ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kuzevanov** Jméno: **Igor** Osobní číslo: **434798**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Vývoj softwarové komponenty pro katalogy SPARQL dotazů

Název bakalářské práce anglicky:

Development of Software Component for SPARQL query Catalogs

Pokyny pro vypracování:

1) Průzkum existujících řešení
2) Definovat případ použití komponenty.
3) Návrh a implementace.
4) Validace - průzkum použitelnosti komponenty pomocí uživatelů

Seznam doporučené literatury:

[1] Nielsen, J. Usability Engineering. Morgan Kaufmann, 1993.
[2] Dumas, J., Loring, B. Moderating Usability Tests. Morgan Kaufmann, 2008.
[3] Kuniavsky, M. Observing the User Experience. Morgan Kaufmann, 2003
[4] Courage, C., Baxter, K. Understanding Your User. Morgan Kaufmann, 2005.
[5] https://www.w3.org/TR/sparql11-overview/
[6] Knowledge Organization Systems - konkretní literatura bude upřesněná

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Bogdan Kostov, skupina znalostních softwarových systémů FEL

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

Datum zadání bakalářské práce: **14.02.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Bogdan Kostov podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

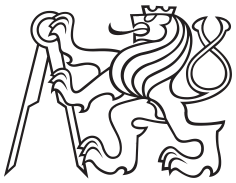
III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Vývoj softwarové komponenty pro katalogy SPARQL dotazů

Igor Kuzevanov

Školitel: Ing. Bogdan Kostov
Květen 2019

Poděkování

V první řadě bych chtěl poděkovat vedoucímu této práce panu Ing. Bogdanu Kostovovi za jeho rady, zpětnou vazbu a pravidelnou pomoc během realizace této práce. Dále bych chtěl poděkovat své matce za poskytnutí duševní a finanční podpory a samozřejmě České republice a Českému vysokému učení technickému za poskytnutí možností dosažení vysokoškolského vzdělání.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Abstrakt

Cílem této práce bylo vytvořit GUI komponentu pro efektivní práci s katalogy SPARQL dotazů.
V první části práce jsou popsaná existující řešení tohoto problému, u nichž byl proveden průzkum uživatelsky nepohodlných nebo chybějících prvků GUI na základě odezvy cílové skupiny uživatelů.
V další části jsem se soustředil na návrhu a implementaci nové GUI komponenty, vylepšující současná řešení. Závěrečnou části je testování použitelnosti výsledné komponenty.

Klíčová slova: semantický web, Linked data, SPARQL, React, komponenta

Školitel: Ing. Bogdan Kostov
Praha, Resslova 307/9, místnost: E-113

Abstract

The goal of this thesis is to create a GUI component for an effective work with SPARQL query catalogs.
In the first part i described existing solutions of this problem including the research of the quality of the user interface designs. In the second part i designed and implemented a new GUI component that could improve existing once.
In the conclusion i prefomed few accessibility tests.

Keywords: semantic web, Linked data, SPARQL, React, component

Title translation: Development of Software Component for SPARQL query Catalogs

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Již na konci 20. století počet stránek na webu neustále rostl a tím komplikoval vyhledávání relevantní informace. Proto v roce 2001 ředitel konsorcia W3C Tim Berners-Lee přišel s myšlenkou sémantického webu, který je rozšířením současného webu, v němž data mají přidělen dobře definovaný význam lépe umožňující počítačům a lidem spolupracovat. Je ale důležité říct, že sémantický web není jenom o přidělení významu, ale také o propojení dat mezi sebou, což by umožňovalo jednoduché nalezení souvisejících informací.

Pro splnění výše uvedených požadavků byl v roce 2006 zaveden koncept Linked Data [2]. Podrobnější popis bude uveden v další kapitole, prozatím je důležité vědět, že jedním z cílů jeho vytvoření bylo umožnění práce s daty na webu jako s relační databází a dotazování na jejich obsah pomocí jazyků podobných SQL. Jedním z nichž je doporučený konsorciem jazyk SPARQL (SPARQL Protocol and RDF Query Language) [1] .

Nechť Linked Data katalog je uzel v sémantickém webu, obsahující informace ve formátu RDF (Resource Description Framework) [16] na základě příslušné tomuto uzlu ontologie (příkladem je projekt DBpedia[5]). Podobně tomu, jak SQL databáze uchovávají data na základě relačních schémat, Linked Data katalog uchovává data podle některé předem určené ontologie.
Dále zadefinujme pojem katalogu SPARQL dotazů - Linked Data katalog, který obsahuje množinu SPARQL dotazů, kde jednotlivé dotazy mohou mít název, textový popis apod. a mohou být kategorizováne.
Příklady katalogů SPARQL dotazů:

- Katalog vzorových a poučných SPARQL dotazů pro práci s Wikidata [15].
- LSQ (Linked SPARQL Queries) [12] - katalog metadat o SPARQL dotazech získaných z logů různých veřejných SPARQL endpointů.

Cílem této práce je potom vyvinout GUI komponentu pro jednodušší práci s katalogy SPARQL dotazů.

1.1 Aktuální stav

V současné době existují softwarová řešení, která umožňují práci se SPARQL dotazy, zahrnující editování, analýzu, našeptávání kódu a další. Chybí ale řešení schopné pracovat s kategorizacemi dotazů.

1.2 Cíle práce

Definujeme cíle práce jako obecné požadavky na grafickou komponentu, které později upřesníme v příslušné kapitole:

- Komponenta dovoluje práci s množinou kategorií dotazů
- Komponenta je přepoužitelná
- Komponenta umí zobrazovat seznam dotazů na základě dané kategorizace
- Komponenta umožňuje efektivní editaci dotazů a jeho kategorií

Kapitola 2
Prostředí

V úvodu jsme zmínili Linked data - důležitý pojem pro pochopení problematiky této práce. Nyní jej popíšeme podrobněji a také některé související definice.

2.1 RDF

Resource Description Framework 1.1 (RDF) [16] je grafový datový model, který je součástí koncepce sémantického webu a reprezentuje tvrzení o zdrojích ve formátu "subjekt - predikát - objekt". Důležitou vlastností tohoto formátu je čitelnost jak lidsky, tak i strojově. Pro pojmenování subjektu, predikátu nebo objektu se používá URI. Množina RDF-tvrzení potom tvoří orientovaný graf, kde vrcholy jsou subjekty a objekty a hrany jsou predikáty.

2.2 Linked data

Nevýhoda klasického webu spočívá v tom, že jeho data nemají vedle sebe kontext o informaci, kterou reprezentují. Dva datové objekty totiž můžou mít vlastnosti se stejným názvem, ale odlišným významem. Linked data je způsob řešení tohoto problému. Model Linked Data předpokládá, že data jsou publikována jako datasety ve formátu RDF a jsou strukturována na základě slovníků (ontologií) a jednotlivé datasety jsou propojené pomocí hypertextových odkazů. Samotný dataset může být složen z více RDF grafů.

2.3 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) [1] je sémantický dotazovací jazyk nad daty ve formátu RDF a je hodně podobný jazyku SQL. SPARQL umožňuje dotazovaní pomocí vzorů RDF grafů, kde uzly nebo hrany můžou být nahrazeny proměnnou. Níže uvedený příklad vrátí všechny dvojice (subjekt a objekt) obsahující predikát “rdfs:label”. Neboli vrátí všechny dvojice vrcholů, které jsou spojeny hranou “rdfs:label”.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT *
WHERE { ?entity rdfs:label ?name . }
```

Pro čtení dat SPARQL podporuje 4 základní typy dotazů:

- SELECT - výstupem je tabulka, obsahující data vyhovující dotazu.

  ```
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  SELECT ?nameX ?nameY ?nickY
  WHERE
    { ?x foaf:knows ?y ;
         foaf:name ?nameX .
      ?y foaf:name ?nameY .
      OPTIONAL { ?y foaf:nick ?nickY }
    }
  ```

- CONSTRUCT - výstupem je RDF graf zkonstruovaný na základě šablony grafu.

  ```
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
  CONSTRUCT { <http://example.org/person#Alice> vcard:FN ?name }
  WHERE { ?x foaf:name ?name }
  ```

- ASK - výstupem je odpověď (boolean), zda existuje řešení pro dotaz.

