SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Tím 16 - MI16

Inteligentný importér verejných datasetov

Inžinierske dielo

Vedúci tímu:
Ing. Jakub Šimko, PhD.

Členovia tímu:
Bc. Ladislav Bari
Bc. Jaroslav Schvantner
Bc. Adam Ševčík
Bc. Adam Talian
Bc. Filip Varga
Bc. Milan Vaško
Bc. Jana Vrabľová

Obsah

Úvod

V tejto dokumentácií poskytujeme informácie o vytvorenom softvérovom systéme “Inteligentný importér verejných datasetov”. Dokumentácia poskytuje technický pohľad na vytvorený softvér, vrátane diagramov a opisov jeho systémovej architektúry a detaily modulov tvoriacich systém. Neobsahuje pohľad na tímové metodiky, riadenie tímu či roly jednotlivých členov.
V prvej kapitole je poskytnutý úvod k tomuto dokumentu.
Druhá kapitola zahŕňa ciele projektu, ktoré boli vytýčené na aktuálny semester.
Tretia kapitola obsahuje pohľad na systém ako celok, diagramy tried a dátové modely systému, vrátane sekcií venovaných detailnejšie kľúčovým komponentom ako RDF či API.
Nasledujú prílohy vo forme príručiek na spúšťanie a prácu so softvérom a linky na rozsiahlu technickú dokumentáciu.

Globálne ciele projektu na zimný semester

V existujúcom riešení bolo možné nahrať základné typy súborov, a to CSV, XLS, XLSX. Existujúce riešenie súbory spracovalo, uložilo, identifikovalo názvy stĺpcov pomocou Elasticsearch-u a umožnilo ich upraviť a navoliť. Záznam o dokumente sa na záver uložil do databázy a softvér poskytol možnosť spracovaný dokument stiahnuť v pôvodnom formáte.
V zimnom semestri sme sa zamerali na detailnejšiu prácu s centrálnym modelom údajov a spracovanie a generovanie RDF súborov. Vykonali sme viaceré korekcie a opravy na existujúcom frontende aj backende.
Rozhodli sme sa vykonať výraznejší refaktoring existujúcich kódov, predovšetkým na frontende. Súčasťou toho je úprava dizajnu frontendu podľa Jednotného dizajn manuálu verejnej správy, aby stránka spĺňala všetky kritéria kladené na softvér používaný verejnou správou.
Do konca semestra plánujeme mať fungujúci projekt so všetkou funkcionalitou opísanou vyššie, s pokrytím všetkých identifikovaných chýb a prepracovaným dizajnom.

Celkový pohľad na systém

Moduly systému

Analýza

Systém pozostáva zo 4 hlavných komponentov, ako bolo opísané v kapitole Systémová architektúra.

Backend

Backend je implementovaný v Jave, jedná sa vlastne o Spring aplikáciu, ktorá poskytuje frontendu služby. Architektúra zodpovedá približne štýlu REST, teda poskytované služby sú REST-ové volania, vrámci ktorých si backend a frontend posielajú dáta uložené v JSON formáte cez HTTP protokol.

Frontend

Frontend je naprogramovaný v Angulari, obsahuje teda vytvorené komponenty obsahujúce príslušné TypeScript, HTML a SCSS súbory. Volá služby backendu, ktoré sú namapované cez REST-ové volania.

Elasticsearch

Fulltextový vyhľadávač Elasticsearch je použitý pre vyhľadávanie nad dokumentami reprezentujúcimi ontológie Centrálneho modelu. Beží ako samostatný proces, pristupuje sa k nemu cez klienta na backende.

PostgreSQL databáza

S Postgresovou databázou komunikuje backend a ukladá do nej údaje o dokumentoch.

Systémová architektúra

Samotný systém sa skladá z Frontendového a Backendového projektu, ako bolo opísané vyššie. Dodatočné externé služby využívané vrámci systému sú Elasticsearch a postgreSQL databáza. Prepojenie daných komponentov je opísané nižšie.

