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.

Draft: 12. 5. 2022

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

Draft: 12. 5. 2022

Poděkování / Prohlášení

TODO

Čestně prohlašuji, že jsem předloženou práci vypracoval samostatně a, že jsem uvedl všechnu použitou literaturu.

Daniel Bourek V Praze, 15. 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 na platformě React. Celý proces je řádně zdokumentován. Na závěr je vyhledávač zevaluován 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 on the React platform. 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 /

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, web funguje už 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 data 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 velmi 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 ty existující mají své nedokonalosti. Často jsou závislé 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. Vyhledávač tak musí umožňovat rozdělení vyhledávání a jeho vizualizace do samostatných modulů.

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 poprvé veřejnosti předvedena 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 vyhovoval jejím časovým podmínkám, byl blízko domu či byl pokryt její pojišťovnou. Klíčové je zde to, že to zařízení zvládlo jen za pomocí webu, díky strojově čitelným standardizovaným datům na webových stránkách.

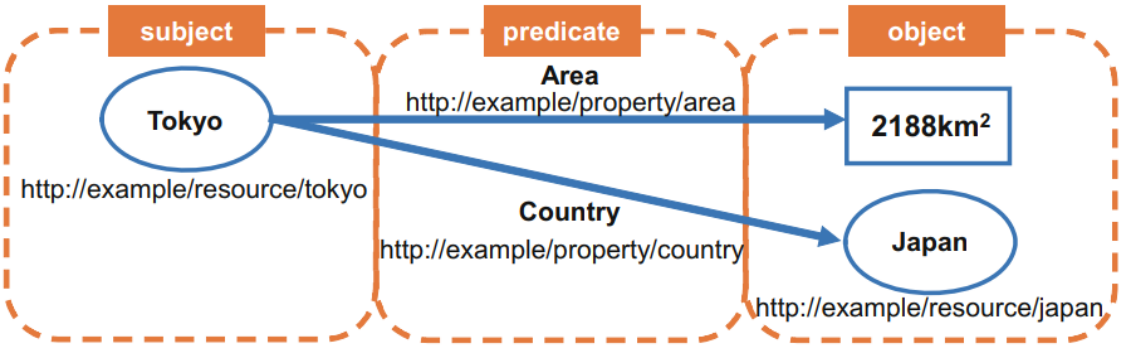
Sémantický web se tak má stát novým evolučním stupněm stávajícího webu (velmi často je nazýván také jako Web 3.0, ale fakticky je spíše jeho částí), kde jsou informace uloženy podle standardizovaných pravidel, což usnadňuje jejich vyhledávání a zpracování [2]. Ona standardizovaná pravidla jsou hlavně Resource Description Framework (RDF) a Web Ontology Language (OWL). Ty byly vyvinuty mezinárodním konsorciem W3C, které ve společnosti s veřejností vyvíjí i jiné webové standardy, pomocí nichž, chtějí rozvinout web do plného potenciálu. Pro ověření pravosti dokumentů (a jejich informací) využívá sémantický web také třeba digitální podpisy a šifrování [3].

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.

2.2.1 RDF

V sémantickém světě je standardem pro vytváření dat formát RDF[4]. RDF je standardizovaný strojově čitelný grafový formát, ve kterém se využívají tzv. triples, česky trojice, k popsání relací ve formátu subjekt - predikát - objekt. Tyto trojice lze znázornit jako orientované hrany v grafu.



Obrázek 2.1. Ukázka RDF trojice, kde je každá entita identifikovaná svou URI [5].

Jednotlivé uzly v grafu jsou pak identifikované pomocí IRI, což je nadmnožinou URI, která povoluje více Unicode znaků. IRI, potažmo URI, může pak vypadat například takto:

```
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Class>
```

Samotná syntaxe RDF definovaná není, nejčastěji se však používá RDF/XML, která dokáže zapsat RDF graf jako XML dokument či Turtle, 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 [6].

