

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

REPOZITÁR RECEPTOV V SIETI PREPOJENÝCH
DÁT
BAKALÁRSKA PRÁCA

2020
IVETA BALINTOVÁ

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

REPOZITÁR RECEPTOV V SIETI PREPOJENÝCH DÁT

BAKALÁRSKA PRÁCA

Študijný program: Aplikovaná informatika
Študijný odbor: Aplikovaná informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Školiteľ: RNDr. Martin Homola, PhD.

Bratislava, 2020
Iveta Balintová



Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Iveta Balintová
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Repozitár receptov v sieti prepojených dát
Recipe repository in the linked open data network

Anotácia: Sieť prepojených dát (LOD) umožňuje publikovanie štruktúrovaných dát na webe. Takéto dáta sú následne strojovo spracovateľné vďaka ich anotácii vhodnými ontológiami.

Cieľ: Cieľom práce je navrhnuť a implementovať repozitár receptov v sieti LOD. Práca sa zameria na vhodné ontológie pre tento účel, prípadne ich úpravu, či tvorbu, a následne na vytvorenie repozitára a manažment dát receptov v tomto repozitári.

Literatúra: [1] Bizer, C., Heath, T. and Berners-Lee, T., 2011. Linked data: The story so far. In Semantic services, interoperability and web applications: emerging concepts (pp. 205-227). IGI Global.
[2] Heath, T. and Bizer, C., 2011. Linked data: Evolving the web into a global data space. Synthesis lectures on the semantic web: theory and technology, 1(1), pp.1-136.
[3] Rychtárik, M. 2019. Inteligentný receptár na báze prepojených dát. Bachelor thesis, Comenius University in Bratislava.

Vedúci: doc. RNDr. Martin Homola, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.
Dátum zadania: 04.10.2019

Dátum schválenia: 14.10.2019

doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Obsah

1	Východisková kapitola	1
1.1	Sémantický web	1
1.1.1	Linked Open Data	2
1.1.2	RDF	4
1.1.3	URI	6
1.1.4	Ontológie a slovníky	6
1.1.5	RDF Schema	7
1.1.6	OWL	8
1.1.7	SPARQL	9
1.1.8	Triplestore	10
1.2	Použité technológie	10
1.2.1	Apache Jena	10
1.2.2	Spring Boot	12
1.2.3	React	12
1.2.4	Redux	12
1.2.5	Axios	13
1.2.6	Bootstrap	13
1.3	Existujúce riešenia	14
1.3.1	Varecha	14
1.3.2	Yummly	15
1.3.3	Bakalárska práca Inteligentný receptár na báze prepojených dát	15
2	Návrh riešenia	17
2.1	Návrh ontológie	17
2.1.1	Protégé	18
2.1.2	Existujúca ontológia	18
2.1.3	Navrhnutá ontológia	18
2.2	Funkcionalita systému	20
2.3	Návrh databázy aplikácie	23
2.4	Návrh backendu aplikácie	23

2.5	Návrh frontendu aplikácie	24
3	Implementácia	26
3.1	Implementačný jazyk	26
3.2	Správa stavu aplikácie	26
3.3	Prepojenie frontendu, backendu a databázy	27

Zoznam obrázkov

1.1	Semantic Web Layer Cake, zdroj: [16]	2
1.2	Orientovaný graf, ktorý vznikol na základe nižšie uvedeného kódu	5
1.3	Výsledok dopytu nad DBpedia	9
1.4	Tok dát v Reduxe	13
1.5	Recept na vianočku, zdroj: [15]	14
1.6	Recept na gazpacho, zdroj: [22]	15
2.1	Grafické vyjadrenie návrhu ontológie o receptoch	17
2.2	Vizualizácia štruktúry balíčkov a tried na backende	23
2.3	Vizualizácia štruktúry balíčkov, komponentov a ich vzťahov na frontende	24
3.1	Vizualizácia štruktúry balíčkov a súborov pracujúcich so stavom aplikácie	27
3.2	Diagram vyjadrujúci prepojenie jednotlivých častí aplikácie	28

Kapitola 1

Východisková kapitola

1.1 Sémantický web

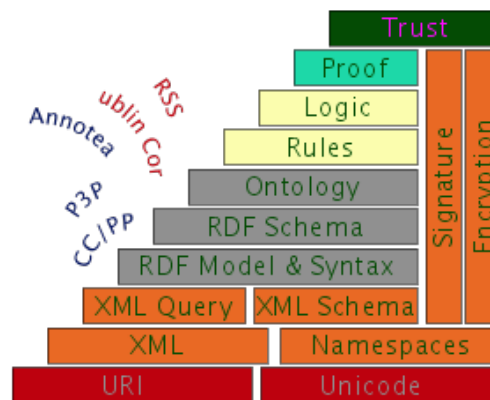
Sémantický web, podľa [21] je rozšírením súčasného webu. Väčšina obsahu, ktorý sa v súčasnosti na webe nachádza je navrhnutá pre ľudí. Programy dokážu rozpoznať, aká časť stránky je hlavička, kde sa nachádza odkaz na inú stránku, avšak nedokážu pochopiť význam toho, čo sa vlastne v jednotlivých častiach nachádza. Riešením má byť práve sémantický web, ktorý umožňuje reprezentovať a voľne publikovať dáta v nejakom vhodnom jazyku (RDF, viď. 1.1.2), ktorý im umožňuje priradiť aj význam. Tradičné systémy na reprezentáciu poznatkov boli väčšinou centralizované, čo si vyžadovalo jednotné definície bežných pojmov. Takáto centrálna kontrola sa však kvôli neustálemu rozrastaniu systému stáva nezvládnuiteľnou, a teda definície a významy priradzujeme použitím ontológií (viď. 1.1.4).

Semantic Web Stack, prípadne Semantic Web (Layer) Cake ilustruje štruktúru sémantického webu. Na obr. 1.1 vidíme ako sú organizované technológie, ktoré sú pre tento web štandardizované. Každá vrstva využíva schopnosti nižšej vrstvy. Tieto vrstvy môžeme rozdeliť do 3 skupín a na základe [12] ich definovať nasledovne:

- Pojmy zo spodnej vrstvy poznáme už z hypertextového webu. Z toho vyplýva, že sémantický web je iba rozšírením klasického webu. URI (viď. 1.1.3) umožňujú jednoznačne identifikovať zdroje. UNICODE je kódovanie slúžiace na reprezentáciu a spracovanie textov v rôznych jazykoch. XML, XML Query, XML Schema a XML Namespaces boli vyvinuté a štandardizované konzorciom W3C. XML je značkový jazyk, ktorý umožňuje vytvárať dokumenty so štruktúrovanými dátami, ktorým sémantický web dodáva význam. XML Query je štandardizovaný jazyk na kombinovanie dokumentov, databáz, či webových stránok, umožňuje robiť dopyty nad XML dátami. XML Schema je jazyk na vyjadrenie obmedzení o XML dokumentoch a špecifikuje, ako formálne opísať prvky v týchto dokumentoch. Názvy elementov si definuje autor XML dokumentu. Pri spájaní viacerých

dokumentov môže dochádzať ku konfliktu názvov v prípade, keď je rovnaký názov použitý s iným významom. Práve tento problém riešia XML Namespaces. Vytvorením prefixu pre namespace a následným používaním prefixu využívame názvy elementov v takom kontexte, v akom sú definované v nami vybranom namespace.

- Druhou skupinou, nachádzajúcou sa v strede, sú technológie štandardizované W3C. Umožňujú budovať aplikácie sémantického webu. Sú nimi RDF Model a Syntax (viď. 1.1.2), RDF Schema (viď. 1.1.5) a Ontológie (viď. 1.1.4). RDF je dátový model, v ktorom vyjadrujeme vzťah medzi hocíjakými dvoma zdrojmi vo forme trojice (subjekt, predikát, objekt). RDF Schema (RDFS) je univerzálny jazyk, ktorý umožňuje definovať nové slovníky. Ontológie, v užšom ponímaní slovníky, sú súbory pojmov a vzťahov medzi pojmami z určitej časti sveta, ktorá nás zaujíma.
- Treťou skupinou sú technológie, ktoré štandardizované zatiaľ nie sú alebo sú iba nápadmi. Na základe súborov pravidiel vieme pomocou logiky odvádzať nové informácie. Tieto odvodené informácie podporujeme dôkazmi. Najvyššie sa nachádza trust(pravda), teda mechanizmus, ktorým by sme vedeli určiť, ktorým informáciám môžeme dôverovať.



Obr. 1.1: Semantic Web Layer Cake, zdroj: [16]

1.1.1 Linked Open Data

Termín Linked Data, spracovaný podľa [3], označuje súbor všetkých dát publikovaných na webe pomocou technológií sémantického webu. Ide v podstate o sémantický web aplikovaný do praxe. Ak sú linked data, teda prepojené údaje, voľne dostupné na opätovné použitie označujeme ich termínom Linked Open Data. Linked Open Data sú silnou kombináciou prepojených a otvorených údajov. Jedným z pozoruhodných

príkladov LOD je je DBpedia – snaha o získanie štruktúrovaných informácií z Wikipédie a ich sprístupnenie na webe.

Zásady pri tvorbe prepojených údajov (Linked data principles) predstavil Tim Berners-Lee. Ide o tieto pravidlá:

1. Používajte URI ako názvy objektov.
2. Používajte identifikátory URI protokolu HTTP, aby ľudia mohli tieto mená vyhľadať.
3. Keď niekto vyhľadá URI, poskytnite užitočné informácie o tom, čo názov identifikuje pomocou otvorených štandardov, ako sú napr. RDF alebo SPARQL.
4. Zahrňte odkazy na ďalšie URIs, aby ľudia mohli dohľadať ďalšie informácie.

Štvrtý princíp prepojených údajov podporuje použitie hypertextových odkazov nie len medzi webovými dokumentami, ale medzi ľubovoľnými objektami, napr. odkaz medzi človekom a miestom, alebo medzi miestom a spoločnosťou. Oproti klasickému webu však majú tieto odkazy aj typy, teda napr. medzi dvoma ľuďmi môže mať odkaz typ priateľ.

Podľa [10], Tim Berners-Lee navrhol 5-hviezdičkovú schému rozvinutia linked open data.

- Jednu hviezdičku získajú dáta, ktoré sú sprístupnené na webe v ľubovoľnom formáte pod otvorenou licenciou. Môžeme si ich prečítať, zdieľať, meniť, je jednoduché ich poskytnúť, avšak dáta sú uzavreté v rámci dokumentu a je ťažké ich z neho dostať.
- Dve hviezdičky získajú dáta, ktoré sú sprístupnené v štruktúrovanej podobe, je možné ich spracovať a proprietárnym softvérom získať agregácie, či výpočty. Dáta sú však stále uzavreté v rámci dokumentu a ich získanie závisí od proprietárneho softvéru.
- Tri hviezdičky sú priradené dátam, ktoré sú sprístupnené v neproprietárnom otvorenom formáte.
- Štyroma hviezdičkami sú hodnotené dáta, v ktorých používame URI na určenie vecí tak, aby sa na nich mohli ľudia odkazovať. Takéto dáta je bezpečné kombinovať s inými dátami, keďže vieme, že v prípade rovnakého URI ide o rovnakú vec. Najbežnejším spôsobom reprezentácie dát je použitie RDF.
- Päť hviezdičiek, a teda najlepšie hodnotenie, získavajú dáta, ktoré sú nalinkované na iné. Vytvárame tým kontext a umožňuje nám to objavovať ďalšie súvisiace dáta.

