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

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. TOMÁŠ KISS**
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 diplomové práce

Student: **Kiss Tomáš, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Informační systémy a databáze
Název: **Webové rozhraní pro správu datových sad RDF**
Web Interface for Managing RDF Data Sets
Kategorie: Databáze
Zadání:

1. Seznamte se s datovým modelem RDF, dotazovacím jazykem SPARQL, existujícími RDF úložišti a jejich aplikačním rozhraním.
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

Při obhajobě semestrální části projektu 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: 18. května 2022
Datum schválení: 11. října 2021

Abstrakt

Cieľom tejto diplomovej práce je poskytnúť základné informácie o problematike dátového modelu RDF. Teoretická časť práce predstavuje okrem možností ukladania RDF dát, aj doteraz existujúce RDF úložiská a sémantický dotazovací jazyk SPARQL. Sú tu stručne uvedené prostriedky využívané na tvorbu klientskych webových aplikácií, s hlavným zameraním na rámec Vue.js. V rámci tejto práce bola na základe získaných poznatkov vytvorená aplikácia, umožňujúca prechádzanie RDF úložísk vo forme webového rozhrania. Výsledné riešenie poskytuje možnosť interaktívneho zadávania dotazov v jazyku SPARQL a manipuláciu s menným priestorom a prefixmi.

Abstract

The main aim of this dimploma thesis is to provide information about RDF data model. Theoretical part of the thesis represents possible ways of storing RDF data, existing RDF storages and SPARQL query language. Furthermore, there is a brief representation of tools used for creation of client web applications with main focus on the Vue.js framework. Acquired knowledge is used to implement web interface enabling browsing of RDF storage and interactive creation of SPARQL queries.

Kľúčové slová

RDF, Resource Description Framework, SPARQL, JavaScript, Vue.js, Klientske webové aplikácie, Turtle, PrimeVue

Keywords

RDF, Resource Description Framework, SPARQL, JavaScript, Vue.js, Client web applications, Turtle, PrimeVue

Citácia

KISS, Tomáš. *Webové rozhraní pro procházení úložiště RDF*. Brno, 2022. Diplomová 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 túto diplomovú prácu som vypracoval samostatne pod vedením pána Doc. Ing. Radeka Burgeta, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Tomáš Kiss
26. apríla 2022

Poďakovanie

Chcel by som sa touto cestou poďakovať vedúcemu práce, Doc. Ing. Radekovi Burgetovi, Ph.D., za vedenie mojej práce a stálu podporu a ochotu poradiť v dôležitých otázkach.

Obsah

1 Úvod **2**

2 Dátový model RDF **4**
2.1 Resource Description Framework 4
2.2 RDF trojice 5
2.3 Menný priestor a prefixy 8
2.4 Serializácia RDF dát 9
2.5 Existujúce RDF úložiská 13
2.6 Dopytovanie nad RDF dátami - SPARQL 15

3 Tvorba klientskych webových aplikácií **19**
3.1 Webové aplikácie 19
3.2 JavaScript 21
3.3 Knižnice a rámce pre tvorbu webových aplikácii 23
3.4 Rámec Vue.js 26

4 Návrh aplikácie **30**
4.1 Prieskum existujúcich riešení 30
4.2 Požiadavky týkajúce sa aplikácie 32
4.3 Vlastnosti navrhnutej aplikácie 33
4.4 Architektúra aplikácie 36

5 Implementácia rozhrania pre prechádzanie RDF úložiska **38**
5.1 Použité technológie 38
5.2 Implementácia navrhnutých funkcionalít 40
5.3 Možné rozšírenia aplikácie 47

6 Testovanie aplikácie **48**
6.1 Zdroje testovacích dát 48
6.2 Spôsoby testovania 48

7 Záver **50**

Literatúra **51**

A Obsah priloženého pamäťového média **54**
A.1 Diplomova praca - Text 54
A.2 Diplomova praca - Zdrojove subory 54

Kapitola 1

Úvod

Vďaka veľkému rozšíreniu internetu vo svete počas uplynulých rokov a jeho relatívne jednoduchej dostupnosti, vzrástol vo veľkej miere i počet webových stránok. Tento rast dodnes nepretržite trvá a každým dňom ich množstvo rapídne rastie ďalej. Tieto stránky obsahujú v sebe užitočné a relevantne dôležité informácie, ktorých nájdenie a získanie sa stáva časom čoraz ťažším a zložitejším. Na tento nepriaznivý fakt upozornili odborníci už pred niekoľkými rokmi a navrhli vytvorenie sémantického webu. Jedná sa o web založený na štandardizovanom popise webových zdrojov, umožňujúci ich jednoduchšie strojové spracovanie. Jedným z jeho stavebných kameňov je aj rámec Resource Description Framework (skratkou *RDF*).

Hlavným cieľom tejto práce je vytvorenie aplikácie umožňujúcej správu RDF dát vo zvolených úložiskách. Tento cieľ je doplnený o vedľajšie ciele, ako je predstavenie základnej problematiky dátového modelu RDF, ktorý sa zameriava na dáta zdieľané navonok. Následne sú popísane jeho elementárne prvky v podobe RDF trojíc, pozostávajúcich z časti subjekt, predikát a objekt. Ich spojenie tvorí RDF grafy, ktoré predstavujú sémantické siete. Okrem toho sa v práci preberajú základne spôsoby prenosu týchto dát, ako aj ich serializácia do súborov a rôzne spôsoby uchovania v implementovaných RDF úložiskách. Demonštrované sú základné operácie zahrňujúce dopytovanie nad úložiskami RDF trojíc alebo ich modifikácia s využitím jazyka SPARQL.

Program pre správu RDF dát bol vyvíjaný v podobe webového rozhrania. V posledných rokoch narástol dopyt o tento typ aplikácií a firmy vynakladajú veľké úsilie na tvorbu rôznych nástrojov, ktoré podporujú a zľahčujú ich vývoj. V rámci práce boli preskúmané a popísané niektoré, z týchto implementačných technológií, ktoré by sa dali použiť pri vytváraní tohto programu.

Táto práca je rozdelená na päť hlavných kapitol. V kapitole 2 je popísaný základný princíp modelu RDF, jeho vlastnosti a oblasti jeho možného využitia. Pre manipuláciu s dátami sú uvedené samotné dopytovacie operácie jazyka SPARQL.

Kapitola 3 sa zaoberá nástrojmi a knižnicami, ktoré sa používajú na tvorbu klientskych webových aplikácií. Z preskúmaných možností sa najviac zameralo na popis rámca Vue.js, ktorý v dnešnej dobe patrí medzi jednu z najobľúbenejších technológií, používaných v tejto oblasti.

Postup návrhu webového rozhrania je popísaný v kapitole 4 a jej implementácia v kapitole 5. Počas tejto činnosti sa vychádzalo z naštudovaných informácií, popísaných v teoretickej časti práce a z preskúmania už existujúcich riešení v tejto oblasti. Testovanie vytvoreného rozhrania bolo prevedené na voľne dostupných dátových súboroch. Postup testovania a jeho výsledky sú uvedené v kapitole 6.

Posledná kapitola 7 obsahuje záverečné zhrnutie celej práce, dosiahnuté ciele a prípadné možnosti na rozšírenie aplikácie v budúcnosti.

Kapitola 2

Dátový model RDF

Sémantický web umožňuje strojom, aby porozumeli významu informácií na World Wide Web (skratkou *WWW*) a jeho cieľom je zjednodušiť ich automatizované strojové spracovanie. Tento web je postavený na základoch, tvorených World Wide Web Consortium[1] (skratkou *W3C*) štandardmi. Medzi ne patria RDF dátový model, RDF schéma a dopytovací jazyk SPARQL. Táto časť práce je venovaná k oboznámeniu sa s týmito štandardmi, stručnému popisu teórie a konceptov, stojacich za týmito pojmami.

Sekcia 2.1 popisuje dátový model RDF, ktorý je známy pre možnosť vyjadriť fakty pomocou trojíc (anglicky *triples*), ktorých štruktúra a základné prvky sú podrobne popísané v sekcii 2.2. Ich zápis je možné previesť RDF gramatikou a syntaxou, predstavenou v sekcii 2.3.

Na rozdiel od niektorých iných dátových modelov, nie je RDF viazaný k jedinému formátu serializácie. Trojice je možné serializovať mnohými spôsobmi, čo ponecháva vývojárom voľnú ruku pri výbere najvhodnejšieho variantu pre ich aplikáciu, uvedené v sekcii 2.4.

Informácie reprezentované pomocou trojíc sa vyznačujú veľkou znovu použiteľnosťou a preto je potrebné pre ich uchovávanie vybrať správny typ úložiska. Zoznam najpoužívanejších databáz uvádza sekcia 2.5.

Potom, čo informácie boli uchované vo zvolenom úložisku, je možné ich analyzovať a prehľadávať pomocou špeciálneho dopytovacieho jazyka SPARQL, predstaveného v sekcii 2.6.

2.1 Resource Description Framework

World Wide Web (slovensky *celosvetová sieť*, skratkou WWW) alebo s kratším označením iba ako Web, bol vytvorený primárne pre používanie ľuďmi. Napriek tomu, že jeho obsah je plne strojovo čitateľný, dáta ktoré obsahuje, nie sú zrozumiteľné pre stroje. Tento fakt má nepriaznivý vplyv na automatizáciu spracovania informácií, ktorých množstvo presahuje limity manuálneho spracovania. Riešenie tohto problému predstavuje používanie metadát – dáta o dátach, spolu s RDF, ktoré umožňuje ich spracovanie.

Resource Description Framework (skratkou *RDF*, v texte sa bude používať hlavne označenie skratkou) je štandardom zavedeným konzorciom W3C, predstavujúci základný spôsob pre vyjadrenie grafu dát. Hlavným cieľom je umožniť ich zdieľanie medzi strojmi, zabezpečiť spoluprácu zariadení, ktoré si vymieňajú strojovo zrozumiteľné dáta cez web. Vďaka

[1] World Wide Web Consortium je konzorcium produkujúce slobodné štandardy pre World Wide Web.

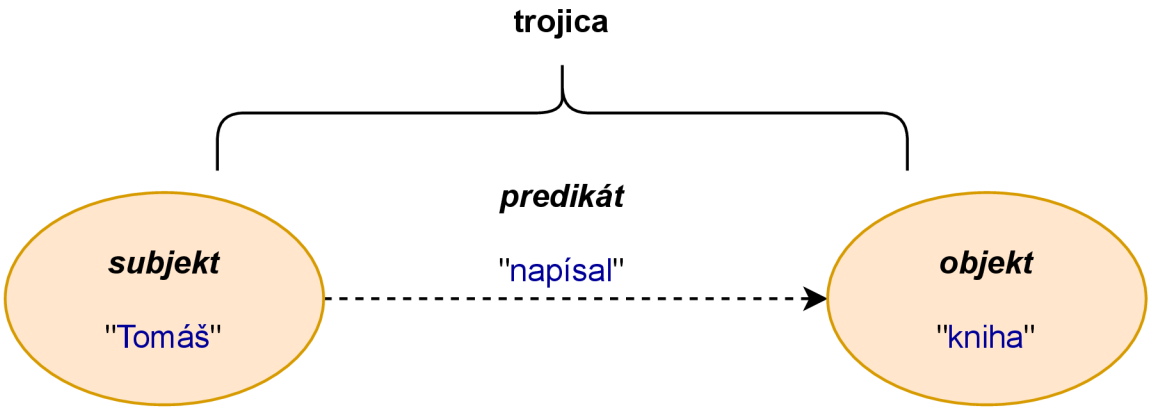
pôvodu štandardu sa v uplynulých rokoch vybudovala v jeho priamom okolí veľká kolekcia nástrojov a služieb.

Základ RDF tvorí model reprezentujúci pomenované vlastnosti a ich hodnoty v prehľadnej podobe. Zabezpečuje rozklad znalostí pomocou flexibilnej metódy na menšie prislúchajúce časti. Tieto časti sú spoločne nazvané ako trojice (anglicky *triples*) a tvoria základné atómy modelu. Existujú presne stanovené pravidlá, určujúce vzťah medzi elementárnymi prvkami trojíc.

Táto sekcia vychádzala z úvodu zdroja [14].

2.2 RDF trojice

RDF dátový model vyjadruje informácie v podobe trojdielnych faktov pomenovaných názvom *trojice* (anglicky *triples*). Každá z týchto trojíc sa podobá do určitej miery na krátke vety, ktoré vyjadrujú obecné fakty. Vety sú tvorené spojením troch častí, známe aj z gramatiky slovenského jazyka: subjekt, predikát a objekt. Príkladom môže byť tvrdenie: „Tomáš napísal knihu.“, kde subjektom je *Tomáš*, predikátom je *napísať* a hodnota vlastnosti je *kniha*.



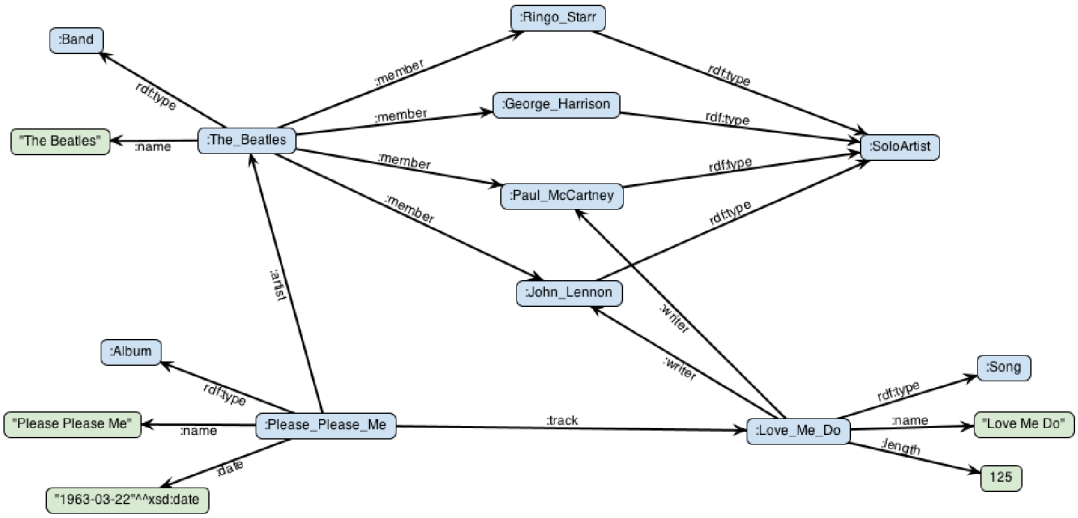
Obr. 2.1: Trojica pozostáva z častí: subjekt, predikát a objekt.

Subjekt predstavuje určitý zdroj, o ktorom je prednesené nejaké tvrdenie. Je reprezentovaný pomocou jednotného identifikátora zdroja (anglicky *Uniform Resource Identifier*, skratkou URI, ďalej sa bude používať hlavne skratka). Takýmto zdrojom môže byť z reálneho sveta napríklad konkrétna osoba, kniha, webová stránka, album. Vlastnosť, určitý aspekt subjektu, sa znázorňuje pomocou predikátu so špecifickým významom. Využíva sa na určenie vzťahu medzi subjektom a objektom v rámci tvrdenia. Konkrétna hodnota tejto vlastnosti je označená už ako objekt. Táto hodnota môže nadobudnúť buď podobu literálu alebo URI.

RDF Graf

Súbor RDF dát je známy ako graf. Pod pojmom *graf* sa v tomto prípade myslí na matematickú štruktúru tvorenú uzlami a hranami. Pri grafickom znázornení nadobúda tvar orientovaného grafu, ako je to znázornené aj na obrázku 2.2. Možnosť zobrazenia súboru trojíc v podobe grafu umožňuje fakt, na základe ktorého objekt jedného tvrdenia môže sa stáť subjektom ďalšieho. Umožňuje to prekrývanie, previazanie trojíc medzi sebou.

Vizualizovať tento graf sa dá pomocou uzlov, tvoriacich subjekty a objekty zo súboru trojíc, prepojených orientovanými hranami. Uzly majú tvar elipsy, v ktorých je zapísané ich meno. Mená takmer všetkých uzlov sú tvorené z URI, označujú, že sa jedná o zdroje, alebo v prípade literálov z reťazcov, iných typovaných hodnôt. Orientované hrany sú znázornené pomocou hrán vedúcich medzi elipsami. Nad hranami sa uvádzajú predikáty.



Obr. 2.2: RDF graf vytvorený spojením viacerých trojíc. U subjektov (modré elipsy) sú namiesto úplných URI používané ich skrátene verzie s prefixmi. Prevzaté z [27].

Jednotný identifikátor zdrojov

V dátovej štruktúre ako je graf je dôležité, aby sme ku každému uzlu priradili unikátny identifikátor. Zabezpečuje to konzistentné odkazovanie sa na uzly cez všetky trojice, ktoré popisujú vzťahy medzi nimi [26]. Je to podobné k obmedzeniam v relačnom dátovom modeli, kde k spojeniu dvoch tabuliek je potrebné nájsť spoločný kľúč, zdieľaný identifikátor riadku.

Jednou možnosťou, ako vytvárať identifikátory, by bolo označovanie uzlov priraďovaním reťazcov spolu so starostlivou kontrolou akéhokoľvek druhu prekrývania sa. V prípade menších aplikácií, by tento prístup aj stačil, ale keď je potrebné koordinovať už viacej súborov dát, znamenalo by to neznesiteľnú záťaž pre odborníkov.

Aby sa predišlo nejasnostiam týkajúcich sa označovania uzlov, rozhodlo sa pri vytváraní štandardu RDF, brať všetko vo svete ako zdroj a označovať ich pomocou URI.

Uniform Resource Identifier (slovensky *jednotný identifikátor zdroja*, skratkou URI) je používaný na identifikáciu alebo pomenovanie zdroja v podobe kompaktného reťazca znakov [26]. Reťazec pozostáva z nasledujúcich častí: názov schémy nasledovaný dvojbodkou, dve lomítka a identifikátor, špecifický pre danú schému označujúci, zdroj. Medzi najčastejšie používané schémy patria *http* a *https*.

Práve tieto schémy často vedú ľudí do omylu, že akýkoľvek reťazec začínajúci s niektorým z týchto dvoch slov, je adresa webovej stránky, ktorú sú schopní navštíviť pomocou webového prehliadača. Napríklad Friend of a Friend (skratkou *FOAF*) je slovník pomenovaných vlastností pre označenie konceptu človeka, používa URI „http://xmlns.com/foaf/0.1/Person“, ale po navštívení adresy nastane iba presmerovanie na domovskú stránku špecifikácie [9].

URI bol navrhnutý tak, aby zastrešoval aj adresy Uniform Resource Locator (slovensky *jednotný vyhľadávač zdrojov*, skratkou URL), používané na označenie zdrojov na internete. URL definuje doménovú adresu servera, umiestnenie zdroja na serveri a protokol, ktorým je možné pristupovať k zdroju. Napriek tomu je potrebné si uvedomiť, že nie každý URI je URL, pričom opačne tvrdenie platí. Prakticky to znamená, že nedá sa predpokladať, že identifikátory URI budú produkovať akékoľvek informácie pri zadaní do prehliadača. Sprístupnenie digitálnych informácií o zdrojoch, dostupných na týchto adresách, sa považuje za nie nutne povinný, radšej iba za osvedčený postup [26].

Jedinou, dá sa povedať nevýhodou URI bolo to, že znaky použiteľné v identifikátoroch boli obmedzené na podmnožinu znakovej sady US-ASCII. Riešenie tohto problému znamenalo zavedenie štandardu Internationalized Resource Identifier (slovensky *internacionalizovaný identifikátor zdroja*, skratkou IRI). Internacionalizované identifikátory umožňujú použiť širší rozsah znakov, napríklad z čínskej alebo cyrilskej abecedy, a tak vytvárať názvy zdrojov v rôznych jazykoch.

Literál

V prípade trojíc v RDF, subjekt a predikát môžu byť iba vo forme URI a objekt môže mať podobu URI alebo literálu. Objekt v skutočnosti môže byť viac než jednoduchý reťazec, máme možnosť k nemu priradiť špecifický dátový typ alebo jazykovú značku, identifikujúcu v akom jazyku je text [9].

Literály sa používajú pre reťazce, dátumy a čísla. V RDF grafoch pozostávajú z dvoch alebo troch časti [8]:

- lexikálna forma - jedná sa o UNICODE reťazec.
- IRI dátového typu - identifikácia dátového typu, napr. integer, boolean, string. Môže byť uvedený aj v podobe celého URI alebo prefixu. Popisuje mapovanie lexikálnej formy na konkrétnu hodnotu.
- jazyková značka - dvojpísmenové kódy špecifikujúce jazyk pre lexikálnu formu a oddeľujú sa pomocou zavináča od reťazca, napríklad „Director“@en a „Directeur“@fr - *en* pre angličtinu a *fr* pre francúzštinu.