  ```
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  ASK  { ?x foaf:name  "Alice" }
  ```

- DESCRIBE - výstupem je RDF graf popisující nalezený zdroj.

  ```
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  DESCRIBE ?x
  WHERE { ?x foaf:mbox <mailto:alice@org> }
  ```

Existují také dvě důležité klauzule:

- GRAPH - umožňuje specifikovat graf z datasetu, v němž se má vyhodnotit grafový vzor.

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

  SELECT ?src ?bobNick
  FROM NAMED <http://example.org/foaf/aliceFoaf>
  FROM NAMED <http://example.org/foaf/bobFoaf>
  WHERE
    {
  ```

```
  GRAPH ?src
  { ?x foaf:mbox <mailto:bob@work.example> .
    ?x foaf:nick ?bobNick
  }
}
```

- SERVICE - umožňuje dotazovaní na data v dalším autonomním datasetu.

```
PREFIX wikibase: <http://wikiba.se/ontology#>
SELECT *
  {
    SERVICE wikibase:label { ?v wikibase:language "en". }
  }
```

2.4 YASQE

YASQE (Yet Another Sparql Query Editor)[11] je grafický editor pro práce se SPARQL dotazy postavený na komponentě CodeMirror. Podporuje zvýraznění syntaxe a mechanismus našeptávání kódu.

2.5 Intelligent Tree Data Management Component

Intelligent Tree Data Management Component[6] je React komponenta pro zobrazení stromu objektů. Je schopna zobrazovat velký počet položek aniž by snížila výkonnost aplikace nebo zhoršila UX.

2.6 React

React[4] je knihovna pro vývoj grafických uživatelských rozhraní v jazyce JavaScript. Její základním objektem je komponenta, která je potomkem React třídy Component a může obsahovat další komponenty. Celá aplikace je potom stromem komponent.
Data uvnitř komponenty ukládáme do vnitřní proměnné “state”, kterou za běhu můžeme libovolně měnit. Přenos dat mezi komponenty se realizuje pomocí “props” [1] - při inicializaci komponenty-potomka ji můžeme předat libovolná data z rodičovské komponenty. “Props” se mění pouze ve chvílí, kdy dojde ke změně “state” rodičovské komponenty. Abychom mohli změnit “state” z komponenty-potomka, musíme do něj předat odkaz na callback-metodu, po jejíž vyvolání se provedou potřebné změny na úrovní předka.
Ve většině případu pro manipulaci se “state” se používají další knihovny:

[1] Zkrátka od “Properties”

Flux, Redux, Mobx. V této práci ale, jak uvidíme později, žádná z nich není potřeba.

Kapitola 3

Rešerše existujících řešení

Protože požadovaná komponenta má specifické požadavky (práce s kategorizací dotazů), je v určitém smyslu unikátní. Ukážeme ale nástroje které splňují jiné požadavky mimo kategorizaci. Budeme si proto pamatovat, že každé z níže popsaných řešení kategorizaci nepodporuje a nebudeme to uvádět do sloupce nevýhod.

3.1 YASGUI

YASGUI je webové grafické rozhraní pro práce se SPARQL dotazy, jehož základem je komponenta YASQE.

Výhody	Nevýhody
■ Přístupná na webu. ■ Umožňuje nastavení SPARQL-endpointu, proti kterému bude dotaz spuštěn. ■ Umožňuje zobrazení výsledků dotazu v několika formátech (tabulka, graf, XML, Google Chart). ■ Umožňuje našeptávání kódu. ■ Podporuje práci s několika dotazy najednou.	■ Nepodporuje nahrávání seznamu dotazů.

3.2 Twinkle

Desktopová aplikace pro práci se SPARQL dotazy, která je nadstavbou nad "ARQ SPARQL query engine"[14]

Výhody	Nevýhody
■ Podporuje také syntaxi ARQ. ■ Podporuje několik oken pro dotazy najednou.	■ Desktopová aplikace. ■ Nepodporuje našeptávání kódu. ■ Nepodporuje nahrávání seznamu dotazů.

3.3 DARQL

Je software pro analýzu SPARQL dotazů.

Výhody	Nevýhody
■ Umožňuje provedení analýzy logu dotazů. ■ Umožňuje vizualizaci grafu dotazu. ■ Umožňuje zobrazení statistik seznamu dotazů na základě kategorizace.	■ Nepodporuje našeptávání kódu. ■ Nepodporuje editaci dotazů.

Kapitola 4
Návrh

V souladu se základy softwarového inženýrství rozlišujeme dvě podstatné skupiny požadavků: funkční a nefunkční. Dále každou z nich lze popsat z pohledů různých členů cílové skupiny uživatelů: vývojář, který tuto komponentu přepoužívá pro účely implementace vlastního softwarů a také koncový uživatel, jehož cílem je práce se samotnými SPARQL dotazy.

4.1 Požadavky

4.1.1 Funkční

Pro uživatele:

1. Komponenta umožňuje přidání nové kategorie.
2. Komponenta umožňuje zvolení množiny kategorií a následně zobrazí seznam dotazů do těchto kategorií spadajících.
3. Komponenta podporuje dva režimy volby kategorií:
   a. Přímý - zobrazí se dotazy, které spadají do jakékoliv kategorií ze zvolené množiny.
   b. Rekurzivní - zobrazí se dotazy pro přímý režim a navíc dotazy, které spadají do všech podkategorií libovolné kategorie ze zvolené množiny.
4. Komponenta zobrazuje nejvíc N (konfigurovatelná hodnota) dotazů na jedné stránce, přičemž každý dotaz má zvýrazněnou syntaxi.
5. Zobrazený dotaz je needitovatelný a obsahuje vedle sebe needitovatelný seznam všech kategorií, do nichž patří přímo.
6. Zobrazený dotaz má vedle sebe tlačítko pro přechod do editace dotazu a jeho kategorií.
7. Obrazovka pro editaci dotazů podporuje:

a. Strom všech kategorií, z něhož uživatel je schopen přiřazovat dotazu další kategorie nebo odstraňovat existující.
b. Editaci kódu dotazu včetně zvýrazňování syntaxe a našeptávání.

Pro vývojáře:

1. Komponenta podporuje konfiguraci propojení komponenty a REST API.
2. Komponenta podporuje konfiguraci počtu dotazů zobrazených na jedné stránce.

4.1.2 Nefunkční

Pro uživatele:

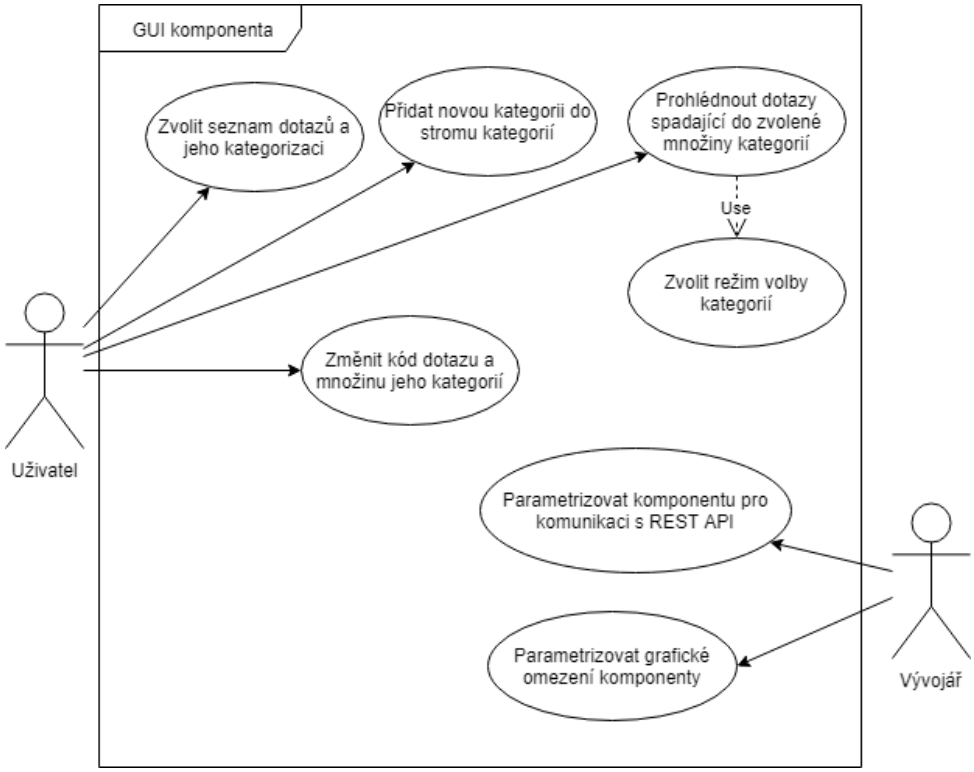
1. Renderování seznamu dotazů nesmí trvat příliš dlouho ani zpomalovat další práci s komponentou.

