na bezpečnosť a spoľahlivosť. Playbook je jazyk nástroja Ansible, ktorý sa používa pre konfiguráciu, nasadenie a riadenie. Môže definovať politiku, ktorú chceme aplikovať na vzdialené systémy alebo súbor krokov vo všeobecnom IT procese.

Ansible sa pre nasadenie portálu dbs.fit.cvut.cz nepoužíval vždy. Keď ale nastala požiadavka na reinštaláciu servera, bolo potrebné použiť nástroj, ktorý tento proces zautomatizuje, pretože je veľmi pravdepodobné, že takáto požiadavka sa bude v budúcnosti opakovať. Manuálna inštalácia serveru je časovo veľmi náročná činnosť. Zároveň pri nej môže vzniknúť množstvo chýb, spôsobené napr. zlou konfiguráciou. Pri tejto príležitosti bol vytvorený Ansible Playbook, ktorý zaobstará inštaláciu a konfiguráciu všetkých potrebných súčastí pre chod portálu dbs.fit.cvut.cz.

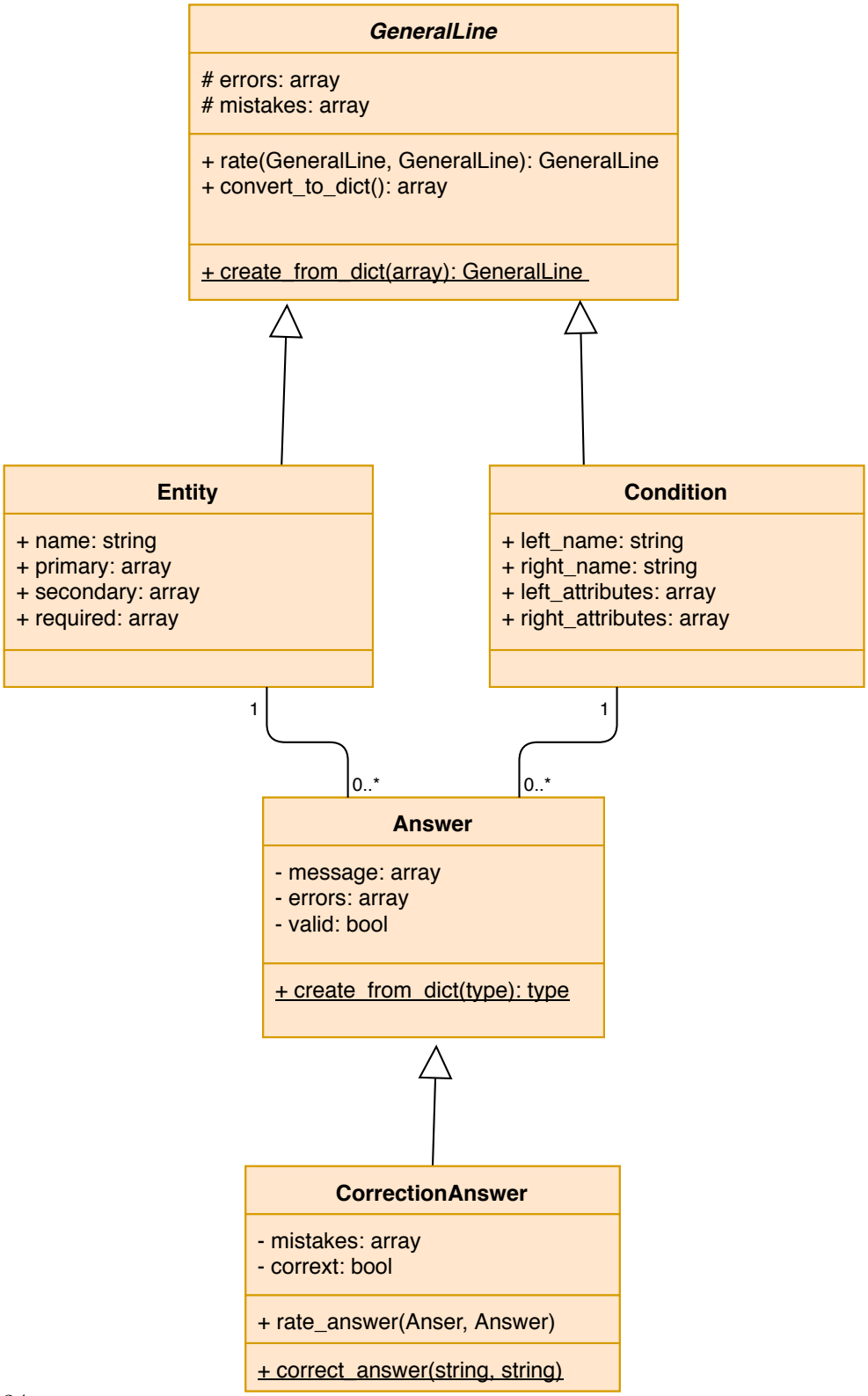
Mojou úlohou bolo rozšíriť túto konfiguráciu aj pre Tralex. Pre tento účel som vytvoril Ansible rolu s názvom `python3-lex`. Ansible rola je spôsob ako automaticky načítať konkrétne súbory s premennými, úlohy a manipulátor na základe známej súborovej štruktúry. Súborová štruktúra z ktorej je zložený tento Playbook:

```
python3-lex
├── defaults
│   └── main.yml
├── files
│   ├── setup.py
│   └── lex
│       └── Zdrojové súbory Tralexu
├── tasks
│   └── main.yml
├── templates
│   ├── lex.conf.j2
│   └── passenger_wsgi.py.j2
└── vars
    └── main.yml
```

