Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Importér verejných datasetov

Dokumentácia inžinierskeho diela

Tímový projekt

Vedúci tímu: Ing. Jakub Šimko Phd.
Ing. Marek Šurek

Členovia tímu: Bc. Matúš Brandajský
Bc. Lukáš Belaj
Bc. Božik Tomáš
Bc. Michal Hrutka
Bc. Tatiana Šlesariková
Bc. Janka Fabušová
Bc. Gabriel Csollei

Študijný program: ISS
November 2017

Obsah

Zoznam obrázkov:

1. Úvod

Obsahom tejto dokumentácie je pohľad na technickú stránku projektu vypracovaného v rámci predmetu Tímový projekt na tému Importér verejných datasetov.

Cieľom projektu je vytvorenie webového nástroja, ktorý nie-technickému používateľovi (napr. úradníkovi) umožní jednoduchý import datasetu určeného na zverejnenie na data.gov.sk. Nástroj na vstupe dostane dataset vo formáte s neznámou sémantikou. Úlohou nástroja je dataset transformovať tak, aby zodpovedal centrálne platnej schéme, na základe ktorej sa potom jednotlivé zverejnené datasety môžu prepájať.

Na tvorbe tohto projektu sa podieľajú Fakulta informatiky a informačných technológií, Úrad podpredsedu vlády SR pre investície a informatizáciu a softvérová firma xIT s.r.o.

2. Globálne ciele projektu na zimný semester

V rámci zimného semestra sme si s produktovým majiteľom vytýčili tieto ciele, ktoré by sme chceli stihnúť. Sú to:

- **Navrhnúť/implementovať používateľské rozhranie** – Tu je potrebné používateľské rozhranie upraviť v čo najprijateľnejšom rozhraní. Úradníčka musí vedieť ovládať rozhranie.
- **Navrhnúť spôsob ako zo vstupu (testový dokument) vytvoriť triplety** – Väčšina formátov je uložená práve v exceli alebo v csv súbore a preto je potrebné, aby sme sa zamerali na jednotnú podobu spracovania dokumentov. Na takúto operáciu nám slúži práve OWL schéma, do ktorej sa dáta budeme snažiť spracovávať
- **Stiahnutie centrálneho modelu a jeho spracovanie pre porovnávanie -** na to aby bola aplikácia funkčná je tento krok dôležitý. Centrálny model spracujeme a uložíme do prijateľnej formy.
- **Pokročilejšie metódy spracovania -** Pokúsime sa navrhnúť pokročilejšie metódy spracovania atribútov zo vstupných súborov. Samozrejme, tu bude potrebné prečítať a spracovať veľa výskumného textu, aby sme správne pochopili problém.

3. Analýza

Pretože sme sa s týmto problémom ešte nikdy nestretli alebo sme sa s ním stretli len okrajovo, bolo potrebné spraviť aj kus analýzy, aby sme mohli problém implementovať.

Prvým problémom je samotné sémantické spracovanie dát. Tu sme začali pracovať na našej analýze v tom smere ako nám odporučili naši vedúci projektu. Počas tejto analýzy sme sa stretli s pojmami ako RDF[1], OWL[2] alebo triplety. Preštudovali sme si aj existujúce riešenia sémantického spracovania dát, ako napríklad GraphDB[3] alebo VOWL[4].

Po preštudovaní problematiky bolo potrebné analyzovať konkrétnu problematiku, s ktorou budeme pracovať. Bolo potrebné naštudovať slovenské štandardy, čo je to centrálny model a ako s ním pracovať. Tu nám pomohol zdroj od vedúceho, kde boli všetky potrebné informácie k centrálnemu modelu[5]. Následne bolo potrebné pozrieť sa na dáta, s ktorými budeme pracovať[6]. Tu sa nachádzajú všetky vstupné datasety, ktoré je možné spracovať. Samozrejme sme sa pozreli ako to funguje v iných krajinách a na akej úrovni majú datasety iné krajiny, konkrétne Spojené kráľovstvo[7].

