HW5 Report

<u>Overview of steps to accomplish the task:</u>

1) **Search Engine Setup using:**

- Apache Solr 6.4.2 core named **Indexes** for main search and snippets and another core named **suggestionEngine** to handle auto-complete requests.
- Tika Post for detection and html content extraction.
- All fields indexed to "_text_" field for default /select searches.
- Few fields like description,title and other informative fields were indexed and copied to **title_autocomplete** field of type text_auto (custom definition type,described in later pages) under the core **suggestionEngine**.
- Indexed Data for NYTimes news website , total html pages indexed: 16575.

2) **Server Setup using(server.js):**

- **NodeJs** to setup a local server on port 3000, accessible by calling **localhost:3000**.
- Accepts requests on "/" to deliver the html home page.
- Accepts request on "**/form**" to take incoming post request and query it on Solr server and returns once results obtained.
- Accepts request on "**/suggest**" to handle auto-complete requests.
- uses "**Solr-node**" and "**Solr-client**" module to query and async communicate with local Solr Cores.

3) **Client setup using(controller.js, index.html, index.css):**

- HTMLBoostrap page with search bar ,submit button,results and angular-material for **auto-complete**.
- **AngularJs** to handle query collection from HTML ,sending post request with query data to node server , collect the data returned from node server, format and update collected data on HTML page in an asynchronous way.
- CSS for styling.

4) **SpellCheck setup using(one.py , ParseWeb.java):**

- **Norvig's Spell Checker** in **Python-2.7-Anaconda** shell with custom strictly-English corpus extracted using **Java JSoup** library.
- Using **Tika Language Identifier** to get English Text only.
- This corpus, **big.txt** file contains all the English sentences from over 16575 websites ,compressed to 27 MB.
- Node Server responsible for running spell-check program in a separate parallel process when query is submitted.

5) **Snippet Generation using(extract.py):**

- **Python2.7-Anaconda** Shell, **NLTK** stop words library.
- **BeautifulSoup** library to extract query-relevant content at run-time**.**
- **Regexx** to filter undesirable chars.

Detailed Explanation of Spell-Check implementation.

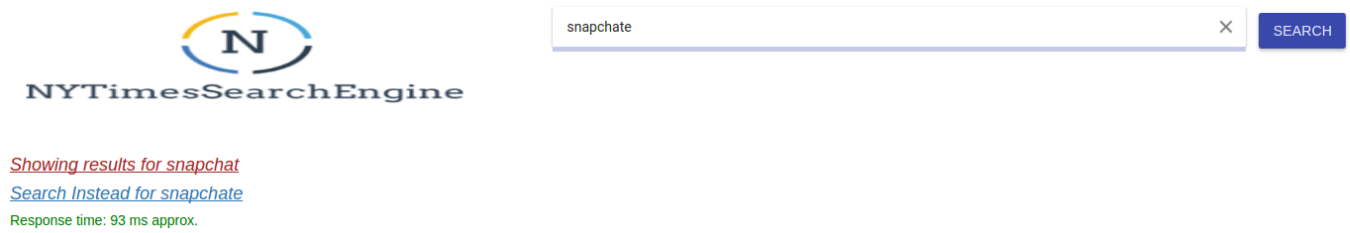
Everytime a query is submitted , the query is tokenized and Norvig's spellchecker is run in a separate child process using nodeJs child_process as follows:

```
var spawn1 = require("child_process").spawn;
var process1 = spawn1('python',["path/SearchEngine/one.py",req.body.searchtext]);
process1.stdout.on('data', function (data){……..
```