1.1.2 RDF

Táto podkapitola je spracovaná podľa [18, 1]. RDF je skratka pre Resource Description Framework. Doteraz je to najjednoduchší a zároveň najsilnejší z formátov, ktorý poskytuje spôsob na vyjadrovanie informácií o zdrojoch. Zdrojom môže byť hocičo, teda dokument, osoba, fyzický objekt a podobne. RDF bol vyvinutý a odsúhlasený W3C. Je určený pre situácie, pri ktorých informácie na webe musia byť spracovávané prevažne aplikáciami, nie len zobrazované ľuďom.

RDF dáta sú zložené z tvrdení. Tvrdenie, v RDF nazývané aj trojica, resp. triple, umožňuje vyjadriť vzťah medzi dvoma zdrojmi a má vždy nasledujúcu štruktúru:

<subjekt> <predikát> <objekt>

Príklad trojice, vyjadrujúci informáciu, že Mount Everest je najvyšší vrch v Himalájach:

<Himaláje> <najvyšší vrch> <Mount Everest>

Pre lepšie pochopenie môžeme RDF dáta skúsiť porovnať s dátami v relačnej databáze, kde sú údaje uložené v tabuľkách. Zisťujeme, že v podstate bunka v tabuľke je prezentovaná tiež troma hodnotami. Identifikátor pre riadok sa nazýva subjekt, je to vec, o ktorej daný výrok hovorí. Identifikátor pre stĺpec sa nazýva predikát, hovorí o nejakej vlastnosti entity daného riadka(subjektu) a hodnota v bunke sa nazýva objekt. Ak by sme teda chceli dáta z relačnej databázy pretransformovať na dáta v RDF formáte, dalo by sa to pomerne jednoducho. V prípade opačného smeru, teda zmeny ľubovoľných RDF dát na relačnú databázu by bolo problémom, že predikáty nie sú pevne dané, a teda jediné, čo by sme vedeli s istotou vytvoriť by bola jedna veľká tabuľka s 3 stĺpcami, pričom každý riadok by vyjadroval jeden triple.

Trojice sa stávajú zaujímavejšími, keď viac ako jedna trojica opisuje rovnakú entitu, teda zdroj. V tom prípade je výhodné vizualizovať ich pomocou súvislého orientovaného grafu, pričom subjekt a objekt sú vrcholmi grafu a predikát je hrana medzi nimi. Skupina rovnakých tvrdení, môže byť preložená do rôznych syntaxí, avšak vždy budú predstavovať ten istý graf. Medzi formáty pre písanie RDF grafov patria Turtle family of RDF languages (N-Triples, Turtle, TriG, N-Quads), JSON-LD, RDFa, RDF/XML. Vyššie uvedená trojica (<Himaláje> <najvyšší vrch> <Mount Everest>) vyjadrená použitím Turtle syntaxe:

dbr:Himalayas dbp:highest dbr:Mount_Everest

RDF je teda predovšetkým systém na modelovanie údajov. Každý vzťah medzi hocíjakými dvoma entitami je presne reprezentovaný, čo umožňuje veľmi jednoduché

spájanie údajov z viacerých zdrojov. Dôvodom ľahkého spájania je to, že nie je potrebné usporiadať stĺpce tabuliek, obávať sa chýbajúcich údajov konkrétneho stĺpca a podobne, keďže vzťah buď existuje alebo nie.

Jednotlivé časti trojice (subjekt, predikát a objekt) môžu byť rôzne reprezentované. Prostredníctvom IRIs (viď. 1.1.3) môžeme vyjadriť subjekt, objekt aj predikát. Literály, jednoduché hodnoty ako sú reťazce, dátumy, čísla, môžu vystupovať iba ako objekt. V pozícií objektu a subjektu môžu vystupovať aj tzv. blank nodes. Sú to prázdne uzly, ktoré sa môžu použiť na označenie zdrojov bez toho, aby boli výslovne pomenované. Naznačujú teda existenciu nejakej veci, ktorá však nie je definovaná prostredníctvom IRI.

Príklad blank node (`_:someone`):

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
```

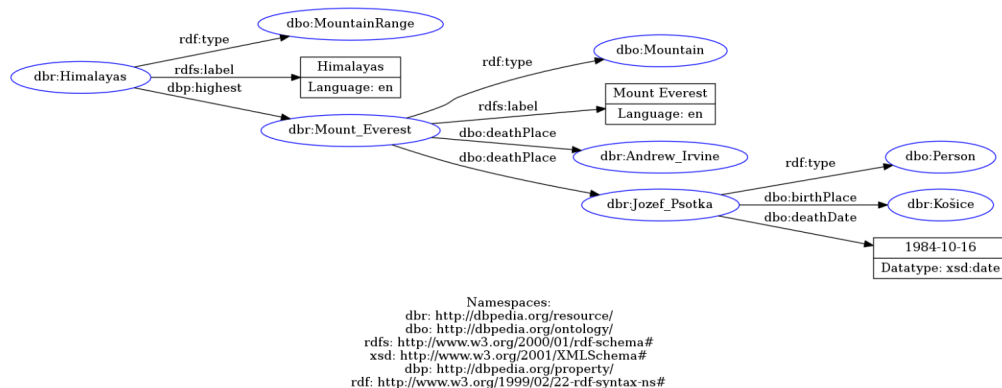
```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
```

```
@prefix ex: <http://example.org/example#>.
```

```
ex:Jane foaf:knows _:someone.
```

```
_:someone foaf:name "John"^^xsd:string.
```

Na obr. 1.2 vidíme graf vyjadrujúci informácie o Mount Evereste, Himalájach a Jozefovi Psotkovi. Tento graf vznikol použitím Turtle syntaxe. Tá je najčitateľnejšia pre ľudí.



Obr. 1.2: Orientovaný graf, ktorý vznikol na základe nižšie uvedeného kódu

```
@prefix dbr: <http://dbpedia.org/resource/>.
```

```
@prefix dbo: <http://dbpedia.org/ontology/>.
```

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

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
```

```
@prefix dbp: <http://dbpedia.org/property/>.
```

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

```
dbr:Himalayas
```

```
a dbo:MountainRange;
```

```
rdfs:label "Himalayas"@en;
```

```
dbp:highest dbr:Mount_Everest.
```

```
dbr:Mount_Everest
```

```
a dbo:Mountain;
```

```
rdfs:label "Mount Everest"@en;
```

```
dbo:deathPlace dbr:Andrew_Irvine;
```

```
dbo:deathPlace dbr:Jozef_Psotka.
```

```
dbr:Jozef_Psotka
```

```
a dbo:Person;
```

```
dbo:birthPlace dbr:Košice;
```

```
dbo:deathDate "1984-10-16"^^xsd:date.
```

1.1.3 URI

URI, podľa [1] ponúka globálnu identifikáciu pre zdroje na celom webe. Jeho syntax umožňuje „dereferenciu“ - použitie všetkých informácií v URI (ktoré špecifikujú meno servera, protokol, číslo portu, meno súboru a podobne) na to, aby lokalizovali súbor. Ak všetky časti fungujú, vtedy môžeme vyhlásiť, že URI je zároveň aj URL, teda URL je špeciálnym prípadom URI. Avšak tento rozdiel nie je podstatný z pohľadu modelovania. Vieme, že ak ukazujú dva vrcholy v RDF grafe na rovnaké URI, ide o ten istý vrchol. Na základe [18] vieme, že IRI je skratka pre International Resource Identifier, a je zovšeobecnením URI. V reťazci IRI môžu byť použité aj znaky, ktoré nie sú v ASCII.

1.1.4 Ontológie a slovníky

Ontológie

Ontológia [13] je súborom presných tvrdení o nejakej časti sveta, ktorá nás zaujíma. Zvyčajne máme k dispozícii množiny pojmov a vzťahy medzi nimi. Označujeme ich ako doménu záujmu alebo predmet ontológie. Jednotlivé tvrdenia sú prezentované ako triedy a vzťahy, a vytvárajú hierarchickú štruktúru celej ontológie. Súčasťou ontológií sú takisto aj obmedzenia a pravidlá, pomocou ktorých je možné odvodzovať nové fakty, ktoré neboli explicitne uvedené. Presnými opismi v ontológiách predchádzame nedorozumeniam, ku ktorým môže dochádzať v oblasti ľudskej komunikácie a zabezpečujeme,

aby sa softvér správal jednotne a predvídateľne.

Slovníky(vocabularies)

Podľa [18, 14], jednotlivé vyhlásenia, ktoré o zdrojoch robíme síce obsahujú IRIs, avšak dátový model nemá žiadne vedomosti o tom, čo v skutočnosti znamenajú. V praxi sa teda využívajú slovníky, ktoré na sémantickom webe definujú pojmy a vzťahy používané na opis a reprezentáciu nejakej oblasti záujmu. Tú následne využívame v nejakej konkrétnej aplikácii. Slovník je základným stavebným kameňom inferenčných techník na sémantickom webe. Príklady existujúcich slovníkov:

- FOAF, teda „Friend of a Friend“, je jedným z prvých RDF slovníkov používaných na celom svete. Popisuje ľudí, ich aktivity a interakciu s inými ľuďmi. Umožňuje rôznym skupinám ľudí popísať napr. sociálne siete bez potreby centralizovanej databázy.
- SKOS, teda Simple Knowledge Organization System, a je využívaný najmä ľuďmi zaoberajúcimi sa informatikou a knihovníkmi.
- DC: Dublin Core, skupina termínov, ktoré sa používajú na opis digitálnych zdrojov (obrázky, webové stránky,...) alebo fyzických zdrojov (knihy, umelecké diela,...). Zahŕňajú vlastnosti ako napríklad "creator", "publisher" alebo "title".
- RDF Schema, popísaná v podkapitole 1.1.5

Porovnanie slovníkov a ontológií

Úlohou ontológií a slovníkov je teda pomáhať pri spájaní dát na sémantickom webe. Predchádzame tým napríklad prípadu, že v dvoch rôznych dátových množinách by boli použité rovnaké pojmy s rôznym významom.

Čo sa týka rozdielu medzi ontológiami a slovníkmi, podľa [14], neexistuje presný rozdiel, na základe ktorého by sme mohli tvrdiť, že niečo je určite slovník a niečo je určite ontológia. Pojmom ontológia označujeme komplexnejšiu množinu dát, a teda za hlavný rozdiel by sa dala považovať úroveň abstrakcie a vzťahov vrámci obsahu. Pri tvorbe ontológií sa využívajú existujúce slovníky na zníženie úsilia, ktoré musíme vynaložiť na budovanie ontológie od nuly.

1.1.5 RDF Schema

RDF Schema [9] je slovníkom pre modelovanie RDF dát. Poskytuje spôsob na opísanie skupín zdrojov, ktoré spolu súvisia a vzťahov medzi týmito zdrojmi. Triedy a vlastnosti v RDFS by mohli byť paralelou k triedam a atribútom používaným v objektovo-orientovaných programovacích jazykoch.