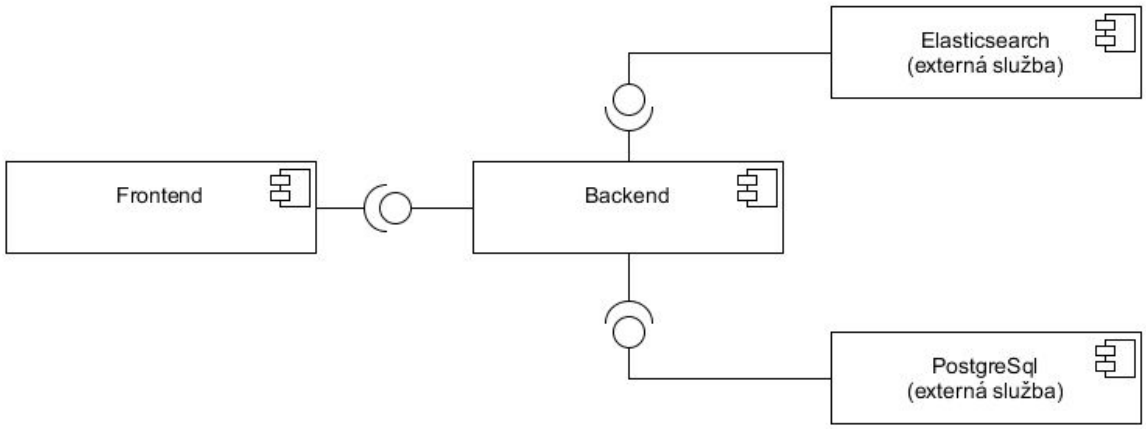
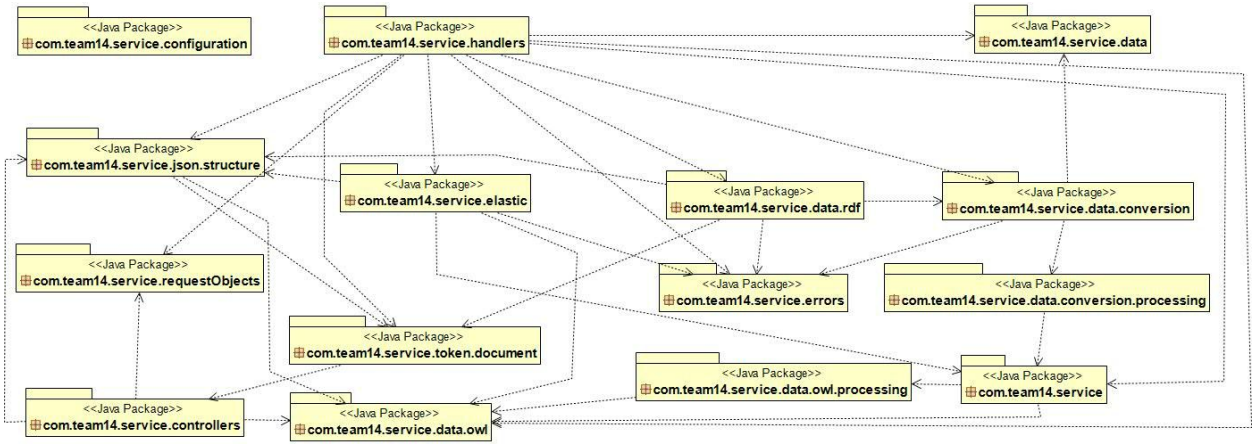
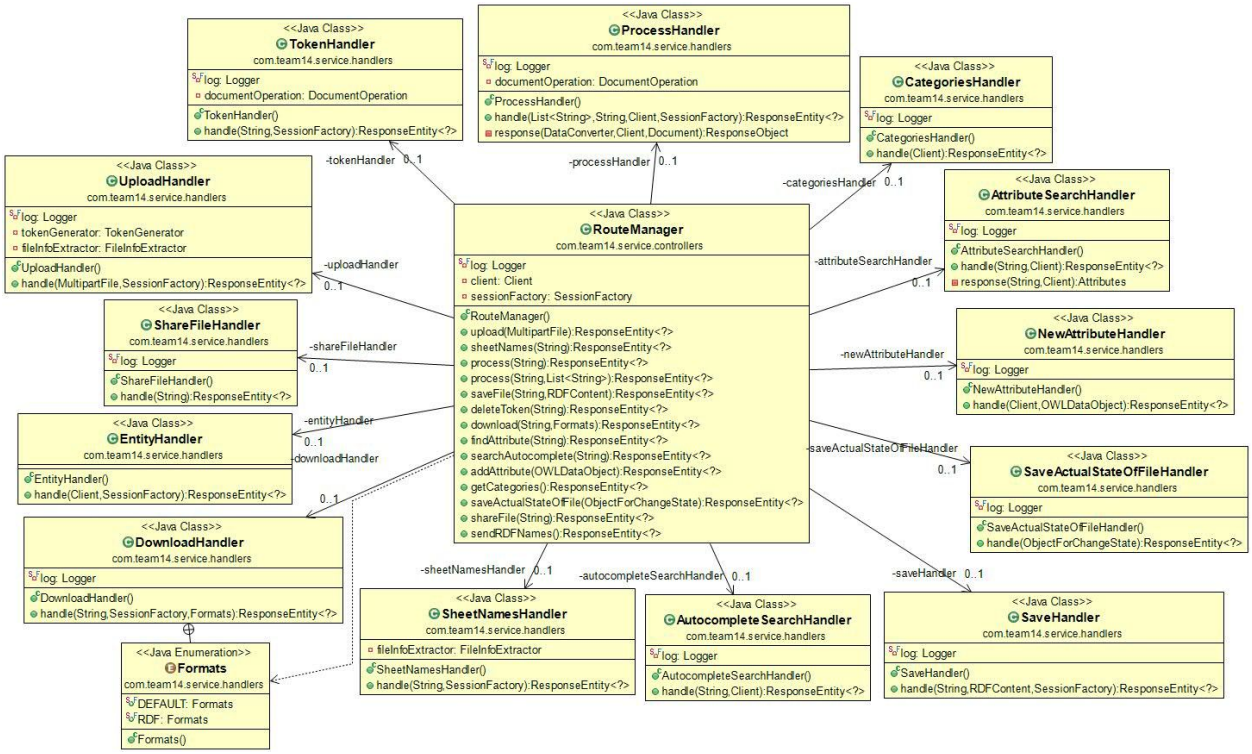


Diagram tried backendu

Nižšie uvedený diagram balíkov reprezentuje základné závislosti medzi balíkmi tried. Hlavnú rolu zohráva balík, kde sú uložené handlery.



Teraz sa pozrieme podrobnejšie na určité skupiny balíkov so silným previazaním a to najmä handlery, OWL objekt či objekty na spracovanie dát.



Tento model reprezentuje vzťah jednotlivých handlerov k RouteManager triede. Tu je vidno, že RouteManager ako controller využíva jednotlivé handlers ako services (v súlade so Springovou idiómou).

