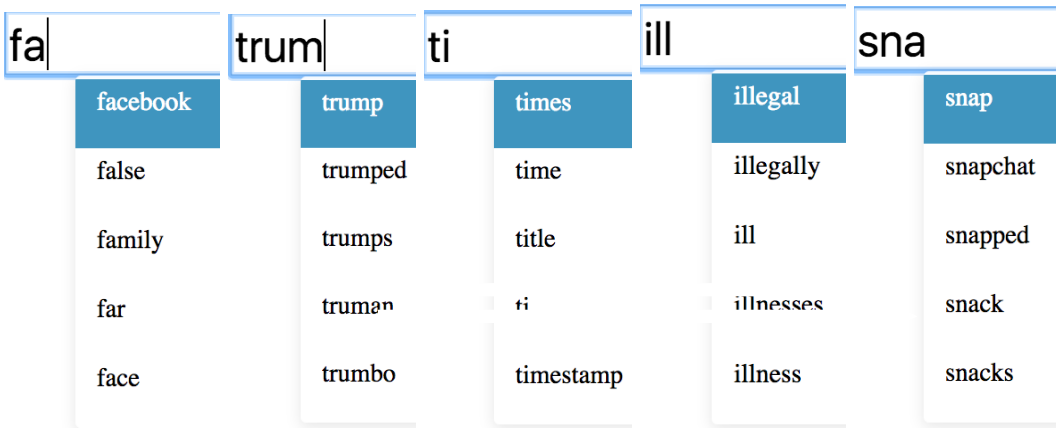
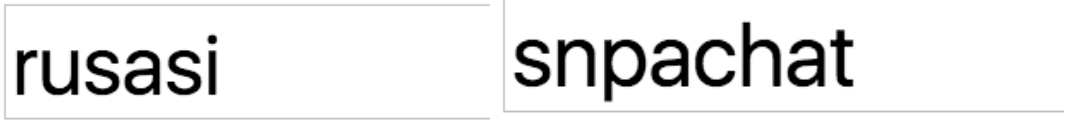
572 HW5 Report

junzhiwa@usc.edu USC-ID:1522904892

Analysis of result: autocomplete:



Spell correction:



Total 2 results
Did you mean: russi?

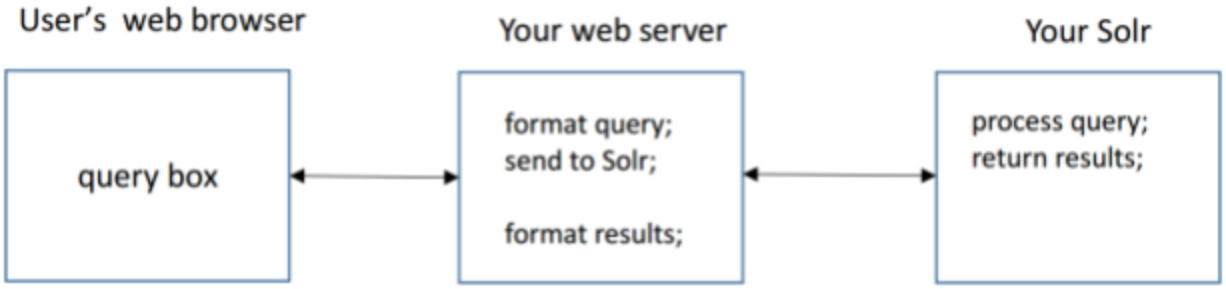
Total 277 results
Did you mean: snapchat?

Total 91 results
Did you mean: brexit?

Total 4042 results
Did you mean: donald?

Total 2974 results
Did you mean: immigration?

Below is a rough outline of my project below.



The project consists of mainly 3 modules: frontend, web server and Apache Solr server.

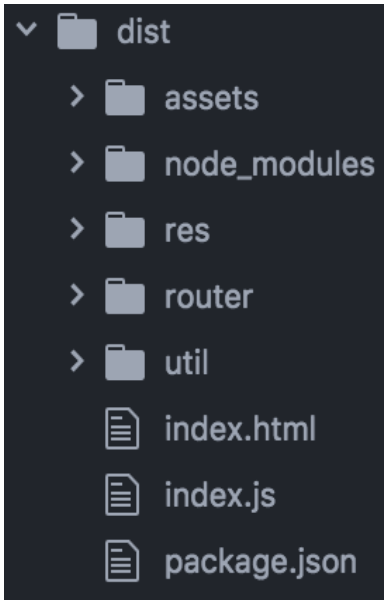
Frontend is a single page website which implements the display of results of query, autocomplete and spell correction. It sends requests to my localhost backend, fetch, format and display the result received from the backend.

