ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Sváček** Jméno: **Filip** Osobní číslo: **457147**

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 modul sémantického vyhledávače a vizualizační modul pro knihovnu React.
4. Ověřte správnost vaší implementace srovnáním s existujícím řešením používaným např. prohlížečem sémantického slovníku státní správy.

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, skupina znalostních softwarových systémů FEL

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

Datum zadání bakalářské práce: **07.09.2020** Termín odevzdání bakalářské práce: **05.01.2021**

Platnost zadání bakalářské práce: **19.02.2022**

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

Czech
Technical
University
in Prague

Semantic faceted search on the React platform

Filip Sváček

Supervisor: Ing. Martin Ledvinka
May 2021

Acknowledgements

I would like to thank the supervisor of my work, Ing. Martin Ledvinka, for his willingness, advice, consultations and commitment, despite the limitations and problems that persist this semester.

Declaration

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

Prague, May 21, 2021

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

V Praze, 21. května 2021

Abstract

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

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

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

Keywords: Semantic web, React, Fasets, NPM, SPARQL

Supervisor: Ing. Martin Ledvinka

Abstrakt

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

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

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

Klíčová slova: Sémantický web, React, Fasety, NPM, SPARQL

Překlad názvu: Sémantické facetové vyhledávání na platformě React

Contents

Figures

Tables

Chapter 1

Introduction

Faceted search is a thing that most people probably use in their everyday lives, but only a minority of them are most likely conscious of it. From e-shops to product labelers to job-searching, faceted search has a substantial usage across all spheres of the internet. The situation is different with the semantic web, where most people have probably not come to direct contact with the semantic web, which mostly has a role in the background.

This project primarily deals with the combination of semantic web and faceted search. It analyses some issues surrounding it, proposes solutions to some problems, and shows the implementation of tools that address said problems. There is a focus on modularity and the possible future scalability of said solutions. As a part of this project, there is a demo implementation to check the usability of the created tools.

1.1 Motivation

There are not many software projects that focus on the issues surrounding the semantic web and faceted search, and they either focus on the back-end side or have severe flaws in them. These issues, combined with insufficient documentation, outdated implementation, high coupling, outdated implementation, and low cohesion of their codebase, mean a newly implemented solution is needed. This project aims to be that solution, aiming to provide a functional implementation that follows the best software engineering practices to become a helpful tool in its area of use-case.

1.2 Goals

Semantic web and technologies related to it are becoming more relevant and more needed with how the web expands and the need to go through a lot of data that is on it. Semantic technologies are becoming to move from the academic sphere as more of a theoretical exercise to the public sphere, where private businesses try to adapt them for commercial purposes. Despite that fact, many tools needed to use the semantic web and fully utilize it are in bad shape. This project aims to be a part small remedy to the wide-reaching issue at stake.

Chapter 2

Used technologies

2.1 HTML

HTML is the most fundamental building block on the web. It defines the structure of web content. This project uses the latest version HTML 5[1].

2.2 CSS

CSS is a style sheet language. Its intended usage is prescribing the presentation of web content. Along with HTML and Javascript, it is one of the key technologies on the web. CSS allows us to separate appearance from the content of web elements, included but not limited to layouts, colors, and fonts[2].

2.3 Facets

Facets are software components that implement one functionality, have one publicly callable interface, and no residual state. In software engineering, their function is to serve as a proven tool for surveying the informational space. Each facet consists of a set of items, which are also called facet values. [15]

2.3.1 Faceted search

Faceted search is a dynamic clustering of items or searched results into classifications, where users can get searched results by any value in any field. Each facet displayed also shows the number of hits within the search that match that category. [15]

2.4 Javascript

Javascript is a scripting language that is most frequently used to develop web technologies that run on the client-side.[3] It's of the key technologies on the