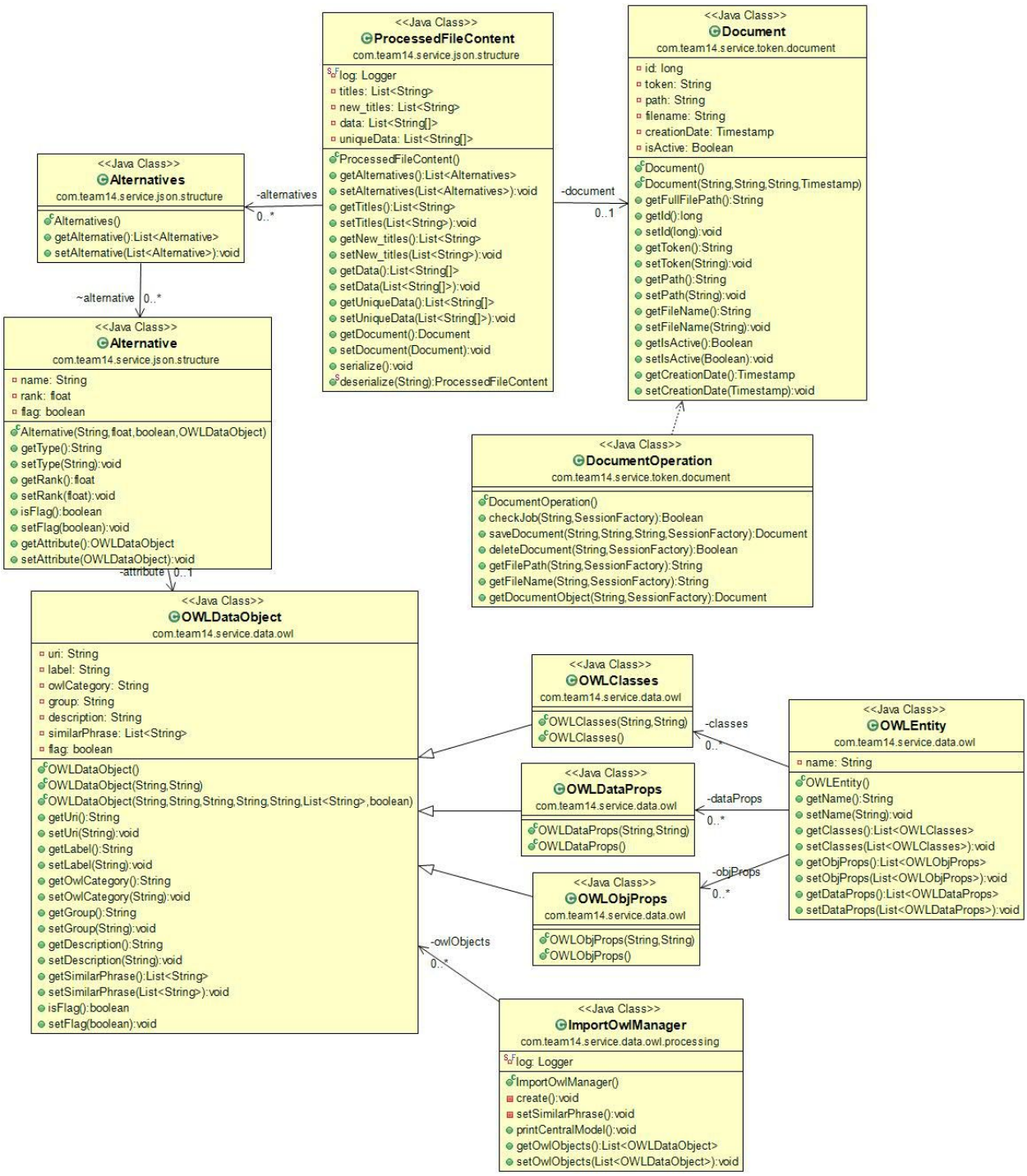
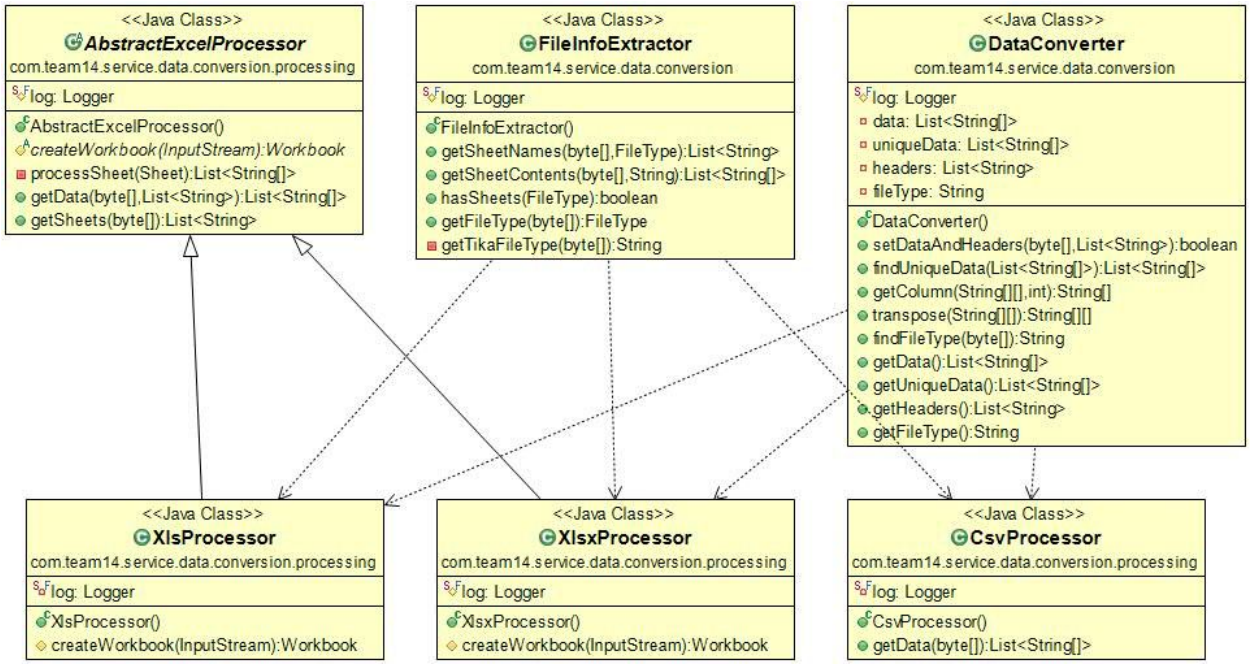
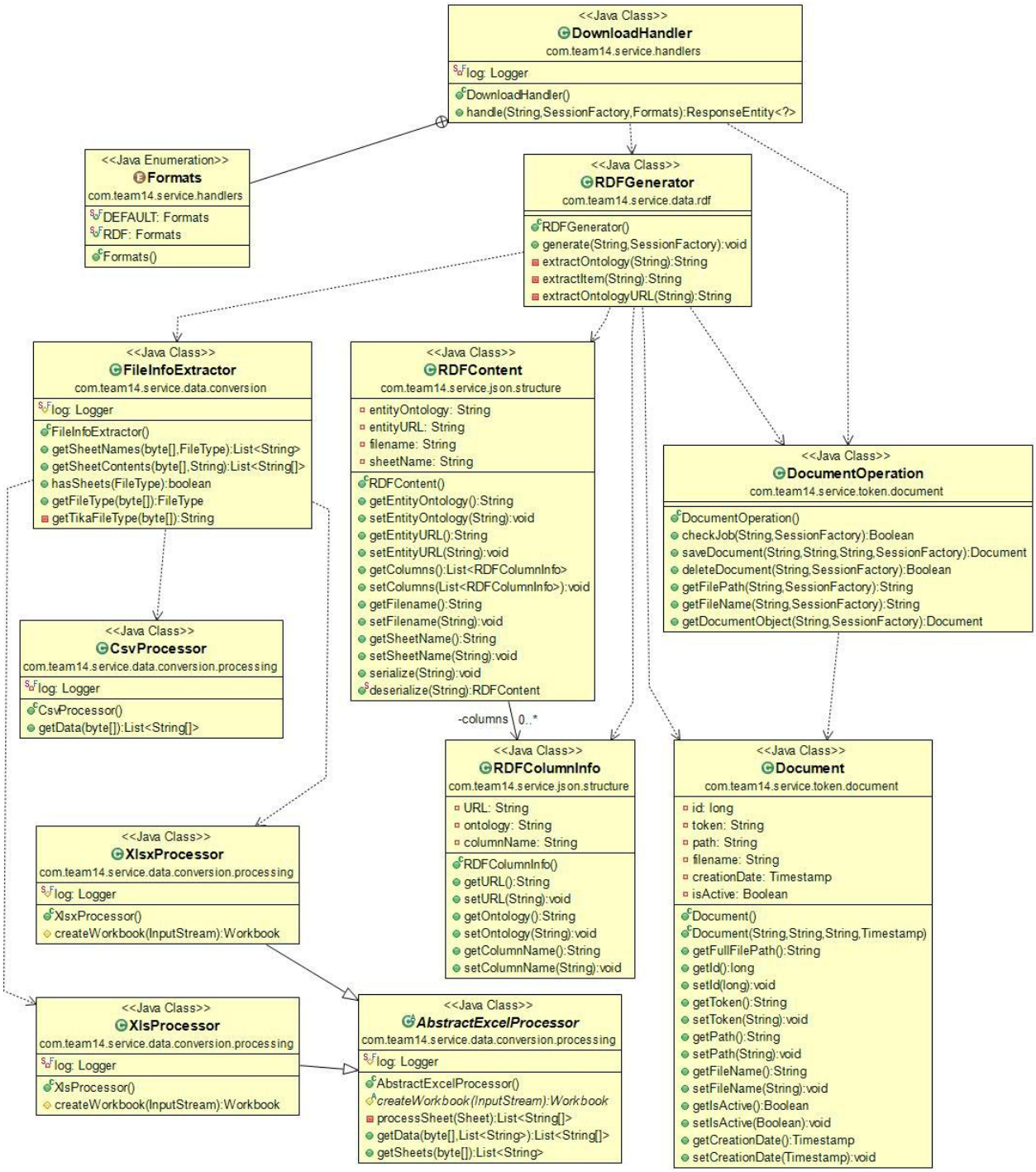