---

[1] https://www.w3.org/RDF/

[2] . https://www.w3.org/OWL/

[3] https://hub.docker.com/r/ontotext/graphdb/

[4] http://vowl.visualdataweb.org/

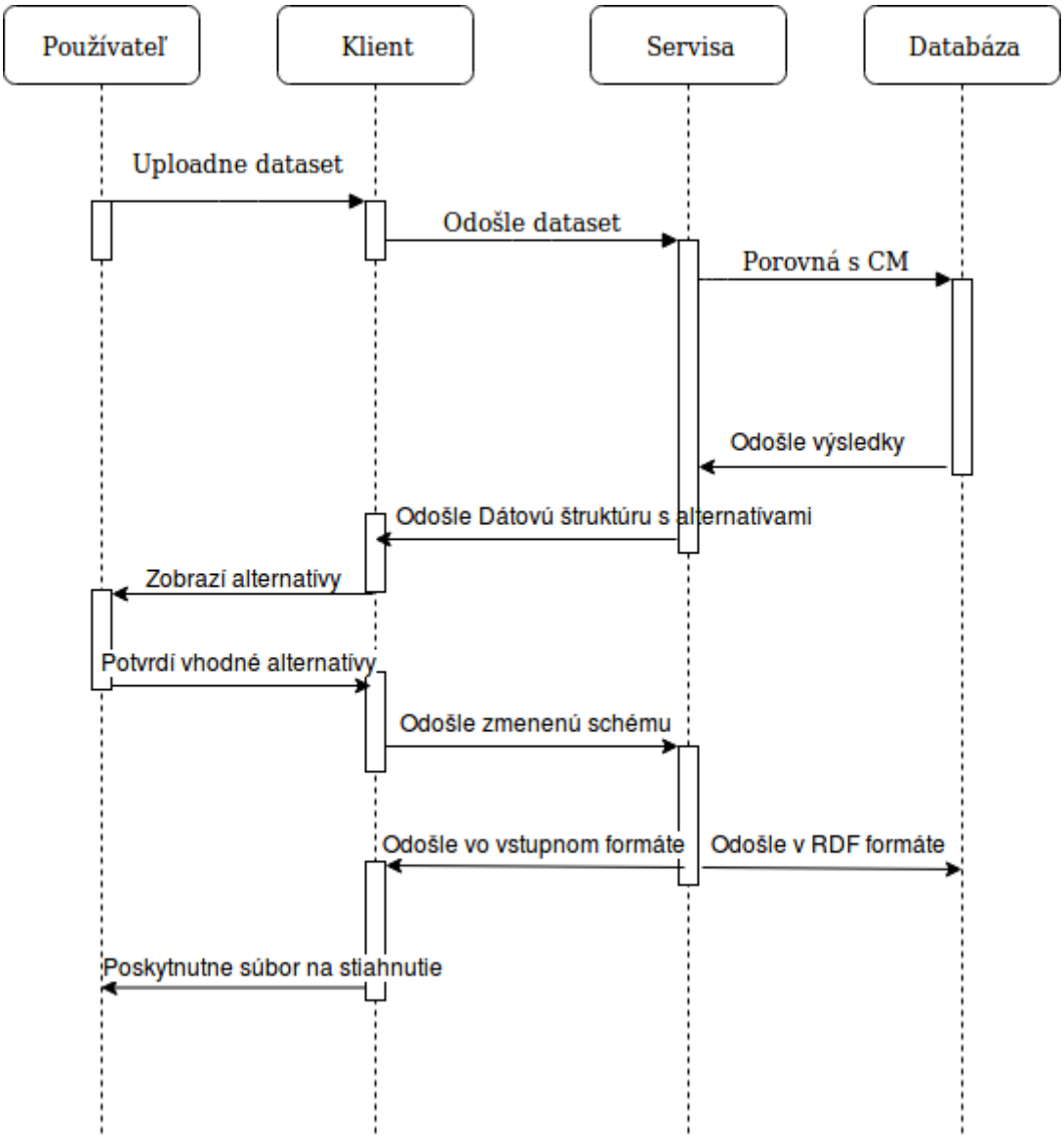
[5] https://wiki.finance.gov.sk/pages/viewpage.action?pageId=16416782

[6] data.gov.sk

[7] data.gov.uk

4. Návrh

Počas návrhu bolo nutné vytvorenie sekvenčného diagramu, ktorý nám ukazuje ako budeme postupovať a ako bude prebiehať chod našej aplikácie. Pomocou neho sme si vytvorili základný obraz o tom, ako bude naša aplikácia



Obrázok 1:Sekvenčný diagram: Celkový chod systému.