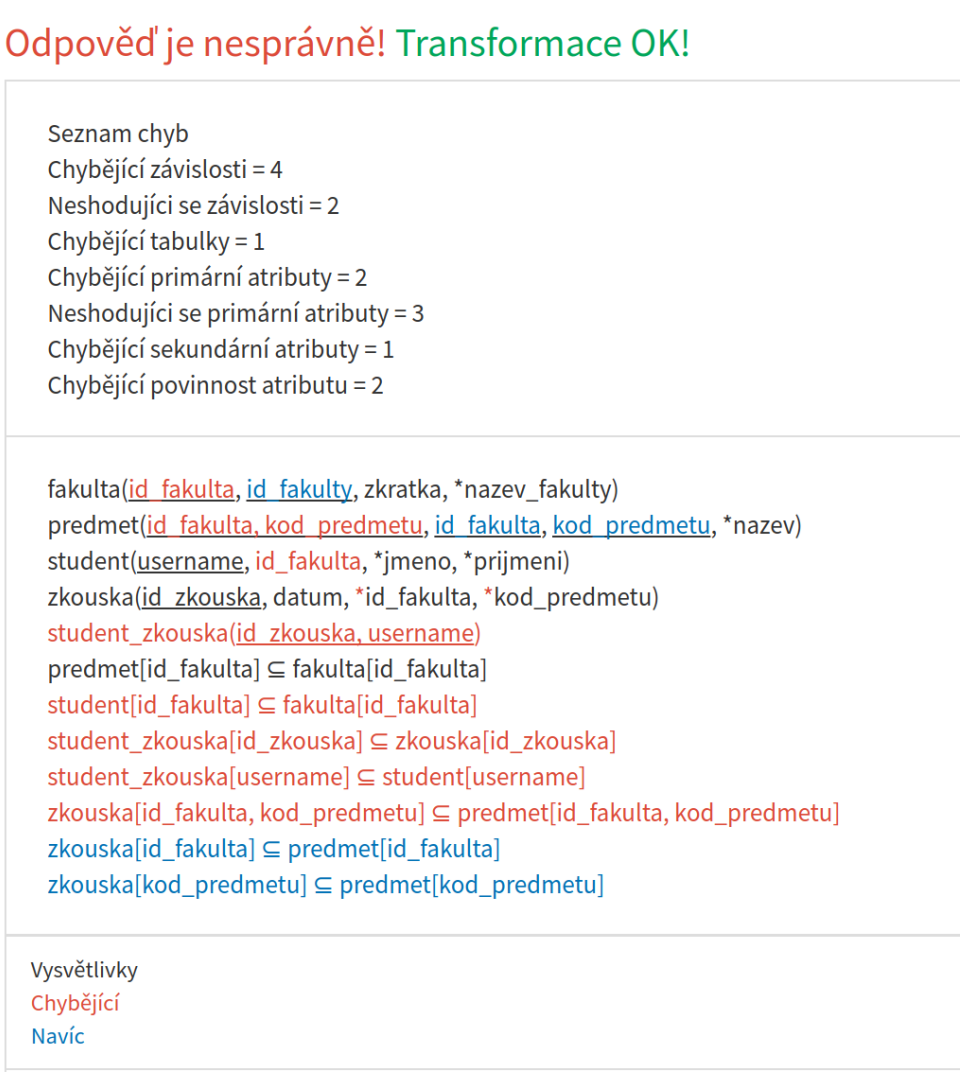
V týchto súboroch definujem virtuálne prostedie, do ktorého sa majú nainštalovať všetky potrebné balíky, ktoré Tralex vyžaduje. Následne definujem virtuálne prostredie v ktorom má Passenger[4] spustiť Tralex a port na ktorom bude Tralex dostupný. Po pridaní tohto Playbooku pre nasadenie portálu dbs.fit.cvut.cz som získal jedným spustením funčný portál dbs.fit.cvut.cz, rožšírený o aplikáciu Tralex bežiacu na adrese 127.0.0.1:7000 pripravenú na použitie.