Diagram tried pre časť funkcionality zaoberajúcou sa ontológiami reprezentovanými cez OWL objekty. Zahŕňa json štruktúry Alternative a Attributes, ktoré sú posielané ako odpovede na frontend a tiež triedu Document, ktorá reprezentuje ukladaný dokument.



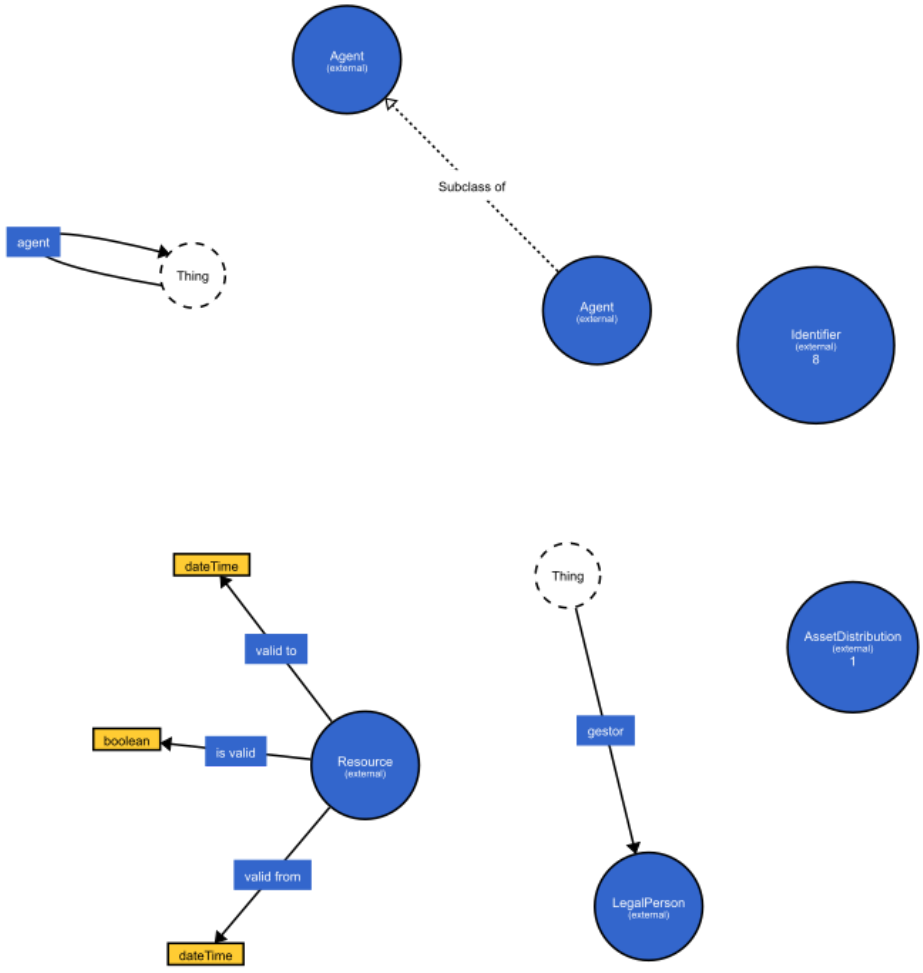
Uvedený diagram tried znázorňuje vzťahy medzi abstraktnou triedou excelovského procesora, jeho konkrétnymi implementáciami a konvertorom dát slúžiacim na spracovanie dát zo štandardných vstupných súborov.



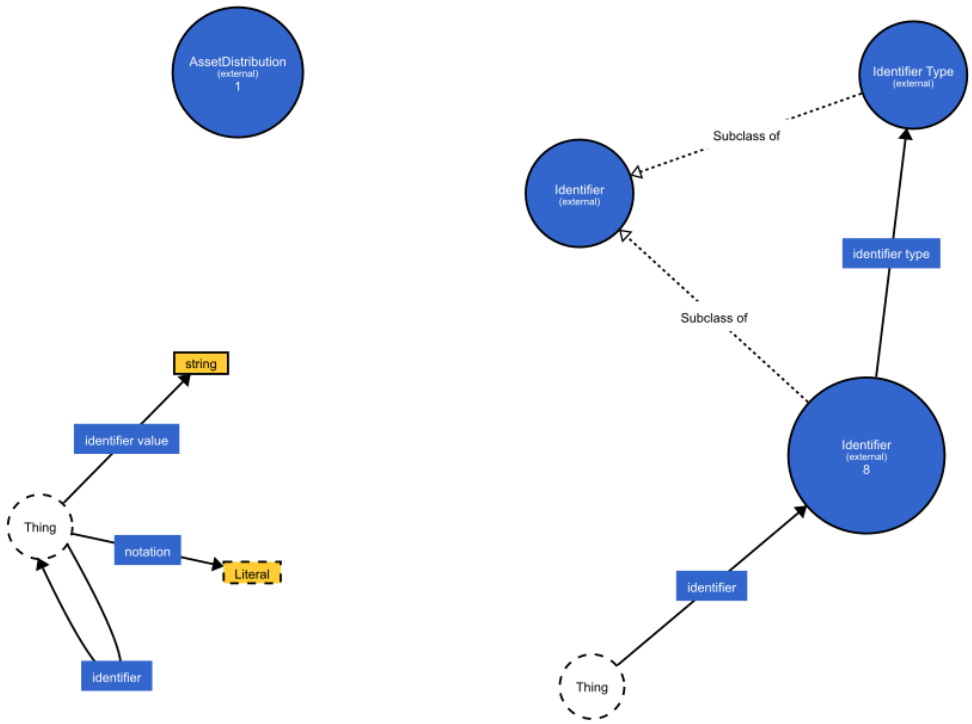
Uvedený diagram vyššie znázorňuje prácu vykonanú pri generácií RDF súboru stiahnutého cez Download handler.