Pro vývojáře:

1. Komponenta bude přepoužitelná jako standardní komponenta v Reactu. To znamená, bude možné ji importovat do dalších projektů a případně rozšiřovat její funkcionalitu.

4.2 Případy užití

Na základě požadavků uvádím i nejdůležitější případy užití komponenty. Dva aktéry jsou uživatel, u něhož zkoumáme práci s samotnou komponentou, a vývojář, který je zodpovědný za parametrizaci komponenty a její integraci do větších systémů.

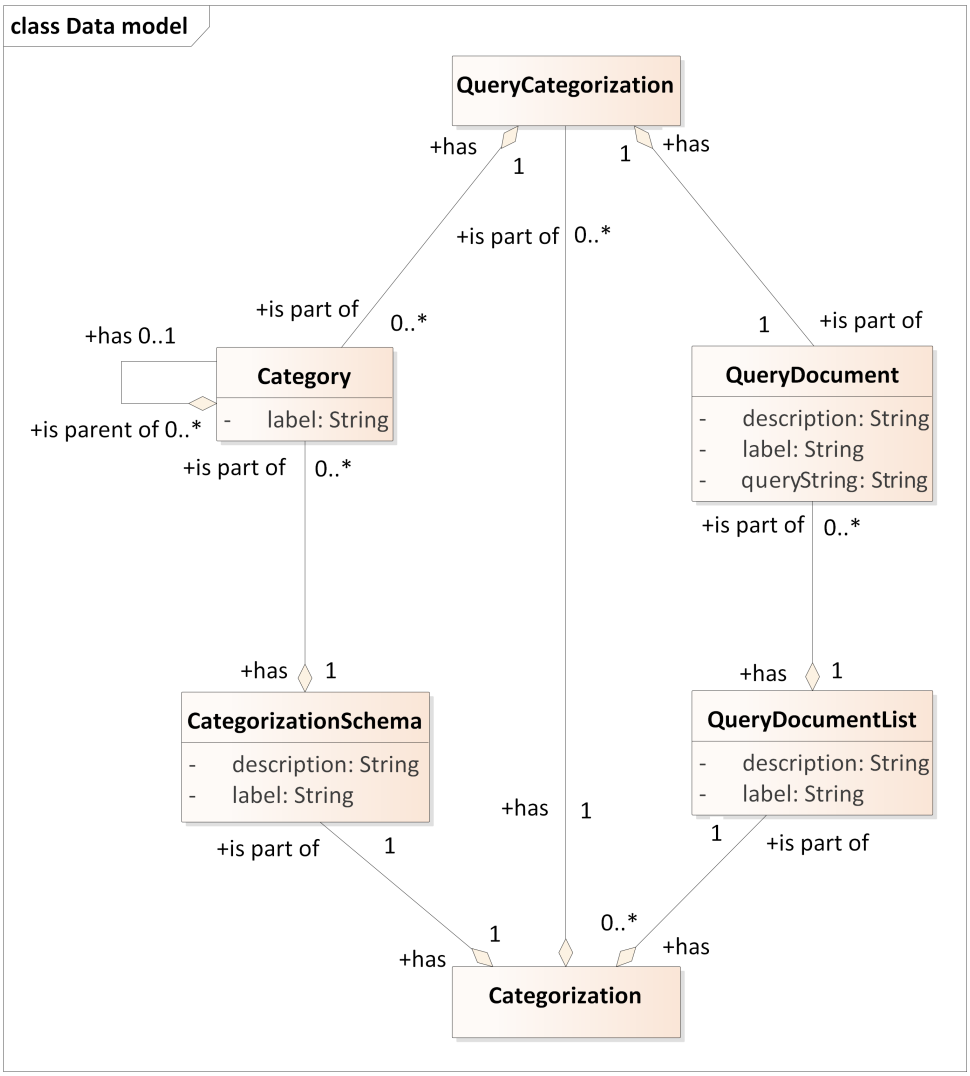


Obrázek 4.1: Diagram případů užití

4.3 Datový model

Zadefinujme datový model, který dále budeme uvažovat při implementaci jednotlivých částí komponenty. Protože hlavní účel komponenty je práce s katalogy SPARQL dotazů, datový model bude takový katalog reprezentovat.

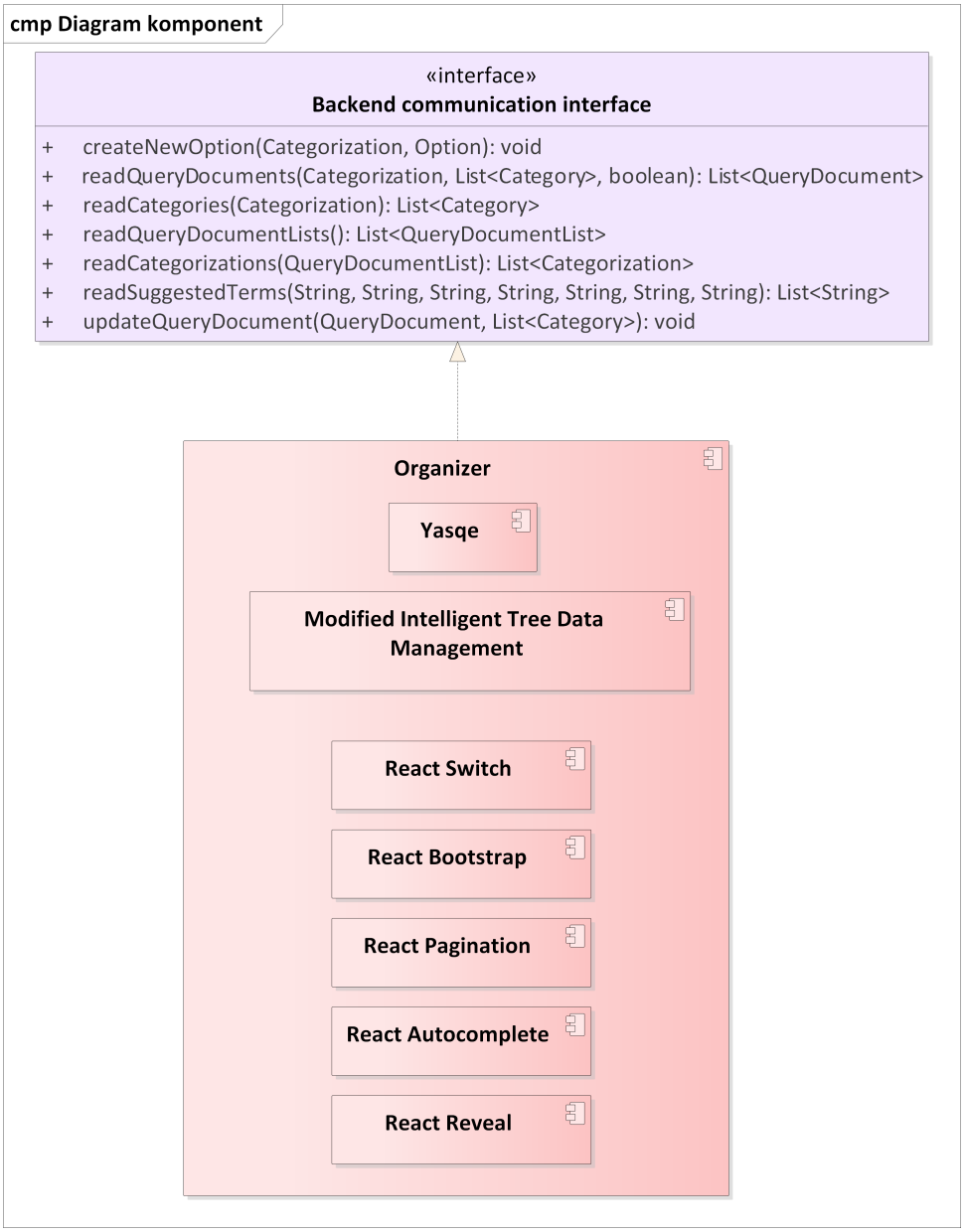
- **QueryDocument** reprezentuje SPARQL dotaz - jeho kód - a další informace kolem něj: název a textový popis.
- **QueryDocumentList** reprezentuje množinu SPARQL dotazů typu QueryDocument a také má název a textový popis.
- **Category** reprezentuje kategorii. Kategorie vždy má název a může, ale nemusí mít rodiče.
- **CategorizationSchema** reprezentuje soubor kategorií typu Category. Má název a textový popis.
- **QueryCategorization** reprezentuje kategorizaci jednotlivého dotazu. Skládá se ze SPARQL dotazu typu QueryDocument a množiny kategorií typu Category.
- **Categorization** reprezentuje kategorizaci nějaké množiny SPARQL dotazů (QueryDocumentList). Pojmem "kategorizace množiny" se myslí množina kategorizací její dotazů (QueryCategorization)