Na sekvenčnom diagrame je vidieť celý chod nášho systému:

1. V prvom kroku je nutné, aby používateľ systému(úradník) nahral dataset, ktorý chce spracovať.

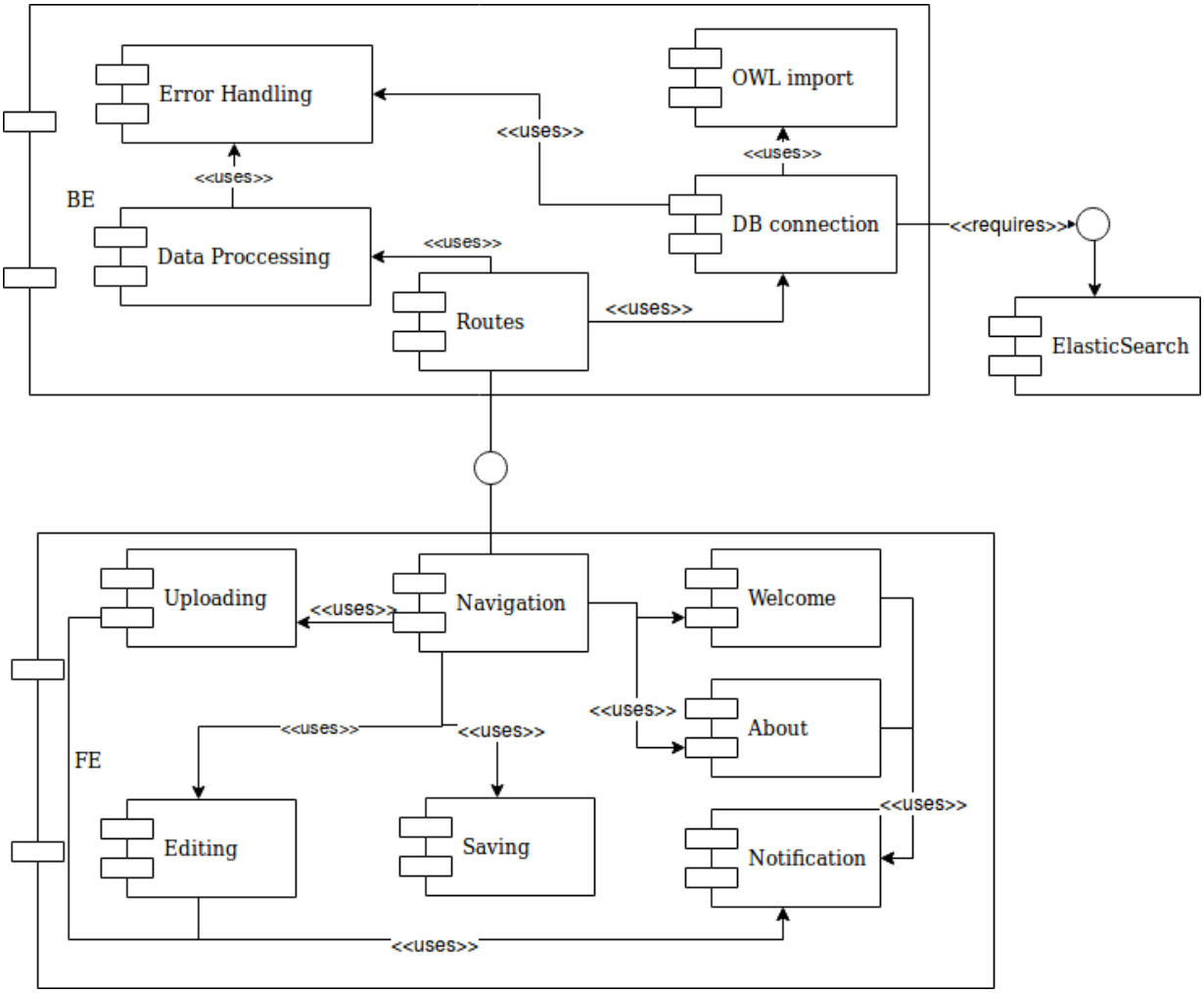
2. Následne sa dataset odošle na Backend, kde sa údaje spracujú s pomocou porovnávanie s centrálnym modelom. Tu sa zistia alternatívy, ktoré sa následne posielajú znovu na Frontend
3. Údaje sa zobrazia používateľovi na Frontende a on musí vybrať z každej alternatívy, ktorá mu bola poskytnutá pre daný atribút.
4. Po vybratí znovu Frontend odošle údaje na backend. Backend údaje spracuje do dvoch podôb. V prvej podobe odošle dataset na Frontend vo vstupnom formáte. Tu klient ponúkne používateľovi stihanuť si upravenú verzu svojho datasetu. V druhej podobe sa údaje uložia do RDF a následne sa uložia do našej databázy.