2.2.2 OWL 2

Pro popsání základních ontologií vzniklo RDF Schema (RDFS), které obsahuje sadu základních tříd k použití [7]. Později se však vyvinul Web Ontology Language (OWL), který je mnohem bohatší a používá se tak pro popis informací o věcech a vztahů mezi nimi neboli ontologií [8]. Oba jazyky jsou, stejně jako ostatní standardy sémantického webu, vytvořeny W3C.

V informatice se ontologií rozumí explicitní a formalizovaný popis určité problematiky. Datový model se sestává:

- Entita (objekt, jedinec, instance) je základní stavební prvek datového modelu ontologie. Entita může být konkrétní (člověk, tabulka, molekula) nebo abstraktní (číslo, pojem, událost).
- Kategorie (třída) je množina entit určitého typu. Podmnožinou kategorie je podkategorie. Kategorie může obsahovat zároveň entity i podkategorie.
- Atribut popisuje určitou vlastnost, charakteristiku či parametr entity. Každý atribut určité entity obsahuje přinejmenším název a hodnotu. Atribut je určen pro uložení určité informace vztahující se k dané entitě.
- Vazba je jednosměrné nebo obousměrné propojení dvou entit. Je možné říci, že vazba je určitým typem atributu, jehož hodnotou je jiná entita v ontologii.

2.2.3 Linked Data

Výše popsané technologie umožňují vznik standardizovaným strukturovaným datům, které nazýváme Linked Data. Ty by se měly navrhovat dle 4 principů:[9]

- 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 umožňovat používat standardy k reprezentaci a přístupu k datům na internetu.[10] 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 pak později definoval hodnocení kvality Linked Data od 1 do 5 hvězdiček.[9] Pro splnění stupně hodnocení musí data splňovat i předchozí stupně. Toto hodnocení je definované takto:

- 1 hvězdička - Data jsou dostupné na webu (v jakýmkoliv formátu), ale s otevřenou licencí, jako Open Data.
- 2 hvězdičky - Data jsou k dispozici jako strojově čitelná strukturovaná data (např. excel místo skenování obrázku tabulky)
- 3 hvězdičky - Data jsou v nechráněném formátu (např. CSV místo excelu).
- 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[11] - obdoba Wikipedia.org se sémantickými daty či GeoNames, popisující geografii na zeměkouli.

[sparql]
2.2.4 SPARQL

Primárním dotazovacím jazykem pro RDF je SPARQL [12]. Ten je syntaxí velmi podobný SQL, funguje však spíše na porovnávání a dosazování RDF trojic - tedy potažmo orientovaných hran grafu. Dotaz se pak skládá z množiny těchto trojic (a dalších klauzulí), 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 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  .
}
```

Jak můžeme vidět na příkladu SPARQL dotazu výše, trojice musí být odděleny pomocí tečky či středníku, přičemž oddělení středníkem je pouze „syntaktický cukr“[1] pro použití stejného subjektu z předchozí trojice.

Je zvykem, že stránky se sémantickými daty obsahují SPARQL endpoint, kam lze zadávat SPARQL dotazy. Často se využívá řešení od firmy OpenLink jménem Virtuoso[13].

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

Kapitola 3
Facetové vyhledávání

V této kapitole si nejprve 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.

popis-facetoveho-vyhledavani]

3.1 Popis

Obrázek 3.1. Ukázka facetů s vysvětlivkami [14]

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], aby uživateli umožnilo jeho dotaz dále upřesnit. 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 [16].

Facety bývají často znázorněny webovými elementy select či checkbox nebo také posuvníky. Omezením, podle kterých facet filtruje výsledky a možnosti ostatních facetů po výběru hodnoty budeme v této práci nazývat „constraints“.

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

[1] Fulltextové vyhledávání je technika hledání textu dle zadání uživatele v elektronických dokumentech a databázích[15].

- 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 omezeními výsledků dle hodnot facetů platí logický AND, tzn. aby se výsledek objevil v sadě výsledku, 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]

[typy-facetu]

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 Bucket facet

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

3.2.5 Text facet

Podobný fulltextovému vyhledávání. Filtruje výsledky dle zadaného textu.

stupu-k-facetovemu-vyhledavani]

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. Tyto řešení byly do práce vybrány jako ukázkové příklady po rešerši dostupných knihoven pro facetové 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[17]. Má mnoho funkcí a možností, nás však zajímá pouze je 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[18]. 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.

