ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Sémantické facetové vyhledávání na platformě React

Daniel Bourek
Softwarové inženýrství a technologie

Květen 2022
Vedoucí práce: Ing. Martin Ledvinka, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bourek** Jméno: **Daniel** Osobní číslo: **478425**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Sémantické facetové vyhledávání na platformě React

Název bakalářské práce anglicky:

React-based Semantic Faceted Search

Pokyny pro vypracování:

1. Srovnejte existující přístupy k facetovému vyhledávání, především pak z hlediska využití sémantických technologií.
2. Navrhněte modul sémantického facetového vyhledávače, který bude umožňovat rozdělení vyhledávání a jeho vizualizace do samostatných modulů.
3. Naimplementujte navržené řešení včetně vizualizačního modulu.
4. Ověřte funkčnost řešení srovnáním s existujícím facetovým vyhledávačem na stránkách https://slovník.gov.cz/prohlížeč.

Seznam doporučené literatury:

[1] D. Allemang, J. Hendler, Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL, Morgan Kaufmann, 2011
[2] R. Wieruch, The Road to learn React: Your journey to master plain yet pragmatic React.js, 2018
[3] G. M. Sacco, Y. Tzitzikas, Dynamic Taxonomies and Faceted Search: Theory, Practice, and Experience, Springer, 2009

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Martin Ledvinka, Ph.D. skupina znalostních softwarových systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **02.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Martin Ledvinka, Ph.D.	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce		podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování / Prohlášení

Chtěl bych poděkovat vedoucímu této práci, Ing. Martinovi Ledvinkovi Ph.D., za výborné vedení této práce a veškerou pomoc poskytnutou při její tvorbě. Opravdu si nedokážu představit lepšího vedoucího bakalářské práce.

Dále bych chtěl poděkovat své rodině za neustálou podporu a hlavně trpělivost při mých studiích. V neposlední řadě také své přítelkyni Cecílii, která mě po celou dobu tvorby práce motivovala, podporovala a podílela se na její jazykové korektuře.

Čestně prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Daniel Bourek
V Praze, 19. 5. 2022

Abstrakt / Abstract

Práce se zabývá problematiku facetového vyhledávání se zaměřením na jeho použití pro sémantická data. Zprvu čtenáře seznámí s pojmem sémantický web a klíčovými technologiemi, které se ho týkají. Poté zanalyzuje přístupy k facetovému vyhledávání a navrhne vhodný pro sémantická data. Jádrem práce je návrh a implementace sémantického facetového vyhledávače. Ten je implementován ve frameworku React, ale je koncipován tak, aby byl použitelný v jakémkoliv JavaScript frameworku. Celý proces je řádně zdokumentován. Na závěr je vyhledávač vyhodnocen a porovnán s již existujícím na adrese `https://slovník.gov.cz/prohlížeč`.

Klíčová slova: sémantický web, facetové vyhledávání, SPARQL, RDF, React

Semantic Web is an extension of the current Web which uses semantic data model to express real-world data in a machine-readable format. This work deals with designing and implementing faceted search for this semantic data. First, it researches and presents key semantic web concepts and technologies. It then analyzes approaches to facet search and suggests a suitable approach for semantic data. The core of the work is the design and implementation of a semantic faceted search engine. The search engine is implemented in the React framework but it is designed to be usable in any JavaScript framework. The whole process is properly documented. Finally, the implemented faceted search is evaluated and compared with the existing one at `https://slovník.gov.cz/prohlížeč`.

Keywords: Semantic Web, facet search, SPARQL, RDF, React

Obsah /

1 Úvod **1**

2 Sémantický web **3**

2.1 Co je sémantický web? 3

2.2 Sémantické technologie 3

2.2.1 RDF 3

2.2.2 OWL 2 4

2.2.3 Linked Data 4

2.2.4 SPARQL 5

3 Facetové vyhledávání **7**

3.1 Popis 7

3.2 Typy facetů 8

3.2.1 Select facet 8

3.2.2 Checkbox facet 8

3.2.3 Range facet 8

3.2.4 Date facet 8

3.2.5 Bucket facet 8

3.2.6 Text facet 8

3.3 Analýza přístupu k facetovému vyhledávání 8

3.3.1 Elasticsearch 9

3.3.2 SPARQL Faceter 10

3.4 Srovnání přístupů z hlediska sémantických technologií . . 11

4 Návrh **12**

4.1 Architektura 12

4.2 sfs-api 14

4.2.1 Facet 14

4.2.2 SfsApi 14

4.2.3 Eventy 15

4.3 react-sfs 16

4.4 Přístup k facetovému vyhledávání 17

5 Implementace **19**

5.1 Použité knihovny 19

5.2 Implementační detaily 19

6 Vyhodnocení **21**

6.1 Testy 21

6.2 Srovnání s existujícím vyhledávačem 22

7 Závěr **24**

7.1 Plány do budoucna 24

Literatura **26**

A Elektronická příloha práce **29**

A.1 Odkazy 29

A.2 Pokyny ke spuštění 29

B Slovníček **30**

Kapitola 1
Úvod

Málokterý vynález ovlivnil svět v takové míře jako vznik World Wide Web (zkráceně WWW či web). Za poměrně krátkou dobu své existence se web rozšířil téměř do každé části našeho života a dnes si bez něj lze svět jen těžko představit. Oproti ostatním ICT technologiím, které se často výrazně inovují a mění každých několik let, funguje web již 20 let téměř stejně.

To se však začíná měnit s příchodem sémantického webu, který zásadně ovlivňuje, jak přistupujeme k datům v internetu – místo relací mezi dokumenty přes hypertextové odkazy můžeme vytvářet relace mezi fakty. Svět lze tak mnohem lépe popsat a stává se pro nás srozumitelnější. Navíc jsou tyto relace jednoduše strojově čitelné, tudíž se stává srozumitelnější nejen pro nás, ale i pro stroje. Ty poté mohou nad těmito daty mnohem přesněji vyhledávat informace či vykonávat automatizace.

Interakce se sémantickými daty vyžaduje nové přístupy k ukládání, zpracování a vyhledávání dat. Právě vyhledáváním v sémantických datech se zabývá tato práce, konkrétně facetovým vyhledáváním. Facetové vyhledávání, tedy zatřídění vyhledaných výsledků do různých kategorií, je v současné době velice rozšířené. Pomáhá nám upřesnit výsledky vyhledávání a najdeme jej například téměř v každém větším e-shopu.

Přístupů k facetovému vyhledávání je více, ne všechny jsou však vhodné pro sémantická data. Nad sémantickými daty tak existuje velmi málo řešení facetového vyhledávání a ta existující mají své nedokonalosti. Často jsou závislá na nějakou platformu nebo jsou velmi omezující.

Snahou této práce je navrhnout a implementovat sémantický facetový vyhledávač, který by byl jednoduše lepší. Měl by tedy být nezávislý na platformě a co nejméně omezující, aby mohl být integrován do různých projektů. K implementování facetového vyhledávače byla sice zvolena platforma React, ale jeho logika vyhledávání by měla být integrovatelná do jakékoliv JavaScript platformy. Vyhledávač tak musí být rozdělen do modulů na samotné vyhledávání a jeho vizualizace.

Jedním z existujících řešení sémantického facetové vyhledávání je SPARQL Faceter, který je dostupný pouze na platformě AngularJS. Toto řešení využívá stránka Prohlížeče sémantického slovníku pojmů Ministerstva vnitra České republiky (MVČR) na adrese `https://slovník.gov.cz/prohlížeč`. Sémantický facetový vyhledávač implementovaný v této práci bude s tímto prohlížečem porovnán.

Cílem této práce jsou:

- Srovnat existující přístupy k facetovému vyhledávání, především pak z hlediska využití sémantických technologií.
- Navrhnout modul sémantického facetového vyhledávače, který bude umožňovat rozdělení vyhledávání a jeho vizualizace do samostatných modulů.
- Naimplementovat navržené řešení včetně vizualizačního modulu.
- Ověřit funkčnost řešení srovnáním s existujícím facetovým vyhledávačem na stránkách `https://slovník.gov.cz/prohlížeč`.

Kapitola 2
Sémantický web

V této kapitole se seznámíme s pojmem sémantický web a popíšeme si klíčové technologie týkající se tohoto pojmu. Ty jsou zásadní pro pochopení fungování světa sémantických dat, a tak tedy i k pochopení této práce.

2.1 Co je sémantický web?

Myšlenka sémantického webu byla veřejnosti poprvé představena 17. května 2001, kdy v časopise Scientific American vyšel článek The Semantic Web [1]. Autory tohoto článku byli Tim Berners-Lee (zakladatel WWW), James Hendler a Ora Lassila, všichni tři jsou zásadními postavami ve vývoji sémantického webu. Na začátku tohoto článku popisují poměrně futuristickou scénku, kde po otevření webové stránky je zařízení schopné samo kompletně porozumět obsahu této stránky. Tedy veškerým informacím na ní napsané, včetně odkazů na jiné stránky a vztahů mezi nimi. Díky tomu pak pouze skrze komunikaci s dalšími stránkami naplánuje návštěvu lékaře, včetně toho, aby vybral lékaře, který vyhovuje časovým možnostem uživatele, byl blízko jeho domu a mohl být zaplacený jeho pojišťovnu. Klíčové je zde to, že to zařízení zvládlo jen za pomocí webu, díky strojově čitelným standardizovaným datům na webových stránkách.