K nášmu projektu sme vytvorili aj klikateľný prototyp[8], ktorý korešponduje so sekvenčným diagramom. Tu je zapracovaný aj UX dizajn, na ktorom sme sa zhodli s našimi vedúcimi.

[8] https://janyfabusova.wixsite.com/team-14

5. Implementácia

5.1 Celkový pohľad na systém



Obrázok 2: Celkový pohľad na systém

Obrázok nad textom vyjadruje celkový pohľad na systém. Aplikácia je realizovaná pomocou klient - server architektúry. FE vyjadruje Frontend, teda klientskú časť, ktorú využíva používateľ a potom BE, čo je vlastne backend, teda serverovú časť. Klient komunikuje so serverom pomocou HTTP protokolu cez web rozhranie. Obrázok ďalej vyjadruje aj závislosti komponentov.

Identifikované komponenty spĺňajú nasledovné funkcie:

Navigation a Routes - Komponenty, ktoré slúžia na komunikáciu medzi klientom a serverom. Komponent Routes momentálne vyjadruje dve časti, prvou je prijatie nahratého súboru a druhé

je prijatie alternatív, po správnom vybratí dokumentu. Vo frontendovej časti je Navigation rozdelený do viacerých balíkov, ktoré sa taktiež starajú o presmerovávanie na stránke. Sem patrí napríklad objekt Router ktorý ponúka angular a jeho použitie je veľmi jednoduché. Volania backendových apín sú rozdelené do viacerých služieb, ktoré dedia od hlavnej AbstractHTTP Service. Ostatné frontendové komponenty sú popísané nižšie.

Data processing - Tento komponent slúži na spracovávanie nahratého datasetu. Rozpozná o aký dokument sa jedná a implementuje ho do java objektov.

Error handling - Tento komponent sa stará o chybové hlášky, ktoré máme pevne definové a je potrebné ich odoslať aj na Frontend. Pomocou nich sa následne používateľovi vypisujú chybové hlášky.

DB Connection - Komponent, ktorý zabezpečuje komunikáciu a pripojenie s Elasticom, zároveň robí aj všetky vyhľadávania nad elasticom.

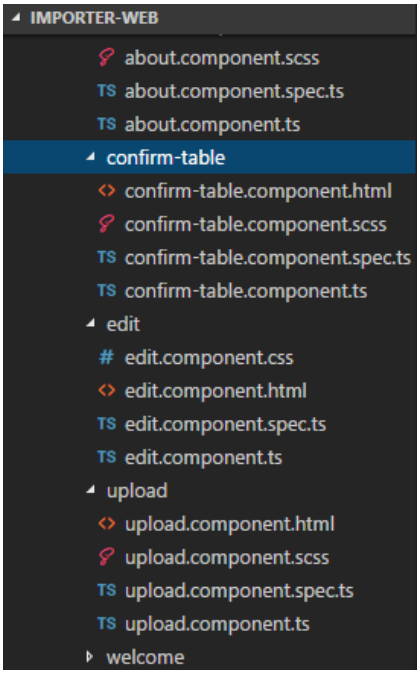
OWL Import - Komponent na spracovanie centrálneho modelu.

