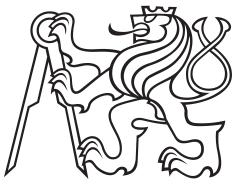


České
vysoké
učení technické
v Praze

# Sémantické facetové vyhledávání na platformě React

**Filip Sváček**

**Školitel: Ing. Martin Ledvinka**
**Leden 2020**

# Poděkování

Rád bych poděkoval vedoucímu mé práce, Ing. Martinu Ledvinkovi, za ochotu, rady, konzultace a nasazení, navzdory omezením a problémům, které i tento semestr přetrvávají.


# Prohlášení

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, ledna 5, 2020

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 5. January 2020

# Abstrakt

Účelem této bakalářské práce je návrh a implementace sémantického facetového vyhledávače, který umožní uživatelům hledat data za pomocí faset na sémantickém webu.

Na vyzkoušení fungování této implementace se vytvořilo demo spisovatelů, nad kterými uživatel vyhledává pomocí fasetů.

Přo řešení byl použit programovací jazyk Javascript a framework React na tvorbu uživatelského rozhraní a NodeJs na tvorbu modulu logiky.



**Klíčová slova:** Sémantický web, React, Fasety, NodeJS, SPARQL

**Školitel:** Ing. Martin Ledvinka



# Abstract

The purpose of this thesis is the design and implementation of semantic facet search, which allows users to query data using facets on the semantic web.

For the purpose of trying out the implementation, I created a demo of Writers, where a user searches data using facets.

For this project, I used the programming language Javascript and the framework React to build the user interface and NodeJs to build logic module.



**Keywords:** Semantic web, React, Fasets, NodeJS, SPARQL

**Title translation:** Semantic faceted search on the platform React

# Obsah

# Obrázky



# Tabulky

# Kapitola 1
# Úvod

Fasetové vyhledávání je věc, se kterou se většina lidí nejspíše setkala v každodenním životě, i když pouze menšina je si toho s největší pravděpodobností vědomá. Od e-shopů, popisů produktů, hledání pracovních pozic nebo kolekcích článků online, fasetové vyhledávání má značné využití ve všech oblastech internetu. Naproti otomu se obecně lidé nesetkali přímo tváří v tvář se sémantickým webem, který hraje roli spíše na pozadí.

Tato práce se zabývá kombinací problematiky sémantického webu a fasetového vyhledávání. Provede se analýza problematiky, implementace nástroje pro řešení problematiky. Důraz je kladen převážně na modulárnost a budoucí rozšiřitelnost. V rámci této aplikace je taky implementováno demo, na kterém se prověří použitelnost vytvořených nástrojů.

## 1.1 Motivace

Aplikacích zabývajících se problematikou kombinace fasetového vyhledávání a sémantického webu není mnoho a existující řešení mají silné neduhy. Tyto neduhy, mezi něž patří nedostačující dokumentace, těžce čitelný kód a neexistující modulárnost znemožňují seznámení se a práci s nimi. To vše vyústilo k tomu, že se navrhlo nové řešení, které by mělo mít podobnou funkcionalitu, zato však lepší implementaci.

## 1.2 Cíle

Sémantický web a s ním přidružené technologie jsou čím dám relevantnější a více potřebné s šířením webu a nutností procházet obrovským množstvím dat, které jsou na něm. Sémantické technologie se přesouvají z akademické pozice do pozice, kde jej velké korporáty adaptují pro komerční využití. Přesto v dnešní době chybí, nebo je ve špatném stavu spousta nástrojů, které jsou nutné pro použití těchto technologií. Cílem této práce je z malé části podílet na opravení této situace.

Aplikací zabývajících se problematikou kombinace fasetového vyhledávání a sémantického webu není mnoho, a existující řešení mají silné neduhy. Tyto neduhy, mezi něž patří nedostačující dokumentace, těžce čitelný kód a neexis-

tující modulárnost znemožňují seznámení se a práci s nimi. To vše vyústilo k tomu, že se navrhlo nové řešení, které by mělo mít podobnou funkcionalitu, zato však lepší implementaci.

# Kapitola 2
# Používané technologie

## 2.1 HTML

HTML je základní stavební blok webu. Definuje význam a strukturu webového obsahu. Tento projekt využívá aktuálně nejnovější verzi HTML 5.[1]

## 2.2 CSS

CSS je stylovací jazyk určený pro popis prezentace dokumentu napsaném v značkovacím jazyce, typicky HTML. Původně bylo vytvořeno, aby odstranilo neduhy počátečních verzí HTML. Společně s HTML a Javascriptem tvoří jednu z klíčových technologií webu. CSS nám umožňuje oddělit vzhled od obsahu, včetně layoutu, barev a fontů.[2]

## 2.3 Facety

Fasety jsou softwarové komponenty, které implementují jednu funkcionalitu, mají jeden veřejně volatelný interface a žádný zbytkový stav. V softwarovém inženýrství slouží jako prověřený nástroj pro průzkum informačního prostoru. Každou fasetu tvoří množina položek, nebo také nazývané fasetové hodnoty.[15]

### 2.3.1 Facetové vyhledávání

Facetové vyhledávání je dynamické shlukování položek nebo vyhledaných výsledků do kategorií, které umožní uživateli získat vyhledané výsledky jakoukoliv hodnotou v jakémkoliv položce. Každý facet zobrazuje počet vyhledaných výsledků ve vyhledání, které odpovídají dané kategorii.[15]

## 2.4 Javascript

Javascript je skriptovací jazyk, který se nejčastěji používá pro vývoj webových technologií, který běží na klientské straně. Jedná se o jednu z klíčových


**Obrázek 2.1:** Názorná ukázka facetového využití v e-shopech, kde máme facety, jejich výběrové hodnoty a počet vyhledatelných výsledků pro každou hodnotu.

technologií webu a tvoří nezbytnou část webových aplikací. [3] Pro tuto práci byly použity primárně dvě Javascriptové technologie: React pro modul vizualizace, a Node.js pro modul logiky.

### 2.4.1 Bootstrap

jedná se o nejpopulárnější framework pro HTML, CSS a Javascript pro vývoj webových stránek, s velkým důrazem na mobilní vývoj. Má v sobě CSS návrhové šablony určené pro typografii, formuláře, tlačítka, navigaci a další komponenty rozhraní. [13] V projektu se používá na stylování verze Bootstrap 4.

### 2.4.2 Mocha

Jedná se o testovací framework v Javascript běžící v prostředí Node.js a ve webovém prohlížeči, určený pro asynchronní testování kódu.[9] V tomto projektu byl tento framework použit pro Unit a Integrační testování modulu logiky.

