VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÝ EDITOR PRO JAZYK SPARQL

WEB EDITOR FOR SPARQL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **DANIEL PAULOVIČ**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Paulovič Daniel**
Program: Informační technologie
Název: **Webový editor pro jazyk SPARQL**
Web Editor for SPARQL
Kategorie: Databáze

Zadání:

1. Seznamte se s dotazovacím jazykem SPARQL a datovým modelem RDF.
2. Prostudujte existující nástroje a knihovny pro tvorbu klientských webových aplikací v jazyce JavaScript. Zaměřte se zejména na rámec Vue.js.
3. Navrhněte webový editor a architekturu související aplikace pro interaktivní zadávání SPARQL dotazů a procházení výsledků dotazu.
4. Po dohodě s vedoucím zvolte vhodnou implementační platformu a implementujte editor jako samostatnou komponentu. Implementujte demonstrační aplikaci umožňující procházení výsledků.
5. Otestujte vytvořené řešení na vhodně zvolených datech.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Lasila, I., Swick, R. R.: Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222
- The W3C SPARQL Working Group: SPARQL 1.1 Overview, W3C Recommendation 21 March 2013, https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Burget Radek, doc. Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 11. října 2021

Abstrakt

Cieľom tejto práce je vytvoriť webový editor pre dotazovací jazyk SPARQL ako komponentu v javascriptovom frameworku Vue.js a webovú aplikáciu využívajúcu túto komponentu pre dotazovanie sa nad SPARQL endpointami. V riešení editoru bola využitá knižnica CodeMirror a knižnica PrimeVue pre prácu s užívateľským rozhraním. Vytvorené riešenie poskytuje voľne dostupnú komponentu ako npm package. Prínosom tejto práce je ľahko importovateľný editor s množstvom funkcií pre uľahčenie práce s písaním dotazov.

Abstract

Target of this thesis is to create web editor for query language SPARQL as component in javascript framework Vue.js and web application using this component to query over SPARQL endpoints. To implement editor were used CodeMirror library and PrimeVue library for user interface. Final version of component is publicly available as npm package. Benefit of this thesis is easily importable editor with many functions to simplify writing queries.

Klíčová slova

RDF, SPARQL, Angular, React, Vue, HTML, DOM, CSS, JavaScript, CodeMirror, PrimeVue, Axios, npm, Vite, webová aplikácia

Keywords

RDF, SPARQL, Angular, React, Vue, HTML, DOM, CSS, JavaScript, CodeMirror, PrimeVue, Axios, npm, Vite, web application

Citace

PAULOVIČ, Daniel. *Webový editor pro jazyk SPARQL*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Radek Burget, Ph.D.

Webový editor pro jazyk SPARQL

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Radeka Burgeta Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Daniel Paulovič
8. května 2022

Poděkování

Rád by som poďakoval svojmu vedúcemu doc. Ing. Radku Burgetovi, Ph.D. za ochotu a odborné vedenie pri konzultáciách.

Obsah

Kapitola 1

Úvod

Existujú dátové úložiská RDF, nad ktorými sa dá dotazovať pomocou špeciálneho jazyka SPARQL. Pre dotazovanie nad RDF úložiskom za pomoci SPARQL sa využívajú SPARQL endpointy, na ktoré sa dané dotazy posielajú pomocou HTTP požiadaviek. Pri tvorbe takýchto požiadaviek webové aplikácie často krát využívajú obyčajný HTML prvok textovej arei. Existuje niekoľko riešení náhrady tejto textovej arei s vylepšenou verziou pre zlepšenie písania týchto dotazov, ale väčšinou sú súčasťou nejakého väčšieho projektu alebo neponúkajú dostatočnú funkcionalitu na naozaj zjednodušenie písania týchto dotazov a teda ušetrenie času pre užívateľa.

Táto bakalárska práca sa zaoberá tvorbou ľahko importovatelnej komponenty webového editora pre špeciálny dotazovací jazyk SPARQL a následným predvedením práce s touto komponentou v ukážkovej webovej aplikácií pre posielanie dotazov na SPARQL endpointy a zobrazovanie ich výsledkov. Komponenta webového editora obsahuje automatické dopĺňanie a zvýrazňovanie kľúčových slov, rôznymi témami, jednoduchým importovaní prefixov do textu za pomoci užívateľského rozhrania a iné.

Práca sa skladá z 8 kapitol. Druhá, tretia a štvrtá kapitola sa zaoberá oboznámením čitateľa s dotyčnými technológiami. Najskôr sa v druhej kapitole 2 venuje dátovému modelu RDF a jazyku SPARQL dotazujúcim sa nad ním. Následne sa v tretej kapitole 3 sa prechádzajú momentálne 3 najpopulárnejšie JavaScriptové rámce pre tvorbu jednostránkových aplikácií: Angular, React a Vue.js. Pri každom sa preberie jeho všeobecný popis a práca s ich komponentami. V štvrtej kapitole 4 sa nachádza základný opis použitých webových technológií a nástrojov pri tvorbe komponenty editora a aplikácie. Medzi opísané technológie a nástroje patria HTML, DOM, CSS, JavaScript, CodeMirror, PrimeVue, Axios, NPM a Vite. Za ňou nasleduje piata kapitola 5 obsahujúca návrhy rôznych častí editora a aplikácie pred začatím samotnej implementácie. Primárne sa jedná o obrázky návrhu užívateľského rozhrania a dôvody pre implementáciu rôznych funkcionalít. Napokon túto kapitolu dopĺňa šiesta kapitola 6 kde sa preberá spôsob riešenia návrhu za pomoci technológií a nástrojov popísaných v kapitole 4. Posledná kapitola pred záverom 7 sa zameriava na otestovanie výsledkov tejto.

Kapitola 2

Popis dátových technológií

Táto kapitola slúži na informovanie čitateľa o dátovom modely RDF, ktorým sa táto práca zaoberá a o dotazovacom jazyku SPARQL pre ktorý bude komponenta editoru vytvorená. Ďalej sa venuje súčasným webovým technológiám. Obsahuje krátky popis jednotlivých technológií a jazykov a ich porovnanie. Ďalej na záver popisuje pomocné webové rámce pre tvorbu editorov zameraných na programovacie jazyky v webovom prehliadači.

2.1 Resource Description Framework

Resource Description Framework [10] alebo ďalej už len ako RDF, je základ pre spracúvanie metadát, metadáta sú údaje o údajoch, napríklad zoznam dokumentov sú metadáta. RDF reprezentuje dáta v zmysle aby ich mohli spracovať iné stroje a pritom nedošlo k strate významu, teda umožňuje zautomatizované spracovanie webových zdrojov. Modelované dáta pomocou RDF sa nazývajú zdroje, popísané jednoduchými vlastnosťami a hodnotami.

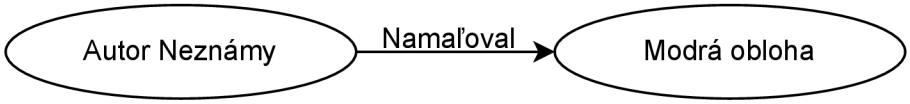
2.1.1 Základy

Základy dátového modelu RDF sú 3 objektové typy:

- **Zdroje** - Všetko popísané RDF výrazom sa nazýva zdroj. Zdroj môže byť celý HTML dokument alebo len XML element v danom dokumente. Zdroje sú vždy nazvané podľa URI
- **Vlastnosti** - Vlastnosť sa vždy vzťahuje k zdroju, a opisuje jeho napríklad jeho aspekt ,vzťah, charakteristiku alebo atribút. Každá vlastnosť má vlastný význam, definuje povolené hodnoty, typy zdrojov, ktoré môže opisovať a vzťahy s ostatnými vlastnosťami.
- **Tvrdenia** - Konkrétny zdroj spolu s názvom a hodnotou vlastnosti pre daný zdroj tvoria RDF tvrdenie. Tieto tri individuálne časti tvrdenia, nazývané ako RDF trojice, subjekt, predikát a objekt po anglický (subject, predicate and object) základom modelovania. Objekt tvrdenia môže byť špecifikovaný pomocou URI alebo to môže byť aj iný zdroj, poprípade len primitívny dátový typ alebo jednoduchý reťazec.

2.1.2 Príklady

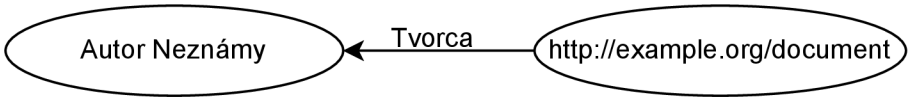
Uvažujme ako jednoduchý príklad nasledujúcu vetu: *Autor Náhodný namaľoval Modrú obloh*u. Kde Modrá obloha predstavuje fiktívny názov maľby a Autor Náhodný je fiktívne meno autora. Dáta reprezrentované grafom by vyzerali nasledovne:



Obrázek 2.1: Graf príkladu RDF tvrdenia

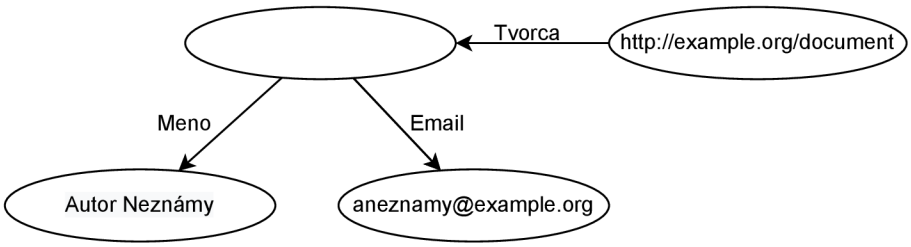
Šípka vždy smeruje od subjektu k objektu.

Uvažujme ako jednoduchý príklad pre ukážku využitia URI ako zdroj nasledujúcu vetu: *http://example.org/document bol vytvorený Autorom Náhodným.*



Obrázek 2.2: Graf príkladu RDF tvrdenia 2

Vyššie uvedené veta sa dá jednoducho rozšíriť, napríklad, keby chceme o Autorovi Neznámom zaznamenať viac informácií ako napríklad email, vek a podobne, bude graf dát vyzerať nasledovne:



Obrázek 2.3: Graf príkladu RDF tvrdenia 3

V obrázku 2.3 sa taktiež využíva **blank node** alebo v preklade prázdny uzol. Jedná o špeciálny prázdny uzol, ktorý nenesie žiadnu hodnotu a slúži na k štrukturalizácií dát do viac samostatných uzlov. Tieto dáta je potrebné jednoznačne identifikovať. Dovoľuje nám to rozdeliť tvorcu dokumentu do viac uzlov a zaznamenať o ňom viac informácií.

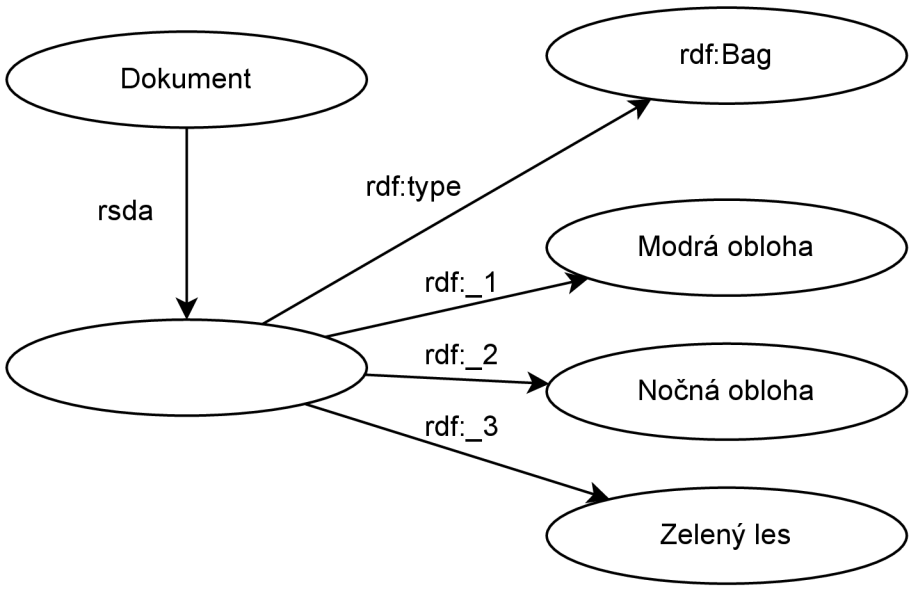
2.1.3 Kontajnery

V prípade potreby odkazovania na zbierku zdrojov, napríklad, že knihu napísalo viac osôb, sa v RDF používajú kontajnery.