Sémantický by se tak měl stát novým evolučním stupněm stávajícího webu [1], kde jsou informace uloženy podle standardizovaných pravidel, což usnadňuje jejich vyhledávání a zpracování [2]. Ona standardizovaná pravidla jsou hlavně Resource Description Framework (RDF) a Web Ontology Language (OWL 2). Ty byly vyvinuty mezinárodním konsorciem W3C, které ve spolupráci s veřejností vyvíjí i jiné webové standardy, pomocí nichž chtějí rozvinout web do plného potenciálu [3].

Pro ověření pravosti dokumentů a jejich informací využívá sémantický web digitální podpisy a šifrování [4].

2.2 Sémantické technologie

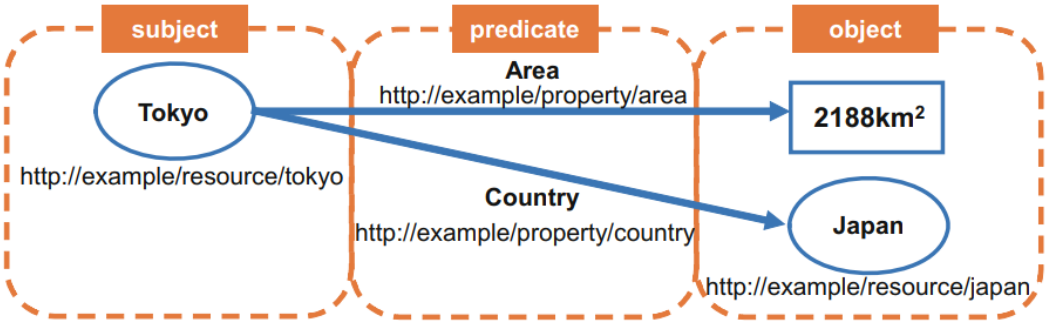
V této sekci si popíšeme relevantní sémantické technologie a formáty. Tyto technologie budeme zmiňovat v této práci často, jelikož jsou základem pro práci se sémantickými daty. Budou tím pádem také využity při implementaci sémantického facetového vyhledávače.

2.2.1 RDF

V sémantickém světě je standardem pro reprezentaci dat formát RDF [5]. RDF je standardizovaný strojově čitelný grafový formát, ve kterém se používají tzv. triples, česky trojice, k popsání relací ve formátu subjekt - predikát - objekt. Takovou trojici lze znázornit jako orientovanou hranu mezi dvěma uzly v grafu, kde predikát je hranou

[1] Sémantický web je také často nazýván jako Web 3.0, ale fakticky je spíše jeho částí.

jdoucí z uzlu subjekt do uzlu objekt. Množina těchto trojic pak popisuje orientovaný graf, který nazýváme RDF graf. Ukázkou takového grafu je obrázek 2.1.



Obrázek 2.1. Ukázka RDF grafu, který je popsaný dvěma trojicemi [6].

Jednotlivé uzly v RDF grafu nazýváme prvky a jsou identifikované pomocí IRI, což je nadmnožinou URI, která povoluje více Unicode znaků. URI je textový řetězec s definovanou strukturou, který slouží k identifikaci zdroje informací [7].

Prvky však mohou být kromě IRI i literály či tzv. blank nodes. Literály se používají přímo pro nějakou textovou, číselnou či časovou hodnotu a lze je tak použít v RDF trojici pouze jako objekt. Blank node představuje uzel bez identifikátoru. Ten může existovat pouze v rámci lokálního RDF grafu a při potřebě identifikace je transformován na IRI.

Syntaxe RDF není definovaná, nejčastěji se však používá RDF/XML [8], která umožňuje zapsat RDF graf jako XML dokument, či Turtle [9], který je více podobný běžnému textu. Obě syntaxe jsou vytvořeny a doporučeny W3C. Pro účely dnešních aplikací je nutné také zmínit existenci formátu RDFJS, který reprezentuje RDF data v jazyku JavaScript [10].

2.2.2 OWL 2

V informatice se ontologií nazývá explicitní a formalizovaný popis určité problematiky. Pro popsání základních ontologií vzniklo RDF Schema (RDFS), které obsahuje sadu základních tříd k použití [11]. Později se však vyvinul Web Ontology Language (OWL 2), který je mnohem bohatší a stal se tak standardem pro popis ontologií [12].

2.2.3 Linked Data

Cílem sémantického webu je propojování informací na webu. Tyto propojená data označuje pojem Linked Data, který vymyslel Tim Berners-Lee [13]. Společně s tím definoval čtyři přincipy, které by měly Linked Data splňovat:

- Používejte URI jako k jména věcí.
- Používejte HTTP URI, aby se mohli lidé na tyto jména podívat na internetu.
- Poskytněte užitečné informace na stránce URI.
- V těchto informacích zahrňte odkazy na další URI, aby mohli lidé objevit související věci.

Linked data tak mají, díky těmto principům, standardizovat reprezentaci dat a přístup k nim na webu [14]. Díky relacím mezi jednotlivými Linked Data sety, tak může vzniknout jeden globální graf dat, podobně jako hypertextové odkazy na klasickém webu spojují všechny HTML dokumenty do jednoho globálního informačního prostoru.

Tim Berners-Lee později definoval také hodnocení kvality Linked Data od 1 do 5 hvězdiček [13]. Pro splnění stupně hodnocení musí data splňovat i předchozí stupně. Toto hodnocení je definované takto:

- 1 hvězdička - data jsou dostupná na webu (v jakémkoliv formátu), ale s otevřenou licencí, jako Open Data.
- 2 hvězdičky - data jsou k dispozici jako strojově čitelná strukturovaná data (například Excel dokument místo obrázku tabulky).
- 3 hvězdičky - data jsou v nechráněném formátu (například formát CSV místo Excel dokumentu).
- 4 hvězdičky - data používají otevřené standardy od W3C (RDF a SPARQL) k identifikaci věcí, aby mohli ostatní lidé na vaše data odkazovat.
- 5 hvězdičky - data jsou propojená s daty jiných lidí, aby poskytovaly kontext.

Mezi největší Linked Data sety patří DBPedia.org [15] - obdoba encyklopedie Wikipedia se sémantickými daty či GeoNames, popisující geografii na zeměkouli.

2.2.4 SPARQL

Primárním dotazovacím jazykem pro RDF je SPARQL [16]. Ten je syntaxí velmi podobný SQL[2], funguje však spíše na porovnávání a dosazování RDF trojic, tedy potažmo podgrafu RDF. Dotaz se pak skládá z množiny těchto trojic (a dalších klauzulí SPARQL), přičemž každý prvek z této trojice může být proměnnou. SPARQL se pak snaží těmto trojicím, a tedy i proměnným, najít řešení. Proměnné jsou značeny prefixem „?“.

Ve SPARQL jsou 4 možné různé druhy dotazů:

- SELECT – podobný SQL SELECT dotazu, tedy vrací data vyhovující dotazu.
- CONSTRUCT – zkonstruuje, dle dotazu, data ve formátu RDF.
- ASK – vrací boolean hodnotu true/false podle toho, jestli existují vyhovující data dotazu.
- DESCRIBE – vrací popis dat, které vyhovují dotazu.

Obsahuje klauzule jako BIND k přiřazování hodnot k proměnným či OPTIONAL, který je podobný k LEFT JOIN z SQL. SPARQL také obsahuje, podobně jako SQL, řadu operátorů, kterými lze zadané výsledky filtrovat. Za zmínku stojí třeba isIRI, který testuje, zda je prvek IRI nebo bound, který testuje, zda má proměnná přiřazenou hodnotu. Pro lepší čitelnost dotazu lze také využít klauzule PREFIX, která nahradí IRI ontologie definovaným prefixem.

```
PREFIX ex: <http://example.com/exampleOntology#>
SELECT ?capital
       ?country
WHERE
  {
    ?x  ex:cityname       ?capital   ;
        ex:isCapitalOf    ?y         .
    ?y  ex:countryname    ?country   ;
        ex:isInContinent  ex:Africa  .
  }
```

[2] SQL je standardizovaný jazyk pro práci s daty v relačních databází.

Jak můžeme vidět na příkladu SPARQL dotazu výše, trojice musí být odděleny pomocí tečky či středníku, přičemž oddělení středníkem je pouze „syntaktický cukr“[3] pro použití stejného subjektu z předchozí trojice.

Je zvykem, že stránky se sémantickými daty, neboli Linked Data sety, obsahují SPARQL endpoint, kam lze zadávat SPARQL dotazy. SPARQL endpoint je nejpoužívanějším způsobem zpřístupnění sémantických dat [17]. Pro implementaci SPARQL endpointu se často využívá řešení od firmy OpenLink jménem Virtuoso [18].

[3] Pojem „syntaktický cukr“ označuje část syntaxe programovacího jazyka, jejímž účelem je usnadnit programátorovi zápis nějaké operace.

Kapitola 3
Facetové vyhledávání

V této kapitole si popíšeme, co je facetové vyhledávání a jak může vypadat z pohledu uživatele. Poté si zanalyzujeme různé přístupy k implementaci facetového vyhledávání díky rešerši populárních řešení. Nakonec tyto přístupy srovnáme z hlediska využití sémantických technologií.

3.1 Popis

Obrázek 3.1. Ukázka facetového vyhledávání s vysvětlivkami [19].

