VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÉ ROZHRANÍ PRO PROCHÁZENÍ ÚLOŽIŠTĚ RDF

WEB INTERFACE FOR RDF STORAGE BROWSING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PATRIK TOMOV**
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: **Tomov Patrik**
Program: Informační technologie
Název: **Webové rozhraní pro procházení úložiště RDF**
Web Interface for RDF Storage Browsing
Kategorie: Databáze

Zadání:

1. Seznamte se s datovým modelem RDF, dotazovacím jazykem SPARQL a existujícími RDF úložišti.
2. Prostudujte existující nástroje a knihovny pro tvorbu klientských webových aplikací v jazyce JavaScript. Zaměřte se zejména na rámec Vue.js.
3. Navrhněte architekturu aplikace pro interaktivní zadávání SPARQL dotazů a procházení obsahu RDF úložiště.
4. Po dohodě s vedoucím zvolte vhodnou implementační platformu a implementujte navrženou aplikaci.
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 webové rozhranie pre prechádzanie úložiska RDF, kde bude môcť užívateľ manipulovať s jeho obsahom. V rámci úložiska bude môcť vytvárať a mazať repozitáre, pridávať a mazať obsah repozitára, zobraziť si obsah repozitára a interaktívne sa v ňom navigovať a tiež vykonávať dotazy v jazyku SPARQL v rámci repozitára. Klientska časť aplikácie je napísaná v jazyku Javascript s využitím frameworku Vue. Pre serverovú časť je použitý RDF4J server, ktorý je nasadený na službu Apache Tomcat. Komunikácia medzi serverom a klientom je zabezpečená pomocou RDF4J REST API. Výsledná aplikácia umožnuje interaktívne a jednoducho prechádzať obsah RDF úložiska.

Abstract

The aim of this thesis is to create a web interface for browsing the RDF storage, where the user will be able to manipulate its content. Within the storage, he will be able to create and delete repositories, add and delete repository content, view and interact with repository content, and also perform SPARQL queries within the repository. The client part of the application is written in Javascript using the Vue framework. The RDF4J server, which is deployed on the Apache Tomcat service, is used for the server part. Communication between the server and the client is secured using the RDF4J REST API. The resulting application allows to interactively and easily browse the content of the RDF storage.

Kľúčové slová

Uživateľské rozhranie, RDF, RDF4J, Javascript, Vue, Typescript, PRIME VUE, RDF4J REST API, SPARQL, Cypress

Keywords

User interface, RDF, RDF4J, Javascript, Vue, Typescript, PRIME VUE, RDF4J REST API, SPARQL, Cypress

Citácia

TOMOV, Patrik. *Webové rozhraní pro procházení úložiště RDF*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Radek Burget, Ph.D.

Webové rozhraní pro procházení úložiště RDF

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána doc. Ing. Radka Burgeta, Ph. D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Patrik Tomov
9. mája 2022

Poďakovanie

Rád by som poďakoval svojmu vedúcemu práce doc. Ing. Radkovi Burgetovi, Ph.D za pomoc a rady pri vytváraní aplikácie a písaní tejto práce.

Obsah

1 Úvod **3**

2 Zoznámenie sa so sémantickým webom a jazykom RDF **4**
2.1 Semantický web 4
2.2 Resource Description Language 5
2.2.1 Typy zdrojov 6
2.2.2 Predpony a menné priestory 6
2.2.3 Textová reprezentácia RDF 7
2.3 RDF úložiská 9
2.3.1 Virtuoso 11
2.3.2 Blazegraph 11
2.3.3 Apache Jena 11
2.3.4 RDF4J 11
2.4 SPARQL 14
2.4.1 Modifikátory výsledku 16
2.4.2 Typy dotazov 17
2.4.3 SPARQL 1.1 UPDATE 18

3 Technológie pre tvorbu klientskych webových aplikácií **20**
3.1 JavaScript 20
3.1.1 JSX 20
3.2 Jednostránková webová aplikácia 21
3.3 Šablóna syntaxe 21
3.4 Objektový model dokumentu 21
3.5 JavaScriptové rámce 21
3.5.1 React 22
3.5.2 Angular 22
3.5.3 Vue 23

4 Návrh implementácie **26**
4.1 Použitie aplikácie 26
4.2 Vybrané technológie 27
4.2.1 Nástroje pre tvorbu užívateľského rozhrania 27
4.2.2 Klientska časť 28
4.2.3 Serverová časť 29
4.3 Návrh uživateľského rozhrania 29
4.4 Návrh testovania 33

5 Implementácia **34**
5.1 Konfiguŕacia serverovej časti 34
5.1.1 Rozhranie API 34
5.2 Implementácia frontendu 35
5.2.1 Node Package Manager 35
5.2.2 Typescript 35
5.2.3 Štruktúra aplikácie 36
5.2.4 Získanie a spracovanie dát 39

6 Testovanie **41**
6.1 Automatizované testovanie 41
6.2 Testovanie užívateľského rozhrania 42

7 Záver **43**

Literatúra **44**

A Obsah pamäťového média **47**

Kapitola 1

Úvod

WWW[1] je bezpochyby najväčším vynálezom, ktorý sa zapísal do dejín ľudstva v podobe internetu. Denne ho využívajú miliardy ľudí po celom svete. V roku 2001 prišiel Tim Barners-Lee s myšlienkou sémantického webu, kedy upozornil na skutočnosť, že súčasný web je len spleť jednotlivých webových stránok, ktorá neustále rastie a v ktorej je stále zložitejšie nájsť relatívne informácie. Sémantický web je rozšírením internetu kde má každá informácia pridelený presne definovaný význam [38]. Je založený na technológií Resource Description Framework (RDF), v ktorej sú reprezentované tieto informácie. Existuje viacero úložisk týchto informácií a spolu so sémantickým webom a jazykom RDF sú bližšie popísane v kapitole 2, ktorá je ešte doplnená o dopytovací jazyk SPARQL[2], pomocou ktorého je možné manipulovať s týmito dátami.

Cieľom tejto práce je vytvoriť webové rozhranie pre prechádzanie tohto RDF úložiska. Neexistuje veľa voľne dostupných aplikácií pre tento účel čo je motiváciu za vytvorením takéhoto riešenia. Téma má zaujala z dôvodu tvorby grafického rozhrania s využitím moderných technológií v jazyku JavaScript, ktorý je spolu s týmito technológiami popísaný v kapitole 3.

Kapitola 4 popisuje návrh implementácie kde je vysvetlené použitie aplikácie a taktiež sú tu popísane vybrané technológie pre tvorbu jednotlivých častí aplikácie. Hlavnou časťou tejto kapitoly je návrh užívateľského rozhrania kde sa čitateľ zoznámi s predbežným vzhľadom aplikácie.

V kapitole 5 je popísaná samotná implementácia aplikácie. V prvej časti je vysvetlená konfigurácia serverovej časti a v druhej časti je popísaná klientska časť kde sa čitateľ zoznámi so štruktúrou aplikácie a jednotlivými postupmi pri implementácií.

V predposlednej kapitole 6 je popis testovania aplikácie, ktoré je rozdelené na dva časti. Automatizované testovanie pomocou nástroja Cypress a testovanie užívateľského rozhrania na konkrétnych užívateľoch.

Záverečné zhrnutie práce sa nachádza v kapitole 7, ktoré je doplnené o možné vylepšenia v aplikácií.

[1] World Wide Web
[2] SPARQL Protocol and RDF Query Language

Kapitola 2

Zoznámenie sa so sémantickým webom a jazykom RDF

V tejto kapitole je bližšie popísane čo je to sémantický web, akú úlohu zohráva v rámci internetu. Ďalej je predstavený jeden zo stavebných kameňov tohto webu RDF, kde je podrobnejšie vysvetlené čo je to a aké dáta ho tvoria. Čitateľ sa tiež zoznámi s niektorými typmi RDF úložiska a nakoniec je popísaný dopytovací jazyk SPARQL, ktorý RDF používa.

2.1 Semantický web

Sémanticky web nie je samostatný web, ale je považovaný za rozšírenie toho súčastného WWW, v ktorom informácie majú presne definovaný význam, čo umožňuje počítačom a ľuďom lepšie spolupracovať. Keďže informácie na tomto webe majú presne definovaný význam a sú zakódované v strojovo čitateľnom formáte, majú množstvo špeciálnych charakteristík, ktoré nie sú dostupné na tradičnom webe, ako napríklad schopnosť strojového spracovania, vyhľadávanie informácií a odvodzovanie znalostí. [36]

Tento web je založený na koncepte prepojených údajov, čo je termín používaný na opis postupov odhaľovania, zdieľania a spájania informácií na webe pomocou najnovších špecifikácií W3C[1], ako je Resource Description Framework (RDF). Čím viac strojovo čitateľných údajov je dostupných na webe, tým viac úloh, ktoré si dnes vyžadujú interakciu užívateľa, možno automatizovať, čo vedie k inteligentnejším aplikáciám a autonómnym agentom.[22]

Prepojené údaje sa rýchlo stávajú dominantným modelom pre integráciu údajov medzi databázami. Sú čoraz rozšírenejší najmä medzi vládami a podnikmi, ktoré považujú RDF za flexibilnejší spôsob reprezentácie svojich údajov, najmä vláda USA[2] a vláda Spojeného kráľovstva[3] ako aj Google, Bing a Yahoo[4]. So zvyšujúcou sa popularitou a dostupnosťou takýchto údajov a zodpovedajúcich technológií sa tiež zvyšuje množstvo softvérových platforiem, ktoré používajú RDF (napr. webová stránka BBC [24]).

[1] World Wide Web Consortium
[2] https://data.gov
[3] https://data.gov.uk
[4] https://schema.org

2.2 Resource Description Language

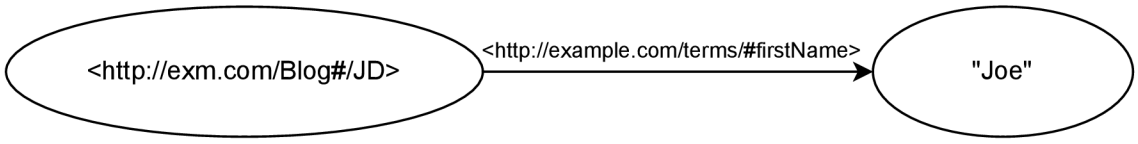
Aby bol sémantický web úspešný v mierke podobnej WWW, je potrebné použiť jazyk, ktorý je dostatočne univerzálny a agnostický na to, aby bol všeobecne užitočný na popis čohokoľvek, čo je úlohou RDF. Prvým cieľom RDF, bolo poskytnúť dátový model metadát na web. V súčastnosti sa objavuje ako dátový model pre sieť údajov (Web of Data) a sémanticky web, poskytujúci logickú organizáciu definovanú z hľadiska dátovej štruktúry na podporu reprezentácie, prístupu, obmedzení a vzťahov objektov záujmu v danej aplikačnej doméne. [25]

Základný dátový model je pomerne jednoduchý. Základná jednotka informácie sa nazýva trojica (triple) (s, p, o) zložená z predmetu (s), predikátu (p) a objektu (o). Každá trojica predstavuje skutočnosť o popisovanej veci (predmet), o konkrétnej vlastnosti (predikát) s danou hodnotou (objekt). Každá časť tejto trojice sa považuje za zdroj (resource), ako sa na nich budem odkazovať v nasledujúcich častiach. Ako konkrétny príklad uvediem tabuľku s ukážkovými dátami, ktorá bude použitá v sekcii 2.4 na vykonávanie rôznych databázových dotazov.

predmet	predikát	objekt
<http://exm.com/Blog#JD>	<http://exm.com/terms#firstName>	"Joe"
<http://exm.com/Blog#JD>	<http://exm.com/terms#lastName>	"Doe"
<http://exm.com/Blog#JD>	<http://exm.com/Blog#owner>	"TestBlog1"
<http://exm.com/Blog#JD>	<http://exm.com/Blog#owner>	"TestBlog2"
<http://exm.com/Blog#JD>	<http://exm.com/Blog#isFollowing>	blog:MS
<http://exm.com/Blog#WR>	<http://exm.com/terms#firstName>	"Will"
<http://exm.com/Blog#WR>	<http://exm.com/terms#lastName>	"Smith"
<http://exm.com/Blog#WR>	<http://exm.com/Blog#owner>	"TestBlog1"
<http://exm.com/Blog#WR>	<http://exm.com/Blog#owner>	"TestBlog2"
<http://exm.com/Blog#WR>	<http://exm.com/Blog#isFollowing>	blog:MS
<http://exm.com/Blog#MS>	<http://exm.com/terms#firstName>	"Mary"
<http://exm.com/Blog#MS>	<http://exm.com/terms#lastName>	"Smith"

Tabuľka 2.1: Ukážkový RDF dátový set

Ako je možné si všimnúť, v tabuľke je hodnota, ktorá je odlišná od ostatným a tou hodnotou je blog:MS. Je to skrátený zápis zdroja, ktorý je bližšie popísaný v sekcii 2.2.2. Kompaktnejší spôsob zobrazenia je pomocou orientovaného grafu.



Obr. 2.1: Príklad trojice v grafovej reprezentácii

Hlavný problém, ktorý môže v RDF nastať je identifikácia nejakého konkrétneho zdroja medzi viacerými zdrojmi. Tento problém je v tomto modeli riešený použitím URIs[5] alebo ich zovšeobecnením IRIs[6], ktoré podporujú kódovanie v Unicode radšej ako v ASCII. Týmto sa

[5]Uniform Resource Identifier
[6]Internationalized Resource Identifier

dostávame k jednotlivým typom, ktoré tieto zdroje môžu nadobudnúť a budú podrobnejšie vysvetlené v nasledujúcej sekcii.

2.2.1 Typy zdrojov

Prvým už spomínaným typom sú URIs, ktoré predstavujú spoločné globálne identifikátory pre zdroje na webe. Formát a syntax URIs sú veľmi podobné známym URLs[7]. Napríklad hodnota JD, ktorá predstavuje nejakú konkrétnu informáciu na webe, môže vo formáte URI vyzerať nasledovne, <htttp://www.exm.com/Blog#JD>. V podstate, URLs sú len špeciálnym prípadom URIs. Po tomto stručnom vysvetlení sa môžme vrátiť k problému RDF a to, ako rozlíšiť rôzne zdroje s rovnakým názvom. URIs toto prirodzene podporujú a riešia to tzv. dereferencovaním, to znamená že každá časť URI je lokátor pre zdroj (napr. server, názov, adresáre, názvy súborov atď.), kde je možné nájsť ďalšie informácie. Pre príklad URI http://www.w3.org/standartds/techs/rdf#w3c_all špecifikuje protokol (http), názov servera (www.w3.org), niektoré adresáre na tomto serveri (standards/techs), dokument (rdf) a lokáciu v tom dokumente (w3c_all). Ak by sme chceli rozlíšiť značku apple od ovocia apple, budu sa líšiť odkazom na rôzne URIs, teda <http://www.apple.com/public/RDF/#Apple> a <http://www.fruits.com/fall/#Apple>. Väčšina informácii v RDF dátových súboroch bude pozostávať práve z URI, ktoré môžu nadobudnúť všetky zdroje predmet, predikát aj objekt. Presnejšie non-URI hodnoty, nazývané literály, budú podporované len pre objekty, čím sa dostávame k druhému typu zdrojov.

Hodnota literálu môže byť sprevádzaná dátovým typom (napr. xsd:integer[8]) a bude sa označovať ako typovaný literál (typed literal) alebo obyčajný literál (plain literal) v opačnom prípade.

Nakoniec, aj keď väčšina zdrojov je reprezentovaná pomocou URI, prípadne literálom, RDF poskytuje možnosť, aby boli zdroje anonymné. Takéto zdroje sa nazývajú anonymné uzly (blank nodes alebo bnodes), ktoré nemajú trvalú identitu. Anonymné uzly sú definované špecifickým menným priestorom označeným _. Hlavným účelom týchto anonymných uzlov je buď konštatovať existenciu nejakého zdroja, ktorý nevieme pomenovať alebo zoskupenie nejakých poznatkov. Napríklad ak chceme poukázať, že anonymný sledujúci pridal komentár do blogu o vede.

2.2.2 Predpony a menné priestory

Väčšina zdrojov je reprezentovaná pomocou URI, ktorého formát nie je pre zápis veľmi ideálny keďže môžu vzniknúť veľmi dlhé reťazce, ktoré by mohli robiť problém pri zobrazovaní trojíc v tabuľkovej alebo aj grafovej forme. Taktiež sa budú často opakovať podreťazce v rámci zdrojov (viď tabuľka 2.1). Pre uľahčenie zápisu a zlepšenie čitateľnosti a zobrazenia je možné v RDF definovať predponu ďalej ako prefix reprezentujúcu menný priestor (namespace) ako napríklad blog:JD kde blog reprezentuje menný priestor <http://exm.com/Blog#>.