Príkladom tried z RDFS je `rdfs:Resource`, čo je trieda, do ktorej patrí každý zdroj. Ďalej trieda `rdfs:Literal`, vyjadrujúca hodnoty ako čísla, či textové reťazce alebo `rdfs:Container`, čo je trieda RDF kontajnerov, ktoré by sme mohli prirovnať k akýmisi poliam alebo množinám v programovacích jazykoch.

Príkladom vlastností z RDFS sú `rdfs:subClassOf`, teda vlastnosť byť podtriedou, `rdfs:domain`, vlastnosť vyjadrujúca doménu nejakého vzťahu, `rdfs:comment`, popis subjektu alebo `rdfs:seeAlso`, teda nejaké ďalšie informácie o subjekte.

1.1.6 OWL

Na základe [13] je OWL deklaratívny jazyk, teda logickým spôsobom popisuje stav vecí. Využíva sa na vyjadrenie ontológií a v podstate je rozšírením RDFS (viď. 1.1.5), napr. o termy, ktoré popisujú množinové operácie. Pomocou nástrojov na odvodzovanie je možné prinášať nové informácie a vytvárať závery, ktoré vyplývajú z toho, čo sme zadefinovali. Spôsob, akým sa tieto závery odvodzujú však závisí od konkrétnych implementácií a nie je súčasťou OWL dokumentu. Existujú rôzne syntaxe pre OWL, ktoré slúžia na rôzne účely, napr. RDF/XML syntax, ktorá ako jediná musí byť podporovaná všetkými nástrojmi OWL. Príklady rôznej syntaxe vidíme nižšie a je v nich vyjadrená hierarchia medzi triedami `Dog` a `Pet`, teda vlastnosť byť podtriedou.

Functional-Style Syntax

```
SubClassOf(:Dog :Pet)
```

RDF/XML Syntax

```
<owl:Class rdf:about="Dog">
<rdfs:subClassOf rdf:resource="Pet"/>
</owl:Class>
```

Turtle Syntax

```
:Dog rdfs:subClassOf :Pet .
```

Manchester Syntax

```
Class: Dog
```

```
SubClassOf: Pet
```

OWL/XML Syntax

```
<SubClassOf>
<Class IRI="Dog"/>
<Class IRI="Pet"/>
</SubClassOf>
```

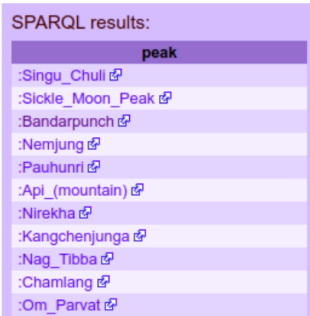
1.1.7 SPARQL

SPARQL, spracovaný na základe zdrojov [4, 2], je jazyk navrhnutý na vytváranie dopytov nad RDF dátami. Kľúčovými slovami v dopytoch sú PREFIX, SELECT, WHERE. Využívať môžeme aj FROM, ORDER BY, LIMIT a podobné výrazy, ktoré sú využívané aj v SQL dopytoch. Podmienky vo WHERE časti píšeme vo forme trojíc, ktoré sú podobné trojiciam v RDF, ale môžu obsahovať premenné. Tie zvyšujú flexibilitu pri porovnávaní s RDF dátami. Takisto sú podporované aj ASK dopyty, ktoré vracajú boolean hodnoty a CONSTRUCT dopyty, pomocou ktorých môžeme vytvárať RDF grafy z výsledkov dopytov.

Ak chceme robiť dopyty nad dátami nepotrebuje žiaden špeciálny softvér, keďže kolekcie dát sú často dostupné prostredníctvom SPARQL endpointov. Je to webová služba, ktorá akceptuje SPARQL dopyty a vracia výsledky. DBpedia je najpopulárnejším SPARQL endpointom.

Nižšie môžeme vidieť dopyt nad endpointom DBpedia (<http://dbpedia.org/snorql/>). Na základe neho dostávame zoznam vrchov, obr. 1.3, ktoré patria do triedy dbo:Mountain a zároveň sa nachádzajú v pohorí Himaláje. Prefix rdf:type, teda príslušnosť k určitej triede, sme nahradili skratkou a. Vo výslednom zozname je každý vrch popísaný príslušajúcim URI, takže sa prostredníctvom neho môžeme dostať k ďalším informáciám o vybranom vrchu.

```
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?peak WHERE {
?peak a dbo:Mountain;
dbo:mountainRange :Himalayas.
}
```



SPARQL results:

peak
:Singu_Chuli ↗
:Sickle_Moon_Peak ↗
:Bandarpunch ↗
:Nemjung ↗
:Pauhunri ↗
:Api_(mountain) ↗
:Nirekha ↗
:Kangchenjunga ↗
:Nag_Tibba ↗
:Chamlang ↗
:Om_Parvat ↗

Obr. 1.3: Výsledok dopytu nad DBpedia

1.1.8 Triplestore

RDF Triplestore je podľa [7] typ grafovej databázy, v ktorej sú uložené sémantické fakty. Triplestory sú uprednostňované na manažovanie prepojených dát, sú flexibilnejšie a lacnejšie ako relačná databáza. Zvládajú sémantické dopyty a používanie odvodzovania na zistenie nových informácií z už existujúcich vzťahov.

1.2 Použité technológie

Pre túto prácu sú kľúčové technológie Java, pričom sú využívané frameworky Apache Jena a SpringBoot, a React, ktorý je knižnicou JavaScriptu.

1.2.1 Apache Jena

Apache Jena [8] je voľne dostupným Java frameworkom, ktorý umožňuje budovať sémantický web (Sémantický web 1.1) a aplikácie, využívajúce prepojené dáta (Linked data 1.1.1). Poskytuje API rozhrania, ktoré umožňujú prácu s RDF dátami. Nižšie sú uvedené tie časti frameworku, ktoré sú v práci použité.

Fuseki

Apache Jena Fuseki je SPARQL server. Podporuje dopyty a aktualizácie RDF dát a môže byť spustený ako samostatný server, služba operačného systému a ako webová aplikácia. Server je integrovaný s TDB, pričom poskytujú trvalú transakčnú ukladaciu vrstvu. Poskytuje protokol SPARQL pre dopytovanie a update cez HTTP.

TDB2

TDB2 je komponent Jena, ktorý umožňuje ukladať RDF dáta a vytvárať nad nimi dopyty. Podporuje celú škálu Jena API rozhraní. Pri prístupe pomocou transakcií sú súbory dát chránené pred poškodením, neočakávaným ukončením procesu alebo zlyhaním systému. TDB2 nie je kompatibilná s TDB1. Líšia sa napr. v tom, že TDB2 poskytuje neobmedzenú veľkosť transakcií, prenášanie modelov a grafov cez transakcie, či lepšiu kontrolu transakcií.

RDF API

RDF API je hlavné API rozhranie, ktoré slúži na vytvorenie, čítanie a zápis do RDF grafov. RDF grafy sú v Jene označované ako Modely (trieda Model). Model poskytuje veľké množstvo metód, ktoré uľahčujú vytváranie aplikácií pracujúcich s RDF dátami. Model môže byť na začiatku prázdny alebo vytvorený z dát nachádzajúcich

sa v súboroch, databázach, či URL adresách. Keďže Model predstavuje RDF graf, tak pozostáva z tvrdení, pričom každé tvrdenie je vyjadrené vo forme trojice. Tvrdenie v Jene reprezentuje trieda `Statement`. Dodržaním RDF špecifikácie môže ako subjekt v inštancií triedy `Statement` vystupovať iba inštancia triedy `Resource`, ako predikát iba inštancia triedy `Property`, pričom ide o podtriedu triedy `Resource`. V prípade objektu ide o inštanciu triedy `RDFNode`, tá je nadtriedou triedy `Resource` a triedy `Literal`, keďže hodnotou vlastnosti, teda objektom, môže byť buď ďalší zdroj alebo literál.

Ontology API

Ontology API umožňuje vývoj ontológií, pričom je jazykovo neutrálny, teda napríklad `OntClass` môže predstavovať OWL triedu alebo RDFS triedu. `OntModel` je rozšírením základnej Jena triedy `Model`. Umožňuje vytvárať triedy, vlastnosti, či inšcie (Individuals) tried z ontológií. Jednou z hlavných výhod na budovaní aplikácie založenej na ontológií je následná možnosť využitia nástrojov na odvodzovanie.

ARQ

ARQ je dopytovací nástroj pre Jenu, ktorý podporuje SPARQL (1.1.7). Sú v ňom podporované `SELECT`, `CONSTRUCT`, `DESCRIBE` a `ASK` dopyty. Nachádza sa v balíčku `org.apache.jena.query`, pričom ten obsahuje kľúčové triedy, ktorými sú `Query`, kontajner pre všetky podrobnosti o dopyte. `QueryExecution` a `QueryExecutionFactory` sú triedy, ktoré reprezentujú vykonanie dopytu. Triedami, ktoré pracujú s dátami, ktoré na základe dopytu dostaneme sú `QuerySolution`, reprezentujúca jeden z vrátených výsledkov, `ResultSet`, ktorá je iterátorom a máme prostredníctvom nej prístup k množine všetkých `QuerySolution` a `ResultSetFormatter`, ktorý umožňuje vrátiť výsledok v forme napr. JSON, text alebo XML.

RDF Connection

`RDF Connection` poskytuje jednotný súbor operácií pre prácu s RDF dátami prostredníctvom SPARQL. Rozhranie je jednotné, teda rovnaké rozhranie sa vzťahuje na lokálne údaje aj na vzdialené údaje.

Query Builder

`Query Builder` umožňuje vytvárať `Ask`, `Construct`, `Select` a `Update` dopyty. Používateľ teda nie je nútený využívať `StringBuilder` alebo podobné spôsoby na tvorbu dopytov. `Builder` pre každý druh dopytu má skupinu metód na jeho vytvorenie, pričom každá metóda vracia builder, kvôli čomu je jednoduché zreťazovať jednotlivé metódy.

1.2.2 Spring Boot

Spring Boot je voľne dostupný Java framework, ktorý sa používa na vytváranie mikroservisov. Pomocou neho dokážeme vytvoriť samostatnú a jednoducho spustiteľnú aplikáciu, s minimálnym množstvom konfigurácií.

1.2.3 React

React je knižnica JavaScriptu na vytváranie používateľských rozhraní. Používa sa na vytváranie webových alebo mobilných "single-pageäplikácií. Umožňuje nám vytvárať opakovane použiteľné komponenty používateľského rozhrania.

Komponent v Reacte môže byť definovaný ako funkcia, čo sa pokladá za najjednoduchší spôsob definovania komponentu, alebo ako trieda. Každý komponent má props, čo sú dáta ktoré mu boli posunuté pri vytváraní. Ďalej má stav, čo je jednoduchý objekt, ktorý sa zvyčajne označuje ako state. V prípade potreby zmeniť stav, musíme využiť funkciu `setState()`, keďže stav sa priamo meniť nesmie. Pri každej zmene stavu dochádza k opätovnému volaniu funkcie `render()`, ktorá zodpovedá za zobrazenie komponentu. V prípade, ak ide o komponent vo forme funkcie, potom takýto komponent funkciu `render` nezahŕňa, ale iba v časti `return` vracia potrebné časti na jeho vykreslenie.