Facetové vyhledávání je zatřídění vyhledaných výsledků do různých kategorií (facetů) dle kterých se dá sada výsledků dále filtrovat. Dá se jím tak obohatit každé vyhledávání, ale často bývá spojeno s fulltextovým vyhledáváním[1], kde uživateli umožňuje dále upřesnit výsledky jeho vyhledávání. Hojně se využívá třeba v e-commerce sektoru, kde podle studie Nielsen Norman Group (NNG) z roku 2018 jsou e-shopy bez facetového vyhledávání výjimkou [21]. Facety bývají často znázorněny webovými elementy select či checkbox nebo také posuvníky. Příklad využití facetů v e-shopu společně s vysvětlikami je ilustrován na obrázku 3.1.

Po zvolení hodnoty facetu, facet, dle této hodnoty, filtruje výsledky vyhledávání a většinou také možnosti ostatních facetů. Podmínky, kterým musí výsledek či možnost facetu vyhovět, aby úspěšně prošli tímto filtrováním, budeme v této práci nazývat „constraints“.

[1] Fulltextové vyhledávání je technika hledání textu zadaného uživatelem v elektronických dokumentech a databázích [20].

Jelikož není definovaný žádný standard facetového vyhledávání, zadefinujeme si, co by měl takový modul facetového vyhledávání splňovat:

- Facet obsahuje možnosti pro každý unikátní výsledek ze sady výsledků.
- Jednotlivé facety lze kombinovat mezi sebou.
- Po výběru hodnoty facetu se musí aktualizovat možnosti ostatních facetů.
- Mezi constraints facetů platí logický AND, tzn. aby se výsledek objevil v sadě výsledků, musí vyhovět všem aktivním facetům.
- Možnosti facetů ukazují počet výsledků, které aplikování facetu s danou hodnotou v aktuálním stavu vrátí.
- Možnosti facetů, které by vrátily prázdnou sadu výsledků se nezobrazují nebo jsou „disabled“[2]

3.2 Typy facetů

Facety si můžeme rozřadit dle toho jakým způsobem se volí jejich hodnoty z pohledu uživatele. V této práci budeme tyto kategorie nazývat jako typy facetů.

3.2.1 Select facet

Facet s možností volby nejvýše jedné hodnoty, podle které je pak sada výsledků filtrována. Ovládacím prvkem bývá select element.

3.2.2 Checkbox facet

Facet s možností volby více hodnot skrz zaškrtávání checkboxů.

3.2.3 Range facet

Facet pro číselná data s možností nastavení rozsahu. Ovládacím prvkem bývá posuvník (input element s hodnotou atributu type range).

3.2.4 Date facet

Facet pro výběr datumu. Většinou otevře komponent kalendáře, tzv. date picker, kde je možné vybrat datum, či časové rozmezí.

3.2.5 Bucket facet

Podobné jako range facet, ale neovládá se posuvníkem, ale jsou nadefinovány a pojmenovány rozsahy, dle kterých se pak dají výsledky filtrovat.

3.2.6 Text facet

Podobný fulltextovému vyhledávání. Filtruje výsledky dle zadaného textu.

3.3 Analýza přístupu k facetovému vyhledávání

Abychom získali lepší přehled o facetovém vyhledávání, zanalyzujeme si dvě jeho řešení, každé s odlišným přístupem k facetovému vyhledávání. Ty byly do práce vybrány jako ukázkové příklady po rešerši dostupných řešeních facetového vyhledávání. Nejdříve zanalyzujeme facetové vyhledávání od Elasticsearch, které je typickým řešení facetového vyhledávání pro běžná data. Poté si rozebereme sémantický facetový vyhledávač SPARQL Faceter.

[2] Jejich HTML ovládací prvek má atribut disabled.

3.3.1 Elasticsearch

Elasticsearch je open source[3] nástroj pro fulltextové vyhledávání nad různými daty [22]. Má mnoho funkcí a možností, nás však zajímá pouze jeho část pro práci s facety.

```
{
  "query": "park",
  "facets": {
    "states": [
      {
        "type": "value",
        "name": "top-five-states",
        "sort": { "count": "desc" },
        "size": 5
      }
    ]
  }
}
```

Výše můžeme vidět ukázkový požadavek na vyhledávání query „park“ společně s definováním facetu pro další filtrování [23]. Facet je definován jménem „top-five-states“ a požaduje 5 možností řazených sestupně dle počtu výskytů. Očekáváme tedy, že možnosti facetů budou poslány společně s výsledky vyhledávání.

Níže pak máme odpověď na tento požadavek. Možnosti facetů jsou strukturovány vždy jako hodnota a počet výskytů.

```
{
  ## odpověď obsahuje více polí, která však pro nás nejsou důležitá
  "results": [
    ## výsledky které vyhovují query "park"
  ]
  "facets": {
    "states": [
      {
        "type": "value",
        "name": "top-five-states",
        "data": [
          {
            "value": "California",
            "count": 8
          },
          {
            "value": "Alaska",
            "count": 5
          },
          {
            "value": "Utah",
            "count": 4
```

[3] Kvůli nedávné změně licence už technicky není úplně open source, stále má však veřejný kód a od založení fungoval jako open source.

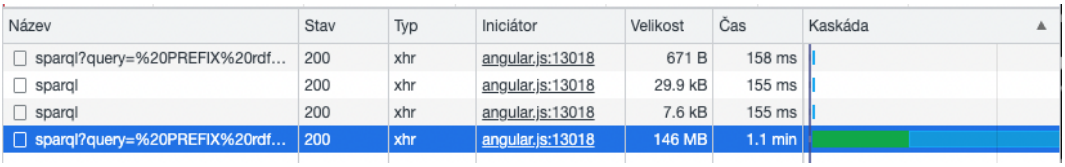
```
            },
            {
              "value": "Colorado",
              "count": 3
            },
            {
              "value": "Washington",
              "count": 3
            }
          ]
        }
      ]
    }
  }
```


3.3.2 SPARQL Faceter

Dalším příkladem přístupu k facetovému vyhledávání je modul na sémantické facetové vyhledávání od výzkumné skupiny Semantic Computing (SeCo) [24] s názvem SPARQL Faceter [25]. Balíček tohoto modulu se jmenuje *sparql-faceter* a je to knihovna pro JavaScript framework Angular.

Tato knihovna je pro nás důležitá ze dvou důvodů. Zaprvé je to jedno z mála řešení facetového vyhledávání pro sémantická data, tudíž může být inspirací pro náš návrh. Zadruhé je touto knihovnou implementován sémantický facetový vyhledávač, se kterým se bude srovnávat i vyhledávač vzniklý v rámci této práce. Srovnání těchto vyhledávačů se budeme věnovat v sekci 6.2, tudíž nebudeme v této sekci „zabrušovat“[4] do úplných implementačních detailů, ale pouze se podíváme, jakým způsobem se posílají požadavky na výsledky a možnosti facetů. Zmíněný existující sémantický facetový vyhledávač[5] budeme dále v práci nazývat jako „prohlížeč MVČR“.

Jako příklad na kterém budeme SPARQL Faceter analyzovat jsme si příhodně vybrali prohlížeč MVČR, který má dva facety. Požadavky, které posílá SPARQL Faceter jsou prosté HTTP POST či GET požadavky na SPARQL endpoint[6]. Samotnou SPARQL query pak skládá právě knihovna SPARQL Faceter na klientské straně.



Název	Stav	Typ	Iniciátor	Velikost	Čas	Kaskáda
sparql?query=%20PREFIX%20rdf…	200	xhr	angular.js:13018	671 B	158 ms	
sparql	200	xhr	angular.js:13018	29.9 kB	155 ms	
sparql	200	xhr	angular.js:13018	7.6 kB	155 ms	
sparql?query=%20PREFIX%20rdf…	200	xhr	angular.js:13018	146 MB	1.1 min	

Obrázek 3.2. Asynchronní požadavky prohlížeče MVČR na SPARQL endpoint při načtení stránky.

Obrázek 3.2 ukazuje asynchronní požadavky, které se odešlou při načítání stránky prohlížeče MVČR. Z tohoto obrázku můžeme vidět, že se při načtení stránky prohlížeče pošlou celkem čtyři asynchronní požadavky na SPARQL endpoint. První dotazuje celkový počtu výsledků. Tento počet se zobrazuje u facetů pro žádnou nevybranou možnost. Další dva požadavky jsou pak na možnosti facetů v tomto prohlížeči,

[4] Abychom z toho nemuseli pak zase „vybrušovat“.
[5] `https://slovník.gov.cz/prohlížeč`
[6] V tomto případě `https://slovník.gov.cz/sparql`.

které jsou dva. Pro každý z facetů se tedy odesílá separátní požadavek na jeho možnosti. Posledním je pak požadavek na samotné výsledky vyhledávání. Všechny požadavky jsou zaslány současně a posílají se při každé změně facetu. Jejich SPARQL query se však samozřejmě liší podle aktuálního stavu.

Tento přístup je velmi odlišný od Elasticsearch, kde se vše posílalo v jednom dotazu. Rozdělení požadavků na možnosti facetů a samotné výsledky však poskytuje několik výhod. Každý z facetů si tak může držet svůj stav a konstruovat si SPARQL query sám. Zároveň tak oddělíme načítání výsledků od facetů a vyhledávání tak může působit svižněji.

3.4 Srovnání přístupů z hlediska sémantických technologií

Způsobů jak implementovat facetové vyhledávání je mnoho, pro sémantická data je to však složitější. Většina řešení jako například zmíněný Elasticsearch nebo Solr nejdou pro sémantická data použít. Zbývají nám tedy pouze knihovny přímo pro sémantické facetové vyhledávání, těch je však minimum a jsou většinou staré, bez aktivního vývoje nebo závislé na platformě. To je případ i zmíněné SPARQL Faceter. Navrhnutí vlastního řešení se tak zdá být vhodné.