Obrázek 4.2: Diagram datového modelu

4.4 Diagram komponent

Teď se podíváme na obecnější diagram komponent. Cílova komponenta (dále ji budeme říkat Organizer) se skládá ze tří důležitých subkomponent: Yasqe komponenta - zobrazuje syntaktický zvýrazněný SPARQL kód a umožňuje našeptávání, Modified Intelligent Tree Data Management komponenta - je modifikací Intelligent Tree Data Management komponenty a je zodpovědná za renderování stromu kategorií dotazů, skupina komponent (React Bootstrap[13], React pagination[13], React Switch[3], React Autocomplete[10], React Reveal[9]) - je určena pro stylování grafického rozhraní a zlepšení UX.
Rozhraní pro komunikaci s backendem je vyhrazeno pro implementaci vývojářem, který pomoci něj dokáže propojit grafickou komponentu se servisou, obsluhující SPARQL katalog.



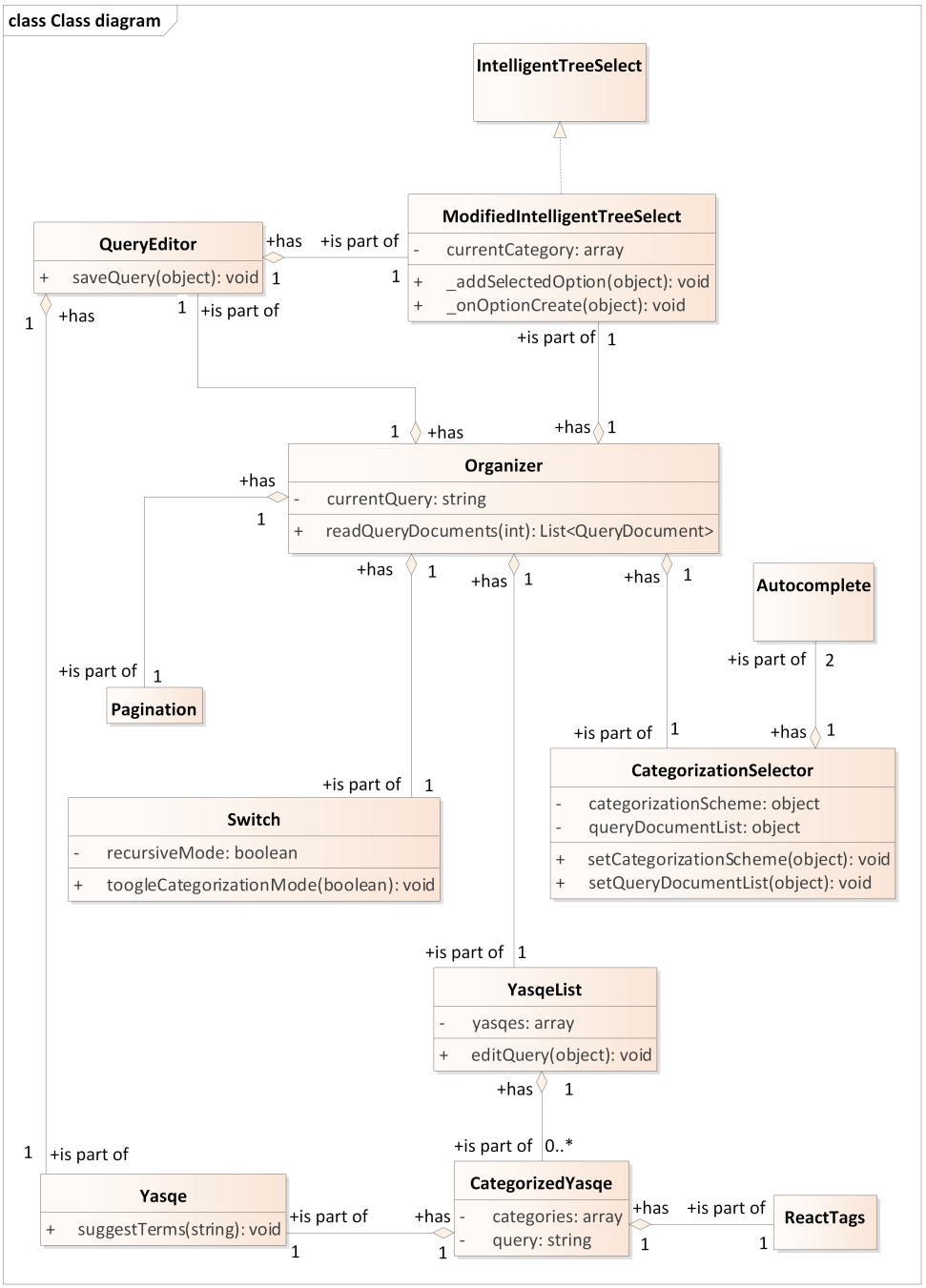
Obrázek 4.3: Diagram komponent

4.5 Class diagram

Detailněji popíšu cílovou architekturu výsledné komponenty pomocí UML class diagramu.

1. Centrálním objektem je **Organizer**, který slouží kontejnerem pro ostatní subkomponenty a uchovává většinu dat uvnitř svého "state". Na diagramu ale obsahuje pouze hodnotu zvoleného pro editaci dotazu - "currentQuery", což je uděláno záměrně pro lepší pochopení návrhu komponenty[1]. Ze stejných důvodu Organizer má definovanou pouze jednu metodu - "loadQueries(array)", která na základě seznamu zvolených kategorií vrátí příslušný seznam dotazů.
2. **CategorizationSelector** je subkomponenta pro zvolení uživatelem seznamu dotazů a jeho kategorizace.
3. Dalším prvkem je **ModifiedIntelligentTreeSelect**, který je rozšířením IntelligentTreeSelect. Uchovává v sobě seznam zvolených kategorií a obsahuje dvě přetížené metody: "_onOptionCreate" zaznamená vytvoření nové kategorie a následně zavolá metodu předka, "_addSelectedOption" nejprve zavolá metodu předka a hned potom obnoví seznam dotazů podle právě změněného seznamu zvolených kategorií.
4. **Switch** je subkomponenta pro přepnutí režimu volby kategorií.
5. **CategorizedYasqe** je subkomponenta, která v sobě spojuje **Yasqe** komponentu a needitovatelný seznam kategorií realizovaný pomocí **ReactTags**.
6. Hned po zvolení kategorií se pomocí komponenty **YasqeList** zobrazí seznam dotazů do těchto kategorií spadající. YasqeList v sobě obsahuje N komponent CategorizedYasqe.
7. Po rozkliknutí dotazu se uživateli zobrazí obrazovka pro editaci dotazů realizována pomocí komponenty **QueryEditor**. Tato komponenta je sestavená z Yasqe editoru a stromu kategorií (ModifiedIntelligentTreeSelect).

