SLOVENSKÁ TECHNICKÁ
UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY
A INFORMAČNÝCH TECHNOLÓGIÍ

Vyhľadávanie informácií

Zadanie 1

Search

Matúš Gáspár

2.ročník

2017/2018

1. Úvod

Úlohou bolo získať dáta z webu, oindexovať ich a vyhľadávať v nich podľa nami zvolených scenárov v ElasticSearchi.

Aplikácia pozostáva z vlastného rozhrania v Jave (Eclipse IDE), ktorá používa Maven dependency, kde v pom.xml súbore sú uložené dependencies. Na začiatku programu možné vybrať si z troch možností:

A. Zapnúť crawler, ktorý bude prechádzať stránkou "hej.sk" a zbierať údaje z ich webu
B. Zozbierané dáta vložiť do ElasticSearchu
C. Samotné vyhľadávanie nad vloženými dátami

2. Crawler

Crawler je naprogramovaný v programovacom jazyku Java. Na začiatku (pri vybraní možnosti crawlovania webu hej.sk), sa do listu vloží počiatočný URL odkaz na stránku, na ktorej sa má začať crawlovať. Následne celé fungovanie je založené na tom, že sa prechádza tento list, až pokým nebude prázdny a vyberá sa z neho vždy ďalší URL odkaz do neho vložený.

```
while (!urlsInProccess.isEmpty()) {
                    url = urlsInProccess.pop();.....
```

Pri prechádzaní každého jedného URL odkazu sa deje to, že sa okontroluje, či sme na správnej stránke a taktiež či sa nenachádzame na URL odkaze s košíkom alebo nejakým ".doc" súborom. V celom crawleri sa používa hashmapa už navštívených URL odkazov, aby sa neprechádzali znova. Preto sa vždy kontroluje, či už daný odkaz nie je v hasmape. Ak nie je, tak sa kontroluje kanonická URLka (s obsahom ? a #) a až následne sa na neho pripojí cez knižnicu "*Jsoup.org*" a vytvorí sa z neho HTML dokument, na ktorom sa pozbierajú všetky URL odkazy.

```
connection = Jsoup.connect(url).userAgent(browser);
htmlDocument = connection.header("Accept-Encoding","gzip, deflate").get();
hrefsOnPage = htmlDocument.select("a[href]");
```

Zbierajú sa HTML stránky produktov, to znamená, že sa vždy kontroluje, či stránka, ktorá je práve prechádzaná, nie je stránka produktu. Ak nie, vloží sa len do hasmapy navštívených. Ak

áno, tak sa vyberú: **kategórie produktu, nadpisy, popisy, počet recenzií, prítomnosť dokumentácie k produktu, výrobca produktu, popisy z tabuliek a samotný URL odkaz.**

Príklad selektoru pre výber nadpisu produktu:

```
heading = htmlDocument.select(".product-summary .page-title");
```

Rovnako ako nadpis sa selektujú aj ostatné dáta zo stránky. Toto všetko sa vykonáva až kým v liste nie je žiadna URLka.

V niektorých prípadoch boli nutné modifikácie vyselektovaných dát ako napríklad počet recenzií:

```
review =
Integer.parseInt(reviews.text().split("\\(")[1].split("\\)")[0].split("
")[1]);
```

Všetky sparsované dáta sa ukladali ako JSONy do súborov (použitá knižnica *JSON.org*), príklad:

```
object.put("Odkaz", url.toString());
```

3. Vkladanie dát do ElasticSearchu

Všetky sparsované dáta sa musia uložiť do ElasticSearchu. Všetko sa to vykonáva pomocou *TransportClient* v Jave. Pripája sa na localhost, na ktorom beží ElasticSearch:

```
this.client = new PreBuiltTransportClient(Settings.EMPTY)
.addTransportAddress(new InetSocketTransportAddress(
InetAddress.getByName("localhost"), 9300));
```

Následne sa prechádzajú všetky uložené súbory s JSONmi v kódovaní UTF-8 nakoľko sa v súboroch nachádzajú slovenské a české slová a vytvorí sa index pre všetky dáta:

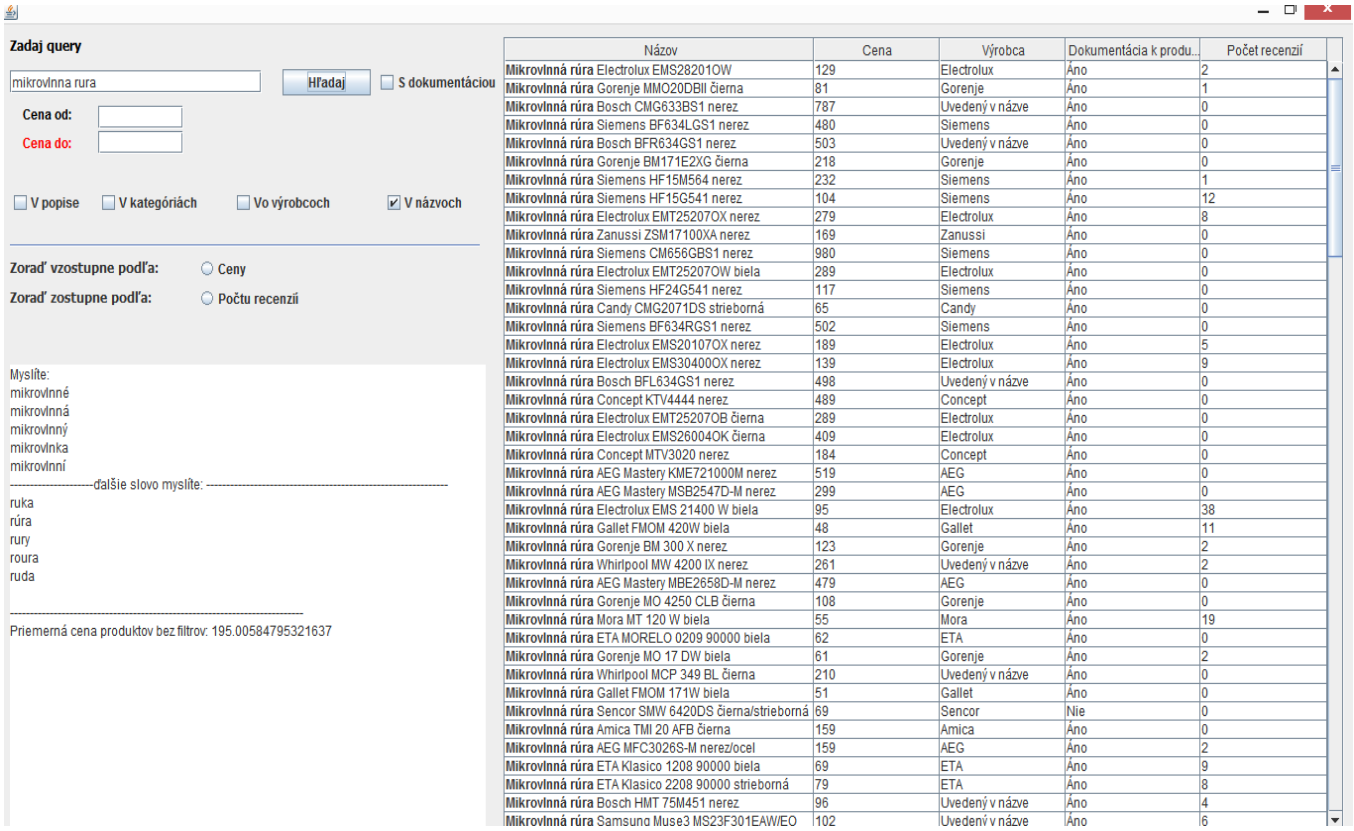
```
String json = new String(Files.readAllBytes(Paths.get(path + index)),
                         StandardCharsets.UTF_8);

IndexResponse indexes = client.prepareIndex("produkty", "produkt",
             Integer.toString(index)).setSource(json,
XContentType.JSON).get();
```

Príklad JSONu v elasticu:


```
{
    "Výrobca": "Siemens",
    "Popis": "Přáli byste si, aby vám příprava jídla přinášela stejnou radost jako stolování? Ke splnění tohoto snu vám pomůže kval
    "Kategória": ["Veľké domáce spotrebiče", "Mikrovlnné rúry"],
    "Odkaz": "https://www.hej.sk/mikrovlnna-rura-siemens-hf24g541/",
    "Cena": 117,
    "Názov": "Mikrovlnná rúra Siemens HF24G541 nerez",
    "Informácie o produkte": {
        "Hlavné informácie": ["Záruka: 24 mesiacov", "Typ: voľne stojacia", "Mikrovlnný výkon: 900 W", "Gril - výkon: 1200 W", "Gri
        "Ďalšie informácie": ["Antibakteriálny vnútorný priestor: Nie", "Ovládanie: mechanické-gombíkové, tlačidlové - zatláčacie",
    },
    "Dokumentácia k produktu": "Áno",
    "Počet recenzií": 0
}
```


4. Aplikácia vyhľadávania v dátach

Hlavné okno aplikácie vyhľadávania vyzerá takto:



Metóda *queryMaker(final TransportClient client)* v triede *Elastic*, je zodpovedná za všetky query, ktoré je možné v aplikácii spúšťať. Query je možné napísať do textové poľa a kliknúť na tlačidlo "Hľadať", vtedy sa hľadajú záznamy (by default) v názvoch produktov. Pre vyhľadávania v názvoch je použité boolQuery, cez JAVA API, ktorý používa QueryBuilder:

```
BoolQueryBuilder query = QueryBuilders.boolQuery();
```