Jak jsme zmínili v podsekci 2.2.4, nejčastějším způsobem zpřístupnění sémantických dat je SPARQL endpoint. Je tedy logické, aby sémantické facetové vyhledávání také používalo SPARQL k získávání výsledků. S tím je pak ale nutné vyřešit jakým způsobem takovou SPARQL query sestavovat. A pokud má být naše řešení přenositelné, jak vůbec takové vyhledávání konfigurovat, aby mělo nějaké rozhraní, ale zároveň umožňovalo komplexní možnosti vyhledávání.

SPARQL Faceter toto řeší pomocí „šablony“ query, kterou v konfiguraci vytvoří uživatel. Tato šablona musí obsahovat klauzuli `<RESULT_SET>`, místo které pak knihovna doplní případné constraints facetů. Taková šablona může vypadat jako příklad níže [26].

```
SELECT * WHERE {
    <RESULT_SET>
    OPTIONAL {
        ?id rdfs:label ?name .
        FILTER(langMatches(lang(?name), "en"))
    }
}
```

To umožňuje uživateli stejně komplexní konfiguraci výsledků jako by ji zadával přímo na SPARQL endpoint.

Složitější je to však s konfigurací facetů. Nejjednodušším řešením je pouhé definování predikátu dle kterého bude facet filtrovat. Toto řešení používá i SPARQL Faceter. To je ovšem vhodné jen pro jednoduché facety, pro složitější už toto řešení aplikovat nejde. U složitějších je opět nutné skládat query, tentokrát z query pro výsledky a constraints ostatních facetů.

Kapitola 4
Návrh

Výsledkem této práce je sémantický facetový vyhledávač. Tato kapitola se bude zabývat jeho návrhem. Nejprve navrhneme architekturu vyhledávače, poté popíšeme jednotlivé části návrhu.

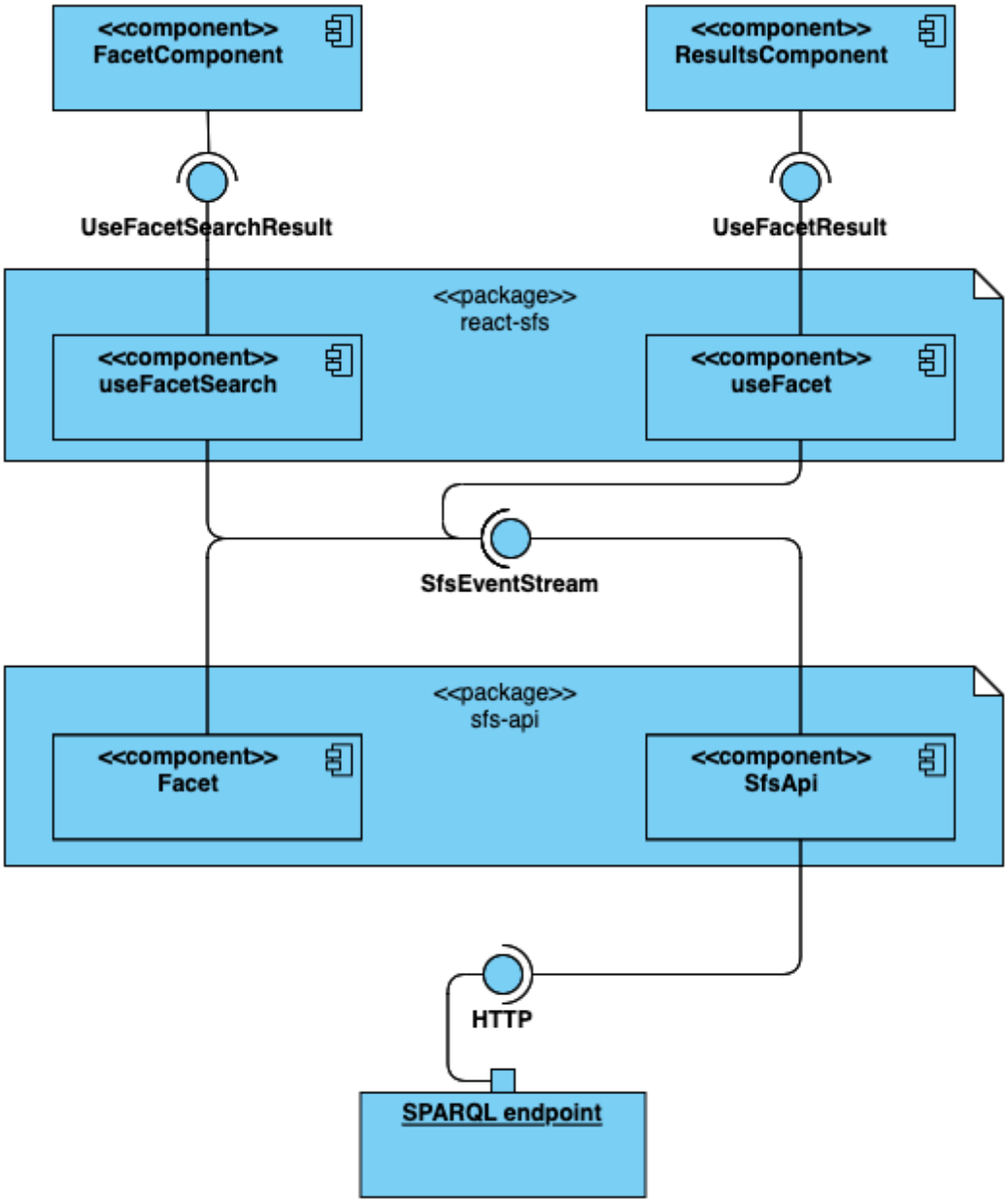
4.1 Architektura

Při návrhu architektury je nutné dbát na to, aby byl vyhledávač rozdělený na moduly vyhledávání a jeho vizualizace. Toto rozdělení nám umožní, aby byl modul vyhledávání zcela nezávislý na použité platformě pro vizualizaci. Realizovat takový modul jde pak dvěma způsoby - buďto jako samostatná serverová služba nebo jako modul bez využití konstruktů nějaké platformy. Jelikož lze získávat sémantická data skrze SPARQL endpoint, který lze dotazovat přímo z prohlížeče, je v našem případě vhodnější jít cestou modulu, který bude využíván přímo na frontendu.

Tento modul bude pojmenován *sfs-api* a bude koncipován tak, aby to byla samostatná knihovna, kterou lze využít v jakémkoliv Javascript frameworku či prostředí. Jelikož však chceme implementovat vyhledávač primárně ve frameworku React, vytvoříme ještě jeden balíček[1] jménem *react-sfs*, který bude ještě zjednodušovat rozhraní *sfs-api* pro React aplikace. Framework React jsme zvolili, jelikož byl původně součástí zadání práce, je to nejpopulárnější framework na tvorbu webových aplikací[27] a autor práce s ním má několikaleté zkušenosti.

Výsledkem tak budou dva balíčky *sfs-api* a *react-sfs*, které budou veřejně publikovány v repozitáři *npm*[2], který je primárním repozitářem Node.js balíčků. Jejich názvy jsou odvozené z SFS[3], což je název naší knihovny a budeme na ni tak v práci dále odkazovat. Architekturu implementace knihovny SFS ilustrujeme na diagramu komponentu na obrázku 4.1.

[1] Balíčkem je myšlen Node.js balíček definovaný souborem package.json.
[2] `https://www.npmjs.com`
[3] Zkratka pro „semantic faceted search“, česky sémantické facetové vyhledávání.



Obrázek 4.1. Diagram komponent sémantického facetového vyhledávače.

Abychom ukázali integraci knihovny SFS, vytvoříme projekt *sfs-react-demo*, který bude ukázkovým příkladem použití této knihovny. Zároveň bude naším sémantickým facetovým vyhledávačem, který budeme srovnávat v sekci 6.2. Na tento vyhledávač budeme dále v práci odkazovat jako na „vyhledávač SFS“.

Součástí publikovaných balíčků nebudou žádné komponenty či webové prvky, které by definovaly vizuální podobu samotných facetů či vyhledávání. Záměrně tuto část necháváme na uživateli knihovny, abychom ho nijak neomezovali a nevnucovali mu vizuální podobu. Může se však inspirovat komponenty v našem vyhledávači *sfs-react-demo*. Díky tomu se dá knihovna snadno integrovat do různých projektů.

4.2 sfs-api

V této sekci si popíšeme důležité části balíčku *sfs-api*. Tyto části jsou jádrem celé knihovny SFS.

4.2.1 Facet

Stav facetu lze definovat jako jeho aktuální možnosti a vybranou hodnotu z nich (nebo žádnou). Tento stav pro nás bude představovat abstraktní třída Facet. Jednotlivé typy facetů budou pak podtřídami třídy Facet. Tento design bude uživateli knihovny SFS umožňovat namodelovat si vlastní facet vytvořením vlastní podtřídy třídy Facet. To může být užitečné pro specifické typy facetů, které nepokryje knihovna SFS.

Typy facetu se liší hlavně tím, jak se skládají jejich dotazy k získání možností či to, jaké constraints utvářejí. Při vytváření vlastního facetu bude mít uživatel možnost napsání vlastní logiky pro tyto části.

```
interface FacetConfig {
  id: string,
  predicate: string,
  labelPredicates?: string[],
}
```

Konfigurace třídy Facet je reprezentována rozhraním FacetConfig, které můžeme vidět výše.