### 2.4.3 JSX

Jde o syntaktické rozšíření Javascriptu, které se používá typicky v kombinaci s Reactem. Není povinné používat, ale představuje dobrou vizuální pomůcku při tvorbě uživatelského rozhraní v JavaScript kódu.[10]

Následuje ukázka JSX formátu, kde JSX formát se uloží do konstanty nazvané element a ta se poté vykreslí pomocí funkce render.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

```
const element = <Welcome name="Helen" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

### 2.4.4 Sparql.js

Sparql.js je parser, který umí převádět JSON objekty do jazyka SPARQL a ze SPARQL do JSON objektu. Byl vydán relativně nedávno a původně se měl použít parser, který byl naimplementován v semestrální práci, na kterou tato práce navazuje. Tato knihovna má ovšem mnohem bohatější funkcionalitu, existující veřejnou dokumentaci a je důkladně otestována, proto bylo rozhodnuto použít jí namísto řešení, které bylo vytvořené v semestrální práci.[11]

### 2.4.5 Node.js

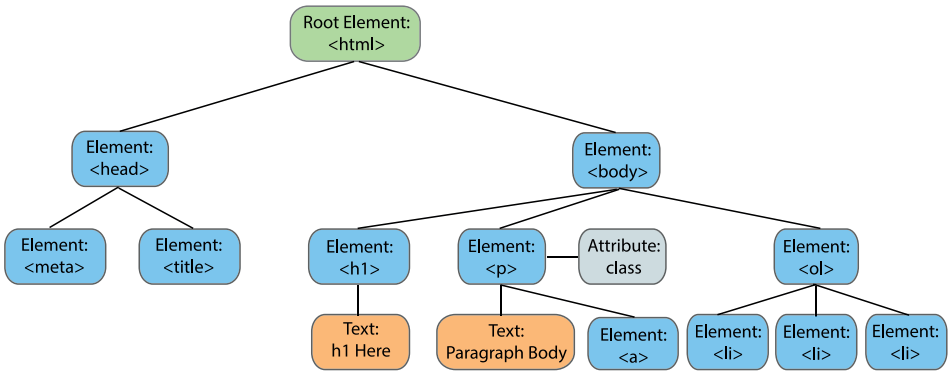
NodeJs je open source runtime prostředí, které umožňuje vývojářům vykonávat kód mimo webový prohlížeč. NodeJS běží asynchronně v jednom vlákně, a dává velký důraz na škálovatelnost a výkonost. [20]

### 2.4.6 NPM

Npm je správce balíčků pro programovací jazyk Javascript. V prostředí Node.js se jedná o defaultní správce balíčků. [12] Všechny balíčky jsou definovány v souboru package.json, z nichž mezi povinné patří název a verze aplikace, ostatní balíčky jsou přidávány podle potřeby v aplikaci.

Následuje ukázka části balíčku npm, která je použita v aplikaci.

```
{
 "name": "bcproject",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
    "dependencies": {
    "@testing-library/jest-dom": "^4.2.4",
    "@testing-library/react": "^9.5.0",
    "@testing-library/user-event": "^7.2.1",
    "bootstrap": "^4.5.2",
    "node-fetch": "^2.6.1",
```




**Obrázek 2.2:** Ukázka struktury DOM webové stránky

```
    "prop-types": "^15.7.2",
    "react": "^16.13.1",
    "react-bootstrap": "^1.3.0",
    "react-dom": "^16.13.1",
    "react-scripts": "3.4.3",
    "sparqljs": "^3.1.2",
    "xmlhttprequest": "^1.8.0"
  },
}
```

## 2.5 Git

Git je distribuovaný systém pro správu verzí, originálně navrhnutý pro koordinaci programátorů spolupracujících na vývoji softwaru. V dnešní době jej používá 40 milionů vývojářů a jde o klíčový nástroj pro vývoj projektů. Je jednoduchý na naučení a nabízí kvalitní kompresy dat a vysoký výkon. V této práci byl použit na pravidelné ukládání projektu a revizi již vytvořeného kódu.[14]

## 2.6 DOM

DOM je platforma a jazykově neutrální interface, který umožňuje programům a skriptům dynamicky přistupovat a aktualizovat obsah, strukturu a styl dokumentu.[16]

V této práci se s DOM manipuluje pomocí Javascriptu pomocí přidávání, odebírání a upravování elementů.

## 2.7 ReactJS

Jedná se o open source knihovnu napsanou v programovacím jazyce Javascript. Vytvořila jí a udržuje ji společnost Facebook společně s komunitou individuál-

ních přispěvatelů. Používá se na tvorbu uživatelských rozhraní na webu. Tato knihovna je velmi populární, dobře zdokumentovaná a její využití na webu neustále roste, což přispělo k jejímu použití v tomto projektu.

### 2.7.1 Props

Jsou argumenty, které se předávají do komponent Reactu. Jsou předávány komponentám pomocí HTML atributů.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

### 2.7.2 Komponenty

Na konceptuální úrovni jsou komponenty něco jako funkce Javascriptu. Slouží pro rozdělení kódu na dílčí části, což zlepšuje přehlednost a znovupoužitelnost. Přijímají vstupy, které se v Reactu nazývají props a vrací React elementy, které popisují, co se má zobrazit na obrazovce.

Následuje ukázka vytvoření a použití komponenty ShoppingList, která vypisuje položky nákupu plus předané parametry ve formě props.

```
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
  }
}
```

#### Prezenční komponenty

Prezenční komponenty jsou funkce, které nespravují žádný stav, s výjimkou stavu, který souvisí s prezentací. Nemají v sobě žádné vnitřní metody a

obecně se starají o generování HTML kódu. V této práci používáme výhradně kontejnerové komponenty pro lepší organizaci struktury projektu.

#### Kontejnerové komponenty

Kontejnerové komponenty jsou třídy, které jsou více komplexní, mají v sobě vlastní metody a mají vnitřní stav, který si udržují.

### 2.7.3 Stav

V Reactu stav představuje všechen dynamický obsah. Chytré komponenty mají v sobě metodu setState, která se používá na nastavení a změnu stavu. Po změně stavu touto metodou se aktualizuje zobrazený změněný obsah na stránce.[17]

Následuje funkce, která vypíše značku auta, který je ve stavu komponenty.

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Tesla",
    };
  }
  render() {
    return (
      <div>
        <h1>My Car is {this.state.brand}</h1>
      </div>
    );
  }
}
```

### 2.7.4 Render

Funkce, která je zodpovědná za renderování HTML ve webové aplikaci. Bere dva parametry, jedním z nich je HTML kód a druhým je HTML element. Protože používáme techniku „render props“ v naší aplikaci, což znamená že se předává kód z jedné komponenty do druhé, tak specifikujeme HTML element pouze v index.js a jinde máme pouze html kód.[18]

Následuje ukázka funkce tick(), která renderuje aktuální čas.

```
function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
```

```
  ReactDOM.render(element, document.getElementById('root'));
}
setInterval(tick, 1000);
```

### 2.7.5 Lifecycle metody

Jsou metody, které se používají v chytrých komponentách. Každá komponenta má životní cyklus, který se dá manipulovat v průběhu tří hlavních fází: Montáž, aktualizace a odpojení. Montáž je životní fáze, kdy se elementy přidávají do DOMu a odpojení je životní fáze. Aktualizace je fáze, kdy se komponenta aktualizuje. Odpojení je fáze, kdy se komponenta odstraňuje z DOMu. V aplikaci se používá metoda componentDiDMount pro načítání dat z endpointu. [17]

Následuje ukázka využití componentDidMount pro načtení dat z endpointu. Po montáži se načtou uživatelé z endpointu, změní se stav aplikace a zobrazí se uživatelské statusy.

```
class FriendStatus extends React.Component {
  constructor(props) {
    super(props);
    this.state = { isOnline: null };
    this.handleStatusChange =

    this.handleStatusChange.bind(this);
  }

  componentDidMount() {
    ChatAPI.subscribeToFriendStatus(
      this.props.friend.id,
      this.handleStatusChange
    );
  }
    render() {
    if (this.state.isOnline === null) {
      return 'Loading...';
    }
    return this.state.isOnline ? 'Online' : 'Offline';
  }
}
}
```

### 2.7.6 Události

Událost je akce, která nastane jakožto výsledek uživatelské akce, jako je například klikání na myš, stisknutí klávesy, nebo vytvoření či úprava souboru. [20]

## 2.8 Sémantický web

Sémantický web je rozšíření současného webu, ve kterém je informacím dán jasně definovaný význam, umožňující strojům a lidem lépe spolupracovat.[4] Hlavním cílem sémantického webu je tedy zaručit, aby Internetová data byla lépe strojově čitelná.