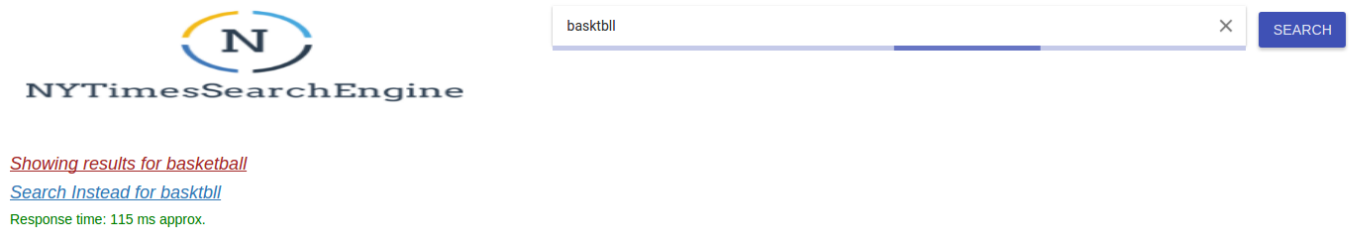
on getting results to emulate Google, the UI goes through three checks based on the spell-check return value, 3 checks are : "Main Submit(First Submit)","Showing results for", "Search Instead"(if spell error found) and on clicking "Search Instead" back to "Did you Mean" check.

Five tests for spell-check :

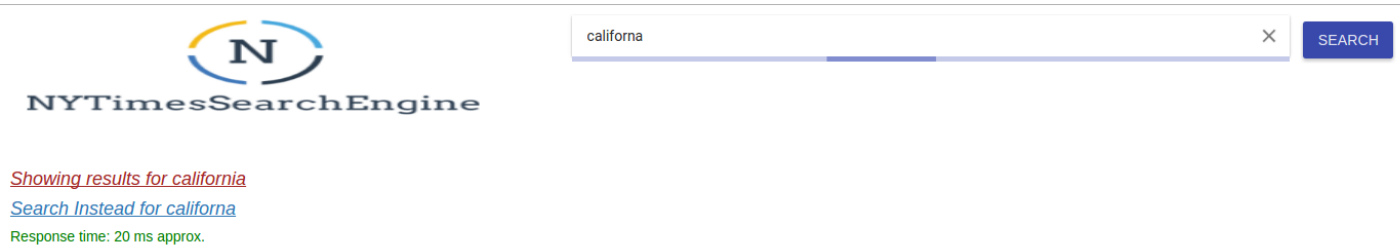
1) s**napchate** → **snapchat** (Edit Distance 1)



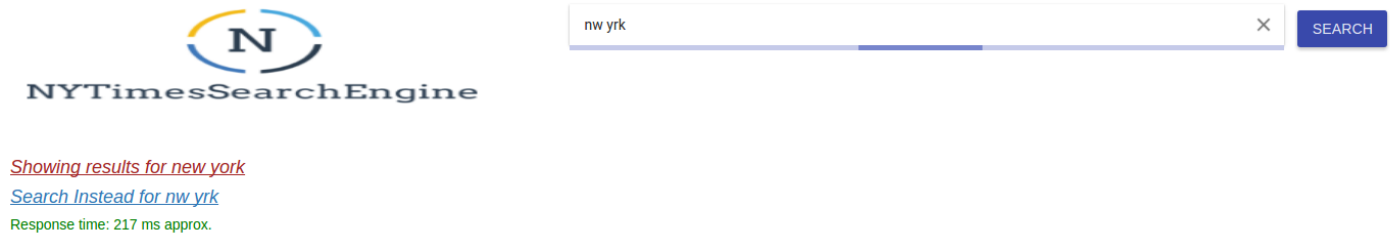
2) **basktbll** → **basketball** (edit Distance 2)



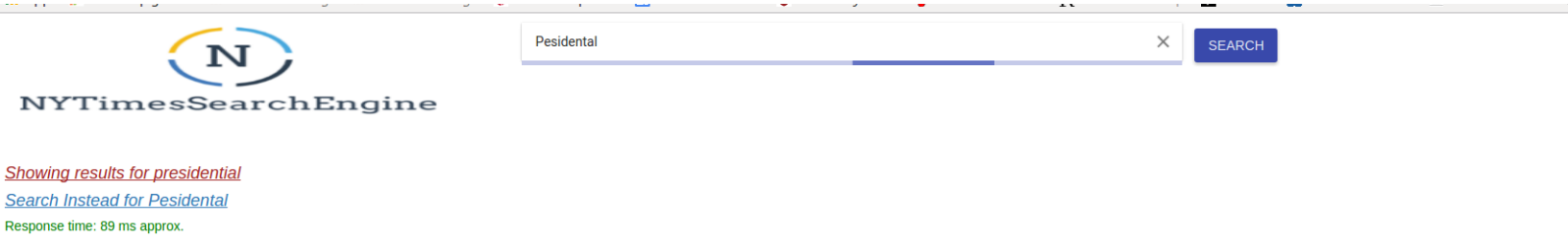
3) **californa** → **california** (edit Distance 1)



4) **nw yrk** → **new york** (multi term fix)



5) **pesidental → presidential** (edit distance 2)



Detailed Explanation of auto-complete implementation.