Součástí FacetConfig je id, které je primárním identifikátorem facetu a musí být unikátní v rámci jednoho SfsApi.

predicate je predikát v rámci RDF trojice použitých k dotazování možností facetu a následného filtrování dle ní. Toto může být IRI a můžeme využít i klauzule PREFIX definované v SfsApiConfig.

labelPredicates jsou pak predikáty, které chceme použít k najití názvů jednotlivých možností facetu. Jsou aplikovány v pořadí elementů v poli a pokud žádný nenajde název nebo labelPredicates nedefinujeme bude použito id možnosti, tedy celé její IRI.

Jednotlivé možnosti facetů jsou reprezentovány rozhraním FacetOption, které je popsáno níže.

```
interface FacetOption {
  value: string,
  label: string,
  count: number,
}
```

value je samotnou hodnotou možnosti, label jejím jménem, pod kterým se zobrazuje a count počet jejího výskytu.

4.2.2 SfsApi

Rozhraním celého facetového vyhledávání bude třída SfsApi. Ta bude mít přístup ke všem facetům a zároveň bude konfigurovat facetové vyhledávání jako celek skrze rozhraní níže.

```
interface SfsApiConfig {
  endpointUrl: string,
  baseQuery: string,
  facets: Facet[],
```

```
  language: string,
  prefixes?: Prefixes,
}
```

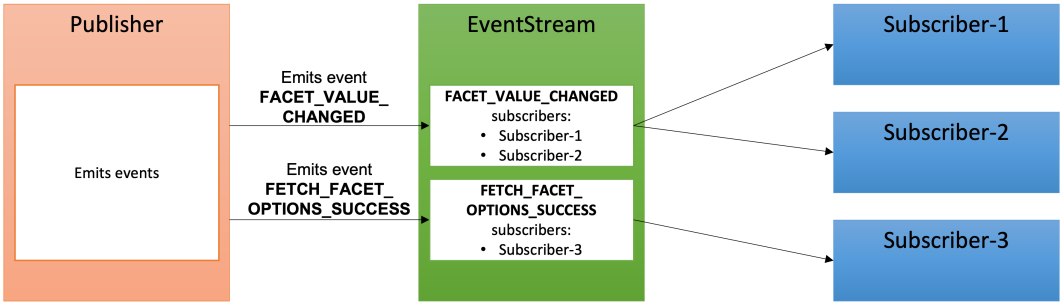
Důležitým atributem je baseQuery, který by měl obsahovat SPARQL query, která se používá k dotazování výsledků. Tato query je klíčová, protože se z ní skládají i query pro možnosti facetů. Skládání query funguje na podobném principu jako byl popsaný pro SPARQL Faceter v sekci 3.4, akorát baseQuery není šablonou, ale přímo query a nepotřebuje tak nějakou speciální klauzuli. Převede totiž baseQuery do objektu SPARQL query a všechny části query v klauzuli WHERE poskytuje jako constraints ostatním facetům.

Součástí této query musí být proměnné _id a _label. Proměnná _id představuje primární identifikátor jednotlivých výsledků a _label pak jejich jméno, kterým by měly být označeny. Na základě tohoto jména jsou pak i fitrovány výsledky v případě fulltextového vyhledávání. Všechny názvy interních SPARQL proměnných jsou prefixovány podtržítkem, tudíž by tak uživatel neměl nazývat jiné své proměnné.

Skrze SfsApi také můžeme iniciovat nové vyhledávání. To vyresetuje stav všech facetů. V rámci hledání také můžeme poskytnout searchPattern, což je fráze, kterou by měla obsahovat proměnná _label. Tato funkce představuje fulltextové vyhledávání zmíněné v sekci popisu facetového vyhledávání 3.1.

4.2.3 Eventy

Hlavní rozhraní pro komunikaci událostí, které nastanou v rámci facetového vyhledávání je třída EventStream. Ta využívá komunikační model Publish-Subscribe ilustrovaným na obrázku 4.2. V tomto modelu mohou vydavatelé (publisher) skrze zprostředkovatele (broker) posílat data na nějaké téma (topic) [28]. Odběratelé (subscriber) mohou toto téma odebírat a zprostředkovatel jim všechna data na to téma rozesílá.



Obrázek 4.2. Diagram komunikačního modelu Publish-Subscribe třídy EventStream[4].

V případě třídy EventStream je ona tím zprostředkovatelem. Vydavatelé a odběratelé jsou ostatní třídy Facet, SfsApi nebo cokoliv dalšího, co uživatel knihovny SFS přihlásí k odběru (např. komponenty facetů). Přihlašování probíhá skrze metodu on(eventType, callback). Parametr eventType je typem eventu, který chceme odebírat, callback pak funkce, která se vykoná když event nastane. Možné typy eventů jsou popsány níže.

```
type EventType =
  | "RESET_STATE"
  | "NEW_SEARCH"
```

[4] Inspirováno obrázkem z prezentace Stanislava Vítka [29].

```
  | "FACET_VALUE_CHANGED"
  | "FETCH_FACET_OPTIONS_PENDING"
  | "FETCH_FACET_OPTIONS_SUCCESS"
  | "FETCH_FACET_OPTIONS_ERROR"
  | "FETCH_RESULTS_PENDING"
  | "FETCH_RESULTS_SUCCESS"
  | "FETCH_RESULTS_ERROR";
```

Eventy v sobě mají další informace podle typu eventu, například FetchFacetOptionsSuccessEvent níže.

```
interface FetchFacetOptionsSuccessEvent {
  type: "FETCH_FACET_OPTIONS_SUCCESS",
  facetId: string,
  options: FacetOption[],
}
```

Díky odebírání těchto eventů může uživatelské rozhraní reagovat na stav facetů. Pro platformu React toto ještě zjednodušuje balíček *react-sfs*, který bude popsán v následují sekci 4.3.

4.3 react-sfs

Balíček *react-sfs* má zjednodušovat rozhraní pro použití knihovny SFS na platformě React. Poskytuje dvě funkce useFacet a useFacetSearch. Tento typ funkcí se ve světě platformy React nazývá „hook“ [30]. Funkce hook umožňují jednoduše sdílet části kódu v různých React komponentách.

Hook useFacet má poskytovat rozhraní pro komponenty jednotlivých facetů. Poskytuje možnosti facetů, aktuální hodnotu nebo stav požadavku na nové možnosti. Toto rozhraní je generické, jelikož hodnoty různých druhů facetů mohou mít různé typy a je popsáno níže.

```
type UseFacet<Value> = (facet: Facet<Value>) => UseFacetResult<Value>;

interface UseFacetResult<Value> {
  options: FacetOption[],
  value: Value | undefined,
  onValueChange: (newValue: Value) => void,
  isFetching: boolean,
  error: any,
}
```

useFacetSearch pak poskytuje podobné rozhraní pro výsledky facetového vyhledávání. Parametrem je tedy třída SfsApi.

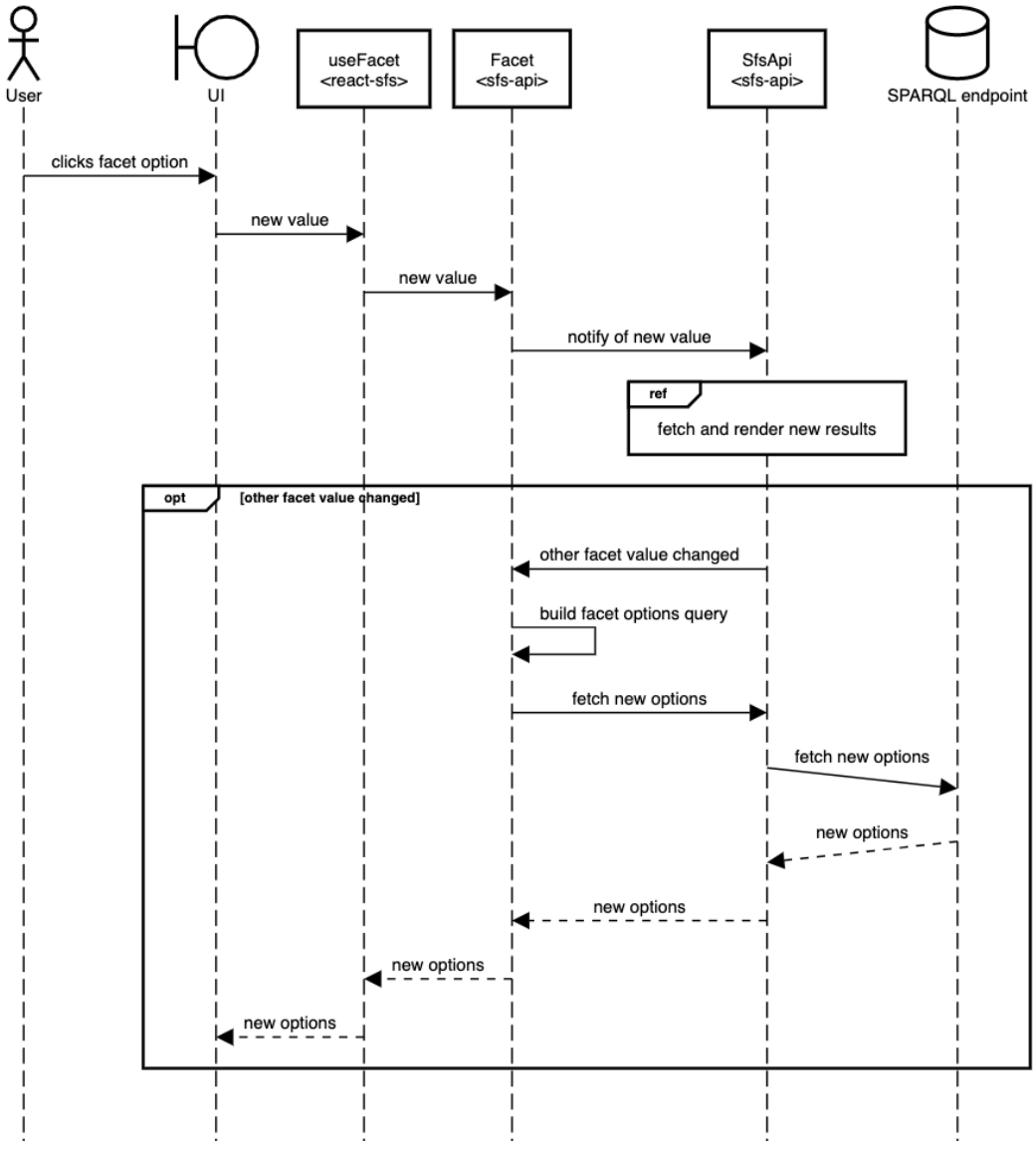
```
type UseFacetSearch = (sfsApi: SfsApi) => UseFacetSearchResult;

interface UseFacetSearchResult {
  results: Results | undefined,
  isFetching: boolean,
  lastSearchPattern: string,
  error: any,
}
```