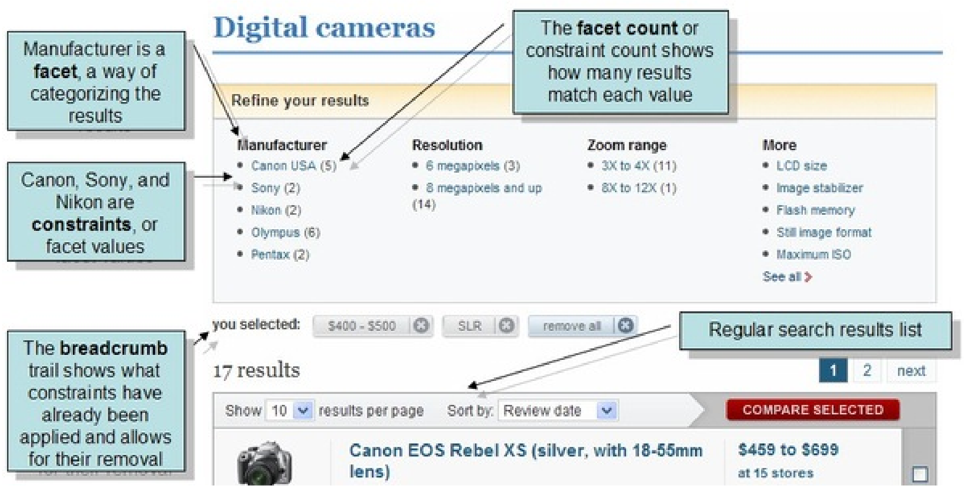

Figure 2.1: Demonstration of facet use in e-shops, where there are facets, their selection values and the number of searchable results for each value.

web and ubiquitous in modern web applications.

2.4.1 Bootstrap

Bootstrap is the most popular framework for HTML, CSS, and Javascript, emphasizing mobile development.[13] It has design templates designed for forms, buttons, navigation, and other such components. The project uses the version of Bootstrap 4.0.

2.4.2 JSX

JSX is a syntactic extension of Javascript, and developers often use it in combination with React. It's not a requirement, but it is an excellent visual tool when working with user interfaces in Javascript.[10]

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

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

2.4.3 Sparql.js

Sparql.js a parser that can parse JSON objects to SPARQL and from SPARQL to JSON.[11] This library was released quite recently, and initially, this project should have used the designed and implemented semester project parser, which this project follows upon. Sparql.js library has a much richer functionary, existing public documentation, and the library includes thorough testing, so this library is preferable to use this instead of the semester project parser solution.[11]

2.4.4 Node.js

NodeJs is an open-source runtime environment, which allows developers to run code outside of the web environment. NodeJS runs asynchronously in one thread and has a massive emphasis on scalability and performance.

2.4.5 NPM

Npm is a package manager for Javascript. In NodeJS, it's a default package manager. [12] There are defined packages in the file package.json, where the name and version of the application required attributes. Other things can be added on and removed based on the application's needs.

To demonstrate this, here is an example of a small part of the npm package, which is used in the application.

```
{
 "name": "bcproject",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
    "@testing-library/user-event": "^7.2.1",
    "dependencies": {
    "@testing-library/jest-dom": "^4.2.4",
    "@testing-library/react": "^9.5.0",
    "bootstrap": "^4.5.2",
    "node-fetch": "^2.6.1",
    "prop-types": "^15.7.2",
    "react": "^16.13.1",
    "react-bootstrap": "^1.3.0",
    "react-dom": "^16.13.1",
    "react-scripts": "3.4.3",
    "sparqljs": "^3.1.2",
    "xmlhttprequest": "^1.8.0"
  },
}
```

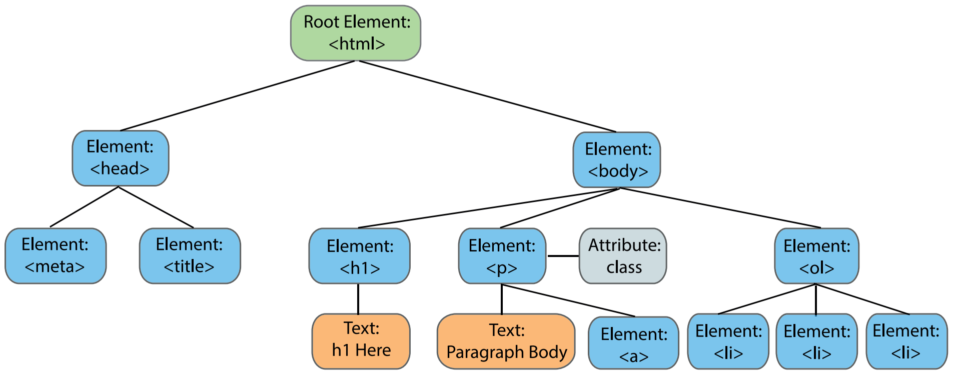

Figure 2.2: An example of a DOM graph

2.5 Git

Git is a distributed control version system, initially designed for coordinating for coordination of programmers working on software development. Over 40 million developers use it, and it is a crucial tool for effective software development. It's easy to learn and offers good data compression and high performance. This project uses it to store project data and for code revision.[14]

2.6 DOM

DOM (picture 2.2) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the document's content, structure, and style. [16]

In this project, DOM manipulates with the help of Javascript to add, update and remove HTML elements.

2.7 ReactJS

React is an open-source library written in Javascript. It's made by Facebook, along with a community of independent developers, and it's used for user interface development on the web. This library is prevalent, well documented, and with a growing use-case, which leads to its use in this project.[18]

2.7.1 Props

Props are arguments that React passes to components. HTML attributes help with the passing of props.