Výhodou Reactu je virtual DOM (virtuálna reprezentácia objektového modelu dokumentu). Všetky zmeny sa najskôr aplikujú na virtuálny DOM a potom pomocou algoritmu sa vypočíta minimálny rozsah potrebných operácií. Na základe toho sa nakoniec aktualizuje skutočný DOM. Takýmto spôsobom sa zabezpečí minimálna časová náročnosť, keďže prerenderujú sa iba prvky, ktoré sa skutočne zmenili.

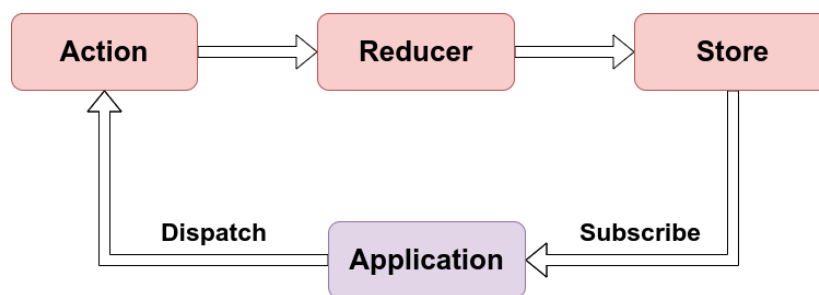
1.2.4 Redux

Redux [6] je knižnica JavaScriptu, ktorá umožňuje spravovať stav našej aplikácie, pričom pod stavom aplikácie si môžeme predstaviť globálny objekt, ktorý uchováva informácie, ktoré v aplikácii používame, napr. pri vykresľovaní komponentov. Troma hlavnými časťami Reduxu sú:

1. Actions (akcie) - jednoduché JavaScript objekty, ktoré musia mať vlastnosť `type`. `Type` je definovaný ako string konštanta a vyjadruje typ vykonávanej akcie. `Action creator` sú funkcie, ktoré vytvárajú akcie. Akcie sú jediným zdrojom informácií pre náš store a umožňujú nám do neho posilať tieto informácie využitím príkazu `store.dispatch()`.
2. Reducer - špecifikuje ako sa stav aplikácie zmení v dôsledku jednotlivých akcií(actions), ktoré sú poslané do storu. Počet reducerov môže byť ľubovoľný,

každý môže spravovať ľubovoľnú časť globálneho statu, avšak nakoniec ich musíme spojiť pomocou funkcie `combineReducers()`.

3. Store - objekt, v ktorom sú uložené informácie, pričom tento stav je jeden pre celú aplikáciu. Tento stav sa nedá priamo meniť, a preto sa vždy vytvorí nový objekt, prepočítame nový stav aplikácie a pôvodný stav aktualizujeme novo vytvoreným objektom.



Obr. 1.4: Tok dát v Reduxe

Dátový tok v Reduxe môžeme vidieť na obr.1.4. V našej aplikácii najprv niečím (napr. stlačením tlačidla) vyvoláme akciu pomocou `store.dispatch()`. Táto akcia je ďalej posunutá reduceru, ktorý vytvorí nový stav, ktorý si následne store uloží. Na zmenu storu zareaguje aj naša aplikácia a aktualizuje používateľské rozhranie.

Redux Thunk - je middleware [5], ktorý umožňuje vytvárať asynchrónne akcie a komunikovať so storom, keďže štandardné akcie v Reduxe sú odosielané synchronne.

1.2.5 Axios

Axios [19] je promise-based HTTP klient, ktorý poskytuje ľahko použiteľné rozhranie API. Umožňuje nám jednoducho vytvárať requesty (požiadavky), ako sú napr. GET, PUT, POST a DELETE. Response (odpoveď) je objekt, ktorý dostávame na našu požiadavku a obsahuje data, status, statusText, headers, config a request. Zachytenie požiadaviek a odpovedí sa vykonáva prostredníctvom `then` alebo `catch`. Pri použití `catch` bude odpoveď dostupná prostredníctvom `error` objektu. Axios štandardne serializuje objekty JavaScriptu na JSON.

1.2.6 Bootstrap

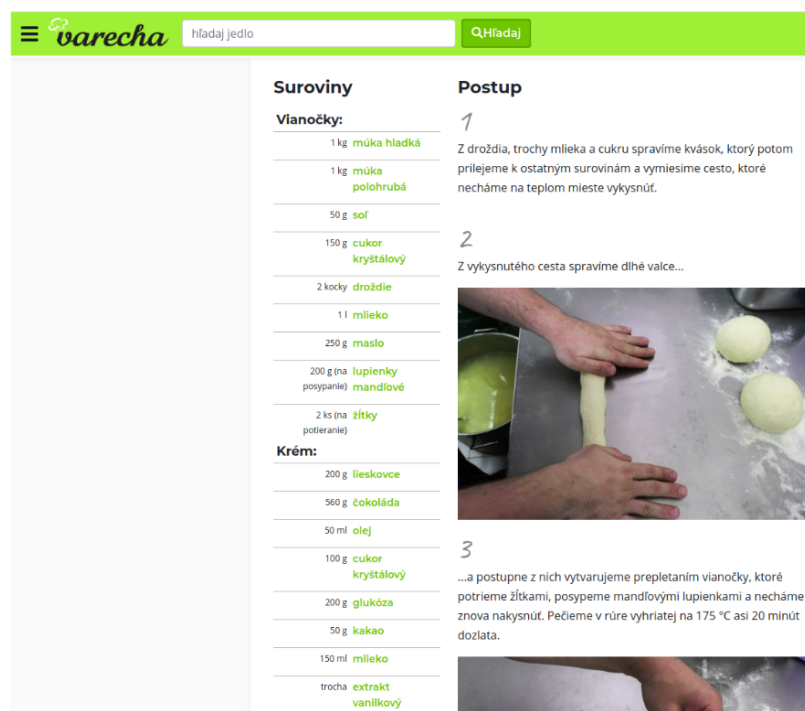
Bootstrap [20] patrí k voľne dostupným front-end knižniciam. Bol vytvorený spoločnosťou Twitter v roku 2011. Obsahuje šablóny založené na HTML a CSS, ktoré slúžia na úpravu typografie, formulárov, tlačidiel, navigácie a rôznych ďalších komponentov.

Hlavnou výhodou je zjednotenie spracovania ľubovoľného používateľského rozhrania webovej aplikácie. Popodporuje aj responzívny design, čo znamená, že rozloženie stránky sa dynamicky prispôsobuje s ohľadom na zariadenie, ktoré práve používame.

1.3 Existujúce riešenia

Táto kapitola sa venuje už existujúcim riešeniam daného problému. Uvádzam bakalársku prácu, ktorá sa touto témou zaoberala minulý rok a vybrala som si jednu slovenskú a jednu zahraničnú webovú aplikáciu, na ktorých je k dispozícii veľké množstvo receptov nie len ako jeden blok textu, ale ako dáta, s ktorými je možné pracovať.

1.3.1 Varecha



Varecha hľadaj jedlo

Suroviny

Vianočky:

- 1 kg múka hladká
- 1 kg múka polohrubá
- 50 g soľ
- 150 g cukor kryštálový
- 2 kocky droždie
- 1 l mlieko
- 250 g maslo
- 200 g (na posypanie) lupienky mandľové
- 2 ks (na potieranie) žltky

Krém:

- 200 g lieskovce
- 560 g čokoláda
- 50 ml olej
- 100 g cukor kryštálový
- 200 g glukóza
- 50 g kakao
- 150 ml mlieko
- trocha extrakt vanilkový

Postup

1
Z droždia, trochy mlieka a cukru spravíme kvások, ktorý potom prilejeme k ostatným surovinám a vymiesime cesto, ktoré necháme na teplom mieste vykysnúť.

2
Z vykysnutého cesta spravíme dlhé valce...

3
...a postupne z nich vytvárame prepletaním vianočky, ktoré potrieme žltkami, posypeme mandľovými lupienkami a necháme znova nakysnúť. Pečieme v rúre vyhriatej na 175 °C asi 20 minút dozlata.

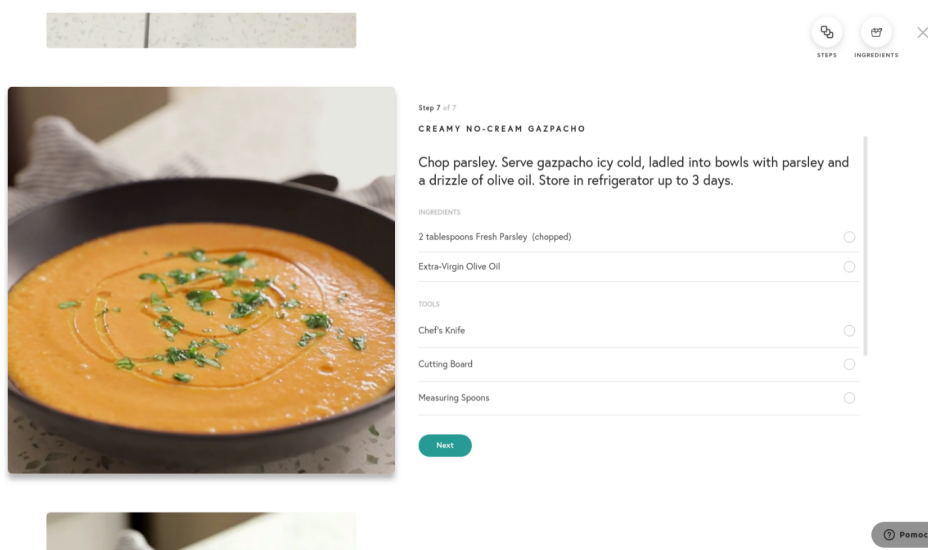
Obr. 1.5: Recept na vianočku, zdroj: [15]

Tento slovenský internetový portál [15] vznikol v roku 2009, pričom stihol zverejniť viac než 60 tisíc receptov. Pre neprihlásených používateľov je možnosť vyhľadávať recepty podľa druhov jedál, krajín, ingrediencií, atď. Pre prihlásených je dostupné aj vytváranie vlastného receptu, pridávanie komentárov, vlastných poznámok.

Vytváranie vlastného receptu, je zaujímavejšie, keďže nám vlastne predstavuje jeho jednotlivé časti. Samostatne sú vyplňané informácie ako názov, úvodný popis, čas varenia, či tepelnej úpravy, množstvo porcií a podobne. Zaradzovanie do rôznych kategórií funguje na základe toho, že autor sám usúdi, kam sa recept hodí a danú kategóriu

zaklikne. V prípade surovín je možnosť rozdeľovať ich na základe toho, v ktorej časti jedla sa využijú, pridávať ich množstvá a jednotky. Pri udaní názvu ingrediencie, ktorý web dokáže rozpoznať, je možné sa dostať kliknutím na ňu k jej nutričným hodnotám alebo ďalším jedlám, v ktorých sa využíva. Samotný postup je odkrokovovaný, avšak nikdy nie je presnejšie povedané, ako by mal daný krok vyzerieť, a teda je to na úvážení autora, čo tam uvedie. Ukážku receptu so surovinami a postupom vidíme na obr. 1.5.