```
{
  ## 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
          }
        ]
      }
    ]
  }
}
```

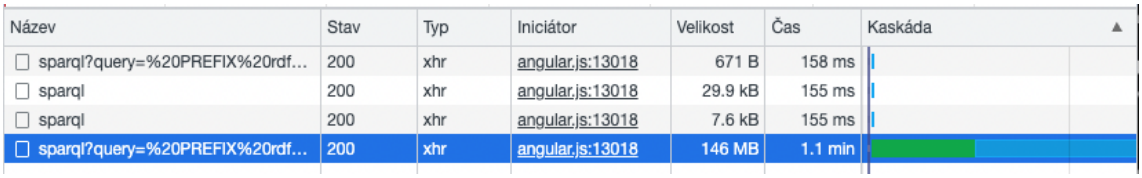

Zde pak máme odpověď na požadavek. Možnosti facetů jsou strukturovány vždy jako hodnota a počet výskytů.

[sparql-faceter]

3.3.2 SPARQL Faceter

Dalším příkladem přístupu k facetovému vyhledávání je modul na sémantické facetové vyhledávání od výzkumné skupiny Semantic Computing (SeCo)[19] s názvem SPARQL Faceter[20]. Balíček tohoto modulu se jmenuje *angular-semantic-faceted-search* 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 porovnávat i vyhledávač vzniklý v rámci této práce. Porovnání 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	

ec-mvcr-network-requests]

Obrázek 3.2. Asynchronní požadavky prohlížeče MVČR při načtení stránky

Obrázek 3.2 ukazuje asynchronní požadavky, které se odešlou při načítání stránky prohlížeče MVČR. Z analýzy asynchronních požadavků můžeme vidět, že SPARQL Faceter posílá požadavek na možnosti facetu pro každý facet zvlášť. Požadavek na výsledky se pošle také zvlášť. To je velký rozdíl oproti Elasticsearch, kde se dotaz na možnosti facetů posílal společně s dotazem na výsledky. Zároveň se posílá ještě jeden požadavek na zjištění počtu všech výsledků, který se zobrazuje u facetů pro

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

žádnou nevybranou možnost. Všechny tyto čtyři požadavky se posílají při každé změně facetu. Jejich SPARQL query se však samozřejmě liší podle aktuálního stavu.

3.4 Srovnání přístupů z hlediska sémantických technologií

TODO moc nevím jak tuto sekci pojmout

[navrh]

Kapitola 4
Návrh

Výsledkem této práce je sémantický facetový vyhledávač na platformě React. 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. Realizovat takový modul jde pak dvěma způsoby - buďto jako samostatná serverová služba nebo jako modul bez využití konstruktů použité platformy, v našem případě React. Jelikož lze získávat sémantická data skrze SPARQL endpoint, který lze dotazovat volat přímo z prohlížeče, je v našem případě vhodnější jít cestou modulu, který bude využíván přímo na frontend.

Tento modul bude pojmenován *sfs-api* a bude koncipován tak, aby to bylo samostatná knihovna, kterou lze využít v jakémkoliv Javascript frameworku či prostředí. Jelikož však chceme implementovat vyhledávač primárně na platformě React, vytvoříme ještě jeden balíček[1] jménem *react-sfs*, která bude ještě zjednodušovat rozhraní *sfs-api* pro React aplikace. Výsledkem tak budou 3 projekty, balíčky *sfs-api* a *react-sfs* a finální projekt *sfs-react-demo*, který bude naším sémantickým facetovým vyhledávačem a zároveň také ukázkovým příkladem použití těchto balíčků. Balíčky *sfs-api* a *react-sfs* 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.

Součástí publikovaných balíčku nejsou 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 snadně integrovat do různých projektů.

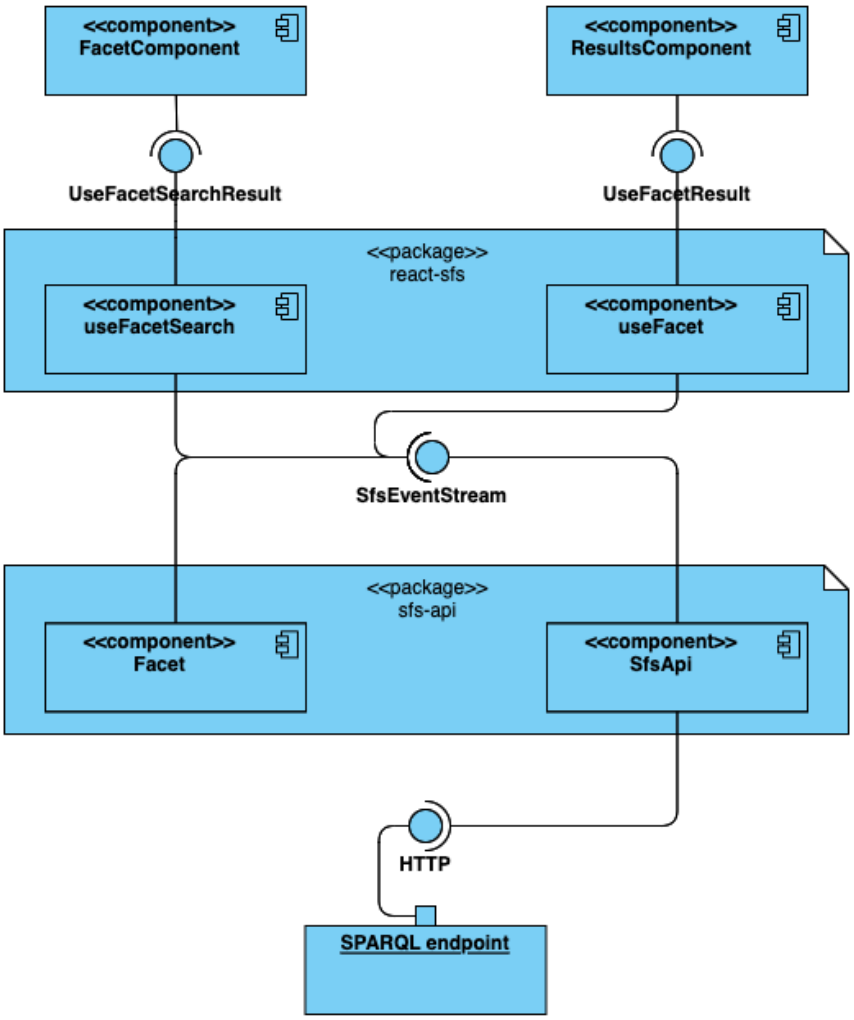
4.2 sfs-api

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

[facet]
4.2.1 Facet

Stav facetu lze definovat jako jeho aktuální možnosti a vybraná hodnota. Tento stav pro nás bude představovat abstraktní třída `Facet`. Jednotlivé typy facetů budou

[1] Balíčkem je myšlen Node.js balíček definovaný 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

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.

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.

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,
}
```