[4]Passenger - aplikačný server, ktorý sa používa pre beh aplikácií v jazyku Python, Ruby a Node.js.

Obr. 3.1: Diagram tried reprezentujúcich odpoveď



Obr. 3.2: Ukážka zobrazenia odpovede po oprave aplikáciou Tralex



Kapitola 4

Testovanie

4.1 Automatické testovanie

Pri vývoji Tralexu som postupne vytváral automatické testy. Automatické testovanie je dôležitá súčasť vývoja aplikácií. Automatické testy umožňujú rýchle a spoľahlivé otestovanie kódu. Pre automatické testovanie sa v jazyku Python používa framework `unittest`[25]. Pri testovaní používam predovšetkým funkcionálne testovanie [18], ktoré netestuje malé celky kódu samostatne ale celú funkcionalitu ako celok. Zameral som sa na používanie reálnych vstupov, pre čo najdôkladnejšie otestovanie. Testy sa nachádzajú v súbore `tests.py`. Testy sú rozdelené do šiestich tried, ktoré sa líšia zložitosťou testu a takisto cieľom čo majú otestovať. Tieto triedy sú `EntityTests`, `ConditionTests`, `RealInputTests`, `TransformationErrorsTest`, `CorrectionTests` a `ApiTest`.

Trieda `EntityTests` testuje normalizáciu jednotlivých tabuliek. Jedná sa o testy špecificky zamerané vždy na nejakú vlastnosť, napr. správne normalizovanie primárnych atribútov, povinných atribútov a podobne.

Obdobne trieda `ConditionTests` testuje normalizáciu jednotlivých integritných obmedzení. Takisto sa jedná o špecificky zamerané testy.

Trieda `RealInputTests` testuje tranformáciu použitím skutočných odpovedí z testov.

Trieda `TransformationErrorsTest` testuje správne vyhodnotenie chyby pri normalizácií v prípade nesprávneho vstupu.

Trieda `CorrectionTests` testuje správnosť opravy odpovedí.

`ApiTest` testuje správne rozhranie API. Táto trieda nedokáže otestovať či aplikácia naozaj správne funguje na serveri ale vie otestovať či jednotlivé prístupové body vracajú správne návratové hodnoty.