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix d:   <http://learningsparql.com/ns/data#> .
@prefix dm:  <http://learningsparql.com/ns/demo#> .

d:item342 dm:shipped     "2011-02-14"^^<http://www.w3.org/2001/XMLSchema#date> .
d:item342 dm:quantity    "4"^^xsd:integer .
d:item342 dm:invoiced    "false"^^xsd:boolean .
d:item342 dm:costPerItem "3.50"^^xsd:decimal .
```

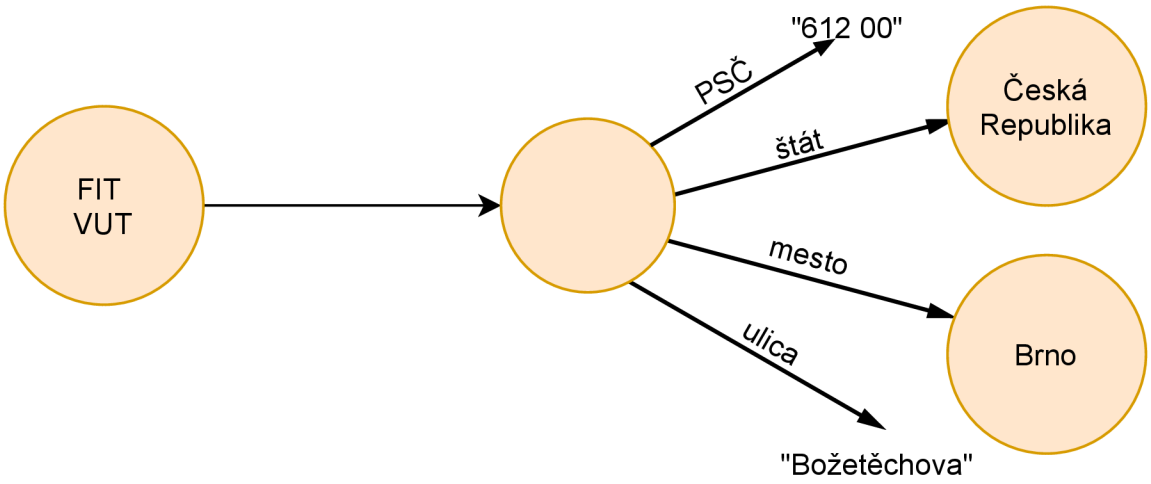
Obr. 2.3: Príklad rôznych literálov s uvedeným dátovým typom. Prevzaté z [9].

Serializujú sa, ako ich lexikálna hodnota, v dvojitých úvodzovkách, za ktorými nasleduje dátový typ, oddelený pomocou dvoch striešok. Napríklad takto: "1"^^xs:integer.

Prázdne uzly

V RDF grafoch takmer všetky uzly majú pridelené meno. Ich pomenovanie, URI alebo reťazec, závisí od toho či, reprezentujú zdroj alebo literál.

RDF ponúka možnosť, aby zdroje boli anonymné. Je to vhodné pre prípady, keď nastane situácia, že nepoznáme URI toho daného zdroja, na ktorý chceme odkazovať alebo chceme zoskupiť určité znalosti [5]. Príkladom môže slúžiť popis určitej budovy, kde je dobré zoskupiť o ňom informácie ako krajina, mesto, ulica, PSČ a modelovať ich pomocou prázdneho uzla.



Obr. 2.4: Znázornenie použitia prázdneho uzla pri modelovaní adresy budovy Fakulty informačních technologií Vysokého učení technického v Brně.

Takéto zdroje reprezentujeme pomocou prázdnych uzlov (anglicky *blank nodes* alebo *bnodes*) bez identifikátora. Nemajú trvalú identitu. Prázdne uzly majú špeciálny priestor mien, označovaný pomocou znaku podčiarkovník (_). Pri zobrazení trojíc prázdne uzly majú identifikátor v podobe *_:id*, kde *id* časť je iba lokálny dočasný identifikátor pre daný graf a nemôže byť použitý pri spojení viacerých grafov. Ak takéto dáta sú kopírované ďalšou aplikáciou, toto označenie nemusí byť zachované, podstatné je, aby všetky spojenia smerujúce do a z uzla boli prenesené.

2.3 Menný priestor a prefixy

RDF, v snahe vyhnúť sa zámene nezávislých ale konfliktných definícií rovnakého pojmu, zaviedol používanie menného priestoru (anglicky *namespace*) pochádzajúceho z XML [8]. Menný priestor slúži na spojenie konkrétneho použitia slova v určitom kontexte so slovníkom, v ktorom je uvedená definícia slova pre daný kontext, význam slova v danom kontexte.

Subjekty a predikáty v každej trojici musia patriť ku konkrétnemu mennému priestoru za účelom zabránenia zmätku medzi podobnými menami, v prípade, ak sa v budúcnosti tieto dáta skombinujú s inými dátami. Preto sú reprezentované vždy pomocou URI. Prefixy, iným slovom predpony, pomáhajú predísť opätovnému výpisu dlhých URI identifikátorov menných priestorov v textoch. Príklad definície prefixov v jazyku TURTLE je uvedený v ukážke 2.1.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>
```

Výpis 2.1: Príklad definície prefixov v jazyku Turtle. Na každom riadku je uvedený prefix a za ním menný priestor, ktorý reprezentuje.

Takmer na ktoromkoľvek mieste v RDF a SPARQL, kde je potrebné používať URI, je možne využiť namiesto toho názov s prefixom, pokiaľ bol jeho prefix správne deklarovaný [9]. Časť názvu za dvojbodkou sa označuje ako lokálny názov. Ako príklad môže slúžiť *dc:titul*, kde lokálny názvom je *titul*. Používanie predpôn patrí medzi dobré praktiky, ale nie je povinné. Pomáhajú vo vytváraní kompaktnejších dopytov, príkazov, ktoré sa tak stavajú aj čitateľnejšie, ako tie, ktoré obsahujú iba celé URI. Je potrebné si zapamätať, že prefixy neidentifikujú menné priestory, iba ich zastupujú a URI sú tie, ktoré ich identifikujú.

V roly prefixu sa môže vyskytovať aj jednoduchá dvojbodka. V praxi sa bežne používa pre také menné priestory, ktoré sa často vyskytujú v danom dokumente. Pomáha to zvýšiť prehľadnosť textu a zjednodušuje čítanie dokumentu pre ľudí.

```
PREFIX : <http://www.w3.org/2000/01/rdf-schema#>
PREFIX an: <http://example.org/animals/>
SELECT ?subclass
WHERE {
     ?subclass :subClassOf an:Animal
}
```

Výpis 2.2: Príklad definície prefixov v podobe jednoduchej dvojbodky v jazyku SPARQL a jeho využitie v dopyte. Dopyt získa všetky podtriedy triedy Animal.

RDF štandard poskytuje menný priestor *<http://www.w3.org/1999/02/22-rdf-syntax-ns#>*, ktorý je často používaný práve pomocou predpony. Medzi štandardnými identifikátormi, uvedenými v tomto priestore mien, patrí aj predikát *rdf:type*, vďaka ktorému je možné previesť priraďovanie typov v RDF [5]. Objekty v trojiciach, v ktorých sa tento predikát vyskytuje, sú chápané ako typy. Zdroje je možné rozdeliť do tried a ich členov označovať ako inštancie triedy. V takomto prípade *rdf:type* vyjadruje to, že subjekt je inštanciou triedy reprezentovanou objektom [7].

```
PREFIX : <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?album
WHERE {
    ?album rdf:type :Album .
}
```

Výpis 2.3: Príklad s použitím prefixu *rdf:type* v podobe dopytu v jazyku SPARQL. Dopyt získa všetky trojice, ktoré ako predikát majú rdf:type a :Album ako objekt.

2.4 Serializácia RDF dát

Odborný termín, označujúci uloženie RDF dát v podobe reťazca bajtov na disk, je *serializácia*. RDF na rozdiel od iných dátových modelov nie je úzko viazaný iba s jediným formátom serializácie. Jeho atómy môžu byť serializované mnohými spôsobmi, pritom zanechávajú voľnú ruku vývojárom, aby si zvolili pre nich najvhodnejšiu formu [16].

V nasledujúcich podsekciách bude prezentovaných päť najpoužívanejších foriem serializácie:

- N-Triples - najjednoduchší zo zápisov
- N3 - kompaktnejšia verzia formátu N-Triples
- Turtle - podobné k N3 iba bez „syntaktického cukru“
- RDF/XML - najčastejšie používaný formát
- RDFa - vložiteľný do iných foriem serializácie, ako je napríklad XHTML

Pri popise formátov sa vychádzalo z článkov [3] [4] [10] [12] [16] a z kníh [5] [22] [26].

N-Triples

N-Triples je najjednoduchšou formou textovej reprezentácie RDF dát. Prináša iba jedinú nevýhodu, čo je nemožnosť používania prefixov, namiesto celých URI, a iných funkcií, ktoré sú známe z ďalej popísaných formátov. Vedie to k zníženiu prehľadnosti textu, spôsobuje problémy pri tlači takýchto dokumentov [5]. Dlhé URI vedú k potrebe používať určitý typ kompresie pre redukovanie využitého úložného priestoru [16]. Parsovanie a serializácia tohto formátu je veľmi jednoduchá a účinná. Existuje mnoho knižníc, podporujúcich tieto činnosti.

Každá z trojíc je uvedená v samostatnom riadku, ukončená bodkou (.). Okrem prázdnych uzlov a literálov, subjekt, predikát a objekt sú reprezentované ako tri kompletné URI, oddelené od seba medzerami. Uzatvárajú sa pomocou lomených zátvoriek < a >. Súbor môže obsahovať aj komentáre, začínajúce znakom #, na samostatných riadkoch. Prípona názvu, používaná pre tento formát, je *.nt*.

```
<http://example.org/#spiderman> <http://example.org/enemyOf>
<http://example.org/#green-goblin> .
```

Výpis 2.4: Príklad serializácie trojíc v N-Triples. Trojica by mala byť uvedená na jedinom riadku, ale z dôvodu prehľadnosti bola rozdelená na dva riadky.

N3

Notation 3, alebo skrátene N3, bol vytvorený ako kompromis medzi jednoduchosťou N-Triples a výraznosťou RDF/XML [5]. Jeho zápis sa veľmi podobá zápisu v N-Triples.

Pre zníženie počtu opakovaní dlhých URI v texte, čo je možné spozorovať u N-Triples súborov, využíva vlastnosti RDF/XML pre skrátenie týchto URI pomocou prefixov. Predpony musia byť deklarované na začiatku dokumentu, pred ich prvým použitím v texte. Deklarácia má tiež podobu trojice, ukončenej bodkou.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

<http://example.org/#spiderman> rdf:type foaf:Person .
```

Výpis 2.5: Príklad serializácie trojíc v N3 s využitím prefixov.

Každý uzol v RDF grafe je potencionálnym subjektom, o ktorom pomocou ďalších trojíc môžeme vyjadriť nejaké fakty, čo vedie k mnohonásobnému opakovaniu jeho mena vo výstupe. N3 ponúka riešenie tohto problému pomocou možnosti kombinovať viacero faktov o jednom subjekte pomocou bodkočiarky (;). Bodkočiarka sa umiestňuje za prvým faktom a ďalej sa už iba uvádzajú predikáty a objekty ďalších faktov [26].

N3 zaviedol špeciálne značenie pre skrátenie *rdf:type*, ktorý bol predstavený v sekcii 2.3, v podobe jediného písmena *a*.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

<http://example.org/#spiderman> a foaf:Person;
    rdf:type foaf:Man; foaf:name "Peter Parker" .
```

Výpis 2.6: Príklad serializácie trojíc v N3 s využitím prefixov. V tomto príklade je možné vidieť využitie *a* namiesto rdf:type a bodkočiarky pre uvedenie viacerých faktov o jednom subjekte.

Ďalšími zaujímavými vlastnosťami N3 sú: možnosť odkazovať sa na celý graf trojíc, ako na jediný zdroj, a to v rámci samotného grafu, ale aj mimo neho. Vytváranie pravidiel, ktoré umožnia odvodiť nové trojice na základe pravdivostných podmienok v existujúcom súbore trojíc [9]. Vďaka týmto vylepšeniam sa dá Notation 3 veľmi ľahko čítať a pomáha tento formát k lepšiemu pochopeniu podstaty RDF trojíc. Avšak z druhej strany, prinášajú tieto možnosti aj relatívne nákladnú serializáciu dát a ťažšie parsovanie týchto súborov [16].

N3 je nadmnožinou N-Triples, preto knižnice umožňujúce parsovanie tohto formátu budú podporovať aj spracovanie N-Triples súborov. Prípona názvu súboru, používaná pre tento formát, je *.n3*.

Turtle

Terse RDF Triple Language, skrátene Turtle, je zjednodušená verzia formátu N3. Jeho populárnosť v posledných rokoch veľmi rastie a je doporučený aj W3C [5]. Je jednoducho čitateľný pre človeka a je dobrým kandidátom pre použitie, ak je potrebné ručné spracovanie a editácia RDF dát.

Turtle podporuje reprezentáciu prázdnych uzlov v podobe *_:id*, kde *id* časť je identifikátorom daného prázdneho uzla. Druhou podporovanou možnosťou je uzatvorenie všetkých dvojíc pozostávajúcich z predikátu a objektu medzi hranaté zátvorky [a], pre ktoré sa má vygenerovať automaticky jeden spoločný prázdny uzol v úlohe subjektu [4].

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:alice foaf:knows _:bob .
_:bob foaf:knows _:alice .

# Someone knows someone else, who has the name "Bob".
[] foaf:knows [ foaf:name "Bob" ] .
```

Výpis 2.7: Príklad serializácie trojíc v N3 s využitím prefixov. V tomto príklade je možné vidieť využitie *_:id* a [], označujúce prázdny uzol v trojiciach .

Ďalšou novinkou, čo Turtle podporuje, je vytváranie kolekcií v podobe zoznamov obsahujúcich sekvenciu prvkov, uzavretých zátvorkami (a) [4]. Prvky sú oddeľované pomocou medzier. Tieto zoznamy sa môžu objavovať v úlohe subjektu alebo objektu.

```
@prefix friend: <http://example.com/friend/> .

friend:Tomas friend:isFriendOf (friend:Adam friend:Robert).
```

Výpis 2.8: Príklad využitia zoznamov prvkov v trojiciach namiesto prvku objekt.

Tento formát odstránil časť syntaktického cukru a vlastností, ktoré boli reprezentované v N3, vďaka čomu sa stal jednoduchšie spracovateľným, parsovateľným. Čo sa týka jeho podpory zo strany knižníc, tak tých, ktorý umožňujú pracovať s ním, je veľký počet a to kvôli jeho veľkej obľube medzi vývojármi.

Prípona názvu súboru, používaná pre tento formát, je *.ttl*.

RDF/XML

RDF/XML formát bol definovaný W3C pre vyjadrenie RDF grafu v podobe XML dokumentu. Vďaka tomu, že RDF a RDF/XML boli predstavené naraz konzorciom W3C, boli často zavádzajúcim spôsobom označované spoločne iba ako RDF.

RDF/XML bol prvým aj najznámejším formátom pre serializáciu RDF dát. V rokoch, keď RDF bol predstavený verejnosti, mnoho systémov bolo schopných spracovať a ukladať XML dáta. Preto sa zo začiatku javilo zavedenie RDF/XML ako logický krok [16]. Časom sa ukázalo, že to nebola správna myšlienka, RDF/XML nebol obľúbený medzi vývojármi. Dôvodom mohlo byť aj to, že je zvláštnou kombináciou dvoch fundamentálne odlišných konceptov, ako je stromová štruktúra a graf, založený na trojiciach. Oproti ostatným štandardom, na vyjadrenie rovnakých informácií, používa oveľa viac slov a textu. Pre vývojára, zaoberajúceho sa XML, môže byť tento formát známy, ale môže vyvolať aj zmätok, lebo neodráža jasne model trojíc [16].

RDF/XML nie je ničím iným, ako dobre formátovaným XML. Bol obohatený o nové dodatočné obmedzenia, ktoré umožňujú jednoduchšiu výmenu, zber a zlúčenie údajov z viacerých modelov [22]. Jeho parsovanie je možné previesť aj iba XML technológiami.

Používanie tejto serializácie je doporučované hlavne v takom prípade, keď je potrebné priamo pracovať s XML formátom.

RDFa

Resource Description Framework in Attributes (slovensky *systém opisu zdrojov v atribútoch*, skratkou RDFa) je technológia pre prenos štruktúrovaných informácií vnútri webových stránok, nie je čistou serializačnou formou pre RDF [12]. Je to spôsob anotácie súborov typu HTML, XHTML. Pridaním atribútov k prvkom týchto súborov je možné im dať sémantický kontext v rámci webových stránok. Umožňuje vložiť subjekt, predikát a objekt do takých súborov, ktoré neboli navrhnuté na umiestnenie RDF dát v nich.

Definuje niekoľko nových, aj už existujúcich HTML atribútov, ako miesta vhodné pre ukladanie alebo identifikáciu prvkov trojíc, uložených do pôvodného obsahu súborov [9]. Štandardne vo všetkých trojiciach, uvedených na stránke, ako URI subjektu, sa berie bázová URI stránka.

```
<p about="http://example.com/spiderman"
property="http://example.com/enemyOf">Green Goblin</p>
```

Výpis 2.9: Príklad umiestnenia RDF trojíc v rámci HTML stránky. V príklade je možné vidieť ako sa dá využiť napríklad jednoduchý paragraf ( ) pre uloženie RDF dát. Atribút *about* označuje subjekt, *property* predikát trojice. Za objekt sa berie text uvedený medzi dvoma značkami paragrafu.

RDFa je vynikajúci pre tvorbu metadát o obsahu dokumentov. Existuje mnoho pomocných programov, schopných vytiahnuť tieto dáta z RDFa atribútov v takom formáte, ktorý je vhodný aj pre spracovanie pomocou jazyka SPARQL. Tieto atribúty parsuje aj Google, pre vylepšenie svojich výsledkov vyhľadávania.

Myšlienka celého RDFa je to, že stačí publikovať obsah súboru iba jediný krát, a to takým spôsobom, že sú v ňom už uvedené dáta čitateľné pre človeka aj pre stroje naraz.

Kombinácia RDF dát so zobrazovacími dátami spôsobuje nárast pamäťových nárokov na uloženie týchto súborov a komplikuje ich štruktúru. Preto parsovanie tohto typu serializácie je oveľa zložitejšie, než v prípade formátov primárne špecializovaných na RDF, ako je napríklad N-Triples.

RDFa môže byť užitočný v takých prípadoch, keď je cieľom zvýšiť sémantiku už existujúcich webových stránok, blogov, aplikácií založených na HTML.

2.5 Existujúce RDF úložiská

Pre uchovanie veľkého množstva trojíc nie je správnym riešením používať formát súborov napríklad Turtle alebo RDF/XML. Oveľa lepšou voľbou je použiť dedikovaný databázový systém, ktorý sa vie postarať o indexovanie dát, rozhodovať sa o tom, ktoré trojice majú byť načítané do pamäte [9].

Na tento účel sa dajú využiť aj relačné databázové systémy, ako sú napríklad Oracle, MySQL, PostgreSQL, ktoré ale neboli optimalizované pre uchovávanie trojíc.

Správcovia databázy sa pre tento účel označujú ako úložiská trojíc (anglicky *triplesores*). Boli vytvorené za účelom uchovávania a následného získavania RDF dát prostredníctvom sémantických dopytov. Predstavujú určitý typ grafových databáz, ktoré uchovávajú dáta v podobe sieti objektov a sú schopné vydedukovať z existujúcich vzťahov zatiaľ neodhalené nové informácie [19]. Sú dostatočne flexibilné a dynamické, umožňujú prepojenie rôznych údajov. Prinášajú možnosť vytvárania veľkých znalostných grafov, pomocou textovej analýzy a indexovania pre sémantické vyhľadávanie.

Existujú také, ktoré boli vytvorené nad relačnými databázovými základmi s bázou SQL, ale problém u nich znamenala implementácia efektívneho dopytovania, mapovanie grafového modelu na SQL dopyty.

Nasleduje predstavenie, v dnešnej dobe, zopár najpoužívanejších RDF úložísk :

OpenLink Virtuoso

OpenLink Virtuoso[2] je univerzálny server, hybrid vytvorený z objektovo-relačného databázového systému (anglicky *object–relational database management system*, skratkou ORDBMS) a webového aplikačného serveru (anglicky *Web Application Server*) [31]. V rámci jediného viacvláknového serverového procesu zabezpečuje manažment SQL, XML a RDF dát.

[2] https://virtuoso.openlinksw.com/

S úložiskami trojíc je možné pracovať, okrem veľa ďalších možností, aj pomocou SPARQL. Je dostupný v open-source, aj v komerčnej verzií.

Blazegraph

Blazegraph[3] je open-source vysoko výkonná grafová databáza, podporujúca RDF dátový model a RDF/SPARQL rozhrania pre programovanie aplikácií (anglicky *Application Programming Interface*, skratkou API) [2]. Produkt pokrýva všetky aplikačné potreby od malých aplikácií, so vstavaným úložiskom, do väčších samostatných aplikácií. Celý bol implementovaný v programovacom jazyku Java.

Apache Jena - TDB

Apache Jena[4] je open-source rámec (anglicky *framework*) sémantického webu pre jazyk Java [32]. Poskytuje programové prostredie pre RDF, RDFS, OWL a SPARQL, spolu s inferenčným mechanizmom, založeným na pravidlách. Grafy reprezentuje v podobe abstraktného modelu a pomocou API umožňuje extrakciu z grafov a vkladanie do nich.

TDB je komponentom tohto rámca, ktorý zabezpečuje ukladanie RDF dát a dopytovanie nad nimi [1]. Na jednom zariadení je schopný byť v úlohe vysokovýkonného RDF úložiska.

RDF4J

Eclipse RDF4J je open-source rámec pre ukladanie, analyzovanie, dopytovanie nad RDF dátami [25]. Ponúka API, pomocou ktorého je možné jednoducho sa pripojiť k všetkým popredným riešeniam RDF úložísk a ku SPARQL koncovým bodom (anglicky *endpoint*). Vo frameworku je možné nájsť podporu pre všetky známe RDF formáty, ako napríklad N-Triples, RDF/XML, Turtle.

RDF4J poskytuje implementáciu veľkej škály nástrojov, ktoré zabezpečujú vývojárom využiť silu RDF a súvisiacich noriem. Dva z najpoužívanejších nástrojov sú RDF4J Server a RDF4J Workbanch.

Server je aplikácia zameraná na manažovanie databázy. Cez SPARQL endpoint zabezpečuje prístup k RDF4J repozitárom ostatným aplikáciám.

Workbanch poskytuje grafické užívateľské rozhranie, cez ktoré je možné pracovať s úložiskami. Pomocou SPARQL dopytov je možné prechádzať, manipulovať dáta v repozitároch, upravovať menné priestory a samotné repozitáre.

GraphDB

Ontotext GraphDB[5] skupina robustných, vysoko efektívnych RDF databáz s možnosťou škálovateľnosti [21].

Implementuje rozhranie rámca RDF4J, špecifikáciu SPARQL protokolu a podporuje všetky formy serializácie RDF dát. Je to jeden z mála úložísk trojíc, ktorý je schopný vykonávať sémantické dedukovanie vo veľkom, umožňuje užívateľom odvodiť nové sémantické fakty z už existujúcich faktov. Jeho silnú stránku predstavuje schopnosť spracovať veľké množstvo dopytov v reálnom čase.

[3] https://blazegraph.com/
[4] https://jena.apache.org/
[5] https://www.ontotext.com/products/graphdb/

2.6 Dopytovanie nad RDF dátami - SPARQL

Simple Protocol and RDF Query Language, skrátene *SPARQL*, poskytuje štandardizovaný dopytovací jazyk pre RDF grafy, podobne ako SQL, naprieč relačnými databázovými systémami. Jazyk nie je limitovaný iba na dáta uchovávané v známych RDF formátoch. Je schopný pomocou nástrojov spracovávať relačné dáta, XML, JSON a aj iné formáty, ako RDF, aby bolo možné zadávať dopyty SPARQL na údaje v týchto formátoch.

Pre zvýšenie jej miery adaptácie, časť syntaxe bola prevzatá z SQL. Presnejšie typický vzor SELECT dopytov – `SELECT <šablóna výsledku> FROM <definícia súboru dát> WHERE <vzor dopytu>` – bol adaptovaný do kontextu SPARQL [5].

Princípom pri vytváraní odpovedí na dopyty, je odštartovanie hľadania pri určitom uzle a postupné prechádzanie medzi uzlami, nasledovaním príslušných hrán až do dosiahnutia cieľa. Tento prístup je značne odlišný od konceptu spojovania tabuliek, ktorý používa SQL.

SPARQL dopyty sa snažia nájsť vzory v grafoch a naviazať na nich premenné pri hľadaní odpovedí. Je možné vytvoriť štyri typy dopytov: *SELECT*, *CONSTRUCT*, *ASK*, a *DESCRIBE*. Všetky z týchto dopytov zdieľajú rovnakú konštrukciu a môžu začínať s blokom textu, vyhradeným pre deklarovanie prefixov. Prefixy priraďujú skrátené identifikátory pre URI menných priestorov, ktoré je možné použiť v celom dopyte. Deklarácia každého prefixu sa začína kľúčovým slovom *PREFIX* na osobitnom riadku.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

Výpis 2.10: Príklad definície prefixov v jazyku SPARQL. Na každom riadku je uvedený prefix a za ním menný priestor, ktorý reprezentuje.

Kľúčové slová v dopytoch, ako sú PREFIX, SELECT, WHERE, sa píšu podľa konvencie veľkými písmenami, ale reálne nie sú citlivé na veľkosť písmen. Komentáre v týchto dopytoch začínajú na novom riadku so znakom #. Súbory, obsahujúce SPARQL dopyty, majú príponu vo forme *.rq*.

Pri popise jazyka SPARQL a jeho kľúčových slov, častí sa vychádzalo z článkov [11] [20] [27] a z kníh [5] [9] [26].

SELECT, FROM a WHERE

Kľúčové slovo SELECT v dopytoch umožňuje identifikovať podmnožinu premenných vo vzore grafu, ktorých väzby nás zaujímajú vo výsledku.

Premenné za slovom SELECT sú silným zástupným znakom (anglicky *wildcard*). Pomocou nich je možné vyjadriť to, že trojice s ľubovoľnou hodnotou na zvolenej pozícií sú pre nás zaujímavé. Hodnoty, ktoré sa vyskytnú na zvolených pozíciách sa uložia do premenných a stanú sa použiteľné ďalej v rámci dopytu. Príklad na tento typ dopytu je znázornený v 2.11.