To demonstrate this, here is an example of pass usage.

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

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

2.7.2 Components

On a conceptual level, components are akin to functions in Javascript. They serve to split up the code into building blocks, which allows for better reusability and clarity of code. They take props as an input and return React elements, which prescribe displaying of elements in the web browser.

To demonstrate this, here is an example of pass component usage, where ShoppingList displays all shopping items and items passed through props.

```
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Item 1</li>
          <li>Item 2</li>
          <li>Item 3</li>
        </ul>
      </div>
    );
  }
}
```

Presentational components

Presentational components don't have an inner state, except for the state related to the presentation. They don't have any methods in them and are generally responsible for simply generating HTML code. In this project, presentational components primarily serve the purpose of user-defined functions to display the incoming data.

Container components

Container components are more complex than presentational components, have an inner state, and have inner methods. They are used in this project as part of the visual module because of the project's need for inner methods and states when communicating with the logic module.

2.7.3 State

State in React represents all dynamic content. Container components have the setState method in them, which React uses for setting state and changing state. After changing state with this method, React updates the web content it generates.[17]

To demonstrate this, here is an example of a function that displays all car id's, which are in the state of the component.

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

2.7.4 Render

It is a function that is responsible for rendering HTML inside the web app. It takes two parameters, the first of them is HTML code, and the second is HTML element. Because this app uses the so-called render props technique, it means that it passes code from one component to another. That's how it specifies HTML element only in the index.js file, and elsewhere we are using only HTML code.[18]

To demonstrate this, here is an example of a function tick that renders time as it passes.

```
function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(element, document.getElementById('root'));
}
setInterval(tick, 1000);
```

2.7.5 Lifecycle metody

Lifecycle methods are methods that components use. Each component has a lifecycle, which can be manipulated during its three key lifecycle phases, mounting, updating, and unmounting.[17]

During mounting, DOM received elements from React. During updating, React updates elements inside DOM. At last, during unmounting, React removes elements from DOM.

To demonstrate this, here is an example usage of componentDidMount for loading data from the endpoint. After mounting all users are loaded from the endpoint, state is changed and user statuses are displayed.

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

    this.handleStatusChange.bind(this);
  }

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

2.8 Semantic web

The semantic web is an extension of the current web, where information has a clearly defined meaning, allowing for better interaction between people and computers.[4] The primary purpose of the semantic web is to guarantee that computer data are well-readable to computers.

2.8.1 SPARQL

SPARQL is a semantic queering language made to manipulate and retrieve data in RDF format in graph databases.[12] SPARQL allows users to execute queries in a database or any other data source with an RDF format.[7]

2.8.2 Query

Query in the semantic web context is a set of technologies and protocols that can return data from the web using programming. This concept applies in many other database languages such as SQL, but in SPARQL, it uses triple patterns to get the desired data. When using SPARQL, it's possible to get complex information, where the table format is a possible way to return information.

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

2.8.3 Linked data

Linked data is a method for publishing structured data with the help of a vocabulary[24], which can be connected and interpreted by computers.[5] A typical example of a linked dataset is DBPedia, which has a subset of Wikipedia's content inside of itself in the RDF format.

2.8.4 Vocabulary

Vocabulary in the context of semantic web defines concepts and relations for describing and representing data in a given field of interest.[6] The aim of vocabularies on the semantic web is to help get rid of ambivalences in a large set of data. Other aims include the ability to find new relations between data types and organizations that deal with knowledge.

2.8.5 Semantic triple

Semantic triple is an atomic entity in RDF data format. It's a set of three entities, which are called subject, predicate, and object. Subject and object represent vertices and predicate the oriented edge pointing from subject to object. Together, they make a statement about semantic data.[4]

The usage of this format rests on the fact that machines read it well. It's possible to address each part of the statement with a unique identifier, and thanks to that, it's easy to operate and query under the semantic data.