Prefixy nie sú globálne identifikátory a mali by byť deklarované s ich menných priestorom na začiatku RDF súboru (podobne ako v prípade kvalifikovaných názvov v XML). Syntax deklarovania prefixu je takýto: PREFIX [názov prefixu]: URI. V prípade deklarovania prefixu v rámci súboru je nutné za URI pridať bodku, ak sa prefixy deklarujú v rámci

[7] Uniform Resource Locator

[8] xsd prefix korešponduje s XML Schema menným priestorom http://www.w3.org/2001/XMLSchema#

dotazu ako v ukážke 2.6 nie je nutné udávať bodku. Spomínaný dátový set (viď tabuľka 2.1) by sa dal zjednodušiť pomocou prefixov nasledovne.

predmet	predikát	objekt
blog:JD	ex:firstName	"Joe"
blog:JD	ex:lastName	"Doe"
blog:JD	blog:owner	"TestBlog1"
blog:JD	blog:owner	"TestBlog2"
blog:JD	blog:isFollowing	blog:MS
blog:WR	ex:firstName	"Will"
blog:WR	ex:lastName	"Smith"
blog:WR	blog:owner	"TestBlog1"
blog:WR	blog:owner	"TestBlog2"
blog:WR	blog:isFollowing	blog:MS
blog:MS	ex:firstName	"Mary"
blog:MS	ex:lastName	"Smith"

Tabuľka 2.2: Ukážkový RDF dátový set s použitím prefixov

2.2.3 Textová reprezentácia RDF

Táto sekcia je inšpirovaná z [1][15] a obsahuje niektoré z najpopulárnejších a najviac používaných serializačných formátov RDF dát.

RDF/XML

RDF/XML bol prvý štandardizovaný formát serializácie RDF. Každý zdroj je definovaný ako rdf:Description XML prvok s rdf:about atribútom, ktorý uvádza jeho URI. Ďalšie charakteristiky týkajúce sa predmetu sú uvedené v podradených prvkoch daného XML elementu. Ako odkaz na URI, by sa mal použiť rdf:resource atribút. V ukážke 2.1 sú uvedené štyri tvrdenia o JD (<htttp://www.exm.com/Blog#JD>).

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22 -rdf-syntax-ns#"
         xmlns:blog="http://exm.com/Blog#"
         xmlns:ex="http://exm.com/terms#">
   <rdf:Description rdf:about="<http://exm.com/Blog#JD>">
      <rdf:type rdf:resource="http://exm.com/Blog#User">
      <blog:hasGender rdf:resource="http://exm.com/terms#Male">
      <ex:firstName>Joe</ex:firstName>
      <ex:lastName>Doe</ex:lastName>
   </rdf:Description>
</rdf:RDF>
```

Výpis 2.1: Ukážka RDF/XML formátu

N3

Tim-Berners Lee chcel niečo lepšie ako RDF/XML a prišiel s formátom N3 (Notation3). Syntax sa začína viac podobať na RDF predmet, predikát, objekt model. Je ľahšie čitateľný na prvý pohlaď a pomáha lepšie pochopiť ako RDF funguje. Serializácia N3 je ale pomerne nákladná, čo môže pri väčších dátových setoch spôsobiť problémy s výkonom. Ma taktiež dosť funkcií, pretože podporuje pravidlá RDF, ktoré sťažujú parsovanie. Pokiaľ nie je potrebné tieto funkcie využívať je lepšie siahnuť po jeho populárnejšom nástupcovi Turtle.

```
<https://www.w3.org/People/Berners-Lee/> <http://schema.org/birthPlace>
"1955-06-08"^^<http://www.w3.org/2001/XMLSchema#date>.
<https://www.w3.org/People/Berners-Lee/> <http://schema.org/birthPlace>
<http://dbpedia.org/resource/London>.
<https://www.w3.org/People/Berners-Lee/> <http://schema.org/birthPlace>
<http://dbpedia.org/resource/London>.
```

Výpis 2.2: Ukážka N3 formátu

Turtle

Turtle je podmnožinou formátu Notation3 (N3) a jednou z jeho silných stránok je veľmi dobre čitateľná syntax. Čítanie RDF vo formáte Turtle je oveľa jednoduchšie, pretože umožňuje definovať predpony na začiatku súboru .ttl, čím sa skráti každá trojica. Ďalšou vlastnosťou Turtle je, že viacero trojíc s rovnakým predmetom je zoskupených do blokov (takže URI nie je opakovane uvádzané).

```
@prefix tim: <https://www.w3.org/People/Berners-Lee/>.
@prefix schema: <http://schema.org/>.
@prefix dbpedia: <http://dbpedia.org/resource/>.

<tim> schema:birthDate "1955-06-08"^^<http://www.w3.org/2001/XMLSchema#date>;
      schema:birthPlace <dbpedia:London>.
```

Výpis 2.3: Ukážka Turtle formátu

N-Triples

Ukladanie RDF v N-Triples formáte je jednoduchšie, pretože každý riadok súboru .nt predstavuje jednu trojicu <predmet> <predikát> <objekt> ukončenú bodkou. Tento formát nepodporuje prefixy ani žiadne funkcie, čo môže mierne zhoršiť čitateľnosť ale na druhú stranu je to jednoducho parsovateľné.

```
<https://www.w3.org/People/Berners-Lee/> <http://schema.org/birthDate>
"1955-06-08"^^<http://www.w3.org/2001/XMLSchema#date>.
<https://www.w3.org/People/Berners-Lee/> <http://schema.org/birthPlace>
<http://dbpedia.org/resource/London>.
```

Výpis 2.4: Ukážka N-Triples formátu

JSON-LD

JSON (JavaScript Object Notation for Linked Dat) je bezpochyby najpopulárnejším spôsobom serializácie údajov vo webových aplikáciach. JSON-LD vzniká rozšírením formátu JSON o @context. Tento objekt slúži hlavne ako mapovanie. Parsovanie tohto formátu je žiaľ tiež veľmi zložité a nákladne.

```
{
  "@context": {
    "dbpedia": "http://dbpedia.org/resource/",
    "schema": "http://schema.org/"
  },
  "@id": "https://www.w3.org/People/Berners-Lee/",
  "schema:birthDate": "1955-06-08",
  "schema:birthPlace": {
    "@id": "dbpedia:London"
  }
}
```

Výpis 2.5: Ukážka JSON-LD formátu

2.3 RDF úložiská

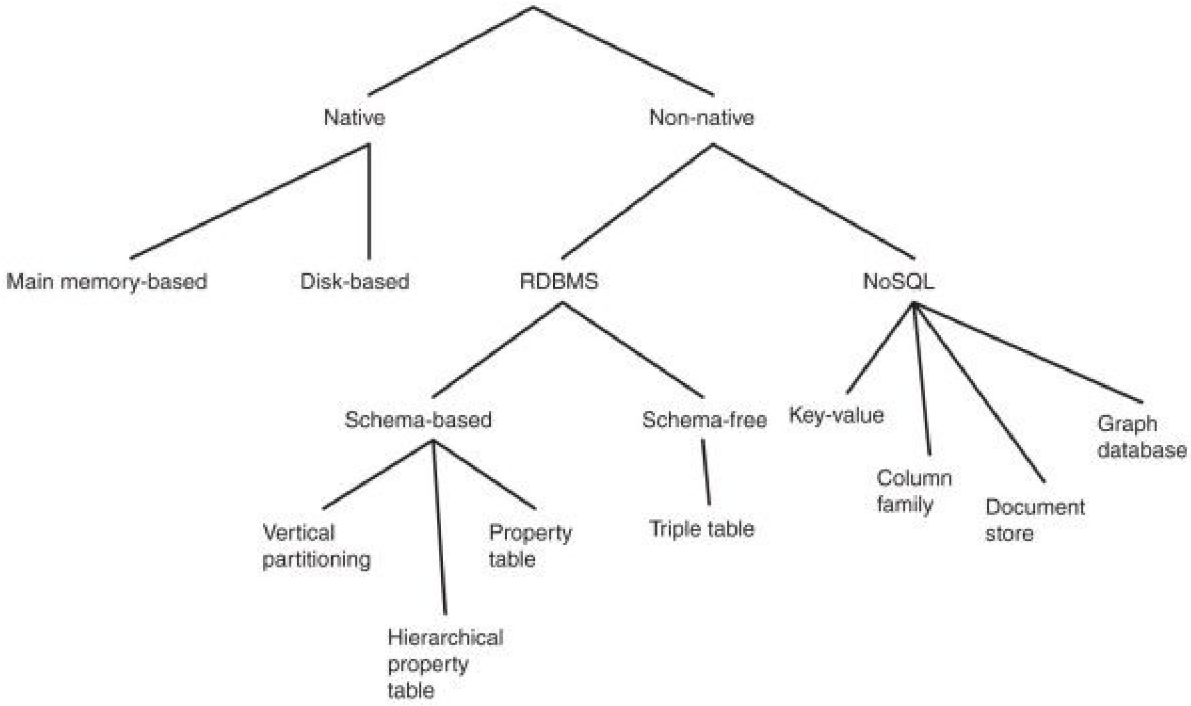
Táto sekcia bude obsahovať rôzne fyzické organizácie na ukladanie a indexovanie RDF dát. Okrem toho, majú tieto rôzne formy ukladania dát vplyv na veľkosť potrebnú na ukladanie súborov ako aj významný vplyv na výkon hlavných operácií vykonávaných na RDF úložisku, ako je spracovanie dotazov. Medzi všetkými existujúcimi systémami môžeme rozlíšiť riešenia, implementujúce svoj vlastný úložiskový backend, označované ako natívne úložiska, a tie ktoré používajú existujúci systém správy databáz, označované ako nenatívne úložiska. Obrázok 2.2 poskytuje klasifikáciu podĺž týchto dvoch osí pre ukladanie a prístup k RDF dátam. [6]

Natívny prístup

Natívny prístup poskytuje spôsob, ako ukladať dáta bližšie k ich dátovému modelu, pričom sa vyhýba mapovaniu na entity DBMS[9], ako sú vzťahy. Využíva trojitý charakter RDF dát ako aktívum a umožňuje riešiť špecifiká svojho grafového prístupu ako je schopnosť na zvládnutie unikátnosti dát a dynamického aspektu ich schémy. Tieto systémy môžu byť široko klasifikované ako diskové (disk-based), trvalé systémy uložené na disku a systémy založené na hlavnej pamäti (main memory-based), ktoré sú nestále.

Perzistentné diskové úložisko je spôsob trvalého ukladania RDF dát do systémového súboru. Tieto implementácie môžu využívať známe indexové štruktúry ako B+ stromy, ktoré sa vo veľkej miere používajú v RDBMS[10]. Treba zvážiť, že čítanie a písanie na disky vyvoláva problém s výkonom, ktorý môže nastať pri práci s veľkým množstvom dát. Táto úvaha motivovala memory-based riešenia, ktorých cieľom je ukladať čo najväčšie množstvo dát, RDF trojíc, slovníkov, ontológií ako je možné mať v hlavnej pamäti.

[9] Database Managament System
[10] Relational Database Managament System



Obr. 2.2: Klasifikácia RDF úložiska prevzaté z [6]

Memory-based RDF úložisko si prideľuje určité množstvo dostupnej pamäte na uloženie celej grafovej štruktúry RDF. Rovnako ako pri disk-based úložisku, tento prístup sa spolieha na indexy a efektívne spracovanie a tiež na techniky založené na viacerých indexoch. Pri práci s memory-based úložiskom sú najviac časovo náročné operácie načítania a parsovania RDF súborov, ale tiež aj vytváranie vhodných indexov. [6]

Nenatívny prístup

Nenatívne prístupy využívajú systém správy databázy na uloženie RDF dát natrvalo. Takýto prístup ťaží z rokov výskumu a vývoja na DBMS systémoch a je to obzvlášť dôležité pre systémy založené na RDBMS. Pre príklad, najviac známe RDBMS systémy majú priemyselnú podporu transakcií a rôzne bezpečnostné opatrenia, ktoré natívnym prístupom chýbajú.

V kategórií RDBMS môžeme rozlišovať medzi prístupmi so schémami a bez schém ako je vidieť na obrázku 2.2. V prístupe bez schém máme na mysli, že existuje len jedná tabuľka označovaná ako triple table a úložisko sa volá triple-store , ktorá je zodpovedná za ukladanie všetkých trojíc. V prístupoch so schémou, na rozdiel od predchádzajúcej reprezentácie, ktorá je priamočiara, sa používajú schémové charakteristiky, ktoré rozdeľujú trojice do rôznych tabuliek. Takéto rozdelenie môže byť organizované na základe vnútornej štruktúry trojíc t.j. predmetu, predikátu a objektu alebo na základe vlastností a tried schém RDFS[11] alebo OWL[12]. Je možné rozlíšiť dve hlavné schémy, tabuľka vlastností (property table) a prístup vertikálneho rozdeľovania (vertical partitioning). Hlavnou motiváciou týchto prístu-

[11] RDF Schema
[12] Web Ontology Language

pov založených na princípe kľúč-hodnota je riešiť distribúciu veľmi veľkých súborov dát cez klaster komoditného hardvéru. [6]

2.3.1 Virtuoso

Virtuoso je multiprotokolový SQL server poskytujúci ODBC[13]/JDBC[14] prístup k relačným údajom uloženým buď v samotnom Virtuoso alebo v akejkoľvek kombinácii externých relačných databáz. Má vstavaný HTTP server a mimo rôznych úložisk poskytuje taktiež podporu ukladania RDF dát. Podporuje tiež SPARQL, ktorý je zabudovaný do SQL na dopytovanie RDF dát. [23]

2.3.2 Blazegraph

Blazegraph je open-source grafová databáza napísaná v Jave, ktorá podporuje RDF/SPARQL APIs. Ako grafová databáza používa Blazegraph grafovú štruktúru uzlov a hrán na reprezentáciu dát. Podporuje tiež dátový model triplestore, ktorý možno považovať za špecializovanú verziu grafových databáz, ktorá je optimalizovaná na ukladanie a získavanie trojíc. [8]

2.3.3 Apache Jena

Apache Jena je open-source framework sémantického webu napísaný v Jave. Poskytuje API pre získanie a zápis dát do RDF grafov. Grafy sú reprezentované ako abstraktný model. Dáta modelu možu byť údaje poskytnuté zo súborov, databáz, URL alebo ich kombinácie. Ako ostatné úložiska podporuje SPARQL, pomocou ktorého je možné vyhľadať a meniť dáta. Jena poskytuje Apache Jena Fuseki SPARQL server, ktorý je možné nasadiť na Java Servlet ako napríklad Apache Tomcat. [3]

2.3.4 RDF4J

RDF4J je open-source modulárny java framework, ktorý umožňuje prácu s RDF dátami. Zahŕňa to ukladanie RDF dát a dopytovanie sa na tieto dáta. RDF4J obsahuje dva nástroje RDF4J Server a RDF4J Workbench a tiež poskytuje RDF4J REST API. Podporuje väčšinu RDF formátov ako RDF/XML, Turtle, N3, N-Triples, JSON-LD, ktoré boli popísané v sekcií 2.2.3. RDF4J Server ponúka dve predpripravené databázy in-memory store a native store, ktoré boli vysvetlené v sekcii 2.3. [26]

RDF4J REST API

Tento popis rozhrania API s jednotlivými metódami, ktoré poskytuje je prevzatý z [26]. RDF4J REST API je HTTP protokol, ktorý plne pokrýva implementáciu SPARQL 1.1 protokolu W3C. Vďaka tomu sa RDF4J server správa ako endpoint SPARQL. RDF4J podporuje prístup k lokálnym úložiskám teda napríklad na serveri Tomcat ale aj k vzdialeným úložiskám pomocou rovnakého API. Toto rozhranie ponúka nasledujúcu funkcionalitu:

[13] Open Database Connectivity
[14] Java Database Connectivity

Zoznam repozitárov

Prehľad repozitárov, ktoré sú dostupné na serveri, je možné získať odoslaním dotazu na adresu <RDF4J_URL>/repositories.
Podporované metódy:

- **GET** - získa zoznam repozitárov vrátane id, popisu a lokácie

Hlavičky žiadosti:

- **Accept** - typ internetového média podporujúci RDF formáty

Dotazy na repozitár

Dotazy na konkrétny repozitár s ID <ID> je možné vyhodnotiť odoslaním dotazu na adresu <RDF4J_URL>/repositories/<ID>.
Podporované metódy:

- **GET** - načíta dáta, ktorým zodpovedá dotaz
- **POST** - vykoná aktualizáciu dát v repozitári

Parametre:

- **query** - dotaz
- **limit** (voliteľný) - špecifikuje maximálny počet výsledkov, ktoré sa majú vrátiť
- **offset** (voliteľný) - špecifikuje počet výsledkov, ktoré sa majú preskočiť

Hlavičky žiadosti:

- **Accept** - typ internetového média podporujúci RDF formáty
- **Content-Type** - špecifikuje typ internetového média tela POST dotazu

Vytvorenie repozitára

Nový repozitár s ID <ID> môže byť vytvorený na serveri zaslaním requestu na adresu <RDF4J_URL>/repositories/<ID>.
Podporované metódy:

- **PUT** - očakáva sa, že telo žiadosti bude obsahovať dokument RDF, ktorý obsahuje formu konfigurácie úložiska serializovanú v RDF. Ak úložisko so zadaným ID predtým existovalo, server požiadavku odmietne. Ak neexistuje, vytvorí sa nové, prázdne úložisko.

Hlavičky žiadosti:

- **Content-Type** - 'text-turtle' typ internetového média

Odstránenie repozitára

Špecifický repozitár s ID <ID> je možné zo servera vymazať odoslaním žiadosti na adresu <RDF4J_URL>/repositories/<ID> pomocou metódy DELETE a táto žiadosť neakceptuje žiadne parametre.

Obsah repozitára

Obsah repozitára je dostupný na adrese <RDF4J_URL>/repositories/<ID>/statements, kde <ID> predstavuje názov repozitára.
Podporované metódy:

- **GET** - načíta dáta z repozitára
- **PUT** - nahradí všetky existujúce dáta v repozitáre. Očakáva sa, že dáta poslané v tejto žiadosti budú obsahovať dokument RDF v jednom z podporovaných formátov RDF.
- **DELETE** - zmaže dáta v repozitári
- **POST** - vykoná aktualizáciu dát v repozitári

Parametre:

- **subj** (voliteľný) - obmedzuje operácie GET a DELETE na dáta, ktoré obsahujú subj, ktorý je zakódovaný na pozícií predmetu
- **pred** (voliteľný) - obmedzuje operácie GET a DELETE na dáta, ktoré obsahujú pred, ktorý je zakódovaný na pozícií predikátu
- **obj** (voliteľný) - obmedzuje operácie GET a DELETE na dáta, ktoré obsahujú obj, ktorý je zakódovaný na pozícií objektu
- **update** (voliteľný) - relevantné len pre operáciu POST. Určuje reťazec aktualizácie SPARQL 1.1, ktorý sa má vykonať. Očakáva sa, že hodnota bude syntakticky platný reťazec aktualizácie SPARQL 1.1
- **context** (voliteľný) - ak je zadaný obmedzuje dáta len na určitý kontext repozitára

Hlavičky žiadosti:

- **Accept**: hodnoty pre GET žiadosť sú typy internetového média podporujúce RDF formáty (viď 2.2.3)
- **Content-Type** - musí byť špecifikované kódovanie akýchkoľvek dát posielaných na server. Sú to hodnoty typov internetového média podporujúce RDF formáty (viď 2.2.3)

Veľkosť repozitára

Velkosť repozitára je dostupná na adrese <RDF4J_URL>/repositories/<ID>/size a predstavuje celkový počet trojíc v repozitári s názvom <ID>.
Podporované metódy:

- **GET** - načíta počet trojíc z repozitára

Parametre:

- **context** (voliteľný) - ak je zadaný obmedzuje výsledný počet len na určitý kontext repozitára

Zoznam kontextov repozitára

Zoznam zdrojov, ktoré sa používajú ako identifikátory kontextu v repozitáry s ID <ID> je dostupný na adrese <RDF4J_URL>/repositories/<ID>/contexts Podporované metódy na túto URL sú:

- **GET** - načíta zoznam zdrojov, ktoré sa používajú ako kontextové identifikátory. Zoznam je naformátovaný ako výsledok dotazu n-tice s jednou premennou contextID, ktorá je viazaná na URI a uzly, ktoré sa používajú ako kontextové identifikátory.

Zoznam menných priestorov

Zoznam menných priestorov, ktoré sa mapujú na daný prefix v rámci repozitára s ID <ID> je dostupný na <RDF4J_URL>/repositories/<ID>/namespaces
Podporované metódy:

- **GET** - načíta zoznam menných priestorov, ktoré boli definované v rámci repozitára.
- **DELETE** - zmaže všetky menné priestory v repozitári

Modifikácia menných priestorov

Deklarácia menného priestoru s prefixom <PREFIX> v repozitári s ID <ID> dostupná na adrese <RDF4J_URL>/repositories/<ID>/namespaces/<PREFIX>.
Podporované metódy:

- **GET** - načíta menný priestor, ktorý bol definovaný pre určitý prefix
- **PUT** - aktualizuje menný priestor so zadaným prefixom, nová hodnota menného priestoru je poslaná ako plain/text v tele žiadosti
- **DELETE** - zmaže menný priestor s daným prefixom

2.4 SPARQL

SPARQL je štandardný dopytovací jazyk RDF, ktorý zodpovedá skupine špecifikácií poskytujúcich jazyky a protokoly na dotazovanie a manipuláciu s RDF grafmi. To znamená, že jazyk SPARQL uľahčuje extrakciu informácií z RDF dát. Podrobnú syntax a sémantiku tohto dotazovacieho jazyka pre RDF definovala W3C [28] kde jádrom SPARQL dotazov je konjunktívna množina tzv. trojitých vzorov (triple patterns). Podobne ako RDF trojica, trojitý vzor má formu predmet, predikát a objekt s rozdielom, že každá časť môže byť premenná. Pre zlepšenie pochopenia, si jazyk SPARQL požičal časť svojej syntaxe od populárneho a široko používaného SQL[15]. Zvláštnosťou jazyka SPARQL je spracovávať RDF grafy a podporovať navigačný prístup k získaniu dát. Cieľom je začať od daného uzla a presúvať sa z uzla na uzol po určitých hranách, kým sa nedosiahne cieľový uzol. Tento prístup je celkom odlišný od vykonávania SQL dotazu, ktoré je založené na spojeniach medzi tabuľkami. Ďalším rozdielom je, že dotazy v jazyku SPARQL nie sú obmedzené na prácu v rámci jednej databázy, federatívne dotazy môžu pristupovať k viacerým dátovým úložiskám (koncovým bodom). Je to možné pretože SPARQL je viac než dotazovací jazyk, je to tiež

[15]Structured Query Language

transportný protokol založený na HTTP, kde je možné pristupovať ku každému koncovému bodu (end-point) prostredníctvom štandardizovanej transportnej vrstvy.

Koncový bod SPARQL je rozhranie pripojené k RDF úložisku (viď 2.3). Prijíma žiadosti cez web a vráti výstup v zvolenom formáte (viz. 2.2.3). Koncový bod sa tiež označuje ako procesor alebo služba, pretože má na starosti spracovanie prijatej žiadosti cez súbor dátového setu. Každý koncový bod má HTTP adresu a riadi sa protokolom SPARQL. Mnohé organizácie poskytli svojim verejným dátam koncové body SPARQL, takže iné webové aplikácie môžu interagovať s ich údajmi. Na oficiálnej webovej stránke W3C je zoznam všetkých týchto organizácií [35]. Napríklad webová aplikácia sa môže pripojiť ku koncovému bodu filmovej databázy RDF a pomocou SPARQL koncového bodu a vyhľadať napríklad herca a všetky jeho filmy v 90. rokoch.

Pred vysvetlením jednotlivých častí SPARQL dotazu a jeho možných variant a rozšírení by som chcel poukázať, že všetky nasledujúce ukážky bude reprezentované na dátovom sete (viď tabuľka 2.1). Syntax SPARQL dotazu je veľmi podobný SQL dotazu s drobnými zmenami. Premenná je identifikovaná znakom (?), za ktorým nasleduje meno danej premennej. Táto premenná bude tvoriť hlavičku výsledného stĺpca a bude obsahovať výsledok alebo výsledky daného dotazu. Podobne ako vo formáte Turtle, je možne pred písaním samotného dotazu deklarovať prefixy, ktoré budú mať rovnaký zmysel ako v samotnom formáte a to, odľahčiť celkovú textovú reprezentáciu údajov. Jednotlivé časti SPARQL dotazu budú vysvetlené na následujúcom jednoduchom dotaze, ktorý získa krstné meno užívateľa s priezviskom Doe.