To implement the auto-complete suggestions ,a separate core called suggestionEngine is being used. This strictly handles requests on "/suggest" request handler to send suggestions.
A new custom field type was created with "KeywordTokenizerFactory" to handle phrase matched suggestions.

```
<fieldType class="solr.TextField" name="text_auto">

   <analyzer type="index">
     <tokenizer class="solr.KeywordTokenizerFactory"/>
     <filter class="solr.StopFilterFactory" words="lang/stopwords_en.txt" ignoreCase="true"/>
     <filter class="solr.LowerCaseFilterFactory"/>
     <filter class="solr.EnglishPossessiveFilterFactory"/>
     <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>
     <filter class="solr.PorterStemFilterFactory"/>
   </analyzer>
   <analyzer type="query">
     <tokenizer class="solr.KeywordTokenizerFactory"/>
     <filter class="solr.SynonymFilterFactory" expand="true" ignoreCase="true"
synonyms="synonyms.txt"/>
     <filter class="solr.StopFilterFactory" words="lang/stopwords_en.txt" ignoreCase="true"/>
     <filter class="solr.LowerCaseFilterFactory"/>
     <filter class="solr.EnglishPossessiveFilterFactory"/>
     <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>
     <filter class="solr.PorterStemFilterFactory"/>
   </analyzer>

  </fieldType>
```

And using this field type a field called title_autocomplete to query on for suggestion.

```
<field name="title_autocomplete" type="text_auto" indexed="true" stored="true"
multiValued="true" />
<copyField source="body" dest="title_autocomplete" />
<copyField source="author" dest="title_autocomplete" />
<copyField source="title" dest="title_autocomplete" />
<copyField source="description" dest="title_autocomplete" />
```

I have used solr.SpellCheckComponent and FuzzyLookUpFactory to implement suggestions.

Solr Search Suggest Component definition :

```
  <searchComponent name="suggest" class="solr.SpellCheckComponent">
   <lst name="spellchecker">
   <str name="name">suggest</str>
   <str name="classname">org.apache.solr.spelling.suggest.Suggester</str>
   <str name="lookupImpl">FuzzyLookupFactory</str>
   <str name="suggestAnalyzerFieldType">text_auto</str>
   <str name="field">title_autocomplete</str>
  </lst>
 </searchComponent>
```

Solr Search Request Handler definition :

```
<requestHandler name="/suggest" class="org.apache.solr.handler.component.SearchHandler">
  <lst name="defaults">
   <str name="spellcheck">true</str>
   <str name="spellcheck.dictionary">suggest</str>
   <str name="spellcheck.count">10</str>
   <str name="spellcheck.collate">true</str>
  </lst>
  <arr name="components">
   <str>suggest</str>
  </arr>
 </requestHandler>
```

Suggestions can be obtained using **spellcheck.q param** from the query of the form:

http://localhost:8983/solr/suggestionEngine/suggest?indent=on&spellcheck.q=aloha&spellcheck=on&wt=json