[1] Jak jsem uvedl v 2.6, všechny potomky komponenty Organizer dostávají od něj určitý kus dat jako "props" a callback-metody, které dovoluje potomkům měnit "state" Organizeru. Proto na diagramu přiřazuji hodnotu subkomponentě, která s touto hodnotou pracuje, i když ve skutečnosti mění tuto hodnotu v Organizeru pomocí callback-metody



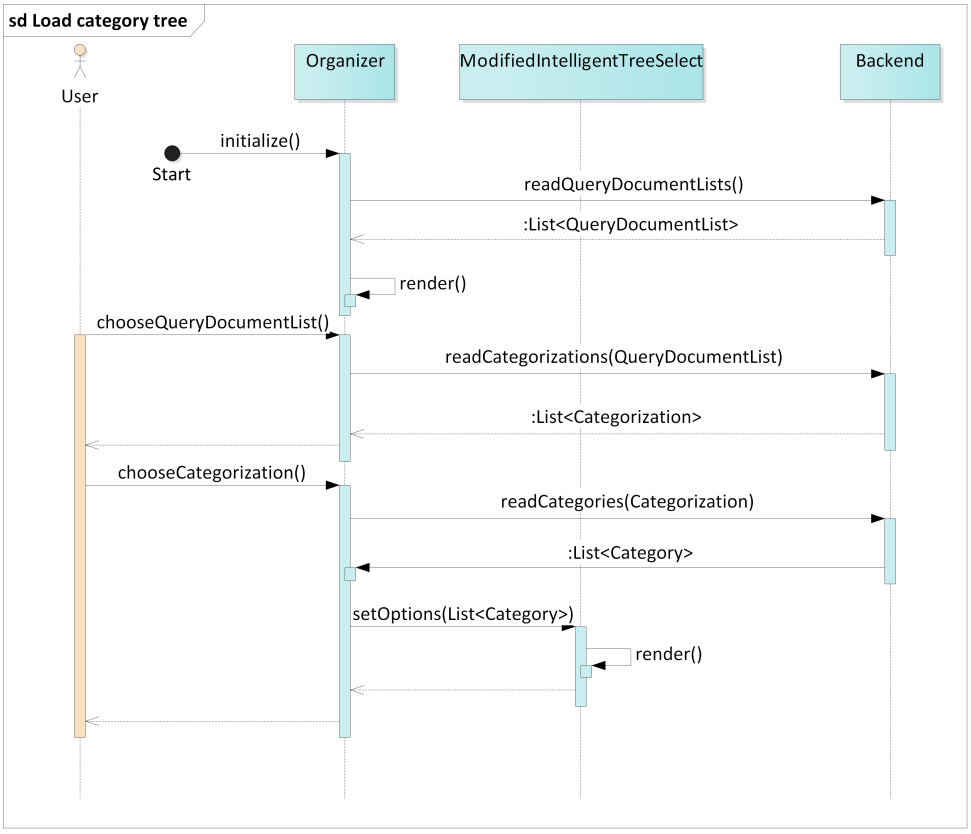
Obrázek 4.4: Class diagram výsledné komponenty

4.6 Sekvenční diagramy

Teď když mám představu o tom, z čeho se výsledná komponenta skládá, proberu její nejdůležitější procesy a to pomocí sekvenčních diagramů. Budu se snažit pokryt co nejvíce případů užití a zároveň popisovat jenom klíčové procesy.

4.6.1 Volba seznamu dotazů a příslušné kategorizace

Hned po inicializaci komponenty se zavolá metoda pro zjištění seznamu seznamů dotazů (QueryDocumentList). Uživatel je poté schopen zvolit libovolný z nich a tímto spustí další metodu, která vrátí seznam možných kategorizací a zobrazí jej uživateli, který volí právě jednu. Nyní se uživateli zobrazí stromečková komponenta umožňující skládání množiny kategorií a je naplněná kategoriemi z příslušné kategorizace.

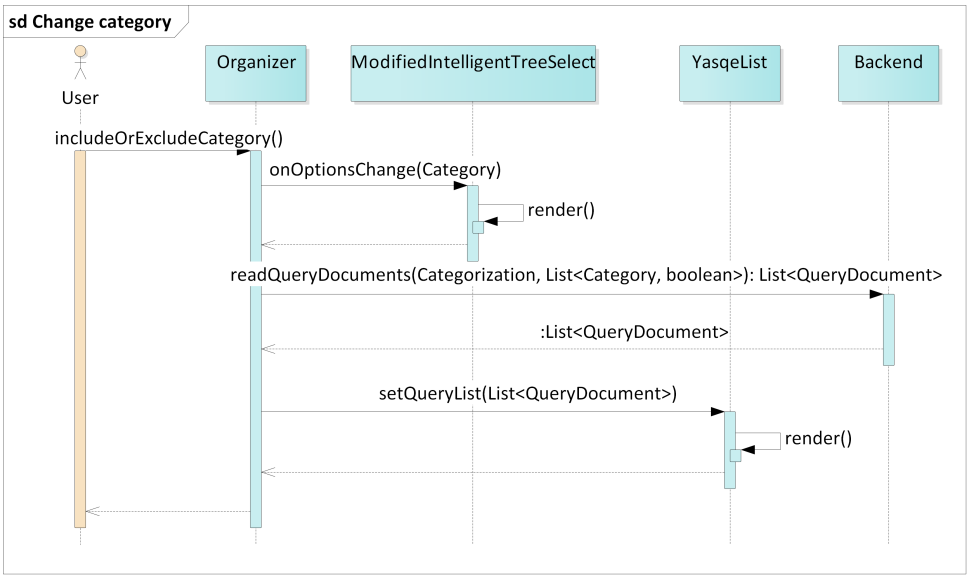


Obrázek 4.5: Proces volby seznamu dotazů a kategorizace

4.6.2 Změna množiny kategorií

Jakmile se ve množině kategorií objeví alespoň jedna kategorie, komponenta načte a vykreslí seznam dotazů, spadající do této množiny. Po jakékoliv

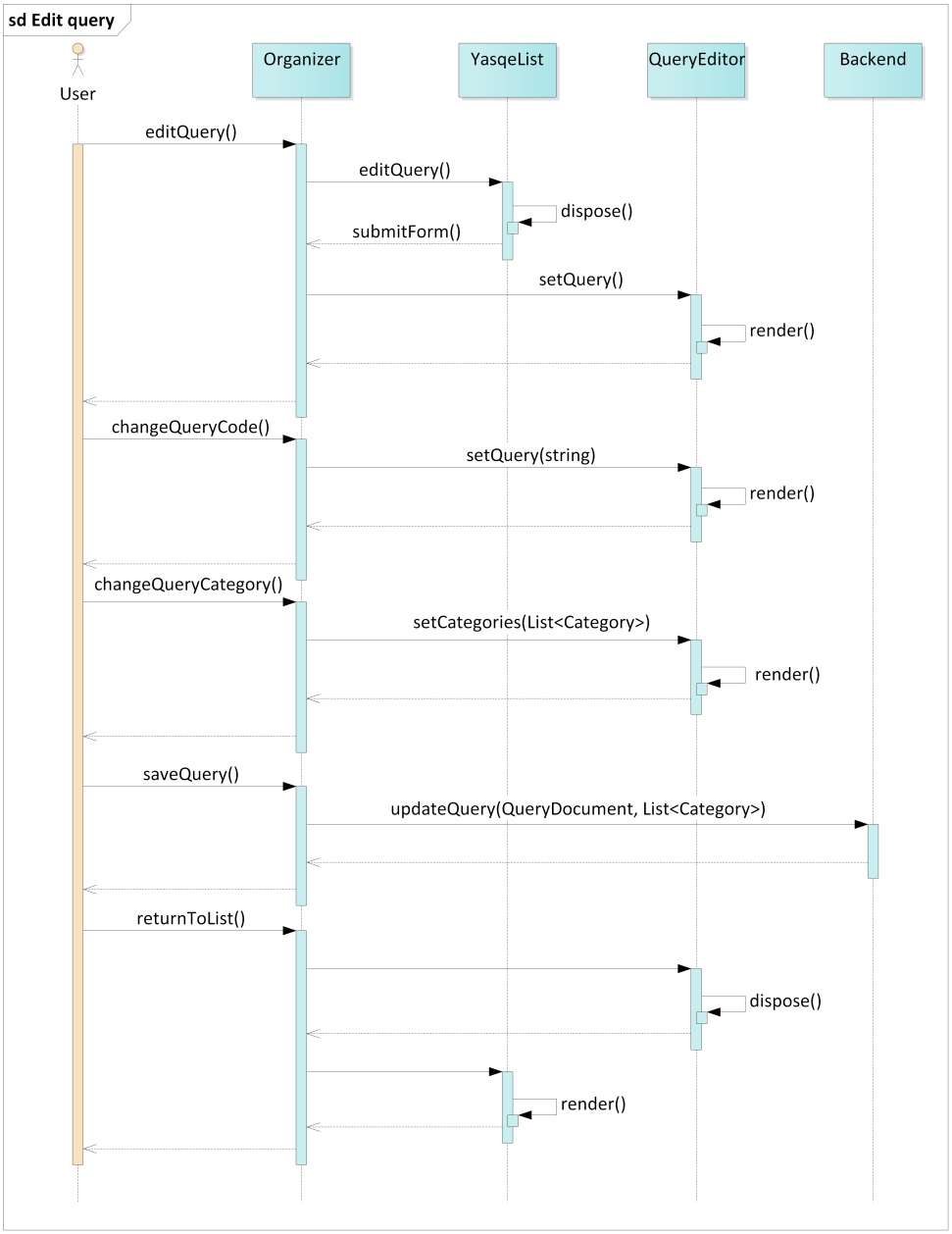
změně ve množině kategorií dojde ke změně “state” stromečkové komponenty a překreslení seznamu dotazů.