```
PREFIX blog: <http://exm.com/Blog#>
PREFIX ex: <http://exm.com/terms#>
SELECT ?name
WHERE {
    ?user ex:lastName "Doe" ;
          ex:firstName ?name ;
}
```

Výpis 2.6: Jednoduchý SPARQL dotaz

V tomto dotaze sú dve premenné ?user a ?name, ale vo výsledku je len jedna premenná pretože sa ako jediná vyskytuje v SELECT klauzule (takáto premenná je kvalifikovaná ako významná). Klauzula WHERE dotazu slúži na obmedzenie RDF podgrafu, ktorý zodpovedá podmienkam. Klauzula SELECT vyberá tu časť podgrafu, ktorá nás zaujíma teda v tomto prípade krstné meno. Klauzuly vo WHERE časti sú definované ako štandardný RDF trojitý vzor s možnosťou premenných. Premenné, ktoré nie sú prítomne v klauzule SELECT sa používajú na prepojenie iných trojitých vzorov (podobné SQL joinu) a sú kvalifikované ako nevýznamne. Výsledkom dotazu 2.6 je tabuľka 2.3.

name
"Joe"

Tabuľka 2.3: Výsledok dotazu 2.6

2.4.1 Modifikátory výsledku

Nie je prekvapením, že jazyk SPARQL poskytuje podobne ako SQL rôzne modifikátory výsledku dotazu ako:

- **LIMIT** - rovnako ako v SQL limit stanovuje hornú hranicu počtu vrátených výsledkov. Ak je počet výsledkov väčší ako limit, vráti sa maximálne limitovaný počet výsledkov. Limit nemôže byť negatívne číslo.
- **DISTINCT** - eliminuje duplicitné riešenia vo výsledku teda ak nám dotaz vráti rovnaké trojice viackrát, vo výsledku sa daná trojica objaví len raz
- **OFFSET** - preskakuje zadaný počet výsledkov, v kombinácií s LIMITom sa používa na stránkovanie (pagination), kedy by načítanie veľkého množstva dát naraz mohlo spôsobiť zhoršenie výkonu prípadne výpadok stránky. Použije sa tak postupne načítanie v rámci jednej stránky, kedy sa načíta napríklad prvých 200 (LIMIT) trojíc a s každou ďalšou stránkou sa načíta ďalších 200 trojíc (LIMIT + OFFSET).
- **ORDER BY** - určuje poradie postupnosti výsledkov, za klauzulou ORDER BY nasleduje typ zoradenia buď ASC(), ktorý zoraďuje výsledky vzostupne alebo DESC(), ktorý zoraďuje výsledky zostupne. Keďže jednotlivé zdroje majú rôzne typy ako bolo vysvetlene v sekcii 2.2.1, SPARQL stanovuje, poradie týchto typov, ktoré by inak neboli zoradené a to nasledovne od najnižšej váhy po najvyššiu:

  1. Prázdne uzly
  2. URIs
  3. Literály

  Obyčajný literál je nižšie ako typovaný literál.
- **GROUP BY** - zoskupuje množiny výsledkov podľa agregačných funkcií ako AVG(), MIN(), MAX() alebo COUNT(). Tieto agregačné funkcie as vyskytuje v klauzule HAVING, ktorá pomocou týchto funkcií špecifikuje obmedzujúcu podmienku výsledného zoskupenia.

Ilustračný príklad s použitím týchto modifikátorov by mohol vyzerať ako v ukážke 2.7

```
PREFIX blog: <http://exm.com/Blog#>
PREFIX ex: <http://exm.com/terms#>
SELECT ?name
WHERE {
    ?user ex:lastName ?firstName ;
          ex:firstName ?name ;
          blog:isFollowing ?user2 ;
          blog:owner ?blog .
    ?user2 ex:firstName "Mary" .
}
GROUP BY ?name
HAVING (COUNT(?blog) > 2)
LIMIT 1
OFFSET 1
```

Výpis 2.7: SPARQL dotaz s využitím modifikátorov

V tomto dotaze sme hľadali užívateľa, ktorý spĺňa klauzulu GROUP BY, tým, že vlastní aspoň dva blogy a obmedzili sme to na jeden výsledok, ktorý ma byť druhý v poradí. Klauzula GROUP BY je splnená dvoma užívateľmi ako vidíme v dátovej tabuľke 2.2. Ak tento fakt obmedzíme na jeden výsledok s offsetom jedna tak dostaneme výsledok, ktorý obsahuje tabuľka 2.4.

name
"Will"

Tabuľka 2.4: Výsledok dotazu 2.7

SPARQL tiež poskytuje ďalšie dodatočné kľúčové slová:

- **OPTIONAL** - povoľuje získať dáta aj v prípade absencie nejakej časti trojitého vzoru. V SQL jazyku OPTIONAL zodpovedá OUTER JOINu.
- **FILTER** - použitím kľúčového slova FILTER a regulárneho výrazu je možné testovať či RDF literály splňujú určitú podmienku
- **UNION** - slúži na prepojenie jednotlivých grafov úložiska

2.4.2 Typy dotazov

Doteraz, mali výsledky SPARQL dotazov podobu množiny n-tíc ako v SQL, ktoré začínali kľúčovým slovom SELECT. Tento typ je prvým z typov dotazov a okrem neho budu v tejto sekcií vysvetlené ďalšie tri možné typy dotazov a to ASK, CONSTRUCT a DESCRIBE. Rozdiel medzi týmito typmi dotazov je v ich využití, ktoré budú vysvetlené pri konkrétom type a taktiež vo výsledku ktoré vracajú.

SELECT

SELECT je základný typ dotazu, má rovnakú funkcionalitu ako v SQL. Výsledkom SELECT dotazu je tabuľka s n stĺpcami, podľa toho na koľko premenných sa dotazujeme. Napríklad v ukážke 2.6 je výsledkom tabuľka s jedným stĺpcom name ale všeobecne sa tam môže vyskytovať ľubovolný počet premenných.

ASK

ASK je typ dotazu, ktorý poskytuje výsledok pozostávajúci z jednej bunky typu boolean, teda true alebo false, podľa toho či daný SPARQL dotaz existuje v rámci dátového setu alebo nie. Používa sa len na testovanie či existuje riešenie pre daný dotaz.

```
PREFIX blog: <http://exm.com/Blog#>
PREFIX ex: <http://exm.com/terms#>
ASK {
  ?user ex:lastName "Doe" ;
        blog:isFollowing ?user2 .
  ?user2 ex:firstName "Mary" .
}
```

Výpis 2.8: SPARQL ASK dotaz

CONSTRUCT

CONSTRUCT má podobnú syntax ako SELECT dotaz okrem časti, ktorá nasleduje za kľúčovým slovom. Umožňuje vytvoriť vlastný graf v kombinácii s hodnotami v ktoré sa nachádzajú v RDF úložisku. Výsledkom CONSTRUCT dotazu je trojica.

DESCRIBE

DESCRIBE dotaz sa pýta na akékoľvek informácie spojené so špecifickou premennou. Hlavný účelom tohto dotazu je zistiť zdroje spojené s danou premennou bez znalosti ich schémy. Tento dotaz je užitočný hlavne ak nepoznáme schému RDF dát a chceli by sme zistiť všetky trojice, ktoré sú spojené s danou premennou. Výsledok DESCRIBE dotazu je podobne ako pri SELECTe tabuľka trojíc.

```
PREFIX ex: <http://exm.com/terms#>
DESCRIBE ?user
WHERE {
    ?user ex:firstName "Joe" .
}
```

Výpis 2.9: SPARQL DESCRIBE dotaz

Výsledok dotazu môžme vidieť v tabuľke 2.5. Vidíme, že sme dostali všetky trojice, ktoré sa týkali užívateľa s krstným menom Joe.

predmet	predikát	objekt
blog:JD	ex:firstName	"Joe"
blog:JD	ex:lastName	"Doe"
blog:JD	blog:owner	"TestBlog1"
blog:JD	blog:owner	"TestBlog2"
blog:JD	blog:isFollowing	blog:MS

Tabuľka 2.5: Výsledok DESCRIBE dotazu 2.9

2.4.3 SPARQL 1.1 UPDATE

Ako je vidieť v predošlých sekciách jazyk SPARQL slúži len ná čítanie informácií z RDF databázy, neslúži na modifikáciu dát. Vznikol preto SPARQL 1.1 UPDATE, ktorý je sprievodným jazykom k SPARQL, a predpokladá sa, ze sa bude používať v spojení s jazykom SPARQL. Ako napovedá názov SPARQL 1.1 UPDATE je určený na aktualizáciu RDF grafov v grafovom úložisku. Grafové úložisko je definované ako premenlivý kontajner RDF grafov spravovaný RDF úložiskom, ktoré príma a spracováva žiadosti o aktualizáciu. Grafové úložisko sa štandardne skladá z jedného anonymného grafu a prípadne nula alebo viac pomenovaných grafov (identifikovaných podľa URIs), na ktoré sa budem odkazovať ako kontext RDF dát. [6]

SPARQL 1.1 UPDATE poskytuje dve kategórie operácií, ktoré je možné na grafové úložisko uplatniť – aktualizácia grafu a správa grafu. Zatiaľ čo prvá operácia súvisí s pridávaním (insert) a mazaním (delete) obsahu (trojíc) do/z RDF grafu, druhá súvisí s pridávaním a mazaním celého kontextu grafového úložiska.

S operáciou aktualizácie grafu sa spájajú klauzuly INSERT DATA a DELETE kde podobne ako pri klauzule SELECT špecifikujeme trojicu, ktorá ma byť pridaná alebo zmazaná.