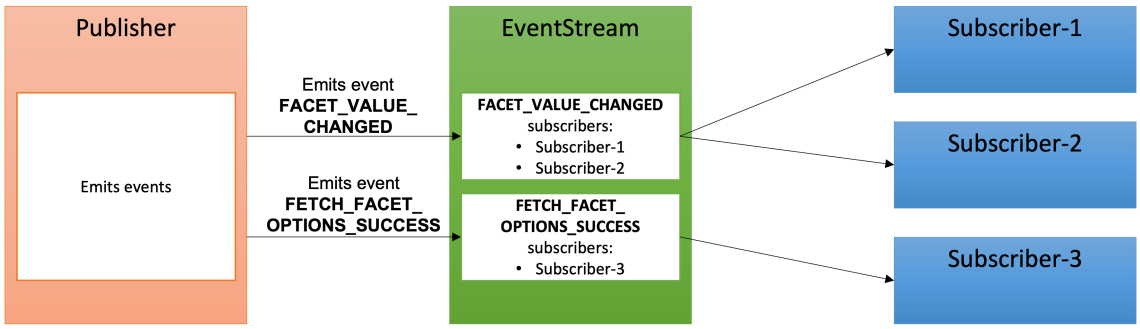
Důležitým atributem je baseQuery, který by měl obsahovat SPARQL query, která se používá k dotazování výsledků. Tato query je klíčová, protože se z ní skládají i query pro možnosti facetů.