ElasticSearch - momentálne slúži na uchovávanie Centrálneho modelu a vyhľadávanie v ňom.

5.2 Frontend

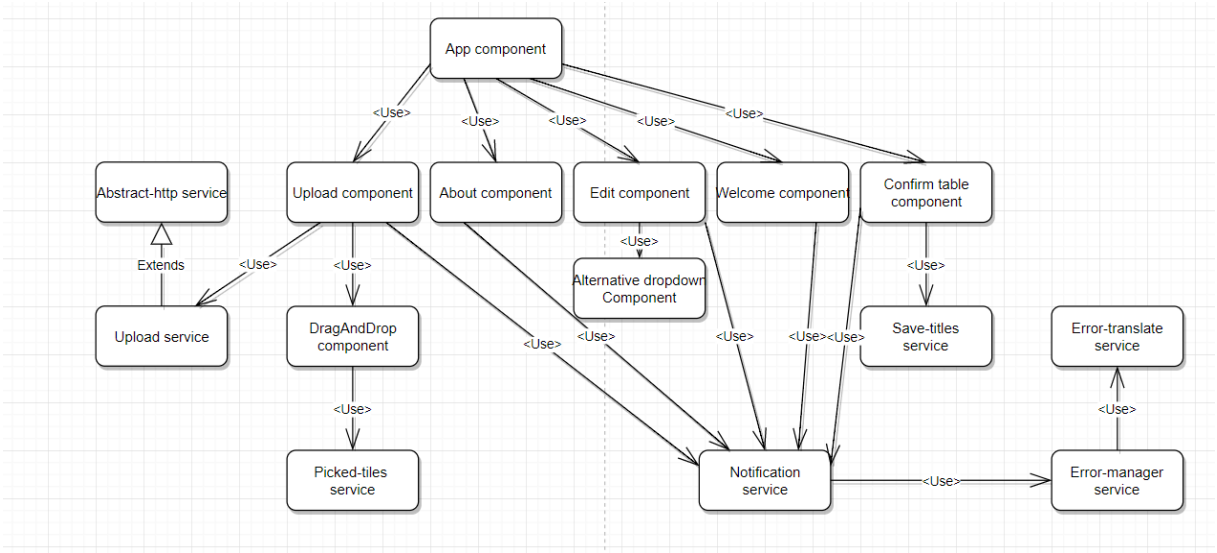
Frontendová časť aplikácie je postavená na modernej technológii Angular 4 od spoločnosti Google, ktorá komunikuje s API za pomoci REST-ových služieb. Tento rámec nám prináša množstvo výhod, na základe ktorých sme sa pre neho rozhodli. Okrem jeho rýchlosti, jednoduchosti a flexibilite má rôzne stratégie, ako tvoriť aplikácie pre rôzne platformy, či už pre webové prehliadače alebo mobilné zariadenia. Kód je členený na jednotlivé komponenty, čím sa stáva prehľadný a ľahko zrozumiteľný. Aplikácia je postavená na Angular-CLI, spolu s čím prichádzajú výhody spojené s inštalovaním závislostí, testovaním a budovaním aplikácie (angl. build application) pre produkčnú alebo vývojársku verziu.

Na obrázku Náhľad na štruktúrované priečinky podľa funkcie je možné vidieť, ako je zdrojový kód rozdelený na priečinky podľa funkcií, ktoré obsahujú komponent, HTML šablónu, službu, CSS štýly a používané triedy.