Automatické testy zaručia, že pri rozšírení existujúcej funkcionality sa nezmení funkcionalita tej predchádzajúcej. Pre každú chybu, ktorá sa v aplikácií objavila, vznikol test, pre jej zamedzenie v budúcich verziách. Takýmto postupom sa stávajú automatické testy stále robustnejšie a účinejšie čo v konečnom dôsledku zaručí kvalitnejšiu a menej chybovú aplikáciu.

4.2 Testy pred nasadením

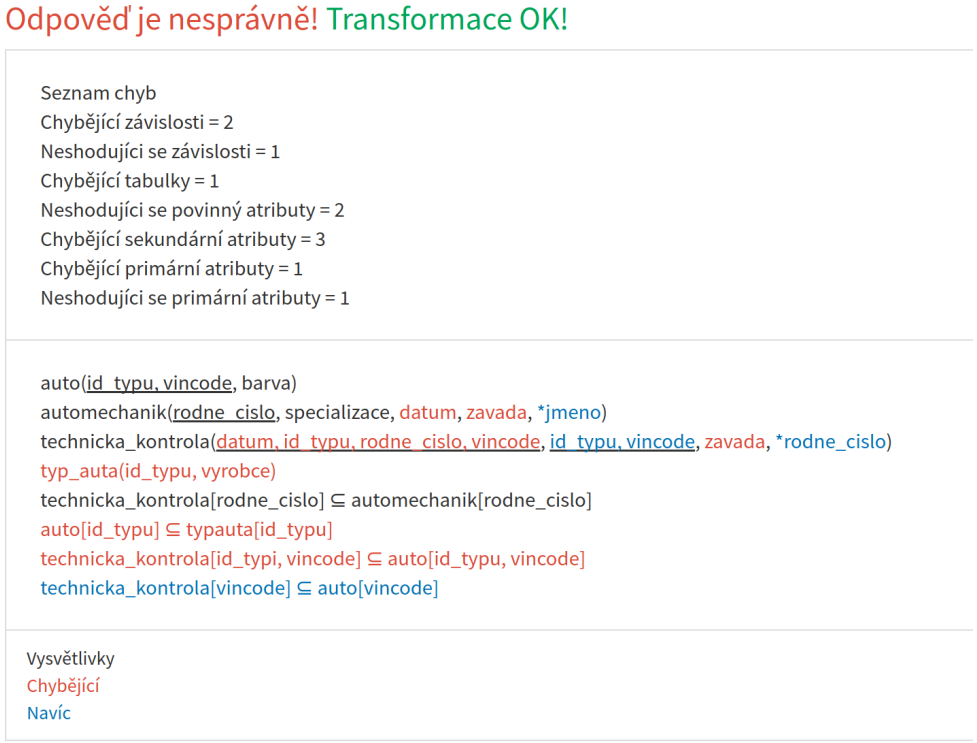
Pred nasadením prvej verzie pre reálne použitie bolo potrebné aplikáciu Tralex riadne otestovať. K tomuto účelu slúžia aj automatické testy, ktoré majú zaručiť bezchybnú funkčnosť. Zároveň som systém testoval aj ja. Na základe existujúcich odpovedí ale aj náhodných vytvorených pre tieto účely, som testoval celý proces pri testovaní študentov. Môj testovací scenár bol následovný: V prvom kroku som vytvoril otázku z pozície učiteľa a pridal som k nej referenčnú odpoveď. Následne som z tejto testovacej otázky vytvoril testovaciu šablónu. Z tejto šablóny som vytvoril test, ku ktorému som priradil študetov a tento test spustil. Potom som za jednotlivých študentov vypĺňal odpovede k tejto otázke. V poslednom kroku som postupne opravol otázky tohto testu a kontroloval správnosť opravy. Konkrétnym príkladom, ktorý som testoval bol vstup, ktorý v predchádzajúcej verzii nefungoval zobrazený na obrázku 1.2. Pre porovnanie prikladám obrázok 4.1 na ktorom je zobrazená oprava v novej verzii pre rovnaký vstup. V novej verzii je oprava tejto odpovede správne zobrazená, s označením jednotlivých chýb.