Obrázek 4.6: Proces změny množiny kategorií

4.6.3 Editace dotazu

Pokud uživatel se dostal k seznamu dotazů je nyní schopen jej editovat. Každý dotaz je zobrazen pomocí Yasqe komponenty (kód je zatím needitovatelný) a má vedle tlačítko pro přechod na obrazovku editoru. Po stisknutí tlačítka komponenta odstraní z obrazovky seznam dotazů a vykreslí komponentu QueryEditor pro příslušný dotaz. V tuto chvíli uživatel má možnost měnit kód nebo množinu kategorií tohoto dotazu, přičemž v tomto případě dojde ke změně “state” komponenty QueryEditor. Až po stisknutí tlačítka pro uložení dotazu se tento modifikovaný dotaz uloží. Po stisknutí příslušného tlačítka se uživatel vrátí zpět na stránku se seznamem dotazů.



Obrázek 4.7: Proces editace dotazu

Kapitola 5
Implementace

V této kapitole popíšu nástroje, které budu používat během implementace a netriviální řešení, které přímo neplynou z návrhu.

5.1 Prostředí

Pro implementaci komponenty potřebuji pouze Node.js pro import knihoven 3. strany a IDE podporující syntaxi React JSX, což v mém případě bude IntelliJ IDEA.

5.2 Části komponenty

5.2.1 Algoritmus našeptávání

Výchozí Yasqe komponentu rozšířím o sofistikovanější algoritmus našeptávání. Od výsledného našeptávače budu očekávat následující chování:

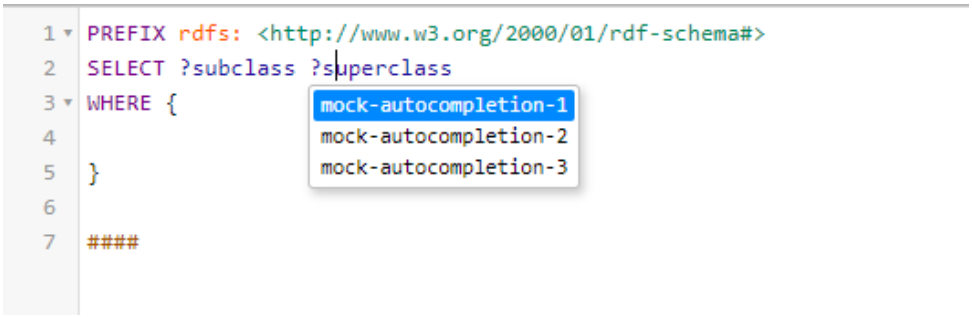
- našeptávač dokáže rozpoznat v jaké části tripletu (subject - predikát - objekt) se nachází kurzor, proto je schopen našeptávat relevantně a to i v případě, že SPARQL dotaz je v nevalidním stavu (například obsahuje syntaktickou chybu)
- našeptávač dokáže rozpoznat, že kurzor se nachází uvnitř konstrukce SERVICE nebo GRAPH a je schopen našeptávat termy relevantní pro příslušný SPARQL-endpoint nebo RDF graf

Příklad toho, jak vypadá našeptávání je na obrázku 5.1.
Jelikož Yasqe dovoluje implementaci vlastního našeptávače, v souladu s dokumentací komponenty potřebujeme implementovat metodu typu:

```
function (doc: yasqe) -> configurationObject: object
```

kde struktura objektu "configurationObject" je popsána na stránce dokumentace komponenty.

Nejprve zadefinujeme několik pomocných metod pro přehlednější zobrazení IRI a to jsou:



Obrázek 5.1: Našeptávání v Yasqe komponentě

- `function iriToPrefixed (token: string) -> token: string`

  metoda má na vstupu token (řetězec nabízený našeptávačem) a nahradí IRI za prefix, je-li definován v hlavičce dotazu. Příklad:

  `http://xmlns.com/foaf/0.1/name -> foaf:name`

- `function prefixedToIri (token: string) -> token: string`

  metoda má na vstupu token (řetězec nabízený našeptávačem) a nahradí prefix za IRI, je-li definován v hlavičce dotazu. Příklad:

  `foaf:name -> http://xmlns.com/foaf/0.1/name`

Nyní začnu implementovat samotný našeptávač. Proto zadefinuju několik metod objektu "configurationObject" a to jsou:

- `function isValidCompletionPosition () -> isValid:boolean`

  metoda by měla kontrolovat, jestli kurzor uživatele uvnitř Yasqe komponenty je v pozici, kdy lze našeptávat. V našem případě metoda bude vždy vracet "true" - našeptáváme všude. Samotné našeptávání proběhne tak, že uvnitř metody "isValidCompletionPosition" budeme nastavovat pomocné hodnoty, které později zpracuje konkrétní implementace metody "readSuggestedTerms".

  - Yasqe komponenta umožňuje pro aktuální pozici kurzoru odhadnout o jaký element SPARQL dotazu se jedná. A to tak, že aktuální token (v tomto kontextu token je objekt, obsahující řetězec, na něhož uživatel klinknul naposled a další data) obsahuje seznam možných elementů. Plný seznam elementů neuvádím, protože jej neuvádí ani autor, ale pomocí průzkumu zdrojových kódu komponenty se podařilo zjistit některé možné elementy a z toho tvrdím následující:
    1. Pokud seznam možných elementů obsahuje "a", nejspíš se jedná o "predikát".
    2. Pokud seznam možných elementů obsahuje "OPTIONAL", nejspíš se jedná o "subjekt".
    3. Jinak předpokládám, že jde o "objekt".

- Zjistím, jestli kurzor je uvnitř konstrukce SERVICE nebo GRAPH tak, že projdu celý kód od začátku až do kurzoru a budu počítat počet levých a pravých složených závorek (“{” a “}”). Pokud narazím na klíčové slovo SERVICE nebo GRAPH a počet levých závorek je větší než pravých, tak předpokládám, že jsem uvnitř jedné z těchto konstrukcí.

```
function preProcessToken (token: object|string) ->
token: object
```

Metoda zpracovává token před tím, než začne vyhledávat relevantní termy. V mém případě metoda jenom zavolá “prefixedToIri(suggestion)” a vrátí její výsledek, protože předpokládám, že našeptávací algoritmus zpracovává IRI, nikoliv prefixy.

- `function postProcessToken (token: object|string, suggestion: string) -> string`

  Metoda zpracovává a vrací již nabídnutý term. V mém případě metoda jenom zavolá “iriToPrefixed(suggestion)” a vrátí její výsledek. Neboli požaduji, aby se termy našeptávaly uživateli s prefixy, nikoliv s IRI.