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 události, které nastanou v rámci facetového vyhledávání je třída EventStream. Ta využívá komunikační model Publish-Subscribe. V tomto modelu mohou vydavatelé (publisher) skrze zprostředkovatele (broker) posílat data na nějaké téma (topic). Odběratelé (subscriber) mohou toto téma odebírat a zprostředkovatel jim všechny data na to téma rozesílá.



Obrázek 4.2. Diagram znázorňující komunikační model Publish-Subscribe třídy EventStream[4]

[4] Inspirováno obrázkem z prezentace Stanislava Vítka[21].

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.

```
type EventType =
  | "RESET_STATE"
  | "NEW_SEARCH"
  | "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 odebíráním těchto eventů může uživatelské rozhraní na stav facetů. Pro platformu React toto ještě zjednodušuje balíček *react-sfs*, který bude popsán v následují sekci 4.3.

[navrh-react-sfs]

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“[22]. 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.

```
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 tedy třída SfsApi.

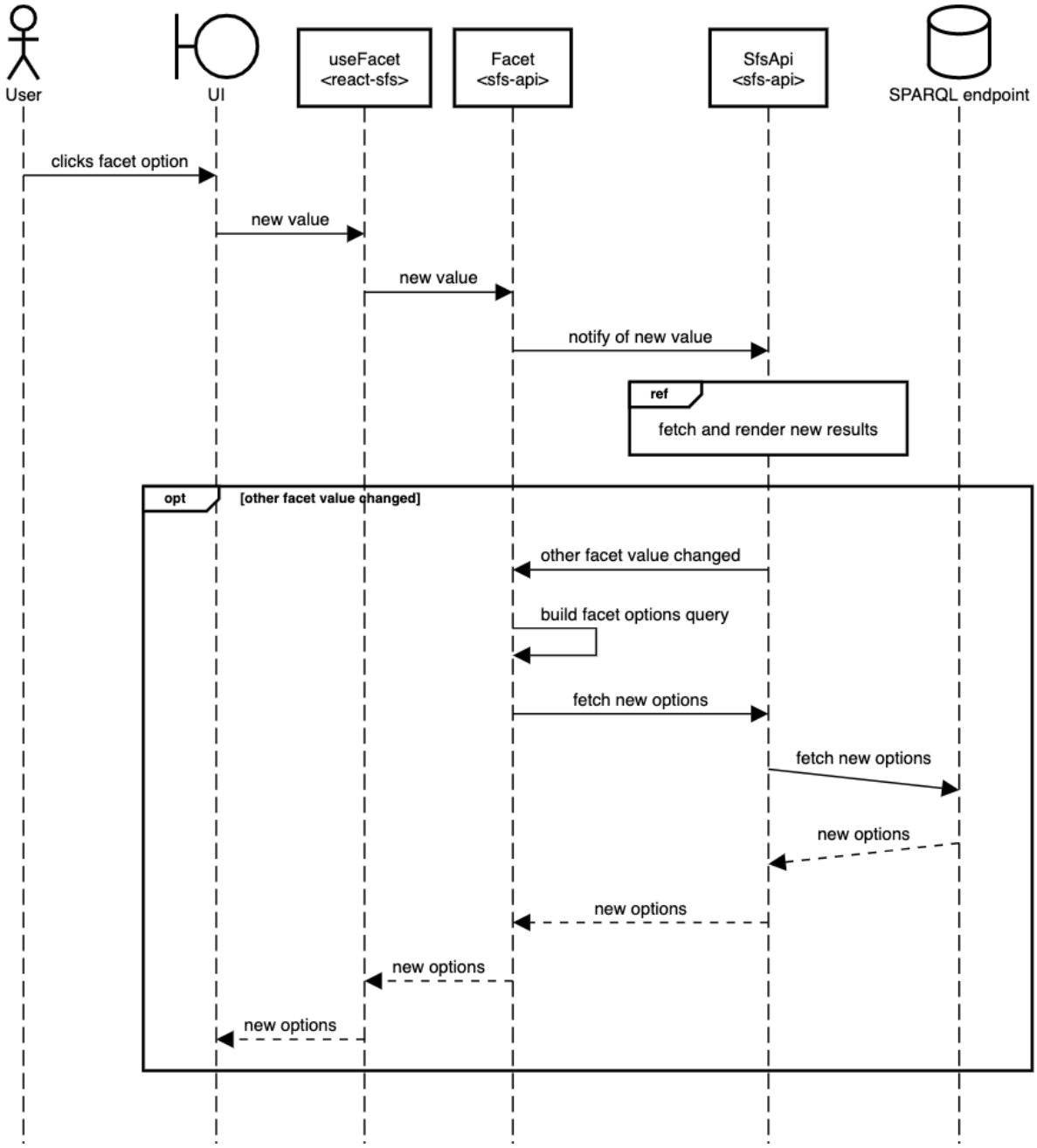
```
type UseFacetSearch = (sfsApi: SfsApi) => UseFacetSearchResult;

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

Tyto rozhraní bylo inspirováno populárními knihovnami jako RTK Query[23] nebo SWR[24].

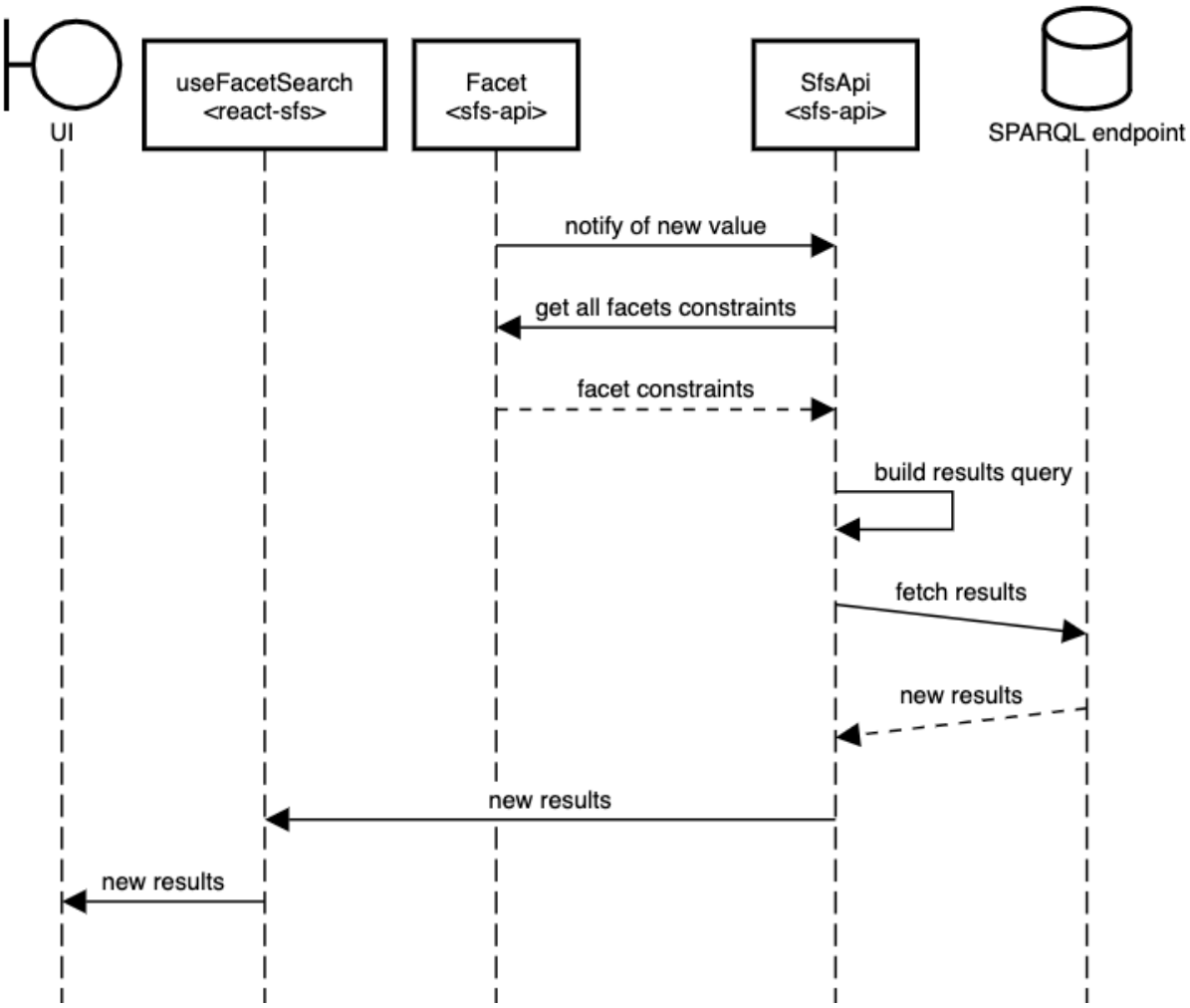
4.4 Přístup k facetovému vyhledávání

V sekci 3.3 jsme si zanalyzovali dva existující přístupy k facetovému vyhledávání. Náš přístup bude velmi podobný tomu druhému, tedy SPARQL Faceter[5]. Každý facet (v našem případě už tím můžeme myslet 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.

[5] To nemusí být úplně překvapivé, jelikož je to také knihovna na sémantické facetové vyhledávání.



Obrázek 4.3. Sekvenční diagram změny hodnoty facetu



Obrázek 4.4. Sekvenční diagram získávání nových výsledků

[implementace]

Kapitola 5
Implementace