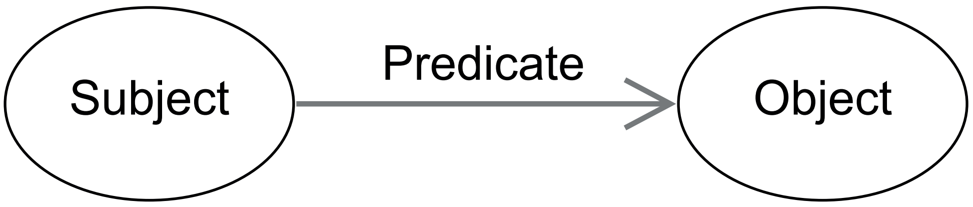

Figure 2.3: Graphical representation of Semantic triple

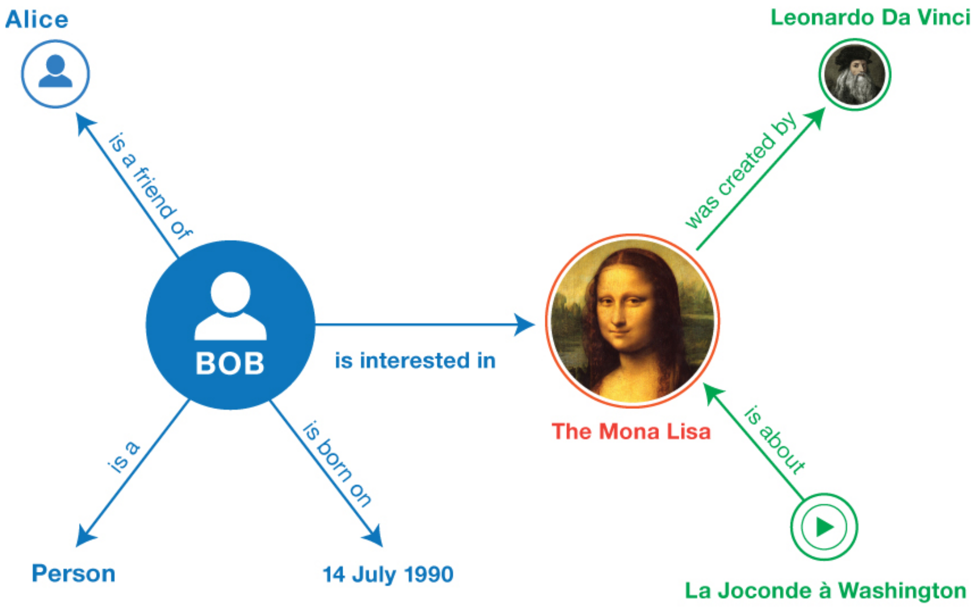

Figure 2.4: Graphical representation of a RDF graph

2.9 RDF

RDF is a framework designed to store information and a standard for data exchange on the web. It consists of a linked web structure and uses URI to name relations between things and between the ends of a link. Its aim is accessible and transferable data structures made for flexible use between many different applications. This linked data structure makes a graph, which people can easily understand.[7]

Chapter 3

Design and analysis

3.1 Comparison with other approaches to faceted search

To know what projects to select to compare this project too, first, we have to define some criteria. The criteria for selecting existing projects are as follows.

1. The other project's functionality needs to be substantially related to this project's functionality.
2. The other project needs to be used by a sufficiently large number of developers.
3. The other project needs to be maintained and work as intended to this day.
4. The other project needs to be publicly available.

The test is not all or nothing, except for number four, which is mandatory. If the other project lacks strength in one of the criteria, the selection is still not out of the question, especially if being particularly strong under other criteria.

3.1.1 Item.js

Criteria match

Items.js[23] matches all of the defined criteria. That project is a very comprehensive faceted search engine that takes a JSON dataset and displays it. According to the NPMJS statistics, it has roughly 600 downloads per day[23]. The solution was last updated a few weeks ago, indicating an active development. Based on several demo examples online, it's working as intended.

Functionality and design comparison

Items.js specializes entirely in creating a facet search app based on the user-defined configuration. In that respect, it's very similar to this project's visual module.

The difference is Items.js is way more comprehensive in its domain, with a more extensive selection of facets, more ways to set up the facets, displaying of selected data, and more settings regarding how many facets will be displayed.

Usecase comparison

While Items.js is a go-to solution for people who want to apply faceted search on their data, its comprehensiveness doesn't make it a good fit for this project, where a leaner solution is needed. It would likely be possible to use it as part of, if not as the whole visual module, but the decision in this project is to create a custom solution in React.

3.1.2 Solr Faceted Search React

Criteria match

Solr Faceted Search React[22] matches points 1, 2, and 4 with some caveats. On the one hand, it's substantially related to this project's functionality. It's a faceted search using React. At last, it has roughly 50 downloads daily[22], according to NPMJS statistics. On the other hand, it has been three years since the last update. Also, it's closely tied with different faceted search technology. Though on balance, with it still being widely used, this project passes the criteria.

Functionality and design comparison

Solr Faceted Search React specializes in faceted search, closely tied to Apache Solr, a prerequisite to running the project. It has extensive options related to configuring facets, displaying them, and the selected results. It also has a wide selection in terms of facet types.

On the contrary, this project doesn't require downloading any external software to work. Data on the semantic web tends to be freely available online, and getting data is usually just a matter of asking an endpoint for it with a SPARQL query.

