Creating a Type-ahead or Suggestions for Your Search with Crafter Search

Options

- Solr Suggest feature
    - Leverages a dictionary to suggest possible values (not your indexed content)
    - http://www.opensourceconnections.com/blog/2013/06/08/advanced-suggest-as-you-type-with-solr/
- Facets
    - Leverages the content in your index

I believe the second option is more meaningful in that it will return results that are more relevant and actionable for the user. This is the option we will look at in this tutorial.

How to

Step 1: Create a Service

Create a REST service for search in your site.

Requirements:

- The service will take the user's current search term and perform a search.
- The service will return the results
- The service will return the number of results found
- The service will return suggestions about what therms the user might be looking for

Our example will focus on PRODUCTS where the user's QUERY is referring to the PRODUCT's TITLE

To create the service, place the following GROOVY file in your scripts folder

/scripts/rest/search/products.get.groovy

```
// get the query parameter
def q = params['query'];
def result = [:]

// build the query statement
def queryStatement = "crafterSite:\"rosie\" ";
queryStatement    += "AND content-type:\"/component/jeans\" ";
queryStatement    += "AND productTitle:*"+q+"* ";

// construct and query operation object
def query = searchService.createQuery();
query = query.setQuery(queryStatement);
query = query.addParam("facet","on");
query = query.addParam("facet.field","productTitle");
query = query.addParam("facet.prefix", q);

// execute the query operation and get the result values
def executedQuery = searchService.search(query);
def productsFound = executedQuery.response.numFound;

// build product results
def products = executedQuery.response.documents;
result.productsFound = productsFound;
result.products = [];
products.each {
   product ->
   def productResult = [:];
   productResult.title = product.productTitle;
   productResult.abc = "abc";
   result.products.add(productResult);
}

// build suggestions
def productNames = executedQuery.facet_counts.facet_fields['productTitle'];
result.suggestions = [];
productNames.each {
   productName ->
   if(productName.value > 0) {
       result.suggestions.add(productName.key);
   }
}

// query diagnostics
result.s = [:];
result.s.q = q;
result.s.resultsFound = productsFound;
//result.s.results = products;

return result;
```

Step 2: Build the UI

The front end experience is built with HTML, Javascript and specifically AJAX

Requirements

- When the user types a value send a request to the server to get instant results
- Display the results and show suggestions about what the user might be looking for
- DO NOT fire a query for every keystroke. This can lead to more load than necessary, instead, batch user keystrokes and send when batch size is hit OR the user stops typing

```
$('#products, #insearch input').typeahead({
           source: function (query, process) {
                      return $.get('/api/1/services/search/suggest.json',
                                  { query: query },
                                  function (data) {
                                     return process(data.suggestions);
                                  })
           }
});
```

Step 3: Add the Search Component to your Template

The search code we built above is 100% Javascript / and CSS. Now we want to incorporate it in to our template. See below:

```
TODO
```