### 2.8.1 SPARQL

SPARQL je sémantický dotazovací jazyk určený pro manipulaci a čtení dat formátu RDF v grafových databázích. [12] SPARQL umožňuje uživateli vykonat dotaz nad databází či jakýmkoliv datových zdroji, který lze mapovat na RDF. SPARQL nedělá nic jiného, než bere popis toho, co aplikace chce ve formě dotazu a vrátí tuto informaci ve formě množiny vazeb či RDF grafu. [7]

### 2.8.2 Dotaz

Dotaz v kontextu sémantického webu znamená sada technologií a protokolů, které mohou programovatelně vrátit data z webu. Tento koncept se používá v řadě databázových jazyků jako je například SQL, ale v SPARQL jsou založeny na trojitých vzorech (triple patterns).[8] Používáním SPARQL je možné z webu získat komplexní informace, které mohou být vráceny v tabulkovém formátu.

Následuje ukázka SPARQL dotazu, který vybírá všechny objekty a subjekty napojené na <http://xmlns.com/foaf/0.1/name> predikát.

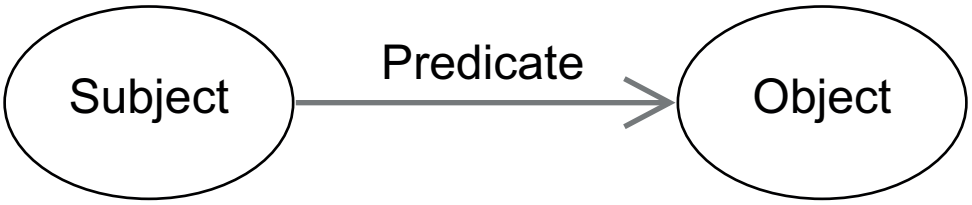
```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
    ?person foaf:name ?name .
}
```

### 2.8.3 Linková data

Linková data je metoda pro publikování strukturovaných dat za pomocí použití slovníků, které se dají společně spojit a interpretovat stroji. [5] Typický příklad linkovaného datasetu je DBpedia, která má v podstatě v sobě obsah wikipedie ve formátu RDF.

### 2.8.4 Slovník

Slovník definuje koncepty a vztahy používané k popisování a reprezentování dat v dané oblasti zájmu. [6] Cíl slovníků v sémantickém webu je pomoct zamezit ambivalenci v různých množinách dat. Mezi další cíle patří možnost nalezení nových vztahů mezi různými datovými typy a organizace znalostí.




**Obrázek 2.3:** Názorná ukázka sémantické trojice

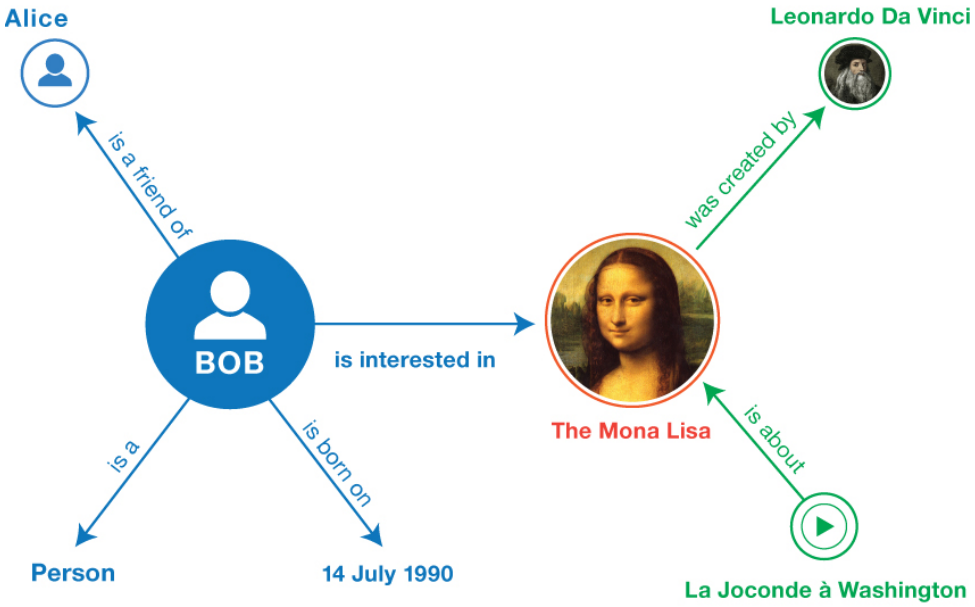
### 2.8.5 Sémantická trojice

Je atomická datová entita v RDF datovém modelu. Jde o množinu tří entit, které se nazývají Subjekt, Predikát a Objekt. Subjekt a Objekt představují vrcholy, Predikát je orientovaná hrana ukazující z Subjektu na Objekt. Společně vytváří výrok o sémantických datech. Tento formát se používá, protože je dobře strojově čitelný. Každá část tohoto výroku se dá adresovat unikátním identifikátorem, díky čemuž se dá nad sémantickými daty jasně dotazovat a operovat.[4]

## 2.9 RDF

RDF je framework určený pro reprezentaci informací na webu a standard pro výměnu dat na webu. Tvoří spojovací strukturu webu a používá URI na pojmenování vztahů mezi věcmi a zároveň mezi konci odkazu. Tato slouží pro snadné a přenositelné struktury, určené pro flexibilní aplikování napříč různými rozdílnými aplikacemi. Tato linkovací struktura vytváří graf, kterému lidé dokážou snadno rozumět.[7]

Používá se jako obecná metoda pro popis a modelování objektů či informací. Příklady aplikace mohou být elektronické knihovny, popisování obsahu pro vyhledávače, či popisování vlastností pro nákupní položky jakými můžou být cena a dostupnost.




**Obrázek 2.4:** Názorná ukázka RDF grafu

# Kapitola 3

# Návrh a analýza řešení

## 3.1 Srovnání s existujícím řešením

Tato práce vznikla převážně proto, že existující řešení je nedostačující pro účely této univerzity. Je tedy nutné porovnat obě práce a analyzovat nedostatky existujícího řešení a dále ukázat, jak tato práce opravuje tyto nedostatky.