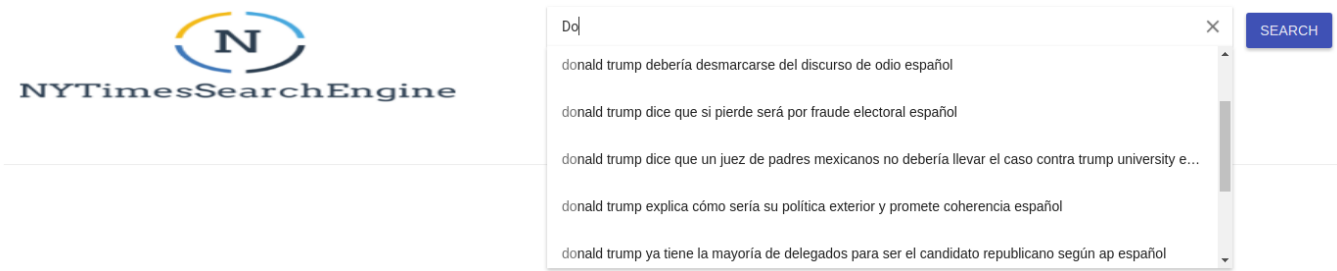
Sample response:


```
{
  "responseHeader":{
    "status":0,
    "QTime":1},
  "spellcheck":{
    "suggestions":[
      "aloha",{
        "numFound":3,
        "startOffset":0,
        "endOffset":5,
        "suggestion":["aloha by roy lichtenstein",
          "alphabet expande su proyecto de viajes compartidos y amenaza a uber – español",
          "alphabet – español"]}],
    "collations":[
      "collation","(aloha by roy lichtenstein)"]}}
```

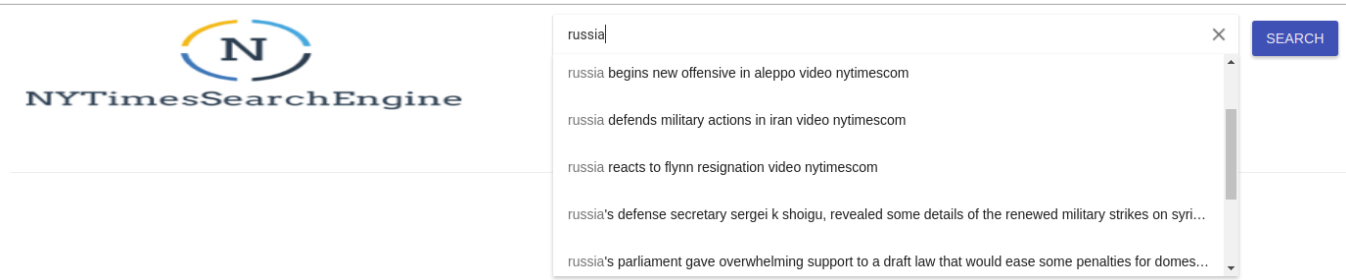
5 Tests for Auto-Complete:

1) California related

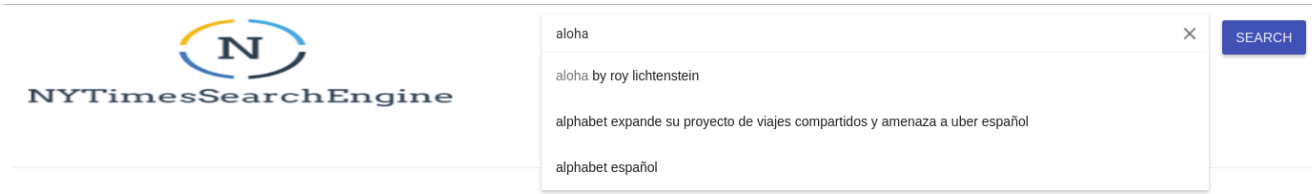
2) Donald Trump related

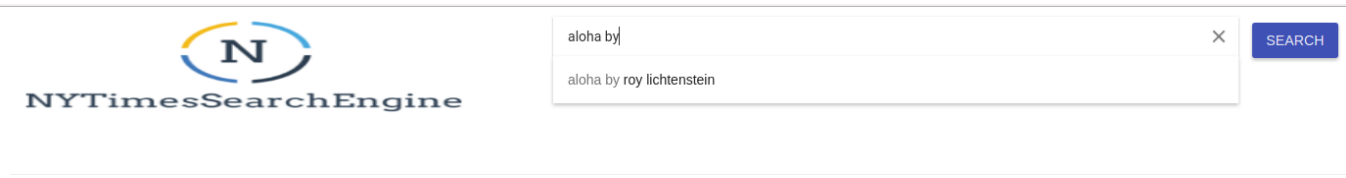


3) Russia Related



4) Changing Autosuggestions with words:





5) Soldier related