Obrázok 3: Náhľad na štruktúrované priečinky podľa funkcie

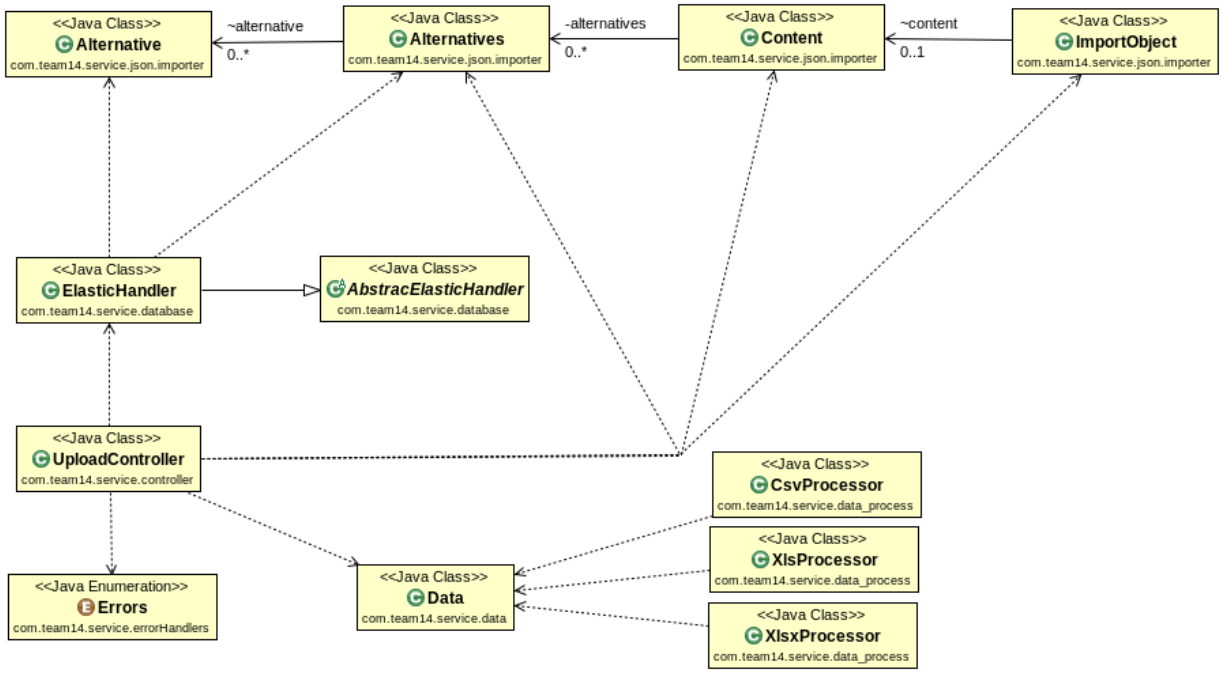
Na ďalšom obrázku (Diagram komponentov a služieb) môžeme vidieť základné služby a komponenty používané v našej aplikácií. Hlavný App komponent využíva všetky ostatné komponenty na základe aktuálnej URL. Upload service je služba, ktorá sa stará o upload súborov. Dedí od služby Abstract-http servive, ktorá poskytuje základné metódy na posielanie HTTP requestov. Ďalej tu je dôležitý Upload komponent, ktorý obsahuje komponent na výber súbor DragAndDrop component a taktiež aj Edit component, ktorý zabezpečuje výber všetkých atribútov. ConfirmTable component slúži na zobrazenie celkového prehľadu zvolených atribútov a ich následné odoslanie na data.gov, poprípade stiahnutie finálne upraveného dokumentu. Všetky komponenty využívajú službu na notifikáciu používateľa Notification service, ktorá spracuje prijaté chybové hlášky to server, umožňuje lokalizáciu aplikácie do viacerých jazykov a taktiež sa stará o jednoduché logovanie do konzoly.



Obrázok 4: Diagram komponentov a služieb

5.3 Backend

Backend je postavený na programovacom jazyku JAVA. Tento jazyk nám prináša mnoho výhod a preto sme sa preň rozhodli. Jednou z výhod je škálovateľnosť aplikácie. Zvolili sme si framework Spring, ktorý nám uľahčuje celkovú prácu a otvára nám možnosti využitia aplikácie. Na komunikáciu používame REST služby. Aplikácia využíva buildovací framework maven, ktorý nám uhahčuje prácu a dovoluje nám integráciu s Dockerom, ktorý aktívne využívame. Dole na obrázku je vidieť celkový model našej aplikácie.