### 3.1.1 Návrh aplikace

Existující řešení neseparuje logiku a vizuální výstup. To způsobuje, že kód je těžce čitelný pro kohokoliv, kdo se nepodílel na tvorbě aplikace, a těžko se udržuje. V neposlední řadě se tento kód těžko testuje. Řešení této práce odděluje logiku od vizuálního výstupu, a zbavuje se všech těchto neblahých vlastností.

### 3.1.2 Typy facetů v konfiguraci

Existující řešení podporuje více druhů facetů, jakými jsou například datový facet, hierarchický facet, textový facet a různé variace základních facetů. Tato funkcionalita nebyla v rámci této práce požadována, s výjimkou výběrových facetů, tím pádem nebyla implementována. Vytvořená aplikace je však stavěná na budoucí rozšiřování, takže není problém kdykoliv chybějící funkcionalitu doplnit.

### 3.1.3 Řešení omezení v konfiguraci

Existují řešení místo objektů pro omezení používá napevno daný textový řetězec v tomto formátu.

```
var constraint = '?id <http://dbpedia.org/ontology/genre>
<http://dbpedia.org/resource/Science_fiction> .';
```

Toto řešení je nevhodné, protože se text řetězce hůře upravuje a spoléhá na to, že formát pro vytvoření dotazu aplikace zůstane nezměněný. V této práci se tedy použil přístup podobný facetům, kde se nadefinují JSON objekty a ty poté zpracuje převáděč JSON objektu na SPARQL.

Následuje ukázka objektu omezení, který je použitý v demu spisovatelů.

```
let constrains = {
    writer: {
        constraintId: 'writer',
        predicate:
        'http://www.w3.org/1999s/02/22-rdf-syntax-ns#type',
        object: 'http://dbpedia.org/ontology/Writer',
    }
}
```

### 3.1.4 Řešení facetů v konfiguraci

Existující řešení je podobné, akorát rozlišuje typy (například výběrových a textových facetů) pomocí představení proměnné boolean chart a případě dalších facetů, které toto řešení rozlišuje, se facety odlišují pomocí představení dalších proměnných.

```
    // Text search facet for names
    name: {
        facetId: 'name',
        predicate:
        '<http://www.w3.org/2000/01/rdf-schema#label>',
        enabled: true,
        name: 'Name'
    },
    // Basic facets
    genre: {
        facetId: 'genre',
        predicate: '<http://dbpedia.org/ontology/genre>',
        enabled: true,
        chart: true,
        name: 'Genre'
    },
```

To není vhodné, protože aplikace se hůře rozšiřuje a mění, když každý typ facetu se rozlišuje pomocí proměnných. Proto v novém řešení se rozlišují facety podle proměnné facetType, kde se specifikuje typ.

Následuje ukázka facetů, které jsou v konfiguračním souboru dema spisovatelů.

```
let facets = {
    name: { //default value facet, not enabled yet
        facetId: 'name',
        facetType: 'text',
        predicate:
        'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
        name: 'Name'
    },
```

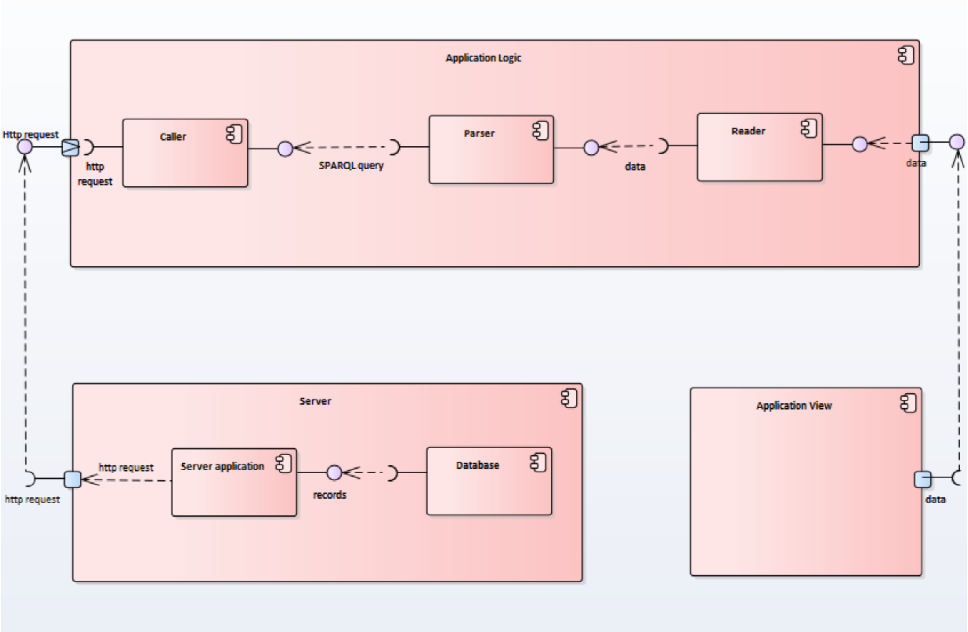
```
    genre: { //basic select facet
        facetId: 'genre',
        facetType: 'select',
        predicate: 'http://dbpedia.org/ontology/genre',
        name: 'Genre',
    },
    birthPlace: { //basic select facet
        facetId: 'birthPlace',
        facetType: 'select',
        predicate: 'http://dbpedia.org/ontology/birthPlace',
        name: 'Birth Place',
    },
    citizenship: { //basic select facet
        facetId: 'citizenship',
        facetType: 'select',
        predicate: 'http://dbpedia.org/ontology/citizenship',
        name: 'United States',
    }
}
```

### 3.1.5 Parsování dotazu z JSON objektu do SPARQL

Existující řešení sémantického vyhledávání nepoužívá obecný Parser JSON objektů do SPARQL dotazu, namísto toho se spoléhá na konkrétní implementaci dotazů pro výběr facetů a pro výběr výsledků. Tento přístup je nevhodný, protože není a modulární aplikace se hůře škáluje pro potřeby vytváření nových typů faset a fasetového vyhledávání. Naproti tomu použití obecného Parseru JSON objektů do SPARQL dotazu zajistí, že aplikace je modulární.

## 3.2 Moduly sémantického fasetového vyhledávače

Sémantický fasetový vyhledávač bude rozdělen na dvě části, na modul logiky a modul zobrazení. Modul logiky bude dále rozdělen na objekty Reader, Parser a Caller. Reader přečte vstupní data aplikace a předá je objektu Parser. Parser bude převádět JSON objekt na SPARLQ dotaz, který předá objektu Caller. Caller pošle dotaz na endpoint, jehož součástí bude vygenerovaný SPARQL dotaz. Na endpointu se vykoná SPARQL dotaz a z endpointu přijde na Caller JSON objekt, ve kterém budou záznamy sémantického vyhledávání. Tyto záznamy se poté předají modulu zobrazení, který příchozí data patřičně zobrazí. Oba tyto moduly budou na sobě logicky nezávislé a každý bude moci použít pro dodatečné účely mimo aplikaci.




**Obrázek 3.1:** Komponent diagram Sémantického facetového vyhledávače

# Kapitola 4
# Implementace

## 4.1 Menu

Vytvořená aplikace z konfigurace je vždycky single-page, všechny vizuální prvky jsou na jedné stránce. Na levém postranním panelu je výběr facetů, nastavených v konfiguraci, pomocí kterých se zobrazí obsah na hlavní stránce.