```
PREFIX blog: <http://exm.com/Blog#>
PREFIX ex: <http://exm.com/terms#>
INSERT DATA {
    blog:GB ex:firstName "Amanda" ;
                ex:lastName "Blin" .
} ;
DELETE WHERE {
    ?s ?p ?o ;
            ex:lastName "Smith" .
}
```

Výpis 2.10: INSERT a DELETE dotaz

Kapitola 3

Technológie pre tvorbu klientskych webových aplikácií

Nasledujúca kapitola je venovaná jazyku Javacript kde je vysvetlené na čo sa tento jazyk používa. Táto kapitola taktiež obsahuje pojmy, ktoré súvisa s týmto jazykom a nakoniec sú popísané jednotlivé rámce JavaScriptu, ktoré sa používajú pri tvorbe klientskych webových aplikácií.

3.1 JavaScript

JavaScript je vysokoúrovňoví, dynamický a interpretovaný jazyk. To znamená, že kód napísaný v JavaScripte neprechádza prechodnou fázou kompilácie, v ktorej je zdrojový kód transformovaný do strojového jazyka, ktorý je pre CPU ľahko spracovateľný. Namiesto toho je JavaScript interpretovaný za behu počítačom, ktorý ho spracováva. Využíva sa hlavne na vývoj klientských častí aplikácií ale je ho možne využiť aj na backend. Používa sa tiež pri vývoji videohier, pri tvorbe desktopových a mobilných aplikácií a pri programovaní serverov s exekučnými prostrediami, ako je Node.js. Na interakciu s webovou stránkou sa používa model objektu dokumentu (DOM). Je dobre známy hlavne ako skriptovací jazyk pre webové stránky, používa ho aj mnoho prostredí bez prehliadača, ako napríklad Node.js, Apache CouchDB a Adobe Acrobat.[14] Dôvod prečo je JavaScript populárny medzi frontendovými vývojármi je, že umožňuje interaktívnejší a dynamickejší prístup. Bez JavaScriptu by sme na webe mali iba HTML a CSS. Tieto samotné by nás obmedzovali na niekoľko implementácií webových stránok. 90% (ak nie viac) webových stránok by bolo statických a mali by sme len dynamické zmeny, ako sú animácie, ktoré poskytuje CSS. JavaScript podporuje matematické výpočty, umožňuje dynamicky pridávať obsah HTML do modelu DOM, vytvára dynamické deklarácie štýlu, načítava obsah z inej webovej stránky. Dokáže taktiež reagovať na užívateľské akcie, kliky myšky, stlačenie tlačítka na klávesnici a mnohé ďalšie. [9]

3.1.1 JSX

JSX je skratka pre JavaScript XML a je to rozšírenie React syntaxu jazyka JavaScript. Myšlienkou je, že syntax je preložená do normálnych objektov JavaScriptu, ktoré dokáže JavaScript spracovať. Je to užitočné, pretože umožňuje písať HTML/XML v rovnakom súbore ako JavaScript to znamená, že je možné využiť celú silu JavaScriptu. [17]

3.2 Jednostránková webová aplikácia

Jednostránková aplikácia (SPA[1]) je webová aplikácia, ktorej celý obsah je tvorený jednou stránkou. To umožňuje rýchlejšiu a užívateľský prívetivejšiu manipuláciu s aplikáciou. Vytvorením jednostránkovej aplikácie je možné získať veľmi bohaté rozhranie s vysokým výkonom. Tento druh vývoja umožňuje písať menej kódu na strane servera a viac kódu na strane klienta, čo poskytuje lepšiu používateľskú skúsenosť. Spôsob akým to SPA dosahuje, je úplným načítaním webovej stránky pri prvom načítaní užívateľom a následne keď na stránke dôjde k zmenám, zmení sa len konkrétna časť webovej aplikácie, čím sa vyhne nutnosti obnovovať celú stránku. [10]

3.3 Šablóna syntaxe

Mnoho moderných JavaScriptových knižníc využíva takzvaný syntax šablóny alebo template syntax. Syntax šablóny je prostriedkom na definovanie dynamickej časti HTML[2] štruktúry. Šablóna sleduje lokálny stav komponenty naviazaním údajov v stave na používateľské rozhranie, ktoré šablóna predstavuje. [33], [2]

3.4 Objektový model dokumentu

Objektový model dokumentu alebo DOM[3] je programovacie rozhranie pre webové dokumenty. Poskytuje spôsob reprezentácie stránky, ktorý programom umožňuje meniť štruktúru, štýl a obsah. To je dôvod, prečo JavaScript môže zmeniť stránku a prečo je taký dôležitý pre JavaScriptové rámce. DOM reprezentuje dokument ako uzly a objekty – spôsob akým môžu programovacie jazyky interagovať so stránkou. [13]

3.5 JavaScriptové rámce

Rámec alebo framework je kolekcia preddefinovaného kódu, ktorá pomáha vývojárom dosiahnuť benefity, ktoré čistý JavaScript sam o sebe neponúka. Flexibilita rámcov sa líši v závislosti od zvoleného rámca. Niektoré určujú spôsob, akým musí byť stránka postavená, zatiaľ čo iné majú lepšiu adaptabilitu.

Často sa pojem rámec mýli s pojmom knižnica, rozdiel je, že knižnica ponúka kolekciu kódu navrhnutú pre špecifické prípady použitia ako napríklad knižnica moment.js, ktorá sa používa na prácu s dátumami, zatiaľ čo rámec ponúka viacero funkcionalít a formuje kostru aplikácie. To znamená, že rámec poskytuje potrebný štandardný alebo tzv. „boilerplate“ kód, ktorý je potrebný pre väčšinu projektov a taktiež poskytuje aj vopred pripravené riešenia bežných problémov pri vývoji s daným programovacím jazykom, v tomto prípade JavaScriptom. Cieľom rámca je tiež zabezpečiť, aby vývojári dodržiavali pravidlá, ktoré daný rámec poskytuje. [4]

[1] Single Page Application
[2] HyperText Markup Language
[3] Document Object Model

3.5.1 React

React patrí medzi ľahké rámce JavaScriptu, ktorý je navrhnutý na postupnú adopciu, čo znamená, že nemusíme vyvárať projekt, v ktorom použijeme 100% react kódu, ale môžeme v projekte použiť toľko Reactu koľko chceme. React používa JSX predstavený v sekcií 3.1.1, nie je to ale povinné. React je jedinečný v tom, že spája užívateľské rozhranie komponentov a ich správania. Aplikácie napísane v Reacte sú vo všeobecnosti postavené na jedinom prvku HTML, ktorý sa často nazýva koreňový prvok. Prvky v Reacte sú nemenné. Jediný spôsob, ako zmeniť prvok v Reacte, je vykreslenie nového prvku. [18]

Komponenty v Reacte sú v skutočnosti funkcie alebo triedy JavaScriptu, ktoré vracajú kód JSX, ktorý sa preloží do konečných objektov JavaScriptu, ktoré sa zobrazia v aplikácii. Lokálny stav vnútri komponentov v Reacte je považovaný za nemenný, aj keď v skutočnosti nie je. So stavom by sa malo zaobchádzať týmto spôsobom, pretože ak jeden zmutuje stav, existuje riziko, že iná nemenná zmena stavu odstráni mutáciu, ktorá bola vykonaná. Od kedy je React tak populárny, medzery na trhu vypĺňajú moduly vytvorené komunitou, ktoré využívajú silu Reactu [16]. React je najpopulárnejší rámec JavaScriptu s vyše 14 miliónov stiahnutiami týždenne so správcom balíkov Node [20].

3.5.2 Angular

Angular je rámec JavaScriptu vytvorený spoločnosťou Google na riešenie problémov na úrovni Googlu. Platforma je navrhnutá tak, aby umožnila viacerým vývojárom hladko spolupracovať na rovnakom projekte. Angular prichádza ako viac úplný rámec s preddefinovanými predvolenými nastaveniami, ako je sieťové pripojenie, smerovač a výber jazyka. Tieto predvolené hodnoty sa neustále testujú a vzájomne overujú, aby sa zabezpečilo, že platforma Angular sa pohybuje vpred stabilným a spoľahlivým tempom [7]. Jednou z týchto preddefinovaných predvolených hodnôt je Typescript, čo je podmnožina JavaScriptu, predstaveného v sekcií 5.2.2.

Komponenty sa v angulari označujú ako direktívy. Direktívy predstavujú len značky na DOM prvkoch, ktoré Angular môže sledovať a spájať s určitým chovaním. Preto Angular oddeľuje UI časť komponentov ako atribúty HTML tagov a ich správanie vo forme JavaScript kódu. To je hlavný rozdiel, medzi Angularom a Reactom s Vue.

Jednou z kľúčových funkcií v Angular je syntax šablóny, ktorá predstavuje spôsob, ako rýchlo vytvárať zobrazenia užívateľského rozhrania s jednoduchou a výkonnou syntaxou. Výkonný nástroj v Angular je ich prístup k náväznosti dát. Náväznosť dát je spojovací most medzi lokálnym stavom v modeli a zobrazením. Model spracováva logiku komponentu a lokálne premenné. Existuje viacero spôsobov, ako spojiť údaje medzi modelom a zobrazením, môže byť buď jednosmerný z modelu do zobrazenia, jednosmerný z pohľadu do modelu alebo obojsmerný, kde sa údaje môžu posielať oboma spôsobmi. Ďalšou funkciou je Angular CLI, čo je nástroj príkazového riadka, ktorý umožňuje rýchlo a efektívne vytvoriť aplikáciu, pridávať komponenty a testy a potom ju okamžite nasadiť. Angular patrí medzi najstarší rámec JavaScript a taktiež aj medzi jeden z najpopulárnejších s vyše 2 miliónov stiahnutiami týždenne pomocou služby Node Package Manager [19].

3.5.3 Vue

Vue je JavaScriptový rámec vytvorený z oboch rámcov Angularu a Reactu kombinovaných do jedného rámca. Je silne inšpirovaný oboma a prijal mnoho populárnych funkcií Angularu a Reactu a tiež sa pokúsil vytvoriť lepšie riešenia vecí, o ktorých sa domnievajú, že v Angulari a Reacte nepracovali až tak dobre. Vue má k Reactu bližšie ako ku Angularu z dôvodu, že základnou myšlienkou oboch rámcov je používanie komponentov a virtuálneho modelu DOM. Taktiež užívateľské rozhranie a správanie je časť komponentov.

Jednou z významných výhod Vue je malá veľkosť rámca, pretože neobsahuje veľa funkcií hneď po vybalení, ale funkčnosť sa dá ľahko rozšíriť pomocou rôznych riešení tretích strán. Často sa porovnáva s Angularom, čo je monolitický rámec, ktorý má množstvo vstavaných funkcií, ktoré sa v aplikácii pravdepodobne vôbec nepoužijú. Samozrejme, tzv. tree-shaking (eliminácia mŕtveho kódu) umožňuje eliminovať nepoužívaný kód, ale veľkosť rámca je stále väčšia v porovnaní s tým, čo ponúka Vue. Plne vybavený projekt Vue.js s Vuex + Vue-router ma veľkosť 30 kb gzip. Zároveň má predpripravená aplikácia skompilovaná a vygenerovaná pomocou angular-cli veľkosť 130 kb zazipovaná. Jeho kompaktná veľkosť a schopnosť zahrnúť moduly tretích strán na rozšírenie funkčnosti robí z Vue.js múdrejšiu voľbu pre tých, ktorí sa starajú o zmenšenie veľkosti, a teda o zvýšenie rýchlosti webovej aplikácie.

Vue využíva syntax šablóny na vytvorenie svojich zobrazení (views) užívateľského rozhrania, ktoré sú veľmi podobné čistému HTML s niekoľkými doplnkami na oveľa lepšie spracovanie dynamického obsahu [30]. Šablóny vo Vue sú validné HTML, tieto šablóny sú preložené do vykresľovacích funkcií virtuálneho modelu DOM, ktoré v kombinácii s ich systémom reaktivity dokážu zistiť minimálny počet komponentov, ktoré musí znova vykresliť a použiť minimálny počet DOM manipulácie pri zmene stavu aplikácie čo je vylepšenie oproti Reactu a pomáha ku zlepšeniu výkonnosti. Aj keď v rámci Vue existuje menej knižníc, existuje prosperujúci ekosystém poháňaný komunitou. V prípade populárnejších modulov, ako sú globálne manipulátory stavu (Vuex) a smerovanie (Vue-router), tie Vue podporuje a stará sa o nich aby sa zabezpečilo, že budú aktualizované. Vue taktiež patrí medzi najpopulárnejšie rámce JavaScriptu s aktuálne výše milión stiahnutiami týždenne službou NPM[4] [21].

Komponenty

Ako bolo spomenuté komponenty hrajú vo Vue frameworku dôležitú úlohu, sú to základne prvky z ktorých je tvorená aplikácia. Tieto kompomennty môžu byť rôzne veľké a je len na užívateľovi ako sa ich rozhodne použiť. Stránka sa môže skladať napríklad z komponenty bočné menu, komponenty vrchný panel a komponenty pre obsah, tieto komponenty sú znovupoužitelné v rámci aplikácie a preto je aplikácia ľahko škálovatelná. Tieto komponenty sú usporiadané do stromu vnorených komponentov. Je to veľmi podobné princípu ako sa vkladajú prvky HTML s tým rozdielom, že Vue implementuje svoj vlastný komponentový model, ktorý umožňuje zapuzdriť vlastný obsah a logiku do každej komponenty. [32]

Každá Vue komponenta musí byť „registrovaná“ aby Vue vedelo nájsť jej implementáciu ak sa s ňou v šablóne stretne. Existujú dva spôsoby ako registrovať komponenty vo Vue a to buď `globálne` alebo `lokálne`. Globálne komponenty je možné v aplikácií použiť kdekoľvek bez toho aby sa museli importovať v súbore v ktorom sa nachádzajú. Príklad takejto komponenty môže byť napríklad bočné menu, ktoré sa vyskytuje na každej stránke aplikácie kedy by sme sa vyhli opakovanému importu komponenty pri každom použití. Hoci

[4]Node Package Manager

použitie globálnych komponentov sa môže zdať pohodlné majú niekoľko nevýhod. Globálna registrácia robí vzťahy menej explicitnými a pri väčších aplikáciach to môže spôsobiť problém s udržateľnosťou. Lokálne registrácia zahŕňa dostupnosť registrovaných komponentov len pre aktuálnu komponentu. [31]

Keďže celá stránka je rozdelená do menších časti, komponentov, tak je počas vývoja potrebné zdieľať dáta medzi komponentami. Komunikácia medzi komponentami je teda nutnosťou a má viacero podôb. Hlavné dva spôsoby sú:

- **komunikácia medzi rodičom a potomkom** - pri tomto type komunikácie rodičovská komponenta odovzdáva údaje potomkovi pridaním argumentu do deklarácie komponenty. Predanie dát je možné buď staticky alebo dynamicky. Tieto dáta sú potom dostupné v potomkovi ako props hodnoty. [27]
- **komunikácia medzi potomkom a rodičom** - pri tomto type rodič odpočúva udalosti, ktoré potomok posiela pomocou metódy $emit

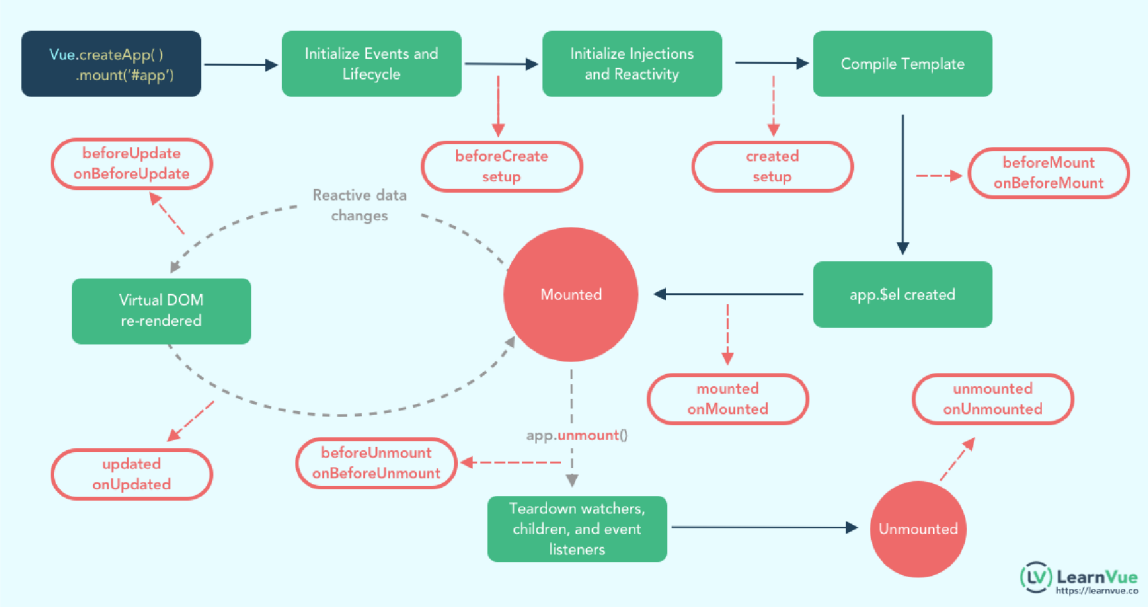
Životný cyklus komponenty

Každá inštancia komponenty Vue pri svojom vytvorení prechádza sériou inicializačných krokov, napríklad potrebuje nastaviť pozorovanie údajov, zostaviť šablónu, pripojiť inštanciu k modelu DOM a aktualizovať DOM pri zmene údajov. Zároveň spúšťa funkcie nazývané háky životného cyklu, čo používateľom dáva možnosť pridať svoj vlastný kód v konkrétnych fázach komponenty.

Vo Vue je životný cyklus komponenty viac prepracovaný ako v Reacte, kde sú na vyber len štyri možnosti a to render(), componentDidMount(), componentDidUpdate() a componentWillUnMount().

Vue dáva vývojárom viac možnosti ako pracovať s komponentou tým, že má výber veľké množstvo hákov. Ako je uvedené na obrázku 3.1 Vue ponúka tieto háky:

- **beforeCreate** - volá sa okamžite po inicializácii inštancie, po vyriešení rekvizít, pred spracovaním iných možností, ako napríklad data() alebo computed
- **created** - volá sa, keď inštancia dokončí spracovanie všetkých možností súvisiacich so stavom
- **beforeMount** - volá sa tesne pred nasadením komponenty. Keď sa zavolá tento hook, komponenta dokončila nastavenie svojho reaktívneho stavu, ale ešte neboli vytvorené žiadne uzly DOM. Prvýkrát sa chystá spustiť efekt vykreslenia modelu DOM
- **mounted** - volá sa po nasadení komponenty. Tento hák sa zvyčajne používa na vykonávanie vedľajších efektov, ktoré si vyžadujú prístup k vykreslenému DOM komponentu, alebo na obmedzenie kódu súvisiaceho s DOMom na klienta v aplikácii vykreslenej serverom
- **beforeUpdate** - volá sa predtým ako sa komponent chystá aktualizovať strom DOM
- **updated** - volá sa po tom, čo je strom DOM aktualizovaný
- **beforeUnmount** - volá sa pred odpojení inštancie z DOM modelu
- **unmounted** - volá sa po odpojení inštancie z DOM modelu