V této kapitole si popíšeme implementaci návrhu z předešlé kapitoly 4. Zdůvodníme si některé 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 jazyku TypeScriptu, což je nadstavba jazyku JavaScript, která jej rozšiřuje o statické typování a předchází tak spoustě chyb.

Na balíček *sfs-api* jsme si vystačili pouze s 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 nejužitečnějšími 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 renderuje vždy jen část listu, která je zobrazená a ne ostatní položky, které se zrovna nezobrazují. List totiž může obsahovat velmi velké množství dat a renderování 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 *Factory pattern*. 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 objektami 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 zabývá vyhodnocením implementovaného vyhledávače z předchozích kapitol 4 a 5. Nejdříve představíme testy, které jsme implementovali k ověření jeho funkčnosti. Hlavní sekcí však bude srovnání s existujícím vyhledávačem, který jsme v sekci 3.3.2 nazvali jako prohlížeč MVČR[1]. Poté představíme plány do budoucna.

[testy]
6.1 Testy

TODO až budou hotové testy v sfs-api

ani-s-existujicim-vyhledavacem]
6.2 Srovnání s existujícím vyhledávačem

V rámci zadání práce máme porovnat implementovaný vyhledávač s existujícím prohlížečem MVČR. To bude náplní této sekce. Dle výsledků srovnání se také zamyslíme nad doporučením nahrazení stávajícího prohlížeče naším novým vyhledávačem.

TODO zmínit rychlost načítání, virtualizace, oddělené načítání, jazyk, nezávislost na vizuální podobě, možnost implementace vlastního facetu, lepší labely

[plany-do-budoucna]
6.3 Plány do budoucna

Ačkoliv je vyhledávač hotový a funguje, dle předcházejících sekcí 6.1 a 6.2, dobře, stále je několik funkcí, které by bylo dobré v budoucnu implementovat.

V první řadě to je stránkování. Virtualizované listy sice fungují dobře, ale stránkování by mohlo lepší na navigaci ve velkém množství výsledků. Dále by bylo příhodné přidat do URL parametrů zvolené hodnoty facetů a dotaz na vyhledávání . To by nám umožnilo vytvářet odkazy na stav vyhledáváče a tedy i jeho výsledků.