```
PREFIX : <http://example.com/music>
SELECT ?cd ?singer
WHERE {
   ?cd a :Cd .
   ?cd :singer ?singer .
   ?singer a :RockSinger .
}
```

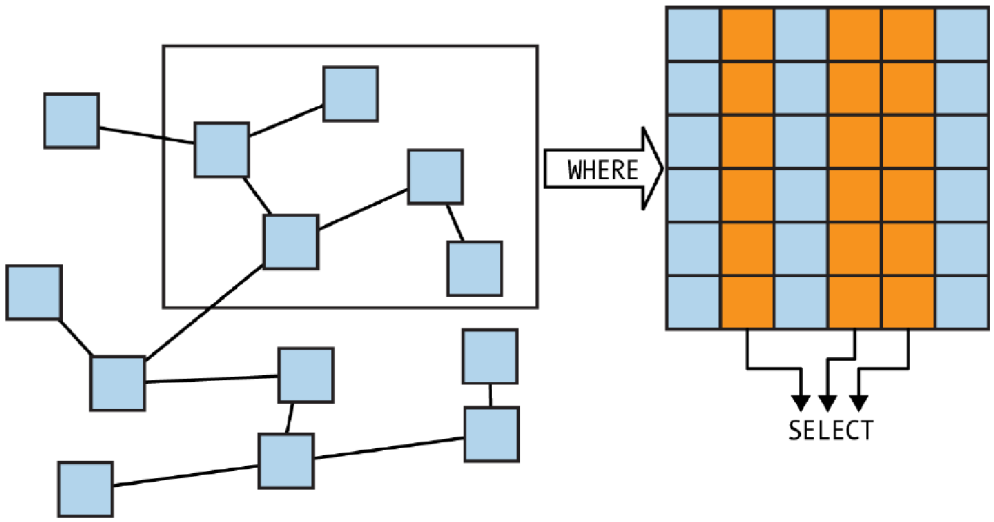
Výpis 2.11: Príklad využívania premenných v SPARQL dopytoch. Premenná s naviazanou hodnotou sa dá ďalej využívať v rámci dopytu v iných vzoroch trojíc. Výsledkom dopytu bude dvojica titul CD a názov speváka, ktorý na danom CD spieva rockové skladby.

Dátovú sadu, nad ktorou budú prevedené dopyty, je možné určiť pomocou kľúčového slova FROM.

SELECT a FROM klauzulu nasleduje WHERE klauzula, ktorá špecifikuje vzor grafu, ktorý je potrebný nájsť v kolekcií trojíc. Premenné vo vzore sú identifikované pomocou reťazcov, začínajúcich znakom *?* alebo *$*.

Klauzuly vo WHERE časti majú podobu štandardných RDF trojíc. Jediný rozdiel predstavuje možnosť nahradiť ľubovoľný prvok trojice premennou. Tie premenné, ktoré nie sú uvedené za slovom SELECT, sa používajú na prepojenie rôznych vzorov trojíc v klauzulách.

Znázornenie vzťahu medzi SELECT a WHERE časťou je zobrazené na obrázku 2.5.



Obr. 2.5: V SPARQL dopytoch WHERE klauzula určuje, ktoré dáta treba vytiahnuť z grafu a SELECT, ktoré časti týchto dát treba vrátiť ako výsledok. Prevzaté z [9].

Ak za slovom SELECT je uvedený symbol *, tak podobne ako u SQL, všetky premenné použité v klauzule WHERE budú zahrnuté do výsledku.

OPTIONAL a FILTER

Kľúčové slovo OPTIONAL umožňuje vrátiť ako výsledok aj také dáta, v ktorých sa nepodarí všetky vzory trojíc naviazať. Na určenie trojíc, v ktorých sa môžu vyskytovať nenaviazané časti, sa používa klauzula OPTIONAL, uvedená v klauzule WHERE. Bez tohto kľúčového slova všetky premenné, určené v časti SELECT, musia mať vo výstupe naviazané hodnoty,

ale s použitím OPTIONAL, je možné vyznačiť, u ktorých je dovolené, aby hodnota vo výstupe chýbala. Táto klauzula pomáha jednoduchšie zvládnuť problémy, spojené s chýbajúcimi dátami. Príklad, na použitie tejto klauzuly, je uvedený v 2.12.

```
PREFIX : <http://example.com/music>
SELECT ?cd ?reldate
WHERE {
   :michael_jackson :singerOf ?cd .
   OPTINAL { ?cd :release_date ?reldate . }
}
```

Výpis 2.12: Príklad využitia kľúčového slova OPTIONAL v SPARQL dopytoch. Výsledkom dopytu budú dvojice, názov CD a dátum vydania. V rámci dvojice môže chýbať hodnota dátumu, ak k danému CD nebola uvedená v grafe.

V dopytoch, grafové vzory sú na určenie vzťahov, ktoré hľadáme v rámci grafu. Často sa stáva situácia, keď je potrebné obmedziť riešenia na základe určitých vlastností subjektu, predikátu alebo objektu. Na filtrovanie vlastností výsledkov slúži kľúčové slovo FILTER, umožňujúce špecifikovanie ďalších obmedzení na naviazanie premenných na hodnoty. Vytvárané obmedzenia používajú určitú sadu operátorov, ktoré slúžia na testovanie premenných v grafovom vzore pre konkrétne podmienky. Napríklad, často je dobré testovať, či určitý literál typu reťazec nasleduje zvolený vzor. Pre tieto typy porovnávaní reťazcov poskytuje SPARQL operátor *regex()*. Príklad na použitie tejto klauzuly spolu s regex() je uvedený v 2.13. Ak FILTER vráti nepravdivú hodnotu, tak aktuálne zvažované riešenie je odstránené z výsledku.

```
PREFIX : <http://example.com/music>
SELECT ?cd ?singer
WHERE {
   ?singer :singerOf ?cd .
   ?singer :name ?name .
        FILTER (regex(?name, "Michael", "i"))
}
```

Výpis 2.13: Príklad využitia kľúčového slova FILTER v SPARQL dopytoch. Výsledkom dopytu budú dvojice, názov CD a meno speváka. Meno speváka musí obsahovať v akejkoľvek podobe reťazec „Michael“. Filtrovanie je tu nezávislé na veľkosti písmen.

Ďalšie užitočné kľúčové slová

SPARQL umožňuje špecifikovanie viacerých grafových vzorov v jedinom dopyte, pomocou kučeravých zátvoriek na zoskupenie trojíc do oddelených vzorov. Všetky tieto vzory sú vyhodnotené spoločne pre vytvorenie výsledku. Použitím kľúčového slova UNION môžeme dosiahnuť, aby všetky uvedené vzory boli vyhodnotené zvlášť. Hlavný výsledok obsahuje akékoľvek údaje, vyhovujúce aspoň jednému z týchto vzorov.

Existujú situácie, keď je nutné zo sady riešení zostaviť nový graf. Na tento účel sa využíva klauzula CONSTRUCT, ktorá nahrádza klauzulu SELECT. Rozdiel medzi dvomi klauzulami je v časti šablóny výsledkov, ktoré určujú šablónu pre trojice v novom grafe. CONSTRUCT uplatní vzor dopytu pre naplnenie hodnôt premenných v šablóne a vráti RDF graf. Príklad, na vytvorenie nového grafu, je uvedený v 2.14.

```
PREFIX : <http://example.com/music>
CONSTRUCT {
    ?person :took_part_in ?cd
}
WHERE {
    {
       ?cd :singing ?person .
    }
    UNION {
       ?cd :playing_instrument ?person .
    }
}
```

Výpis 2.14: Príklad využitia kľúčového slova CONSTRUCT a UNION v SPARQL dopytoch. Výsledkom dopytu bude nový graf vytvorený z trojíc, ktoré vyhovovali podmienkam. Union v tomto dopyte spája dohromady výsledky dvoch vzorov.

SPARQL prináša jednoduché kľúčové slovo ASK, ktoré slúži na testovanie toho, či určitý vzor sa nachádza v grafe. V dopytoch nahrádza kľúčové slovo WHERE a ako výsledok je vrátená pravdivostná hodnota, ktorá označuje, či existuje riešenie pre vzor v grafe. Príklad, na tento typ klauzuly, je uvedený v 2.15.