Usecase comparison

While that solution is more comprehensive than this project, it's closely tied to Apache Solr. The project is practical only when using the Solr platform. The project would not be helpful in the context of the semantic web.

3.1.3 SPARQL Faceter

Criteria match

SPARQL Faceter[21] matches points 1 and 4. SPARQL Faceter tries to implement almost the same functionality as this project. On the other hand, SPARQL Faceter project is not very active. The last update is from 3 years ago, and demo examples in the project are not working, indicating that not many people are using it.

On the other hand, SPARQL Faceter is so substantially related to this project that it passes the criteria. SPARQL Faceter heavily influenced this project. At the same time, this project tries to correct some things in SPARQL Faceter.

Functionality and design comparison

Both projects are very similar in functionality, aside from the other project having more defined facets. It's in the design the difference is more severe. SPARQL Faceter doesn't separate logic from a user interface. The query creation process is more cumbersome, adding up strings together instead of some more high-level parser. Because of these issues, it's also not very expandable.

Usecase comparison

At this moment, the other project is not very usable for its intended purpose. Queries sent to the backend stopped working, and fixing the issues would take more effort than it's worth. In contrast, this project tries to address a lot of the issues that plague the SPARQL Faceter and hopefully is successful in that goal.

3.2 Summary

While there are many-faceted search projects online, only a handful of them is coming close to what this project requires. All fall short, either because they offer only a small part of the solution or they try to tackle something a bit different. And if they are coming close to what this project requires, they lack in other areas, such as implementation and design.

3.3 Modular design

There is a need for separation in the app into two basic modules: The logic module and the visual module.

The main reason for the modular design is that the logic is independent of the visual platform. That is necessary because visual platforms come and go, and as they get older, their support on the web goes away. Independence of logic from visual platform guarantees the longevity of the project. When the visual platform is not supported anymore, the logic stays intact, so it's possible to use it in combination with a newer visual platform.

Another big reason is the separation of concerns, where each module has clearly defined boundaries of responsibility. The separation of concerns is necessary for the maintainability and reusability of code.

The logic module's responsibility is taking facet data, parsing them into a SPARQL query, and returning data from the endpoint. The visual's module is responsible for displaying a faceted search based on the project configuration and facets and results from the endpoint.

Logic Module is wholly independent, whereas the visual module is reliant on the Logic Module. This reliance only extends to giving logic module facet data from the user's side, setting up data from pagination, and receiving data, so coupling remains low.

3.4 Component design

To ensure smooth modeling of implementation details, a component diagram (picture 3.1) with established interfaces is needed and analysis of component's interfaces.

3.4.1 Making SPARQL query into a HTTP request

EndpointCaller, which is an object that that receives queries and sends data to enpoint, creates an HTTP request for the backend from a SPARQL query by taking the backend's URL address, encoding Sparql query into a query URI component, and from that assembling the final HTTP request.

3.4.2 Passing selected facets to logic module

When the user is selecting facets, the visual module calls the logic module, and the app passes changed data into the logic module.

3.4.3 Communicating with a SPARQL endpoint

The communication between the project and endpoint is just sending HTTP POST Request to get sought-after data, which the visual module displays.

3.4.4 Parsing a query from JSON to SPARQL

For generating queries, the app uses an existing library that functions as a generalized JSON to SPARQL parser.

The only function that the logic module serves to generate SPARQL queries is to prepare the data in the format that the Sparql.js library can parse. For that reason, there are JSON template variables inside the logic module, which the logic module modifies for the library to parse the data and create needed queries.