1.3.2 Yummly



Obr. 1.6: Recept na gazpacho, zdroj: [22]

Podľa informácií na stránke [22] je tu k dispozícii viac ako 2 milióny receptov a má približne 26 miliónov používateľov. V prípade, že sme prihlásení, môžeme si v profile nastavovať rôzne preferencie, diéty, alergie a chute. Na základe všetkých týchto informácií nám aplikácia dokáže poskytovať stále relevantnejšie údaje bez toho, aby sme to museli vždy sami vyhľadávať.

Niektoré ingrediencie sa dajú jedným kliknutím na nich dohľadať priamo v obchode. Čo sa týka vyhľadávania, okrem bežných parametrov, ktoré sme si uviedli napr. aj pri portály Varecha [15], je možné vyhľadávať jedlá na základe spôsobu prípravy. Popisy postupov boli takisto podobné, pri videorecepte však boli aj samostatne uvedené nádoby, či iné zariadenia a zároveň aj ingrediencie potrebné na vykonanie daného kroku, tak ako vidíme na obr. 1.6. Nachádzajú sa tu však aj recepty, ktoré ako postup obsahovali odkaz na článok alebo inú stránku.

1.3.3 Bakalárska práca Inteligentný receptár na báze prepojených dát

Téme vytvárania receptov sa venovala minuloročná bakalárska práca [17]. Jej cieľom bolo najmä navrhnúť a implementovať webovú aplikáciu, ktorá spravuje a zobrazuje dáta nad štandardom sémantického webu.

Došlo k preskúmaniu existujúcich LOD zdrojov venujúcich sa tejto tématike, a zároveň k návrhu vlastnej ontológie. Jej hlavnou triedou bola trieda Recipe, ktorá popisovala samotný recept. Ten má dva hlavné atribúty, ktoré tvoria každý recept. Sú nimi ingrediencie a postup. Zoznam ingrediencií, a následne samotná ingrediencia sú reprezentované samostatnými triedami. Každá ingrediencia má nejakú jednotku merania a zároveň množstvo. Tieto vlastnosti sú popísané triedou Mass. Postup je v tejto ontológii reprezentovaný ako pole reťazcov. Ďalšími atribútmi prislúchajúcimi k triede Recipe sú autor, čas prípravy, počet porcií, kategórie, a zároveň hodnotenie, ktoré má tiež samostatnú triedu.

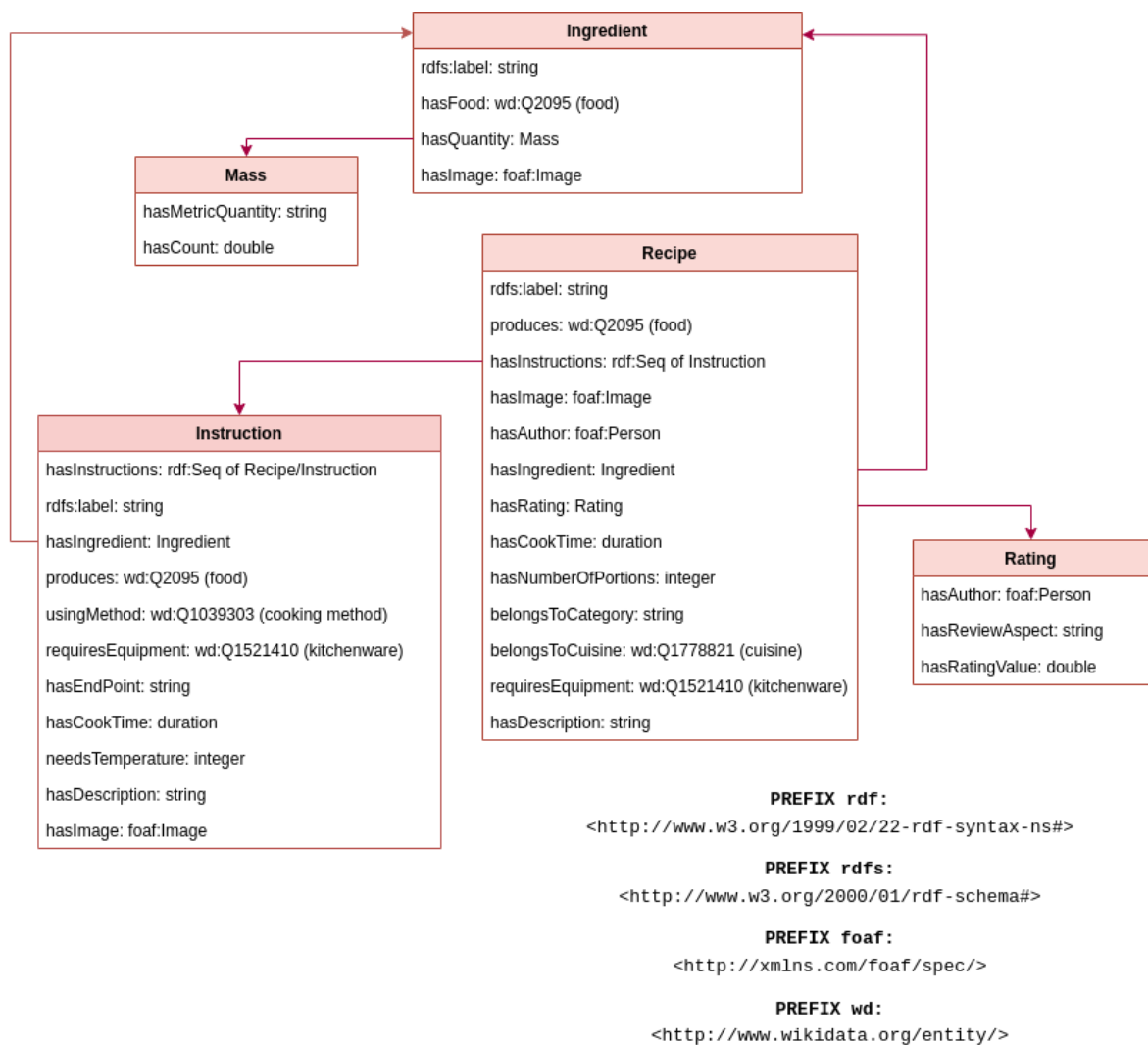
Začiatkové dáta boli čerpané zo stránky DBpedia. Na ich uskladnenie bol využitý triple store TDB, ktorý je komponentom Jena, čo je voľne dostupný Java framework pre vytváranie sémantického webu. Dopyty a aktualizácie boli vykonávané na SPARQL server Fuseki. Ten je spojený s TDB, používaného na trvalé uskladnenie dát. Pri pripojení na server a následné dopytovanie a aktualizáciu dát bola využitá knižnica EasyRdf, ktorá nám umožňuje ľahké vytváranie a prácu s RDF dátami. Okrem informáciách o receptoch a ingredienciách sa v aplikácii využívajú aj prihlasovacie údaje, či informácie o obľúbených receptoch. Z toho dôvodu boli využité dva typy databáz, na oddelenie osobných a verejných dát.

Čo sa týka výslednej aplikácie tá umožňuje používateľovi registráciu, vyhľadávanie receptu na základe rôznych kritérií ako sú napríklad ingrediencie, dĺžka prípravy, kategória alebo zásoby. Ďalej je možné pridávať ingrediencie do chladničky, vytvárať nákupný zoznam, vytvárať a upravovať recepty, a reagovať na ne, teda hodnotiť ich, exportovať do pdf, označovať ako obľúbené a podobne. Niektoré časti funkcionality sú prístupné len prihláseným používateľom.

Kapitola 2

Návrh riešenia

2.1 Návrh ontológie



Obr. 2.1: Grafické vyjadrenie návrhu ontológie o receptoch

2.1.1 Protégé

Na vytvorenie ontológie sme využili Protégé, ktorý je možné stiahnuť na stránke <https://protege.stanford.edu/>. Protégé poskytuje grafické používateľské rozhranie, je to voľne dostupný editor ontológií a framework pre budovanie inteligentných systémov. Umožňuje modelovať triedy, predikáty, či vytvárať obmedzenia na jednotlivé predikáty. Následne je možné uložiť takto vytvorenú ontológiu v niektorej zo syntaxí pre jazyk OWL. Pri tvorbe našej ontológie sme využívali túto základnú funkcionálnosť.

2.1.2 Existujúca ontológia

Naša ontológia nadväzuje na ontológiu o receptoch v minuloročnej bakalárskej práci [17]. Hlavnou úlohou bolo rozšíriť pôvodnú ontológiu o triedu, respektíve triedy, ktoré budú reprezentovať jednotlivé kroky (inštrukcie) v postupe receptu, keďže predtým bol postup iba pole reťazcov. V priebehu tvorby ontológie však došlo aj k ďalším menším zmenám v existujúcich triedach. Zároveň boli z pôvodnej ontológie odstránené triedy ingredientList a Food. Trieda ingredientList iba zapuzdrovala všetky ingrediencie patriace ku konkrétnemu receptu a trieda Food, ktorá vyjadrovala už existujúcu triedu a v našom prípade nebolo potrebné k nej pridávať žiaden ďalší predikát.

2.1.3 Navrhnutá ontológia

Nižšie je popísaná ontológia podľa obr. 2.1. Pri popisovaní ontológie je v zátvorkách uvedený presný názov predikátu tak, ako sa nachádza v ontológii a v jej grafickom vyjadrení na obr. 2.1. V pravom dolnom rohu obr. 2.1 sú uvedené využité prefixy v prípade, že trieda alebo predikát boli prevzaté z už existujúceho slovníka.

Popis tried existujúcich v predchádzajúcej ontológii

Hlavnou triedou je trieda Recipe, ktorá popisuje recept ako celok. Každý recept môže mať priradený obrázok (hasImage), názov vytváraného receptu zrozumiteľný pre používateľa (rdfs:label), autora (hasAuthor), čas potrebný na prípravu (hasCookTime), počet porcií, ktoré daný recept vytvára (hasNumberOfPortions), stručný popis, ktorý však ešte nepatrí k postupu (hasDescription) a možnosť zaradiť recept do nejakej kategórie (belongsToCategory), pričom tento predikát môže byť použitý opakovane v prípade, že recept patrí do viacerých kategórií. Ďalšími predikátmi sú príslušnosť k nejakej národnej kuchyni (belongsToCuisine), požadované kuchynské potreby (requiresEquipment), hodnotenia daného receptu inými používateľmi (hasRating). V tom prípade je objektom inštancia triedy Rating, pričom každé hodnotenie má autora (hasAuthor) a môže obsahovať nejaký slovný komentár (hasReviewAspect) alebo číselne vyjadrenú spokojnosť s receptom (hasRatingValue). Každý recept vytvára nejaké jedlo, ktoré sa

v niektorých prípadoch môže použiť ako ingrediencia v inom recepte, a preto bolo potrebné recept zaradiť aj do triedy Food (produces). Toto zaradenie nám ďalej aj umožňuje vyhľadávať recepty, ktoré vytvárajú rovnaké jedlo.

Predikát umožňujúci priradzovať k danému receptu ingrediencie (hasIngredient) spája triedu Recipe s triedou Ingredient. Každá ingrediencia môže mať názov zrozumiteľný pre používateľa (rdfs:label), obrázok (hasImage), jednotku merania (hasMetricQuantity), množstvo (hasCount), pričom posledné 2 spomenuté predikáty patria triede Mass. Každá ingrediencia je nejaké už existujúce jedlo (hasFood).