```
PREFIX : <http://example.com/music>
ASK
{
    ?cd :singer :michael_jackson .
    ?cd :singer :eminem .
}
```

Výpis 2.15: Príklad využitia kľúčového slova ASK v SPARQL dopytoch. Výsledkom dopytu je pravdivostná hodnota na základe toho, či existuje také CD, na ktorom spievali obaja speváci aspoň v jednej skladbe.

Z klasických SQL kľúčových slov sú tiež podporované: LIMIT, DISTINCT, OFFSET, ORDER BY, GROUP BY a HAVING.

Kapitola 3

Tvorba klientskych webových aplikácií

Webové aplikácie v dnešnej dobe patria medzi najpoužívanejšie typy aplikácií. Sú potrebné pre každodenný život, aj pre profesionálne prostredie. V úvode už bolo spomenuté, že internet sa stal vo veľkej miere súčasťou života ľudí. Pripájajú sa k nemu pomocou rôznych zariadení a používajú ho na vybavovanie dôležitých osobných, či pracovných činností alebo iba pre rôzne formy zábavy, oddychu. Tento trend spôsobil nárast dopytu o webové aplikácie, ktoré začali postupne nahradzovať doteraz používané desktopové aplikácie. Ľudia si rýchlo zvykli na to, že často bez nutnosti inštalácie špeciálnych aplikácií, sú schopní rovnaké činnosti spraviť online na odlišných zariadeniach, ako sú napríklad mobilný telefón, notebook alebo televízor. Firmy začali reagovať na dopyt na trhu. Začali sa zaoberať vývojom a tvorbou nástrojov, ktoré umožnili tvorbu takýchto aplikácií, snažia sa zrýchliť a zjednodušiť určité etapy vývoja. Táto kapitola sa zaoberá práve nástrojmi, ktoré sú vhodné pre tvorbu klientskych webových aplikácií.

Prvá sekcia 3.1 je úvodom do problematiky webových aplikácií a základného konceptu. Poukazuje na rozdiely medzi bežnými webovými stránkami a aplikáciami.

V prevažnej väčšine projektov, ktoré sa dnes zameriavajú na web, sa objavuje určite v nejakej podobe a miere jazyk JavaScript, popísané v sekcii 3.2. V dnešnej dobe sa moderný webový vývojár bez poznatku tohto jazyka nezaobíde.

Podobne, ako u iných jazykoch, ako sú napríklad Python, C++, Java, tak aj u JavaScriptu, sa časom vytvoril značný počet podporných knižníc a aplikačných rámcov. Tieto buď rozširujú základné schopnosti jazyka, pridávajú nové funkcionality, alebo boli postavené práve na tomto jazyku. Sekcia 3.3 predstavuje také knižnice a rámce, ktoré často slúžia ako základne stavebné kamene web aplikácií.

Jeden z najpoužívanejších rámcov v posledných rokoch je Vue.js. Stal sa obľúbeným pre svoju jednoduchosť a strmú učiacu sa krivku. Jeho predstavenie je uvedené v sekcii 3.4.

3.1 Webové aplikácie

Záujem o web ešte nikdy nebol taký vysoký, ako je teraz. Moderný web konštantne rastie a postupne viac a viac zariadení má prístup k nemu a vie s ním komunikovať.

Na začiatku vzniku informačných technológií užívatelia komunikovali s aplikáciami umiestnenými na serveroch pomocou terminálov. Aplikácia vtedy bežala na jedinom mieste a všetci s ňou pracovali. Postupným vývojom osobných počítačov a ich rastúcim výkonom sa rozšírili

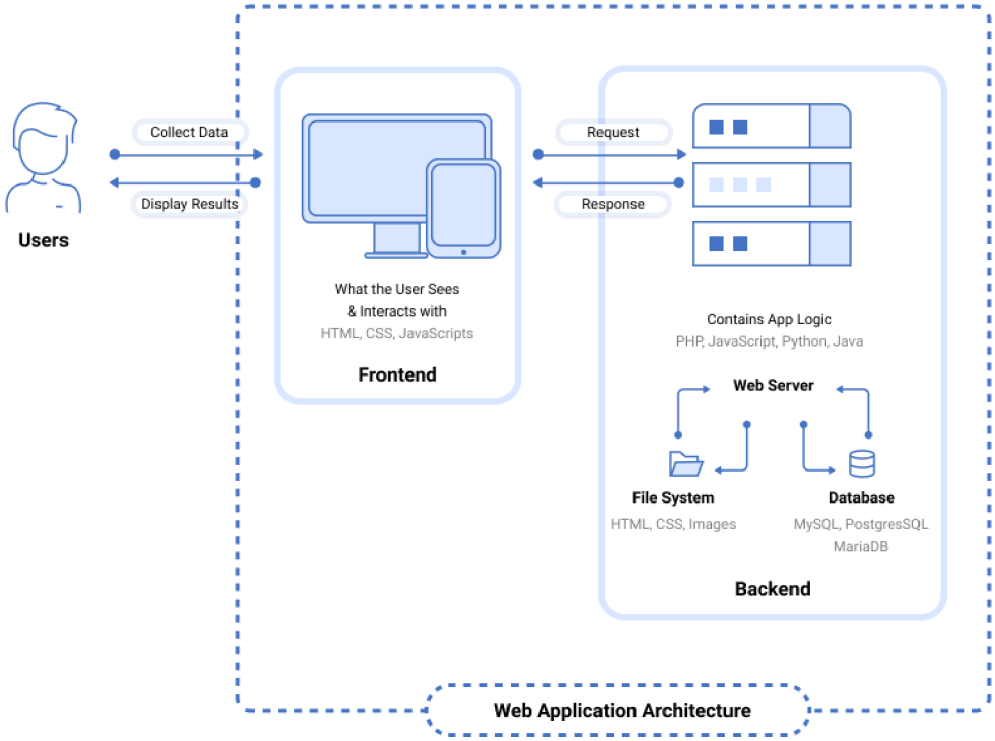
tlstí klienti. Ich biznis logika už bola použitá v aplikáciách distribuovaných k užívateľom. Nevýhodou tohto prístupu boli problémy so samotnou distribúciou ku klientom, ako aj s udržiavaním rôznych verzií na jednotlivých zariadeniach.

S rozmachom webu sa stalo možným kombinovať predošlé postupy. Teraz už servery distribuujú softvér na základe požiadaviek prichádzajúcich od užívateľov, namiesto toho, aby boli u nich lokálne inštalované. Dlhé roky webové štandardy a výkon samotných prehliadačov zaostávali za možnosťami natívnych aplikácií. Odvtedy sa to rapídne zlepšilo a web sa stal nástrojom pre dodávanie stále komplexnejších aplikácií na mnoho rôznych zariadení a stal sa prirodzeným miestom pre ich vývoj.

Webová aplikácia je počítačový program, ktorý používa webový prehliadač (anglicky *web browser*) pre zobrazenie a prevedenie určitých činností. Tieto aplikácie nasledujú klient-server model a pozostávajú z dvoch časti. Jedna z nich je na strane užívateľa, klienta (anglicky *client-side*) a druhá na strane servera (anglicky *server-side*) a udržiavajú medzi sebou aktívne sieťové prepojenie. Pojem klient označuje program, ktorý sa používa na spustenie aplikácie. V úlohe takýchto programov vystupujú webové prehliadače, ako sú napríklad Google Chrome, Mozilla Firefox a Safari, dostupné na rôznych typoch platforiem.

Medzi bežne používané webové aplikácie patria: webová pošta, online maloobchodný predaj, online bankovníctvo a online aukcie.

Jednotlivé aplikácie ponúkajú funkcionality, ktoré sa bežne rozdeľujú medzi tri programy: webový server, aplikačný server a databáza. Požiadavky prichádzajúce od klienta sú obdržané a spravované webovým serverom, kým aplikačný server prevedie činnosti spojené s požadovanou úlohou. Databázy slúžia na uloženie všetkých potrebných informácií.



Obr. 3.1: Diagram, zobrazujúci možnú architektúru webovej aplikácie. U jednotlivých častí sú uvedené programovacie jazyky pomocou, ktorých sa dajú vytvoriť. Prevzaté z [23].

Vývoj týchto aplikácií má rôznu úroveň obťažnosti, závisí na komplexnosti poskytovaných funkcionalít. Typickým sú krátke vývojové cykly, prevedené v rámci menších vývojových tímov. Pre budovanie prezentačnej vrstvy (anglicky *front-end*) sa využívajú technológie HTML5, kaskádové štýly (anglicky *Cascading Style Sheets*, skratkou CSS) a JavaScript. Tieto jazyky umožňujú vytvoriť zdrojový kód, ktorý je vykonateľný na prehliadači. Jazyky ako Python, Java, PHP a Ruby sú bežne používané na programovanie skriptov, ktoré pobežia na strane servera. Niektoré z vytvorených aplikácií potrebujú spracovanie na strane servera (anglicky *server-side processing*) a označujú sa ako dynamické. Tie, ktoré nevyužívajú server takýmto spôsobom, sa označujú ako statické.

Nevýhodou webových aplikácií, oproti natívnym aplikáciám, sa môže zdať ich závislosť na aktívnom internetovom spojení, cez ktoré komunikujú so serverom. S objavením sa nových technológií, ako sú service workers (slovensky voľne preložene *servisný pracovník*) a progresívnych webových aplikácií (anglicky *Progressive Web Apps*, skratkou PWA) sa táto nevýhoda postupne stáva minulosťou. Tento nový typ aplikácie sa podobá webovým stránkam, ale prináša nové možnosti v podobe off-line práce, push notifikácií a prístupu k hardvéru.

Rozdiel medzi web aplikáciami a stránkami

Webové aplikácie sa dnes stali v takej miere súčasťou nášho života, že niekedy si ich už ani nevšimneme. Vo veľkej miere splynuli s webovými stránkami a ľudia prestali vnímať rozdiel medzi nimi a označujú ich spoločne pod jedným názvom ako webové stránky.

Webové stránky (anglicky *website*) sú tvorené skupinou globálne dostupných internetových stránok, dostupných pod jediným doménovým menom. Stránky sú medzi sebou prepojené pomocou hypertextových odkazov, umožňujúcich navigáciu na webe. Tieto stránky sú venované konkrétnej téme alebo účelu, ako sú správy, obchodovanie, zábava a sociálne siete. Medzi najznámejšie patria *google.com*, *wikipedia.com* a *youtube.com*. Predstavujú účinný spôsob predvádzania produktov a služieb, pomáhajú pri budovaní značiek firiem.

Najväčšie rozdiely medzi webovými aplikáciami a stránkami sú uvedené v nasledujúcej tabuľke 3.1, ktorá bola inšpirovaná obsahom stránky [15].

Táto sekcia vychádzala z knihy [18] a webových článkov [13] [15].

3.2 JavaScript

JavaScript, skrátene *JS*, bol vytvorený v máji 1995 vývojárom Brendan Eich za necelých desať dní [24]. Počas rokov jeho existencie, vystupoval pod niekoľkými menami ako Mocha, LiveScript a najnovšie *ECMAScript*. JavaScript a ECMAScript v zásade znamenajú to isté. Prvý termín odkazuje na jazyk a jeho implementácie, druhý termín ECMAScript označuje jazykovú normu a jazykové verzie.

Po boku HTML a CSS patrí medzi základné technológie webu. Skoro sto percent webových stránok ho využíva na strane klienta, na riadenie reagovania stránok. Väčšina webových prehliadačov má zabudovaný v sebe dedikovaný JavaScript engine pre vykonanie kódu v zariadení užívateľa.

Vďaka jeho štandardizácie a veľkej adaptácie internetu sa stal najpoužívanejším programovacím jazykom na planéte.

Parametre	Aplikácia	Stránka
Vytvorené pre	Je vytvorená pre interakciu s užívateľom.	Pozostáva zo statického obsahu verejne prístupného pre všetkých návštevníkov.
Interakcia používateľa	Užívateľ okrem čítania obsahu aj manipuluje s dátami.	Užívateľ si môže prehliadnuť vizuálny obsah na stránke, ale nemôže s ňou manipulovať.
Autentizácia	Tieto aplikácie vyžadujú autentizáciu pre ochranu citlivých dát pred nepovolenou manipuláciou alebo prístupom k nim.	Autetnizácia nie je povinná pre informačné stránky. Užívateľ môže byť požiadaný o registráciu pre získanie prístupu k podrobnejším informáciám alebo pre získanie pravidelných aktualizácií.
Zložitosť úlohy	Poskytované funkcie sú značne komplexnejšie oproti webovým stránkam.	Stránky iba zobrazujú kolekcie dát a informácie.
Kompletnosť	Aplikácie sú súčasťou stránok a vyvíjajú sa v rámci nich. Samy osebe nevytvárajú úplnú webovú stránku, ale sú v nich obsiahnuté.	Tvoria kompletný produkt prístupný cez prehliadač.
Kompilácia	Aplikácia musí byť predkompilovaná pred nasadením.	Stránka nemusí byť predkompilovaná.
Nasadenie	Všetky zmeny vyžadujú kompilovanie celého projektu a znovu nasadenie.	Malé zmeny nevyžiadajú úplnú kompiláciu, stačí iba aktualizovať HTML kód.

Tabuľka 3.1: Niekoľko fundamentálnych rozdielov medzi webovými aplikáciami a stránkami inšpirované z článku [15].

Vlastnosti jazyka JavaScript

Definícia vlastností tohto jazyka je veľmi rozsiahla, lebo JavaScript môže byť označovaný ako procedurálny jazyk. Je založený na prototypoch, je imperatívny, slabo typovaný a dynamický [6].

Pri návrhu jeho syntaxe sa vychádzalo zo známeho jazyka *C*, ale konvencie boli prevzaté z jazyka *Java*. Napriek podobnému názvu, Java a JavaScript, navzájom nesúvisia a majú odlišnú sémantiku i účely.

Slabá typovanosť JS znamená, že určité typy sú implicitne priradené na základe použitých operácií. Podobne, ako ostatné skriptovacie jazyky, aj tento jazyk je dynamicky typovaný. To znamená, že nepožaduje špecifikáciu dátového typu premenných a tie môžu teda odkazovať na hodnotu akéhokoľvek typu. Napríklad, premenná pôvodne viazaná na číslo, môže byť znova priradená k reťazcu.

Oproti mnohým objektovo orientovaným jazykom, ktoré používajú triedy pre dedičnosť, JS používa prototypovanie. Pre opätovné použitie správania sa objektov používa zovšeobecnené objekty, ktoré je potom možné klonovať a rozšíriť.

Ako multi-paradigmatický jazyk, JS podporuje udalosťami riadené, funkcionálne a imperatívne štýly programovania. Ponúka rozhranie pre programovanie aplikácií (anglicky *Application Programming Interface*, skratkou API) pre prácu s textom, regulárnymi vý-

razmi, dátumami, štandardnými dátovými štruktúrami a objektovým modelom dokumentu (anglicky *Document Object Model*, skratkou DOM).

Bežné využitie jazyka

JavaScript je použitý vo väčšine dnešných webových stránok. Využíva sa najmä ako programovací jazyk na strane klienta, ako časť webového prehliadača, ktorý umožňuje vývojárom vytvárať vylepšené užívateľské rozhrania s dynamickými funkcionalitami [6].

Počas uplynulých rokov využívania JS v praxi sa jeho schopnosti rozšírili. Medzi prvé ponúkané funkcie patrili interakcie s formulármi, detegovanie činností užívateľa na stránke a ich informovanie o udalostiach, napríklad pomocou funkcie *alert()*. V dnešnej dobe sa pomocou nej dajú realizovať aj činnosti, ako sú [6]:

- Formuláre, overujúce vstupné hodnoty pred odoslaním údajov na webový server.
- Automatické doplňovanie - vyhľadávacie pole môže navrhovať výsledky počas písania a to na základe toho, čo už bolo zapísané.
- Periodické načítavanie informácií bez nutnosti interakcie užívateľa.
- Generovanie vyskakovacích reklám.
- Presmerovanie používateľa na inú stránku.
- Animácie webových stránok, ako napríklad blednutie a zatmievanie predmetov, zmena veľkosti a ich presúvanie.

Existuje veľmi málo vecí, ktoré nie je možné vykonať pomocou JS, hlavne, keď je kombinovaný aj s inými technológiami. Na druhej strane, pri tvorbe aplikácií, je potrebné myslieť aj na tú situáciu, keď podpora tohto jazyka z nejakého dôvodu nie je zapnutá v prehliadači a treba správnym spôsobom reagovať na túto situáciu.

Jednou z dôležitých zásad, na ktoré je potrebné myslieť pri voľbe tohto jazyka je, že JS by nemal byť nikdy jedinou mierou ochrany. Aplikácie vyžadujúce ochranu citlivých dát pred neoprávneným prístupom alebo autentizáciu užívateľov, by mali mať implementované aj iné ochranné prostriedky na serveroch. JavaScriptová ochrana môže byť jednoducho prelomená vďaka tomu, že celý zdrojový kód stránok sa dá prehliadnuť na zariadení užívateľa. Ak by bol JS jednoducho vypnutý v prehliadači, tak je vypnutá celá ochrana a validácia dát.

Formy využívania tohto jazyka sa už rozšírili i mimo svoje korene, ktoré tvorili prehliadače. JavaScript motory (anglicky *engines*) sa stali súčasťou množstva ďalších softvérových systémov, na serveroch a aplikáciách, používaných mimo prehliadačov. Medzi známe aplikácie, ktorých fungovanie bolo programované pomocou JS, patrí aj Adobe Acrobat.

Vďaka veľkému množstvu rámcov, ktoré vychádzajú z JavaScriptu, sa tento jazyk v menšej miere začal používať aj u vstavaných systémov.

Popis vlastností a využívania jazyka JS v tejto sekcii vychádza z kníh [6] [24].

3.3 Knižnice a rámce pre tvorbu webových aplikácii

Knižnice a rámce predstavujú verejne dostupný, znovu použiteľný kód, vytvorený inými vývojármi. Ich cieľom je pomáhať pri riešení často sa opakujúcich úloh a problémov, zrýchľujúc tak celkový priebeh vývoja. Podporujú tak dobre známy princíp minimálneho opakovania rovnakého kódu (anglicky *Don't repeat yourself*, skratkou DRY) v rámci implementácie.

Tieto dva termíny sa často zamieňajú, pričom je medzi nimi značný technický rozdiel. Tento rozdiel spočíva v inverzii ovládania. Pri využívaní knižnice, programátor je zodpovedný za riadenie toku programu. On rozhoduje o tom, kedy a ako zavolá metódy, poskytované knižnicou. V prípade rámcov je to opačne, riadenie toku je už dopredu stanovené. Programátorovi poskytujú niekoľko miest na vloženie vlastného kódu a podľa potreby volajú kód, ktorý on vytvoril. Rámce sú zvyčajne komplexnejšie. Definujú kostru, kde aplikácia definuje svoje vlastné vlastnosti na vyplnenie kostry.

Kvôli veľkej obľube a rozšírenosti jazyka JavaScript sa od jeho existencie vytvorilo mnoho knižníc a rámcov, ktoré sú na ňom založené a rozširujú jeho možnosti. Následne uvádzame tie najpoužívanejšie, ktoré sa používajú pri tvorbe klientskych webových aplikácií.

jQuery a jQuery UI

jQuery[1] je malá, rýchla, open-source javascriptová knižnica, ktorá kladie dôraz na interakciu medzi JavaScriptom a HTML. Pri návrhu jej syntaxe bolo cieľom zjednodušiť navigáciu v dokumente, výber DOM elementov, vytváranie animácií, spracovanie udalostí a vývoj Ajax aplikácií s ľahko použiteľným rozhraním API, ktoré funguje v mnohých prehliadačoch. Jej syntax sa veľmi podobná CSS, a preto je pre začiatočníkov ľahké sa ju naučiť a používať. V dnešnej dobe je ešte stále prítomná v miliónoch najpoužívanejších stránok na svete, ale jej obľuba medzi vývojármi je už na zostupe.