## 4.2 Strom komponent

Pohled se dělí na strom komponent, kde na vrcholu je komponenta App, která se stará o zobrazení výběrů faset a výsledků, které se podle daného výběru zobrazí. Předává parametry dotazu vytvořené modulem logiky, funkci handler určenou pro komunikaci dolních komponent s horní a identifikaci, která se bere z konfigurace.

### 4.2.1 FacetSelection

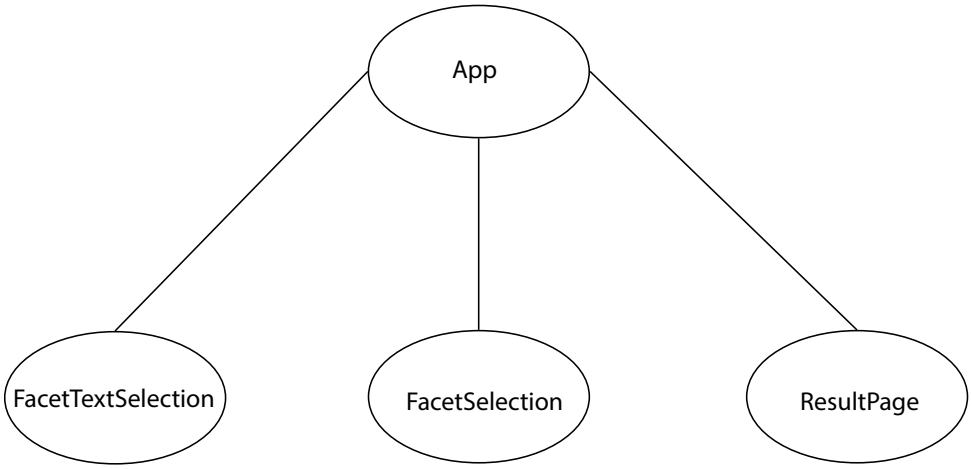
Slouží k zobrazení výběru selektoru základních faset. Posílá dotaz na výběr faset, který dostane od komponenty aplikace na endpoint, od kterého dostane výsledky, které se poté zobrazí. Pomocí funkce handler dá komponentě App vědět, zda došlo ke změně stavu výběru facetu.

### 4.2.2 FacetTextSelection

Slouží k zobrazení textového vstupu pro textové facety. Na základě textového vstupu napsaném uživatelem se mění jak dotazy pro výběr faset, tak dotaz pro zobrazení výsledků. Pomocí funkce handler dá komponentě App vědět, zda došlo ke změně textového facetu. V projektu zatím slouží pouze jako šablona pro budoucí rozšíření funkcionality.

### 4.2.3 Resultpage

Slouží k zobrazení výsledků, které modul logiky sestaví na základě výběru faset. Dostane parametr dotazu vytvořený modulem logiky. Pokud se zobrazí více než 10 výsledků na stránce, tak uživatel může stránkovat mezi výsledky.




**Obrázek 4.1:** Názorné Grafické zobrazení stromu komponent v Reactu

Stránkováním se předá komponentě App informace, na které stránce uživatel je, a podle čísla stránky se komponentě ResultPage předá upravený dotaz.

## 4.3 Konfigurace facetů

Pro konfiguraci aplikace používá pět parametrů: facets, endpointUrl, resultQueryTemplate, selectQueryTemplate a prefixes. Aplikace nejprve načte v konstruktoru facety ze souboru config.js, a rozdělí je na textové a základní. Do objektu se uloží další parametry z config souboru, jako je url adresa endpointu a prefixy.

Následuje ukázka objektu settings, který se načítá modulem logiky.

```
let settings = {
    facets : facets,
    endpointUrl : 'http://dbpedia.org/sparql',
    resultQueryTemplate: resultQueryTemplate,
    selectQueryTemplate: selectQueryTemplate,
    prefixes: []
};
```

### 4.3.1 Endpoint

EndpointUrl je adresa, na které se daný SPARQL dotaz vykoná a ze které přijdou výsledky, které se zobrazí v modulu zobrazení.

### 4.3.2 šablona dotazu výsledků

SelectQueryTemplate je šablona SPARQL dotazu, která se používá pro načtení objektů výběrových facetů. Jako formát obsahuje „<RESULTSET0>" a „<RESULTSET1>", oba jsou určené jakožto zástupné textové řetězce, které

modul logiky nahradí vygenerovanými pod-dotazy. Modul logiky z této šablony vygeneruje dotaz pro každý výběrový facet.

Následuje ukázka šablony pro výběrový dotaz uložená v modulu logiky, která se používá v demu spisovatelů.

```
SELECT DISTINCT ?cnt ?facet_text ?value WHERE {
  {
    {<RESULT_SET0>} }
    BIND("-- No Selection --" AS ?facet_text)
  }
  UNION
  {
    SELECT DISTINCT ?cnt ?value ?facet_text WHERE {
      {<RESULT_SET1>}
      FILTER(BOUND(?value))
      BIND(COALESCE(?value, <http://ldf.fi/NONEXISTENT_URI>)
      AS ?labelValue)
      OPTIONAL {
        ?labelValue skos:prefLabel ?lbl.
        FILTER(LANGMATCHES(LANG(?lbl), "en"))
      }
      OPTIONAL {
        ?labelValue rdfs:label ?lbl.
        FILTER(LANGMATCHES(LANG(?lbl), "en"))
      }
      OPTIONAL {
        ?labelValue skos:prefLabel ?lbl.
        FILTER(LANGMATCHES(LANG(?lbl), ""))
      }
      OPTIONAL {
        ?labelValue rdfs:label ?lbl.
        FILTER(LANGMATCHES(LANG(?lbl), ""))
      }
      BIND(COALESCE(?lbl, IF(!(ISURI(?value)), ?value, "")) AS
      ?facet_text)
    }
  }
}
```

### 4.3.3 šablona dotazu výběru faset

ResultQueryTemplate je šablona SPARQL dotazu, která se používá pro vygenerování SPARQL dotazu z faset. Jako formát obsahuje „<RESULTSET>" určený jako zástupný textový řetězec, který modul logiky nahradí vygenerovaným pod-dotazem. Dotaz může obsahovat prefixy, ty však musí být zahrnuté v konfiguraci. Pro výběr je v šabloně nutné použít proměnnou id.