Ontológie centrálneho modelu

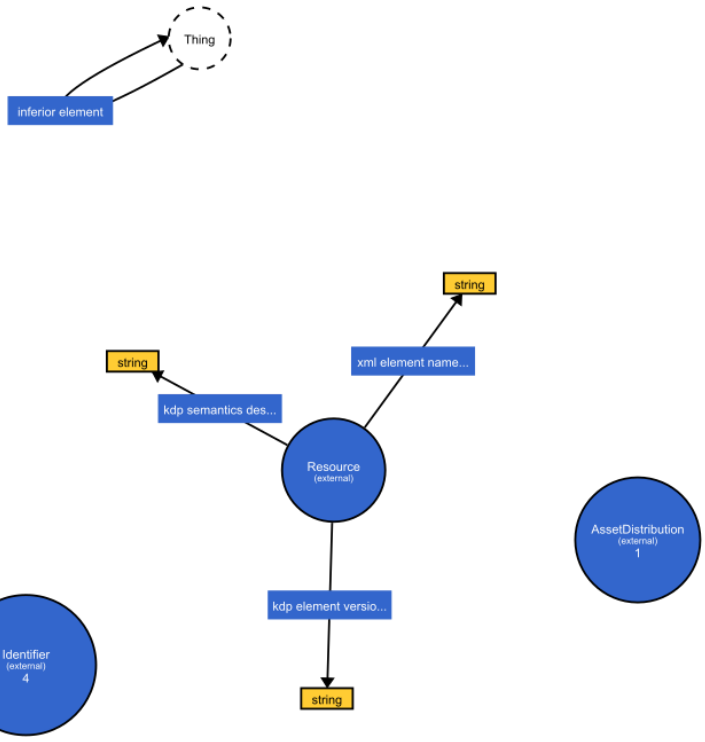
Nasleduje opis OWL objektov, ktoré znázorňujú jednotlivé ontológie definované priamo Úradom vlády. Tieto ontológie sú definované pomocou štandardu OWL. Sú dostupné aj online na stránkach data.gov.sk, vrámci nášho projektu ich načítavame zatiaľ z lokálne uloženého dátového modelu, tzv. Centrálneho modelu.



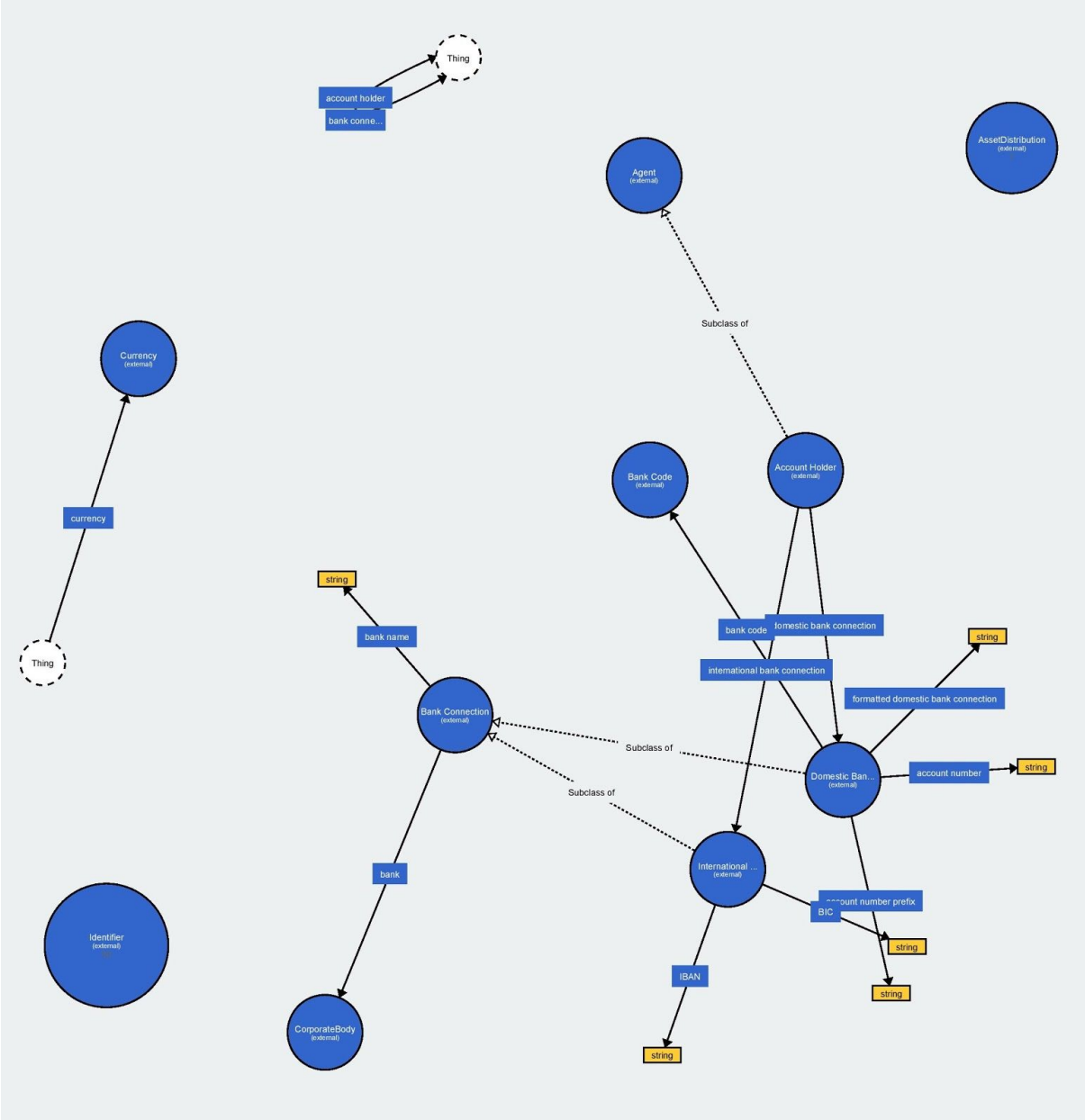
Ontológia zdrojov



Ontológia identifikátora



Ontológia katalógu dátových prvkov



Ontológia finančných entít

Ostatné ontológie kvôli rozmerom sú dostupné len v tímovom repozitári pre dokumentáciu: https://code.xit.camp/upvii/dataset-importer/docs/wikis/Pou%C5%BE%C3%ADvan%C3%A9-ontol%C3%B3gie

Kľúčovým komponentom importéra je tvorba RDF súborov.