Další věci ke zlepšení do budoucna jsou spíše vývojářské. Například třída `CheckboxFacet` z *sfs-api* by mohla být generická místo definovaného typu hodnot `string[]` nebo by se dalo vylepšit rozhraní *sfs-api* skrz třídu `EventStream`. Při používání třídy `EventStream` totiž může dojít k tomu, že se při více asynchronních požadavcích najednou změní pořadí jejich odpovědí a komponenty by mohly ukazovat špatný stav.

[1] `https://slovník.gov.cz/prohlížeč`

Kapitola 7
Závěr

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

Poté 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 jejich srovnání 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 pomocí 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 srovnání jsme doporučili, aby byl starý vyhledávač nahrazen po domluvě a drobných úpravách nově vzniklým. Zároveň jsme tímto srovnáním splnili poslední cíl této práce.

Jelikož se dle výsledků vyhodnocení zdá, že je knihovna SFS navržena dobře, mohla by se začít používat jako primární řešení facetového vyhledávání pro sémantická data. K tomu je však nejspíše potřeba ještě doimplementovat další funkce, které jsme popsali v sekci 6.3. Jelikož je celý projekt open source, mohlo by se tak stát už za pomoci dalších členů open source komunity.

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 však mění. S průlomy v posledních letech v oblasti umělé inteligence či internetu věcí se opět o sémantickém webu mluví jako o jedné z přicházejících klíčových technologií ve vývoji webu a internetu a šlo by tak očekávat jeho výrazný vývoj přímo v následující letech. 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 rozšíření sémantického webu.

Literatura

ic-american-article]

[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.

cambridge-semantics]

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

mantic-web-security]

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

[w3c-rdf]