Obr. 3.1: Životný cyklus Vue komponenty prevzaté z [12]

Vue CLI

Podobne ako angular aj vue ponúka nástroj príkazového riadku. Cieľom Vue CLI[5] je stať sa štandardným nástrojovým základom pre ekosystém Vue. Zabezpečuje, aby rôzne nástroje na zostavovanie fungovali hladko spolu s rozumnými predvolenými nastaveniami, takže sa je možné sústrediť na písanie aplikácie namiesto toho, aby sa trávili zbytočné hodiny s konfiguráciami. Zároveň stále ponúka flexibilitu na úpravu konfigurácie každého nástroja [34].

[5]Command line interface

Kapitola 4

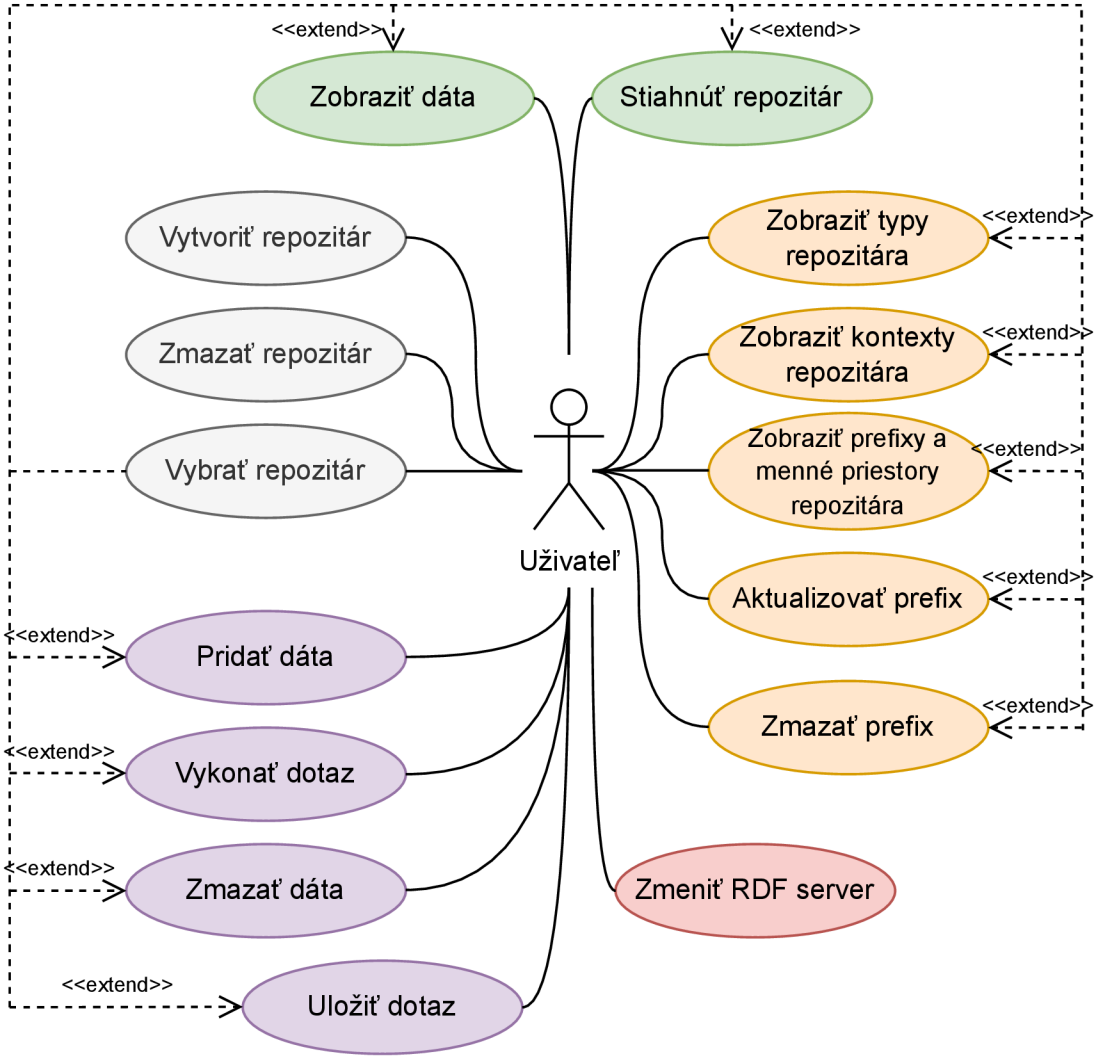
Návrh implementácie

V tejto kapitole je čitateľovi bližšie popísane použitie aplikácie, ktoré je doplnené o diagram prípadov použitia. Taktiež sa v tejto kapitole nachádza popis vybraných technológií pre klientsku časť, serverovú časť ako aj tvorbu návrhu užívateľského rozhrania. Ďalej je predstavený podrobný návrh užívateľského rozhrania a nakoniec návrh testovania.

4.1 Použitie aplikácie

Po otvorení aplikácie sa užívateľ ocitne na stránke s repozitármi, kde sa načítajú repozitáre vo forme tabuľky, ktoré sú dostupné na RDF serveri. Na tejto stránke bude môcť užívateľ spravovať repozitáre, teda vytvárať nové repozitáre, mazať, prípadne prechádzať repozitáre v tabuľke kde si ich bude môcť zoraďovať podľa rôznych atribútov a v neposlednom rade vybrať repozitár s ktorým bude v rámci celej aplikácie pracovať. Po vybraní repozitára ho to presmeruje na kartu `About Repository`, ktorá v hornej časti obsahuje informácie o danom repozitári a v spodnej časti je preklikávacie menu kde sa užívateľ môže prekliknúť medzi položkami `Types`, `Context` a `Namespaces`. V prvých dvoch prípadoch sa zobrazí len tabuľka s daným obsahom podľa názvu položky v menu, čiže typy repozitára alebo kontexty repozitára. V poslednom prípade sa zobrazia dve tabuľky jedna s obsahom prefixov a im prisluchajúcim namespacov a druhá pre modifikáciu tejto tabuľky. Uživateľ si bude môcť vybrať prefix, kde dostane na výber z hodnôt v tabuľke. Po vybratí daného prefixu sa mu predvyplnia aktuálne hodnoty prefixu a menného priestoru ktoré pomocou tlačítka `DELETE` môže buď vymazať alebo pomocou tlačítka `UPDATE` aktualizovať. Karta `About Repository` je súčasťou karty `Explore` kde sa užívateľovi zobrazí obsah repozitára vo forme trojíc s kontextom v tabuľke, ktorú môže prechádzať a taktiež kliknúť na jednotlivé zdroje, kedy sa po kliknutí zobrazia len trojice s daným zdrojom, taktiež ma možnosť tento zdroj zadať ručne do políčka `Resource`. Vedľa tohto textového poľa sa nachádza tlačítko `DOWNLOAD`, ktoré stiahne po kliknutí obsah repozitára vo vybranom formáte. Súčasťou karty `Explore` bude taktiež karta `Query` kde bude môcť užívateľ zadávať do editora dotazy v jazyku SPARQL (viď 2.4), ktoré sa následne vyhodnotia a zobrazia výsledky podľa toho čo daný dotaz vracia (viď 2.4.2). Na tejto karte bude môcť užívateľ taktiež ukladať jednotlivé dotazy pod názvom ktorý zadá. Tieto dotazy budú uložené do lokálneho úložiska v rámci prehliadača, takže ich užívateľ nestratí po reštartovaní aplikácie. Ďalšia stránka, ktorú bude mať užívateľ na výber je `Update`, kde bude môcť rôzne modifikovať repozitár. V hornej časti stránky bude mať k dispozícií editor, ktorý bude povoľovať zadávať len dotazy na modifikáciu dát (viď 2.4.3). Spodná časť bude rozdelená na dve polovice. Na ľavej strane bude možné pridať

dáta do repozitára v rôznych formátoch. Na pravej strane bude môcť užívateľ mazať dáta z repozitára buď celého pomocou tlačítka `REMOVE ALL` alebo podľa jednotlivých zdrojov alebo kontextu pomocou tlačítka `REMOVE`. Poslednou stránkou bude `System` na ktorej bude možné meniť RDF server.



Obr. 4.1: Diagram prípadov použitia v aplikácií

4.2 Vybrané technológie

V tejto sekcií budú predstavené jednotlivé technológie, ktoré budú použité pri implementácií a tvorbe návrhu jednotlivých časti aplikácie.

4.2.1 Nástroje pre tvorbu užívateľského rozhrania

Existuje veľké množstvo nástrojov pre tvorbu užívateľského rozhrania a pri výbere, ktorý nástroj použiť neexistuje žiadna správna voľba. Keďže témou tejto práce je tvorba webo-

vého rozhrania tak návrh riešenia zohráva dôležitú rolu. Ja som pri výbere nástroja riadil následujúcimi bodmi:

- voľná dostupnosť nástroja
- prechádzajúca skúsenosť s nástrojom
- jednoduchosť vytvárania návrhu
- dostupnosť nástroja na webe

Väčšina nástrojov tieto body spĺňa a preto som sa nakoniec rozhodol pre nástroj Figma, s ktorým som mal predchádzajúcu skúsenosť.

4.2.2 Klientska časť

Táto sekcia popisuje vybrané technológie pre tvorbu klientskej časti aplikácie.

HTML

HTML je značkovací jazyk, nie programovací jazyk keďže neumožňuje vytvárať metódy, ktorý sa používa k vytvoreniu základnej obsahovej kostry webových stránok. Umožňuje užívateľovi vytvárať štruktúru stránky pomocou jeho charakteristických prvkov a to značiek (tagov) a ich vlastnosti (atribútov). Užívateľ si vie obsah rozdeliť do jednotlivých blokov, pridať nadpisy, podnadpisy, odkazy a rôzne iné prvky. Sam o sebe je veľmi jednoduchý a používa sa vo všetkých webových stránkach väčšinou v kombinácií s CSS a JavaScriptom [5].

CSS

Kaskádové štyly alebo CSS boli navrhnuté organizáciou W3C a prvá verzia vznikla už v roku 1996 [37], ktorá umožňovala prácu s písmami, okrajmi a farbami. Odvtedy prešlo CSS viacerými vylepšeniami. Pomocou kaskádových štýlov je možné pridať obsahu stránky teda HTML kódu jeho vzhľad. CSS umožňuje jednotlivým elementom stránky zmeniť farby, veľkosť, typ písma a mnohe ďalšie, taktiež sa používa pri umiestňovaní prvkov na stránke. Medzi dva najpoužívanejšie modely pre umiestňovanie sa používa `Flexbox` a `Grid` model. Pri implementácií sa tak nevyhneme použitiu ako HTML tak CSS a tiež bude použitý Flexbox model, pri vytváraní finálneho umiestnenia komponentov v aplikácií.

Vue

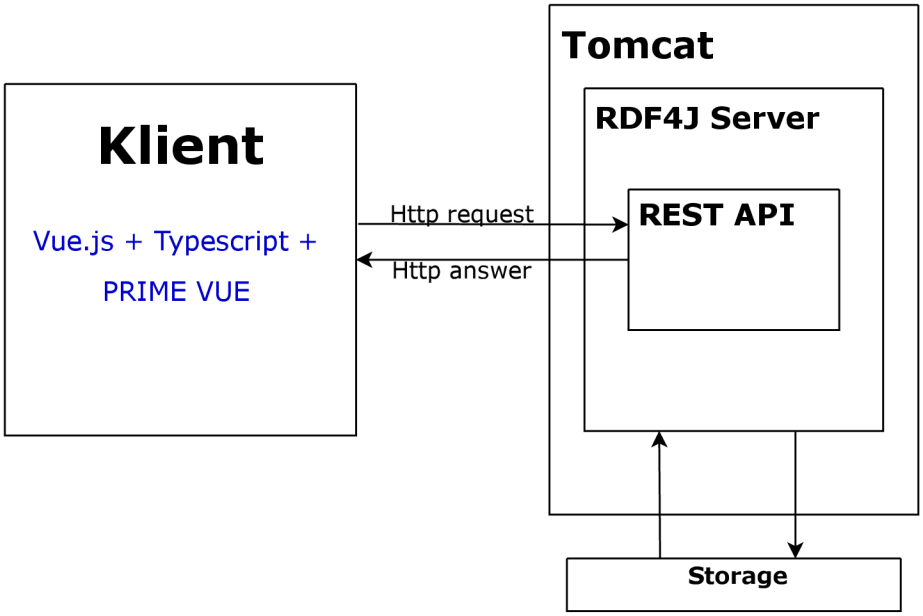
Pri výbere technológie pre tvorbu klientskej časti aplikácie som sa rozhodoval medzi troma najpopulárnejšími JavaScriptovými rámcami, ktoré sú bližšie popísane v sekcii 3.5. V dnešnej dobe sa jednotlivé rámce vo výkone až tak nelíšia, takže to nebol hlavný faktor, ktorý ovplyvňoval výber. Pri implementácií riešenia som chcel zamerať hlavne na jednoduchosť, flexibilitu a moderné riešenie.

Angular patrí medzi najstarší framework z tejto trojice a väčšina nových aplikácii je postavená na Reacte alebo Vue. Angular tiež v základe obsahuje veľké množstvo vstavaných knižníc, ktoré sa vo väčšine nevyužijú a vyžaduje čas na naučenia sa jeho konceptov. Z týchto dôvodov som sa rozhodol Angluar nepoužiť. Voľba teda zostáva medzi Vue a Reactom. Aj keď sú to veľmi podobné frameworky rozhodol som sa v aplikácií použiť Vue z týchto dôvodov:

- flexibilita frameworku
- predchádzajúca skúsenosť s frameworkom
- odporúčanie v zadaní práce

4.2.3 Serverová časť

Úlohou serveru je sprostredkovať kominukáciu medzi klientskou časťou aplikácie a databázou. Server teda musel spĺňať nasledujúce podmienky a to podporu RDF a podporu REST API pre komunikáciu s klientskou časťou. Pre tieto účeli bude použity Eclipse RDF4J framework, ktorý bol dopredu dohodnutý a vybraný pre plnenie tejto úlohy a bude nasadený na Apache Tomcat webový server.



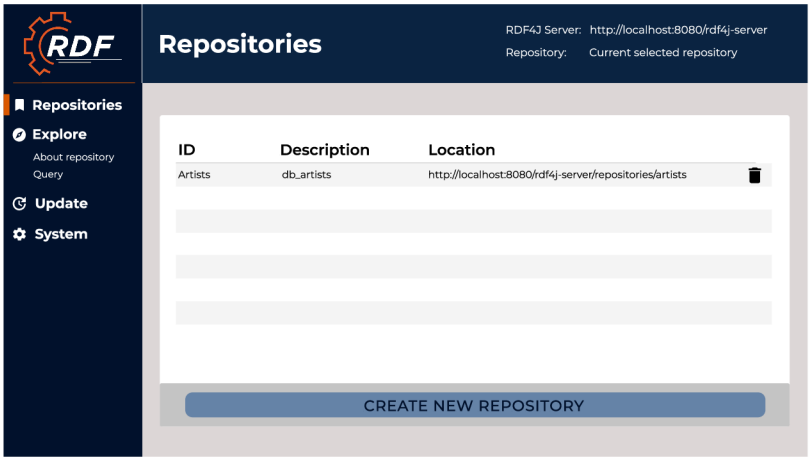
Obr. 4.2: Návrh architektúry aplikácie

4.3 Návrh uživateľského rozhrania

Hlavnou časťou tejto práce je užívateľské rozhranie, keďže užívateľ s touto časťou pracuje je dôležité aby orientácia v aplikácii bola jednoduchá a intuitívna. Preto som využil nástroj Figma, pomocou ktorého som spravil podrobný návrh. Návrh jednotlivých stránok aplikácie vychádza z RDF4J REST API operácií, ktoré je možné na daným repozitárom vykonať. Užívateľské rozhranie výslednej aplikácie je veľmi podobné prvotnému návrhu s menšími zmenami. Všetky stránky boli navrhnuté tak aby mal užívateľ operácie podobného charakteru na jednej stránke a vyhol sa tak zbytočným preklikom a zvýšilo to tak efektivitu prechádzania RDF úložiska. Pre tvorbu jednotlivých komponentov aplikácie bude použitá sada nástrojov Prime Vue, ktorá sa používa pre úpravu vzhľadu webových stránok a to tak, že ponúka už preddefinované štýly.

Repositories

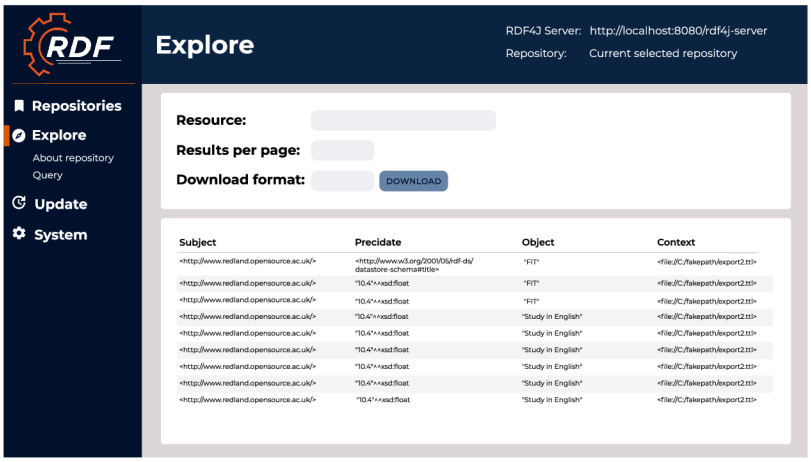
Po spustení aplikácie sa užívateľovi otvorí stránka s repozitármi dostupnými na RDF servery. Stránka sa bude skladať z jednej tabuľky s repozitármi. Pri každom repozitári budú dve tlačítka jedno pre vybranie repozitára a druhé pre zmazanie. Užívateľ si bude môcť ľubovolne zoraďovať repozitáre podľa názvu, popisu alebo lokácie teda url.



Obr. 4.3: Návrh stránky Repositories

Explore

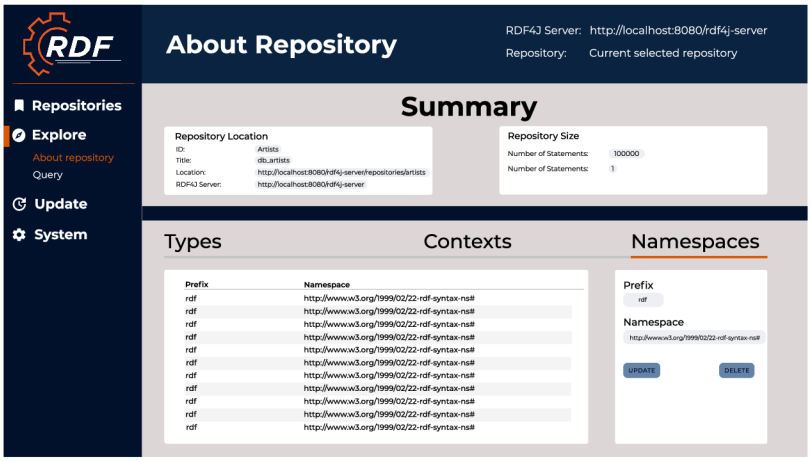
Stránka explore obsahuje tabuľku s obsahom repozitára vo forme trojíc s kontextom. Užívateľ si môže prezerať obsah zoraďovať trojice podľa jednotlivých zdrojov prípadne si filtrovať trojice s konkrétnym zdrojom buď zadaním zdroja do inputu `Resource` alebo kliknutím daný zdroj. Taktiež bude pri každom zdroji v tabuľke tlačítko na skopírovanie zdroja, čo uľahči užívateľovi prácu v rámci aplikácie. Na tejto stránke bude taktiež možné stiahnuť obsah repozitára podľa zvoleného formátu.



Obr. 4.4: Návrh stránky Explore

About Repository

Ako napovedá názov stránky, tak táto stránka bude obsahovať informácie o zvolenom repozitári. V hornej časti stránky budú okná s informáciami o repozitári, v spodnej časti bude preklikávacie menu, ktoré obsahuje ďalšie informácie o repozitári vo forme tabuliek ako jeho typy – sú výsledkom dotazu `SELECT DISTINCT ?type WHERE  ?subj rdf:type ?type` , kontexty – predstavujú všetky kontexty respektívne pomenované grafy, ktoré sa v repozitári nachádzajú a menné priestory spolu s prefixmi, ktoré slúžia pre úpravu zdrojov. Pri tejto poslednej tabuľke bude taktiež okno, ktoré bude umožňovať užívateľovi editovať alebo mazať prefixy spolu s menným priestorom.



Obr. 4.5: Návrh stránky About Repository

Query

Kliknutím na odkaz `Query` v paneli bočného menu sa užívateľ dostane na stránku dotazovania. Bude mať k dispozícii editor do ktorého môže zadávať SPARQL dotazy, jednotlivé dotazy si bude môcť uložiť pre neskoršie použitie.

Uložiť dotaz bude povolené iba vtedy, ak bude do susedného textového poľa zadaný názov. Po kliknutí na tlačítko `SAVE QUERY` sa dotaz uloží pod daným názvom. Ak daný názov existuje, zobrazí sa alert so správou, že dotaz s daným názvom už existuje podobne ako pri chýbajúcom názve. Dotazy sa ukladajú do lokálneho úložiska prehliadača.

Po kliknutí tlačítka `SAVED QUERIES` sa zobrazí modal s uloženými dotazmi vo forme tabuľky kde každý riadok tabuľky bude obsahovať názov dotazu, za ktorým nasledujú tri tlačítka:

- **Select** - vyberie daný dotaz, zatvorí modal s uloženými dotazmi a jeho hodnotu vloží do editora
- **Edit** - otvorí ďalší modal s poliami názov dotazu a hodnota dotazu kde môže uživateľ upraviť daný dotaz
- **Delete** - odstráni uložený dotaz z lokálneho úložiska a pre bezpečnosť sa zobrazí dialógové okno s potvrdením

Obr. 4.6: Návrh stránky Query

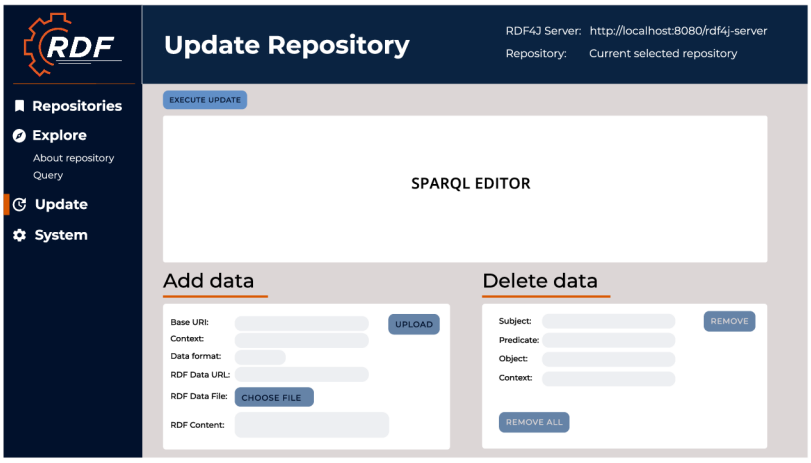
Update

Stránka `Update` sa skladá z troch častí a to časť `Add Data`, ktorá umožňuje modifikovať obsah repozitára a to viacero spôsobmi, časť s editorom kde užívateľ môže pridať obsah pomocou SPARQL 1.1 Update dotazov, teda sú povolené iba dotazy INSERT DATA a DELETE, viac info v sekcií 2.4.3 a posldná časť `Delete Data` na mazanie obsahu repozitára. Spôsoby ktorými je možné pridať obsah v časti `Add Data` sú:

- zadanie adresy URL s RDF obsahom
- vloženie lokálneho súboru
- zadanie serializovaných údajov RDF do textovej oblasti

Taktiež je možné špecifikovať základne URI a kontext pre trojice. Kontext sa dá predstaviť ako štvrtý prvok každej trojice RDF, ktorý špecifikuje graf v rámci úložiska. V rámci všetkým možností pridania obsahu je potrebné zvoliť formát obsahu, ktorý je vkladaný.

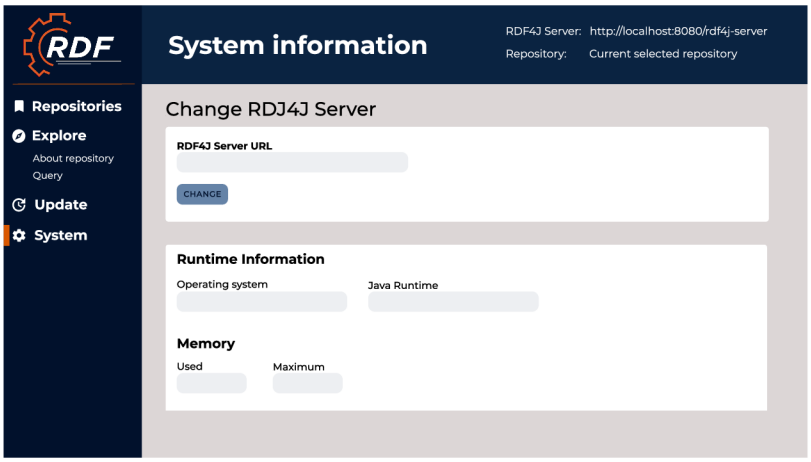
Časť `Delete Data` bude obsahovať štyri polia pre predmet, predikát, objekt a kontext kde bude možné zadať hodnoty ale len v kombinácií jedného zdroja z trojice a kontextu. Po kliknutí na `REMOVE` sa potom z úložiska odstránia všetky trojice, ktoré zodpovedajú daným hodnotam. V tejto časti je tiež tlačítko `REMOVE ALL`, ktoré zmaže celý obsah repozitára. Po stlačení sa zobrazí dialógové okno s potvrdením. Vo všetkých troch častiach stránky sa nachádza tlačítko `CLEAR`, ktoré resetuje jednotlivé polia častí do vychádzajúcej pozície.



Obr. 4.7: Návrh stránky Update

System

Stránka `System` bude slúžiť na zmenu RDF serveru.



Obr. 4.8: Návrh stránky System

4.4 Návrh testovania

Testovanie bude prebiehať dvoma spôsobmi. Keďže ide o testovanie grafického rozhrania rozhodol som sa použiť automatické testovanie pomocou nástroja Cypress a manuálne testovanie s dopredu pripravenými scenármi, ktoré budu predané skupine užívateľov rôznych vekových kategórii z rôznych oblasti. Pomocou automatického testovania bude preverená základná funkcionalita aplikácie a pomocou manuálneho testovania bude preverené rozmiestnenie jednotlivých prvkov aplikácie. Užívatelia budú pozorovaní a taktiež im bude po otestovaní aplikácie predaný krátky dotazník kde budú hodnotiť zložitosť jednotlivých úloh. Na konci dotazníka budú mať priestor kde môžu napísať celkový dojem z aplikácie, prípadne nápady na zlepšenia. Podľa týchto výsledkov budu v aplikácií vykonané prípadne zmeny.

Kapitola 5

Implementácia

Zámerom tejto práce je vytvoriť webové rozhranie pre prechádzanie úložiska RDF, preto bude táto kapitola obsahovať hlavne implementáciu klientskej časti aplikácie kde budú predstavené postupy, ktoré boli použité pri implementácií ako napríklad získanie dát prostredníctvom rozhrania API, ich spracovanie a následne zobrazenie. Ďalej bude predstavená štruktúra aplikácie a tiež budú popísane niektoré hlavné časti ako globálne úložisko (viď 5.2.3), smerovanie (viď 5.2.3) a popis jednotlivých komponentov aplikácie (viď 3.5.3).

5.1 Konfiguŕacia serverovej časti

Ako bolo spomenuté v sekcií 4.2, pre učely serveru je použitý RDF4J Server. Tento server má nasledujúce požiadavky:

- Java 8 Runtime Enviroment alebo novšiu
- Java Servlet Container, ktorý podporuje Java Servlet API 3.1 a Java Server Pages (JSP) 2.2 alebo novšie.

Keďže je tento server použitý ako koncový bod SPAQRL na zasielanie dotazov a následne spracovanie odpovedí je potrebné aby tento server aktívne bežal počas používania aplikácie, takže ho stačí nasadiť na Java Servlet Container, ktorý je v tomto prípade Apache Tomcat a službu spustiť. Apache Tomcat bol stiahnutý z ich oficiálnej stránky[1] a potom nainštalovaný na lokálny systém. Ďalším krokom bolo nasadenie RDF4J frameworku na Tomcat server. Najprv bolo treba stiahnúť RDF4J SDK z ich oficiálnej stránky[2], ktoré obsahuje súbor `rdf4j-server.war` tento war súbor bol potom nasadený na Tomcat službu.