Pokiaľ nie je zaškrtnutý checkbox "V názvoch" tak sa vykonáva boolQuery s occurrence-om *must:*

```
query.must(QueryBuilders
.matchQuery("Názov", window.queryField.getText()).minimumShouldMatch("30%")
.fuzziness(Fuzziness.AUTO).operator(Operator.AND).slop(5));
```

Inak sa používa boolQuery s occurrence-om *should* nakoľko chcem zobraziť výsledky, ktoré sú relevantné aj pre záznamy, kde sa slovo nachádza v názvoch. Používam *fuzziness* na to, aby bolo možné sa do určitej miery pomýliť v písaní slov, použitý je aj *operator AND,* aby daný názov obsahol všetky hľadané slová (nielen niektoré z nich), následne je použitý aj *slop s hodnotou 5,* na to, že hľadané slová nemusia byť bezprostredne za sebou, ale môže byť aj o 5 miest ďalej:

```
query.should(QueryBuilders
                        .matchQuery("Názov", window.queryField.getText())
                        .minimumShouldMatch("30%")
            .fuzziness(Fuzziness.AUTO).operator(Operator.AND).slop(5));
```

Vyhľadávať je možné rovnako aj kategóriách produktov, v popise produktov a aj vo výrobcoch. Možné je aj označiť, hľadať tie produkty, ktoré majú dokumentáciu. Tieto checkboxy je možné zaškrtnúť všetky naraz. Pri jednotlivých checkboxoch, okrem "V názvoch" sa používa boolQuery s occurrencom *must.*

Vyhľadávanie v popisoch je podobné ako v názvoch s pridaním *boostu*, aby za zvýšilo skóre výsledkov nájdených v popisoch:

```
query.must(QueryBuilders.matchQuery("Popis", window.queryField.getText())
.fuzziness(Fuzziness.AUTO).slop(50).boost(5));
```

Ostatné query ako hľadanie vo výrobcoch, hľadanie produktov s dokumentáciou a hľadanie v kategóriách sú takéto, všetky s occurrencom *must*:

```
query.must(QueryBuilders.matchQuery("Výrobca", window.queryField.getText())
.fuzziness(Fuzziness.AUTO));

query.must(QueryBuilders.matchQuery("Dokumentácia k produktu", "Áno"));

query.must(QueryBuilders.matchQuery("Kategória",
window.queryField.getText()).fuzziness(Fuzziness.AUTO));
```

Nakoniec sa vyskladá výsledná query cez *.prepareSearch()*, kde sa nastaví default filter pre cenu od 0 po 999999999 (táto cena môže byť taktiež zmenená používateľom a následne sú záznamny filtrované podľa zadanej ceny) a vypíše sa prvých 1000 nájdených záznamov do tabuľky v GUIku*:*

```
responses.add(client.prepareSearch().setQuery(query).setSize(1000).setPostF
ilter(QueryBuilders.rangeQuery("Cena").from(cena_od).to(cena_do)));
```

V aplikácii sa nachádzajú aj tlačidlá pre zoradenie výsledkov zostupne podľa ceny a vzostupne podľa recenzií. Zoradenie je urobené spôsobom, že nájdené výsledky sa neustále ukladajú do premennej, ku ktorej potom zoraďovanie pristupuje:

```
Iterator<String> hits = getHits(responses.get(responses.size() - 1)
.addSort("Cena", SortOrder.ASC).execute().actionGet(), urls,
model).iterator()
```

Jednoducho je len ku získaným *hits* (nájdených matchov) pridané *.addSort* podľa toho či to chceme vzostupne alebo zostupne *SortOrder.ASC* a *SortOrder.DES.*

V JSONe môžu byť tieto query vyskladané nasledovne, príklad ako je to možné:

```
{
  "query": {
    "bool" : {
      "must" : {
        "match" : {
          "Názov" : {
            "query":  "mikrovlnn rura electorlux",
            "fuzziness" : "AUTO",
            "operator": "and",
            "slop": 5
          }
        }
      },
      "filter" : {
        "range" : {
          "Cena" : { "gte" : 0, "lte" : 900 }
          }
      },
      "minimum_should_match" : "30%",
      "boost" : 1.0
    }
  },
  "sort" : [{"Cena" : {"order" : "asc"}}]
}
```

Vyššie napísaný JSON je možné upraviť na každú z daných requestov, ktoré boli v JAVE napísané cez QueryBuilder. V JSONe sa nachádza *sort* (ktorý je zapnutý pri zoradení cien, je tam možné pridať zoradenie podľa recenzií rovnakým spôsobom), nachádza sa tam *fuzziness* (ktorý používam cez queryBuilder), *boost, range, operator*, a to všetko je v boolQuery, teda presne tak ako som to robil v JAVE cez queryBuilder.