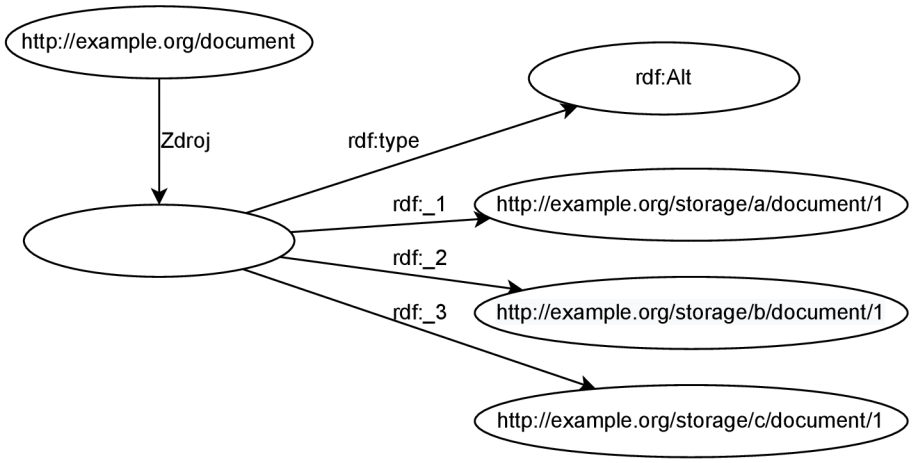
V RDF sú definované 3 typy kontajnerov:

- **Taška (rdf:Bag)**: Ide o neusporiadaný zoznam literálov alebo zdrojov. Tašky sa používajú na vyhlásenie, že vlastnosť ma viac hodnôt a že poradenie, v akom sú hodnoty uvedené, nie je dôležité. Môže obsahovať aj duplicitné hodnoty.

- **Sekvencia (rdf:Seq)**: Rovnaký princíp ako Taška s rozdielom, že sa jedná o usporiadaný zoznam. Duplicitné hodnoty sú taktiež povolené. Môže byť využitá napríklad pre zachovanie abecedného poradia hodnôt.
- **Alternatíva (rdf:Alt)**: Zoznam zdrojov alebo literálov, ktoré predstavujú alternatívy pre jednu hodnotu vlastnosti. Môže to byť napríklad poskytnutie alternatívnych jazykových prekladov názvu diela.



Obrázek 2.4: Jednoduchá ukážka rdf:Bag

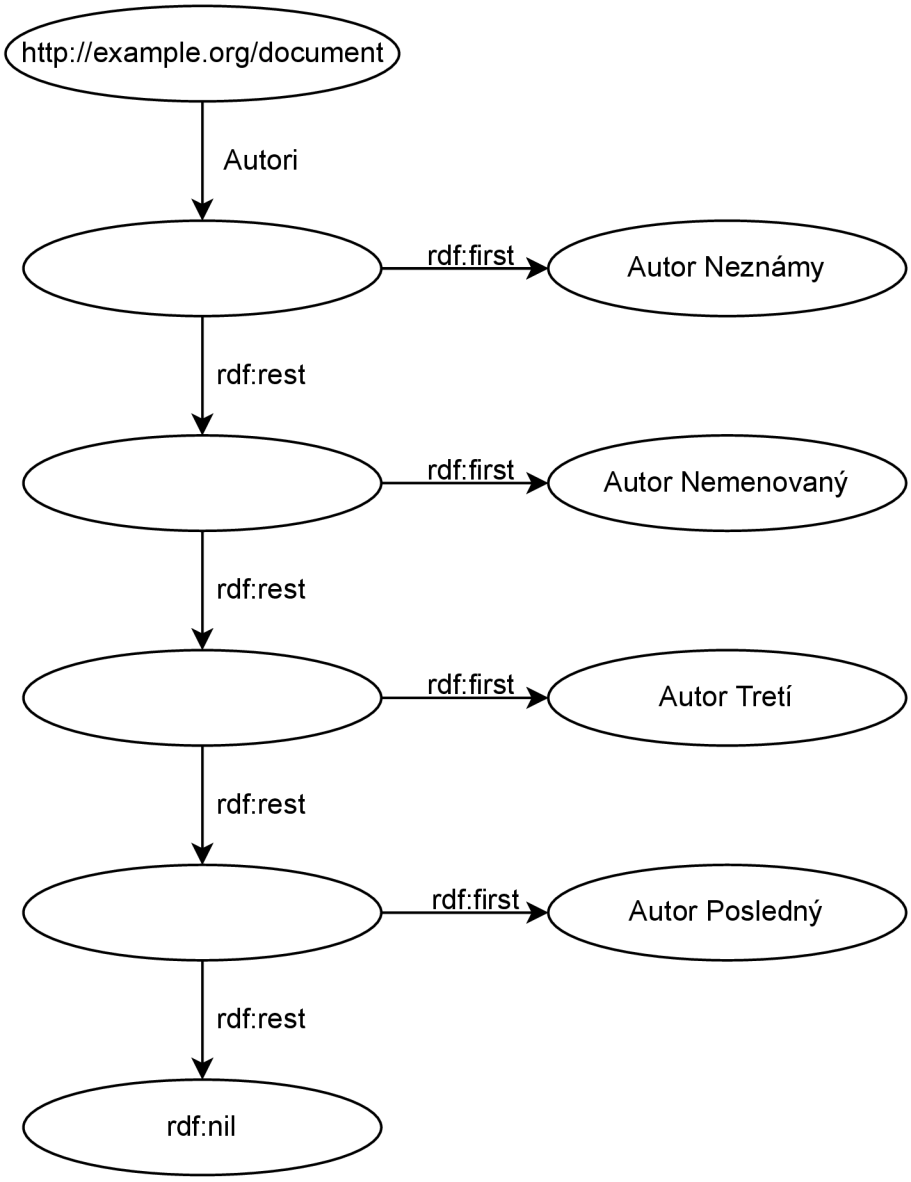


Obrázek 2.5: Jednoduchá ukážka rdf:Alt

2.1.4 Kolekcie

Kolekcie nám na rozdiel od kontajnerov typu rdf:Bag, rdf:Seq, rdf:Alt dovoľujú určiť pevný počet uzlov. Pri kontajneroch môže vždy existovať ďalší uzol odkazujúci na nový literál

alebo zdroj. Kolekcia funguje ako zoznam pri programovaní. Mám začiatok zoznam definovaný nejakým prázdnym uzlom a následne sa odkazujeme pomocou rdf:first na aktuálny uzol alebo pomocou rdf:rest, ktorá nás presunie na ďalší prázdnym uzol, ktorý má opäť rdf:first a rdf:rest. Takto to pokračuje dokým sa pomocou rdf:rest nenarazí na uzol obsahujúci rdf:nil, čo značí koniec zoznamu. Ukážkou je obrázok 2.6, zobrazujúci vetu: *Dokument http://example.org/document bol vytvorený autormi Autor Neznámy, Autor Nemenovaný, Autor Tretí a Autor Posledný.*



Obrázek 2.6: Jednoduchá ukážka kolekcie rdf

2.2 SPARQL

SPARQL [12] je dotazovací jazyk pre dátový model RDF. Bol štandartizovaný organizáciou W3C v roku 2008. Aktuálna verzia je 1.1, ktorá rieši nedostatky pôvodnej verzie a rozširuje

ju. Slúži pre dotazovanie a manipuláciu obsahu grafov RDF na Webe alebo v RDF úložišti. Podporuje dotazovanie nad SPARQL endpointami za pomocou rôznym typov príkazov, ako napríklad SELECT. SELECT dotazy vracajú premenné väzby, ASK dotazy vracajú hodnotu typu boolean. CONSTRUCT dotazy zostavia nové RDF grafy z výsledkov dotazov.

Výsledky dotazov sú podporované v 4 formátoch:

- Extensible Markup Language (XML)
- JavaScript Object Notation (JSON)
- Comma Separated Values (CSV)
- Tab Separated Values (TSV)

2.2.1 Príklady

Nasledujúce príklady sú prevzaté z článku o SPARQL [12]. Je daný nasledujúci graf nachádzajú sa na URL 'http://example.org/alice' obsahujúci osobné informácie o Alice a jej sociálnych kontaktoch. Ilustrácia využíva Turtle syntax [1].

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<http://example.org/alice#me> a foaf:Person .
<http://example.org/alice#me> foaf:name "Alice" .
<http://example.org/alice#me> foaf:mbox <mailto:alice@example.org> .
<http://example.org/alice#me> foaf:knows <http://example.org/bob#me> .
<http://example.org/bob#me> foaf:knows <http://example.org/alice#me> .
<http://example.org/bob#me> foaf:name "Bob" .
<http://example.org/alice#me> foaf:knows <http://example.org/charlie#me> .
<http://example.org/charlie#me> foaf:knows <http://example.org/alice#me> .
<http://example.org/charlie#me> foaf:name "Charlie" .
<http://example.org/alice#me> foaf:knows <http://example.org/snoopy> .
<http://example.org/snoopy> foaf:name "Snoopy"@en .
```

Príklad SELECT dotazu

Pomocou SPARQL SELECT dotazu sme schopný z grafu získať mená osôb a počet ich známostí nasledujúcim dotazom:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name (COUNT(?friend) AS ?count)
WHERE {
    ?person foaf:name ?name .
    ?person foaf:knows ?friend .
} GROUP BY ?person ?name
```

[1] https://www.w3.org/TR/turtle/

Príklad UPDATE dotazu

Pomocou nasledujúceho dotazu pridáme Alice novú známosť Dorothy do základného grafu a vymažeme všetky mená známostí Alice s označením Anglického jazyka.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/> .

INSERT DATA {
    <http://www.example.org/alice> foaf:knows [ foaf:name "Dorothy" ].
} ;
DELETE { ?person foaf:name ?mbox }
WHERE { <http://www.example.org/alice> foaf:knows ?person .
        ?person foaf:name ?name FILTER ( lang(?name) = "EN" ) .}
```

Kapitola 3

Webové technológie

Táto kapitola sa zaoberá opisom 3 najpopulárnejších JavaScriptových rámcov/knižníc vhodných pre tvorbu jednostránkových aplikácií podobných tomuto zadaniu. Všetky 3 z nich využívajú koncept komponent a sú teda vhodné aj na tvorbu editoru.

3.1 Angular

Jedná sa o aplikačný rámec a vývojovú platformu na vytváranie jednostránkových aplikácií. Jeho základom je TypeScript, čo je silne typovaný programovací jazyk, ktorý stavia na JavaScripte. Bol vytvorený v roku 2016 spoločnosťou Google. Bol vytvorený z Angular.js, ktorý bol komplikovanejšou verziou súčasného Angularu [1]. Angular ako platforma obsahuje rámec založený na komponentách pre vytváranie škálovateľných webových aplikácií, integrované knižnice pre širokú škálu funkcií, ktoré zahŕňajú smerovanie, správu formulárov, komunikáciu klient-server a mnoho ďalších. Mimo to ponúka sadu vývojárskych nástrojov pre vývoj, zostavenie, testovanie a aktualizovanie kódu. Je vhodný pre projekty všetkých rozsahov, od projektov jedného vývojára až po aplikácie na podnikovej úrovni.

Komponenty

Komponenty sú základné stavebné bloky tvoriace aplikáciu. Komponenta obsahuje triedu TypeScript s `@Component()` dekorátorom, HTML šablónov a štýlmi. Dekorátor `@Component()` špecifikuje nasledujúce informácie pre Angular:

- CSS selektor, ktorý definuje, ako sa komponenta používa v šablóne. HTML elementy, ktoré zodpovedajú tomuto selektoru, sa stanú inštanciami komponenty.
- HTML šablóna, ktorá inštruuje Angular, ako vykresliť komponentu.
- Voliteľná sada CSS štýlov, ktoré definujú vzhľad prvkov HTML šablóny.

Naledujúce ukážky kódov k Angular sú všetky prevzaté článku o Angular [1]. Jednoduchá ukážka minimálnej komponenty:

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world',
  template: `
    <h2>Hello World</h2>
    <p>This is my first component!</p>
  `
})
export class HelloWorldComponent {
  // The code in this class drives the component's behavior.
}
```

Túto jednoduchú komponentu je možné pridať do šablóny napísaním:

```
<hello-world></hello-world>
```

Po vykreslení Angularom výsledný DOM vyzerá nasledovne:

```
<hello-world>
<h2>Hello World</h2>
<p>This is my first component!</p>
</hello-world>
```

Komponenty v Angular majú svoj vlastný životný cyklus [2]. Tento cyklus sa skladá z niekoľkých fáz umožňujúcich manipuláciu s danou komponentou. Samotné funkcie fáz sú:

- **ngOnChanges()** - zavolá sa vždy, keď Angular nastaví alebo resetuje vstupné vlastnosti viazané na dáta. Volá sa pred *ngOnInit()*
- **ngOnInit()** - zavolá sa raz, vždy po *ngOnChanges()*, aj v prípade, že sa *ngOnChanges()* nezavolalo(keď komponenta nemá viazané vstupné vlastnosti)
- **ngDoCheck()** - zavolá sa okamžite po *ngOnChanges()* pri každom behu detekcie zmien, a hneď po prvom behu *ngOnInit()*
- **ngAfterContentInit()** - zavolá sa raz po prvom *ngDoCheck()*
- **ngAfterContentChecked()** - zavolá sa po *ngAfterContentInit()* a každom ďalšom *ngDoCheck()*
- **ngAfterViewInit()** - zavolá sa raz po prvom *ngAfterContentChecked()*
- **ngAfterViewChecked()** - zavolá sa po *ngAfterViewInit()* a po každom nasledujúcom *ngAfteRContentChecked()*
- **ngOnDestroy()** - zavolá sa pred tým ako Angular zničí komponent/direktívu

Šablóny

Každá komponenta má HTML šablónu, ktorá deklaruje, ako sa daná komponenta vykresľuje. Túto šablóna sa definuje buď priamo, alebo cestou k súboru.