5.1.1 Rozhranie API

API ako prostredník je na strane serveru a najčastejšie komunikuje medzi niekym kto si žiada nejaké informácie teda klient a databázou, ktorá dané informácie uchováva. Spracováva tak dotazy od klienta ne konkrétny koncový bod v tomto prípade SPARQL endpoint na ktorom existuje inštancia RDF4J servera. V rámci aplikácie pre prácu s repozitármi sú použité metódy, ktoré sú popísané v sekcií 2.3.4.

[1] https://tomcat.apache.org/
[2] https://rdf4j.org/

5.2 Implementácia frontendu

Implementácia frontedu začala založením Vue.js projektu pomocou nástroja Vue CLI 3.5.3. V termináli bol zadaný príkaz: `vue create <názov_projekt>` a nasledovala konfigurácia projektu. Medzi defaultnou a manuálnou konfiguráciou bola vybraná manuálna kde boli pridané ďalšie dodatočné knižnice mimo základných knižníc babel a eslint ako:

- **vue-router** - pomáha prepojiť URL/históriu prehliadača a komponenty Vue, čo umožňuje určitým cestám vykresliť akýkoľvek pohľad, ktorý je s ním spojený.
- **typescript** - pre statické typovanie premenných a detekciu chýb pri kompilácií projektu

5.2.1 Node Package Manager

Tento postup manuálnej konfigurácie nemusel byť nutný pretože existuje nástroj príkazového riadka Node Package Manager (NPM), ktorý inštaluje, aktualizuje alebo odinštaluje balíčky Node.js v aplikácii. Je to tiež online úložisko pre balíčky Node.js s open-source zdrojovým kódom. Komunita ľudí na celom svete vytvára užitočné moduly a publikuje ich ako balíčky v tomto úložisku, ktoré je potom možné jednoducho doinštalovať do projektu pomocou príkazu `npm install <package-name>`. Všetky balíčky tretích strán sa nachádzajú v súbore `package.json`, kde je tiež možné pridať balíček s konkrétnou verziou a následne príkaz `npm install` nainštaluje všetky balíčky v tomto súbore.

5.2.2 Typescript

TypeScript vznikol v roku 2012 a bol vyvinutý a udržiavaný firmou Microsoft [11]. Bol publikovaný ako open source. Typescript je rozšírením jazyka JavaScript, ktorý ho rozširuje o statické typovanie, triedy, rozhrania a ďalšie princípy známe z OOP. JavaScript je síce veľmi užitočný jazyk, ktorý umožnil do statického webu pridať dynamické prvky ale nevýhodou JavaScriptu je, že neumožňuje zadávať dátové typy premenných čo sa môže byť na jednu stranu výhoda v niektorých prípadoch ale je to skôr považované za nevýhodu pretože s pribúdajúcou veľkosťou aplikácie je kód ťažšie čitateľný a udržiavaný. Tieto problémy sú vyriešené práve príchodom Typescriptu, ktorý ponúka statické typovanie. Existujú dva typy typovania:

- **statické typovanie** - pri statickom typovaní sa pri deklarácií premennej určí, pre aký typ je premenná určená. Hodnota premennej je tak obmedzená na dátový typ určený pri deklarácií. Výhodou je napríklad známosť typu pri kompilácií a možnosť kontroly kódu, kompilátor zistí, že bola priradená do premennej hodnota s iným dátovým typom ako pri deklarovaní premennej a kód neskompiluje, kým nebude chybá opravená čo je presne princip fungovania Typescriptu. Ďalšiou výhodou je, že je výsledný program rýchlejší a výkonnejší.
- **dynamické typovanie** - dátový typ je pri dynamickom typovaní priradený s hodnotou. Ak uložíme do premennej hodnotu iného dátového typu, zmení sa dátový typ celej premennej. Je tak možné do jednej premennej priradiť viacero hodnôt ako číslo, reťazec atď čo zmenšuje veľkosť výsledného kódu a teda zrýchľuje vývoj. Na druhej strane pri väčších aplikáciach kód začína byť neprehľadný, programy začínajú byť pomalšie a chyby v dátových typoch sú odhalené až pri behu programu.

Typescript patrí medzi typované jazyky takže všetky premenné musia byť deklarované s dátovým typom. Na jednej strane to môže spomaliť vývoj, pretože si nemôžeme priradovať hodnoty do premenných ako chceme ale na druhú stranu typescript pred spustením zkontroluje či všetky dátové typy sedia. Aj keď je Typescript statický typovaným jazykom je možné priradiť premennej viac dátových typov a to pomocou zložených typov, ktoré sa pri deklarácii premennej oddeľujú znakom | ako napríklad `var test = 5 as number | string`. Keďže nová verzia Vue 3 je napísana v Typescripte a plne podporuje Typescript rozhodol som sa ho v aplikácií použiť. [29]

5.2.3 Štruktúra aplikácie

Po vytvorení projektu bola automatický vygenerovaná zložka `node_modules`, ktorá obsahuje základné knižnice aj knižnice tretích strán, ktoré boli pridané do projektu pri vývoji, zložka `public`, ktorá obsahuje súbor `index.html`, ktorý je základnou šablónou pre jednostránkovú aplikáciu (viď 3.2). Ďalej bolí vygenerované ďalšie konfiguračné súbory, ktoré sú potrebné pre spustenie aplikácie. Obsahom zložky src sú zdrojové kody aplikácie rozdelené do podpriečinkov podľa fuknčnosti a sú to:

- **assets** - obsahuje všetky obrázky použité v rámci aplikácie a súbor `global.css` v ktorom sú definované globálne štýly pre jednotlivé komponenty
- **components** - obsahuje globálne komponenty, ktoré sa v aplikácií opakujú ako bočné menu a horný panel
- **router** - obsahuje súbor `router.ts` v ktorom je implementovaná logika smerovania v aplikácií, ktoré bude bližšie popísané v sekcií 5.2.3
- **services** - zložka services obsahuje súbor `APIService.ts`, ktorý slúži na volanie jednotlivých requestov, ktoré boli popísane v sekcii 5.1.1 a súbor `HelperUtils.ts` v ktorom sú implementované pomocné funkcie ako napríklad funkcia pre parsovanie výsledkov dotazu
- **store** - zložka obsahuje súbor `store.ts`, ktorý aplikácia používa na uloženie informácie, ktoré chceme mať dostupné vo všetkých komponentách. Viac v samostatnej časti 5.2.3
- **views** - zložka obsahuje jednotlivé komponenty aplikácie, kde každá komponenta predstavuje vzhľad pre danú stránku aplikácie. Tieto komponenty budú bližšie popísane v časti 3.5.3

Store

Vo Vue ako aj v iných frontendových rámcoch existuje pojem obchod alebo store, ktorý predstavuje centralizované miesto, kde sa môžu ukladať dáta, ktoré sa opakujú naprieč aplikáciou. K týmto údajom majú prístup všetky komponenty aplikácie. Ak by v aplikácií nebol použitý store museli by sa potrebné dáta z rodičovskej komponenty do potomka odovzdávať cez props a naopak by musela každá rodičovská komponenta načúvať udalostiam aby mala prístup k dátam z potomka viz. 3.5.3. To sa môže pri väčšom počte komponet stáť rýchlo neudržateľné a zhroší to tak kvalitu výsledného riešenia. Preto som sa rozhodol v aplikácií použiť Pinia store, ktorý slúži na presne tieto účely.

Pinia je samostatný balíček, ktorý bolo treba do projektu doinštalovať pomocou NPM nástroja. Tento store pomáha pri udržiavaní nasledujúcich údajov v aplikácií, ktorými sú napríklad:

- **selectedRepository** - objekt vybraného repozitára, ktorý bol zvolený pre prácu v rámci aplikácie, názov tohto repozitára je viditeľný v hornej lište každej stránky
- **rdfServerUrl** - url adresa rdf serveru, ktorá je dostupná v hornej lište každej stránky
- **collapsed** - boolean hodnota, ktorá udáva či je stiahnué bočné menu
- **numberOfStatements** - počet trojíc v repozitári
- **numberOfContexts** - počet kontextov v repozitári

Smerovanie

Kedže je Vue jednostránková aplikácia a pri prechádzaní zo stránky na stránku nevyžaduje interakciu so serverom, je treba vykresľovanie daných komponent pre jednotlivé stránky riešiť iným spôsobom a to pomocou knižnice `Vue Router`. Je to oficiálna knižnica pre navigáciu po stránkach a bolo potrebné ju pridať do projektu nástrojom NPM. Jednotlivé cesty k stránkam aplikácie sú definované v súbore `routes.ts` kde pri každej ceste treba zadať nasledujúce parametre:

- **path** - označuje cestu, pri ktorej sa ma komponenta vykresliť
- **name** - označuje názov komponenty, používa sa pri dynamickom smerovaní
- **component** - komponenta, ktorá sa pri presmerovaní na danú cestu vykreslí

Na vytvorenie smerovacích odkazov pomocou Vue router knižnice sa používa namiesto známych `<a>` tagov `<rounter-link>`, ktorému je staticky predananá cesta alebo dynamicky predaný názov komponenty v atribúte `to`, ktorá sa ma vykresliť. Ďalším tagom spojený so smerovaním je `<routner-view>`, ktorý zobrazí komponentu na mieste kde je tento tag umiestnený, podľa URL alebo názvu komponenty, ktoré boli predané `<rounter-link>` tagom. Tento tag je umiestnený v root komponente aplikácie `App.vue` teda jednotlivé komponenty sa vykresľujú do root komponenty aplikácie podľa URL adresy. Všetky cesty aplikácie sú vytvorené v poli routes, ktoré je predané ako parameter výslednému router objektu, ktorý aplikácia používa.

Komponenty

Obsah zložky src tvoria zdrojové kódy pre jednotlivé Vue komponenty, ktoré sú rozdelené do priečinkov podľa stránok aplikácie, taktiež sa v tomto priečinku nachádzajú komponenty `HomePage.vue`, ktorá slúži ako úvodná stránka aplikácie s tematickým obrázkom a krátkym popisom aplikácie a `PageNotFound.vue`, ktorá sa vykreslí v prípade, že nebolá nájdená smerovacia cesta. Aplikácie je rozdelená do štyroch časti `Repositories`, `Explore`, `Update` a `System`, ktoré sú k dispozícií v bočnom menu. Bočné menu a horná lišta sú globálne komponenty, ktoré sú súčasťou každej stránky.

Zložka `Repositories` obsahuje komponentu `RepositoriesPage.vue`, ktorá sa zobrazí po stlačení tlačítka `GET STARTED` na úvodnej stránke. Stránka obsahuje tabuľku s jednotlivými repozitármi dostupnými na RDF servery. Táto tabuľka je implementovaná využitím

Prime Vue knižnice ktorá poskytuje rôzne typy tabuliek, ktorým je možne jednoducho pridať rozšírenú funkcionalitu ako napríklad zoraďovanie záznamov v tabuľke abecedne podľa hodnôt v jednotlivých stĺpcoch alebo zadať maximálnu výšku tabuľky a ak obsah presiahne povolenú výšku je možné použiť v tabuľke scroll bar. Presne tieto vlastnosti som využil pri tvorbe tabuľky repozitárov. Na tejto stránke je potrebné vybrať si repozitár s ktorým sa bude pracovať inak nie je možné prekliknúť na iné stránky okrem možnosti `System` kde sa nepracuje s repozitárom. Po vybratí repozitára sa uloží objekt repozitára s jeho menom, popisom a lokáciou do storu kde bude prístupný naprieč aplikáciou.

Zložka `Explore` obsahuje komponenty `ExlorePage.vue`, `AboutRepositoryPage.vue`, `QueryPage.vue` a `QueryResultPage.vue`. Komponenta `ExlorePage.vue` slúži na prechádzanie samotného obsahu repozitára kde jednotlivé trojice s kontextom sú zobrazené vo forme tabuľky. Získanie a spracovanie dát, ktoré sa zobrazia v tabuľke ne popísane v samotnej sekcií (viď 5.2.4). Táto tabuľka je taktiež implementovaná pomocou Prime Vue komponenty ale na rozdiel od tabuľky repozitárov nie je povolená možnosť zoraďovať jednotlivé stĺpce abecedne a to kvôli tomu, že v prípade veľkých dátových setov by to to bolo neefektívne. Naopak bol tabuľke pridaný paginator, ktorý umožňuje prechádzať trojice po jednotlivých stránkach a tiež je možné nastaviť počet výsledkov, ktoré sa zobrazia v rámci jednej stránky tabuľky. Keďže sa v aplikácií pracuje so zdrojmi je vhodné túto prácu užívateľovi vylepšiť a to tým, že každý zdroj má vedľa seba v tabuľke tlačítko na skopírovanie zdroja a vie tak jednoducho pracovať so zdrojmi bez nutnosti použitia klávesnice.

Komponenta `AboutRepositoryPage.vue` je rozdelená na dve časti, vrchná časť obsahuje dve okná s informáciami o repozitári, ktoré sa vyberú zo storu kde sú po výbere repozitára uchované. Spodná časť stránky je implementovaná pomocou takzvaného tab menu kde sa užívateľ môže prekliknúť medzi položkami Types, Contexts a Namespaces. Po kliknutí na hociktorú z nich sa zobrazí podobná tabuľka ako na stránke s repozitármi kde je možné jednotlivé hodnoty tabuliek zoraďovať abecedne. Položka Namespaces sa trochu líši od ostatných dvoch a okrem tabuľky je tu pridané okno kde je možne jednotlivé prefixy mazať alebo aktualizovať. V okne sa nachádzajú inputy pre text a select box ktoré sú implementované pomocou Prime Vue komponent. V select boxe sú hodnoty prefixov, ktoré sa nachádzajú v tabuľke a po zvolení nejakého prefixu sa predvyplnia hodnoty, ktoré je potom možné jednoducho zmazať alebo pozmeniť.

Komponenta `QueryPage.vue`, je tvorená editorom pre zadávanie SPARQL dotazov, pre ktorý bola využitá knižnica CodeMirror[3] a do projektu bola pridaná nástrojom NPM ako dodatková knižnica. Zadávanie dotazov nie je hlavnou úlohou tejto práce takže tento editor slúži len ako doplnková komponenta kde je možné zadať dotaz. Nepodporuje syntax sparql takže v prípade zle napísaného dotazu neohlási chybu. Druhou časťou stránky je okno s tlačítkami pre prácu s editorom ako tlačítko `EXECUTE`, ktoré po stlačení vykoná zadaný dotaz, tlačítko `CLEAR`, ktoré vyčistí editor a tlačítka na správu query dotazov, ktoré si užívateľ môže uložiť pod nejakým menom a pracovať s nimi neskôr. Ukladanie dotazov je v rámci prehliadača a pre implementáciu je použitý localStorage[4].

Zložka `Update` obsahuje komponentu `UpdateRepositoryPage.`, ktorá je zložená z troch častí Add data, Delete data a editor. Editor je podobne ako v komponente `QueryPage.vue` implementovaný pomocou knižnice CodeMirror. Obmedzením tohto editoru je, že dovoľuje zaslať iba update dotazy. Časti Add data a Delete data sú tvorené rôznymi komponentami Prime Vue ako napríklad input text, text area, dropdown a slúžia na pridávanie alebo

[3] https://codemirror.net/index.html

[4] https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

mazanie dát v rámci repozitára. Po kliknutí na tlačítko `UPLOAD` alebo `REMOVE` sa pomocou `ApiService.ts` zašle dotaz na server so zadanými dátami a vykoná sa akcia.

Posledná zložka `System` obsahuje komponentu `SystemPage.vue`, ktorá umožňuje zmeniť RDF server.

5.2.4 Získanie a spracovanie dát

Pre získanie dát zo serveru sú použité metódy, ktoré ponúka RDF4J REST API (viď 2.3.4). V tejto sekcií sa budem venovať stránke `Explore`, na ktorej uživateľ pracuje s celým obsahom repozitára. Keďže obsah repozitára, môže presahovať milióny trojíc rozhodol som sa použiť stratégiu lazy loading kedy sa načítavajú dáta z úložiska až vtedy kedy to je naozaj potreba. Tento postup vyžaduje použitie modifikátorov výsledku a to konkrétne `LIMIT` a `OFFSET`. Pôvodne bola použitá get metóda na získanie obsahu repozitára z API, táto metóda ale nepodporuje parametre limit a offset takže na miesto nej je zaslaný dotaz na repozitár, ktorý tieto parametre podporuje.