Tato rozhraní byly inspirovány populárními knihovnami jako RTK Query [31] nebo SWR [32].

4.4 Přístup k facetovému vyhledávání

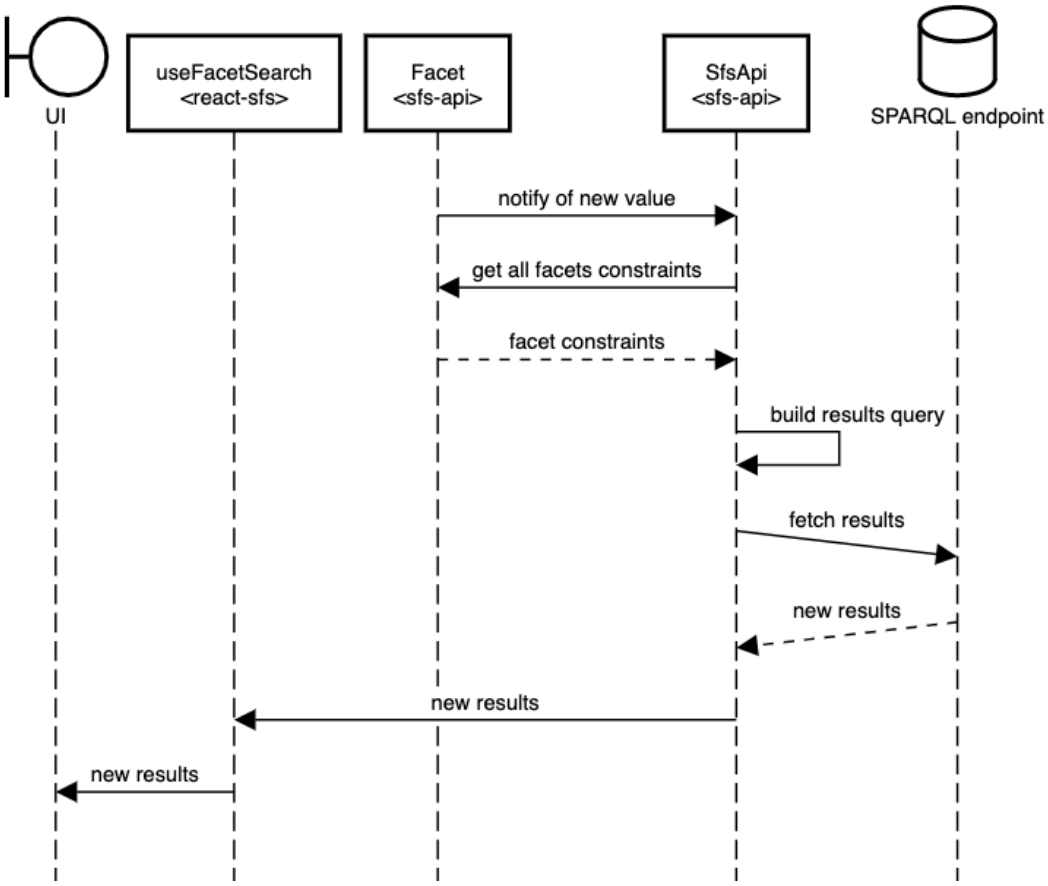
V sekci 3.3 jsme si zanalyzovali dva existující přístupy k facetovému vyhledávání. Náš přístup bude velmi podobný tomu druhému, tedy SPARQL Faceter.

Každý facet (v našem případě už tím myslíme přímo naší třídu `Facet`) si bude sám sestrojovat dle aktuálního stavu SPARQL query na získání jeho možností. Odesílat tento dotaz na SPARQL endpoint však bude vždy třída `SfsApi`. Ta doplní query facetu o constraints ostatních facetů, či omezení fulltextovým vyhledáváním, které je také součástí facetového vyhledávání. Přidá také klauzule `PREFIX`, které jsou definovány atributem prefixes. Tento proces je ilustrován na sekvenčním diagramu na obrázku 4.3.



Obrázek 4.3. Sekvenční diagram změny hodnoty facetu.

Dotaz na výsledky pak bude v režii SfsApi. Ta po změně hodnoty facetu zkonstruuje query na výsledky zakomponováním constraints od jednotlivých facetů. Požadavek s touto query odešle na SPARQL endpoint a po získání výsledků notifikuje zainteresované instance. To je opět znázorněno sekvenčním diagramem na dalším obrázku 4.4.



Obrázek 4.4. Sekvenční diagram získávání nových výsledků.

Kapitola 5
Implementace

V této kapitole si popíšeme implementaci návrhu z předešlé kapitoly 4. Zdůvodníme si některá rozhodnutí při vývoji a použité knihovny.

5.1 Použité knihovny

Při vývoji jsme se snažili, aby knihovny měly co nejméně závislostí, aby byly nenáročné na místo a neinstalovaly uživateli závislosti, které nepotřebuje nebo nechce. Zde si popíšeme využité knihovny. Všechny tři projekty jsou napsané v jazyce TypeScript, což je nadstavba jazyku JavaScript, která jej rozšiřuje o statické typování a předchází tak spoustě chyb.

Na balíček *sfs-api* jsme si vystačili pouze se třemi závislostmi. Využíváme knihovnu *fetch-sparql-endpoint*[1]. Ta nám zjednodušuje volání SPARQL endpointu a serializaci příchozích dat do formátu RDFJS. Vybrali jsme ji hlavně proto, že je skutečně jednoduchá, neimplementuje nepotřebné věci navíc a je udržována poměrně aktuální.

Balíček datového modelu RDFJS je druhou závislostí. *sfs-api* využívá ještě knihovnu *sparqljs*[2] ke parsování SPARQL dotazů do JavaScript objektů. Skrze tyto objekty můžeme pak query lépe upravovat.

Knihovna *react-sfs* má jako závislost pouze *sfs-api*.

Jako součást zadání této práce bylo rozhodnuto, že na implementaci vyhledávače bude použit JavaScript framework React. K vytvoření projektu vyhledávače, tedy *sfs-react-demo* jsme použili doporučenou metodu vytváření React projektů *create-react-app*. Ta vytvoří předpřipravený projekt s minimální konfigurací a potřebnými závislostmi.

Abychom si usnadnili vytváření UI vyhledávače, využili jsme populární knihovnu MUI (známou též jako Material-UI) a použili připravené komponenty z ní.

Za zmínku pak ještě stojí knihovna *react-virtuoso*, kterou využíváme na virtualizaci listů s výsledky a možnostmi facetů. Virtualizace listu znamená, že se vykresluje vždy jen část listu, která je zobrazená a ne ostatní položky, které se zrovna nezobrazují. List totiž může obsahovat velmi velké množství dat a vykreslování všech položek by mohlo znatelně zpomalit naší aplikaci.

5.2 Implementační detaily

Implementace probíhala poměrně podle plánu s minimálními odchylkami od návrhu. V této sekci si však zmíníme některé detaily, které jsou zajímavé a které byly třeba zpočátku vymyšleny jiným způsobem.

Původně bylo zamýšleno, že při konfiguraci budou `SfsApi` předány pouze objekty rozhraní `FacetConfig`, se specifikovaným typem facetu a `SfsApi` teprve při konstrukci

[1] `https://www.npmjs.com/package/fetch-sparql-endpoint`
[2] `https://www.npmjs.com/package/sparqljs`

vytvoří jednotlivé instance podtříd. Dobře by se na toto dal využít návrhový vzor Factory pattern [28]. To se však ukázalo jako těžko proveditelné, pokud chceme zachovat možnost implementovat si vlastní podtřídu Facet. Třída SfsApi by těžko věděla jakou třídu zkonstruovat pro custom facety. Lepší tedy pro nás bylo předat třídě SfsApi rovnou instance podtříd třídy Facet.