Popis triedy Instruction

Každý recept musí mať nejaký postup (hasInstructions). Objektom tohto predikátu je kontajner rdf:Seq, ktorý sa používa v prípadoch, kedy potrebujeme poznať poradie pridávaných položiek. V tomto prípade je možné do kontajnera pridávať inštancie triedy Instruction.

Pri definovaní krokov v postupe receptu sme uvažovali nad viacerými možnosťami ako ich vytvárať. Plánovali sme oddeľovať triedu Description, Stage a Step, vďaka ktorým by sa dal recept pekne rozčleniť. Avšak nie každý recept má takúto štruktúru a pri iných typoch receptov by boli častokrát zbytočné triedy Stage, či Description. V prípade samotného kroku sme uvažovali nad tým, že bude obsahovať triedu Activity, ktorá presne popíše, ktorú surovinu, akým spôsobom, a za akých podmienok bude spracovávať. Toto by síce dokonale zobrazilo každý krok receptu, avšak praktické využitie takto podrobnej ontológie by bolo príliš náročné.

Rozhodli sme sa preto pre možnosť vytvoriť jedinú triedu Instruction. Inštrukcia môže zaobšľávať celú postupnosť krokov k receptu, môže vyjadrovať nejakú časť krokov, ktoré tvoria samostatnú časť jedla, napr. v prípade koláča to je inštrukcia na vytvorenie plnky, cesta, či polevy, alebo môže inštrukcia vyjadrovať jediný krok. Využitie triedy bude teda závisieť od toho, ako bude vyzeráť postup k receptu. Práve kvôli rôznorodosti postupov sme museli vytvoriť dostatočne všeobecnú triedu, aby mohla umožniť rovnako ľahko vytvoriť recept s jedným krokom, ako aj recept s pomerne zložitým postupom, v ktorom sú isté časti receptu oddelené do samostatných celkov. Používateľovi je umožnené priradiť techniku prípravy, ktorú je potrebné využiť pri danom kroku (usingMethod), napr. vyprážanie, pečenie, možnosť priradiť obrázok (hasImage), názov (rdfs:label), teplotu, pri ktorej má byť daná inštrukcia vykonávaná (needsTemperature), požadované kuchynské potreby (requiresEquipment), či potraviny spracovávané v danej časti receptu (hasIngredient). Trieda Instruction ďalej umožňuje vyjadriť čas potrebný na vykonanie nejakého kroku (hasCookTime). V niektorých prípadoch je trvanie kroku vyjadrené slovne, napr. pečieme, kým nezhnedne, vtedy použijeme predikát, ktorý vyjadruje práve prechod ingrediencie do nejakého finálneho stavu (ha-

sEndPoint). Ak ide o skupinu krokov, ktoré vytvárajú celý postup k receptu alebo časť receptu, napr. prípravu cesta pri pečení koláča, využijeme predikát `hasInstructions`. Ten ako objekt obsahuje sekvenciu, pomocou `rdf:Seq`, ďalších inštrukcií alebo receptov v prípade, že by sme sa v nejakej časti receptu chceli odkázať na už existujúci recept. Ak chceme popisovať už koncový krok, ktorý sa skladá iba zo slovného popisu a ďalej sa nečlení, využijeme predikát `hasDescription`. V prípade, že nejaká inštrukcia vytvára jedlo, môžeme využiť predikát `produces`.

2.2 Funkcionalita systému

V tejto podkapitole popisujeme funkcionality systému. Ide o prehľad toho, čo používateľ môže využívať v prípade, že túto webovú aplikáciu navštívi.

1. **Registrácia používateľa** - Používateľ má možnosť sa zaregistrovať vyplnením formulára, kde je nutné uviesť používateľské meno a heslo.
2. **Prihlásenie používateľa** - Prihlásenie do aplikácie prebieha prostredníctvom rovnakého formulára ako registrácia, teda stačí vyplniť používateľské meno a heslo, pričom po prihlásení získa používateľ prístup k dovtedy neprístupným stránkam, ktorými sú vytváranie nového receptu a zoznam jeho vlastných receptov.
3. **Zobrazenie všetkých receptov aj pre neprihlásených používateľov** - Používateľ si môže zobraziť všetky recepty naraz, pričom ku každému receptu bude vidieť jeho názov a popis.
4. **Vyhľadávanie receptov aj pre neprihlásených používateľov podľa:**
 - **názvu** - Používateľ má možnosť vyhľadať recept zadaním jeho názvu. Vyhľadávané sú všetky recepty, ktorých časť názvu sa zhoduje s tým, čo používateľ zadal do vyhľadávacieho okna.
 - **dĺžky prípravy** - Používateľ má možnosť zvoliť si interval dĺžky prípravy jedla, teda iba spodnú hranicu, iba hornú hranicu, alebo obidve naraz.
 - **ingrediencií**
 - **autora**
 - **národnej kuchyne**
 - **spôsobu prípravy jedla**
 - **kuchynských potrieb**
 - **kategórie**

V prípade vyhľadávania podľa ingrediencií, autora, národnej kuchyne, spôsobu prípravy jedla, kuchynských potrieb a kategórie má používateľ možnosť uviesť položky z jednotlivých skupín, ktoré má recept obsahovať aj ktoré obsahovať nemá.

V prípade vyhľadávania podľa ingrediencií, autora, národnej kuchyne, spôsobu prípravy jedla, kuchynských potrieb a kategórie, ak používateľ uvedie jednu alebo viacero položiek, ktoré recept nemá obsahovať, z výsledných receptov budú odstránené všetky, ktoré aspoň jednu z týchto položiek obsahujú.

V prípade vyhľadávania podľa ingrediencií, autora, národnej kuchyne, spôsobu prípravy jedla, kuchynských potrieb a kategórie, ak používateľ uvedie jednu alebo viacero položiek, ktoré recept musí obsahovať má možnosť zvoliť si jeden z dvoch spôsobov vyhľadávania pri každej z týchto skupín. Pri prvom spôsobe sa vo výsledných receptoch objavajú len tie recepty, ktoré obsahujú všetky položky, ktoré používateľ chce zahrnúť zároveň alebo druhý spôsob, pri ktorom obsahujú výsledné recepty aspoň jednu z položiek (napr. ak používateľ chce v recepte ryžu a kuracie mäso, pri prvom spôsobe budú vyhľadávané tie recepty, ktoré obsahujú obe ingrediencie a pri druhom recepty, ktoré obsahujú aspoň jednu z nich).

Položky zadané používateľom budú porovnávané rovnako ako názov v bode 1, teda vybrané, respektíve odstránené budú tie recepty, v ktorých sa časť názvu položky vybranej skupiny v recepte zhoduje s tým, čo používateľ zadal.

5. **Zobrazenie receptu** - Každý používateľ, prihlásený aj neprihlásený, si môže zobraziť podrobnosti o recepte, ktoré autor receptu pri vytváraní zadal. Keďže pri receptoch ide o rôznorodé dáta, ktoré môžu mať uvedené rôzne vlastnosti, zobrazované budú len tie údaje, ktoré používateľ vo formulári pri vytváraní receptu vyplnil. Nevyplnené položky sa pri zobrazení odignorujú. Pri každom recepte sa budú zobrazovať rovnaké, respektíve podobné recepty, na základe jedla, ktoré autor uviedol pri vytváraní receptu pri vlastnosti produces.
6. **Pridanie vlastného receptu** - Každý prihlásený používateľ má možnosť vytvoriť vlastný recept, ktorého ako autor bude figurovať, teda každý recept má práve jedného autora. Povinnými poliami pre vytvorenie receptu sú názov, popis, jedlo, ktoré recept vyprodukuje, pričom to musí byť jedlo zahrnuté v databáze, aspoň jedna ingrediencia, a aspoň jeden krok v postupe. Ďalej je možné uviesť počet porcií, čas potrebný na prípravu, kategórie a národné kuchyne do ktorých recept patrí, kuchynské potreby nutné na prípravu a spôsoby prípravy. Pri vyplňaní postupu receptu je možné uvádzať kroky aj podkroky. Ak má krok podkroky, v tom prípade je nutné uviesť názov celého kroku. Používateľ má možnosť uviesť pri jednotlivých krokoch a podkrokoch aj ingrediencie, spôsoby prípravy,

kuchynské potreby, čas potrebný na prípravu alebo popis stavu, v ktorom daný krok, respektíve podkrok končí, teplotu a takisto jedlo, v prípade, že krok vytvára niečo, čo môžeme označiť za hotové jedlo.