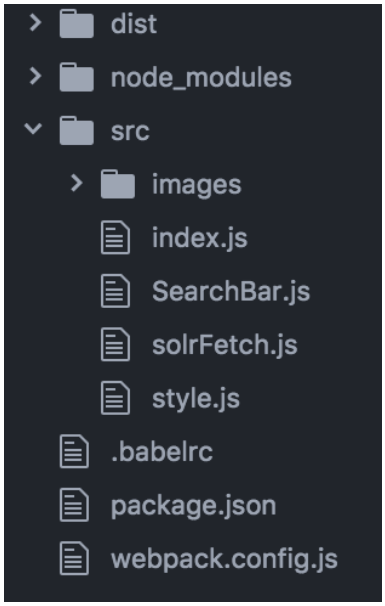
below is my frontend project directory. The main stacks I use are React, Bootstrap, and Webpack. The autocomplete component I use is here:
https://github.com/reactjs/react-autocomplete

Backend is responsible for communication with Apache Solr server and frontend. It receives HTTP Get request from frontend, parses the url and format the query url and sends it to Solr.

The util directory consists of 3 main modules. ContentHelper.js is responsible for parsing website content and formatting snippet for each query-doc pair. speller.js is responsible for spell error detection and correction. solrFetcher.js is responsible for fetching query results from Solr server and formatting it.

Below is my backend project directory. The main stacks I use is ES6, Node and Express.





Backend directory Frontend directory

Spell error detection and correction:

I use the code in this url:
http://blog.astithas.com/2009/08/spell-checking-in-javascript.html
Since this code contains some bugs therefore doesn't work. I made some modifications and you can see the modified code in my /util/speller.js.
The main idea is:
Train step: the program loads big.txt file as the train text, and generates a dictionary.txt file based on train. The dictionary.txt is a stringified JSON Object recording the terms and their frequency in the big.txt file.
Apply step: the program receives a word, say A

1. Check if A is in the dictionary, if yes, return it.
2. Generate all 1-edit-distance (Levenshtein Algorithm) words of A as candidates. Filter those in the dictionary, return the term that has the highest frequency.
3. If no candidates left, generate all 2-edit-distance words of A as candidates. Then do the same thing as step2.

Generate big.txt
I use Apache Tika as the html parse tool and transfer it to text. The code is attached.

Autosuggest

I use Solr native autosuggest module to implement it.
Below is a screenshot of part of my configuration file solrconfig.xml.

```
<requestHandler name="/suggest" class="solr.SearchHandler">
  <lst name="defaults">
    <str name="suggest">true</str>
    <str name="suggest.count">20</str>
    <str name="suggest.dictionary">suggest</str>
  </lst>
  <arr name="components">
    <str>suggest</str>
  </arr>
</requestHandler>
```

```
<searchComponent name="suggest" class="solr.SuggestComponent">
  <lst name="suggester">
    <str name="name">suggest</str>
    <str name="lookupImpl">FuzzyLookupFactory</str>
    <str name="nonFuzzyPrefix">3</str>
    <str name="field">_text_</str>
    <str name="suggestAnalyzerFieldType">string</str>
    <str name="buildOnOptimize">true</str>
  </lst>
</searchComponent>
```

The property of “nonFuzzyPrefix” is the number of prefix characters match. Here we set 3, means suggested words have at least 3 characters prefix as same as your query word. The property of suggest.count is the number of suggested words returned. Since some suggested words are not alphabetic, I process them in my backend solrFecther.js file, exclude non-alphabetic, and return top 5 words with highest weight to frontend.

Snippet:

The snippet function is processed by ContentHelper.js, it accepts the doc file path, and fetch the corresponding txt file parsed by Apache Tika in Java, splits the text by “/[\n]{2,}/”. That means I splits the content mostly by paragraph, sometimes maybe unstable. And return the first paragraph that contains the query terms.