V prípade, že som odhalil chybu spôsobenú aplikáciou Tralex, ihneď som pridal tento vstup ku automatickým testom.

4.3 Testovanie v reálnom použití

Prvé testovanie novej aplikácie Tralex prebehlo v rámci testov, ktoré sa písali v rozmedzí 17.4.2018 a 19.4.2018. Týchto testov sa zúčastnilo 216 študentov. To predstavuje dostatočnú testovacie vzorku pre overenie či je moja aplikácia použiteľná a spoľahlivá. Pri tomto testovaní sa nevyskytla žiadna fatálna chyba, ktorá by znamenala pád aplikácie a teda nemožnosť študenta odoslať odpoveď. Pri tomto testovaní bolo odhalené nedostatočné informovanie študenta o nesprávnom vstupe. Namiesto konkrétnej hlášky s identifikáciou na ktorom znaku chyba nastala dostal študent iba hlášku o tom, že je v systéme chyba. Na základe toho som upravil spracovanie výnimiek na súčasný stav opísaný v kapitole 3.3.2. Zároveň sa ukázalo, že veľmi častá syntaktická chyba študentov pri odpovedi je podčiarknutie aj konca zátvorky pri označovaní atribútov ako primárne v tabuľke. To spôsobovalo označenie malého množstva odpovedí za nevalidné, čo malo za následok zníženie efektivity opráv. Z tohto dôvodu som upravil konečný automat pre rozpoznanie takejto chyby pridaním stavu `PRIMARY_END_SWITCH`. Zároveň som pridal nový typ hlášky akú môže

Obr. 4.1: Ukážka opravy vstupu, ktorý v predchádzajúcej verzii spôsobil chybu



študent pri odpovedaní dostať. O tejto chybe je študent informovaný hláškou v oranžovom okne namiesto červenom, pretože to výsledok tranformácie neovplivňuje. Pri analýze dôvodu častého podtrhávania zátvorky som odhalil, že štýl písma aký sa používal, nezobrazol podtrhnutie pod zátvorkou. Preto študent nemohol vedieť, že sa dopustil chyby. Kvôli tomu bol zmenený štýl písma aby bolo podtrhnutie zátvorky správne zobrazené.

Ďalšia vlna testovania prebehla o týždeň neskôr medzi 24.4.2018 a 26.4.2018 so zapracovanými opravami z predchádzajúceho testovania. V tomto termíne písalo testy 223 študentov. Pri tomto testovaní bola odhalená chyba pri oprave integritných obmedzení. V prípade ak sa zdrojová tabuľka zhodovala, ale cieľová bola rozdielna v oprave sa učiteľovi zobrazilo iba `[] ⊆ []`, namiesto správnych tabuliek. Túto chybu som takisto odstránil a oprava funguje tak ako je popísaná v kapitole 3.4. Zároveň bolo odhalené, že chýba identifikácia znaku na ktorom nastala chyba v prípade chyby typu `UNKNOWN_ERROR`. Túto identifikáciu som do aplikácie Tralex takisto pridal. Pri tomto testovaní vznikla požiadavka od učiteľov, aby sa pri oprave, v prípade, že študent iba neoznačil atribút ako povinný, neoznačil ako chýbajúci, ale aby sa ako chýbajúca zobrazila hviezdička pred týmto atribútom. Túto požiadavku som takisto zrealizoval ako je popísané v kapitole 3.4.