Dalším detailem, který se lehce proměnil bylo skládání constraints z SfsApi a ostatních facetů do query facetu. Zpočátku se tyto constraints přidávaly do query až v třídě SfsApi, nakonec se však vytvořila metoda v SfsApi pro získání constraints a skládání se tak přesunulo do samotných facetů. To umožňuje větší variabilitu pro implementaci vlastního facetu. Uživatel si může sám rozhodnout, jak chce naložit s aktuálními constraints ostatních facetů či celého facetového vyhledávání.

S constraints souvisí i další detail, který popíšeme. S částmi SPARQL query jsme chtěli co nejvíce pracovat jako s objekty dle typů ze *sparqljs*. Výhodami toho je menší chybovost díky typovosti těchto objektů a zároveň jasnější a srozumitelnější kód. Problém však nastal při implementování constraints facetu. Pokud totiž měl facet v FacetConfig jako predicate IRI, které bylo složeno z prefixu definovaným v SfsApiConfig, nastal problém jak tento constraints reprezentovat. Nejvhodnějším řešením se nakonec zdálo být používat pouze jeden Parser[3] k sestrojování objektů. Ten je v SfsApi a má tedy přístup k prefixům ze SfsApiConfig.

[3] Třída Parser je součástí knihovny *sparqjs* a vygeneruje JavaScript objekt z textové reprezentace SPARQL query s použitím předaných prefixů.

Kapitola 6
Vyhodnocení

Tato kapitola se bude zabývat vyhodnocením implementovaného vyhledávače z předchozích kapitol 4 a 5. Nejprve představíme testy, které jsme implementovali k ověření jeho funkčnosti. Pak srovnáme vyhledávač SFS s vyhledávačem, který jsme v sekci 3.3.2 nazvali jako prohlížeč MVČR. Na konci srovnání pak bude na jeho základě doporučeno či nedopručeno nahrazení prohlížeče MVČR vyhledáváčem SFS.

6.1 Testy

K otestování knihovny SFS jsme se rozhodli využít jednotkové testy. Tyto testy však prozatím testují pouze balíček *sfs-api*, jelikož je hlavní částí celého vyhledávání. Pro účely této práce se to zdá dostačující, ale do budoucna by bylo dobré zvýšit počet jednotkových testů a vytvořit testy i v projektech *react-sfs* a *sfs-react-demo*.

Testy SfsApi se prvně zaměřují na kontrolu konfigurace, jak můžeme vidět na příkladu níže.

```
it('should throw an error on baseQuery without _id variable', () => {
    expect(() => {
      new SfsApi({
        endpointUrl: "https://dbpedia.org/sparql",
        facets: [],
        baseQuery: "SELECT DISTINCT ?x ?_label WHERE {?x ?y ?_label}",
        language: "en"
    });
  }).toThrowError("SfsApi baseQuery has to SELECT ?_id variable. " +
   "Check documentation for more info.")
});
```

U facetů pak testujeme například správné emitování eventů.

```
it('emits FETCH_FACET_OPTIONS_SUCCESS event', done => {
  jest.spyOn(sfsApi, "fetchBindings")
     .mockImplementation(() => {
       const stream = new PassThrough();
       stream.end();
       return Promise.resolve(stream);
     });
   sfsApi.eventStream.on("FETCH_FACET_OPTIONS_SUCCESS", event => {
     done();
   });
   facet.refreshOptions();
});
```

6.2 Srovnání s existujícím vyhledávačem

V rámci zadání práce máme porovnat implementovaný vyhledávač s existujícím prohlížečem MVČR. Hlavní rozdíly těchto vyhledávačů jsou vypsány v tabulce 6.1 a dále popsány v této sekci.

	Prohlížeč MVČR	Vyhledávač SFS
Rychlost načtení stránky	70 sekund	3,9 sekund
Vlastní vizuální podoba	NE	ANO
Nezávislost na platformě	NE	ANO
Možnost implementace vlastního facetu	NE	ANO
Virtualizace listů	NE	ANO
Volba jazyku	ANO, ale jen výsledky	NE

Tabulka 6.1. Srovnání vyhledávačů.

Největší rozdíl mezi vyhledávači můžeme sledovat už při prvotním načtení stránky. Zatímco prohlížeč MVČR z 10 měření načetl výsledky průměrně za **70 sekund**, vyhledávač SFS je načetl průměrně za **3,9 sekund**. K tomuto je nutné říct, že je to především způsobeno množstvím přenesených dat. Prohlížeč MVČR totiž ukazuje více informací o výsledcích vyhledávání a tím pádem se do něj přenáší téměř 140 MB dat, zatímco do vyhledávače SFS zhruba 10.5 MB. Není tedy oprávněné toto brát jako výhodu naší implementace vyhledávače. Je ovšem k zamyšlení, zda je žádoucí takové množství informací při načtení stránky načítat, jelikož čekat déle než minutu na načtení výsledků je z pohledu uživatele velmi nevhodné.

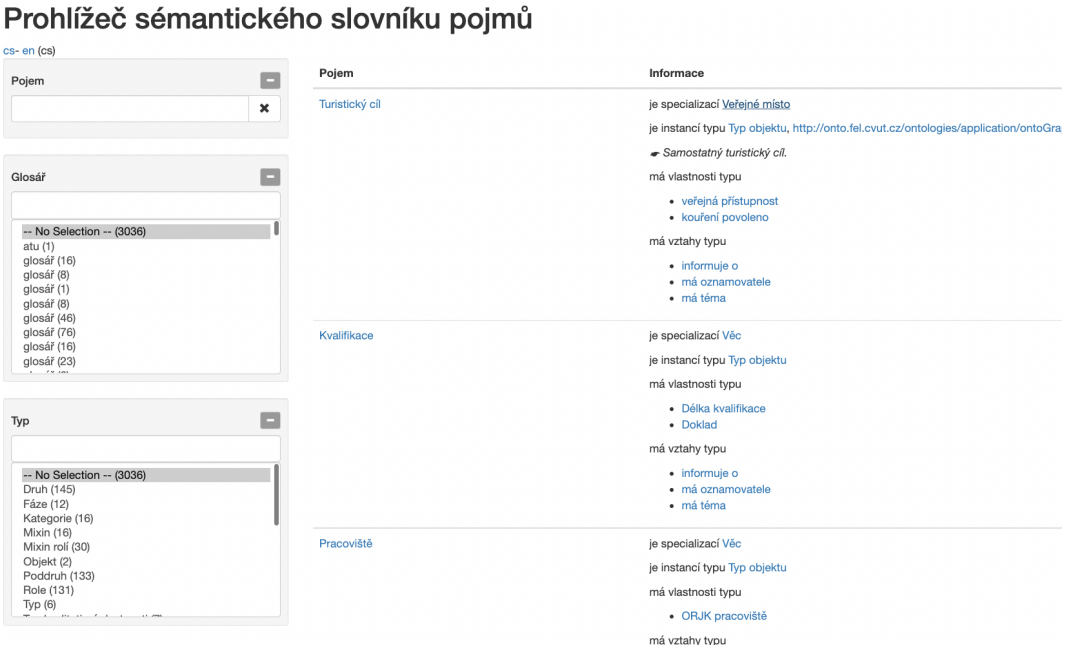
Rozdíly, ve kterých je však vyhledávač SFS rozhodně lepší jsou možnost vlastní vizuální podoby, nezávislost na platformě a možnost implementace vlastního facetu. Vizuální podoba prohlížeče MVČR je totiž omezena použitou knihovnou SPARQL Faceter, která je však už 6 let stará a na jejím vzhledu je to znát. Navíc je knihovna SPARQL Faceter navržena pouze pro platformu AngularJS a tudíž omezuje uživatele na použití této platformy. Jak vypadá prohlížeč MVČR si můžeme prohlédnout na obrázku 6.1 nebo na jeho adrese `https://slovník.gov.cz/prohlížeč`.

Vyhledávač SFS nechává vizuální podobu čistě na uživateli a dá se využít na kterékoliv JavaScript platformě, tudíž své uživatele nijak neomezuje. K tomu přispívá i možnost implementace vlastního facetu pro případy nepokryté knihovnou SFS. Vzhled vyhledávače SFS je na obrázku 6.2 nebo si lze lokálně spustit projekt *sfs-react-demo*[1].

Vyhledávač SFS je také lepší v tom, že využívá virtualizované listy pro výsledky i možnosti facetů a vyžaduje tak méně výkonu. Naopak však zatím nepodporuje volbu jazyka jako prohlížeč MVČR. U něho se tato možnost však aplikuje pouze na výsledky a navíc se nijak neukládá, tudíž je nutné při každém obnovení stránky znovu tuto možnost zvolit.

Pak je zde několik problematických částí prohlížeče MVČR, které v tabulce nejsou zmíněny. Jsou to například nicneříkající názvy glosářů v příslušném facetu, nesmyslné řazení výsledků a možností facetů nebo špatné stylování tabulky s výsledky, ze které není vidět celý sloupec s glosáři. I tyto části lze vidět na obrázku 6.1. Za zmínku stojí také otravné načítací kolečko výsledků, které vždy posune celou tabulku s výsledky pod sebe.

[1] Pokyny ke spuštění můžeme najít v příloze A.



Obrázek 6.1. Stránka prohlížeče MVČR.

Obrázek 6.2. Stránka vyhledávače SFS.

Na základě tohoto srovnání doporučuje autor této práce nahradit prohlížeč MVČR novým vyhledávačem za použití knihovny SFS. Nový vyhledávač by byl založený na vyhledávači použitým v tomto srovnání (tedy projekt *sfs-react-demo*), ale musel by nejspíše projít drobnými úpravami, aby plně splňoval představy správců stávajícího prohlížeče.