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

[rdf-community]

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

[rdfjs-data-model]

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

[w3c-rdfs]

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

[w3c-owl]

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

[w3c-linked-data]

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

3c-linked-data-wiki]

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

[dbpedia]

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

[w3c-sparql]

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

[openlink-virtuoso]

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

ets-explained-image]

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

extove-vyhledavani]

[15] *Fulltextové vyhledávání*. Dostupné na `https://cs.wikipedia.org/wiki/Fulltextové_vyhledávání`.

[nng-study]

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

[elasticsearch]

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

ticsearch-facet-api]

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

[seco]

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

sparql-faceter-docs]

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

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

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

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

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

Příloha A
Elektronická příloha práce

Elektronickou přílohu s názvem *Daniel-Bourek-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

Requests for correction

[rfc-1] možná udělat vlastní obrázek

[rfc-2] možná rozšířit

[rfc-3] u všech listingů chybí labely, nevím jak je přidat

[rfc-4] tohle se mi rozlejzá na dvě stránky a nevím jak to elegantně udržet pohromadě

[rfc-5] je ok začít sekci obrázkem?

[rfc-6] je ok mit zdroj na fulltextové vyhledávání z wikipedie?

[rfc-7] jak tuto sekci pojmout?

[rfc-8] míchám v celé kapitole budoucí čas s přítomným, je to ok?