4.4 Výsledky testovania

Po oprave chýb a realizovaní úprav, ktoré vyplynuli z reálneho testovania 4.3, prebehlo opätovné kovertovanie odpovedí pomocou funkcie `transfromFromOld-Tralex`. Po analýze týchto odpovedí som zistil nasledujúce výsledky. Iba 12 odpovedí z celkového počtu 439 je po normalizácií označených ako nevalidné. Tu je treba dodať, že vo všetkých 12 prípadoch sa v odpovedi naozaj vyskytovala syntaktická chyba. To znamená neúspech normalizácie iba v 2,7% prípadov. Tým pádom sa mi podarilo naplniť hlavný cieľ a to zaručiť správnosť pri opravovaní odpovedí učiteľom, čo bol hlavný nedostatok predchádzajúceho riešenia. Zároveň bolo 73 odpovedí opravených úplne automaticky. To znamená, že učiteľom pre manuálnu opravu ostalo 366 odpovedí z čoho v 354 prípadoch bolo použité zobrazenie s označením chýb, zjednodušujúce opravu. To znamená, že 97% odpovedí bolo zobrazených tak aby učiteľovi uľachčili ich opravu. To považujem za splnenie cieľu, ohľadom zjednodušenia opráv otázok typu transformácia konceptuálneho modelu do zjednodušeného relačného zápisu.

Záver

Cieľom mojej bakalárskej práce bolo navrhnúť a zrealizovať rozšírenie portálu dbs.fit.cvut.cz o normalizáciu odpovedí typu transformácia konceptuálneho modelu do zjednodušeného relačného zápisu. Zároveň bolo mojim cieľom zjednodušiť učiteľom opravu týchto otázok vhodným zobrazením študentovej odpovedi pri oprave. Tieto cieľe sa mi podarilo naplniť.

Aplikácia Tralex, ktorá vznikla v rámci tejto bakalárskej práce realizuje normalizáciu a opravu odpovedí. Táto aplikácia je už aktuálne (od 15.5.2018) nasadená a používaná v systéme portál dbs.fit.cvut.cz. Nové zobrazenie v portáli dbs.fit.cvut.cz pri manuálnej oprave odpovedí je prehľadnejšie, čo vedie k zjednodušeniu tejto činnosti pre učiteľov. Výsledky testovania zároveň preukázali funkčnosť a hlavne spoľahlivosť tejto aplikácie. Prepacované spracovanie chýb na vstupe presne informuje ohľadom chyby a pozície, na ktorej táto chyba nastala. Študenti sú tak pri písaní testov, kedy je čas veľmi drahocenný, presne informovaný ohľadom typu chyby a pozície na ktorej sa v odpovedi nachádza.To v konečnom dôsledku vedie k menšiemu počtu syntaktických chýb a teda menšiemu počtu zlyhania normalizácie.

V budúcnosti by sa dala táto práca rozšíriť o automatické ohodnotenie chýb na strane portálu dbs.fit.cvut.cz. Ďalšia možnosť rozšírenia je pridanie našepkávania názvov tabuliek študentovi pri písaní testu. Tým by sa mohli odstrániť chyby z nepozornosti a zvýšiť percento automaticky opravených odpovedí.

Verím, že sa učiteľom bude výsledok mojej práce páčiť, pretože by mala v konečnom dôsledku viesť k šetreniu ich drahocenného času.

Literatúra

[1] prof. Ing. Jan Holub, P.: 1. Základní pojmy [online]. 2017, [cit. 1.5.2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-AAG/_media/lectures/01/bi-aag-01-zakladni_pojmy.pdf

[2] prof. Ing. Jan Holub, P.: 11. Konečný automat jako lexikální analyzátor, lex/flex generátory [online]. 2017, [cit. 1.5.2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-AAG/_media/lectures/11/bi-aag-11-lexikalni_analyzator.pdf

[3] Bohumír Zoubek, T. K.: Softwarový proces [online]. 2009, [cit. 2.5.2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-SI2/_media/lectures/1_softwareprocess.pdf

[4] Ing. Michal Valenta, P.: Normalizace a normální formy. 2017, [cit. 14.5.2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-DBS/_media/lectures/b162/dbs_lec_8_cz_normalizace.pdf