Angular rozširuje HTML o ďalšiu syntax, ktorá umožňuje vkladať dynamické hodnoty z komponenty. Angular automaticky aktualizuje vykreslený DOM, keď sa zmení stav komponenty. Príklad využitia tejto funkcie je vkladanie dynamického textu, ako je znázornené na nasledujúcom príklade.

```
<p>{{ message }}</p>
```

Obsah správy pochádza z triedy komponenty.

```
import { Component } from '@angular/core';

@Component ({
  selector: 'hello-world-interpolation',
  templateUrl: './hello-world-interpolation.component.html'
})
export class HelloWorldInterpolationComponent {
    message = 'Hello World!';
}
```

Keď aplikácia načíta komponentu a jej šablónu, bude DOM vyzerať nasledovne:

```
    <p>Ahoj svet!</p>
```

Dvojité zložité zátvorky inštruujú Angular, aby interpoloval obsah v nich.

Angular taktiež podporuje väzby vlastnostní, ktoré pomáhajú nastaviť hodnoty pre vlastnosti a atribúty prvkov HTML a odovzdať hodnoty prezentačnej logike aplikácie.

```
<p
  [id]="sayHelloId"
  [style.color]="fontColor">
  You can set my color in the component!
</p>
```

Hranaté zátvorky označujú, že vlastnosť alebo atribút je spojený s hodnotou v triede komponenty.

Podobne sa obyčajné zátvorky používajú pre priraďovanie funkcií k udalostiam:

```
<button
  type="button"
  [disabled]="canClick"
  (click)="sayMessage()">
  Trigger alert message
</button>
```

Pri kliknutí na button sa vyvolá nasledujúca funkcia definovaná v triede komponenty:

```
sayMessage() {
    alert(this.message);
}
```

3.2 React

React [11] je deklaratívna a flexibilná JavaScriptová knižnica pre tvorbu užívateľských rozhraní. Bol vytvorený spoločnosťou Facebook v roku 2013. Jeho účel bol ako nástroj na jednoduchý manažment internetových reklám. Rozsahom a komplexnosťou je veľmi odlišný od Angularu. Nejedná sa o rámec ale o rozsiahlu knižnicu, ktorá obsahuje veľké množstvo podporných knižníc umožňujúcich ľahký vývoj aplikácií. React je teda možné pridať do existujúceho projektu napísaného pomocou iného nástroja.

Samotná knižnica React sa zameriava na prezentačnú vrstvu, takže funkcionalitu ako získavanie dát alebo navigácia je potrebné doplniť knižnicami tretích strán.

Líši sa spojením HTML s logikou náhľadu pomocou technológie JavaScript XML, v skratke JSX [7]. JSX je syntaktické rozšírenie pre JavaScript, ktoré produkuje prvky, ktoré sú kombináciou kódu HTML a reťazcov. Tieto prvky dovoľujú priraďovať do ľubovolných premenných hodnoty obsahujúce prvky HTML. React využívá virtuálny DOM podobne ako Vue.js, čo umožňuje úpravu len potrebnej časti DOM. Pri Reacte sa nepoužíva životný cyklus aplikácie ako pri Angular a Vue.js.

Komponenty

Rovnako ako pri Angular sú hlavným stavebným blokom Reactu komponenty. Komponenty preberajú parametre nazvané **props** a vracia hierarchiu zobrazení, ktoré sa zobrazia prostredníctvom render metódy. Render metóda vracia popis, ktorý React preberia a zobrazí výsledok na obrazovke.

3.3 Vue.js

Vue [14] je JavaScriptový rámec pre tvorbu užívateľských rozhraní. Zakladá si na HTML, CSS a JavaScripte a poskytuje deklaratívny a komponentný programovací model pre efektívne rozvíjanie užívateľského rozhrania rôznych škál.

Dve základné funkcie Vue sú:

- **Deklaratívne vykresľovanie**: Vue rozširuje štandardné HTML o syntax šablóny, ktorá nám umožňuje deklaratívne popísať výstup HTML na základe stavu JavaScriptu.
- **Reaktivita**: Vue automaticky sleduje zmeny stavu JavaScriptu a efektívne aktualizuje DOM, keď dôjde k zmenám.

Jednosúborové komponenty

Vo väčšine prípadoch vytvárame komponenty Vue pomocou formátu súboru podobného HTML, ktorý sa nazýva po anglicky **Single-File Component**, v preklade jednosúborové komponenty, ide o súbory s príponou *.vue. Jedná sa o súbor zahŕňajúci logiku komponenty(JavaScript), šablónu(HTML), a štýly(CSS) do jedného súboru. Nasledujúca ukážka je prevzatá z článku o Vue [14].