jQuery UI[2] je kolekcia ovládacích prvkov v grafickom užívateľskom prostredí, animovaných vizuálnych efektov a tém, budovaných nad jQuery. Podporuje interakcie, ako sú napríklad presúvanie, sťahovanie a triedenie elementov. Prináša prvky, ako sú výber dátumu, posuvníky, ukazovateľ postupu, popisky a mnoho ďalších tém.

React.js

React.js[3], tiež známa ako ReactJS alebo React, je open-source, front-endová knižnica, vytvorená a udržiavaná spoločnosťou Facebook. Umožňuje jednoduché vytváranie interaktívnych užívateľských rozhraní.

React sa zaoberá iba so správou stavu a zobrazovania príslušných dát a komponentov na obrazovke, a preto sa samostatne nedá využívať pre tvorbu celých aplikácií. Nespolieha sa na Document Object Model, ale namiesto toho používa ľahký virtuálny DOM, ktorý informácie zobrazuje rýchlo.

Jej výhodou je, že je deklaratívna, čo znamená, že stačí stanoviť, čo chce programátor robiť, nie ako to robiť. Cena za rýchlosť zobrazovania sa odráža v množstve kódu, potrebného pre vytvorenie komponentov.

D3.js

Data-Driven Documents[4], skratka D3, je javascriptová knižnica, zameriavajúca sa na dynamickú a interaktívnu vizualizáciu dát v podobe rôznych grafov na stránkach. Využíva prístup založený na dátach a používa ho na manipuláciu s DOM. Podporuje HTML, CSS a SVG.

[1] https://jquery.com/
[2] https://jqueryui.com/
[3] https://reactjs.org/
[4] https://d3js.org/

Parsley

Parsley[5] je JS knižnica používaná na validáciu vstupov vo formulároch. Jej intuitívne API je schopné vybrať vstupy priamo z HTML tagov, bez nutnosti vytvárania javascriptového kódu. Vie dynamicky detegovať a kontrolovať zmeny vo vstupoch formulárov.

Leaflet

Leaflet[6] je open-source JavaScript knižnica pre tvorbu interaktívnych máp, vhodných aj pre mobilné zariadenia. Medzi možné typy interakcií patria posúvanie ťahom, navigácia pomocou klávesnice, približovanie pomocou gest na dotykových povrchoch a mnoho ďalších.

AngularJS

AngularJS[7] je open-source webový rámec zameriavajúci sa na front-end. Používa sa hlavne na tvorbu single-page aplikácií, ktoré pozostávajú z jedinej stránky, ktorej obsah sa mení dynamicky, na základe interakcie s užívateľom. Cieľom je zjednodušenie vývoja a testovania aplikácií, ktoré majú architektúru typu model–view–controller (MVC) a model–view–viewmodel (MVVM). Snaží sa oddeliť prezentačné, dátové a logické komponenty od seba do samostatných vrstiev.

Rámec prispôsobuje a rozširuje tradičný HTML pre prezentovanie dynamického obsahu pomocou obojsmernej väzby dát. Táto väzba umožňuje automatickú synchronizáciu medzi modelom a pohľadom v aplikácií.

Node.js

Node.js[8] je open-source, multiplatformové, asynchrónne a udalosťami riadené behové prostredie, ktoré vykonáva javascriptový kód mimo webového prehliadača. Tento rámec umožňuje vytváranie nástroja príkazového riadku a skripty spúšťané na strane servera pre vytváranie dynamického obsahu webových stránok, pred ich zobrazením v prehliadači.

Vďaka jeho jednoduchosti, rýchlosti a dobrej podpory zo strany komunity, sa stal veľmi obľúbeným nástrojom. Často sa využíva pre tvorbu backendu, REST API, služieb v reálnom čase, ako sú chatové aplikácie, hry a nástroje.

Express.js

Express.js[9] je beck-endový rámec vytvorený pre Node.js. Poskytuje mechanizmy, middlewary, pre spracovanie prichádzajúcich požiadaviek na server. Obsahuje smerovaciu tabuľku na vykonávanie akcií na základe URL a HTTP metód.

Kým sám je dosť minimalistický, tak vývojári vytvorili kompatibilné balíky middlewaru, na riešenie takmer akéhokoľvek problému s vývojom webu. Existujú preň rozširujúce knižnice pre prácu s cookies, reláciami, prihlasovaním užívateľov, URL parametrami, bezpečnostnými hlavičkami a mnoho ďalších.

[5] https://parsleyjs.org/
[6] https://leafletjs.com/
[7] https://angularjs.org/
[8] https://nodejs.org/en/
[9] https://expressjs.com/

Ember.js

Ember.js[10] je produktívny a časom otestovaný JS rámec pre tvorbou jednostránkových web aplikácií. Zahrňuje v sebe osvedčené postupy, bežné idiómy z iných rámcov, kladie veľký dôraz na spätnú kompatibilitu.

Na rozdiel od mnohých iných rámcov, je možné na vytvorenie vývojového prostredia použiť celú sadu nástrojov, založených na technológii Ember. Medzi ne patrí aj napríklad rozhranie príkazového riadka Ember CLI, pracujúce ako základ pre Ember aplikácie a ponúka generátory kódu na vytváranie nových entít.

Je zaujímavé, že zatiaľ, čo sa tento rámec zameriava na vývoj webových aplikácií, je možné ho použiť aj na vytváranie mobilných a počítačových aplikácií – slúžil napríklad aj pre tvorbu Apple Music.

Backbone.js

Backbone.js[11] je veľmi ľahký, jednoduchý rámec, založený na MVC architektúre s RESTful JSON rozhraním. Poskytuje štruktúru aplikáciám pomocou modelov s väzbou typu kľúč-hodnota a vlastnými udalosťami, kolekciami s bohatým rozhraním API, s veľkým množstvom funkcií, pohľadov s deklaratívnym spracovaním udalostí. Keď akcia z užívateľského rozhrania spôsobí zmenu atribútu modelu, tak všetky pohľady, ktoré zobrazujú stav modelu sú upozornené a majú možnosť sa znova vykresliť s novými informáciami.

Ionic

Zaujímavým rámcom je Ionic[12], ktorý umožňuje vyvíjať hybridné mobilné, počítačové a progresívne webové aplikácie. Tie sú založené na moderných technológiách a postupoch pre webový vývoj. Tvorba prebieha s využitím webových technológií, ako sú CSS, HTML5 a Sass. Pre vytvorenie užívateľského rozhrania umožňuje používanie rámcov Angular, React alebo Vue.js

Aplikácie, tvorené pomocou tohto rámca, je možné nahrať do mobilných obchodov s natívnymi aplikáciami, ako sú Play Store alebo App Store.

3.4 Rámec Vue.js

Vue.js je progresívny javascriptový rámec pre tvorbu užívateľského rozhrania a jednostránkových aplikácií (anglicky *single-page applications*) [28]. Označuje sa ako progresívny kvôli tomu, že umožňuje začať budovanie aplikácie s vytvorením minimálneho úsilia, lebo jadro rámca sa zameriava iba na pohľadovú vrstvu. S rastom komplexnosti aplikácie je možné využiť ďalšie podporované knižnice a tak rozšíriť prístupné funkcionality [17]. Vue.js sa dá komplexne škálovať podľa požiadaviek projektu. V skutočnosti jeden z dôvodov, prečo vývojári si často vyberú práve tento rámec, je to, že sa dá ľahko s ním začať pracovať. S malou réžiou na začiatku projektu sa vývojár môže pustiť do práce a vytvárať výsledky bez ďalších zložitostí, prítomných pri iných rámcoch.

Používa vývojový model, založený na komponentoch, ktoré umožnia kombinovať a spájať Vue komponenty do projektov.

[10] https://emberjs.com/
[11] https://backbonejs.org/
[12] https://ionicframework.com/

Medzi kľúčové funkcie Vue patria šablóny, komponenty a obojsmerná dátová väzba, ale asi jeho najvýraznejšou vlastnosťou je reaktívnosť systému. V zásade to znamená, že zmena objektu vo Vue automaticky vyvolá nenápadnú aktualizáciu Vue šablón.

Pri popise častí samotného rámca a jeho rozširujúcich knižníc sa vychádzalo z knihy [17] a z stránok [28] [30] [29].

Šablóny

Vue.js pre zobrazovanie dát na stránkach využíva šablóny (anglicky *templates*) [28]. Na ich tvorbu bola vyvinutá špeciálna syntax, založená na HTML. Bola doplnená o takzvané v-direktívy, ktoré boli vytvorené z HTML atribútov pridaním prefixu „v-“, napríklad v-bind:class. Každá šablóna je platným HTML kódom, ktorý vie analyzovať väčšina prehliadačov.

Pre naviazanie vlastností priamo v HTML sa využíva značenie, ktorá sa volá syntax fúzov (anglicky *mustache syntax*) [17]. Uzatvára vlastnosti pomocou dvoch kučeravých zátvoriek, napríklad takto: *{{názov vlastnosti}}*. V rámci týchto väzieb je možné uviesť aj javascriptové výrazy, ktoré umožňujú na danom mieste previesť jednoduché výpočty, porovnanie hodnôt, zobrazenie výsledkov na základe vyhodnotenia ternárnych operátorov alebo volanie metód.

Táto syntax sa dobre využíva na naviazanie a zobrazenie hodnôt premenných, ktoré vedia nadobudnúť textovú podobu, ale nedá sa využiť pre naviazanie hodnôt k atribútom HTML prvkov. Na tento účel bola zavedená direktíva: v-bind. Pripája sa k atribútu elementu a pôvodná textová hodnota sa nahradí názvom premennej z Vue inštancie, ktorej hodnota bude použitá na tomto mieste.

Príklad využitia direktívy: `<div v-bind:name=”`*myName*`”></div>`.

Obojsmerná väzba dát

Vue.js používa obojsmernú väzbu dát (anglicky *two-way data binding*), čo značne uľahčuje prácu programátora v určitých aspektoch. Zjednodušuje synchronizáciu užívateľských vstupov s dátovým modelom aplikácie. Na využitie tejto funkcionality sa používa direktíva *v-bind*, ktorá pracuje so vstupnými prvkami, ako sú textové pole, časti formulárov, zaškrtávacie políčko. Viaže hodnotu vstupu na príslušnú vlastnosť dátového objektu, takže okrem toho, že vstup dostane počiatočnú hodnotu premenných, pri aktualizácii vstupu sa aktualizujú aj samotné premenné.

Zjednodušene, pri obojsmernej dátovej väzbe, ak používateľ vykoná zmenu dátového modelu prostredníctvom metódy, uvidíme, že používateľské rozhranie zobrazí aktualizáciu. Ak vykonáme zmenu v používateľskom rozhraní, dátový model sa aktualizuje tiež.

Komponenty

Štruktúra vytváraných stránok v rámci Vue sa rozdeľuje na menšie jednotky, komponenty. Sú to pomenované inštancie, opakovane použiteľné, podobné k widgetom. Najlepšie sa dajú porovnať k objektom v objektovo orientovanom programovaní (skratkou *OOP*). Podobne ako objekty, majú svoje vlastné atribúty, metódy a funkcie.

Komponent je samostatný kus kódu, ktorý predstavuje časť stránky. Komponenty majú vlastné dáta, vlastný javascriptový kód a často vlastný štýl. Môžu obsahovať iné komponenty a vedia navzájom komunikovať. Komponentom môže byť niečo také malé, ako je tlačidlo alebo ikona, alebo to môže byť niečo väčšie, napríklad formulár, ktorý sa často

používa v rámci viacerých stránok alebo celá stránka. Hlavnou výhodou rozdelenia kódu na komponenty je, že kód zodpovedný za každý kúsok stránky je blízko zvyšku kódu pre daný komponent. Keďže sú komponenty úplne sebestačné, autonómne, je zabezpečené, že žiadny kód vo vnútri komponentu neovplyvní iné komponenty alebo nebude mať žiadne vedľajšie účinky. V hierarchii komponentov, komunikácia medzi komponentmi môže prebiehať dvomi spôsobmi. Nadriadený komponent vie posunúť dáta podradenému, pomocou naviazania informácií parametrom pri volaní komponentu. Komunikácia opačným smerom prebieha pomocou vyvolania udalostí na nižšej úrovni a následného zachytenia komponentom na vyššej úrovni.

Väčšina aplikácií Vue sa stáva nakoniec kolekciou komponentov Vue, ktoré spolupracujú pri zobrazovaní údajov a reagujú na interakcie používateľa [17].

Vuex

Každá aplikácia má určité údaje vo forme hodnôt, ktoré súvisia s voľbami používateľa alebo zobrazovanými informáciami. Často sú tieto dáta využívane na viacerých miestach, v rámci jednej aplikácie alebo zdieľané medzi komponentmi. Tieto informácie spoločne tvoria stav aplikácie. Sledovanie a spravovanie týchto informácií sa označuje ako správa stavu (anglicky *state management*).

Vuex je knižnica, spravovaná tímom Vue, ktorá poskytuje správu stavov, spolu s niektorými ďalšími službami a funkciami [17]. Predstavuje jedno centralizované úložisko. Môže sa používať v celej aplikácii na ukladanie a prácu s globálnym stavom. Poskytuje možnosť overovať údaje vstupujúce do systému, aby sa zabezpečilo, že údaje, ktoré z neho vychádzajú, sú predvídateľné a správne. Toto úložisko je dostupné pre všetky Vue komponenty v programe.

Vuex má štyri hlavné časti: State, Getters, Mutations a Actions [30]. State (slovensky *stav*) obsahuje všetky údaje, ktoré zdieľame so systémom Vuex. Podobá sa jednému veľkému javascriptovému objektu s mnohými atribútmi.

Gettery ponúkajú spôsob konsolidácie výsledkov, ktoré sú založené na údajoch v úložisku. Je možné ich považovať za vypočítané vlastnosti, ktorým je umožnené prostredníctvom úložiska ich opakované použitie vo viacerých komponentoch Vue. Napríklad, takáto vlastnosť môže byť veľkosť poľa uloženého v stave.

Modifikáciu hodnôt v úložisku je možné previesť iba prostredníctvom mutácií (Mutations). Mutácia je funkcia, ktorá je schopná synchronizovane zmeniť stav a podobá sa na udalosť.

Asynchrónne zmeny stavu je možné previesť iba pomocou akcií (Actions). Sú podobné k mutáciám, ale sú použité pre úlohy, ktoré by mohli vyžadovať spätné volania alebo čakaciu dobu, napríklad volanie servera.

Vue-Router

Vo webových aplikáciach je smerovač (anglicky *router*) tá časť, ktorá je zodpovedná za synchronizáciu aktuálne zobrazeného pohľadu, s obsahom adresného riadku prehliadača. Inými slovami, spôsobuje zmenu adresy URL pri kliknutí na niečo na stránke a pomáha zobraziť správne pohľady pri návšteve špecifických adries.

Oficiálnou knižnicou poskytujúcou smerovač pre Vue je Vue-Router [29]. Má niekoľko pekných funkcií, ako sú napríklad vnorené trasy, modulárna konfigurácia, parametre trasy, parametre reťazca dopytu a zástupné znaky.

Pred prvým použitím smerovača je potrebné namapovať komponent na trasy, aby vedel presne, kde a čo má zobraziť. Pre vytváranie odkazov v šablónach sa namiesto bežných HTML značiek typu `<a>` používajú komponenty knižnice `<router-link>` [29]. Umožňujú Vue-Routeru odkazovať sa na iný komponent, zmeniť adresu URL bez opätovného načítania stránky, zvládnuť generovanie adresy URL a jej kódovanie.

Komponent `<router-view>` slúži pre zobrazenie komponent, zodpovedajúcim navštíveným cestám v rámci aplikácie. Vyznačuje miesto v šablónach pre ich vykreslenie.

Kapitola 4

Návrh aplikácie

Táto kapitola sa zaoberá návrhom aplikácie, ktorá je hlavým cieľom tejto práce. Na základe zadania bolo nutné navrhnúť aplikáciu, ktorá umožňuje užívateľovi zadávať interaktívnym spôsobom SPARQL dopyty a prechádzať obsah RDF úložísk. V nasledujúcich sekciách sú uvedené požiadavky a vlastnosti, týkajúce sa tejto aplikácie. Následne je uvedený popis plánu pre realizáciu požadovaných funkcionalít.

Prvým krokom bol prieskum už existujúcich nástrojov v rámci tejto problematiky, ktorý je popísaný v sekcii 4.1. Cieľom bolo zistiť ich stav a hlavne, aké základné operácie ponúkajú svojim zákazníkom.

Požiadavky týkajúce sa výsledného programu sú predstavené v sekcii 4.2. Stanovené boli na základe zadania a zistených rysov skúmaných riešení.

Sekcia 4.3 uvádza navrhnutú architektúru a najpodstatnejšie vlastnosti, stanovené na základe analýzy popísaných požiadaviek.

4.1 Prieskum existujúcich riešení

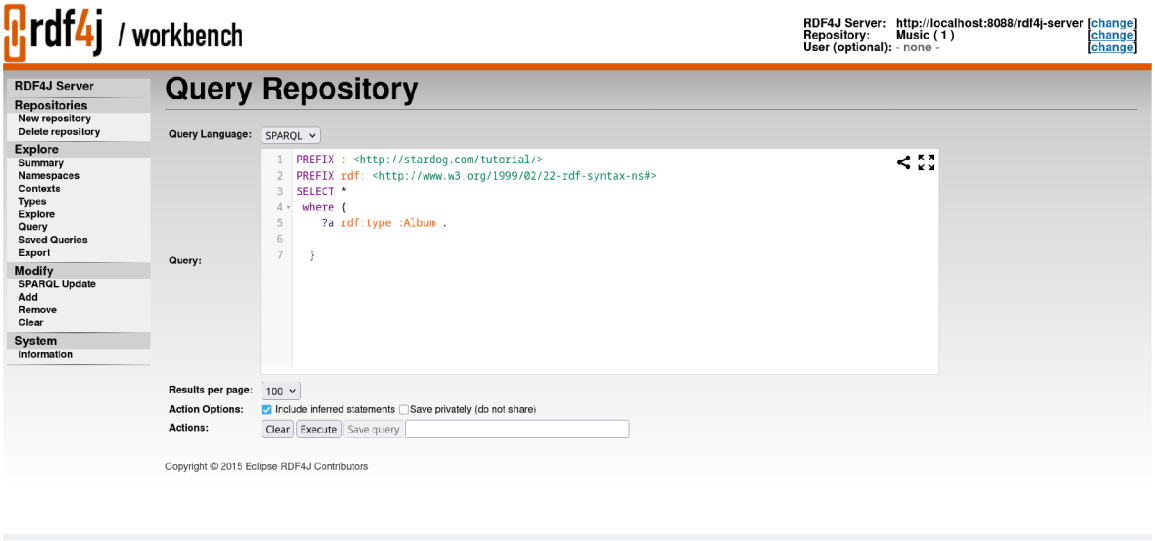
Na trhu je možné nájsť niekoľko komerčných aplikácií a nástrojov, ktoré boli vytvorené pre prácu s RDF úložiskami. Okrem základných funkcionalít, ako je možnosť prechádzať a modifikovať obsah jednotlivých repozitárov, ponúkajú aj editor pre zadávanie SPARQL dopytov. Príkladom môžu byť veľké firmy a nimi ponúkané nástroje, ako je napríklad *RDF Graph Server a Query UI* od spoločnosti Oracle alebo *GraphDB* od spoločnosti Ontotext.

Eclipse RDF4J, predstavený v sekcii 2.5, ponúka nástroj s názvom *RDF4J Workbench*, ktorého súčasťou sú užívateľsky orientované funkcionality. Poskytuje webové rozhranie na dopytovanie, aktualizáciu a skúmanie repozitárov uložených v RDF4J Serveri. Ukážku editora pre vytváranie dopytov vo zvolenom jazyku je možné vidieť na obrázku 4.1.