```
SELECT DISTINCT *
WHERE {
  {
    ?subj ?pred ?obj .
  }
  UNION
  { GRAPH ?context { ?subj ?pred ?obj } }
}
```

Výpis 5.1: SPARQL dotaz na získanie celého obsahu repozitára

Tento dotaz vráti všetky trojice v repozitári, ktoré majú kontext alebo nie. Ak je stránka `Explore` navštívená prvýkrát pošle sa tento dotaz s limitom 10 a offsetom 0. S každou ďalšou kliknutou stránkou sa tieto parametre zväčšia o 10 alebo o toľko koľko má užívateľ nastavené v rámci jednej stránky tabuľky. Ak sa zvolí predošlá stránka tabuľky tak sa tieto parametre zmenšia o daný počet.

V tabuľke je tiež možné kliknúť na zdroj, kedy sa zobrazia len trojice či už s kontextom alebo bez, ktoré obsahujú daný zdroj na jednej z pozícií predmet, predikát alebo objekt. V nasledujúcich ukážkach bude hodnota `$variable`, predstavovať kliknutú položku v tabuľke.

```
SELECT DISTINCT *
WHERE {
 { ?subj ?pred ?obj .
   {
      { VALUES ?subj { $variable } ?subj ?pred ?obj }
      UNION
      { VALUES ?pred { $variable } ?subj ?pred ?obj }
      UNION
      { VALUES ?obj { $variable } ?subj ?pred ?obj }
   }
 }
 UNION
 {
```