Kapitola 6
Testování

V této kapitole se budu věnovat testování výsledné komponenty. Softwarové testy můžeme rozdělit do dvou základních skupin: funkční a nefunkční. Do první skupiny obvykle patří jednotkové, integrační, systémové, akceptační, regresní, smoke testy a testy uživatelského rozhraní. Do druhé potom spadají například testy odezvy a použitelnosti, zátěžové a penetrační testy. Jelikož cílem této práce bylo vytvoření pouze grafické komponenty, budu se zabývat jenom nefunkčními testy. Samozřejmě je vždy místo i pro funkční testy, ale vzhledem k relativní jednoduchosti konečné implementace komponenty a časovému omezení na dokončení práce považuji nefunkční testy za nejdůležitější. Protože komponenta neřeší otázky bezpečností a nebyly stanoveny žádné omezení na odezvu budu se věnovat pouze testům použitelnosti.
Podle návrhu Jakoba Nielsena[7] všechny testy použitelnosti rozdělím na dvě větší skupiny: s uživatelem a bez uživatele. Sice existuje velké množství testů z obou skupin, zvolím si pouze jeden typ testů pro každou ze skupin.

6.1 Prerekvizita

Jelikož cílem této práce bylo vytvoření pouze grafické komponenty, pro její plné otestování jsou potřeba dvě věci:

1. Implementovat rozhraní pro komunikaci s backendem.
2. Implementovat backend pro komunikaci s katalogem SPARQL dotazů.

Samotný backend již implementoval vedoucí této práce. Rozhraní pro komunikaci jsem implementoval osobně.

6.2 Testování bez uživatelů

Testování bez uživatelů (používám jako překlad pro usability inspection[8]) je skupinou metod testování, kde inspektor (osoba provádějící test) zkoumá uživatelské rozhraní. Tyto metody jsou:

- Kognitivní průchod - metoda zjišťuje míru náročností rozhraní pro splnění uživatelem předem definovaného úkolu. V každém kroku úkolu inspektor položí 3 otázky a odpoví na ně:

  - Q1: Je uživateli zřejmé, co má udělat?
  - Q2: Spojí si uživatel popis akce s danou akcí?
  - Q3: Dostane uživatel dostatečnou zpětnou vazbu?

- Heuristická evaluace - metoda zjišťuje problémy použitelnosti tak, že inspektor zkontroluje, jestli prvek rozhraní není v rozporu se seznamem heuristik.

6.2.1 Průběh testování

Testovat komponentu budu pomocí metody kognitivního průchodu na základě seznamu úkolů, kde každý úkol bude obsahovat scénář jeho správného splnění.

- Úkoly.
  - Přejít na obrazovku editace dotazu, obsahujícího kategorii “Graphs”.
    1. Zvolit seznam dotazů “Holst query list”.
    2. Zvolit jakoukoliv kategorizaci.
    3. Zvolit kategorii “Graphs”.
    4. Přejít na obrazovku editace libovolného dotazu.
  - Změnit kód dotazu a přidat jej novou kategorii “Others”.
    5. Libovolně změnit kód dotaz.
    6. Přidat dotazu kategorii “Others”.
    7. Uložit změnu.
  - Najít změněný dotaz pomocí nově přidané kategorie.
    8. Vrátit se na hlavní obrazovku.
    9. Odstranit kategorii “Graphs”.
    10. Přidat kategorii “Others”.
    11. Najít v seznamu dotazů právě změněný dotaz, obsahující kategorie “Graphs” a “Others”.

Úkol	Q1	Q2	Q3
1.1	Ano.	Ano.	Ano.
1.2	Ano.	Ano.	Ano.
1.3	Ne, chybí textová pomůcka vedle stromečkové komponenty.	Ano.	Ano.
1.4	Ano.	Ano.	Ano.
2.1	Ano.	Ano.	Ano.
2.2	Ano.	Ano.	Ano.
2.3	Ano.	Ano.	Ne, uživateli se nezobrazí žádná hláška o úspěšném či neúspěšném uložení.
3.1	Ano.	Ano.	Ano.
3.2	Ano.	Ano.	Ano.
3.3	Ano.	Ano.	Ano.
3.4	Ano.	Ano.	-

Z tabulky je vidět, že komponenta má minimálně dva nápady na vylepšení. Tomuto by se měl věnovat čas v další iteraci vývoje.

6.3 Testování s uživatelem

Testování s uživatelem (používám jako překlad pro usability testing[7]), jak je vidět z názvu, je skupina metod založených na testování pomocí reálných lidí z cílové skupiny uživatelů. Tyto metody většinou zahrnují natáčení videa a audia, ideálně tak, aby se dalo vidět směry pohledů uživatele, případné pohyby jeho těla a jakékoliv manipulace uživatele s testovacím zařízením (např. pohyby myše a stisknutí kláves) pro následnou detailnější analýzu. Testování se také zúčastňuje moderátor, který je po celou dobu testu se nachází vedle uživatele a sděluje mu úkoly, případně poskytuje pomoc s jejich splněním.
Během této práce bohužel nezbyl čas pro tento typ testování.

Kapitola 7

Závěr

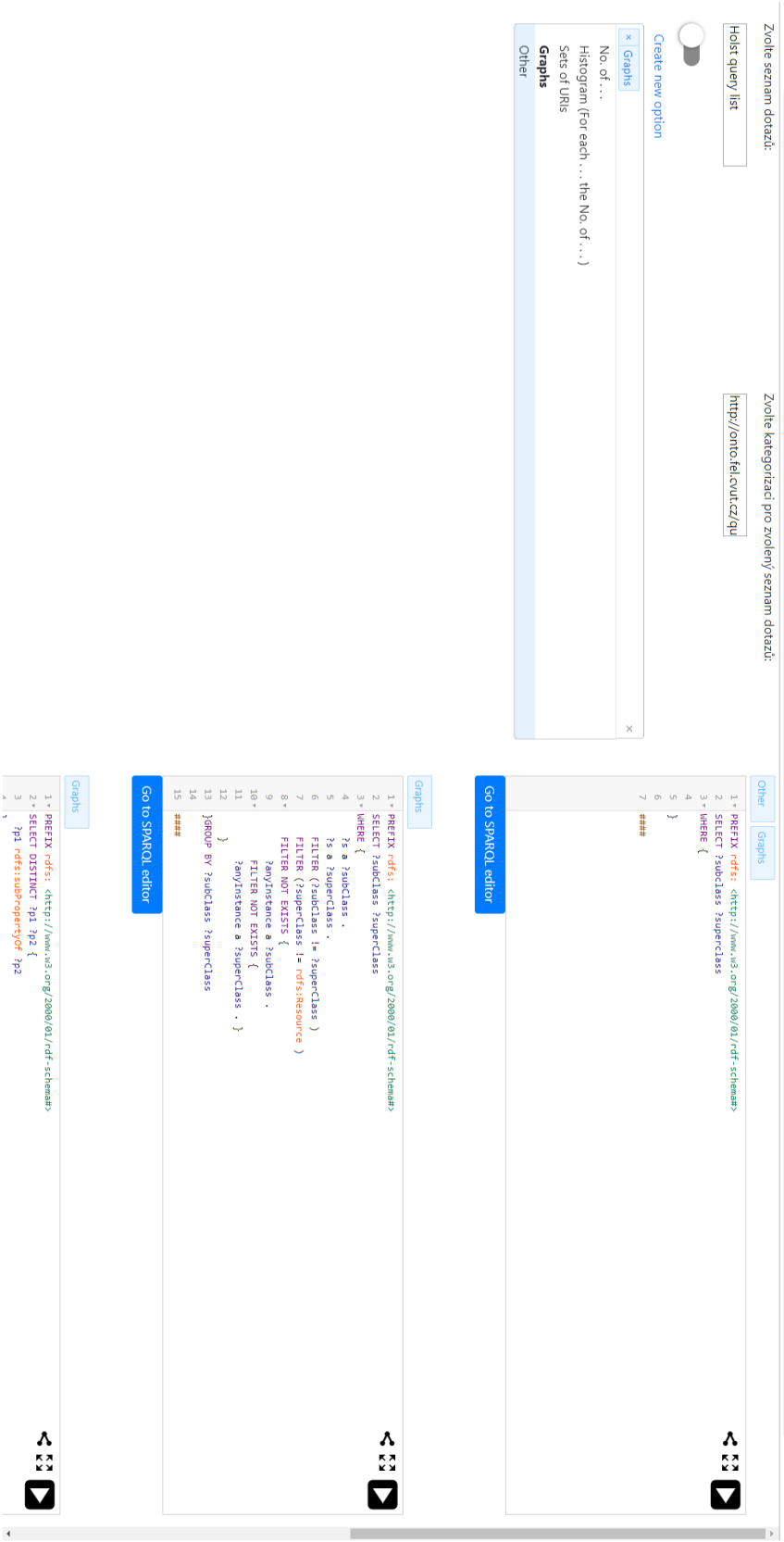
Nakonec se mi podařilo dosáhnout většinu cíli této práce a implementovat komponentu pro práci s katalogy SPARQL podle předem specifikovaných požadavků. Jeden z cílů ale zněl jako:

> `Komponenta umožňuje efektivní editaci dotazů a jeho kategorií.`

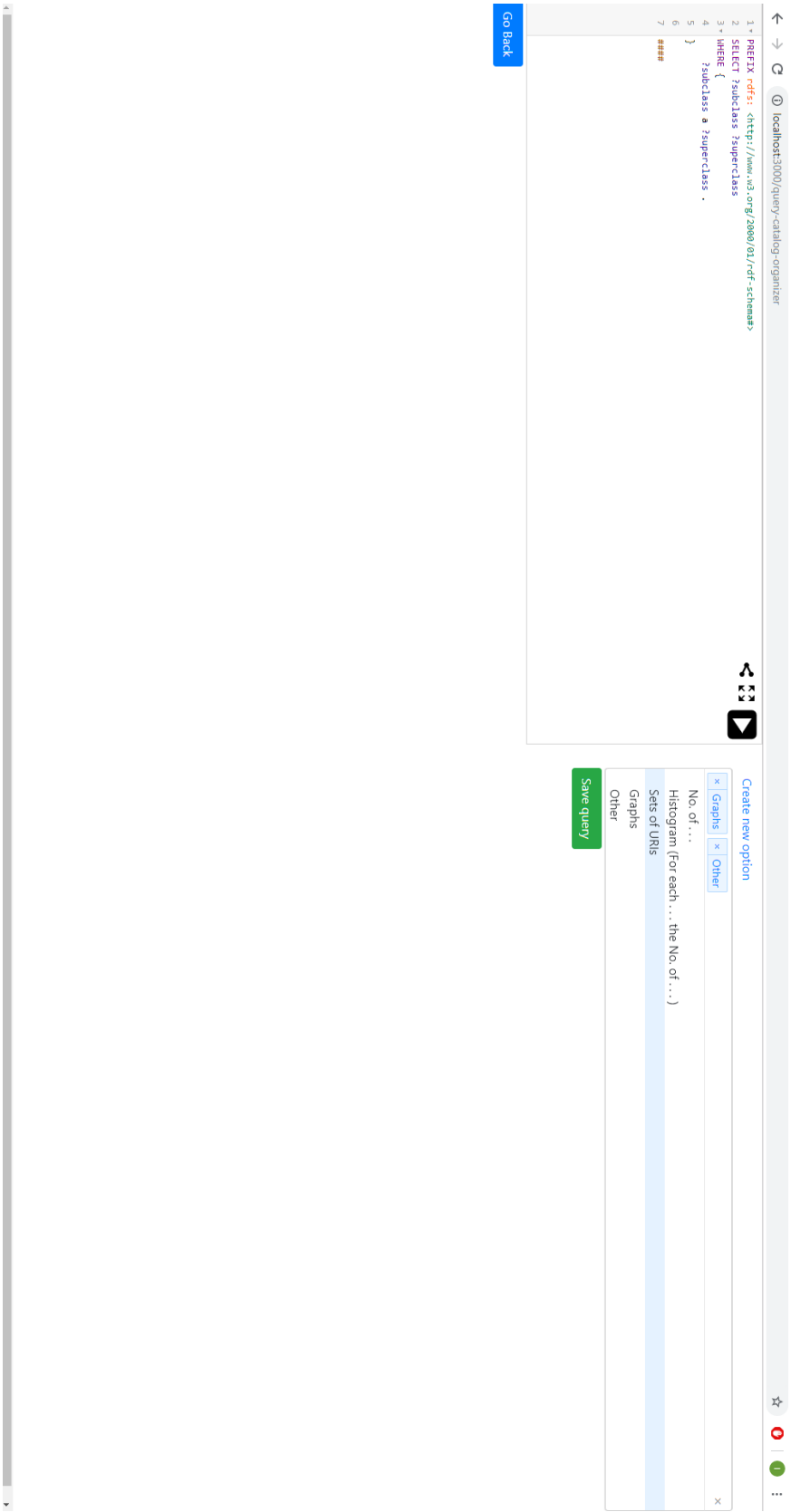
"Efektivní" se rozumí editace pohodlná pro maximálně velkou podmnožinu množiny cílové skupiny uživatelů. Proto podle zásadních principů softwarového inženýrství by se mělo provést více testů, hlavně testů s uživateli, na základě kterých by se s velkou pravděpodobností uskutečnily další změny uživatelského rozhraní.
Také během vývoje se zjistilo, že by se uživatelům hodila možnost přidání nových dotazů do katalogů. Proto nelze říct, že práce byla kompletně dokončena a existuje prostor pro její rozšíření.

Příloha A

Pohledy komponenty



Obrázek A.1: Hlavní obrazovka komponenty



Obrázek A.2: Obrazovka editace dotazu

Příloha B

Příručka

Pro použití komponenty je potřeba ji naimportovat jako klasickou React komponentu a poslat do "props" potřebná data - počet SPARQL dotazů na jedné stránce a objekt, obsahující implementované metody rozhraní pro komunikace s backendem:

```
render() {
    return (
        <Organizer pageSize={5} rest={{
            readQueryDocuments: (categorization, categories, isRecursiveMode)=>{},
            readCategories: (categorization)=>{},
            readQueryDocumentLists: ()=>{},
            readCategorizations: (queryDocumentList)=>{},
            readSuggestedTerms: (lastValidQuery, currentQuery, contextType,
            contextUri, currentQueryCursor, position, predicate)=>{},
            createNewOption: (categorization, createdOption)=>{},
            updateQuery: (queryDocument, queryCategories)=>{},
        }}/>
    )
}
```

Komponenta je dostupná v repositáři NPM https://www.npmjs.com/package/sparql-organizer .
Kompletní zdrojové kódy včetně implementace pro testování jsou dostupné v repositáři KBSS https://kbss.felk.cvut.cz/gitblit/summary/16gacr!sparql-suggestions.git .

Příloha C

Literatura

[1] {SPARQL} 1.1 Overview. {W3C} recommendation, W3C, mar 2013.

[2] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, MarMar 2009.

[3] M. Englund. React switch. `https://www.npmjs.com/package/react-switch`. [Online; accessed 23-April-2019].

[4] Facebook. React framework. `https://reactjs.org/`. [Online; accessed 18-April-2019].

[5] Leipzig University, University of Mannheim. Dbpedia. `https://wiki.dbpedia.org`. [Online; accessed 15-May-2019].

[6] J. Lečbych. Intelligent tree data management component. May 2018.

[7] J. Nielsen. *Usability Engineering*. 1993.

[8] J. Nielsen. Usability inspection methods. In *Conference companion on Human factors in computing systems - CHI '94*, 1994.

[9] R. Nosov. React reveal. `https://github.com/rnosov/react-reveal`. [Online; accessed 23-April-2019].

[10] Open source. React autocomplete. `https://github.com/reactjs/react-autocomplete`. [Online; accessed 23-April-2019].

[11] L. Rietveld. Yet another sparql query editor. `https://yasqe.yasgui.org/`. [Online; accessed 08-May-2019].

[12] M. Saleem. The linked sparql queries dataset. `https://aksw.github.io/LSQ/`. [Online; accessed 15-May-2019].

[13] P. V. Stephen J. Collings, Matthew Honnibal. React bootstrap. `https://react-bootstrap.github.io/`. [Online; accessed 23-April-2019].

[14] The Apache Software Foundation. ARQ - A SPARQL Processor for Jena. `https://jena.apache.org/documentation/query/`. [Online; accessed 23-April-2019].

[15] Wikidata. Sparql query service/queries/examples. `https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples`. [Online; accessed 15-May-2019].

[16] D. Wood, M. Lanthaler, and R. Cyganiak. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, Feb. 2014. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.