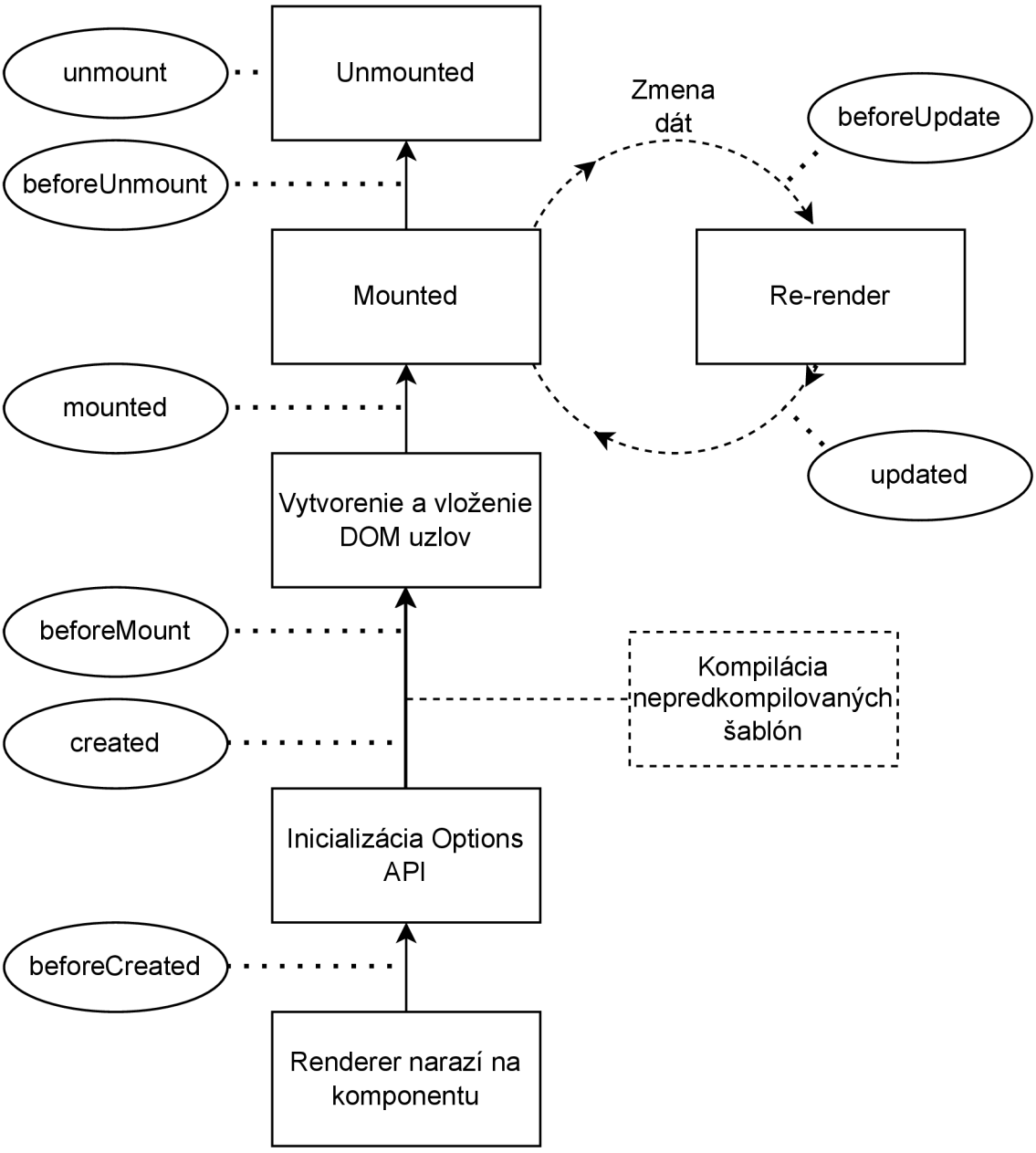
```
<script>
export default {
  data() {
    return {
      count: 0
    }
  }
}
</script>

<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

Životný cyklus Vue

Každá komponenta Vue pri svojom vytvorení prechádza sériou inicializačných krokov, ako zostavenie šablóny, nastavenie pozorovania údajov, pripojenie inštancie k DOM a aktualizovanie DOM pri zmenách. Taktiež spúšťanie funkcií životného cyklu [15], ktoré dávajú užívateľom možnosť pridať svoj vlastný kód v konkrétnych fázach. Tieto funkcie sa nazývajú po anglicky lifecycle hooks, ďalej ako fázy životného cyklu.



Obrázek 3.1: Graf fáz životného cyklu

Fázy životného cyklu, ktoré môžeme v komponente využiť pre spustenie potrebného kódu, sú v obrázku 3.1 označené elipsami. Sú to beforeCreated, created, beforeMount, mounted, beforeUpdate, updated, beforeUnmount, unmount.

Kapitola 4

Použité webové technológie

Táto kapitola vysvetľuje základné pojmy pri práci s vytváraním webových stránok a technológie použité pre implementáciu. Mimo tieto technológie a nástroje sa ako základ aplikácie používa Vue 3, ktorý je vysvetlený v kapitole 3.3.

4.1 HTML

HyperText Markup Language [5] je najzákladnejším stavebným blokom webu. Definuje význam a štruktúru webového obsahu. Pojem HyperText označuje odkazy, ktoré navzájom spájajú webové stránky, či už v rámci jednej webovej lokality alebo medzi webovými lokalitami. Odkazy sú základným aspektom webu. Odovzdaním obsahu na internet a jeho prepojením so stránkami vytvorenými inými ľudmi so stávate aktívnym účastníkom World Wide Webu. Používa značky na anotovanie obsahu na zobrazenie vo webovom prehliadači.

4.2 DOM

DOM je skratka pre Document object model [4], v preklade objektový model dokumentu. Je to dátová reprezentácia objektov, tvoriace štruktúru a obsah dokumentu na webe. Ide o programovacie rozhranie pre webové dokumenty. Predstavuje stránky ako dokument z uzlov a objektov, programy môžu za jeho pomoci meniť štruktúru, štýl a obsah dokumentu.

Vlastnosti, metódy a udalosti pre manipuláciu webovej stránky sú usporiadané do objektov. Jedným z týchto objektov je napríklad `document`. DOM je vytvorený z viacero spolupracujúcich API.

4.3 CSS

Cascading Style Sheets (CSS) [3] je jazyk pre šablóny so štýlmi, ktoré sa používajú pre popis prezentovania dokumentu napísaného v HTML alebo XML. Popisuje ako by sa prvky mali vykresliť, ako napríklad zmenu písmu, farby, rozloženie obsahu a podobne. Patrí medzi základné jazyky otvoreného webu a je štandardizovaný vo webových prehliadačoch.

4.4 JavaScript

JavaScript [6] je multiplatformový, objektovo orientovaný skriptovací jazyk, ktorý sa používa na vytváranie interaktívnych webových stránok. Existujú pokročilejšie verzie JavaScriptu

na strane servera ako napríklad Node.js, ktoré umožňujú pridať na webovú stránku viac funkcií. Vo vnútri webového prehliadača môže byť JavaScript prepojený s objektami jeho prostredia, aby sa nad nimi zabezpečila programová kontrola.

JavaScript obsahuje štandardnú knižnicu objektov, ako sú Array, Date a Math a základnú sadu jazykových prvkov, ako sú operátory, riadiace štruktúry a príklady. JavaScript na strane klienta rozširuje základný jazyk poskytovaním objektov na ovládanie prehliadača a jeho modelu objektu dokumentu (DOM 4.2). Toto umožňuje aplikácii umiestniť prvky do formulára HTML a reagovať na udalosti používateľa, ako sú kliknutia myšou, zadávanie formulárov a navigácia po stránkach. Pri práci s JavaScriptom sa využili základné praktiky z literatúry od O. Žáru[16].

4.5 CodeMirror

CodeMirror[1] je komponenta editora dotazov. Základná knižnica poskytuje iba komponentu editora bez sprievodných tlačidiel, automatického dokončovania alebo iných funkcií. Poskytuje bohaté rozhranie API, na základe ktorého je možné takúto funkčnosť jednoducho implementovať. Obsahuje taktiež veľa doplnkov, ktoré sa dajú dodatočne importovať, ktoré túto funkcionalitu dosahujú, ako napríklad témy pre užívateľské rozhranie.

4.6 PrimeVue

PrimeVue[2] je knižnica zameraná na tvorbu užívateľského rozhrania. Je vytvorená open source poskytovateľom užívateľských rozhraní, PrimeFaces[3], ktorý vytvárajú užívateľské rozhranie pre Angular, React, Vue a aj základný JavaScript. PrimeVue obsahuje plno pred vytvorených komponent, ktoré sme schopný nakonfigurovať podľa našich potrieb. Tieto komponenty majú vlastné udalosti, ktoré môžeme zachytávať, metódy ktoré môžeme vyvolať pre spustenie danej funkcionality a vlastnosti, ktorých nastaveniami môžeme komponentu ovplyvniť. Všetky tieto komponenty sa taktiež riadia na základe témy, ktorú si môžeme vybrať v dokumentácií a importovať jej štýl do nášho projektu.

4.7 Axios

Axios je izomorfný HTTP klient pre node.js aj prehliadač. Tvorí XMLHttp požiadavky z prehliadača. Umožňuje nám poslať rôzne požiadavky a zachytiť odpovede, teda získavať dáta z backend pre frontend, ako napríklad dáta v podobe formátu JSON. V ukážkovej aplikácií sa využíva spolu s vue-axios obalením dostupným ako npm balík[4], ktoré viaže samotný axios na vue inštanciu.

4.8 NPM

NPM [9] je najväčší register softvéru na svete. Je používaný open source vývojármi po celom svete za účelom zdieľania balíkov. Taktiež mnoho organizácií používa npm pre ri-

[1] https://codemirror.net/
[2] https://www.primefaces.org/primevue/
[3] https://www.primefaces.org/
[4] https://www.npmjs.com/package/vue-axios

adenie súkromného vývoja. Pod npm sa rozumie 3 pojmom: Samotnej webovej stránke[5], rozhraní príkazového riadka (CLI) a registra obsahujúceho balíky. Webová stránka umožňuje prehľadávať všetky verejne dostupné balíčky a spravovať vlastný profil ako aj sledovať vlastné balíčky. Za pomoci rozhrania príkazového riadka sme schopný komunikovať s registrom pre sťahovanie a inštaláciu balíčkov do nášho projektu alebo aj globálne do nášho terminálu, zverejňovanie našim balíčkov, ich spravovanie veľa ďalších funkcionalít.

4.9 Vite

Vite [13] je nástroj pre zostavovanie, ktorého cieľom je poskytnúť rýchlejší vývoj pre moderné webové projekty. Pozostáva z 2 hlavných funkcionalít. Prvou je vývojový server, ktorý prináša vylepšenia funkcií oproti natívnym modulom ES. Druhou funkcionalitou, ktorú využívame v našom projekte, je zostavovanie kódu pre produkciu za pomoci knižnice RollUp[6]. Zjednodušuje konfiguráciu výstupu pre produkciu, namiesto konfigurácie RollUp, konfigurujeme samotný Vite.

[5] https://www.npmjs.com/
[6] https://rollupjs.org

Kapitola 5

Návrh webového editora a architektúri aplikácie

V tejto kapitole sa na začiatku nachádzajú ukážky existujúcich riešení ako aj komponenty editora tak aj aplikácie. Následne sa venuje postupne návrhu editora a potom aplikácie na základe nedostatkov existujúcich riešení. Súčasťou návrhu sú obrázky rozloženia užívateľského rozhrania a funkcionality spolu s ich odôvodnením. Pri aplikácií je taktiež popis jej architektúry.

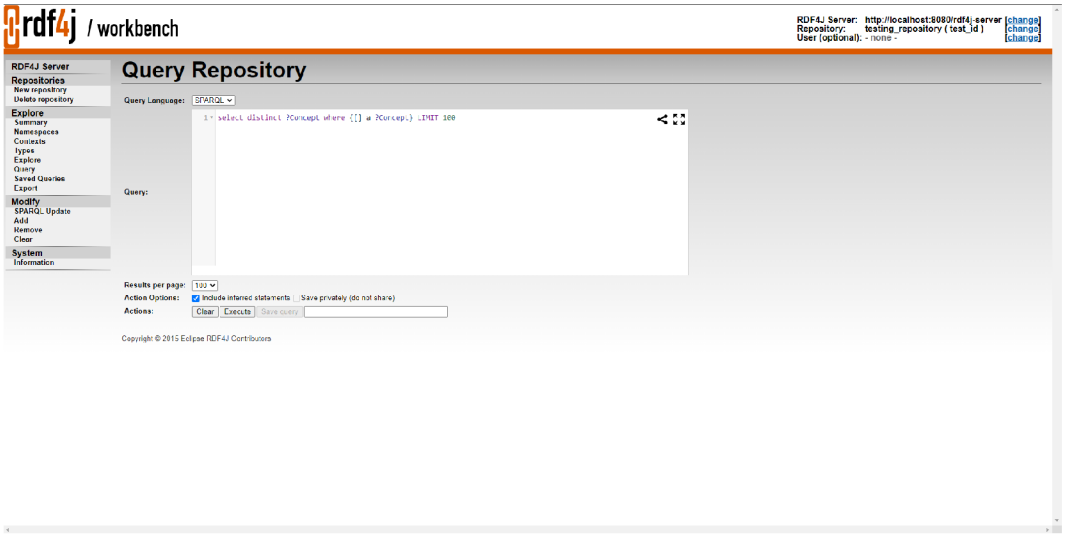
5.1 Existujúce riešenia

Táto podkapitola sa zaoberá existujúcimi riešeniami, ich popisom, prácou s nimi a ich nedostatky, ktoré sa neskôr využijú v nasledujúcich podkapitolách. V tejto kapitole sa najskôr prechádzajú existujúce riešenia aplikácií a následne editory, ktoré tieto aplikácie využívajú pre písanie dotazov.

RDF4J

Eclipse RDF4J[1] je open source modulárny Java framework pre prácu s údajmi. To zahŕňa analýzu, ukladanie, odvodzovanie a dopytovanie takýchto údajov. Ponúka API, ktoré možno pripojiť ku všetkých popredným riešeniam ukladania RDF. Umožňuje pripojenie sa ku koncových bodom SPARQL a vytvárať aplikácie.

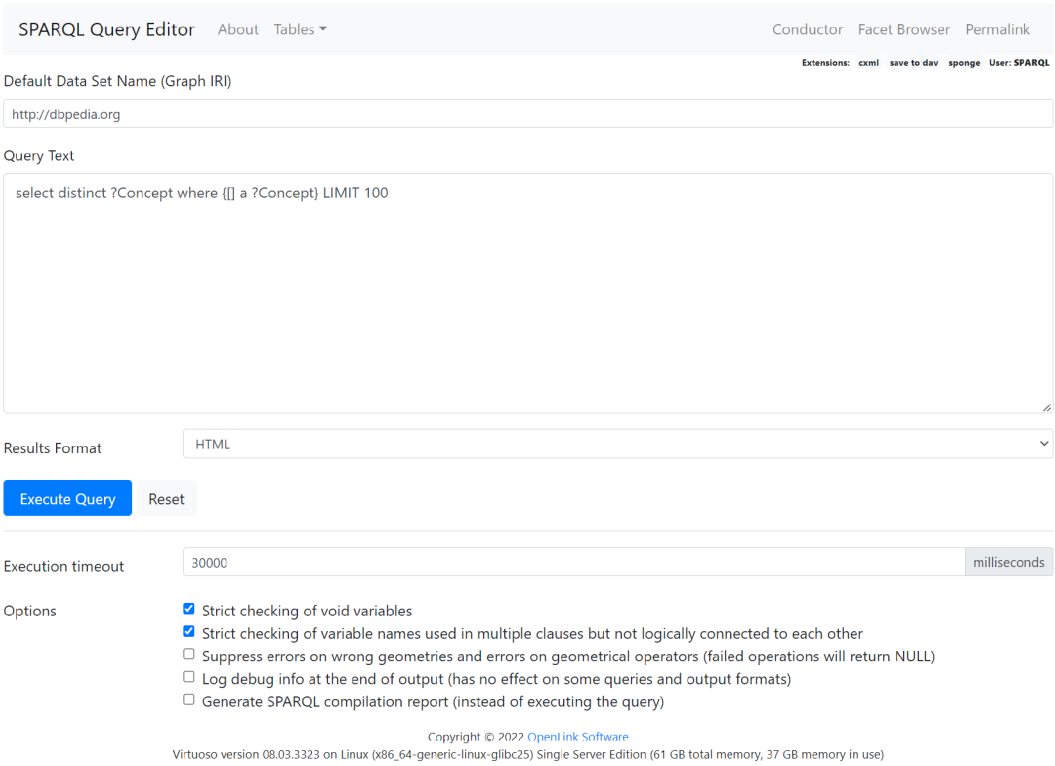
[1] https://rdf4j.org/



Obrázek 5.1: Ukážka rdf4j workbench

V obrázku 5.1 je zobrazený editor pre posielanie dotazov na konkrétne vybraté úložisko v RDF4J workbench. Editor ma zvýraznenie SPARQL syntaxe a očíslovanie riadkov. Obsahuje funkcionalitu zväčšenia do režimu celej obrazovky a ukladania dotazov. Podporuje ukladanie dotazov pod zvoleným názvom pre budúce použitie. RDF4J využíva pre editor CodeMirror komponentu podobne ako naše riešenie, ale bez akýchkoľvek vylepšení. Komponenta samotná sa taktiež nedá nikde sama o sebe získať pre využitie v projektoch.

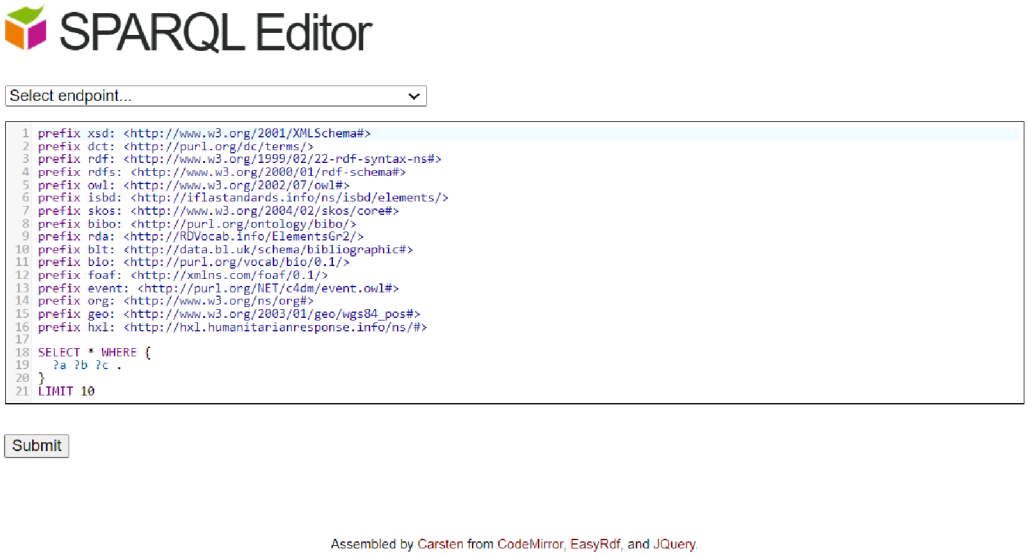
https://dbpedia.org/sparql/



Obrázek 5.2: Ukážka Dbpedia SPARQL query editora

Prvý výsledok, ktorý vyskočí pri vyhľadávaní *SPARQL online editor* v Google prehliadači. Ako na obrázku 5.2 vidieť, aplikácia funguje ideálne pre posielanie samotných dotazov a je možné si vybrať spôsob zobrazených dát. V čom je aplikácia nedostatočná je samotných editor pre písanie dotazov, ktorý je len obyčajná html text area neobsahujúca žiadne quality of life funkcionality.

http://sparql.carsten.io/

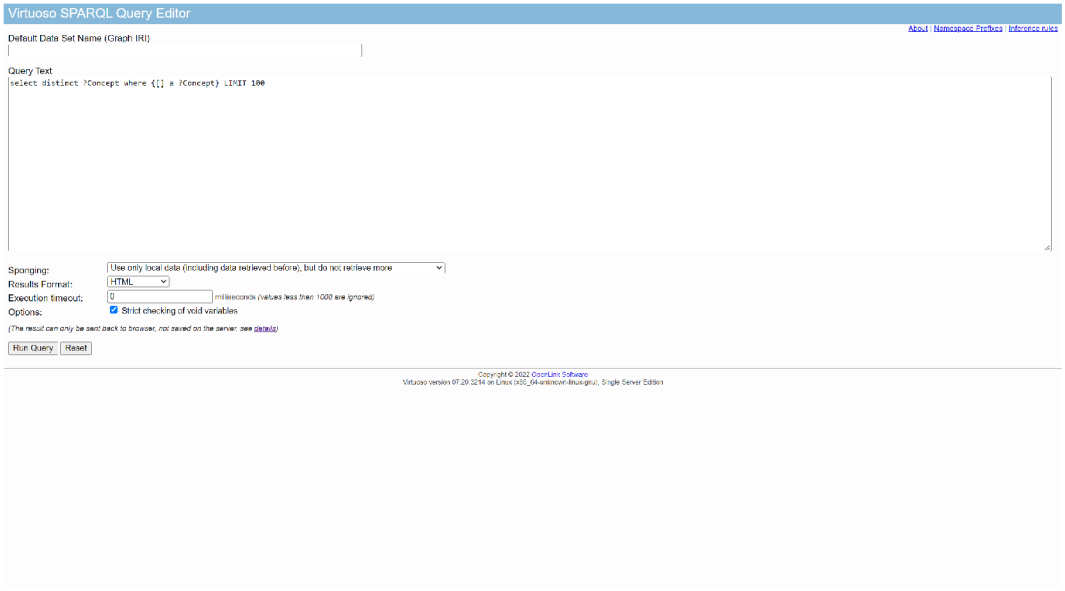


Obrázek 5.3: Ukážka Carsten SPARQL query editora

Myslenou funkcionalitou je táto aplikácia najpodobnejšou tej našej. Je možné zadávať vlastnú adresu SPARQL endpointu, na ktorú sa pošle dotaz. Je vytvorená pomocou knižnice EasyRdf[2]. Žiaľ po testovaní posielaní dotazov na hociktorý SPARQL endpoint, vždy vráti rôzne chybové hlášky. Editor, ktorého ukážka ja na obrázku 5.3 obsahuje zvýraznenie syntaxe a očíslovanie riadkov, a opäť sa jedná o komponentu knižnice CodeMirror.

[2] https://www.easyrdf.org/

https://www.orpha.net/sparql



Obrázek 5.4: Ukážka Orpha SPARQL query editora

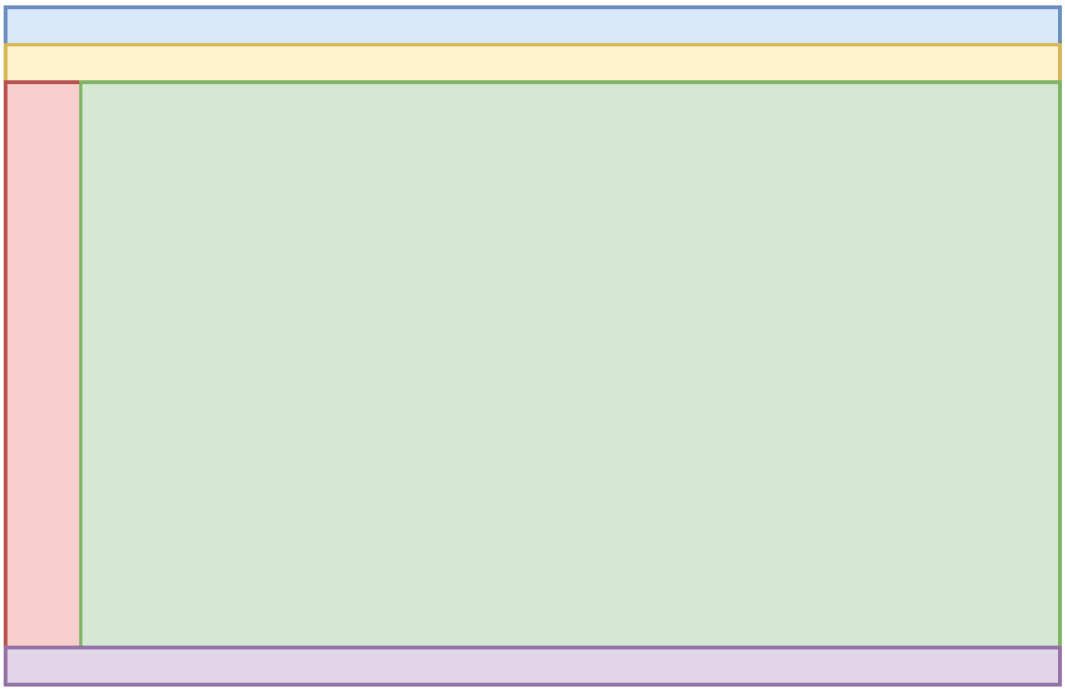
Riešenie podobné riešeniu dbpedia 5.1. V ukážke 5.4 vidieť, že aj editor, aj aplikácia sa dá popísať tými istými slovami.

5.2 Návrh editoru

Táto časť sa venuje komponente editora. Výsledok jej implementácie je NPM 4.8 balík importovateľný do iných projektov. Jej primárnym cieľom je zlepšiť písanie dotazov v jazyku SPARQL. Najzákladnejšími požiadavkami sú zvýraznenie a doplňanie kľúčových slov.

5.2.1 Návrh užívateľského rozhrania

Cieľom užívateľského rozhrania komponenty je podobať sa programátorskému IDE ako je napríklad Visual Studio Code. Mala by obsahovať menu pre otváranie okien s rôznou funkcionalitou ako je spravovanie prefixov a pod., panel pre prepínanie medzi kartami dotazov, očíslovanie riadkov na aktuálnej karte, miesto pre písanie dotazu a spodný panel obsahujúci informácie pre užívateľa. V obrázku 5.5 by sa panel s funkcionalitou nachádzal v modrej časti, prepínanie kariet v žltej, očíslovanie riadkov v červenej, miesto pre písanie dotazu v zelenej a napokon vo fialovej by bolo vypísané informácie, medzi ktoré sú naplánované aktuálne číslo riadku a pozície na ňom.



Obrázek 5.5: Návrh rozloženia užívateľského rozhrania editora

5.2.2 Spravovanie prefixov

Pri písaní dotazov sa prefixy vždy definujú na začiatku dotazu, a môže sa zdať užívateľom stratou času si vždy otvoriť nejaký textový súbor kde ich má uložené a prekopírovať ich do dotazu.

Cieľom tejto časti editora je ukladanie prefixov pre budúce použitie a následné jednoduché vkladanie prefixov do dotazov. Myšlienka je mať možnosť vytvárať skupiny prefixov a vkladanie prefixov do týchto skupín. Následne mať možnosť vložiť celú skupinu alebo len konkrétny prefix zo skupiny do dotazu pomocou užívateľského rozhrania. Pre zobrazenie prefixov, v konkrétnych skupinách by bol najideálnejší dátový zoznam, kde pre každý riadok by bol zobrazený namespace a uri pre daný prefix. Konkrétny prefixy by mali byť upraviteľné a vymazateľné zo zoznamu. Zoznamy by sa mali dať vytvárať pod konkrétnym názvom a taktiež vymazateľné.

5.2.3 Prepínanie kariet

Pri dotazovaní na SPARQL endpointy je často krát potrebné posielať rovnaké dotazy znova napríklad po aktualizácií dát. Tu nastáva zdĺhavý proces prepísania/prekopírovania dotazu pre získanie výsledkov po tom ako sme poslali dotaz pre aktualizáciu. Tento problém rieši rdf4j workbench s ukladaním dotazov pod rôznymi názvami. Ideálnejší spôsob riešenia je ako ponúkajú rôzne programovacie IDE, mať otvorených niekoľko kariet kódu naraz a prepínania medzi nimi.

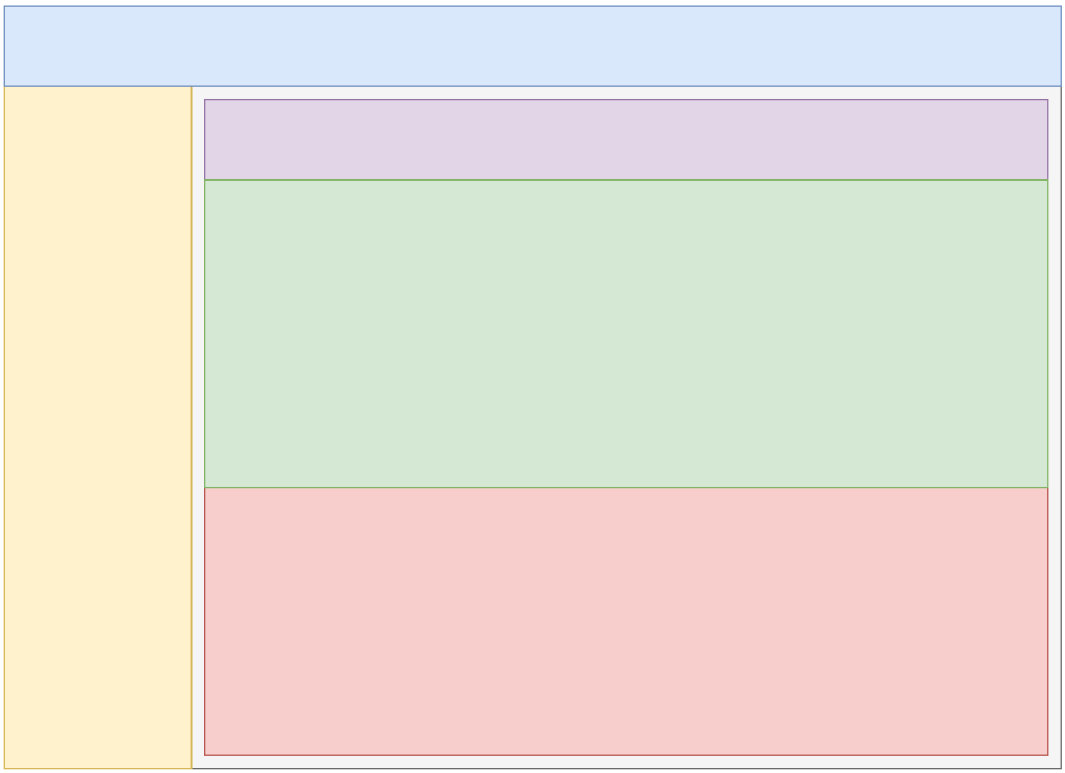
5.2.4 Spravovanie preferencii

Spravovanie preferencií by malo byť samostatné okno možné otvoriť z menu editora. Malo by obsahovať možnosť zmeny CodeMirror témy a miesto pre budúcu podobnú funkcionalitu pre užívateľské nastavenia.

5.3 Aplikácia

Primárnym cieľom aplikácie je demonštrácia práce s komponentou editora. Aplikácie využije editor pre napísanie dotazu v jazyku SPARQL a prepošle ho na žiadaný SPARQL endpoint podľa zadanie adresy a ukáže výsledky daného dotazu.

5.3.1 Rozloženie užívateľského rozhrania



Obrázek 5.6: Návrh rozloženia užívateľského rozhrania aplikácie

V obrázku 5.6 sa nachádza návrh užívateľské rozhrania pre našu aplikáciu. Základom každej aplikácie je navigácia. Menu pre prepínanie medzi konkrétnymi náhľadmi sa bude nachádzať na boku,v zóne označenej na obrázku v žltej farbe. Modrá zóna je vyznačená pre banner, obrázok alebo nadpis web stránky. Taktiež by sa dala využiť pre druhé menu obsahujúce rôzne funkcionalitu alebo taktiež navigáciu. V našom prípade bude najskôr obsahovať len názov webovej aplikácie. Okrem samotnej navigácie je druhým najdôležitejších prvkom užívateľského rozhrania miesto pre zobrazenie samotného návrhu, a to je v obrázku vyznačené sivou farbou, obsahujúci rovno ukážku návrhu náhľadu editora.

Náhľad editora bude obsahovať v zelenej farbe komponentu editora, ktorej návrh užívateľského rozhrania sa vysvetľuje v 5.2.1. Vo fialovej farbe sa bude nachádzať lišta pre zadanie

adresy cieľového SPARQL endpointu, na ktorý sa bude posielať dotaz spolu s tlačítkom pre poslanie dotazu. Po stlačení tohto tlačítka sa zobrazí nové okno, označené červenou farbou, ktoré bude obsahovať výsledky dotazu.

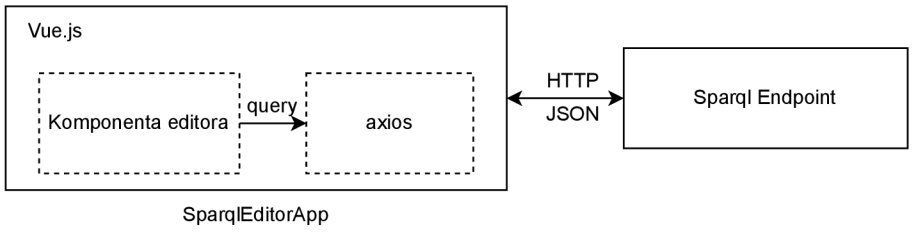
5.3.2 Navigácia

Naša aplikácia obsahuje len 2 náhľady. Jeden pre samotný editor, ktorý je primárnym cieľom aplikácie a druhý náhľad, s názvom About, obsahujúci popis projektu a odkazy na zdrojové súbory ako aj komponenty editora, tak aj aplikácie.

Pre prepínanie náhľadu a teda samotnej navigácie po webovej stránke sa využíva vue router. Ten funguje na základe definovania súbora *index.js* v zložke *router*. Tu po importovaní potrebných pluginov pre funkciu samotného routera definujeme navigáciu pre našu aplikáciu. Náhľad editora je nastavený priamo na /, čo znamená, že po zadaní adresy našej aplikácie, sa priamo zobrazí tento náhľad. Náhľad **About** sa zobrazí po zadaní */about* za adresou našej aplikácie.

5.3.3 Architektúra aplikácie

Architektúra aplikácie spočíva vo frontendovej aplikácií posielajúcej dotazy napísaných v komponente editora, ktoré sa za pomoci knižnice Axios posielajú ako požiadavky na verejné SPARQL endpointy umožňujúce prístup. Tieto SPARQL endpointy nám na požiadavky odpovedia s výsledkami dotazu vo forme JSONu, ktorý aplikácia spracuje a ukáže výsledok.



Obrázek 5.7: Návrh architektúry aplikácie

Kapitola 6

Implementácia

V tejto kapitole sa popisuje čitateľovi samotná implementácia. Postupne popíše prácu s nástrojmi ukázanými v kapitole 4. Prvým bodom implementácie bolo oboznámenie sa s npm pre inštaláciu nástrojov, ktoré sa využili aj pri vytvorení projektu.

Inštalácia balíčkov pomocou npm sa vykonáva pomocou príkazu:

```
npm install <názov balíčka>
```

Pri pridaní parametru -g sa nainštaluje balíček globálne do nášho systému a nie do projektu. Pre odstránenie balíčka sa využíva príkaz *uninstall*, a pre aktualizovanie všetkých nainštalovaných balíčkov v projekte *npm update*. Pred prvým spustením projektu je nutné zavolať *npm install* pre stiahnutie a vygenerovanie všetkých knižníc, na ktorých projekt závisí. Po tom môžeme spustiť vývojársky server za pomoci príkazy *npm run serve*, kde *serve* je preddefinovaný skript v súbore *package.json*. Tento server automaticky reaguje na zmeny vykonané v kóde a vždy našu aplikáciu aktualizuje, čo napomáha rýchlemu vývoju.

6.1 Vytvorenie projektu

Pre vytvorenie projektu s JavaScriptových rámcom Vue.js je najskôr potrebné nainštalovať Vue CLI globálne za pomoci NPM. Pri inštalácií Vue CLI sme použili -g parameter a názov balíčka **@vue/cli**. Vue CLI je úplný systém pre rýchlo vývoj Vue.js. Vue CLI zabezpečuje, aby rôzne nástroje na zostavovanie fungovali spolu. Vue CLI poskytuje príkaz vue v terminály.

Projekt sa za pomoci vue príkazu vytvára pomocou príkazu:

```
vue create <názov projektu>
```

Tento príkaz vytvorí základnú štruktúru projektu spolu s jednoduchou ukážkou HelloWorld komponenty. Pre nás v tomto bode dôležité 3 súbory. Súbor *main.js* sa využíva ako vstupný bod celého projektu. V ňom registrujeme všetky komponenty globálne, a zároveň doň importujeme súbor *App.vue*, v ktorom vytvárame inštancie samotného Vue. Pri implementácií komponenty editora sme s projektom počas vývoja narábali rovnako ako s aplikáciou aby sme boli náš kód schopný testovať na vývojovom serveri.

6.2 Editor

Táto sekcia kapitoly sa venuje implementácií komponenty editora. Postupne prechádza spôsoby implementácie každej funkcionality.

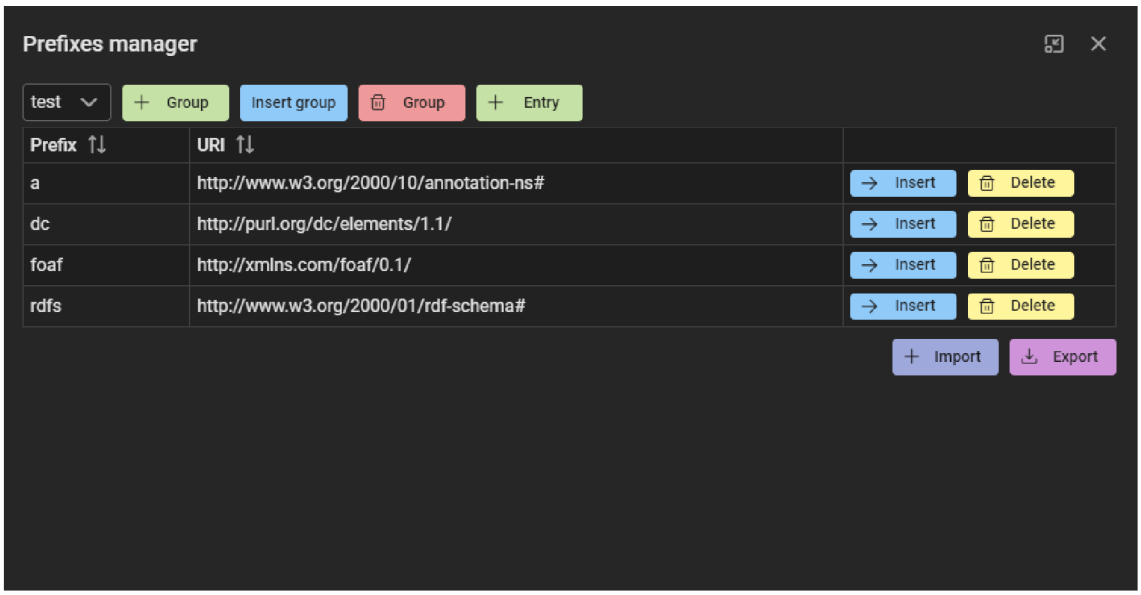
6.2.1 CodeMirror

Pre vytvorenie inštancie CodeMirror knižnice, musíme mať najskôr html prvok textovej arei. Následne vyvoláve funkciu CodeMirror knižnice *fromTextArea()*, ktorá berie ako parameter prvok text arei z DOM dokumentu a nastavenia pre túto inštanciu. Medzi základné nastavenia patria zapnutie očíslovania riadkov, téma, mód, skratka pre vyvolania automatická dopĺňania, ktorá je v našom prípade CTRL+Space a podobne.

Automatické dopĺňanie

Automatické dopĺňanie sa rieši pomocou pred pripraveného módu *Anyhint*, ktoré svojou funkcionalitou dopĺňa znova použité výrazy, ako napríklad skorej v kóde sa napíše slovo select, tak pri druhom písaní tohto slova po stlačení skratky CTRL+Space, ponúkne tento výsledok pre automatické dopĺňanie. My túto základnú funkcionalitu odstraňujeme a nahradzujeme naším vlastným listom kľúčových slov, ktoré máme definované všetky veľkými písmenami a následne tento list kopírujeme a vytvoríme list obsahujúci kľúčové slová s malými písmenami a tieto listy spojíme. Kód pre prepísanie tohto Anyhint módu, bol prevzatý z verejnej diskusie na stack overflow [1] a následne pozmenený.

6.2.2 Spravovanie prefixov

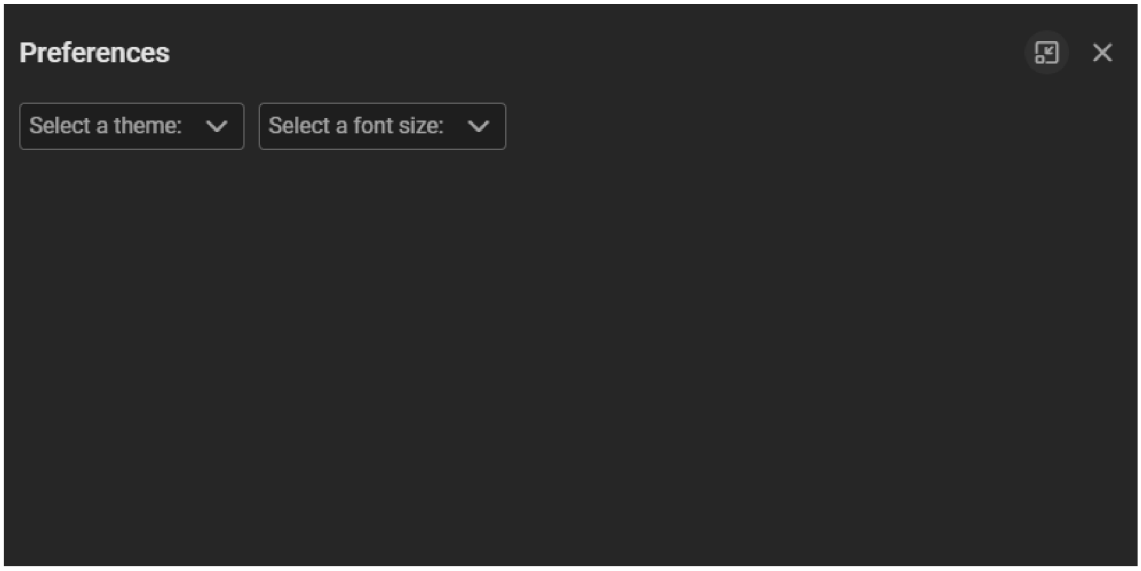


Obrázek 6.1: Ukážka okna spravovania prefixov

[1] https://stackoverflow.com/questions/19244449/codemirror-autocomplete-custom-list

6.2.3 Okno preferencií

Okno preferencií, vytvorené rovnaký spôsobom ako okno spravovania prefixov za pomoci komponenty dialógového okna z PrimeVue obsahuje dropdowny pre zmenu témy a fontu. Dropdown tém obsahuje všetky tému, ktoré CodeMirror ponúka. Táto implementácia vyžadovala importovanie všetkým súborov so štýlmi, kde každá téma má jeden súbor. Následne sa téma mení pomocou premennej CodeMirror



Obrázek 6.2: Ukážka okna spravovania preferencií

6.2.4 Ukladanie dát

Pre ukladanie dát sa využíva lokálne úložisko [8] v prehliadači. Lokálne úložisko spolu s úložiskom relácie majú možnosť ukladať dáta do prehliadača. Rozdiel medzi nimi je, že úložisko relácie sa vymaže po ukončení relácie, teda po zatvorení stránky. Lokálne úložisko zostane v prehliadači dokým nie je manuálne vymazané, v prípade prehliadania v súkromnom/anonymnom okne, sa lokálne úložisko vymaže po zatvorení prehliadača.

Pre prístup k lokálnemu úložisku využívame nasledujúci príkaz pre ukladanie objektu pod voliteľným názvom:

```
window.localStorage.setItem('name', <object>)
```

A a na podľa zvoleného názvu objekt z lokálneho úložiska môžeme načítať za pomoci príkazu:

```
window.localStorage.getItem('name')
```

Je nutné podotknúť, že objekt sa ukladá ako JSON, pred vložením ho do lokálneho úložiska ho prerobíme pomocou JSON funkcie *JSON.stringify(<názov objektu>)* a po načítaní získame pôvodný hodnotu pomocou funkcie *JSON.parse(<názov objektu>)*.

6.2.5 Zabalenie komponenty pomocou Vite pre NPM

Pre sprístupnenie komponenty ako NPM balík musíme aplikáciu zabaliť pre rôzne výstupy. Vhodným nástrojom pre toto je Vite využívajúci RollUp.js.

Prerobenie aplikácie editora do pluginu

Pre využitie nástroja vite na zabalenie komponenty musíme mať nejaký vstupný bod pre našu komponentu, pozostávajúcu z 3 komponent a využívajúcu rôzne externé pluginy a komponenty. Náš vstupný bude je súbor *index.js* nachádzajúci sa v zložke *src*. Obsahuje importy daných externých pluginov a komponent, podobne ako pri aplikácií, keď ich registrujeme globálne. Na rozdiel toho, že nevytvárame konštantnú premennú app z komponenty App.vue, za pomoci ktorej potom registrujeme komponenty globálne, ale vytvárame export pre inštaláciu komponenty, obsahujúci kód, ktorý sa spustí keď v cieľovej aplikácií, zavoláme **app.use('NázovPluginu')**. V tomto exporte registrujeme všetky pre nás potrebné externé pluginy a komponenty globálne, a taktiež globálne registrujeme našu komponentu. K registrácií externých pluginov dochádza z dôvodu, že sa pri implementácií nenašiel spôsob ako registrovať plugin **PrimeVue** lokálne, a daný plugin je potrebný pre všetky komponenty z tejto knižnice pre užívateľské rozhranie.

Konfigurácia vite

Na základe konfiguračného súboru *vite.config.js* sa vytvárajú súbory *vue-sparql-editor.umd.js* a *vue-sparql-editor.es.js*, ktoré sa zverejňujú ako npm balík. Tieto súbory sa ďalej nastavujú v konfigurácií balíka ako exporty. V tomto súbore sa nastavil vstupný bod pre náš export, jeho názov a definujú sa v ňom externé knižnice, ktoré sa nemajú vložiť do výstupného súbora. Medzi tieto ignorované knižnice sa nastavili všetky externé využité knižnice okrem codemirror knižnice, ktorá z neznámeho dôvodu nefungovala správne v zverejnenom balíku správne, pokiaľ bola vložená medzi tieto ignorované knižnice.