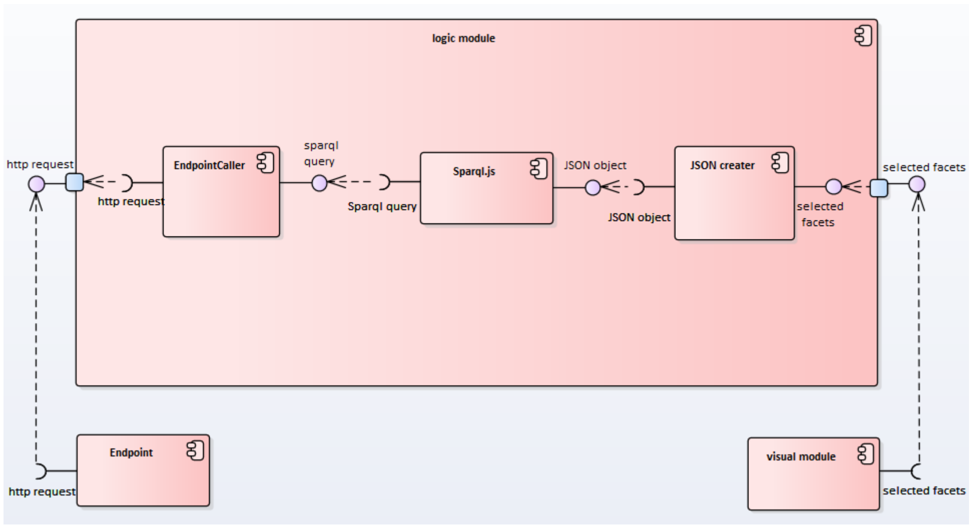

Figure 3.1: Component diagram

3.5 Other design elements and decisions

3.5.1 JSON to SPARQL parser

This project uses a universal JSON object to the SPARQL parser because it allows for a flexible and maintainable way to build SPARQL queries. It allows for easy setting of facet data in the query, unlike if the query was built from scratch, adding various strings together until the final query is made, which is the process in the SPARQL Faceter project.

3.5.2 Facets

For the project to filter data effectively, we need two types of facets: Text facets and select facets. Text facets filter data through text input the user writes in the app, and select facets filter data by selecting pre-loaded options.

3.5.3 Constraints

Because it's not advisable to search the entire endpoint when querying for data, the project uses constraints. Constraints in the app's context are Semantic triples that the app uses to narrow down data. Constrains would be pre-loaded and user defined, and are in every single query all the time.

For example, let's assume that we are searching for writers on the DBpedia.

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

Instead of returning all data that the endpoint has in the beginning, and dbpedia endpoint has a vast amount of data, constraint named writer allows us to perform faceted search only above writers, and above no other set of data.

3.5.4 Query templates

Query templates are user-defined and needed in the process of creating the final query. They are used to offer flexibility to the user. If the entire query were created from scratch by the app, the possible data selected would be far more limited, or very cumbersome. There would be many unaccounted possibilities in the data selection process were it not for user-provided query templates. Even if it were possible to setup such a query from scratch in the logic module, it would not be very practical.

To demonstrate this, here is an example of a query template for select facets.

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

It's evident from this template that setting up optional values and binding them in the logic module using methods would not be very practical. It's much more user-friendly to let the user write a large part of the query that stays static during the facet selection process and generate the rest using the logic module. The parts generated by the logic module are called subqueries.

In the example above, subqueries would be places in the result set 0 and result set 1, with the rest of the query being a template.

Select facet query template

The project needs to build a query to get facet content endpoint. For that purpose, there has to be a user-defined template that the logic module takes and builds the final query from it.

Result query template

The project needs to build a query to get results from the endpoint. For that purpose, there has to be a user-defined template that the logic module takes and builds the final query from it.

3.5.5 Subqueries

The logic module's job is to generate subqueries, which in combination with user-provided query templates, the app uses to make the final query. The project should use subqueries because of the need to filter the results in selecting facets and displaying the results.

3.5.6 Configuration

Outside of being separated into two modules, the project would be heavily reliant on user-defined configuration, so this chapter is tasked with their design.

The app would use two config objects, one object to set up the logic module and the other to set up the visual module. This separation into two files is necessary because the logic module is wholly independent of the visual module, but the visual module relies on the logic module for data. Settings from the visual module aren't necessary when solely using the logic module, so it's best to separate the two configurations.

Visual module configuration

Visual module configuration would be dependent on the type of visual platform.

Logic module configuration

The logic module configuration needs to consist of all the data for module needs to work. Namely, it needs to have query templates, endpoint address, defined facets and constraints, without which the logic module could not work. All of these things need to be mandatory when using the visual module and the logic module. So in essence it serves as a precondition on the selected result data, in this case meaning that every single returned record needs to be a writer.

Chapter 4

Implementation

4.1 Configuration

4.1.1 Visual module configuration

React is used in this project for the visual module because it's a simple, well-documented library that millions of developers use and is regularly maintained and updated.

For configuring the visual module in react, the project uses two parameters: Title and Iterator.

Title

The parameter title takes a JSX function that displays the names of the columns.

An example of the JSX function Title displaying column names.

```
export const Title = () => {
    return (
        <tr>
            <th>Writer</th>
            <th>Abstract / Works</th>
            <th>Birth Place</th>
            <th>Notable Work</th>
        </tr>
    )
}
```

Iterator

The parameter Iterator is a function that and displays the incoming data in the best way the user sees fit. It's used to offer flexibility to users, which can be useful with more difficult incoming data patterns.