Obrázok 5: Celkový model BE

Data - V tejto triede sa najskôr zistí typ uploadnutého súboru a následne sa tento súbor z byteového poľa presype do vnútornej reprezentácie (List Stringov) a zvlášť sa uloží prvý riadok zo súboru ktorý predstavuje názvy stĺpcov.

CsvProcessor - Ak sa v triede "Data" zistí, že uploadnutý súbor je CSV, tak sa pomocou tejto triedy uloží do vnútornej reprezentácie pomocou frameworku Jackson.

XlsProcessor - Ak sa v triede "Data" zistí, že uploadnutý súbor je XLS, tak sa pomocou tejto triedy uloží do vnútornej reprezentácie pomocou frameworku Apache POI a tiež sa zvlášť uloží prvý riadok súboru ako pri CSV.

XlsXProcessor - Podobný princíp ako pri XlsProcessor.

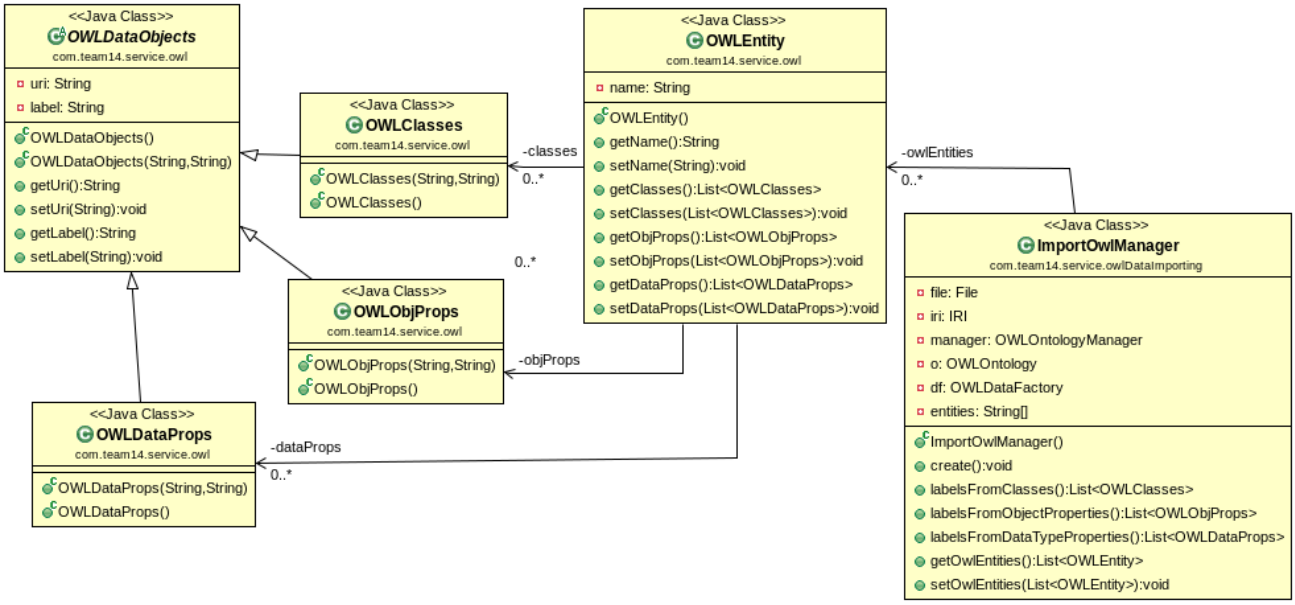
UploadController - Táto trieda slúži ako route a posielajú sa tu všetky requesty na Backend, vykonáva sa tu celé spracovanie údajov na vyššej úrovni.

Errors - Zachovávajú sa tu všetky chybové hlášky, ktoré sa posielajú na FE

ElasticHandler a AbstracElasticHandler - Tieto objekty zabezpečujú komunikáciu medzi BE a databázou.

ImportObject - Objekt, ktorý sa odosiela po uploadnutí ako JSON object, tento object obsahuje aj nasledovné objekty: Content, Alternatives a Alternative a majú nasledovnú štruktúru:

```
{
 "error" (string),
 "content" {
   "titles" (array, string),
   "data" (array, array),
   "alternatives" (array) {
     "alternative" (array) {
       "type" (string),
       "rank" (float)
     }
   }
 }
}
```



Obrázok 6: Spracovanie centrálneho modelu

OWLDataObjects - Táto trieda obsahuje URI a label (meno). Predstavuje jeden OWL objekt.