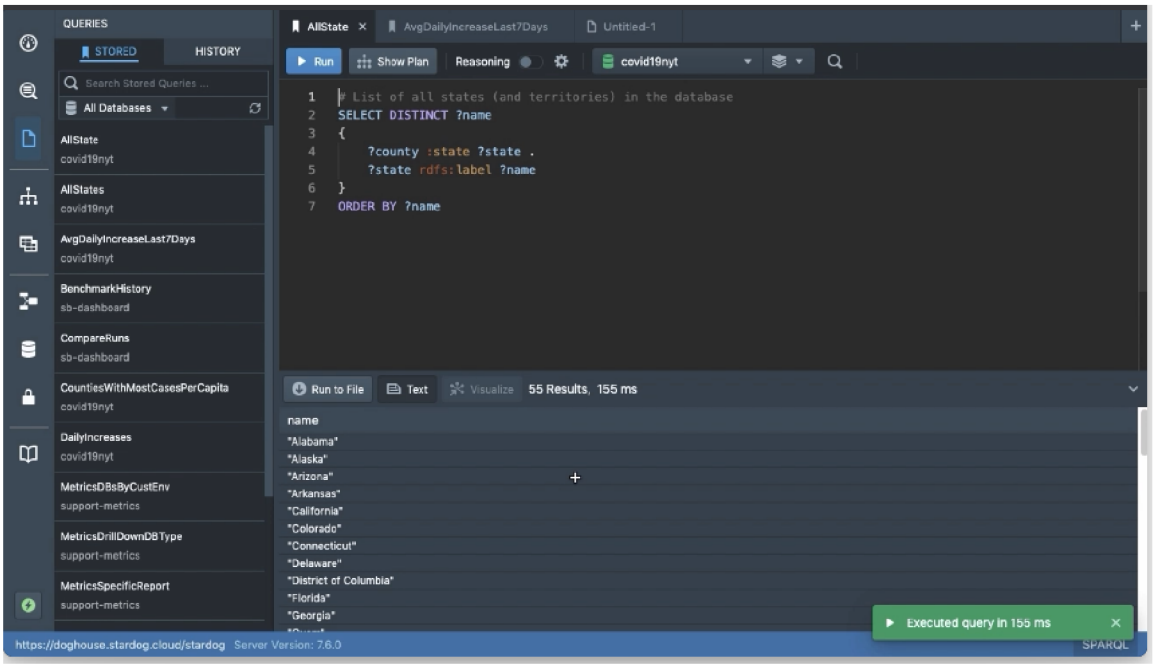
Zaujímavou aplikáciou je *Studio*[1] od spoločnosti Stardog. Medzi základné funkcionality, ktoré ponúka, patria: pracovný priestor pre vytváranie dopytov, interaktívne modelovanie dát a prieskum údajov spolu s vizualizáciou grafov. Ukážku zabudovaného editora a vizualizácie RDF grafu, v rámci aplikácie, je možné vidieť na obrázku 4.2 a 4.3.

U oboch vyššie spomenutých nástrojov, zabudované editory majú veľmi užitočné vlastnosti, ako je zabudované zvýraznenie a kontrola syntaxe, automatické doplňovanie kľúčových slov a definície prefixov. Tieto vylepšenia pomáhajú v urýchlení práce vývojára a zvyšujú jej efektívnosť.

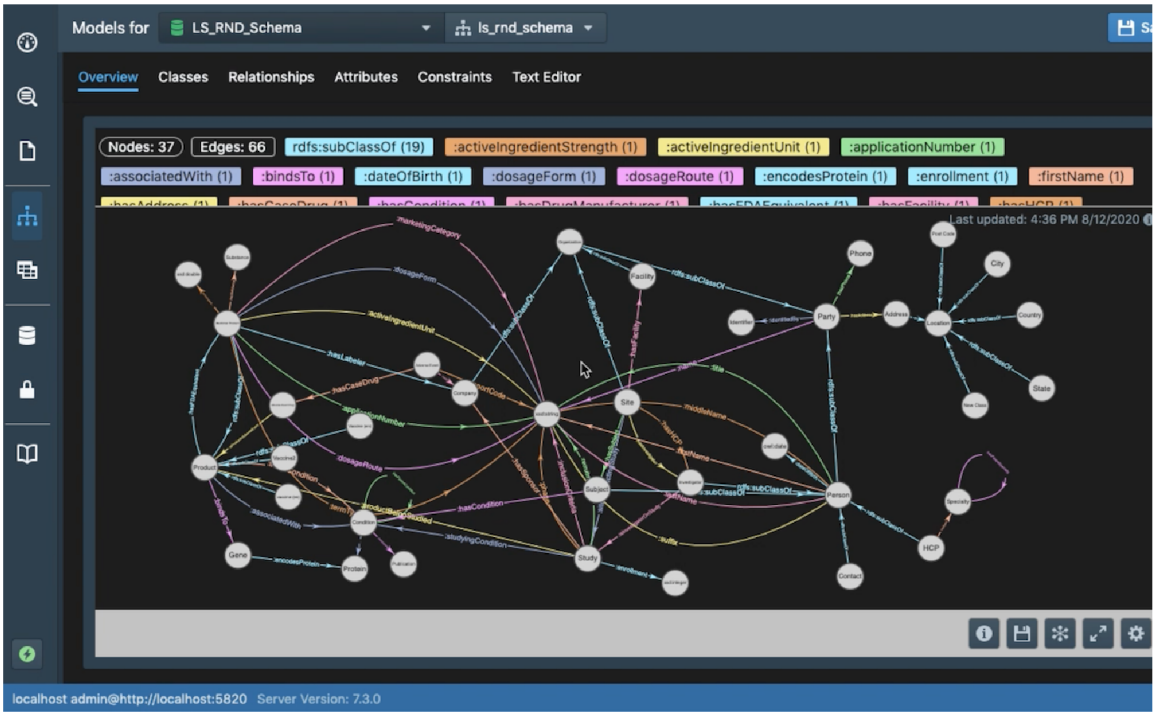
[1] https://www.stardog.com/studio/



Obr. 4.1: Ukážka vzhľadu zabudovaného editora pre vytváranie dopytov v nástroji RDF4J Workbench.



Obr. 4.2: Ukážka vzhľadu zabudovaného editora pre vytváranie dopytov v nástroji Stardog Studio.



Obr. 4.3: Ukážka vzhľadu vizualizácie RDF grafu v nástroji Stardog Studio.

4.2 Požiadavky týkajúce sa aplikácie

Pri návrhu aplikácie sa bral ohľad na samotné zadanie práce, na výsledky skúmania už existujúcich riešení a na potreby potencionálnych budúcich užívateľov aplikácie. Na základe týchto informácií sa rozhodlo, že na výsledné riešenie budú kladené nasledujúce funkčné požiadavky:

- Komunikácia so servermi spravujúcimi RDF databázy (sekcia 2.5).
- Zobrazenie a úprava obsahu RDF úložísk.
- Poskytnutie editora pre interaktívne vytváranie dopytov.
- Uľahčenie písania SPARQL dopytov (sekcia 2.6).
- Prehľadné zobrazenie výsledkov dopytov.
- Prehľad a editácia prefixov, vytvorených pre priestory mien.

Po konzultácií a dohode s vedúcim diplomovej práce sa rozhodlo, že vytváraný program bude napojený na už existujúci projekt s názvom *FitLayout*[2]. Tento projekt ponúka rozšíriteľný rámec pre segmentáciu a analýzu webových stránok. Táto vytváraná aplikácia bude komunikovať so serverom FitLayoutu, ktorý v rámci nej zabezpečuje ukladanie RDF dát. Preto, ako ďalšia požiadavka, bolo zabezpečenie kompatibility tohto programu.

[2] https://github.com/FitLayout

4.3 Vlastnosti navrhnutej aplikácie

Na základe zamerania, charakteru diplomovej práce a preskúmaných technológií, ktoré boli popísané v kapitolách zaoberajúcich sa teoretickou časťou práce (kapitola 3), sa rozhodlo, že aplikácia bude typu webovej aplikácie. Pri výbere vlastností bola snaha, čo najjednoduchšie a efektívne splniť stanovené a vyššie popísané požiadavky (sekcia 4.2).

Podrobnejší rozbor zvolených vlastností s odôvodnením ich výberu je uvedený nižšie :

Webová aplikácia

Tak, ako bolo spomenuté aj v kapitole 3, mnoho projektov sa v posledných rokoch realizuje v podobe webových aplikácií. Dôvodom je všade dostupné rýchle pripojenie k internetu a dobrá podpora zo strán vývojových prostriedkov a technológií.

Hlavným faktorom pre výber tejto formy aplikácie je teda jednoduchý prístup, a to buď cez lokálnu sieť alebo internet. Typ pripojenia závisí hlavne na umiestnení a vlastnostiach servera, s ktorým je potrebné ďalej spolupracovať. Potencionálny budúci užívateľ tak vie jednoducho využiť funkcionality ponúkané aplikáciou aj počas cestovania, za predpokladu dobrého sieťového spojenia.

Ďalším dôvodom pre používanie aplikácie bol nízky nárok na hardvérové vybavenie, bez nutnosti inštalácie dodatočných softvérových vybavení. Toto riešenie prináša veľkú flexibilitu, týkajúcu sa použitého technologického vybavenia. Ako klient, postačuje niektorý zo širokej škály existujúcich webových prehliadačov, ktorý je schopný zobrazovať jednoduché stránky a má v sebe zabudovaný softvérový komponent, vykonávajúci javascriptový kód (anglicky *engine*).

Očakávaný typ odpovedi

Zo skúmaných existujúcich riešení, najviac serverov bolo schopných na prijatých dopytoch vrátiť svoje odpovede buď v podobe jednoduchého textu (*Content-Type: text/plain*), XML dát (*Content-Type: application/sparql-results+xml*) alebo vo formáte JavaScript Object Notation (skratkou *JSON*).

Z týchto uvedených typov sa najjednoduchšie pracuje v JavaScripte s objektami formátu JSON, ktorý, ako aj názov naznačuje, bol sám odvodený z tohto jazyka. Vďaka tomu nevyžaduje žiadne ďalšie prostriedky pre jeho správne parsovanie v jazyku JS.

Na základe dohody s vedúcim práce a toho, že aplikácia bude implementovaná s využitím javascriptového rámca sa rozhodlo podporovať spoluprácu iba so servermi, ktoré sú schopné odpovedať na dopyty s obsahom vo formáte JSON. Tento predpoklad spĺňuje aj FitLayout server.

Interaktívne vytváranie dopytov

Textové editory sú neoddeliteľnou súčasťou každodennej práce programátora alebo užívateľa, ktorý musí v určitej podobe napísať zdrojový kód. V dnešnej dobe sa stali dôležitým komponentom vývojových prostredí (anglicky *Integrated Development Environment*, skratka IDE). Sú doplnené o funkcionality, ktoré značne urýchľujú, zjednodušujú písanie kódu a zlepšujú jeho prehľadnosť. Napríklad takýmito možnosťami sú farebné odlíšenie kľúčových slov od bežného textu, kontrola správneho používania zátvoriek, automatická nápoveď ďalších slov a ich doplňovanie.

Editor zastupuje centrálnu úlohu aj v navrhovanej aplikácii. Pomocou neho bude zabezpečené požadované interaktívne dopytovanie nad RDF dátami. Na základe zadania bude podporovať vytváranie príkazov v jazyku SPARQL.

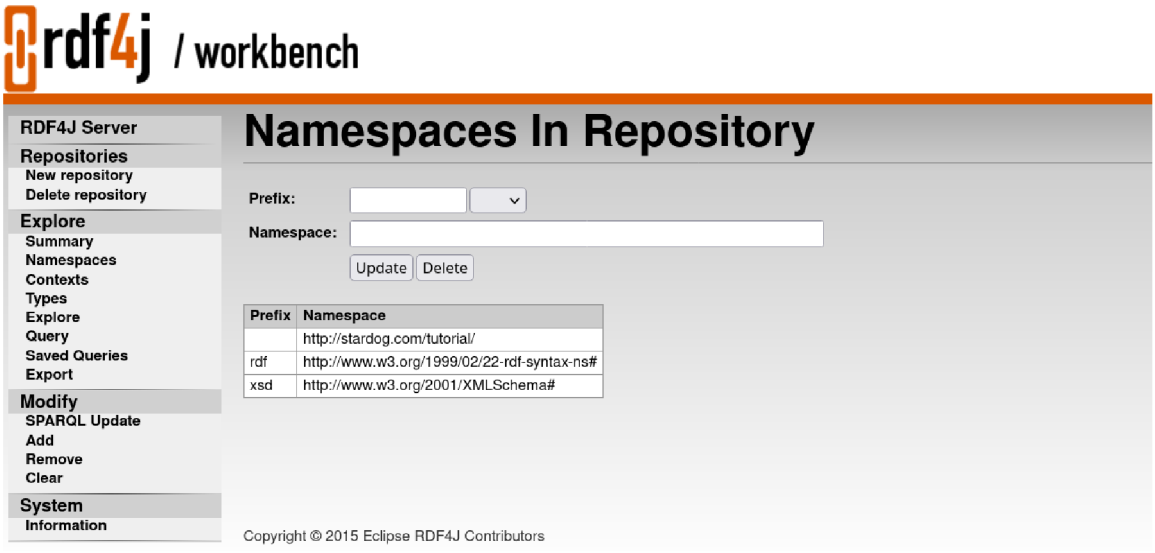
Aby sa uľahčila práca budúceho užívateľa, tak editor bude vybavený možnosťami, ako je napríklad zabudované zvýraznenie syntaxe (anglicky *syntax highlighting*) a kontrola syntaktickej správnosti napísaného dopytu. Užívateľovi budú v editore počas kontroly vyznačené miesta, kde sa nachádzajú chyby, spolu s nápoveďou možnej príčiny a jej riešením.

V jazyku SPARQL, popísané v sekcii 2.6, je možné definovať prefixy pre jednotlivé priestory mien, aby v dopytoch nebolo potrebné uvádzať ich celé URI. Rozhodlo sa odbremeniť užívateľa od nutnosti zapisovať definície týchto prefixov pri každom jednom vytvorenom dopyte a editor bol rozšírený o ďalšiu funkciu. Touto funkciou je automatické rozpoznávanie prefixov v texte dopytu a doplnenie ich definícií do textu.

Spravovanie prefixov a priestorov mien

Prefixy, iným slovom predpony, ako už bolo popísané v sekcii 2.3, sa používajú na to, aby sa dalo predísť opätovnému výpisu dlhých URI identifikátorov menných priestorov v textoch. Tvoria dôležitú súčasť RDF dát a práce s nimi. Preto v RDF úložiskách sa ponúka možnosť vytvárania nových menných priestorov spolu s príslušnými predponami. Príklad je možné vidieť na obrázku 4.4.

Vyvíjaná aplikácia, na základe návrhu, bude obsahovať vyhradenú časť iba pre prácu s priestormi mien nachádzajúcimi sa v úložisku, ktorú bude aktuálne užívateľ preskúmavať. Medzi jeho možnosťami bude zadávanie nových menných priestorov s prefixmi, úprava už existujúcich alebo ich prípadne odstránenie.



Obr. 4.4: Ukážka možnosti definície prefixov a menných priestorov v nástroji RDF4J Workbench.

Užívateľské rozhranie

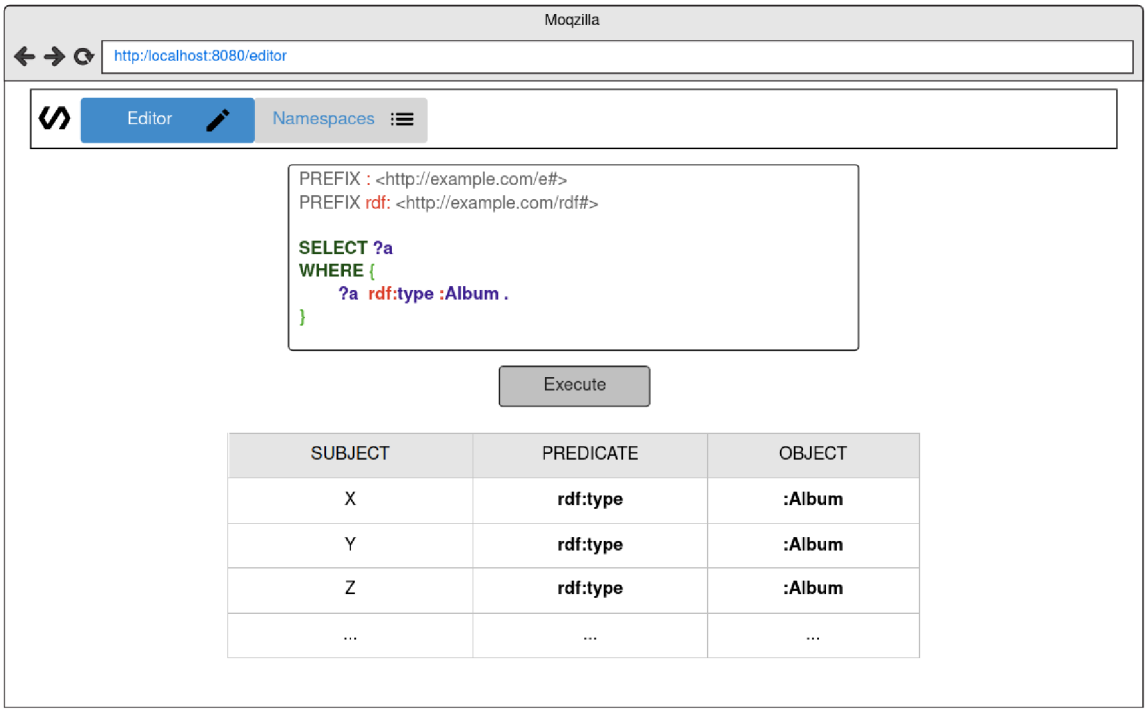
Dôležitou súčasťou každej webovej aplikácie je jej užívateľské rozhranie, najmä miera jej intuitívnosti. Ak rozloženie a obsah stránok je dobre navrhnutý, tak to vie značne urýchliť a znížiť čas potrebný na prevedenie krokov jednotlivých úloh. Schopnosť jednoduchého zorientovania sa v rozhraní má kladný vplyv na užívateľa, ktorý vykonáva potrebnú činnosť a motivuje ho, aby ďalej používal danú aplikáciu a nerozhodol sa pre hľadanie inej alternatívy.

Pri návrhu vzhľadu aplikácie bol zámerne zvolený minimalizmus pre lepšiu prehľadnosť obsahu. Cieľom bolo, aby jednotlivé komponenty zobrazené na stránke obsahovali iba nevyhnutné tlačidlá alebo formuláre. Tento spôsob reprezentácie obsahu stránky menej rozptyľuje samotného užívateľa, ktorý vďaka tomu sa vie lepšie koncentrovať na vykonávanú prácu.

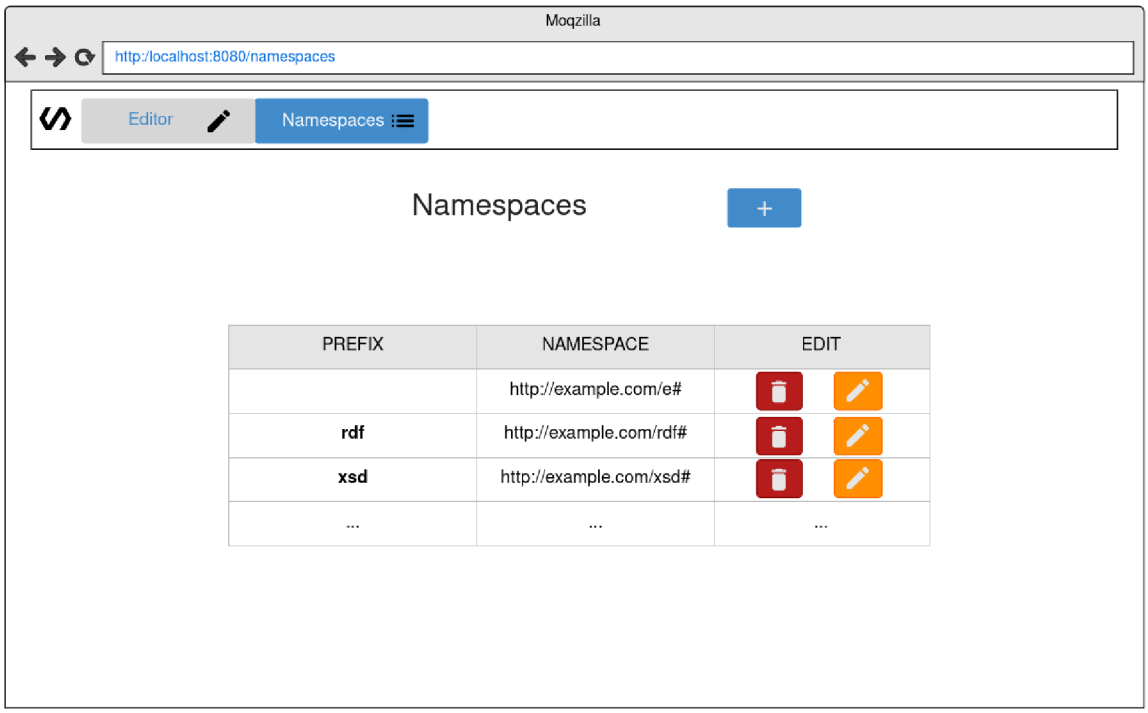
Vzhľadom na to, že sa jedná o webovú aplikáciu, tak je potrebné počítať aj s tým, že sa dá k nej pristupovať z rôznych zariadení s rozličnou veľkosťou zobrazovacích plôch. Počas vytvárania komponentov, tvoriacich rozhranie, je potrebné zabezpečiť ich responzívnosť, aby podľa možností prispôsobovali svoj vzhľad prítomným rozmerom a voľným plochám na stránke.

Na obrázku 4.5 je znázornený návrh rozloženia prvkov na stránke, ktorá je zodpovedná za zadávanie SPARQL dopytov pomocou editora a následného zobrazenia výsledkov v podobe prehľadnej tabuľky. Pri zobrazení výsledkov je potrebné myslieť aj na to, že SPARQL dopyty môžu mať aj iný výsledok, ako je množina trojíc. Napríklad v prípade ASK je to pravdivostná hodnota. Aj tieto typy výsledkov je potrebné správne zobraziť.

Stránku zodpovednú za manipuláciu s priestormi mien je možné vidieť na obrázku 4.6.



Obr. 4.5: Mock-up časti plánovanej aplikácie, ktorá bude obsahovať editor pre interaktívne zadávanie dopytov. Po úspešnom prevedení dopytu sa výsledky zobrazia v podobe tabuľky rozdelenej na tri stĺpce, zodpovedajúce jednotlivým častiam výsledných trojíc.



Obr. 4.6: Mock-up časti plánovanej aplikácie, ktorá bude zodpovedná za správu prefixov a menných priestorov v úložiskách RDF dát. Okrem možnosti vytvorenia nových takýchto entít bude u každej už existujúcej entite možnosť jej úpravy alebo zmazania.

4.4 Architektúra aplikácie

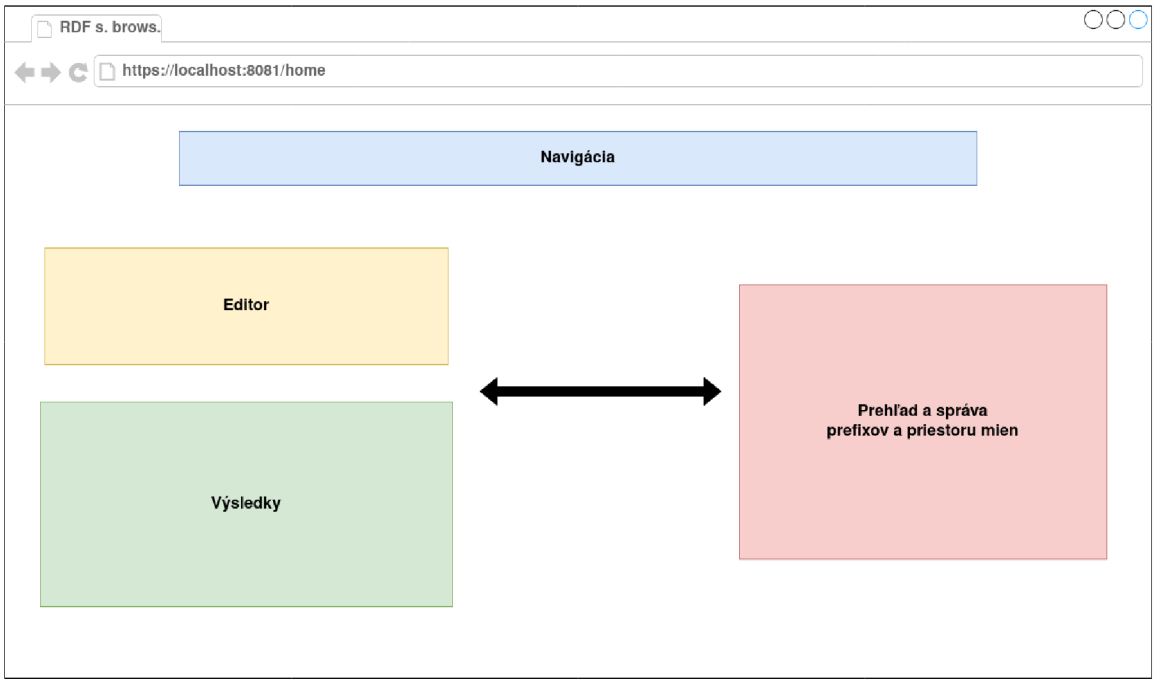
Aplikácia vychádza z dobre známej architektúry jednostránkovej webovej aplikácie (anglicky *Single-Page Application*, skratkou SPA), ktorej obsah je tvorený iba jedinou stránkou. Tento prístup k tvorbe aplikácií prináša kladné vlastnosti, ako je rýchlejšia a užívateľsky prívetivejšia manipuláciu so stránkou a jej obsahom, reakcia na požiadavky v reálnom čase.

V SPA k obnoveniu stránky nikdy nedochádza. Namiesto toho všetok potrebný kód, ako je HTML, JavaScript a CSS, prehliadač získa buď prvým načítaním stránky, alebo príslušné zdroje sa načítajú dynamicky podľa potreby a pridávajú sa na stránku, zvyčajne v reakcii na akcie používateľa.

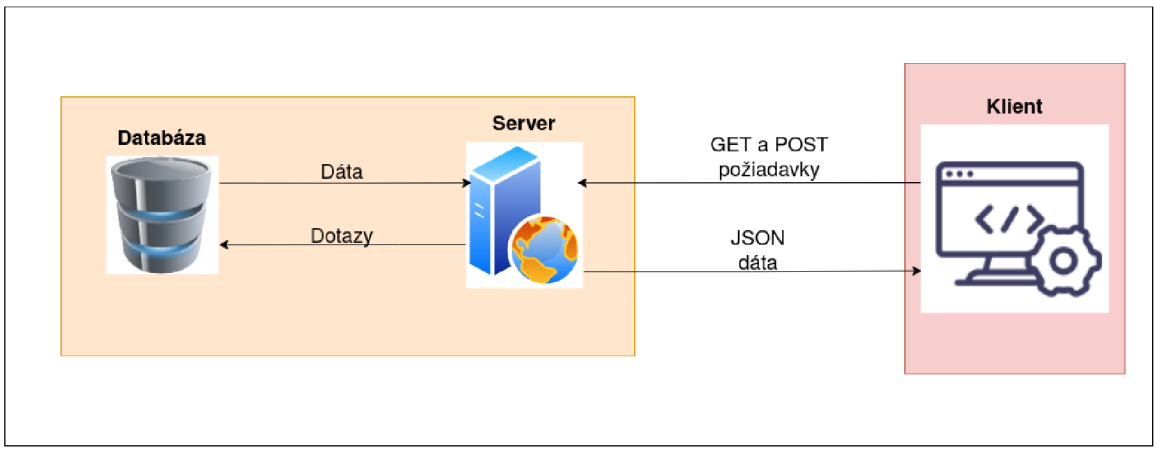
Napriek tomu, že aplikácia pozostáva iba z jedinej stránky, ponúkané funkcie je možné rozdeliť do menších logických stránok, medzi ktorými sa dá navigovať, napríklad na základe určitej časti URL.

Jednotlivým možnostiam ponúkaným užívateľovi, ktoré boli popísané v sekcií 4.3, budú priradené samostatné komponenty. Tieto komponenty budú implementovať ich fungovanie a určovať ich vzhľad. Meniť aktuálne zobrazované časti stránky bude možné pomocou navigácie medzi komponentmi. Príklad striedania sa komponent je možné vidieť na obrázku 4.7.

Komunikácia medzi klientom a príslušným serverom, zodpovedným za správu úložiska RDF dát, bude prebiehať pomocou dopytov typu GET a POST. Klientska aplikácia bude očakávať všetky odpovede vo formáte JSON.



Obr. 4.7: Znázornenie návrhu striedania sa komponentov na stránke podľa používanej funkcionality. V rámci aplikácie je iba jediná stránka, na ktorej sa vždy iba menia aktuálne zobrazené komponenty.



Obr. 4.8: Architektúra vytváranej aplikácie. V rámci projektu sa implementuje klient (vyznačený červenou). Server a databáza sú uvedené pre úplnosť, pre lepšie pochopenie fungovania aplikácie.

Kapitola 5

Implementácia rozhrania pre prechádzanie RDF úložiska

V tejto kapitole je uvedené postupné pretváranie vytvoreného návrhu, popísaného v kapitole 4, na výslednú reálne použiteľnú aplikáciu. Pri popise implementácie nebolo cieľom podrobne predstaviť všetky detaily funkcií a samotného konštruovania programu, iba vyzdvihnúť tie časti programu, ktoré môžu byť zaujímavé pre čitateľa.

V sekcii 5.1 sú predstavené niektoré najpodstatnejšie technológie, ktoré boli použité pri realizácii aplikácie. Stručne sú uvedené základné informácie a zdôvodnenie toho, prečo boli zvolené. Po preskúmaní existujúcich knižníc a rámcov, určených na tvorbu webových aplikácií, padla voľba na rámec Vue.js. Bol zvolený ako hlavná implementačná platforma.

Implementáciou očakávaných funkcionalít aplikácie sa zaoberá sekcia 5.2. Obsahuje popis štruktúry a tvorby samotných komponentov a stránok, spolu s ich prepojením do jedného celku.

Na záver v sekcií 5.3 sú predstavené budúce možné rozšírenia aplikácie na zvýšenie jej úrovne.

5.1 Použité technológie

Pre zjednodušenie tvorby komponentov boli zvolené a využité nasledujúce technológie, spĺňajúc zadaním stanovené požiadavky:

Vue Prism Editor a Prism.js

Na internete je možné nájsť mnoho knižníc, implementovaných editorov kódu a textu, založených na prehliadači, napr. Ace, CodeMirror, Monaco alebo Trix, ktoré sú zabudovateľné do webových stránok. Na základne tohto faktu sme sa rozhodli nevytvárať vlastný editor. Efektívnejšie bolo použiť niektorý z už existujúcich, s prípadným jeho prispôsobením podľa vlastných potrieb.

Voľba padla na knižnicu *Vue Prism Editor*[1], ktorá ponúka jednoduchý editor kódu so zvýraznením syntaxe a číslovaním riadkov. Vyššie spomenuté príklady často môžu byť prehnaným riešením. Ich mnohé zabudované funkcionality sú zbytočné v prípadoch, keď je potrebný iba kompaktný jednoúčelový editor. Silná stránka zvolenej knižnice sa ukáže práve v takýchto situáciách, kde je potrebná kompaktnosť, malý rozmer a jednoduchosť. Ideálna

[1] https://github.com/koca/vue-prism-editor/tree/feature/next

je pre tvorbu malých formulárov, z ktorých môžu používatelia odoslať kód na server. Medzi jej zabudované funkcionality patria:

- Modulárne zvýrazňovanie syntaxe pomocou knižnice tretích strán
- Automatické odsadenie na nových riadkoch
- Vrátenie späť / Opakovanie celých slov pomocou klávesových skratiek
- Automatické prispôsobenie sa k veľkosti stránky
- Číslovanie riadkov

Pre fungovanie zvýrazňovania syntaxe je potrebné zaistiť knižnice tretích strán, ktoré budú túto vlastnosť zabezpečovať. V rámci projektu bol pre tento účel využitý *Prism.js*[2]. Jedná sa o ľahký, rozšíriteľný zvýrazňovač syntaxe, vytvorený s ohľadom na moderné webové štandardy. Podporuje viac než 270 jazykov, medzi ktoré patrí aj SPARQL, s možnosťou rozšírenia o vlastné jazyky.

SPARQL.js

Užitočnou funkciou vývojových prostredí a editorov kódov je poukázanie na chyby v užívateľom vytvorených blokoch kódov. Pre uľahčenie práce užívateľa sa často táto funkcionalita doplňuje aj s odporučením možných príčin problémov a ich riešením.

SPARQL.js[3] je knižnica, ktorá je schopná prekladať dopyty vytvorené v SPARQL do JSON formátu a späť. Vďaka tomu sa dajú jednoduchšie vytvárať, modifikovať a analyzovať dopyty, vytvorené v tomto jazyku, v javascriptových aplikáciách.

Knižnica je vybavená veľmi dobrým zabudovaným syntaktickým analyzátorom. Tento analyzátor uľahčuje určovanie možných riešení chýb vďaka tomu, že vráti nielen presné číslo riadku, na ktorom sa vyskytuje chyba, ale aj potrebné modifikácie dopytu, pre odstránenie chyby.

N3.js

N3.js[4] je knižnica implementujúca nízkoúrovňovú špecifikáciu stanovenú v RDF.js[5]. Umožňuje tak pracovať s RDF dátami veľmi jednoducho a intuitívne v JavaScripte.

Medzi funkcionality, ponúkané knižnicou, patria:

- Analýza a načítanie trojíc vo formáte Turtle, N3, N-Triples
- Zápis dát pomocou trojíc vo formáte Turtle, N3, N-Triples
- Ukladanie trojíc do hlavnej pamäti

