For this assignment, I have extended the search engine which I wrote for homework 4 using Mercury News data. The features added were autocomplete, spell correction, and snippet generation.

The features adhered to the following specifications: For autocomplete, up to ten suggestions were to be generated upon every character entry, starting with the first character. For spell correction, a non-solr spell checker was to be integrated and correct queries once submitted. And for snippet generation, the documents retrieved were to be parsed for text containing the query terms. Further design decisions made are explained below.

Prior to working on specific features, two modifications were made. The Solr configuration was modified and the query default was change to 'AND' instead of 'OR'. This was done by adding a line to "solrconfig.xml". This modification was made based on the suggestion from the specifications document in order to simplify the snippet process for multiword queries. Also, there was a major reorganization of code which separated php and php generated html from raw html.

Spell correction is done on each individual word of the query upon submission. Spell correction is only done if the original query doesn't produce any results and search results for the corrected query are automatically returned along along with a message communicating that a correction was made.

The spell correction functionality relies is handled by an external php program based of cody by Peter Norvig. The source can be found at:
http://www.phpclasses.org/package/4859-PHP-Suggest-corrected-spelling-text-in-pure-PHP.html#download
"SpellCorrector.php" is called by the main program "webapp2.php". When the json field numFound equals 0, the original query is exploded and each word is passed to the spell corrector. The results are combined into a query which is then sent to Solr to be processed.

"SpellCorrector.php" depends on "big.txt". "big.txt" is an amalgam of text used to form a vocab for the corrector. In order to produce relevant corrections, it was necessary to parse the websites which form the corpus and collect words. The Apache Tika library was used for this purpose. Using code from TutorialsPoint as reference, a java program was created that iterated through the corpus of web pages provided for homework 4 and parsed the text.

Autocomplete occurs whenever a character is entered and beings with the first character. For simplicity, the search engine only suggests based on the the final word. For example, "Donlad Trum" autocompletes to "Donlad Trump".

The autocomplete feature utilizes the SuggestComponent provided by Solr and required a search component and respective request handler be added to "solrconfig.xml".

A separate program, “auto.php”, handles autocomplete, receiving partial queries from the main program, identifying the last word, submitting requests to Solr, and funneling results back to be displayed by the main program. The main program interfaces with “auto.php” using ajax/jquery.

Snippets are shown for every returned document and give context as to where in the document query terms lie. As per my implementation, only the text portion of retrieved html documents are analyzed and the returned context is a sentence containing all keywords. The assumption made is that a period serves as the end of a sentence, but this can lead to odd edge cases where “Donald Trump Jr.” is considered a full sentence. If the query terms are not found an error message appears in place of suggesting that these terms may be in non-text portions of the html.

To implement snippets, I had to install/use DOM php module. After the top suggested documents are retrieved, the DOM module is used to scrape the paragraph text from each site. The text is then split into sentences and matching is done on the keywords of the query.

Upon initial testing, the system works as intended. Sample output is shown:
Five examples of spell correct working:

1. [Donlad trumf] produces [donald trump]
2. [Paur Allan] produces [paul allen]
3. [Carefornia] produces [california]
4. [Chicigo] produces [chicago]
5. [Fakebook] produces [facebook]

Five examples of auto-complete working:

1. [Reddi] suggests [Reddit]
2. [Faceb] suggests [Facebook]
3. [Microso] suggests [Microsoft]
4. [Chicag] suggests [Chicago]
5. [Mercede] suggests [Mercedes]

There was a terrible mistake related to spell correction only realized after the demo:
In the Java program used to produce “big.txt”:

Correct: FileInputStream inputstream = new FileInputStream(f);

Original: FileInputStream inputstream = new FileInputStream(new File(“./home/pradeep/Desktop/test.html”)); // For the purpose of testing

As can be seen, the original program only used one file (Repeated n times) to produce “big.txt”, and this lead to a considerable vocabulary shrinkage. Rerunning the program and swapping out the file drastically improved results.

For future iterations, there are a variety of improvements which could be made. Speed is one. The speed at which the query is processed and corrected is pathetically slow, and there is a good 10-15 second lag between when the user hits submit and when results are displayed.
For autocomplete, there should be better filtering as a lot of “garbage” seems to come up especially for the initial couple of letters. Snippets can be improved by looking at non-text and context could be improved by matching a cloud of words surrounding the query terms and not simply just words in the same sentence. Finally, the ui looks disgusting…But then again, it’s my first time developing a webapp.