OWLEntity - OWLEntity vyjadruje jednu entitu v Centrálnom modeli, každá takáto entita obsahuje: Classes, Object properties a Data properties. Pretože má každá z týchto "tried" iné správanie, tak sme si vytvorili tri rozdielne triedy, ktoré budeme v budúcnosti využívať.

ImportOwlManager - Táto trieda spracuje celý centrálny model a spracuje ho do listu triedy OWLEntity. Následne je možné nahraťCentrálny model do databázy.

6. Testovanie

6.1 Testovanie na Backende

V rámci testovania uplatňujeme testovanie pomocou Unit testov. Na backende sme použili na testovanie framework JUnit. Prebehlo viacero testovacích scenárov, ktoré sme si rozdelili do nasledovných kategórií:

1. Testy na správne spracovanie:

- CSV: V tomto teste sa kontroluje či sa správne načítal csv súbor, ktorý sa z byte-ového poľa uloží do vnútornej reprezentácie pomocou frameworku Jackson.
- XLS: V tomto teste sa kontroluje či sa správne načítal xls súbor, ktorý sa z byte-obého poľa uloži do vnútornej reprezentácie pomocou frameworku Apache POI.
- XLSX: Podobný test ako pri XLS, len na súbor typu xlsx.

Tiež sa tu kontroluje či sa správne zvlášť uložila hlavička zo súborov (prvý riadok) a zvlášť dáta za súborov.

2. Testy na správne určenie typu nahratého súboru: V tomto teste sa skontroluje či sa správne určili typy súborov pre csv, xls a xlsx.

3. Test na pripojenie do Elasticsearch a vyhľadanie alternatív pre testovací súbor: V tomto teste sa testuje pripojenie k Elasticsearch a následné vyhľadanie názvov stĺpcov z testovacieho súboru v Elasticsearch.

4. Test na OWL: V tomto teste sa testuje či sa načítali všetky typy entít z OWL súborov.

Ako separátny test sme vytvorili testovací skript v jazyku Python. V repozitári tohto skriptu je zložka, ktorá obsahuje skupiny testovacích súborov, organizovaných tak, že každá skupina je v samostatnej zložke a musí obsahovať súbor, ktorého názov končí slovom *original*. Tento súbor je vo formáte, ktorý je ideálny pre publikáciu. Ostatné súbory v zložke vychádzajú z tohto súboru, ale obsahujú určité chyby, ktoré má byť naša aplikácia schopná riešiť.

Po spustení skript prejde všetky zložky, načíta dáta zo súboru so správnou formou dát a tie porovnáva s dátami, ktoré vráti naša aplikácia po spracovaní dát z ostatných súborov. Následne vypíše sumár výsledkov. Pre hlbšie preskúmanie výsledkov je vhodná verzia v jupyter notebook-u, kde sa dá zavolať funkcia na vypísanie všetkých dát, ktoré naša aplikácia

nedokázala správne spracovať.

6.2 Testovanie na Frontende

Kedže na frontende používame framework Angular 4, ktorý je populárny, veľmi používaný a existuje k nemu množstvo nástrojov na testovanie, tak sme sa rozhodli použiť Jasmine, testovací nástroj. Tento nástroj sme si zvolili hlavne kvôli jeho výbornej kompatibilite s angularom. Výhodou je takisto samostatné testovanie jednotlivých komponentov a oddelenie ich závislostí. To znamená, že každý komponent je vytvorený samostatne a následne sú otestované všetky funkcie, ktoré obsahuje. Jasmine podobne ako Angular vyrieši všetky nevyhnutné závislosti testovaného komponentu, čo je nesmierne užitočné. Každá testovaná funkcia je vždy zavolaná a v prípade funkcií s argumentmi sú s nimi aj zavolané a následne je buď otestovaný výstup funkcie alebo zmena stavu komponentu, v ktorom sa funkcia nachádza.