Následuje ukázka šablony pro výsledný dotaz nastavená uživatelem, která se používá v demu spisovatelů.

```
SELECT * WHERE {
  {<RESULT_SET>}
  FILTER(BOUND(?id))
  OPTIONAL {
    ?id <http://www.w3.org/2000/01/rdf-schema#label> ?name.
    FILTER(LANGMATCHES(LANG(?name), "en"))
  }
  OPTIONAL { ?id <http://dbpedia.org/propertybirthDate>
  ?birthDate. }
  OPTIONAL { ?id <http://dbpedia.org/propertydeathDate>
  ?deathDate. }
  OPTIONAL { ?id <http://dbpedia.org/ontology/thumbnail>
  ?depiction. }
  OPTIONAL {
    ?work__id <http://dbpedia.org/ontology/author> ?id;
      <http://www.w3.org/2000/01/rdf-schema#label>
      ?work__label;
      <http://xmlns.com/foaf/0.1isPrimaryTopicOf> ?work__link.
    FILTER(LANGMATCHES(LANG(?work__label), "en"))
  }
  OPTIONAL { ?id <http://xmlns.com/foaf/0.1isPrimaryTopicOf>
  ?wikipediaLink. }
  OPTIONAL {
    ?id <http://dbpedia.org/propertybirthPlace> ?birthPlace.
    FILTER(LANGMATCHES(LANG(?birthPlace), "en"))
  }
  OPTIONAL {
    ?id <http://dbpedia.org/ontology/abstract> ?abstract.
    FILTER(LANGMATCHES(LANG(?abstract), "en"))
  }
  OPTIONAL {
    ?id (<http://dbpedia.org/ontology/notableWork>/
    <http://www.w3.org/2000/01/rdf-schema#label>)
    ?notableWork.
    FILTER(LANGMATCHES(LANG(?notableWork), "en"))
  }
}
```

### 4.3.4 Prefixy

Prefixes je pole prefixů, které se používají v dotazech. Každý záznam obsahuje klíč, který představuje název prefixu, a hodnotu, která představuje webovou adresu prefixu. Pokud nějaký dotaz používají nějaký prefix, tak se musí daný

prefix dát do tohoto pole, jinak se nevygeneruje dotaz. V případě, že dotazy neobsahují žádné prefixy, tak by mělo být pole prefixů prázdná proměnná.

### 4.3.5 Omezení

Omezení jsou defaultně volené hodnoty, nad kterými se provádí fasetové vyhledávání. Jsou napevno zvolené v konfiguraci a mají v sobě hodnotu objektu a predikátu.

### 4.3.6 Fasety

Fasety se dělí na dva typy: na fasety, u kterých se objekt vybírá v aplikaci a fasety, které už mají předem vybraný objekt. Facety mají proměnnou facetId, ve které je unikátní klíč, reprezentující daný faset. Každá hodnota klíče musí být unikátní, jinak aplikace nebude fungovat. Facety mají proměnnou name, ve které je textový řetězec, který se vypíše nad selektorem objektů v modulu vizualizace. Nemusí být unikátní.

#### Textové fasety

Textové fasety mají jako daný vstup textový řetězec. Používáme, když hledáme nám známé klíčové slovo, pomocí kterého omezujeme počet výsledků a hodnot výběrových facetů.

#### Výběrové facety

Výběrové facety mají jako daný vstup výběr položek. Mají u každé položky zobrazený počet vyhledatelných výsledků, které daná položka facetu má. Používáme pro hledání hodnot, který daný facet může nabývat.

## 4.4 Poddotazy

Modul logiky má v sobě napevno uložené šablony SPARQL pod-dotazů v JSONu, ze kterých se pomocí jejich úpravy sestavuje výsledný dotaz, který se za pomocí Parseru převede do části SPARQL dotazu. Tato část dotazu společně s uživatelem nastavenou šablonou v konfiguraci nám sestaví výsledný dotaz, který je určený na vykonání na endpointu. V tomto modulu jsou defaultně uložené tři objekty dotazu ve formátu, který přijímá knihovna sparqlJS. Tyto objekty slouží jako kostra pro tvorbu dotazů. Dva objekty jsou určené pro tvorbu dotazů výběru facetů, a jeden objekt je určený pro tvorbu dotazu výběru.

Následuje ukázka kostry poddotazu, ze které se poté sestavuje výsledný dotaz.

```
this.resultsubQueryJSON = {
    queryType: "SELECT",
    distinct: true,
```

```
    variables: [
        {
            termType: "Variable",
            value: "id"
        }
    ],
    where: {
        type: "bgp",
        triples: []
    },
    order: [
        {
            expression:
                {
                    termType: "Variable",
                    value: "id"
                }
        }
    ],
    limit: 10,
    offset: 0
};
```

## 4.5 Veřejné metody modulu logiky

Modul logiky poté sestavuje dotazy tak, že při výběru, změně či odstranění objektu u každého facetu v objektu se provede tvorba SPARQL dotazu. Tato změna se vždycky provede po zavolání dostupně veřejných metod setObject-ToFacet(), removeObjectToFacet() a offsetObject().

### 4.5.1 Odstranění objektu faset

Metoda removeObjectToFacet () odstraní hodnotu posledního vybraného facetu a dotazu, z důvodu výběru, aby se všechny zobrazené facetové selektory aktualizovali, a poté se nastaví facetu prázdná hodnota objektu. Má parametry faceftId, podle kterého se najde unikátní facet.

### 4.5.2 Odstranění všech objektů faset

Metoda removeObjectFromAllFacet() nastaví prázdnou hodnotu objektu u všech facetů, které jsou v modulu logiky. Určená pro využití v případě, kdy uživatel chce znova začít vyhledávání záznamů.

### 4.5.3 Nastavení objektu faset

Metoda setObjectToFacet() uloží hodnotu posledního vybraného facetu a dotazu, z důvodu výběru aby se zobrazený používaný facetový selektor neměnil během používání, a poté nastaví do facetu novou hodnotu vybraného objektu, přepisující hodnotu objektu starého. Má parametry facetId, podle kterého se najde unikátní facet, a newValue, který se používá na změnu staré hodnoty.

### 4.5.4 Nastavení hodnoty Offset

Metoda offsetObject() se používá v případě, kdy uživatel stránkuje mezi záznamy. Má parametr pageNumber, které udává číslo stránky, na které uživatel momentálně je. Tato hodnota je o jednu hodnotu menší než hodnota, která se zobrazí uživateli. Na základě hodnoty stránky se změní hodnota offset u kostry JSON objektu výsledného dotazu. Defaultně aplikace vybírá 10 záznamů u výsledného dotazu, hodnota OffSet se teda spočte vynásobením hodnoty stránky deseti.

### 4.5.5 Získání výsledného dotazu

Metoda getResultQuery() se používá na získaní výsledného dotazu, ve kterém jsou záznamy, které uživatel hledá. Tento dotaz se následně využije pro odeslání na endpoint.

### 4.5.6 Vytvoření modifikovaného zdroje

Metoda makeModifiedResource() se používá pro získávání zdrojů při vybírání hodnoty fasetu v aplikaci. Má parameter item, představující vybranou položku v selektoru.