Spomenuté operácie sú asynchrónne, rýchle a zvládajú aj spracovanie väčších prúdov dát alebo dokumentov.

[2] https://prismjs.com/
[3] https://github.com/RubenVerborgh/SPARQL.js/
[4] https://github.com/rdfjs/N3.js/
[5] http://rdf.js.org/#

PrimeTek a PrimeVue

Tvorba komponentov, ich vzhľadu a užívateľského rozhrania potencionálne zaberie veľké množstvo času. Zložitosť tejto úlohy sa úmerne zvyšuje s komplexnosťou cieľového rozhrania aplikácie. Aby sa dal využiť tento čas inak a venovať pozornosť iným implementačným činnostiam, ako je napr. tvorba vrstvy operujúcej so samotnými dátami (back end), boli vytvorené a zverejnené knižnice komponentov. Tieto knižnice obsahujú znovu použiteľné komponenty, tvoriace časti používateľského rozhrania, s vopred definovaným vzhľadom a chovaním. V mnohých prípadoch sa dajú tieto vlastnosti prispôsobiť aktuálnym potrebám programátorov. V úlohe takéhoto komponentu môže stáť aj napríklad jednoduché tlačidlo, posúvač (anglicky *slider*), ale aj celý formulár.

PrimeTek je jedna z vedúcich poskytovateľov UI knižníc. Ich knižnice obsahujú elegantné, vysokovýkonné, bezplatné a plne prispôsobiteľné komponenty užívateľského rozhrania. Sú využiteľné v rámcoch ako sú: JavaServer Faces (*PrimeFaces*), Angular (*PrimeNG*), React (*PrimeReact*) a najnovšie už aj vo Vue.js (*PrimeVue*).

V rámci diplomovej práce sa využíva knižnica **PrimeVue**[6], ktorá je sadou natívnych komponentov s otvoreným zdrojovým kódom (open-source). Ďalej v sebe zahrňuje rôzne alternatívy základných tém (napr. Material, Bootstrap, ...) a celých šablón, ako aj návrhár umožňujúci návrh vlastných tém.

5.2 Implementácia navrhnutých funkcionalít

Ako už bolo spomenuté, pre implementáciu bol zvolený rámec Vue.js a to konkrétne jeho najnovšia verzia 3.1. Aplikácia a samotné užívateľské rozhranie bolo rozdelené tak, aby každý príslušný komponent mal na starosti iba malý počet, ideálne iba jedinú z funkčných požiadaviek. Pri jednotlivých popisoch častí programu sú uvedené aj ukážky ich reálneho vzhľadu.

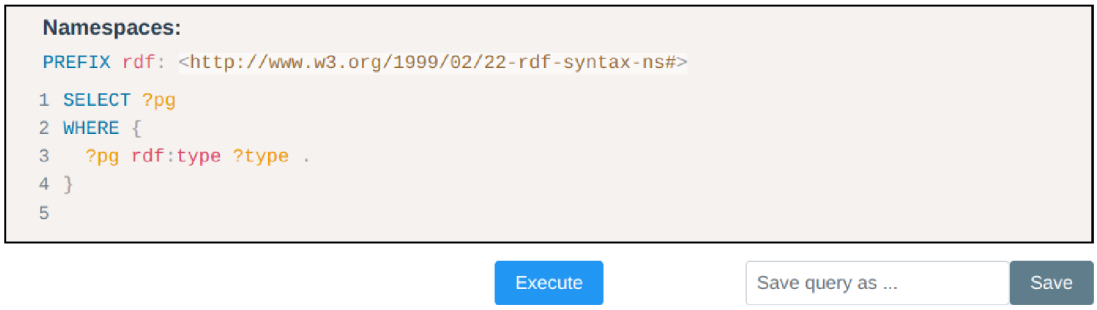
Aby mal užívateľ kompletný prehľad o aktuálnych udalostiach v rámci programu, je informovaný o výsledku každej prevedenej operácie. Informovanie prebieha pomocou malých upozornení, so stručným textom a farbou symbolizujúcou závažnosť informácie. Tieto upozornenia napodobňujú dobre známe *push* notifikácie z mobilných zariadení. Správnosť vyplnenia vstupných textových polí prebieha na strane klienta, s prípadným upozornením užívateľa na nesprávny formát vstupu. Informácie potrebné pre bezchybné fungovanie programu, ako je napríklad názov zvoleného úložiska alebo adresa servera, sa získavajú z častí URL a konfiguračného súboru *config.js*.

Editor pre vytváranie dopytov

Editor pre vytváranie dopytov zohráva centrálnu úlohu v poskytovaných funkcionalitách, preto je umiestnený na hlavnej, úvodnej stránke vytvorenej aplikácie. Bol vytvorený, ako samostatný komponent s názvom **RdfEditor** a je zobrazený na obrázku 5.1.

Ako už bolo spomenuté v sekcií 5.1, pri jeho tvorbe sa rozhodlo vychádzať z už existujúceho riešenia *Vue Prism Editor*, s dodatočným prispôsobením malých detailov. Potrebný zvýrazňovač syntaxe bol dodaný v podobe knižnice *Prism.js*. Farebná téma zvýraznenia bola vybraná v podobe základnej varianty pre jazyk SPARQL. Farebné kombinácie, ktoré obsahuje, dostatočne odlišujú kľúčové slová a znaky od bežných slov. Zvýrazňovacia trieda

[6] https://www.primefaces.org/primevue/#/



Obr. 5.1: Editor vytvorenej aplikácie. Na obrázku je možné vidieť spôsob zvýraznenia kľúčových slov. Pod nadpisom „Namespaces:“ je umiestnená automaticky doplnená definícia prefixu *rdf* pre menný priestor *http://www.w3.org/1999/02/22-rdf-syntax-ns#*.

užívateľom napísaného textu vytvára príslušnú HTML reprezentáciu zobraziteľnú v tele editora.

Aby nebolo nutné pri každom novom dopyte vytvárať znova definície prefixov pre jednotlivé priestory mien, bola zabezpečená automatizácia tejto činnosti. Vkladanie definícii je prevedené automaticky editorom počas písania textu dopytov. Po počiatočnom namontovaní komponentu sa získajú definície všetkých menných priestorov prítomných v aktuálne preskúmavanom úložisku. Pri každej zmene obsahu textového poľa sa prevedie hľadanie prefixov v tele dopytu. Ich rozpoznávanie je zabezpečené pomocou príslušných javascriptových regulárnych výrazov. Pre každý rozpoznaný prefix sa vytvorí príslušný kód, ktorý sa umiestni do hornej časti editora. Definícia je automaticky zmazaná pri zmazaní prefixu z textu.

Po stlačení modrého tlačidla **Execute** je otázka považovaná za hotovú a následne sú spustené ďalšie funkcie. Tieto funkcie zabezpečia to, aby text mal správny formát, neobsahoval chyby a bol preveditelný na strane servera. Základnú kontrolu predstavuje overenie syntaktickej správnosti dopytu. Táto činnosť je zabezpečená zabudovaným parserom knižnice *SPARQL.js*, ktorý bol predstavený v sekcií 5.1.

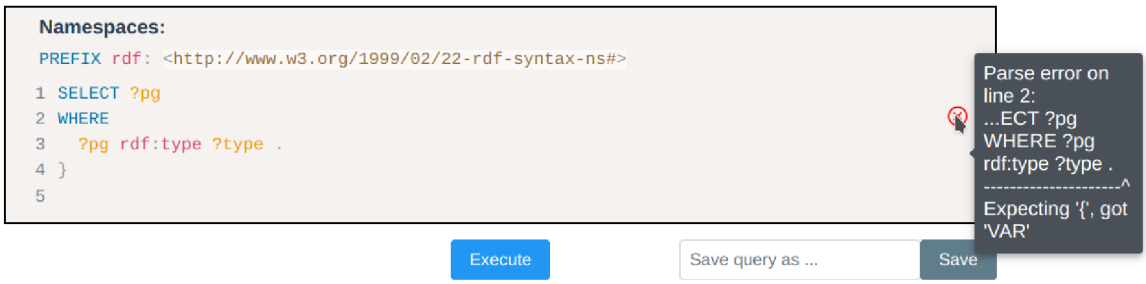
Ak sa v dopyte vyskytla chyba, tak parser vráti o nej základné informácie, ako sú: číslo riadku, podstata chyby a jej možné riešenia. Užívateľ je informovaný o nájdenom probléme pomocou červenej ikonky, umiestnenej na príslušných riadkoch. Bližšie informácie sa mu ukážu až po umiestnení kurzora myši nad ikonkou. Ukážku o spôsobe informovania je možné vidieť na obrázku 5.2.

Správne napísané SPARQL dopyty sú následne odoslané na vopred určený koncový bod servera, využitím POST žiadosti.

Opätovné využívanie vytvorených dopytov

Funkcionalita opätovného využívania vytvorených dopytov odstraňuje nutnosť znovu vytvárať často prevedené dopyty užívateľa. Bola to praktická vec využitá aj pri samotnej implementácii aplikácie, keď bolo potrebné často reštartovať program. Reštart spôsobil stratu obsahu editora a vynútil monotónne opakovanie rovnakých krokov. Vďaka uloženým dopytom bolo možné sa v priebehu pár sekúnd vrátiť späť k predchádzajúcemu stavu.

Možnosť ukladania a opätovného využívania už existujúcich dopytov je zabezpečené komponentom **SavedQueries**. V rámci neho sa využíva a pracuje s jedným typom webo-



Obr. 5.2: Na obrázku je možné vidieť spôsob informovania užívateľa o chybe, ktorá sa vyskytla vo vytvorenom dopyte. Okrem vyznačenia riadku, na ktorom sa chyba vyskytuje, užívateľ dostane v popise stručnú rekapituláciu problému. Pri danom problému sa uvádza vždy aj jeho možné riešenie.

vého úložiska (anglicky *web storage*). Webové úložisko umožňuje programátorom ukladať informácie priamo v prehliadači na strane klienta [33]. Uložené dáta majú formát dvojice, kľúč a hodnota. Na základe doby životnosti týchto dát rozlišujeme dva typy tohto úložiska. Prvým z nich je lokálne úložisko (anglicky *local storage*), u ktorého informácie pretrvajú aj po zatvorení a opätovnom otvorení prehliadača. Druhým je úložisko relácie (anglicky *session storage*), kde sú dáta k dispozícii iba počas trvania relácie stránky. Na základe charakteru ukladaných dát, u ktorých je cieľom dosiahnuť znovu použiteľnosť pri každom spustení programu, bol zvolený prvý typ úložiska.

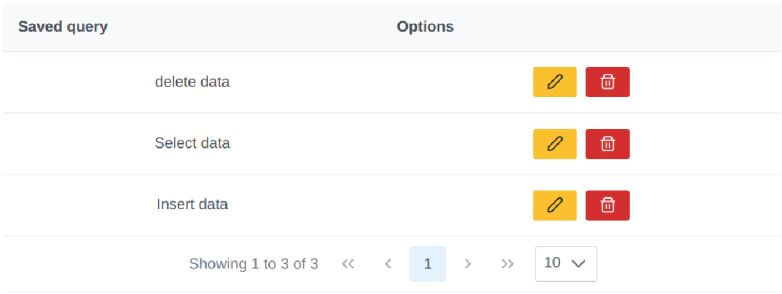
Pre nahranie novovytvoreného dopytu do lokálneho úložiska stačí užívateľovi previesť dve jednoduché kroky. Vo vyššie popísanom editore je potrebné vyplniť názov, pod ktorým bude dopyt zaregistrovaný a potom kliknúť na sivé tlačidlo **Save** (slovensky *uložiť*).

Obr. 5.3: Na obrázku je vyznačené červeným obdĺžnikom časť pod editorom, ktorá obsahuje miesto pre vyplnenie názvu dopytu a sivé tlačidlo **Save** pre potvrdenie uloženia dopytu.

O spravovanie všetkých aktuálne uložených dopytov sa stará komponent *SavedQueries*. Jeho úlohou je: zobrazenie, editácia a zmazanie. V rámci komponentu sú názvy dopytov z lokálneho úložiska uvedené v jednoduchej tabuľke spolu s dvojicou tlačidiel. Ukážka tabuľky sa nachádza na obrázku 5.4

Modifikáciu dopytu je možné previesť pomocou prvého žltého tlačidla s ikonou ceruzky. Po jeho stlačení je užívateľ presmerovaný späť na komponent *RdfEditor*, v ktorom sa zobrazí telo dopytu. Prevedené zmeny si užívateľ uloží pomocou rovnakého postupu, ako pri nahrávaní nového dopytu. Definitívne zmazanie dopytu je prevediteľné použitím červeného

tlačidla s ikonou smetného koša. Po jeho použití je užívateľ vyzvaný, aby potvrdil vykonanie plánovanej akcie.



Obr. 5.4: Na obrázku je možné vidieť komponent *SavedQueries*. Všetky aktuálne uložené dopyty sú zobrazené v jednoduchej tabuľke. Na každom riadku vedľa názvu dopytu sa nachádzajú dve tlačidlá. Prvé žlté tlačidlo slúži pre modifikáciu tela dopytu. Pomocou druhého červeného tlačidla je možné previesť definitívne zmazanie zvoleného dopytu.

Zobrazenie výsledkov a preskúmavanie zdrojov

Zobrazenie výsledkov, užívateľom prevedených dopytov, je úlohou komponentu **QueryResults**. Odpovede od servera prichádzajú vo formáte JSON, N-Triples alebo pravdivostnej hodnoty. Pri ich spracovávaní sa berie ohľad na to, aký typ SPARQL dopytu bol prevedený. V prípade dát vo formáte N-Triples sa používa knižnica *N3.js* na ich parsovanie a prevedenie do vhodnej internej reprezentácie v kóde.

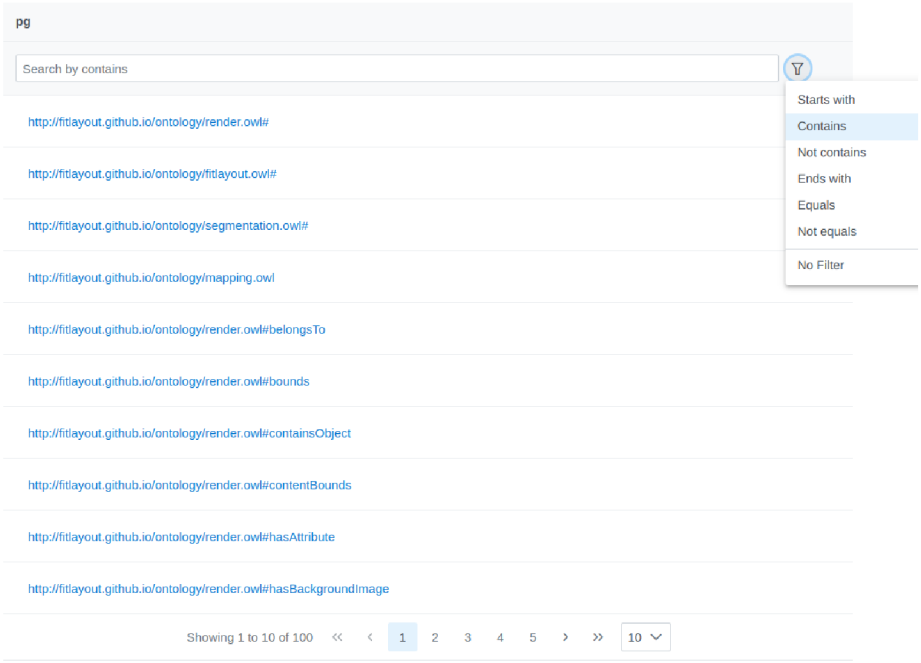
Výsledky *SELECT* a *CONSTRUCT* dopytov sú umiestnené do prehľadnej tabuľky. Táto tabuľka je tvorená použitím PirmeVue komponentu *DataTable*. Príklad takejto tabuľky je znázornený na obrázku 5.5.

Jednotlivé stĺpce predstavujú voľné premenné zo základu dopytov, ktorých názvy sú uvedené v ich hlavičkách. V stĺpcoch sú vymenované všetky možné hodnoty týchto premenných. Aby sa dalo jednoduchšie preskúmavať tieto hodnoty, tak pri každom z týchto stĺpcov je poskytnuté vyhľadávanie a filtrovanie podľa rôznych podmienok.

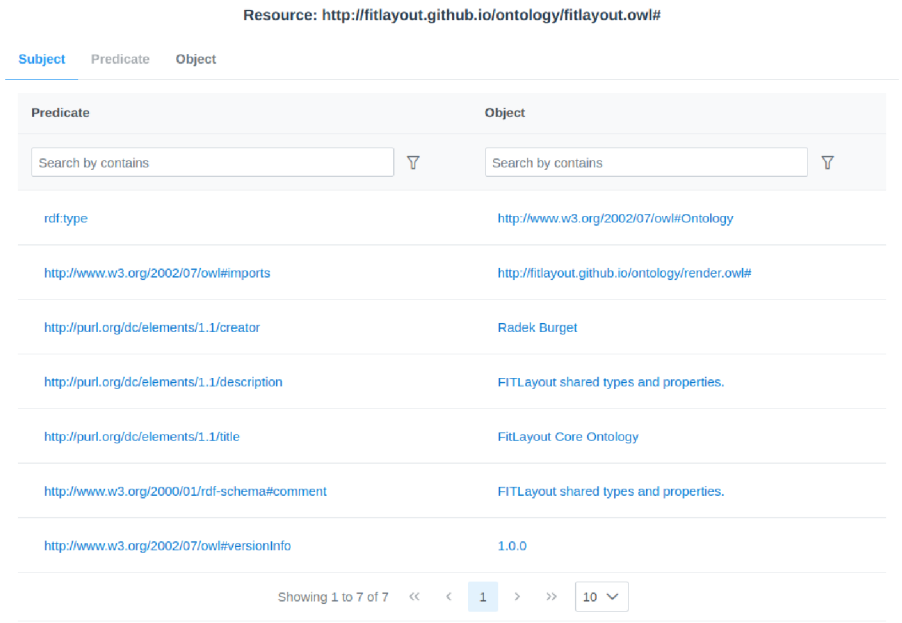
Výsledky *ASK* a *UPDATE* dopytov sú užívateľom zobrazené v podobe informatívnych textov a výstižných upozornení.

Aby užívateľ mal lepší prehľad, presnejšie informácie o dátach umiestnených v rámci zvoleného úložiska, tak mu je zabezpečená možnosť preskúmavania jednotlivých zdrojov. Vďaka tejto možnosti získa predstavu o vzťahoch medzi dátami.

Táto funkcia spočíva v tom, že pre zvolený zdroj sú zistené všetky trojice (anglicky *triples*), ktoré sa vyskytujú v danom repozitári. Po kliknutí na ktorúkoľvek z hodnôt v tele stĺpci sa prevedie výmena komponentu *QueryResults* na *ResourceExplorer*. Pri jeho počiatočnom namontovaní sa pošle nová POST žiadosť (anglicky *request*) na server. Táto žiadosť v sebe nesie SPARQL dopyt, zisťujúci príslušné trojice a jej výsledok sa spracuje do panelu kariet (PrimeVue komponenta *TabView*). Príklad tohto panelu je možné vidieť na obrázku 5.6. Dáta sú ďalej rozdelené do troch kategórií podľa toho, aké úlohy práve zastáva zvolený zdroj v jednotlivých trojiciach. Tieto kategórie sú: subjekt, predikát, objekt. Ku každej z nich je priradená vlastná karta v rámci hlavného panelu. Názov karty udáva vždy aktuálnu úlohu zvolenej položky a v stĺpcoch sú umiestnené zvyšné časti trojíc. V preskúmavaní zdrojov sa dá pokračovať kliknutím na ďalšie položky.



Obr. 5.5: Na obrázku je možné vidieť komponent *QueryResults*. Názov stĺpca tabuľky je rovnaký, ako názov voľnej premennej použitej v dopyte. V stĺpci sú vymenované všetky možné hodnoty, ktoré môže premenná nadobúdať. Pre ich jednoduchšie preskúmanie je poskytnutá možnosť vyhľadávania a filtrovania obsahu podľa rôznych podmienok.



Obr. 5.6: Na obrázku je možné vidieť komponent *ResourceExplorer*. Panel, ktorý ju tvorí, je rozdelený do troch kariet. Každá z nich reprezentuje jednu úlohu, v ktorej sa môže vyskytovať zvolená položka v trojiciach. Stĺpce tabuliek na kartách vždy doplňujú ostatné dve kategórie. V telách stĺpcov sú umiestnené zvyšné časti tvoriace spoločne celé trojice.

Spravovanie prefixov a priestorov mien

Na spravovanie prefixov a priestorov mien bol vyčlenený samostatný komponent s názvom **NamespacesMaintenance**. Rovnako, ako už v prípade vyššie popísaných komponentov, pri prvotnom namontovaní komponenty sa prevedie dopyt, získavajúci všetky menné priestory v danom repozitári. Následne sa tieto dáta zobrazia v jednoduchej tabuľke.

Každý riadok tabuľky obsahuje jeden prefix, priestor mien, ktorý reprezentuje, a dve tlačidlá. Prvé žlté tlačidlo s ikonou ceruzky otvára editačný formulár, v ktorom je možné prepísať názov zvoleného priestoru mien. Červené tlačidlo, druhé v poradí, slúži pre odstránenie zvoleného prefixu. Pred samotným zmazaním je užívateľ vyzvaný na to, aby potvrdil túto nevratnú operáciu. Priestor mien, ku ktorému bol daný prefix priradený, je tiež odstránený.

Priradiť nové predpony k menným priestorom sa dá pomocou modrého tlačidla. Toto tlačidlo je umiestnené nad samotnou tabuľkou a je označené ikonou znaku plus. Po jeho použití sa zobrazí užívateľovi formulár so vstupnými poľami pre zadanie predpony a názvu menného priestoru.

Pri vytváraní nového prefixu, aj pri samotnej modifikácií, je prevedená kontrola vstupných textových polí. Kontroluje sa napríklad správny formát prefixu, či názov menného priestoru obsahuje prefix *http://* alebo *https://* a tak ďalej.

Vzhľad a štruktúru komponentu *NamespacesMaintenance* vidieť na obrázku 5.7

Obr. 5.7: Obrázok znázorňuje komponent *NamespacesMaintenance*. Jeho hlavná časť je tvorená trojstĺpcovou tabuľkou. Na každom riadku v poradí zľava doprava sú uvedené: názov prefixu, názov priestoru mien, ku ktorému je priradená a dve tlačidla. Prvé, žlté tlačidlo slúži na modifikáciu názvu menného priestoru a druhé, červené tlačidlo na zmazanie prefixu. Novú predponu je možné k priestoru mien priradiť pomocou modrého tlačidla umiestneného nad tabuľkou.

Vkladanie nových RDF dát

Keď užívateľ vytvára nové úložisko a chce ho naplniť novými dátami alebo chce rozšíriť už existujúce o nové dáta, tak potrebuje mať na to miesto, kde to vie zrealizovať. V rámci projektu toto miesto zastáva komponent **InsertData**.

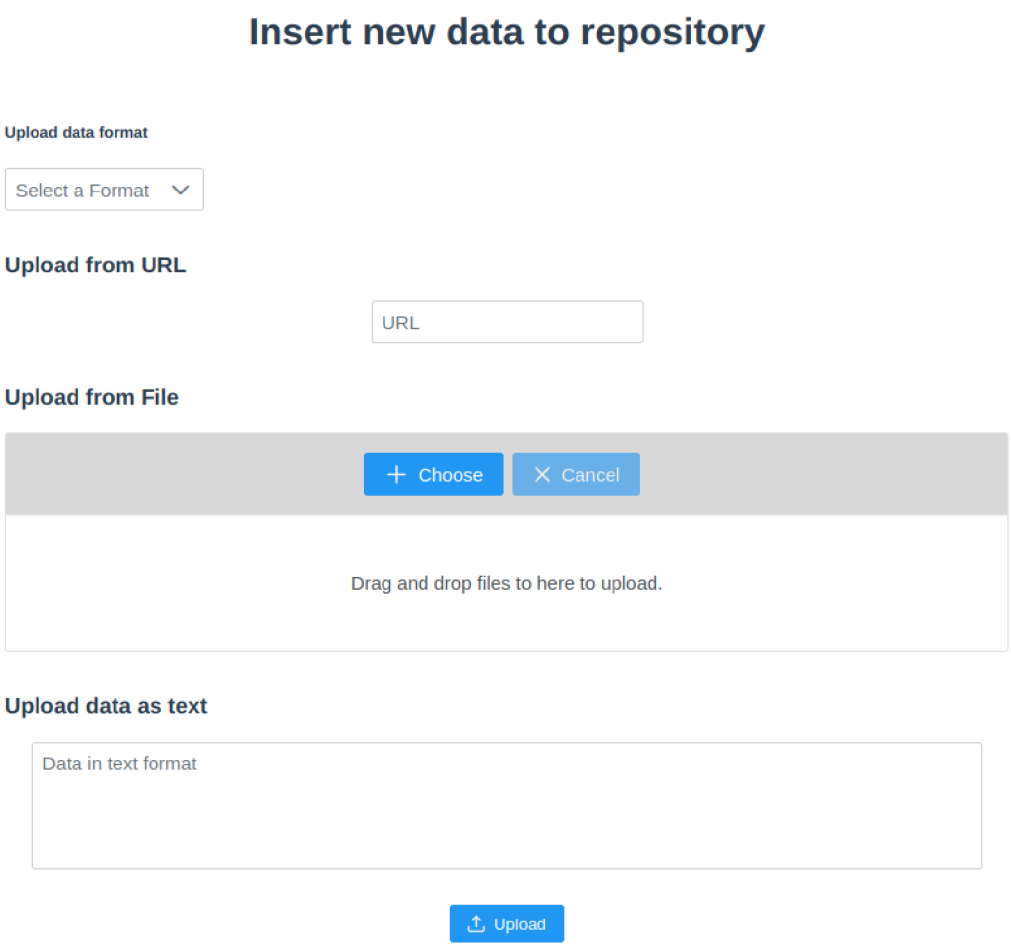
Užívateľom v rámci tohto komponentu sa ponúkajú tri typy spôsobov, ako zvolené dáta nahrať do svojho repozitára. Prvou z nich je možnosť zadania webovej adresy (URL), na ktorej sa nachádza súbor obsahujúci RDF dáta.

Ak je cieľom nahrať dáta z lokálne pripraveného súboru, tak užívateľ môže využiť druhú možnosť, v podobe pokročilého nahrávača súborov s podporou *drag and drop* (slovensky voľne preložené *pritiahnuť a spustiť*) funkcie.

Poslednou možnosťou je ručný zápis informácií do vyhradeného textového poľa, napríklad v prípade potreby rýchlo nahrať menšie množstvo dát.

Pre úspešné odoslanie a nahranie dát na server je potrebné nastaviť hlavičku *Content-Type* v POST žiadosti, aby server úspešne rozpoznal o aký formát dát sa jedná. Formát zadaných dát je možné si zvoliť z rozbaľovacej ponuky (anglicky *dropdown menu*), ktorá je umiestnená v hornej časti komponentu.

Dáta je potom možné odoslať pomocou modrého tlačidla *Upload*. Ukážka komponentu je uvedená na obrázku 5.8.