V projekte sa taktiež nachádza suggest(autocomplete), highlighting (vyznačovanie nájdených slov v názvoch) a agregácia typu priemerná cena.

Suggest(autocomplete)

Suggest je vykonávaný pomocou POST request priamo na localhost kde je spustený ElasticSearch. JSON, ktorý je vykonávaný pri autocomplete:

```
"{\r\n" +
"  \"suggest\": {\r\n" +
"    \"autocomplete\" : {\r\n" +
"      \"text\" : \""+query+"\",\r\n" +
"      \"term\" : {\r\n" +
"        \"field\" : \""+field+"\"\r\n" +
"           }\r\n" +
"      }\r\n" +
"  }\r\n" +
"}";
```

Hodnotou textu v JSONe je napísané slovo pri hľadaní daného záznamu. Hodnotou vo fielde je daný stĺpec v tabuľke (ja to vykonávam nad stĺpcom "Názov"). Všetko sa to vykonáva hneď ako používateľ napíše aspoň 5 písmen. Ak nemá čo navrhnúť tak, nenavrhne nič. V programe sa vykonáva to, že z daného *responsu*, získam objekt JSONu zo *suggest* a z neho zoberiem pole JSONov z *autocomplete.* Takto dostanem JSON, v ktorom sú *options,* kde sa nachádzajú dané návrhy. Tieto options si iba zoberiem a vypíšem.

Highlighting(vyznačovanie slov v názvoch)

Highlighting je vykonávaný pomocou GET requestu na localhost so spusteným ElasticSearchom. JSON na higlighting:

```
"{\r\n" +
"  \"query\": {\r\n" +
"    \"match\" : {" + "\""+field+"\": {\r\n" +
"           \"query\" : \""+query+"\",\r\n" +
"           \"fuzziness\": \"AUTO\"," +
"           \"operator\": \"and\"\r\n" +
"                }\r\n" +
"             }\r\n" +
"           },\r\n" +
"   \"highlight\" : {\r\n" +
"      \"fields\" : {\r" +
```

```
"        \"Názov\" : {}\r\n" +
"    }\r\n" +
"  }\r\n" +
"}";
```

Vykonáva sa to tak, že response ktorý príde vráti JSON *highlight,* v ktorom sa nachádzajú otagované slová v názvoch produktov (nakoľko robím highlight pre názvy). Príklad: rúra . Využívam automatický fuzziness na čiastočný *misspelling a* taktiež používam operátor AND, aby to v názvoch obsahovalo všetky slová, ktoré používateľ vyhľadáva.

To znamená, že je potrebné získať všetky slová medzi týmito tagmi. Tieto slová získam a následne prechádzam tabuľku s výsledkami, kde aplikujem html tag(<b>) na zvýraznenie daných slov v názvoch.

Agregácia(priemerná cena)

Query pre vypočítanie priemernej ceny produktov je možné vidieť iba pri hľadaní slov v názvoch produktov (t.j. pri zaškrtnutej položky "v názvoch") . Vytvára sa POST request na localhost s ElasticSearchom. JSON pre agregáciu je nasledovný:

```
"{\r\n" +
"  \"query\": {\r\n" +
"    \"match\" : {" + "\""+field+"\": {\r\n" +
"           \"query\" : \""+query+"\",\r\n" +
"           \"fuzziness\": \"AUTO\"," +
"           \"operator\": \"and\"\r\n" +
"                 }\r\n" +
"             }\r\n" +
"           },\r\n" +
"  \"aggs\": {\r\n" +
"    \"avg_grade\" : {\r\n" +
"           \"avg\" : {\r\n" +
"      \"field\" : \"Cena\"\r\n" +
"           }\r\n" +
"       }\r\n" +
"  }\r\n" +
"}";
```

Priemerná cena je vypočítaná z cien vrátených produktov (hits). Používam fuzziness, operator AND. Objekt JSONu, ktorý sa vráti po úspešnom vypočítaní priemernej ceny je *aggregations* a z neho je potrebné si zobrať *avg_grade* a *value*. Tu sa potom nachádza výsledná vypočítaná priemerná cena.

Pri pracovaní z JSONom v Jave je vždy nutné nastaviť na aký localhost pristupovať, aký typ requestu a následne mu poslať JSON ako string:

```
String url = "http://localhost:9200/produkty/produkt/_search";
URL obj = new URL(url);
HttpURLConnection connection = (HttpURLConnection) obj.openConnection();

connection.setRequestMethod("GET");
```

Zaujímavosť v projekte:

Aplikácia obsahuje aj jednu zaujímavú vec a tou je, že na zobrazené produkty v tabuľke je možné kliknúť a tým zobraziť daný produkt vo webovom prehliadači:

```
Desktop.getDesktop().browse(uri);
```