RDF

RDF (Resource Description Framework) je rámec, na popis zdrojov na webe. Je napísaný v jazyku XML. Hlavnou myšlienkou, ktorá stojí za RDF, je snaha priblížiť web sémantickému webu. Na vyjadrenie jednotlivých vzťah sa používajú tzv. Triplety. Zápis týchto tripletov je vo forme podmet-prísudok-predmet (subject-predicate-object). Pri RDF určujeme resource, property a property value. Resource je v podstate všetko, čo môže mať URI. Property je resource, ktorý má meno a property value je hodnota property. Viac môže na povedať jednoduchý príklad nižšie.

```
<?xml version="1.0"?>

<RDF>
  <Description about="https://www.example.org/rdf">
    <author>Jan Egil Refsnes</author>
    <homepage>https://www.example.org</homepage>
  </Description>
</RDF>
```

Resource - https://www.example.org/rdf - podmet (subject)
Property - author, homepage - prísudok (predicate)
Property value - Jan Egil Refsnes, https://www.example.org - predmet (object)

RDF generátor

Úlohou RDF generátora, je vygenerovať RDF súbor z práve spracovaného datasetu. Tento generátor má svoj presne určený formát RDF súborov, ktoré vytvára.
Príklad takéhoto vygenerovaného súboru je možné vidieť nižšie. Tento súbor vznikne po vygenerovaní z nasledovnej tabuľky:

givenName	alternativeName	birthNumberCode	confessionType	idCardCode	physicalAddress
Peter	Zelený	961010/6058	ateizmus	EB47512	Ulica Mieru 13, Bratislava

```
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE rdf:RDF [
        <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
        <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
        <!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>
```

```
]>

<rdf:RDF
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns="http://example.org/data/ludia#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:location="https://data.gov.sk/def/ontology/location#">

        <rdf:Description rdf:about="http://example.org/data/...token.../physicalPerson-1"
rdf:type="http://example.org/data/ludia/physicalPerson">
                <givenName xml:lang="Sk">Peter</givenName>
                <alternativeName xml:lang="sk">Zelený</alternativeName>
                <birthNumberCode xml:lang="sk">961010/6058</birthNumberCode>
                <confessionType xml:lang="sk">ateizmus</confessionType>
                <idCardCode xml:lang="sk">EB47512</idCardCode>
                <location:physicalAddress xml:lang="sk">Ulica Mieru 13,
Bratislava</location:physicalAddress>
        </rdf:Description>

</rdf:RDF>
```

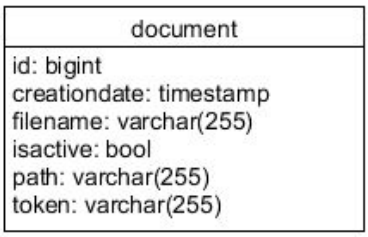
Ako je možné vidieť, je zadefinovaný aj nový namespace. Táto definícia vznikla na základe toho, že v danej tabuľke sa nachádzajú atribúty, ktoré sú z rôznych ontológií. Kým väčšina atribútov je z ontológie fyzickej osoby, fyzická adresa je z ontológie lokácií. Preto bolo potrebné túto ontológiu zadefinovať pomocou namespacu a potom ju korektne používať.

Dátový model

Dátový model v našom riešení je momentálne v stave v akom ho zanechal predchádzajúci tím a databáza nie je veľmi využívaná. Okrem iného v nej bolo plánované ukladať aj dáta o používateľoch čomu ale nebol zatiaľ prispôsobený dátový model. My plánujeme využívať túto databázu trocha viac.

Databáza

Ako bolo spomenuté, databáza sa zatiaľ využíva minimálne. Obsahuje iba jednu tabuľku dokumentu, ktorá obsahuje základné informácie o dokumente uloženom na serveri po jeho nahratí. Dátový model je zobrazený nižšie.



Možnosti databázy nie sú momentálne využité (len jedna tabuľka), máme však v pláne rozšíriť jej použitie a ukladať v nej dáta, ktoré sa v súčasnosti ukladajú v serializovaných súboroch.

ElasticSearch

Projekt využíva na ukladanie dát ElasticSearch, najmä z dôvodu, že je v týchto dátach potrebné sofistikované fulltextové vyhľadávanie. Využívame jeden Index, ktorý je potrebné pred prácou na projekte vytvoriť a naplniť údajmi. Na tento úkon využívame kombináciu Bash skriptu (vytvorenie indexu) a Java programu (na naplnenie údajmi).
Štruktúra tohto indexu je vyjadrená nasledovnou tabuľkou:

Meno	Typ	Popis
url	keyword	jednoznačný identifikátor ontológie vo forme URL
sk_name	text	viditeľný názov určený pre potreby používateľa
owl_category	text	kategória
owl_category_raw	keyword	kategória určená pre interné potreby vyhľadávania v indexe (agregačné dopyty)
description	text	čitateľný popis
similar_phrase	text	ponúkané alternatívy s podobným názvom
group	text	skupina, do ktorej atribút patrí

Návrh a implementácia backendu

Hlavnou riadiacou triedou backendu je RouteManager. Ten spravuje všetky volania tvorbou príslušných handlerov a volaním ich metód spracovania. Každý handler vracia ResponseEntity, ktorú dokáže Springový framework spracovať ako JSON, ktorý posiela ako odpoveď na frontend.
Handlery, ktoré sa nachádzajú v projekte, sú:

AttributeSearchHandler - tento zabezpečuje vyhľadávanie atribútov v inštancii ElasticSearch
AutocompleteSearchHandler.java - zabezpečuje dáta pre autocomplete na stránke s vyhľadávaním atribútov
CategoriesHandler - vracia zoznam kategórií atribútov
DownloadHandler - umožňuje stiahnuť spracované tabuľky v rôznych formátoch
EntityHandler - pomocou tohto handleru je možné vyhľadávať entity
NewAttributeHandler - umožňuje vytváranie nových atribútov
ProcessHandler - jadro backendu, spracováva úradníkmi nahrané súbory
SaveActualStateOfFileHandler - ukladá stav rozpracovanej tabuľky
SaveHandler - uloží nahrávaný súbor
ShareFileHandler - získava dáta ohľadom spracovávanej tabuľky pri zdieľaní súboru s iným úradníkom
SheetNamesHandler - dokáže vytiahnuť názvy hárkov z Excel súboru
TokenHandler - zabezpečuje vymazávanie dokumentov
UploadHandler - obsluhuje nahrávanie súboru na server

Backend prešiel v 2. šprinte rozsiahlym refaktorom, v ktorom sme sa pokúsili viac využiť Spring Boot najmä v dvoch ohľadoch:

- lepšie spracovanie chýb
- využívanie Spring Boot services

Tieto dva aspekty boli hlavnými nami identifikovanými nedostatkami kódu, ktoré sa nachádzali v predchádzajúcej implementácii backendu.

Tieto problémy sme adresovali vytvorením triedy RouteExceptionHandler, ktorá využíva Spring Boot na to, aby zachytila všetky vyhadzované Exceptiony a poskytla dostatok informácií o vzniknutej chybe (stack trace a pod.)

Okrem toho sme zo všetkých dovtedajších handlerov vytvorili Spring Boot services, ktoré využíva už spomínaný RouteManager. Tieto services sú do neho Spring Bootom injektované a ten ich následne môže používať.