```
// vite.config.js
const path = require('path')
const { defineConfig } = require('vite')
const vue = require('@vitejs/plugin-vue')

module.exports = defineConfig({
  build: {
    lib: {
      entry: path.resolve(__dirname, 'src/index.js'),
      name: 'vue-sparql-editor',
      fileName: (format) => `vue-sparql-editor.${format}.js`
    },
    rollupOptions: {
      external: ['vue', 'primevue', 'primeflex', 'primeicons'],
      output: {
        globals: {
          vue: 'Vue',
        },
      },
    },
  },
  plugins:[vue()],
})
```

Po napísaní tohto súboru, po zavolaní príkazu *npm run build* sa vygenerovala zložka dist, obsahujúca súbory vhodné pre zverejnenie ako npm balík.

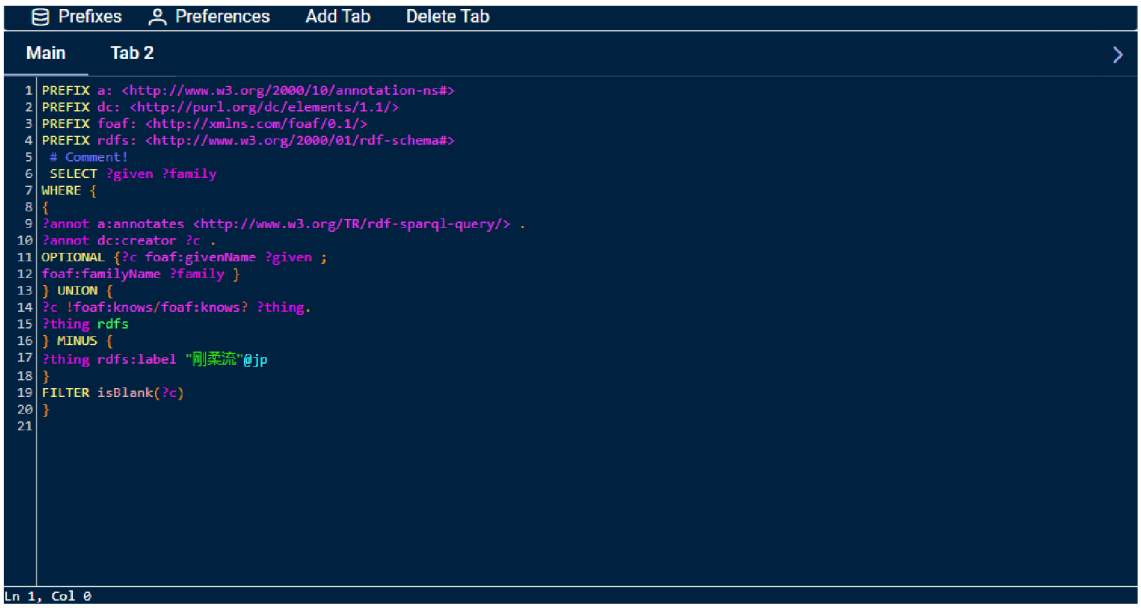
Konfigurácia balíka

Pre zverejnenie npm balíka musíme pridať minimálnu konfiguráciu do súbora *package.json* popisujúcu exporty, verziu, názov a zložku obsahujúcu dané súbory. Pridaná konfigurácia pozostáva z tejto ukážky:

```
"name": "vue-sparql-editor",
"files": [
  "dist"
],
"version": "1.0.5",
"main": "./dist/vue-sparql-editor.umd.js",
"module": "./dist/vue-sparql-editor.es.js",
"exports": {
  ".": {
    "import": "./dist/vue-sparql-editor.es.js",
    "require": "./dist/vue-sparql-editor.umd.js"
  },
  "./dist/style.css": "./dist/style.css"
},
```

6.2.6 Finálna verzia

Výsledná komponenta je dostupná na NPM[2] a jej zdrojový kód na GitHube[3].



Obrázek 6.3: Ukážka finálnej verzie komponenty editora

[2] https://www.npmjs.com/package/vue-sparql-editor
[3] https://github.com/Densik50/vue-sparql-editor

6.3 Aplikácia

Pri implementácií aplikácie sa aplikovali rovnaké kroky ako pri tvorbe komponenty editora pre vytvorenie projektu.

6.3.1 Importovanie komponenty editora do aplikácie

Pre importovanie našej komponenty editora do nášho projektu, ju prv musím nainštalovať rovnakým spôsobom ako všetky ostatné nástroje a knižnice pomocou príkazu *npm install*. Názov nášho pluginu obsahujúceho túto komponentu je **vue-sparql-editor**. Po inštalácií, plugin a jeho štýly importujeme v súbore *main.js* pomocou príkazov:

```
import 'vue-sparql-editor/dist/style.css'
import VueSparqlEditor from 'vue-sparql-editor';
app.use(VueSparqlEditor)
```

Potom môžeme využívať v aplikácie HTML prvok **<SparqlEditor>**. Pre získanie dotazu do aplikácie z tohto prvku, si k tomuto prvku môžeme nastaviť vue referenciu a keď zavoláme pomocou tejto referencie funkciu getCode() na tento prvok, tak nám vráti dotaz aktuálne zvolenej karty editora.

6.3.2 Posielanie dát

Posielanie dát sa rieši pomocou knižnice Axios popísanej bližšie v časti 4.7, táto knižnica dovoľuje jednoducho posielať rôzne požiadavky.

Pri posielaní dát na SPARQL endpointy sa využívajú metódy GET a POST. GET metóda sa používa pre SELECT dotazy kratšie ako 1.9MB a metóda POST pre dotazy väčšie pokiaľ sa jedná o SELECT dotazy a zároveň pre dotazy typu UPLOAD, to jest dotazy pre vkladanie dát ako INSERT DATA alebo odstraňovanie ako DELETE. Naša aplikácia posiela dotazy len za pomoci metódy GET, nakoľko sa pri vytváraní aplikácie nepodarilo nájsť verejný SPARQL endpoint, povolujúci posielať takéto požiadavky.

HTTP požiadavky, ktoré posielame na SPARQL endpointy v našej aplikácií obsahujú payload vo formáte definovanom na wiki stránky pre Virtuoso SPARQL Query Service[4], ktoré väčšina SPARQL endpointov využíva.

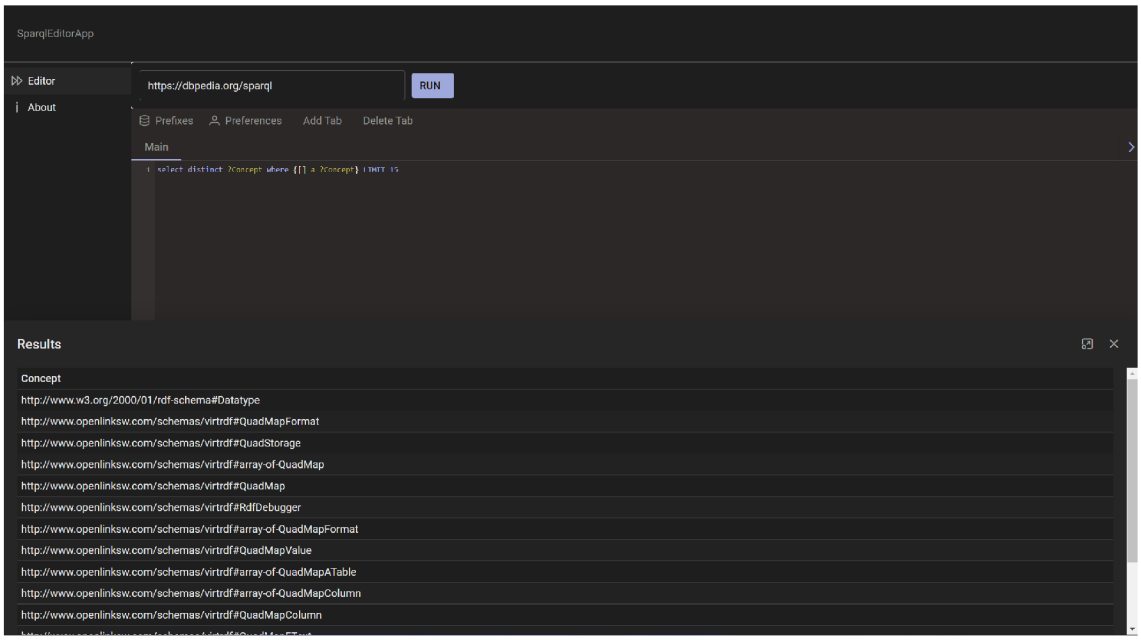
6.3.3 Zobrazovanie dát

Demonštračná aplikácia zobrazuje dáta vrátené odpoveďou typom application/json, teda sa jedná o formát JSON. Vrátenú odpoveď ukladáme pod premennou *resposne*, v ktorej sa samotné dáta nachádzajú pod objektom *data* a ďalej v ňom pod *results*.

Pre efektívne zobrazenie dát využívame dátovú tabuľku podobne ako pri spravovaní prefixov pri komponente editora. Stĺpce tabuľky definujeme dynamicky za pomoci v-for, ktorá vytvorí html prvok alebo v našom prípade vue komponentu pre každý prvok v pol. Dynamicky sa vytvárajú podľa vráteného objektu *head.vars*. Samotné dáta sú v *bindings*.

V prípade chybného dotazu sa nezobrazí žiadna tabuľka ale ani chybová hláška, nakoľko väčšina SPARQL endpointov má rozdielnu chybovú hlášku.

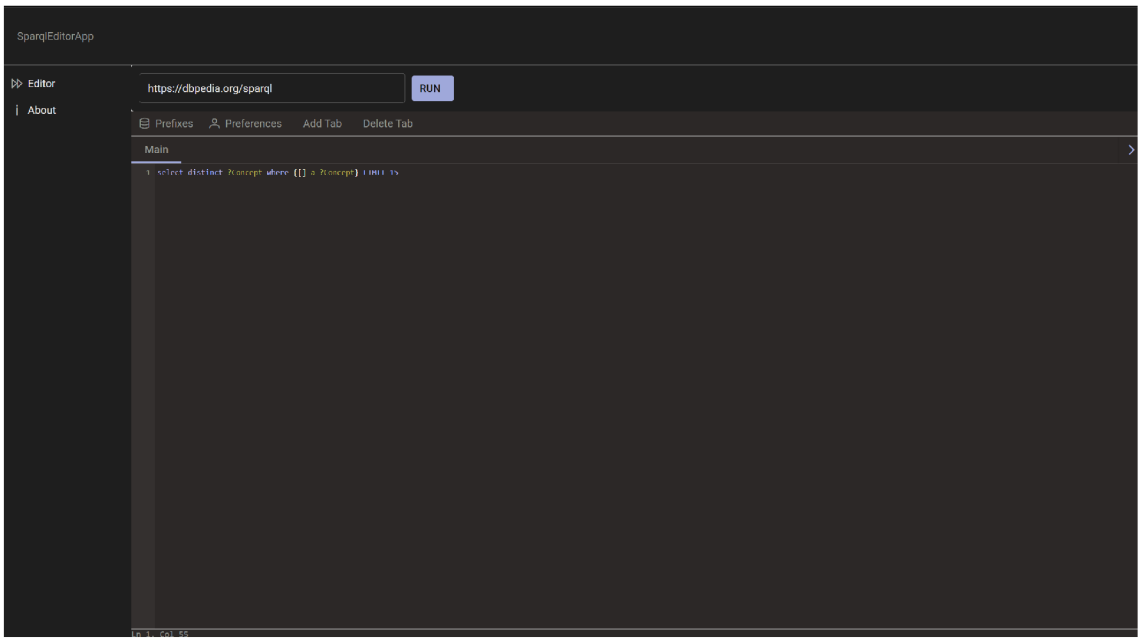
[4] http://vos.openlinksw.com/owiki/wiki/VOS/VOSSparqlProtocol



Obrázek 6.4: Ukážka okna zobrazených dát

6.3.4 Finálna verzia

Finálna verzia aplikácie je dostupná na GitHube[5].



Obrázek 6.5: Ukážka finálnej verzie aplikácie

[5] https://github.com/Densik50/vue-sparql-editor-example

Kapitola 7

Testovanie

Testovanie aplikácie prebieha na lokálnom vývojárskom severy spustenom pomocou príkazu *npm run serve*. Cieľom testovania je overiť funkcionalitu výslednej aplikácie a komponenty editora. Pri testovaní sa zistili, že niektoré SPARQL endpointy, ako napríklad SPARQL endpoint Orpha, ktorého aplikáciu sme si ukazovali v kapitole 5.1, prijíma požiadavky z aplikácie Postman, ale nepovoľuje prístup pre požiadavky poslané z prehliadača z dôvodu nastavenia hlavičky CORS[1]. V rámci testovania aplikácie sa testovalo na jednoduchých dotazov funkčnosť zostavovania dátovej tabuľky a posielanie na rozdielne SPARQL endpointy. V prípade posielania chybných dotazov alebo dotazov, ktoré vyžadujú POST metódu, aplikácia nevypíše žiadnu tabuľku.

7.1 Testovanie komponenty editora

7.1.1 Zvýraznenie syntaxe

V tomto teste sa zaoberá zobrazením zvýraznenie syntaxe textu dotazu v komponente editora. Test prebiehal zadaním dotazu do našej komponenty a následne do nejakého iného riešenia, pre porovnanie.

[1] https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Obrázek 7.1: Ukážka zvýraznenia syntaxe našej komponenty

Obrázek 7.2: Ukážka zvýraznenia syntaxe iného riešenia

7.1.2 Automatické dopĺňanie

V tomto teste sa zaoberá testovaním automatického dopĺňania. V rámci technickej dokumentácie je tu len obrázok otvorenia menu dopĺňania po stlačení klávesovej skratky CTRL+Space po zadaní aspoň 1 znaku daného kľúčového slova.

Obrázek 7.3: Ukážka dopĺňania kľúčových slov

7.2 Testovanie aplikácie

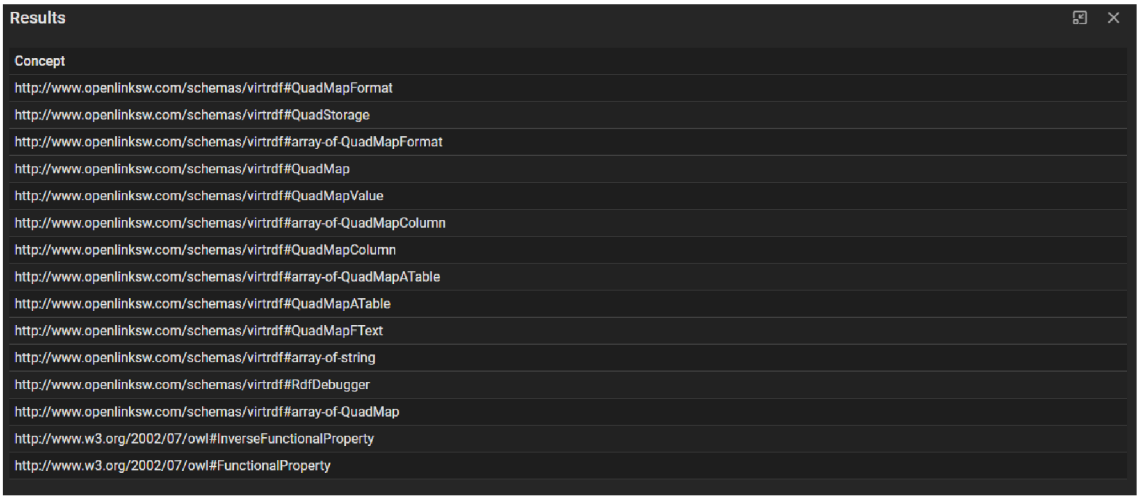
7.2.1 Vytváranie dynamických stĺpcov

Pre otestovanie správnej funkčnosti dynamického zostavovania stĺpcov dátovej tabuľky využijeme 2 dotazy typu SELECT. Oba dotazy sa posielali na dpedia.org/sparql.

SELECT dotaz číslo 1