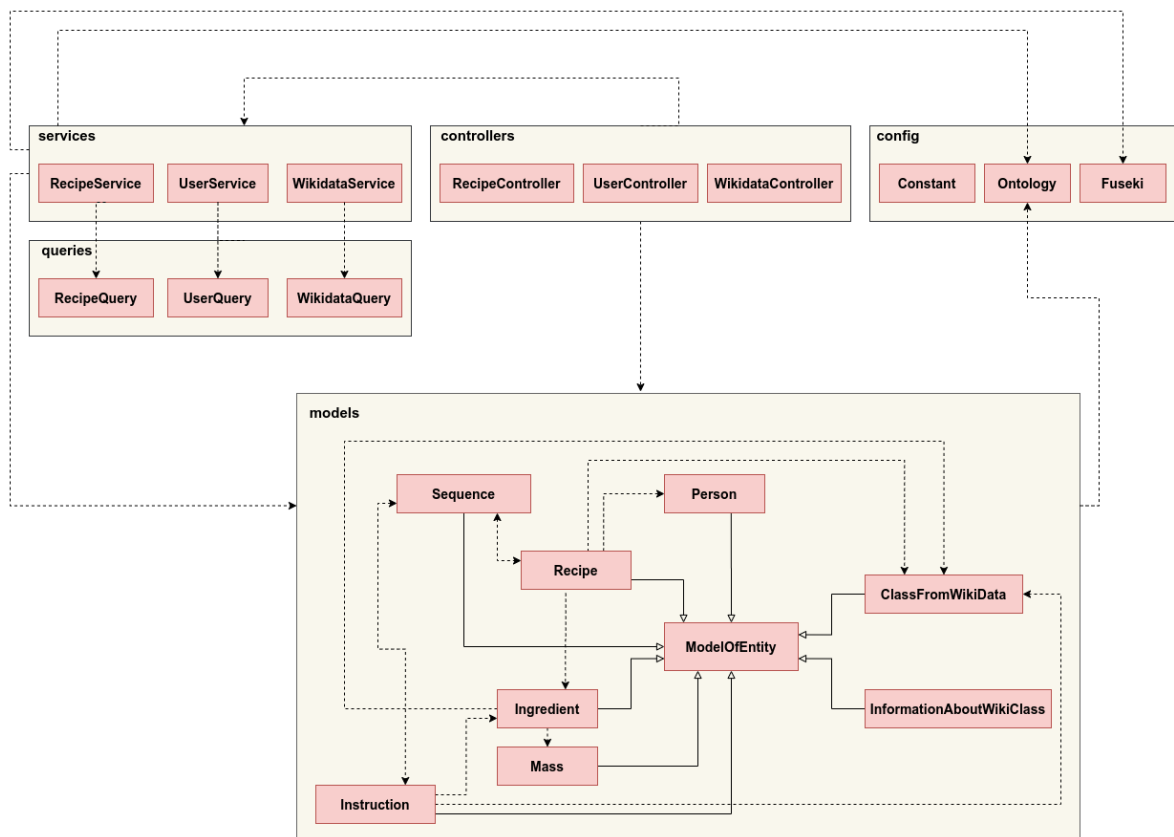
7. **Úprava vlastného receptu** - Jednotlivé časti receptu môže používateľ upraviť po kliknutí na tlačidlo Edit, ktoré sa mu však zobrazí iba v prípade, že sa nachádza v časti stránky so zoznamom jeho vlastných receptov. V tom prípade sa používateľovi zobrazí recept v takej forme ako keď je vytváraný, avšak všetky polia budú predvyplnené. Ak po úprave ostanú vyplnené všetky povinné polia, úprava sa bude považovať za úspešnú.
8. **Možnosť zobraziť si všetky národné kuchyne, metódy prípravy, kuchynské potreby a jedlá (ingrediencie) zahrnuté v databáze** - Aj neprihlásený používateľ si môže zobraziť jeden zo zoznamov, ktorý obsahuje všetky položky z vybranej kategórie. Tieto položky (entity) sú prebrané z wikidata, pričom ich používateľ využíva pri vytváraní receptu v poliach, v ktorých pridáva názov ingrediencie, spôsobu prípravy, kuchynskej potreby a národnej kuchyne.
9. **Možnosť zistiť bližšie informácie o jedle (ingrediencií), národnej kuchyni, spôsobe prípravy a vybranej kuchynskej potrebe** - Ak sa používateľ dostane do zoznamu nejakej z uvedených kategórií (funkcionalita z bodu 8), o každej z položiek má možnosť dozvedieť sa viac. Po kliknutí na položku sa mu zobrazia informácie, ktoré sú o nej dostupné na stránke wikidata vo forme tabuľky s dvoma stĺpcami property a value. Každá vlastnosť (property) a jej hodnota (value) je odkazom, takže sa používateľ dostane aj k ďalším podrobnejším informáciám, avšak bude už presmerovaný priamo na stránku wikidata, keďže to budú informácie, ktoré nie sú priamo viazané na recepty.
10. **Odkazy pri prihlasovaní používateľa, detaile a vytváraní receptu** - Možnosť dozvedieť sa viac o nejakej entite nie je možné iba vyššie uvedeným spôsobom (funkcionalita z bodu 9). Pri vytváraní receptu aj pri prihlasovaní používateľa sú vo formulári pri všetkých poliach uvedené názvy, ktoré používateľovi hovoria, čo má do poľa zadať. Tieto názvy sú však aj odkazmi na ontológie, v ktorých je daná vlastnosť zadaná, a teda kliknutím na nich sa používateľ dostane k ich podrobnejšiemu popisu. Pri detaile receptu, sa tu okrem odkazov na ontológie nachádzajú aj odkazy na triedy z wikidata, v prípade, že niektorá časť receptu tieto triedy obsahuje.

2.3 Návrh databázy aplikácie

Sú vytvorené dve databázy. Jedna databáza obsahuje informácie o používateľoch aplikácie, teda ich prihlasovacie údaje. Druhá databáza ukladá všetky recepty s informáciami, ktoré k nim boli uvedené. Informácie sú do databázy vkladané na základe vytvorenej ontológie (2.1.3). Ďalej táto databáza obsahuje aj všetky entity importované z wikidata, teda ingrediencie, kuchynské potreby, spôsoby prípravy jedla a národné kuchyne. Ku každej z týchto 4 skupín je v databáze uložená vlastnosť `rdfs:label`, kvôli tomu, aby používateľovi mohla byť táto entita zobrazená s názvom, ktorému porozumie a zároveň s vlastnosťou `rdfs:comment`, ktorý hovorí o tom, do ktorej z týchto 4 skupín patrí.

2.4 Návrh backendu aplikácie

Backend aplikácie zodpovedá za dopyty vytvárané nad databázou, umožňuje vytvárať triedy, ktoré sa budú zhodovať s triedami vo vytvorenej ontológii (2.1.3) a poskytuje REST API rozhranie. Na obr. 2.2 vidíme vizualizáciu tried a balíčkov, ktoré budú zodpovedať za túto funkcionality. Práca s databázou sa nachádza v balíčku `services`,



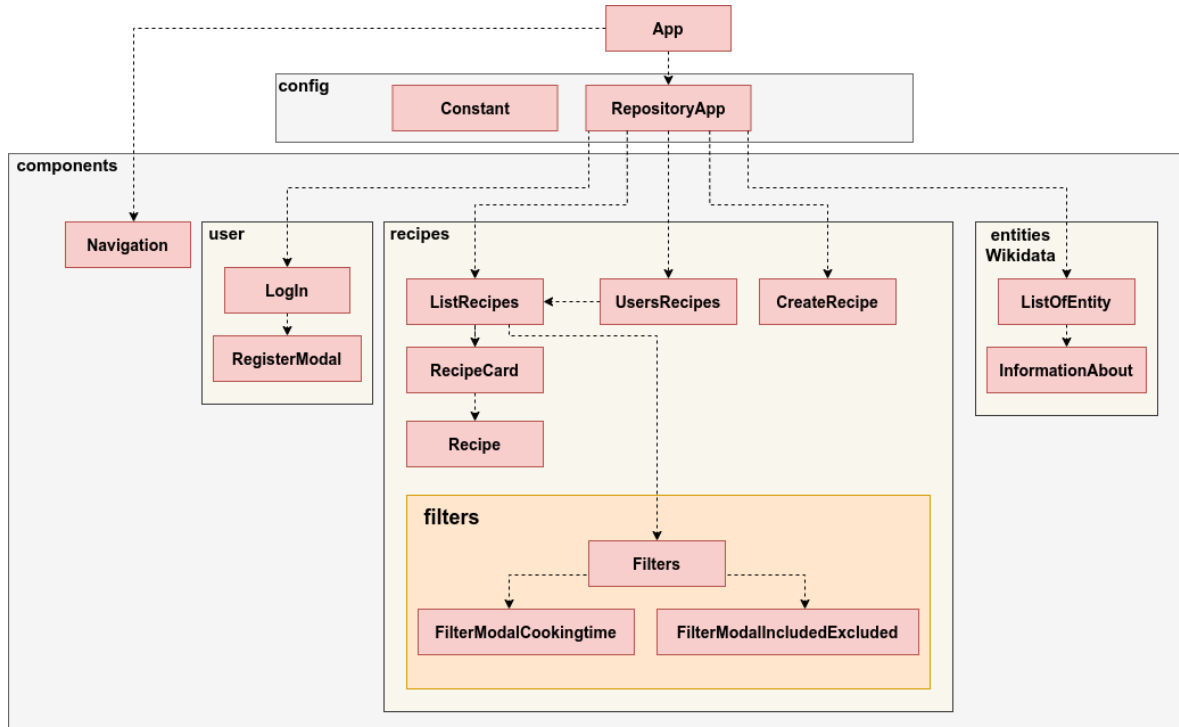
Obr. 2.2: Vizualizácia štruktúry balíčkov a tried na backende

pričom sú v ňom oddelené tri súbory na základe toho, aká entita má byť výsledkom

ich dopytu. Tento balíček využíva balíček queries, ktorý zodpovedá za vytváranie potrebných dopytov a triedu Ontology a Fuseki. Fuseki zodpovedá za pripojenie sa na databázu, pričom toto pripojenie je opakovane používané a Ontology obsahuje ontológiu na základe ktorej s dátami pracujeme. REST API je vytvárané v balíčku controllers, čo teda znamená, že bude obsahovať triedy s metódami, ktoré budú obsluhovať URL adresy pre requesty, ktoré budú prichádzať. V balíčku models sa nachádzajú triedy, ktoré reprezentujú triedy z ontológie, pričom sú využívané v services a controllers ako návratové typy.

2.5 Návrh frontendu aplikácie

Frontend vytvára používateľské rozhranie a teda umožňuje komunikovať používateľovi s aplikáciou. Grafické rozhranie aplikácie bude pozostávať z viacerých komponentov, ktoré sú v balíčku components. Sú to komponenty, ktoré sa používateľovi môžu zobrazíť v prehliadači pričom ich úlohou je len čítať aktuálny stav aplikácie, na základe toho zobrazovať jednotlivé komponenty, a prípadne na základe určitej akcie od používateľa vytvoriť reakciu, teda spustiť funkciu, ktorá dokáže stav aplikácie zmeniť. Pre jednoduchšie priblíženie toho, aké komponenty v aplikácii vystupujú a ako sú medzi sebou prepojené si môžeme pozrieť obr. 2.3. Jednotlivé komponenty sú roztriedené



Obr. 2.3: Vizualizácia štruktúry balíčkov, komponentov a ich vzťahov na frontende

do balíčkov na základe funkcionality, ktorú plnia. Balíček user dáva možnosť registrácie alebo prihlásenia sa používateľovi, pričom poskytuje 1. a 2. bod z funkcionality

systemu (2.2). Zobrazovanie zoznamov receptov, ich detailov, či ich vytváranie umožňuje balíček recipes, pričom je tu zahrnutá funkcionálnosť (2.2) od 3.bodu až po bod 7. Filtrovanie receptov z bodu 4 je umožnené vďaka balíčku filters. Balíček entitiesWikidata zodpovedá za funkcionálnosť systému(2.2) z bodov 9 a 10, teda vďaka týmto komponentom má používateľ možnosť zobraziť si všetky triedy prevzaté z wikidata, prípadne detailné informácie o nich. Presmerovávanie medzi stránkami je vytvorené v RepositoryApp, pričom navigácia umožňuje používateľovi sa medzi týmito stránkami jednoducho prepínať. V súbore Constant sú uložené konštanty, ku ktorým má prístup celá aplikácia a môže ich v prípade potreby využívať.