An example of the JSX function Iterator displaying received data.

```
const Element = ({name, abstract, birthDate, deathDate}) => {
    return (
        <tr>
            <td>{name}</td>
            <td>{abstract}</td>
            <td>{birthDate}</td>
            <td>{deathDate}</td>
        </tr>
    );
};

const Iterator = ({data}) => {
    return data.map
    (
        (data) => {
            return (
                <Element
                    key={data.id.value}
                    name={data.name ? data.name.value : ""}
                    abstract={data.abstract ?
                    data.abstract.value : ""}
                    birthDate={data.birthDate ?
                    data.birthDate.value : ""}
                    deathDate={data.deathDate ?
                    data.deathDate.value : ""}
                />
            )
        }
    )
}
```

4.1.2 Query creation process

First, facets and contraints are loaded. From there, they are used to create subqueries. These subqueries are then combined with query templates to create a query. This process is used both for select facet query and result query.

4.1.3 Logic module configuration

For configuring the logic module, the project uses four parameters: facets, endpointUrl, selectQueryTemplate and resultQueryTemplate.

An example of an object logicConfig, which is used by the logic module.

```
    let logicConfig = {
        facets : facets,
        endpointUrl : 'http://dbpedia.org/sparql',
        resultQueryTemplate: resultQueryTemplate,
        constrains: constrains,
        selectQueryTemplate: selectQueryTemplate
    };
    let logicModule = new LogicModule(logicConfig)
```

EndpointUrl

EndpointUrl is an address of the SPARQL endpoint. The app sends an HTTP Request and receives data, and the app then displays data in the Visual module.

selectQueryTemplate

SelectQueryTemplate is a string defined by the user to create a query. It contains values <RESULTSET0> and <RESULTSET1>, which are used as placeholder strings. This is where a subquery is placed after it's generated in the process of generating a final query that is then used for an endpoint.

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

One is used to count the total number of results when not having a selected facet value, and another one used for returning facet values.

ResultQueryTemplate

ResultQueryTemplate is a string defined by the user to create a query. It contains a value <RESULTSET>, which is used as a placeholder string. This is where a subquery is placed after it's generated in the process of generating a final query that is then used for an endpoint.

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

There is one placeholder string, named result set. It's use is to get result records based on facet values.

Constrains

Constrains are objects that have parameters named Object, Predicate, and ConstraintId. Object and Predicate both must be graph nodes, whereas ConstraintId should be a unique identification string.

Facets

Facets are objects that have parameters named FaceType, Predicate, Name, and FacetId. FaceType must either be a string with value text or with value select, defining whether the facet is a select facet or a text facet. Predicate must be a graph node. Name is the displayed label for the facet in the Visual module.

4.2 JSON object templates

The logic module uses JSON object templates which it uses to build queries using SPARQL Parser. These templates are variables inside the module, and their purpose is to help in query creation. There five of these constants inside the logic module, which the library sparql.js can parse when loaded with facet data.

An example of JSON object template, that the app uses to build a SPARQL query.

```
this.resultsubQueryJSON = {
    queryType: "SELECT",
    distinct: true,
    variables: [
        {
            termType: "Variable",
            value: "id"
        }
    ],
    where: {
        type: "bgp",
        triples: []
    },
    order: [
        {
            expression:
                {
                    termType: "Variable",
                    value: "id"
                }
        }
    ],
    limit: 10,
    offset: 0
};
```

4.2.1 resultCountJSON

The app uses a JSON object template named resultCountJSON to help build a query that gets the record count for pagination.

4.2.2 resultsubQueryJSON

The app uses a JSON object template named resultsubQueryJSON to help build a query that gets the record result records.

4.2.3 textFacetTemplateJSON

The app uses a JSON object template named textFacetTemplateJSON to add facet text input for both select facets and the result records.

4.2.4 firstSubQueryJSON

The app uses a JSON object template named firstSubQueryJSON to build a select facet query. The query it help to build gets the number of total records.

4.2.5 secondSubQueryJSON

The app uses a JSON object template named secondSubQueryJSON to build a select facet query. The query that it helps to build is used for returning facet values.

4.3 Logic module’s public methods

The logic module has public methods that the visual module accesses to get data. It also has public setters to take data from the site.

4.3.1 getResultPromise

The app uses getResultPromise to get a Promise of results of the faceted search from the endpoint.

4.3.2 getResultCountPromise

The app uses getResultCountPromise to get a Promise of the number of results of the faceted search from the endpoint.

4.3.3 getFacetSelectionPromises

The app uses getFacetSelectionPromises to get Promises of the select facet values from the endpoint.

4.3.4 setOffsetToResultQuery

The app uses setOffsetToResultQuery when a user is paging between records. It has a parameter named pageNumber, which sets the value of the page user clicks at minus one.

4.3.5 setTextToTextFacet

The app uses setTextToTextFacet to set data from the text facet that the user writes inside the text input. It has a parameter facetId, a key that points to the facet value inside, and an object parameter, which is the data that the user wrote inside the text input.