```
    GRAPH ?context {
     { VALUES ?subj { $variable } ?subj ?pred ?obj }
     UNION
     { VALUES ?pred { $variable } ?subj ?pred ?obj }
     UNION
     { VALUES ?obj { $variable } ?subj ?pred ?obj }
    }
 }
}
```

Výpis 5.2: SPARQL dotaz na získanie všetkých trojíc s danou hodnotou

V prípade ak má zdroj hodnotu literálu je tento dotaz odlišný pretože sa hodnota môže nachádzať len na pozícií objektu ako bolo spomenuté v sekcii 2.2.1.

```
SELECT DISTINCT ?subj ?pred ?obj ?context
WHERE {
  { ?subj ?pred ?obj .
      { VALUES ?obj { $variable } ?subj ?pred ?obj }
  }
  UNION
  {
     GRAPH ?context {
        values ?obj { $variable }
        ?subj ?pred ?obj
     }
  }
}
```

Výpis 5.3: SPARQL dotaz na získanie všetkých trojíc s danou hodnotou literálu

Nakoniec je možné zobraziť všetky trojice v repozitári, ktoré majú určitý kontext a je na to použitý nasledujúci dotaz.

```
select ?subj ?pred ?obj ?context where {
  values ?context { ${variable} }
  GRAPH ?context {?subj ?pred ?obj}
}
```

Výpis 5.4: SPARQL dotaz na získanie všetkých trojíc s danym kontextom

Kapitola 6

Testovanie

V tejto kapitole je predstavené testovanie aplikácie, ktoré prebiehalo dvoma spôsobmi. Prvým je automatizované testovanie aplikácie pomocou End-To-End testovania a druhým manuálne testovanie aplikácie s užívateľmi, ktorým boli zadané dopredu pripravené scenáre. Na automatizované testovanie bol použitý nástroj Cypress, ktorý je popísaný nižšie.

6.1 Automatizované testovanie

Cypress je komplexný testovací nástroj založený na JavaScripte, ktorý sa používa na všetky druhy testovania testovanie. End-to-end testovanie znamená, že sa testuje celková funkcionalita aplikácie narozdiel od jednotkových testov, ktoré testujú len určitú časť aplikácie. Testovanie pomocou Cypress nástroja simuluje správanie užívateľa. Umožňuje klikať na ľubovolné elementy stránky, zvoliť element za základe id alebo triedy, manipulovať s DOM štruktúrou, čítať alebo zapisovať údaje z textových polí, odosielať formuláre, presmerovať na inú stránku a rôzne ďalšie veci. Cypress bol do projektu pridaný ako všetky ostatné externé knižnice pomocou nástroja NPM. Po nainštalovaní sa v projektu vytvorí zložka `tests`, ktorá obsahuje zložku `e2e`. V tejto zložke sa nachádzajú zložky ako napríklad `screenshots`, kde Cypress ukladá obrázky s chybovou hláškou z neúspešného testu, `videos` kde sa vytvoria videá z jednotlivých testov a zložka `specs` v ktorej sa nachádzajú samotne testy aplikácie. Testy je možné spustiť dvoma spôsobmi buď pomocou príkazu `npx cypress run`, ktorý spustí všetky testy v aplikácii alebo pomocou príkazu `npx cypress open`, ktorý otvorí grafické rozhranie tohto nástroja v ktorom je možné spúšťať jednotlivé testy manuálne a zároveň pozorovať ako sa vykonávajú. V aplikácií sa nachádza jedna testovacia sada, ktorá sa skladá z testov:

- **Create new repository** - v tomto teste sa testuje vytvorenie nového repozitára na stránke `Repositories`, kde sa pomocou tlačítka `Create new repository` otvorí modal pre vyplnenie hodnôt, ktoré sa vyplnia a následne sa vytvorí repozitár. Nakoniec sa skontroluje, či je daný repozitár v zozname.
- **Add content to empty repository** - testuje sa pridanie obsahu do repozitára kde sa kontroluje počet trojíc pred pridaní a po pridaní obsahu
- **Delete statment with specific value** - testuje sa zmazanie konkrétnej trojice z obsahu repozitára kde sa kontroluje výsledný počet trojíc po zmazaní
- **Add content to repository and replace data** - testuje sa pridanie a prepísanie obsahu repozitára kde sa následne kontroluje výsledny počet trojíc prepísaného obsahu

- **Select query with all results and check expected results** - v tomto teste sa vykoná dotaz z očakávaným počtom trojíc vo výsledku kde sa následne kontroluje či počet sedí
- **Remove all statements from repository** - v tomto teste sa zmaže celý obsah repozitára a kontroluje sa či je obsah prázdny
- **Delete repository** - nakoniec sa testuje zmazanie celého repozitára a kontroluje sa či počet repozitárov po zmazaní sedí

Táto sada testov funguje ako jeden možný scenár použitia aplikácie a jednotlivé testy prebiehajú za sebou chronologicky. Tieto testy mali otestovať len základnú funkcionalitu jednotlivých operácií a hlavným testovaním bolo testovanie užívateľského rozhrania s reálnymi užívateľmi.

6.2 Testovanie užívateľského rozhrania

Testovanie užívateľského rozhrania prebiehalo s užívateľmi v rôznych vekových kategóriach z IT oblasti aj mimo nej. Keďže použitie aplikácie vyžaduje spustený RDF server, testovanie prebiehalo s každým užívateľom zvlášť cez nástroj Team Viewer kde užívatelia používali môj počítač a ja som ich pri tom sledoval. Užívateľovi bol poslaný súbor `úlohy.txt`, ktorý obsahuje tri scenáre podobné testovacej sade v automatizovaných testoch, ktoré zahŕňajú všetky operácie, ktoré je možné v aplikácii vykonať. Keďže téma tejto práce je celkom špecifická, starší užívatelia mali väčšie problémy navigovať sa v aplikácií čo je pochopiteľné, keďže sa v aplikácií nachádzajú anglické pojmy a pojmy ako query atď. Po nápovede ale boli schopní jednotlivé úlohy dokončiť. Mladším užívateľom išli jednotlivé úlohy jednoduchšie aj keď im bolo niekedy treba mierne pomôcť. Na konci testovania bol každému užívateľovi poslaný dotazník, v ktorom mali hodnotiť obtiažnosť jednotlivých scenárov a nakoniec mohli napísať možné nápady na vylepšenie. Z týchto odpovedí vznikli malé úpravy v aplikácie ako zmena farieb tlačítok, pretože pôvodná farba vyvolávala v užívateľoch pocit, že je tlačítko uzamknuté. Taktiež bol upravený spôsob pridania obsahu do repozitára pomocou súboru kedy bolo treba vybrať súbor kliknutím na tlačítko a následne bolo treba kliknúť na to isté tlačítko aby sa súbor nahral a to robilo problémy užívateľom keďže sa na stránke nachádzalo tlačítko `UPLOAD`.

Kapitola 7

Záver

Cieľom tejto bakalárskej práce bolo vytvoriť webové rozhranie pre prechádzanie úložiska RDF. Pre dosiahnutie tohto cieľa bolo potrebné preštudovať samotný dátový model RDF, dotazovací jazyk SPARQL a nástroje pre tvorbu klientskej časti aplikácie v jazyku JavaScript.

Výsledná aplikácia umožňuje užívateľom rýchlo a jednoducho prechádzať úložisko RDF. Poskytuje užívateľom možnosť vytvárať a mazať repozitáre v rámci úložiska, zobraziť si obsah repozitára vo forme tabuľky kde je možné kliknúť na daný zdroj a následne sa zobrazia len trojice v repozitári, ktoré daný zdroj obsahujú. Ďalej je možné pridať obsah do repozitára v troch rôznych formách a to buď pomocou URL, ktorá obsahuje RDF dáta, súboru, ktorý obsahuje RDF dáta alebo pomocou samotného obsahu, ktorý je v jednom z RDF formátov. Taktiež je možné mazať obsah repozitára a to buď podľa konkrétneho zdroja v trojici alebo celý obsah. Užívateľ ma k dispozícií editor na zadávanie SPARQL dotazov nad repozitárom. Tieto dotazy si môže uložiť na neskoršie použitie a nemusí tak písať jednotlivé dotazy stále ručne. Aplikácia tiež umožňuje pridávať a meniť menné priestory repozitára, stiahnuť celý obsah repozitára a v neposlednom rade je možné zmeniť URL adresu RDF serveru s ktorým aplikácia pracuje.

Pre serverovú časť aplikácie bol použitý RDF4J server, ktorý bol nasadený na Apache Tomcat webový server.

Klientska časť aplikácie bola implementovaná pomocou JavaScriptového frameworku Vue.js s využitím Typescript rozšírenia. Typescript s jeho vlastnosťami ako napríklad statické typovanie zabezpečoval, že bol kód vždy prehľadný vďaka čomu som sa pri implementácii výrazne nezdržoval.

Plusom ktorý by som vyzdvihol je zobrazenie obrovských dátových setov kde sa počet trojíc pohybuje v miliónoch kde riešením je načítavanie dát pomocou takzvaného lazy loadingu, kde sa načíta len počet trojíc, ktorý je viditeľný užívateľovi. Keďže hlavným zameraním aplikácie je prechádzanie RDF úložiska a nie zadávanie SPARQL dotazov, tak editor použitý v aplikácií nepodporuje syntax SPARQL jazyka a teda pri chybnom dotaze alebo preklepe nehlási chybu. Implementácia editora s plnou podporou syntaxe SPARQL jazyka by mohlo byť možným rozšírením aplikácie.

Aj keď práca s RDF dátovým modelom a jazykom SPARQL bola pre mňa úplne niečo nové, keďže som sa s tým ešte nestretol tak mi táto práca priniesla skúsenosti v oblasti vývoja klientskej časti aplikácie a tiež mi dala prehľad znalosti v oblasti RDF, ktoré určite v budúcnosti využijem.

Literatúra

[1] Addlesee, A. *Understanding Linked Data Formats* [online]. Október 2018 [cit. 2022-05-03]. Dostupné z: https://medium.com/wallscope/understanding-linked-data-formats-rdf-xml-vs-turtle-vs-n-triples-eb931dbe9827.

[2] Angular. *Template syntax* [online]. [cit. 2022-04-02]. Dostupné z: https://angular.io/guide/template-syntax.

[3] Apache Jena. *Apache Jena Fuseki* [online]. [cit. 2022-05-03]. Dostupné z: https://jena.apache.org/documentation/fuseki2/index.html.

[4] Christensson, P. *Framework Definition* [online]. 2013 [cit. 2022-04-02]. Dostupné z: https://techterms.com/definition/framework.

[5] contributors, M. *HTML: HyperText Markup Language* [online]. [cit. 2022-04-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML.

[6] Curé, O. a Blin, G. *RDF Database Systems: Triples Storage and SPARQL Query Processing*. 1. vyd. 2014 [cit. 2022-05-03]. 243 s. ISBN 987-0-12-799957-9.

[7] Fluin, S. *Why Developers and Companies Choose Angular* [online]. 2017 [cit. 2022-04-02]. Dostupné z: https://medium.com/angular-japan-user-group/why-developers-and-companies-choose-angular-4c9ba6098e1c.

[8] GN, T. *Getting started with RDF using Blazegraph* [online]. [cit. 2022-05-03]. Dostupné z: https://thejeshgn.com/2020/12/11/getting-started-with-rdf-using-blazegraph/.

[9] Kantor, I. *An Introduction to JavaScript* [online]. 2019 [cit. 2022-04-02]. Dostupné z: https://javascript.info/intro.

[10] Koďousková, B. *CO JE JEDNOSTRÁNKOVÁ WEBOVÁ APLIKACE (SPA) A KDY JI VYUŽÍT?* [online]. 2021 [cit. 2022-04-02]. Dostupné z: https://www.rascasone.com/cs/blog/jednostrankova-webova-aplikace-spa.

[11] Kvapil, J. *Lekce 1 - Úvod do TypeScriptu* [online]. 2018 [cit. 2022-04-20]. Dostupné z: https://www.itnetwork.cz/javascript/typescript/uvod-do-typescriptu.

[12] learnvue. *What are the Vue Lifecycle Hooks* [online]. [cit. 2022-04-10]. Dostupné z: https://learnvue.co/2020/12/how-to-use-lifecycle-hooks-in-vue3/#what-are-the-vue-lifecycle-hooks.

[13] MDN CONTRIBUTORS. *Introduction to the DOM* [online]. [cit. 2022-04-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

[14] MDN CONTRIBUTORS. *JavaScript* [online]. 2022 [cit. 2022-04-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[15] MEINDERTMA, J. *What's the best RDF serialization format?* [online]. Jún 2019 [cit. 2022-05-03]. Dostupné z: https://ontola.io/blog/rdf-serialization-formats/.

[16] META PLATFORMS. *Getting Started* [online]. [cit. 2022-04-02]. Dostupné z: https://reactjs.org/docs/getting-started.html.

[17] META PLATFORMS. *Introducing JSX* [online]. [cit. 2022-04-02]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html.

[18] META PLATFORMS. *React* [online]. [cit. 2022-04-02]. Dostupné z: https://reactjs.org/.

[19] NPM. *Angular* [online]. [cit. 2022-04-02]. Dostupné z: https://www.npmjs.com/package/@angular/cli.

[20] NPM. *React* [online]. [cit. 2022-04-02]. Dostupné z: https://www.npmjs.com/package/react.

[21] NPM. *Vue* [online]. [cit. 2022-04-02]. Dostupné z: https://www.npmjs.com/package/vue.

[22] ONTOTEXT. *What Is the Semantic Web?* [online]. [cit. 2022-04-10]. Dostupné z: https://www.ontotext.com/knowledgehub/fundamentals/what-is-the-semantic-web/.

[23] OPENLINK SOFTWARE DOCUMENTATION TEAM. *What is Virtuoso?* [online]. [cit. 2022-05-03]. Dostupné z: http://vos.openlinksw.com/owiki/wiki/VOS/VOSIntro.

[24] RAIMOND, Y., SCOTT, T., SINCLAIR, P., MILLER, L., BETTS, S. et al. *Case Study: Use of Semantic Web Technologies on the BBC Web Sites* [online]. 2010 [cit. 2022-04-10]. Dostupné z: https://www.w3.org/2001/sw/sweo/public/UseCases/BBC/.

[25] RDF WORKING GROUP. *Resource Description Framework (RDF)* [online]. 2014 [cit. 2022-04-10]. Dostupné z: https://www.w3.org/RDF/.

[26] RDF4J. *The Eclipse RDF4J Framework* [online]. [cit. 2022-05-03]. Dostupné z: https://rdf4j.org/about/.

[27] ROY, S. *Component Communication in Vue.Js* [online]. August 2019 [cit. 2022-04-20]. Dostupné z: https://medium.com/js-dojo/component-communication-in-vue-js-ca8b591d7efa.

[28] SEABORNE, A. a PRUD'HOMMEAUX, E. *SPARQL Query Language for RDF* [online]. Január 2008 [cit. 2022-04-10]. Dostupné z: https://www.w3.org/TR/rdf-sparql-query/.

[29] TYPESCRIPT. *TypeScript for the New Programmer* [online]. [cit. 2022-04-20]. Dostupné z: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html.

[30] VUE.JS. *Comparison with Other Frameworks* [online]. [cit. 2022-04-02]. Dostupné z: https://v2.vuejs.org/v2/guide/comparison.html?redirect=true.

[31] Vue.js. *Component Registration* [online]. [cit. 2022-04-20]. Dostupné z: https://v2.vuejs.org/v2/guide/components-registration.html.

[32] Vue.js. *Components Basics* [online]. [cit. 2022-04-20]. Dostupné z: https://vuejs.org/guide/essentials/component-basics.html.

[33] Vue.js. *Template syntax* [online]. [cit. 2022-04-02]. Dostupné z: https://vuejs.org/guide/essentials/template-syntax.html.

[34] Vue.js. *Vue* [online]. 2006 [cit. 2022-04-10]. Dostupné z: https://cli.vuejs.org/guide/.

[35] W3C. *SparqlEndpoints* [online]. [cit. 2022-04-10]. Dostupné z: https://www.w3.org/wiki/SparqlEndpoints.

[36] W3C. *Semantic Web* [online]. 2015 [cit. 2022-04-02]. Dostupné z: https://www.w3.org/standards/semanticweb/.

[37] Wikipedia. *CSS* [online]. [cit. 2022-04-20]. Dostupné z: https://en.wikipedia.org/wiki/CSS.

[38] Wikipedia. *Semantic Web* [online]. [cit. 2022-04-20]. Dostupné z: https://en.wikipedia.org/wiki/Semantic_Web.

Príloha A

Obsah pamäťového média

- xtomov02.pdf - technická správa práce
- xtomov02.zip - zdrojové kódy
- xtomov02_latex.zip - zdrojové kódy latexu
- testovacie_sady.zip - testovacie súbory a scenáre
- README.md - návod na spustenie aplikácie