Testovanie backendu

Na backende je napísaných niekoľko unit testov. Tie riešia predovšetkým konverziu nahrávaných súborov na surové dáta.

Návrh a implementácia frontendu

V tomto projekte je na frontende použitý framework Angular, ktorý diktuje základnú štruktúru. Vzhľadom na relatívne malú veľkosť aplikácie sa frontend skladá z jedného modulu, ktorý je rozdelený do viacerých komponentov, pričom každý z nich sa skladá z implementácie controlleru v TypeScripte, definovania štýlu v SCSS, layoutu v HTML a testov písaných taktiež v TypeScripte, využívajúc testovacie frameworky Jasmine a Karma.
Využívaná je pritom menná konvencia odporúčaná priamo autormi Angularu a podporovaná ich podporným programom zvaným ng. Komponent TestPage by sa využitím týchto konvencií rozdelil do súborov pomenovaných nasledovne:

test-page.component.html
test-page.component.scss
test-page.component.spec.ts
test-page.component.ts

Pri testoch je potrebné podotknúť, že sa jedná o súbory, ktoré sú automaticky generované Angularom a nie sú zatiaľ implementované bližšie. Tento a mnohé ďalšie nedostatky sme identifikovali v 3. šprinte a plánujeme ich napraviť v nadchádzajúcich šprintoch.

Okrem komponentov sa v projekte nachádzajú služby (Services), ktoré slúžia na zabezpečenie komunikácie s backendom. V prípade, že komponent potrebuje komunikovať s backendom, musí tak urobiť práve prostredníctvom služieb.

Modul, ktorý všetky stránky na frontende prepája, je AppRoutingModule, ktorý určuje, ktorý komponent sa zobrazí používateľovi po navštívení určitej adresy.

Nasleduje zoznam hlavných komponentov, teda komponentov, z ktorých každý reprezentuje jednu samostatnú stránku. Poradie, v akom tieto komponenty uvádzame, kopíruje poradie, v akom ich používateľ navštívi.

UploadComponent - stránka umožňujúca úradníkom nahrať súbor
ChooseSheetComponent - v prípade, že používateľ nahrá Excel súbor, musí si na tejto stránke vybrať hárky, ktoré z daného súboru chce spracovať
EditComponent - predstavuje stránku s úpravou úradníkom nahranej tabuľky
AttributeSearchComponent - na tejto stránke je možné vyhľadávať názvy stĺpcov v centrálnom modeli

AttributeCreateComponent - tu je možné vytvoriť nový stĺpec v prípade, že používateľ nenašiel žiadny vyhovujúci
ConfirmCreateAttr - stránka, na ktorej sa potvrdzuje vytvorenie nového atribútu
ConfirmTableComponent - na tejto stránke sa zobrazí sumarizácia tabuľky, ktorú úradník upravoval
ConfirmTableModalComponent - tu sa potvrdzujú zmeny vykonané pri úprave tabuľky
DownloadComponent - obsahuje odkazy, z ktorých sa dá stiahnuť spracovaná tabuľka v rôznych formátoch

Okrem týchto hlavných komponentov sú v projekte zdieľané komponenty, ktoré poskytujú generickú funkcionalitu a sú používané viacerými komponentami. Jedná sa o veci ako dropdown, tooltip, spinner pri načítavaní, atď.

Tieto sú ukryté v samostatnej zložke shared spolu s ďalším typom tried - modelom. Model slúži čisto na ukladanie dát posielaných pomocou HTTP požiadaviek medzi frontendom a backendom. Rozhrania modelu presne korešpondujú s modelovými triedami, ktoré sú definované na backende - teda majú rovnaký názov a obsahujú rovnaké členské premenné.

API

Primárne je API zadefinované v repozitári “*web*”. API je z veľkej časti prevzaté od minuloročného tímu a v budúcich šprintoch plánujeme urobiť jeho revíziu za účelom odstránenia zbytočností a celkového zlepšenia kvality.

/upload

Metóda: POST
Vstup:

- **uploadFile: string**
    - parameter formulára, obsahuje celý nahrávaný súbor

Výstup:
JSON Objekt:

- **token: string**
    - - identifikátor nahraného súboru
- **action: string**
    - enum vyjadrujúci akciu, ktorá sa má s nahrávaným súborom udiať
    - môže nadobúdať hodnoty SELECT_SHEETS a PROCESS:
        - SELECT_SHEETS vyjadruje, že súbor obsahuje viac hárkov, z ktorých si používateľ musí vybrať tie, ktoré chce spracovať
        - PROCESS vyjadruje, že súbor nemá viac hárkov a obsahuje tak len jednu tabuľku, ktorá sa môže hneď spracovať

Popis:
Upload sa zavolá pri nahratí súboru. Súbor sa vo výsledku na serveri uloží a naspäť sa vrátia informácie o ďalšom kroku.

/sheetNames

Metóda: GET
Vstup:

- **fileToken: string**
    - token identifikujúci predtým nahraný súbor

Výstup:
JSON Objekt:

- **sheets: string[]**
    - pole všetkých hárkov vytiahnutých zo súboru identifikovaného vstupným tokenom

Popis:
Sheet names sa volá pri vyberaní hárkov na spracovanie. Tie sú potrebné na zobrazenie možností na výber hárkov na frontende.

/processSingleDocument

Metóda: POST
Vstup:

- **fileToken: string**
    - token identifikujúci predtým nahraný súbor

Výstup:
JSON Objekt:

- **titles: string[]**
    - pôvodné názvy stĺpcov v nahranom súbore
- **new_titles: string[]**
    - názvy stĺpcov vybrané používateľom
    - toto pole je rovnako veľké ako pole titles
    - v tomto konkrétnom volaní sú všetky hodnoty **null** (pretože ešte používateľ žiadne stĺpce nevybral)
- **data: string[][]**
    - samotné dáta nahraného súboru - tabuľka
- **uniqueData: string[][]**
    - podobné ako data, s tým rozdielom, že sú vymazané duplicitné riadky
- **alternatives: Alternatives[]**
    - zoznam alternatívnych názvov pre každý stĺpec

Objekt Alternatives:

- **alternative: Alternative[]**

    - objekt Alternatives slúži ako obal pre alternatívy samotné vyjadrené objektom Alternative

Objekt Alternative:

- **name: string**
    - názov alternatívy
- **rank: float**
    - Atribút zdedený od predchádzajúceho tímu, pravdepodobne pravdepodobnosť toho, že alternatíva je správna
- **flag: boolean**
    - atribút zdedený od predchádzajúceho tímu, na jeho účel sme neprišli
- **attribute: OWLDataObject**
    - bližšie informácie o alternatíve

Objekt OWLDataObject:

- **URI: string**
    - URI identifikátor objektu
- **label: string**
    - názov objektu
- **owlCategory: string**
    - kategória, do ktorej je objekt zaradený
- **group: string**
    - skupina, do ktorej objekt patrí
- **description: string**
    - bližší popis objektu spresňujúci jeho význam
- **similarPhrase: string[]**
    - podobné výrazy
- **flag: boolean**
    - atribút zdedený od predchádzajúceho tímu, na jeho účel sme neprišli

Popis:
Pri processSingleDocument sa udeje získavanie údajov z nahraného súboru. Súbor vo formáte nepodporujúcom viac hárkov je tu prekonvertovaný na surové dáta, ktoré sa posielajú frontendu na ďalšie spracovanie.