Kapitola 7
Závěr

V práci jsme se seznámili s pojmem sémantický web a popsali si jeho klíčové technologie. Detailněji jsme si ukázali především technologie k ukládání a dotazování dat.

Dále jsme si popsali facetové vyhledávání a zároveň zadefinovali jak by měl vypadat správný facet. Rozdělili jsme si typy facetů dle toho jak vypadají z pohledu uživatele. Následně jsme zrešeršovali dostupná řešení facetového vyhledávání a vybrali dvě, každé s jiným přístupem. Zaměřili jsme se také na srovnání přístupů k facetovému vyhledávání z hlediska využití sémantických dat, čímž jsme splnili první z cílů této práce.

S nabytými poznatky jsme se pustili do návrhu sémantického facetového vyhledávače. Návrh jsme pojali spíše jako návrh knihovny SFS za pomoci které pak lehce implementujeme samotný vyhledávač. Knihovnu jsme rozdělili na balíčky *sfs-api*, který je jádrem celé knihovny a *react-sfs*, který poskytuje funkce k jednoduché implementaci na platformě React. Oba balíčky jsme publikovali v repozitáři Node.js balíčků *npm*. Samotný vyhledávač je pak projekt *sfs-react-demo*, který tyto knihovny využívá a implementuje vizuální vrstvu vyhledávače. Tímto jsme splnili druhý a třetí cíl této práce.

Abychom ověřili funkčnost vyhledávače implementovali jsme několik testů. Hlavně jsme však vytvořený vyhledávač srovnali s již existujícím Prohlížečem sémantického slovníku pojmů udržovaným Ministerstvem Vnitra České Republiky (MVČR) ve spojení s FEL ČVUT. S ohledem na výsledky autor práce doporučil, aby byl starý prohlížeč nahrazen novým. Ten by používal knihovnu SFS a byl postaven na projektu *sfs-react-demo*. Tímto srovnáním jsme splnili poslední cíl této práce.

Ačkoliv vývoj sémantického webu v druhé polovině minulého desetiletí spíše stagnoval a nepodařilo se mu možná rozšířit tak, jak si jeho autoři přáli, v posledních letech se to mění. S průlomy v posledních letech v oblasti umělé inteligence či internetu věcí se opět o sémantickém webu mluví jako o jedné z dalších klíčových technologií ve vývoji webu a internetu. Tyto oblasti mají totiž pro sémantický web spoustu využití a je tudíž možné, že v následujících letech opět zažije výrazný vývoj. Jestli se predikce potvrdí ukáže čas, ale je možné, že i tato práce pak bude využívána v nově vzniklých aplikacích jako řešení sémantického facetového vyhledávání a pomůže tak adopci sémantického webu.

7.1 Plány do budoucna

Aby měla knihovna SFS větší předpoklady k tomu být používaná jako primární řešení facetového vyhledávání pro sémantická data, je třeba doimplementovat ještě několik funkcí.

V první řadě to je stránkování. Virtualizované listy sice fungují dobře, ale ve velkém množství výsledků se lépe orientuje a pohybuje pomocí stránek. Navíc by to bylo

výhodnější z hlediska rychlosti a vytížení SPARQL endpointu, jelikož bychom mohli dotazovat jen část výsledků pro danou stránku.

S tím souvisí i přidání URL parametrů. V těch by totiž mohly být, kromě čísla aktuální stránky, také zvolené hodnoty facetů a dotaz na vyhledávání. To by nám umožnilo vytvářet odkazy na stav vyhledáváče a jeho výsledků.

Určitě je také potřeba přidat podporu více typů facetů, jako například range facet či date facet. Jelikož je celý projekt open source, mohly by být tyto funkce implementovány už za pomoci dalších členů open source komunity.

Literatura

[1] Berners-Lee, Sir Tim, James Hendler a Ora Lasilla. The Semantic Web. *Scientific American*. Scientific American, a division of Nature America, Inc., 2001, ročník 2001, č. Vol. 284. No. 5, s. 34-43. Dostupné na DOI https://www.jstor.org/stable/10.2307/26059207.

[2] *Semantic University*. Dostupné na `https://cambridgesemantics.com/blog/semantic-university/intro-semantic-web`.

[3] *About W3C*. Dostupné na `https://www.w3.org/Consortium`.

[4] *The Security of the Semantic Web - Secrecy, Trust and Rationality*. Dostupné na `https://www.w3.org/People/n-shiraishi/work/Security-of-RDF.html`.

[5] *RDF 1.1 Concepts and Abstract Syntax*. Dostupné na `https://www.w3.org/TR/rdf11-concepts`.

[6] *Understanding Linked Data Formats*. Dostupné na `https://rdf.community/understanding-linked-data-formats`.

[7] *Uniform Resource Identifier (URI) definition*. Dostupné na `https://www.techtarget.com/whatis/definition/URI-Uniform-Resource-Identifier`.

[8] *RDF 1.1 XML Syntax*. Dostupné na `https://www.w3.org/TR/rdf-syntax-grammar`.

[9] *RDF 1.1 Turtle*. Dostupné na `https://www.w3.org/TR/turtle`.

[10] *RDF/JS: Data model specification*. Dostupné na `https://rdf.js.org/data-model-spec`.

[11] *RDF Schema 1.1*. Dostupné na `https://www.w3.org/TR/rdf-schema`.

[12] *OWL 2 Web Ontology Language Document Overview (Second Edition)*. Dostupné na `https://www.w3.org/TR/owl2-overview`.

[13] *Linked Data*. Dostupné na `https://www.w3.org/DesignIssues/LinkedData.html`.

[14] *LinkedData*. Dostupné na `https://www.w3.org/wiki/LinkedData`.

[15] *About DBpedia*. Dostupné na `https://www.dbpedia.org/about`.

[16] *SPARQL 1.1 Query Language*. Dostupné na `https://www.w3.org/TR/sparql11-query`.

[17] Rakhmawati, Nur Aini, Jurgen Umbrich, Marcel Karnstedt, Ali Hasnain a Michael Hausenblas. A Comparison of Federation over SPARQL Endpoints Frameworks. In: *Knowledge Engineering and the Semantic Web*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. s. 132–146. Communications in Computer and Information Science.

[18] *Virutoso OpenLink*. Dostupné na `https://virtuoso.openlinksw.com`.

[19] *What is faceted search*. Dostupné na `https://stackoverflow.com/questions/5321595/what-is-faceted-search`.

[20] *What is Full-Text Search and How Does it Work?* Dostupné na https://www.mongodb.com/basics/full-text-search.

[21] *The State of Ecommerce Search*. Dostupné na https://www.nngroup.com/articles/state-ecommerce-search.

[22] *Elasticsearch*. Dostupné na https://www.elastic.co/elasticsearch.

[23] *Search API facets*. Dostupné na https://www.elastic.co/guide/en/app-search/current/facets.html.

[24] *Semantic Computing Research Group*. Dostupné na https://seco.cs.aalto.fi.

[25] *SPARQL Faceter - Client-Side Faceted Search Using SPARQL*. Dostupné na http://semanticcomputing.github.io/angular-semantic-faceted-search.

[26] *FacetResultHandler*. Dostupné na http://semanticcomputing.github.io/angular-semantic-faceted-search/#/api/seco.facetedSearch.FacetResultHandler.

[27] *Most used web frameworks among developers worldwide, as of 2021*. Dostupné na https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web.

[28] Gamma, Erich. *Design patterns*. 1st edition vyd. Boston: Addison-Wesley, 1995. ISBN 0-201-63361-2.

[29] *Architektury IoT systémů, databáze*. Dostupné na https://cw.fel.cvut.cz/wiki/_media/courses/b0b37nsi/lectures/nsi-lec-04.pdf#page=4.

[30] *Introducing Hooks*. Dostupné na https://reactjs.org/docs/hooks-intro.html.

[31] *React Hooks API*. Dostupné na https://redux-toolkit.js.org/rtk-query/api/created-api/hooks#usequery.

[32] *API Options*. Dostupné na https://swr.vercel.app/docs/options.

Příloha A
Elektronická příloha práce

Elektronickou přílohu s názvem *Daniel-Bourek-elektronicka-priloha.txt* lze nálezt společně s touto prací v Digitální knihovně ČVUT na adrese `https://dspace.cvut.cz/`. Zároveň je její obsah uveden níže v této příloze.

A.1 Odkazy

- Repozitář s projektem sfs-api
  `https://github.com/bouredan/sfs-api`
- Repozitář s projektem react-sfs
  `https://github.com/bouredan/react-sfs`
- Repozitář s projektem sfs-react-demo
  `https://github.com/bouredan/sfs-react-demo`

A.2 Pokyny ke spuštění

1. Naklonujte si git repozitář s sfs-react-demo pomocí příkazu „git clone git@github.com:bouredan/sfs-react-demo.git“.
2. V adresáři s naklonovaným repozitáři nainstalujte projekt pomocí příkazu „yarn install“ nebo „npm install“.
3. Spusťte si lokální verzi projektu pomocí příkazu „yarn start“ nebo „npm start“.
4. V prohlížečí si otevřete `http://localhost:3000`, kde najdete spuštěný projekt.

Příloha B
Slovníček

API	Application programming interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICT	Informační a komunikační technologie
IRI	Internationalized Resource Identifier
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
UI	uživatelské rozhraní
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WWW	World Wide Web
W3C	World Wide Web Consortium
XML	Extensible Markup Language