Kapitola 3

Implementácia

3.1 Implementačný jazyk

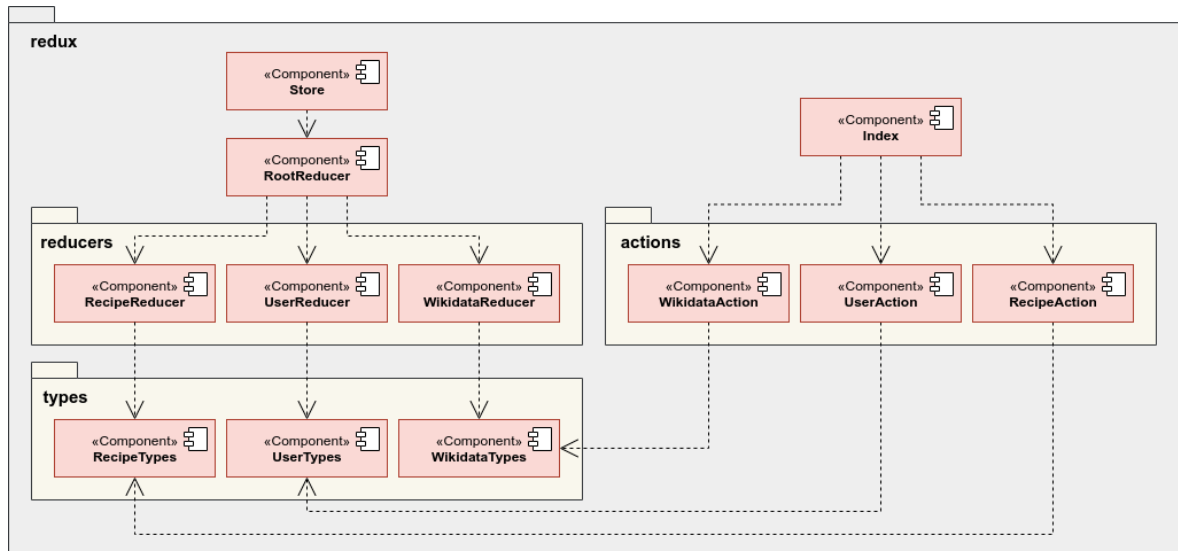
Backend aplikácie pracuje v jazyku Java. Keďže potrebujeme vytvoriť REST API, na ktoré bude možné posilať requesty z frontendu, na túto úlohu využijeme Spring Boot. Knižnicou, ktorú sme si vybrali pre prácu s RDF dátami v jazyku Java je Apache Jena. Tvorí ju viacero častí, ktoré spolu, pričom kľúčové časti pre našu aplikáciu sme uviedli vo východiskovej kapitole (1.2.1).

Frontend aplikácie je tvorený v knižnici React, ktorý patrí medzi najrýchlejšie frameworky vo svojej kategórii, najmä kvôli tomu, že ak dôjde k zmene vlastností nejakého vnoreného komponentu, nie je nutné prepočítavať a znovu vykresľovať všetky nadradené komponenty, ale zmeny sa vykonajú iba v tých komponentoch, v ktorých je to nevyhnutné. Na jednoduché dizajnovanie komponentov je využitá knižnica Bootstrap. Pri vytváraní formulárov je použitá knižnica Formik, ktorá uľahčuje prácu s formulármi v Reacte a zároveň je možné ju použiť s Yup, čo je nástroj na vytváranie validačnej schémy pre JavaScript, a teda je jednoduchšie kontrolovať vstup od používateľa.

3.2 Správa stavu frontendu aplikácie

Celá webová aplikácia sa v Reacte skladá z komponentov. V prípade ak potrebujeme medzi komponentmi vzájomne komunikovať je to možné zabezpečiť pomocou parametrov. Posilať si všetky informácie v parametroch však nie je vždy jednoduché, najmä v prípade, ak všetky komponenty nie sú hierarchicky usporiadané je potrebné medzi dvoma komponentami na jednej úrovni zdieľať rovnakú informáciu. Pre jednoduchšie a prehľadnejšie spravovanie stavu celej aplikácie sme využili Redux.

Ako vidíme na obr. 3.1, balíček redux je rozdelený do 3 ďalších balíčkov, pričom každý z týchto balíčkov obsahuje práve 3 súbory. Je to z toho dôvodu, že každý zodpovedá za jednu z častí aplikácie, teda za používateľa (za prihlásenie, informácie o ňom),



Obr. 3.1: Vizualizácia štruktúry balíčkov a súborov pracujúcich so stavom aplikácie

recepty (jednotlivé filtre, ktoré je možné nastaviť, zoznam práve zobrazených receptov, detail o recepte) alebo za informácie o triedach z wikidata, ktoré sú súčasťou receptov. V balíčku types sú konštanty, konkrétne sú to typy akcií, ktoré sú volané, aby zmenili stav aplikácie. V balíčku reducers, obsahuje každý reducer, čo je funkcia s parametrami, obsahuje definíciu toho, aká zmena má v stave nastať v prípade, ak dôjde k niektorému z typov akcií, ktoré sú zadefinované v balíčku types. Všetky reducers sú skombinované do jedného hlavného reduceru, ktorý je následne použitý v súbore Store. Tento súbor vytvára akýsi "sklad" všetkých informácií, ktorý môže čítať akýkoľvek komponent z aplikácie. V balíčku actions sú akcie, ktorými ako jedinými môžeme meniť stav aplikácie z nejakého komponentu, pričom tieto akcie sú všetky dostupné z jediného súboru index v balíčku redux.

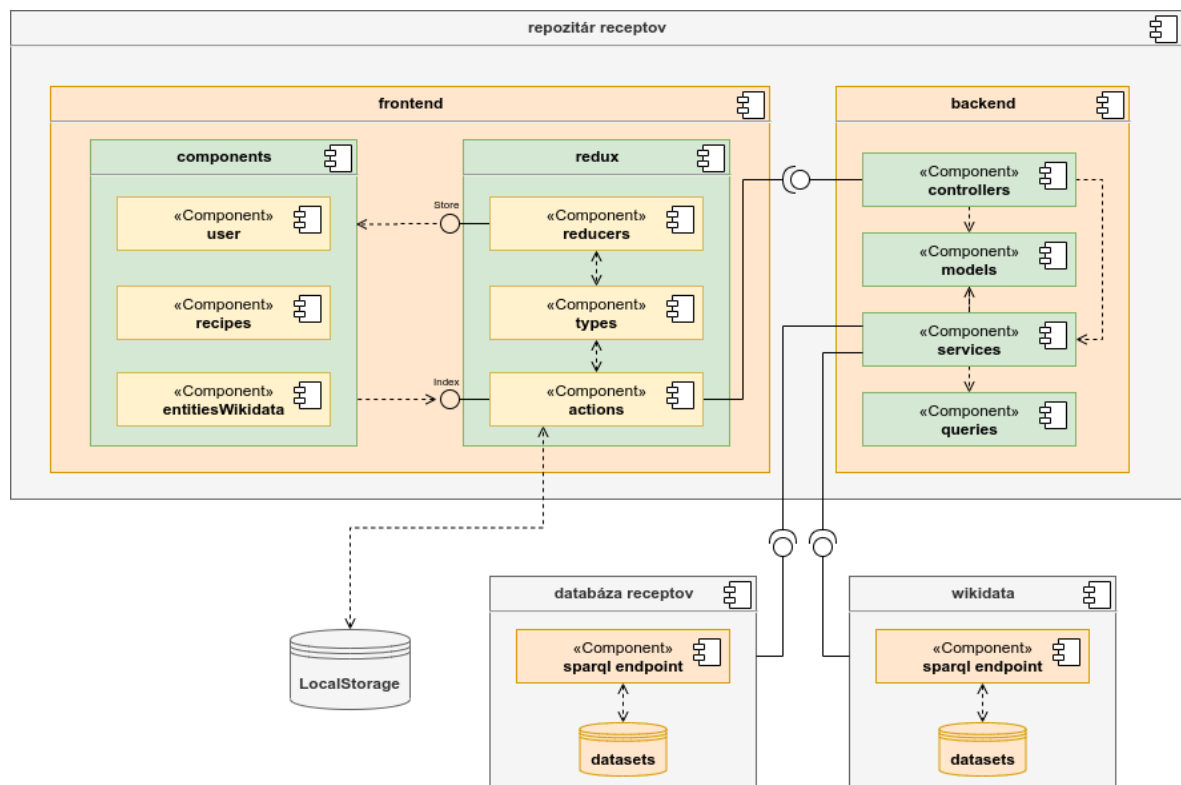
Keďže potrebujeme dispatch-ovať (spúšťať) aj asynchrónne akcie, v našom prípade konkrétne pri volaní REST API, využívame aj `redux-thunk`.

Stav aplikácie ostáva zachovaný iba do času, kým nedôjde k znovu načítaniu stránky alebo presmerovaniu na inú stránku aplikácie. To spôsobuje problém napr. pri prihlásení používateľa, pričom pri prechode na novú stránku sa stav, ktorý si zachovával informáciu o používateli, vynuluje. Z toho dôvodu sme museli využiť `local storage`, ktorá si informácie o používateli dokáže uchovať aj pri prepínaní sa medzi stránkami.

3.3 Prepojenie frontendu, backendu a databázy

Každá z častí, ktorú sme popísali v časti Návrh (kap. 2) je samostatným komponentom. Požadovanú funkcionality však bude celá aplikácia plniť len v prípade, že všetky komponenty budú vzájomne prepojené. Toto prepojenie je zachytené na diagrame z

obr. 3.2.



Obr. 3.2: Diagram vyjadrujúci prepojenie jednotlivých častí aplikácie

Frontend posiela pomocou knižnice axios requesty na backend. Tieto requesty sú posielané z balíčka actions v reduxe. Na strane backendu sú tieto requesty spracovávané v balíčku controllers, v ktorom metódy obsluhujú URL adresy pre jednotlivé requesty. Odpovede, ktoré tieto metódy posielajú vo forme JSON sú získavané z databázy. S databázou komunikujú iba metódy balíčka services. Tie vytvárajú dopyty, ktoré sú posielané na sparql endpointy pomocou triedy RDFConnection, ktorá je súčasťou Apache Jena. Sparql endpoint potrebný pre našu aplikáciu poskytuje stránka wikidata a zároveň server Apache Fuseki, ktorý nám poskytuje triplestore, v ktorom si udržíme dáta o receptoch, ktoré sme vytvorili.

Literatúra

- [1] *Semantic web for the working ontologist modeling in RDF, RDFS and OWL*. Denise E. M. Penrose, 2008.
- [2] *Learning SPARQL*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2011.
- [3] *Linked Data: Evolving the Web into a Global Data Space (1st edition). Synthesis Lectures on the Semantic Web: Theory and Technology*. Morgan & Claypool, 2011.
- [4] SPARQL 1.1 overview. W3C recommendation, W3C, March 2013. <http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/>.
- [5] redux thunk, 2020.
- [6] Dan Abramov and the Redux documentation authors. redux, 2020.
- [7] Ontotext AD. What is rdf triplestore?, 2019.
- [8] The Apache Software Foundation. Apache jena, 2020.
- [9] Ramanathan Guha and Dan Brickley. RDF schema 1.1. W3C recommendation, W3C, February 2014. <http://www.w3.org/TR/2014/REC-rdf-schema-20140225/>.
- [10] Michael Hausenblas. 5 star open data, 2020.
- [11] Facebook Inc. React, 2020.
- [12] Craig Knoblock. The semantic web, 2020.
- [13] Peter Patel-Schneider Bijan Parsia Markus Krötzsch, Sebastian Rudolph and Pascal Hitzler. Owl 2 web ontology language primer (second edition). W3C recommendation, W3C, 2012. <http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/>.
- [14] Keio Beihang MIT, ERCIM. Vocabularies, 2020.
- [15] a.s. Varecha PEREX. Varecha, 2020.

- [16] Eric Prud'hommeaux. Semantic web specification at w3c, 2020.
- [17] Matej Rychtárik. Inteligentný receptár na báze prepojených dát, 2019.
- [18] Guus Schreiber and Yves Raimond. RDF 1.1 primer. W3C note, W3C, June 2014.
<http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/>.
- [19] Bootstrap team. axios, 2020.
- [20] Bootstrap team. Bootstrap, 2020.
- [21] James Hendler Tim Berners-Lee and Ora Lassila. A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 284(5):34–43, 2001.
- [22] X. Yummly, 2020.