4.3.6 setObjectToBasicFacet

The app uses setObjectToBasicFacet to set selected data from the select facet. It has a parameter facetId, a key that points to the facet value inside, and an object parameter, which is the data that the user wrote inside the text input.

4.4 Communication with the backend and processing of results

The results are obtained using the getJSON () method, and the app sends an HTTP request to the backend SPARQL query encoded in the URL. The method returns the results that came from the endpoint and stores them in the state of the component.

Chapter 5
Evaluation

5.1 Introduction

To evaluate that the implemented solution works, a demo that tests the solution's functionality is required. For that purpose, the project implements a government vocabulary explorer demo. There exists another solution that SPARQL Faceter implements[24], and unlike all other publicly available demos, this one still works.

The project uses a configuration that will allow us to compare the functionality of this application with an existing solution. A publicly available demo has been selected, named Semantic Government Vocabulary Explorer. The purpose is to create a replica that functions the same way for validation purposes.

5.2 Demo description

Demo's purpose is to show the meaning of terms in different contexts of the law. Because terms or words often don't have one consistent meaning, they can cause ambiguity. For instance, the word building has several different meanings in the different areas of law. This demo tries to solve this problem by showing terms across different glossaries, showing information about the terms, information about the term, and the link from the glossary.

5.3 Demo facets

Demo contains three defined facets: one text facet and two select facets. Demo uses the text facet to filter term names based on the user input. It uses a glossary facet to select from which glossary the term comes from, and it uses a type facet to select what specifies the type of term.

5.4 Comparing both demos

When it comes to functionality, both demos function more or less the same. They both have the same facets. The queries they use to get data from the backend are virtually the same by design. One key area where they differ is that this solution paginates results, while the SPARQL Faceter demo doesn't and instead displays thousands of results outright. That means this demo has much shorter load times and therefore is more user-friendly to use.

Another difference is that the demo implemented by SPARQL Faceter allows users to switch languages from Czech to English, whereas this demo now supports only the Czech language. This feature has not been requested and is not working fully in the SPARQL Faceter demo, as the labels for facets stay the same regardless of the language. It would also make the configuration slightly more demanding for the user. Still, it's not a problem to either create a demo in a different language or to implement this functionality in the future.

All in all, based on the implemented demo and the comparison to the existing one, both the logical and visual modules work well and match expectations.

Chapter 6
Conclusion

6.1 Summary

Requirements demand a detailed analysis and design so there would be a proper app implementation. The analysis establishes modules and their responsibilities within the project. The analysis also defines all components of the projects and describes their interfaces. Lastly, the analysis touches on how to design user-defined configurations for the application to work correctly.

As part of the project, there is an implemented app that separates logic from the user interface. Thanks to that, the app will be way more maintainable in the case of change in Javascript's technologies, which could happen in the future.

There is an implemented demo of vocabulary terms to show that the app works. The demo allows users to test the visual module and the logic module by using it and comparing the functionality to another existing solution.

There is a comparison to other existing faceted search solutions. The existing solutions are not suited for the semantic web or have severe flaws that make them hard to use. Universally, going over existing solutions can lead to preventing repeating the mistakes in them. That has been the case in this project also.

6.2 Future development

Just because the assignment is complete doesn't mean the project is fully complete. There can always be things to be added upon later. This project does not implement every conceivable possibility for the types of facets it uses, for example. The only implemented types of facets are those that the assignment demanded. But all is done so that whenever there is a need to expand the project, there should be no problems expanding it from following good software practices to having sufficient documentation.

The project is in a good state for future expansion, and it depends on the potential developers, who are going to use it, how they are going to use it, and for what purpose they are going to use it.

Chapter 7
Sources

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

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

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

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

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

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

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

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

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

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

21. SPARQL Faceter [online]. github.com [cit. 2021-01-05].
Available at: https://github.com/SemanticComputing/angular-semantic-faceted-search

22. Solr faceted search [online]. npmjs.com [cit. 2021-05-05].
Available at: https://www.npmjs.com/package/solr-faceted-search-react

23. itemsjs [online]. npmjs.com [cit. 2021-05-05].
Available at: https://www.npmjs.com/package/itemsjs

24. SPARQL Faceter demo [online]. https://slovnik.gov.cz/ [cit. 2021-05-05].
Available at: https://slovník.gov.cz/prohlížeč/

25. Linked Data [online] https://www.coursehero.com/file/93934947/Linked-Datadocx/ [cit. 2021-05-05].

Chapter 8

Pictures

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

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

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

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

Chapter 9
Attachment

The project address is https://gitlab.fel.cvut.cz/svacefil/bcproject

After cloning the project

```
npm install
```

To install all the related packages of the npm environment.

After that

```
npm start
```

a project starts running