Obr. 5.8: Komponent *InsertData* ponúka tri možnosti nahrania dát do zvoleného úložiska. Prvou z nich je zadanie URL súboru, ktorý obsahuje príslušné RDF dáta. Pokročilý nahrávač súborov s podporou *drag and drop* predstavuje druhú možnosť. Užívateľ môže buď jednoducho pretiahnuť potrebný súbor do vyznačeného poľa alebo po stlačení modrého tlačidla *Choose* si ho môže vyhľadať pomocou prieskumníka súborov. Tretia možnosť je umiestnená v dolnej časti komponentu v podobe textového poľa. Do neho je potrebné ručne napísať vlastné dáta. Z rozbaľovacej ponuky v hornej časti komponentu musí užívateľ vybrať presne o aký formát dát sa jedná, pred ich odoslaním.

5.3 Možné rozšírenia aplikácie

Implementovaný program v plnej miere spĺňa požiadavky stanovené v zadaní diplomovej práce. Napriek tomu existujú také časti aplikácie, ktoré by mohli byť v budúcnosti rozšírené a vylepšené. Týkajú sa vzhľadu alebo ďalších funkcionalít.

Ako príklad môže slúžiť komponent *RdfEditor* zabezpečujúci tvorbu SPARQL dopytov. Pre ešte väčšie uľahčenie práce užívateľov by sa dalo zaistiť automatické doplňovanie, dokončovanie slov. Editor počas písania by navrhoval slová, ktoré by považoval za pravdepodobné, že budú nasledovať po aktuálnom slove. Rozpísané kľúčové slová by bolo možné pomocou stlačenia tlačidla *Tab* dokončiť. Ďalej by bolo možné pridať funkciu vygenerovania šablón, kostry často používaných dopytov, ako je napríklad *SELECT* alebo *ASK*. Užívateľ by iba doplňoval názov voľných premenných.

Taktiež by bolo možné vylepšiť textové pole v komponente *InsertData* s podobnými funkcionalitami. Kontrola správneho formátu dát, na základe zvoleného typu, by vedela ušetriť čas užívateľov.

V neposlednej rade by bolo potrebné previesť zmeny vzhľadu stránok a komponentov, aby si svoj obsah lepšie prispôsobovali rozmeru dostupného voľného miesta na zobrazovacej ploche. Táto úprava by sa hlavne týkala zariadení s malým displejom, ako sú mobilné zariadenia.

Kapitola 6

Testovanie aplikácie

Táto kapitola sa zameriava na popis testovania vytvorenej aplikácie. Nasledujúce sekcie popisujú to, aké dáta a postupy boli použité pri realizácií testov. Pri tejto činnosti sa hlavne zameriavalo na správne splnenie zadaním stanovených požiadaviek a použiteľnosť navrhnutých funkcionalít.

6.1 Zdroje testovacích dát

Jednou dôležitou podmienkou testovania je to, aby boli k dispozícií vhodné testovacie dáta. U týchto dát je potrebné zabezpečiť, aby mali vhodné vlastnosti, ako je napríklad ich formát, a aby boli k dispozícií v dostatočnom množstve. S týmito dátami je potom možné skúšať implementované funkcionality.

Hlavným zdrojom dát pri testovaní v tomto projekte boli voľne dostupné dátové sady (anglicky *datasets*) zo stránok DBpedia[1] a Datahub[2]. Výhodou týchto zdieľaných dátových sád bol ich veľký objem, zahŕňali v sebe veľké množstvo trojíc. Ďalej obsahovali značný počet priestorov mien a prefixov, čo umožnilo dobré overenie možnosti ich spravovania a automatického doplňovania dopytov.

6.2 Spôsoby testovania

Počas celého vývoja aplikácie bolo testovanie vykonávané manuálne. Po každej významnej zmene kódu boli overené funkčnosti už existujúcich a práve implementovaných, modifikovaných funkcionalít. Aby testy lepšie odzrkadlili reálnu prevádzku a využívanie aplikácie, tak boli vyskúšané rôzne scenáre, situácie, do ktorých by sa mohli dostať užívatelia počas ich bežnej činnosti. Boli postupne vyskúšané jednotlivé kroky, ktoré by potreboval previesť používateľ aplikácie.

Dôležité bolo overiť to, ako aplikácia reaguje na nesplnenie určitých očakávaných podmienok. Či je schopná vysporiadať sa správne s chybovými stavmi, spôsobenými zo strany užívateľa alebo servera. Medzi takéto stavy môže patriť napríklad chybové hlásenie od servera o nemožnosti previesť požadovanú operáciu, neexistencia zvoleného úložiska a tak ďalej. Pri výskyte takýchto chýb sa dbalo na dostatočné informovanie užívateľa o príčinách zlyhania.

[1] https://databus.dbpedia.org/dbpedia/
[2] https://old.datahub.io/dataset?tags=format-rdf

Vo formulároch, vyskytujúcich sa v jednotlivých komponentoch, sa overovala správnosť kontroly požadovaného formátu a vyplnenia povinných vstupov. V prípade vytvoreného editora sa kontrolovala správnosť použitých regulárnych výrazov, ich schopnosť rozpoznať definované prefixy v tele dopytov. Následne bolo overené ich automatické nahradzovanie s príslušnými mennými priestormi v dopyte posielanom na server.

Tak ako už bolo uvedené v sekcií 6.1, použité dátové sady obsahovali často väčšie množstvo dát. Táto vlastnosť bola vhodná na overenie schopnosti aplikácie zobrazovať výsledky dopytov so značným počtom trojíc, napríklad v komponente *QueryResults* alebo *ResourceExplorer*. Pri týchto testoch sa zistilo značné spomalenie v zobrazení dát, keď ich počet presiahne niekoľko tisíc kusov. Aplikácia je stále použiteľná, iba stráca svoju rýchlosť, plynulosť behu. Dôvodom tohto nedostatku môže byť preťaženie, spojené so spracovaním v použitých *PrimeVue* komponentoch.

Lokálne testovanie aplikácie

Pre lokálne testovanie aplikácie bolo potrebné zaistiť vhodný server prijímajúci odoslané žiadosti. Na tento cieľ bol využitý rámec *RDF4J*, presnejšie jeho serverová časť, ktorá už bola predstavená v kapitole 2, sekcia 2.5. Pre zabezpečenie rámcom požadovaného behového prostredia bol využitý Java servlet kontajner, *Apache Tomcat*, verzia 9. Druhý, z využitých serverov, bol samotný server tvoriaci súčasť projektu *FitLayout*. Overilo sa schopnosť preposielania dotazov medzi ním a vytváranou aplikáciou.

Vytvorenie NPM balíku

Po dohode s vedúcim práce, aplikácia bola vytvorená tak, aby sa dala integrovať do už existujúceho projektu. Preto boli jednotlivé komponenty zaisťujúce požadované funkcionality navrhnuté tak, aby sa dali jednoducho nasadzovať aj samostatne v iných projektoch. Ich fungovanie bolo prispôsobené podmienkam stanoveným v danom projekte.

Najjednoduchším spôsobom zdieľania týchto komponentov bolo vytvorenie jedného samostatného balíka. Tento balík bol následne publikovaný do verejného online registra *NPM* pod názvom *rdf-storage-browsing-web-interface*[3]. Týmto spôsobom sa zabezpečilo možné využívanie aj v iných klientskych aplikáciách, bezplatne podľa potreby.

Pri vytváraní balíka bola využitá knižnica *Rollup.js*[4], ktorá zaistila kompiláciu malých kúskov kódu do jednej knižnice.

Integrácia do aplikácie PageView

Komponenty zahrnuté v publikovanom balíku boli ďalej testované a používané v rámci projektu *PageView*[5]. Tento projekt zastupuje úlohu webovej klientskej aplikácie, v rámci hlavného projektu *FitLayout*. Overovala sa ich schopnosť integrácie do iných prostredí a správna komunikácia s ďalšími komponentmi.

[3] https://www.npmjs.com/package/rdf-storage-browsing-web-interface
[4] https://rollupjs.org/guide/en/
[5] https://github.com/FitLayout/PageView

Kapitola 7

Záver

Hlavným zámerom tejto diplomovej práce bolo vytvorenie aplikácie, ktorá umožňuje správu RDF dát v užívateľom vybraných úložiskách. Tento zámer bol ďalej doplnený o problematiku dátového modelu RDF a predstavenie jeho elementárnych prvkov v podobe RDF trojíc. Možnosti, zahrňujúce dopytovanie nad úložiskami RDF informácií alebo ich modifikácie, boli prezentované pomocou jazyka SPARQL.

Vďaka tomu, že internet a webové stránky sa stali neoddeliteľnou súčasťou života človeka, narástol dopyt o webové aplikácie, tvoriace značnú konkurenciu desktopovým aplikáciám. Firmy vytvárajú rôzne nástroje pre uľahčenie vývoja takýchto aplikácií, aby vedeli uspokojiť dopyt trhu. Cieľom tejto práce bolo oboznámenie sa s niektorými, v dnešnej dobe najpoužívanejšími, nástrojmi v tejto oblasti.

Pre získanie lepšej predstavy o fungovaní a vlastnostiach programov, pracujúcich s RDF úložiskami, boli preskúmané už existujú aplikácie, ktoré sa často používajú v takýchto oblastiach a poskytujú aj zabudované editory. Skúmal sa ich stav, kladné stránky a možnosti vylepšenia.

Na základe získaných poznatkov z teoretickej časti práce a preskúmaných riešení bola navrhnutá aplikácia, predstavujúca hlavný výstup tejto práce. Bolo potrebné stanoviť jej štruktúru, architektúru a funkcionality, ktoré bude ponúkať budúcim užívateľom. Ako najlepšia voľba sa javila implementovať ju ako webovú aplikáciu, tvorenú jedinou stránkou a striedajúcimi sa komponentmi, s využitím rámca Vue.js. Pri vývoji bol kladený dôraz na prehľadnosť a užívateľskú prívetivosť. Cieľom bolo čo najlepšie zjednodušiť a zrýchliť často opakované operácie.

Po konzultácií a dohode s vedúcim diplomovej práce sa rozhodlo, že vytváraný program bude napojený na už existujúci projekt s názvom *FitLayout*. Bude prispôsobený tak, aby vedel vhodným spôsobom spolupracovať a komunikovať s jeho serverom.

Výsledná aplikácia funguje podľa očakávania, vyplývajúceho zo zadania. Z jeho komponentov bol vytvorený a publikovaný samostatný balík pre uľahčenie ich možného budúceho využitia v iných klientskych aplikáciách. Testovanie prebiehalo pomocou verejných dátových sád rôznej veľkosti, napodobňujúcich reálne používanie bežnými užívateľmi.

Napriek spoľahlivému fungovaniu a kladných vlastností aplikácie existujú aj také časti programu, týkajúce sa vzhľadu alebo funkcionalít, ktoré by mohli byť v budúcnosti rozšírené a vylepšené. Ako príklad by mohla slúžiť funkcia generovania šablón pre často používané typy dopytov alebo automatické navrhovanie nasledujúcich slov, ktoré by bolo doplnené do centrálneho editora aplikácie.

Literatúra

[1] Apache Jena. *TDB* [online]. jena.apache.org, 2018 [cit. 2021-10-08]. Dostupné z: https://jena.apache.org/documentation/tdb/.

[2] Bebee, B. *Blazegraph* [online]. blazegraph, február 2020 [cit. 2021-10-08]. Dostupné z: https://github.com/blazegraph/database/wiki.

[3] Beckett, D. *RDF 1.1 N-Triples* [online]. W3C, február 2014 [cit. 2021-10-07]. Dostupné z: http://www.w3.org/TR/n-triples/.

[4] Beckett, D., Berners Lee, T., Prud'hommeaux, E. a Carothers, G. *RDF 1.1 Turtle* [online]. W3C, február 2014 [cit. 2021-10-07]. Dostupné z: https://www.w3.org/TR/turtle/.

[5] Blin, G. a Cure, O. *RDF Database Systems: Triples Storage and SPARQL Query Processing*. 1. vyd. Elsevier, Inc., 2015 [cit. 2021-10-05]. ISBN 978-0-12-799957-9.

[6] Blumenthal, S. *JavaScript: JavaScript For Beginners - Learn JavaScript with ease in HALF THE TIME - Everything about the Language, Coding, Programming and Web Pages that you need to know!* 6. vyd. CreateSpace Independent Publishing Platform, 2017. ISBN 978-1548799380.

[7] Brickley, D. a Guha, R. *RDF Schema 1.1* [online]. W3C, február 2014 [cit. 2021-10-06]. Dostupné z: https://www.w3.org/TR/rdf-schema/.

[8] Cyganiak, R., Wood, D. a Lanthaler, M. *RDF 1.1 Concepts and Abstract Syntax* [online]. W3C, február 2014 [cit. 2021-10-05]. Dostupné z: https://www.w3.org/TR/rdf11-concepts/.

[9] DuCharme, B. *Learning SPARQL*. 2. vyd. Oreilly Media, Inc., 2013 [cit. 2021-10-01]. ISBN 978-1-449-37143-2.

[10] Gandon, F. a Schreiber, G. *RDF 1.1 XML Syntax* [online]. W3C, február 2014 [cit. 2021-10-07]. Dostupné z: https://www.w3.org/TR/rdf-syntax-grammar/.

[11] Harris, S. a Seaborne, A. *SPARQL 1.1 Query Language* [online]. W3C, marec 2013 [cit. 2021-10-09]. Dostupné z: https://www.w3.org/TR/sparql11-query/.

[12] Herman, I., Adida, B., Sporny, M. a Birbeck, M. *RDFa 1.1 Primer - Third Edition* [online]. W3C, marec 2015 [cit. 2021-10-07]. Dostupné z: https://www.w3.org/TR/rdfa-primer/.

[13] Indeed Editorial Team. *What Is a Web Application? How It Works, Benefits and Examples* [online]. Indeed, apríl 2021 [cit. 2021-10-15]. Dostupné z: https://www.indeed.com/career-advice/career-development/what-is-web-application.

[14] Lassila, O. a Swick, R. R. *Resource Description Framework (RDF) Model and Syntax Specification* [online]. W3C, február 1999 [cit. 2021-10-03]. Dostupné z: https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[15] Martin, M. *Difference between Website and Web Application (Web App)* [online]. Guru99, október 2021 [cit. 2021-10-15]. Dostupné z: https://www.guru99.com/difference-web-application-website.html.

[16] Meindertsma, J. *What's the best RDF serialization format?* [online]. Ontola, jún 2019 [cit. 2021-10-07]. Dostupné z: https://ontola.io/blog/rdf-serialization-formats/.

[17] Nelson, B. *Getting to Know Vue.js: Learn to Build Single Page Applications in Vue from Scratch*. 1. vyd. Apress, 2018. ISBN 978-1-4842-3780-9.

[18] Northwood, C. *The Full Stack Developer: Your Essential Guide to the Everyday Skills Expected of a Modern Full Stack Web Developer*. 1. vyd. Apress, 2018. ISBN 978-1484241516.

[19] Ontotext. *What is an RDF Triplestore?* [online]. ontotext.com [cit. 2021-10-08]. Dostupné z: https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf-triplestore/.

[20] Ontotext. *What is SPARQL?* [online]. ontotext.com [cit. 2021-10-09]. Dostupné z: https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/.

[21] Ontotext. *About GraphDB* [online]. ontotext.com, 2021 [cit. 2021-10-08]. Dostupné z: https://graphdb.ontotext.com/documentation/standard/about-graphdb.html.

[22] Powers, S. *Practical RDF*. 1. vyd. Oreilly Media, Inc., 2003 [cit. 2021-10-07]. ISBN 978-0-596-00263-3.

[23] Rajput, M. *Web Application Architecture: Everything You Need to Know About* [online]. Mindinventory, január 2021 [cit. 2021-10-15]. Dostupné z: https://www.mindinventory.com/blog/web-application-architecture/.

[24] Rauschmayer, D. A. *JavaScript for impatient programmers (ES2021 edition)*. 1. vyd. Dr. Axel Rauschmayer, 2021. ISBN 978-1-09-121009-7.

[25] RDF4J. *The Eclipse RDF4J Framework* [online]. rdf4j.org [cit. 2021-10-08]. Dostupné z: https://rdf4j.org/about/.

[26] Segaran, T., Evans, C. a Taylor, J. *Programming the semantic web*. 1. vyd. Oreilly Media, Inc., 2009 [cit. 2021-10-01]. ISBN 978-0-596-15381-6.

[27] STARDOG. *Learn SPARQL* [online]. stardog.com, marec 2018 [cit. 2021-10-03]. Dostupné z: https://www.stardog.com/tutorials/sparql/.

[28] Vue.js. *Introduction* [online]. 2021 [cit. 2021-10-18]. Dostupné z: https://v3.vuejs.org/guide/introduction.html.

[29] VUE.JS. *Vue Router guide* [online]. 2021 [cit. 2021-10-19]. Dostupné z: https://next.router.vuejs.org/guide/.

[30] VUE.JS. *What is Vuex?* [online]. 2021 [cit. 2021-10-19]. Dostupné z: https://next.vuex.vuejs.org/.

[31] W3. *What is an RDF Triplestore?* [online]. w3.org, 2012 [cit. 2021-10-08]. Dostupné z: https://www.w3.org/2001/sw/wiki/OpenLink_Virtuoso.

[32] W3. *Apache Jena* [online]. w3.org, január 2018 [cit. 2021-10-08]. Dostupné z: https://www.w3.org/2001/sw/wiki/Apache_Jena.

[33] W3SCHOOLS. *HTML Web Storage API* [online]. 2021 [cit. 2022-03-14]. Dostupné z: https://www.w3schools.com/html/html5_webstorage.asp.

Príloha A

Obsah priloženého pamäťového média

Popis obsahu jednotlivých adresárov priloženého DVD nosiča:

A.1 Diplomova praca - Text

V tomto adresári sa nachádza text diplomovej práce vo formáte PDF a zdrojových súborov použitých k jeho vytvoreniu:

- **LATEX** - adresár obsahujúci zdrojové súbory písomnej práce vytvorenej v LaTeXe
- **PDF** - adresár obsahujúci písomnú prácu vo formáte *PDF*

A.2 Diplomova praca - Zdrojove subory

Adresár uchováva všetky potrebné zdrojové súbory pre fungovanie aplikácie:

- **Aplikacia** - adresár, obsahujúci zdrojové súbory potrebné pre beh aplikácie
- **READEME.md** - súbor vytvorený vo formáte *Markdown*. Obsahuje vymenované závislostí aplikácie a postup pre lokálne nasadenie programu.