/processMultipleSheets

Metóda: POST
Vstup:

- **fileToken: string**
    - token identifikujúci predtým nahraný súbor

Výstup: rovnaký, ako pri /processSingleDocument
Popis:

Analogicky k processSingleDocument, deje sa tu získavanie údajov z nahraného súboru. Rozdiel oproti processSingleDocument je v tom, že tu sú spracovávané súbory, ktoré majú podporu viacerých hárkov.

/saveFile

Metóda: POST
Vstup:

- **token: string**
    - token identifikujúci predtým nahraný súbor

Výstup: *true* alebo *chyba*
Popis:
Volanie *saveFile* slúži na finálne uloženie údajov, ktoré si zvolil používateľ pri spracovaní súboru.

/download

Metóda: GET
Vstup:

- **token: string**
    - token identifikujúci predtým nahraný súbor
- **format: string**
    - enum vyjadrujúci, v akom formáte chce používateľ daný súbor stiahnuť
    - možné hodnoty:
        - **DEFAULT** - súbor sa odošle vo formáte, v akom bol do systému nahraný
        - **RDF** - súbor je odoslaný vo formáte RDF

Výstup: súbor v požadovanom formáte
Popis:
Download je volaný po skončení úpravy nahraného súboru - frontend používateľovi poskytuje možnosť stiahnuť si svoju prácu.

/findAttributes

Metóda: GET
Vstup:

- **attribute: string**
    - výraz, ktorý bude vyhľadaný v internej databázi atribútov

Výstup:
Json Objekt: **Attributes**

- **attributes: OWLDataObject[]**
    - Štruktúra objektu OWLDataObject popísaná vyššie

Popis:

FindAttributes je volaná pri vyhľadávaní možných názvov stĺpca. Je to v podstate rozšírená verzia *searchAutocomplete*, ktorá vráti menej informácií za účelom šetrenia sieťovej komunikácie.

/searchAutocomplete

Metóda: GET
Vstup:

- **phrase: String**
    - časť slova na doplnenie

Výstup:
JSON Objekt: **AutocompleteContent**

- **autocompleteHits: string[]**
    - obsahuje zoznam slov vyhodnotených nástrojom Elasticsearch

Popis:

- Používa sa pri vstupných poliach s automatickým dopĺňaním. Frontend odošle aktuálny obsah vstupného pola (reťazec X) a backend vráti slová, ktorých časťou je reťazec X.

/addAttribute

Metóda - POST
Vstup:

- **attribute: OWLDataObject**
    - popísaný vyššie

Výstup

- true, prípadne chyba

Popis
AddAttribute je volaný pri vytváraní nového atribútu z frontendu. Používať by sa mal len v prípade, že používateľ nenájde hľadaný atribút v množine existujúcich.

/getCategories

Metóda: GET
Vstup: žiaden
Výstup:
JSON Objekt: Category Content

- **categories: string[]**
    - zoznam kategórií

Popis
Slúži na získanie zoznamu kategórií, z ktorých je možné vyberať pri vytvorení názvu nového stĺpca.

/saveActualStateOfFile

Metóda: POST
Vstup:

- changeState: ObjectForChangeState (v body)
    - **changedColumn: String**
        - zmenený názov stĺpca
    - **token: String**
        - identifikátor spracovávaného súboru
    - **columnNumber: int**
        - index upravovaného stĺpca

Výstup:

- Pravdivostnú hodnotu (true/false), prípadne chybu

Popis:
Toto volanie slúži na uloženie aktuálneho stavu spracovávaného súboru na serveri.

/shareFile

Metóda: POST
Vstup:

- token: string
    - reprezentuje identifikátor spracovávaného súboru

Výstup:
JSON objekt: **ProcessedFileContent**

- **titles: string[]**
- **new_titles: string[]**
- **data: string[][]**
- **uniqueData: string[][]**
- **alternatives: Alternative[]**
- **document: Document**
- Popisy sú uvedené vyššie

Popis:
ShareFile je volané zo stránky pre úpravu nahranej tabuľky. Obsahuje aktuálne údaje rozpracovanej tabuľky, ktoré sa priebežne ukladajú volaním *saveActualStateOfFile* popísaného vyššie.

/getEntites

Metóda: GET
Vstup: žiaden
Výstup:

Pole:

- **object: Entity**
    - **uri: string**
        - uri adresa používaná na identifikáciu entity pri generovaní RDF
    - **label: string**
        - názov entity

Popis:
Slúži na získanie zoznamu entít zobrazených na edit komponente. Z týchto entít si používateľ vyberie entitu, ktorú reprezentuje riadok tabuľky v nahranom súbore. Vybraná entita je zasielaná na backend a používaná pri generovaní RDF súboru-

Testovanie frontendu

Aktuálne sme v procese písania unit a e2e testov na frontende. Na tento účel sú využívané frameworky Karma (na unit testy) a Protractor (na e2e testy). Oba druhy testov využívajú Jasmine na ich popis. Protractor v pozadí využíva Seleniové jadro.

Unit testami máme pokryté vytvorenie všetkých komponentov a služieb (services). Plánujeme pokryť aj ich funkcionalitu.

End-to-end testami máme aktuálne pokrytý upload súboru a korektné presmerovanie na ďalšie stránky, vrátane ošetrenia chýb v prípade nesprávneho formátu načítaného súboru.

Príručky

Príručky na nasledujúcich stranách sú exportované z tímovej wiki a slúžia na rozbehanie technológií, nasadzovaniu dockeru a správny štýl programovania.

Technická dokumentácia

Hlavná časť technickej dokumentácie pozostáva zatiaľ z generovaných javadocov pre backend. Sú uložené na tímovom webe.

Linky na repozitáre, tímovú dokumentáciu, tímovú stránku:

Backend: https://code.xit.camp/upvii/dataset-importer/importer-services
Frontend: https://code.xit.camp/upvii/dataset-importer/importer-web
Wiki a metodiky: https://code.xit.camp/upvii/dataset-importer/docs
Javadocs:
http://labss2.fiit.stuba.sk/TeamProject/2018/team16iss-it/javadoc/overview-summary.html
Tímový web: http://labss2.fiit.stuba.sk/TeamProject/2018/team16iss-it/index.html