## 4.6 Komunikace s backendem a zpracování výsledků

Výsledky se získají pomocí metody getJSON(), pošle http request na backend s použitím metody GET a se SPARQL dotazem zakódovaným v URL. Metoda vrací výsledky, které přišli z endpointu, a ukládá je do stavu komponenty.

```
getJSON = function (queryUrl) {
    let results;
    let xhr = new XMLHttpRequest();
    xhr.open("GET", queryUrl, true);
    xhr.onload = function () {
        if (xhr.readyState === 4) {
            if (xhr.status === 200) {
                results = JSON.parse(xhr.responseText).results
                .bindings;
                this.setState({ results });
```

```
            } else {
                console.error(xhr.statusText);
            }
        }
    }.bind(this);
    xhr.onerror = function () {
        console.error(xhr.statusText);
    };
    xhr.send();
}
```

Výsledky z backendu se nejprve načtou pomocí metody componentDidMount()

```
componentDidMount() {
    let url = this.props.url;
    let sparql = this.props.query
    let SPARQLQuery = url + encodeURIComponent(sparql) +
    "&format=json";
    this.getJSON(SPARQLQuery)
    document.getElementById(this.props.id).
    addEventListener("change", this.changeState);
}
```

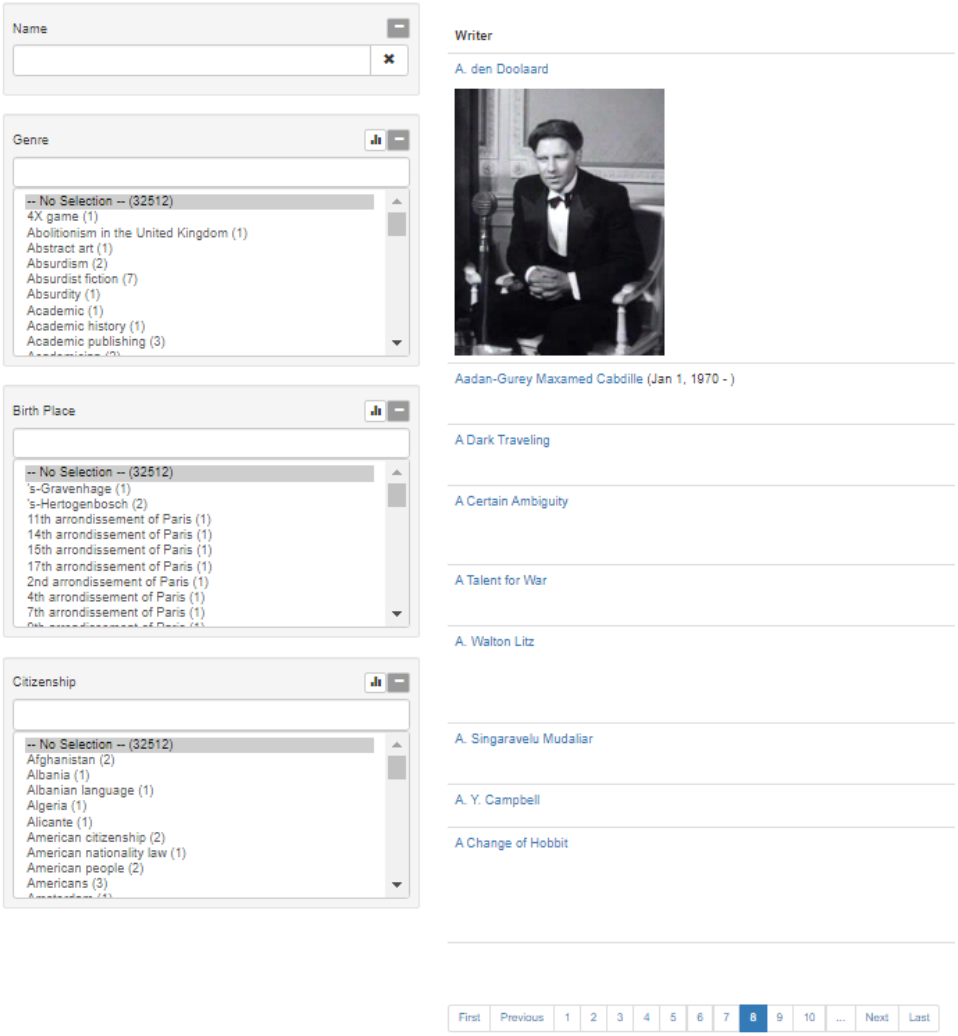
A poté se renderují po změně stavu v komponentě.

```
render() {
    return
    <Form.Control id={this.props.id} as="select" multiple>
    {
        this.formatResults(this.state.results).map((record) =>
        <option key={record.facet_text.value}
        value={record.facet_text.value} >
        {record.facet_text.value+
        " (" + record.cnt.value + ")"}
        </option>)
        }
    </Form.Control>
    }
```

## 4.7 Demo

Abychom ověřili správnost implementace aplikace, tak použijeme konfiguraci, která nám umožní srovnat funkcionalitu této aplikace s již existujícím řešením. Byl vybráno veřejně dostupné demo, které zobrazuje seznam spisovatelů podle výběru žánru, země, ve které se narodili, a země, jíž jsou občany.

V demu používáme jako endpoint https://dbpedia.org/sparql, která má v sobě strukturovaná data těžená z Wikipedie ve formě RDF.




**Obrázek 4.2:** Ukázka dema existujícího řešení pro výběr a zobrazení spisovatelů na základě facetů žánr, datum narození a občanství

V základu aplikace vybírá všechny spisovatele, které nalezne na endpointu Dbpedia. Základní výběrové facety mají vybráno možnost "—No-Selection—". Filtrujeme záznamy tak, že vybíráme hodnoty facetů, které jsou k dispozici u základních výběrových facetů a psaním textového vstupu u textového facetu. Výběr se ruší kliknutím na defaultní hodnotu "—No-Selection—"u základních výběrových facetů.

# Kapitola 5

# Testování aplikace

## 5.1 Unit testy

Cílem unit testů je otestovat část logického modulu zodpovědného za tvorbu dotazů. Vzhledem k tomu, že sémantický vyhledávač je důkladně otestován, tak v unit testech se bude testovat pouze očekávaný formát výsledných dotazů v závislosti na nastavení objektu LogicModule. Celkem je v aplikaci 8 Unit testů a testují se v nich všechny kombinace nastavení objektu pro konfiguraci vyhledávání spisovatelů.

Následuje ukázka testu, který má na starosti ověřit, zda dotaz odpovídá dotazu, který pošle existující demo endpointu.

```
it('test1', function () {
    let logicModule = new LogicModule(settings);
    logicModule.setObjectToFacet("genre",
    "http://dbpedia.org/resource/Journalism")
    logicModule.setObjectToFacet("birthPlace",
    "http://dbpedia.org/resource/United_States")
    logicModule.setObjectToFacet("citizenship",
    "http://dbpedia.org/resource/United_States")
    let resultQuery = logicModule.getResultQuery();
    expect(resultQuery).
    toContain
    ("?id <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>" +
    <http://dbpedia.org/ontology/Writer>;\n" +
        "    <http://dbpedia.org/ontology/genre>" +
        "<http://dbpedia.org/resource/Journalism>;\n" +
        "    <http://dbpedia.org/ontology/birthPlace>" +
        <http://dbpedia.org/resource/United_States>;\n" +
        "    <http://dbpedia.org/ontology/citizenship> "
        <http://dbpedia.org/resource/United_States>.")
    let selectFacetQueries =
    logicModule.getSelectFacetQueries()
})
```

## 5.2 Integrační testy

cílem integračním testů je otestovat, zda dotazy odeslané na endpoint vytvořené při různých konfiguracích objektu LogicModule odpovídají výsledkům existujícího dema.