[5] Ing. Michal Valenta, P.: Transformace konceptuálního modelu na relační [online]. 2018, [cit. 2.5.2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-DBS/_media/lectures/b162/dbs_lec_5_cz_transformace.pdf

[6] Pejša, P.: *Systém pro podporu testování studentů v BI-DBS*. Bakalárska práca, České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

[7] Tomáš Krátký, B. Z.: Architektura a design [online]. 2016, [cit. 14.5.2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-SI2/_media/lectures/4_architecturedesign.pdf

[8] prof. Ing. Jan Holub, P.: 1. Převody mezi RG, RV a KA [online]. 2017, [cit. 5.5.2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-AAG/_media/lectures/05/bi-aag-05-prevody-rg-rv-ka.pdf

[9] Foundation, T. P. S.: Unicode HOWTO [online]. 2018, [cit. 1.5.2018]. Dostupné z: https://docs.python.org/3/howto/unicode.html

[10] Beazley, D. M.: Documentation [online]. 2018, [cit. 1.5.2018]. Dostupné z: http://www.dabeaz.com/ply/ply.html

[11] Kubiš, M.: *Překladač z relační algebry do SQL*. Bakalárska práca, České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

[12] Team, P.: Welcome to Flask [online]. 2010, [cit. 1.5.2018]. Dostupné z: http://flask.pocoo.org/docs/1.0/

[13] Foundation, N.: Documentation [online]. 2018, [cit. 1.5.2018]. Dostupné z: https://doc.nette.org/cs/2.4/

[14] Kotonya, G.; Sommerville, I.: *Requirements Engineering*: *Processes and Techniques*. John Wiley & Sons, Inc., 1998, ISBN 0471972088.

[15] Tomáš Krátký, B. Z.: Requirements Engineering [online]. 2007, [cit. 2.5.2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-SI2/_media/lectures/3_requirements_intro.pdf

[16] Conservancy, S. F.: About [online]. 2018, [cit. 2.5.2018]. Dostupné z: https://git-scm.com/about/

[17] Robin Obůrka, P. P.: Git, the information manager from hell. 2017, [cit. 14.5.2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-GIT/_media/lectures/gitzs2017.pdf

[18] Tomáš Krátký, B. Z.: Testování [online]. 2017, [cit. 2.5.2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-SI2/_media/lectures/7_testing.pdf

[19] Tomáš Krátký, J. T., Bohumír Zoubek: Design and Construction [online]. 2016, [cit. 3.5.2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-SI2/_media/lectures/6_design_construction.pdf

[20] prof. Ing. Jan Holub, P.: 7. Zásobníkové automaty [online]. 2017, [cit. 14.5.2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-AAG/_media/lectures/07/bi-aag-07-zasobnikove_automaty.pdf

[21] Foundation, P. S.: pip 10.0.1 [online]. 2018, [cit. 3.5.2018]. Dostupné z: https://pypi.org/project/pip/

[22] Foundation, P. S.: JSON encoder and decoder [online]. 2018, [cit. 6.5.2018]. Dostupné z: https://docs.python.org/3/library/json.html

[23] Dowling, M.: Guzzle Documentation [online]. 2015, [cit. 10.5.2018]. Dostupné z: http://docs.guzzlephp.org/en/stable/

[24] Red Hat, I.: Ansible Documentation [online]. 2018, [cit. 8.5.2018]. Dostupné z: http://docs.ansible.com/ansible/latest/index.html

[25] Foundation, P. S.: Unit testing framework [online]. 2018, [cit. 8.5.2018]. Dostupné z: https://docs.python.org/3/library/unittest.html

Dodatok A

Zoznam použitých skratiek

JSON JavaScript Object Notation

HTML HyperText Markup Language

API Application programming interface

REST Representational state transfer

DBS Databázové systémy

PLY Python Lex-Yacc

BNF Backus-Naur Form

IT Information Technology

Dodatok B

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy aplikácie Tralex
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `BP_Machala_Filip.pdf` ..... text práce vo formáte PDF