```
SELECT DISTINCT ?Concept WHERE {[] a ?Concept} LIMIT 15
```

Dotaz č.1 zobrazuje nasledujúcu dátovú tabuľku:



Obrázek 7.4: Ukážka výsledku dotazu 1

SELECT dotaz číslo 2

```
SELECT DISTINCT * WHERE { ?s ?p ?o} LIMIT 15
```

Dotaz č.2 zobrazuje nasledujúcu dátovú tabuľku:

Results

s	p	o
http://www.openlinksw.com/virtrdf-data-formats#default-iid	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#default-iid-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#default-iid-blank	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#default-iid-blank-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#default-iid-nonblank	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#default-iid-nonblank-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#default	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#default-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-dt	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-dt-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-fixeddt-x-stub	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-fixeddt-x-stub-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-lang	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat

Obrázek 7.5: Ukážka výsledku dotazu 2

7.2.2 Posielanie dotazov na rozdielne SPARQL endpointy

Pre toto testovanie môžeme taktiež využiť dotaz:

```
SELECT DISTINCT * WHERE { ?s ?p ?o} LIMIT 15
```

Dbpedia

Posielanie dotazu na SPARQL endpoint `dpedia.org/sparql`.

Výsledok je rovnaký ako pri teste 7.6.

Southampton

Posielanie dotazu na SPARQL endpoint `https://sparql.data.southampton.ac.uk/`.

Results

s	p	o
http://id.southampton.ac.uk/generic-products-and-services/ATM	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.org/goodrelations/v1#ProductOrServicesSomeInstancesPlaceholder
http://id.southampton.ac.uk/generic-products-and-services/ATM	http://www.w3.org/2000/01/rdf-schema#label	ATM
http://id.southampton.ac.uk/generic-products-and-services/AirTravel	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.org/goodrelations/v1#ProductOrServicesSomeInstancesPlaceholder
http://id.southampton.ac.uk/generic-products-and-services/AirTravel	http://www.w3.org/2000/01/rdf-schema#label	Air Travel
http://id.southampton.ac.uk/generic-products-and-services/Airmail	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.org/goodrelations/v1#ProductOrServicesSomeInstancesPlaceholder
http://id.southampton.ac.uk/generic-products-and-services/Airmail	http://www.w3.org/2000/01/rdf-schema#label	Airmail
http://id.southampton.ac.uk/generic-products-and-services/Alcohol	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.org/goodrelations/v1#ProductOrServicesSomeInstancesPlaceholder
http://id.southampton.ac.uk/generic-products-and-services/Alcohol	http://www.w3.org/2000/01/rdf-schema#label	Alcohol
http://id.southampton.ac.uk/generic-products-and-services/AlcoholAdvice	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.org/goodrelations/v1#ProductOrServicesSomeInstancesPlaceholder
http://id.southampton.ac.uk/generic-products-and-services/AlcoholAdvice	http://www.w3.org/2000/01/rdf-schema#label	Alcohol advice
http://id.southampton.ac.uk/generic-products-and-services/ArtExhibitions	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.org/goodrelations/v1#ProductOrServicesSomeInstancesPlaceholder
http://id.southampton.ac.uk/generic-products-and-services/ArtExhibitions	http://www.w3.org/2000/01/rdf-schema#label	Art Exhibitions
http://id.southampton.ac.uk/generic-products-and-services/ArtGallery	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.org/goodrelations/v1#ProductOrServicesSomeInstancesPlaceholder
http://id.southampton.ac.uk/generic-products-and-services/ArtGallery	http://www.w3.org/2000/01/rdf-schema#label	Art Gallery
http://id.southampton.ac.uk/generic-products-and-services/BankAccounts	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.org/goodrelations/v1#ProductOrServicesSomeInstancesPlaceholder

Obrázek 7.6: Ukážka výsledku posielania dotazu na druhý SPARQL endpoint

Kapitola 8

Záver

Výsledkom tejto bakalárskej práce bolo vytvoriť ľahko importovatelnú komponentu editora pre špeciálny jazyk SPARQL pre písanie dotazov umožňujúcich spravovanie dátového úložiska RDF. Táto komponenta je dostupná ako npm balík. Ďalším výsledkom je demonštračná aplikácia schopná posielať SELECT dotazy napísané pomocou tejto komponenty na SPARQL endpointy a následne zobraziť výsledky tohto dotazu ako dátovú tabuľku.

Pri vytváraní návrhu komponenty editora sa skúmali rôzne nástroje a webové aplikácie, kde sa pracuje s jazykom SPARQL pre písanie dotazov. Následne sa zobrali nedostatky týchto riešení a na základe neho sa vytvoril samotný návrh, v ktorom sa prechádzajú rôzne navrhnuté funkcionality. Pri návrhu týchto funkcionalít bolo dôležité držať sa reálny a splniteľných cieľov v rámci bakalárskej práce, aby ich výsledkom bol funkčný produkt, ktorý vizuálne odpovedá moderným aplikáciám.

Výsledná komponenta editora obsahuje zvýraznenie syntaxe pre jazyk SPARQL, automatické dopĺňanie kľúčových slov po napísaní aspoň jedného znaku a stlačení klávesovej skratky 'CTRL+Space', pamätanie si uložených prefixov zoskupených v skupinách, ktoré sa dajú vložiť na začiatok dotazu za pomoci užívateľského rozhrania, menenie tém pre jednoduché zladenie výzoru užívateľského rozhrania spolu s aplikáciou využívajúcou túto komponentu a možnosť písania viac dotazov naraz za pomoci zabudovanej funkcie prepínateľných kariet.

Komponenta editoru aj aplikácia boli implementované s JavaScriptových rámcom Vue.js. Voľbu tohto rámca považujem za pozitívnu pre jednoduchosť práce s ním. Ako základ pre samotný editor sa využila knižnica CodeMirror. Pri tvorbe užívateľského rozhrania sa využíva knižnica PrimeVue. Pre možnosť využitia výsledného npm balíka obsahujúceho komponentu editora nielen v Vue.js sa využil nástroj Vite využívajúci RollUp.js. Pri aplikácií sa posielani samotných dotazov na SPARQL endpointy využíva knižnica Axios.

Možným rozšírením komponenty editora by mohla byť funkcionalita overujúca chybnosť napísaných dotazov v reálnom čase už pri ich písaní. Zobrazené chyby sa mohli podčiarknuť alebo zobraziť na boku pri očíslovaní riadkov.

Literatura

[1] Angular. *What is Angular?* [online]. [cit. 2022-04-15]. Dostupné z: https://angular.io/guide/what-is-angular.

[2] Angular. *Lifecycle hooks* [online]. [cit. 2022-04-26]. Dostupné z: https://angular.io/guide/lifecycle-hooks.

[3] MDN. *CSS: Cascading Style Sheets* [online]. [cit. 2022-04-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS.

[4] MDN. *Introduction to the DOM* [online]. [cit. 2022-04-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

[5] MDN. *HTML Basics* [online]. [cit. 2022-04-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics.

[6] MDN. *Introduction* [online]. [cit. 2022-04-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction.

[7] React. *Tutorial: Intro to React* [online]. [cit. 2022-04-16]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html.

[8] MDN. *Window.localStorage* [online]. [cit. 2022-04-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage.

[9] npm. *About npm* [online]. [cit. 2022-05-05]. Dostupné z: https://docs.npmjs.com/about-npm.

[10] Lassila, O. a Swick, R. R. Resource Description Framework (RDF) Model and Syntax Specification. [online]. The World Wide Web Consortium (W3C). 1999, [cit. 2022-04-15]. Dostupné z: https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[11] React. *Introducing JSX* [online]. [cit. 2022-04-25]. Dostupné z: https://reactjs.org/tutorial/tutorial.html.

[12] Aranda, C. B., Corby, O. et al. *SPARQL 1.1 Overview* [online]. [cit. 2022-04-15]. Dostupné z: https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/.

[13] Vite. *Getting Started* [online]. [cit. 2022-05-05]. Dostupné z: https://vitejs.dev/guide/.

[14] Vue.js. *Introduction* [online]. [cit. 2022-04-15]. Dostupné z: https://vuejs.org/guide/introduction.html.

[15] Vue.js. *Lifecycle Hooks* [online]. [cit. 2022-04-25]. Dostupné z: https://vuejs.org/guide/essentials/lifecycle.html.

[16] Žára, O. *JavaScript - Programátorské techniky a webové technologie*. Computer Press, 2015. ISBN 978–80–251–4573–9.

Příloha A

Obsah priloženého pamäťového média

- xpaulo04.pdf - PDF súbor tejto práce
- xpaulo04.zip - zdrojové súbory tejto práce
- src.zip - zdrojové súbory komponenty editora a aplikácie v monorepozitári
  - component - zdrojové kódy komponenty editora
  - app - zdrojové kódy demonštračnej aplikácie
  - README.md - textový súbor obsahujúci návod pre aplikáciu