Následuje ukázka testu, který má na starosti ověřit, zda výsledky dotazu odpovídají výsledkům, které byly získány při použití dema existujícího řešení.

```
it('test1', function () {
    let logicModule = new LogicModule(settings);
    logicModule.
    setObjectToFacet("genre",
    "http://dbpedia.org/resource/Journalism")
    logicModule.
    setObjectToFacet("birthPlace",
    "http://dbpedia.org/resource/United_States")
    logicModule.
    setObjectToFacet("citizenship",
    "http://dbpedia.org/resource/United_States")
    logicModule.
    getSelectFacetQueries();
    let resultQuery
    = logicModule.getResultQuery();
    let selectQueries
    = logicModule.getSelectFacetQueries();
    let keys =
    Object.keys(selectQueries).map(result => result)
    let results =
    logicModule.getResultsSync(resultQuery);
    let array = []
    for(let item in results) {
        array.push(results[item].name.value)
    }
    expect(array).
    toStrictEqual(["Lloyd Grove", "Linda Wolfe"])

})
```

# Kapitola 6
# Závěr

## 6.1 Shrnutí

Bylo docíleno vytvoření aplikace, která odděluje logiku od uživatelského rozhraní. Díky tomu bude aplikace mnohem lépe udržovatelná a bude lépe přizpůsobitelná v případě obměně Javascriptových technologií, které může nastat v budoucnosti.

Universální parser JSON objektu do SPARQL je od doby zadání práce vytvořen a je používán jako standardní knihovna, která je důkladně otestována. U této knihovny se předpokládá, že pro účely použití v této aplikaci bude fungovat spolehlivě a nebude dělat žádné problémy.

Provedla se analýza a srovnání s existujícím řešením, a došlo na vylepšení použitelnosti konfiguračního souboru a práce s veřejně dostupnými metodami, díky čemuž je aplikace mnohem flexibilnější a snadnější na práci oproti existujícímu řešení.

Dále se provedlo praktické vyzkoušení aplikace při tvorbě dema se spisovateli a srovnání jeho funkcinality s demem spisovatelů existujícího řešení. Za pomoci použití integračních a unit testů má aplikace prověřenou funkčnost v modulu logiky a díky demu.

Proces instalace a spuštění je díky použití moderních technologií Node.js a React mnohem více uživatelsky přívětivý.

## 6.2 Budoucí vývoj

Jak už bylo v práci zmíněno, tento projekt má zatím pouze funkcionalitu, která byla požadována během proběhlých konzultacích. Chybí implemenace facetů pro výběr data, hierarchický facet a různé variace základních facetů, které jsou zahrnuté v existujícím řešení.

Naštěstí projekt je ve stavu, že oproti existujícím řešením není problém jej o tuto funkcionalitu rozšířit, případně použít modul logiky k jiným účelům, než pouze single-page aplikace, nad kterou se pomocí faset vyhledávají požadovaná data. Bude možné přidat nové facety a vytvářet různé modifikace existujících facetů pro konktrétní využití.

Projekt je tedy v dobrém stavu pro učely možného budoucího rozšíření a

záleží pouze na vyvojářích, kteří jej budou využívat, jak s ním budou zacházet a na co v budoucnu přijdou.

# Kapitola 7

# Zdroje


1. HTML Tutorial. W3Schools [online]. [cit. 2021-01-05].
   Dostupné z: https://www.w3schools.com/html/
2. CSS. W3Schools [online]. [cit. 2021-01-05].
   Dostupné z: https://www.w3schools.com/css/
3. Javascript. W3Schools [online]. [cit. 2021-01-05].
   Dostupné z: https://www.w3schools.com/js/
4. The semantic Web made easy. W3.org [online]. [cit. 2021-01-05].
   Dostupné z: https://www.w3.org/RDF/Metalog/docs/sw-easy
5. Linked Data. W3.org [online]. [cit. 2021-01-05].
   Dostupné z: https://www.w3.org/standards/semanticweb/data
6. VOCABULARIES. W3.org [online]. [cit. 2021-01-05].
   Dostupné z: https://www.w3.org/standards/semanticweb/ontology
7. Questions on RDF, Ontologies, SPARQL, Rules. . . . W3.org [online]. [cit. 2021-01-05].
   Dostupné z: https://www.w3.org/2001/sw/SW-FAQ#whrdf
8. Abstract. W3.org [online]. Harris, Garlik, 2013 [cit. 2021-01-05].
   Dostupné z: https://www.w3.org/TR/sparql11-query/
9. Mocha. W3.org [online]. [cit. 2021-01-05].
   Dostupné z: https://mochajs.org/
10. [11] Introducing JSX. W3.org [online]. [cit. 2021-01-05].
    Dostupné z: https://reactjs.org/docs/introducing-jsx.html
11. SPARQL.js – A SPARQL 1.1 parser for JavaScript.
    Https://www.npmjs.com/ [online]. [cit. 2021-01-05].
    Dostupné z: https://www.npmjs.com/package/sparqljs

12. About npm. Https://www.npmjs.com/ [online]. [cit. 2021-01-05].
Dostupné z: https://docs.npmjs.com/about-npm

13. Bootstrap 4 Tutorial. W3Schools [online]. [cit. 2021-01-05]. Dostupné z:
https://www.w3schools.com/bootstrap4/

14. Git –distributed-is-the-new-centralized. Git [online]. [cit. 2021-01-05].
Dostupné z: https://git-scm.com/

15. What is Faceted Search? SearchHub [online]. SearchHub, 2009 [cit. 2021-01-05].
Dostupné z:
http://web.archive.org/web/20140316215652/
http://searchhub.org/2009/09/02/faceted-search-with-solr/

16. What is the HTML DOM? Git [online]. W3schools [cit. 2021-01-05].
Dostupné z: https://www.w3schools.com/whatis/whatis_htmldom.asp

17. State and Lifecycle. ReactJS [online]. reactjs.org [cit. 2021-01-05].
Dostupné z: https://reactjs.org/docs/state-and-lifecycle.html

18. Rendering Elements. ReactJS [online]. reactjs.org [cit. 2021-01-05].
Dostupné z: https://reactjs.org/docs/rendering-elements.html

19. Handling Events. ReactJS [online]. reactjs.org [cit. 2021-01-05].
Dostupné z: https://reactjs.org/docs/handling-events.html

20. NodeJS About [online]. NodeJS [cit. 2021-01-05].
Dostupné z: https://nodejs.org/en/about/

# Kapitola 8

# Obrázky

1. Obrázek 2.1.
   http://searchhub.org//wp-content/uploads/2012/08/CNET_faceted_search.jpg

2. Obrázek 2.2.
   https://snipcademy.com/img/articles/javascript-document-object-model/dom.svg

3. Obrázek 2.3.
   https://upload.wikimedia.org/wikipedia/commons/thumb/8/88/Basic_RDF_Graph.svg/640px-Basic_RDF_Graph.svg.png

4. Obrázek 2.4.
   https://www.w3.org/TR/rdf11-primer/example-graph.jpg

# Kapitola 9

# Příloha

Adresa projektu je https://gitlab.fel.cvut.cz/svacefil/bcproject
Po naklonování projektu je nutné spustit příkaz

```
run npm install
```

Aby se nainstalovali všechny přidružené balíčky prostředí npm.
Poté se pomocí příkazu

```
run start
```

spustí projekt v Reactu.
v prostředí npm se spustí po instalaci mocha příkaz

```
mocha tests/sparql-creator.js
```

na spuštění testů.