Textkernel Search! REST API Cheatsheet

Overview

Search! is a semantic search engine and matching engine. Documents are represented as a collection of relevant fields (jobtitle, degree_direction) and each field can have multiple values. For the HR Hackathon we have 2 separate “repositories” (environments) holding candidates profiles (hackathon_cv) and vacancies (hackathon_vac).

Authentication

Every Search! environment has an associated password. However, for this event, we set up a temporary password-free environment. You however need to retrieve an *accessToken* and then **provide it to every** API call as an additional parameter (unlisted in this documentation). You can get one via HTTP:
$ curl *SEARCH_URL/requestAccessToken* -d ‘environment=hackathon_cv’
$ “4a9aea35-91c0-41da-9741-74dd74a21a8f”

Searching

SERVICE:
GET *SEARCH_URL/search*
PARAMETERS:
action => string: **must** be search
accessRoles => list of strings: access roles on documents, default [all] is ok
request => object {
query => string: user typed query - see query language desc (empty)
queryParts => list of **QueryPart**: parsed parts of a previous query, used together with current query (empty)
resultOffset => int: requested result item offset - pagination (0)
provideTagcloud => string: the field name of a requested **Tagcloud** (empty)
suppressResultList => boolean: whether *not* to return results (false)
inputLanguage => string: used to translate codes of the input query (empty)
outputLanguage => string: used to translate codes in the results (empty)
suppressCorrection => boolean: whether *not* to add corrected versions of keywords automatically (false)
}
RETURNS:
SearchResult => object {
matchSize => int: *total* number (not the *returned* number) of matching docs
hasMoreResults => boolean: whether **matchSize** > returned number of results
resultItems => list of **ResultItem** objects: the actual results
queryParts => list of QueryPart object: all parsed parts of executed query
newQueryParts => list of QueryPart object: new parsed parts of executed query

isOrCombined => boolean: whether input query and **queryParts** were ORed
facetCounts => map string -> **ItemCounts** objects: facet item counts
tagcloud => **Tagcloud** object: tagcloud if requested
searchEngine => string: search engine queried
synonyms => map string -> list of **SynonymSection** objects: synonyms added to each query part
}

where
ResultItem => object {
docID => string: id used when indexing
score => float: score of item, with respect to performed query
fields => list of string: fields of the item
queryPartScores => list of list of integers: indicates which query parts are matched (1) or not (0) by the item
}
ItemCounts => object {
itemCounts => map string -> integer: aggregated values -> counts for query
}
Tagcloud => object {
field => string: name of field the tagcloud is built upon
items => list of string: tagcloud items
weight => list of float: item aligned list of weights signaling “popularity” over total
}
SynonymSection => object {
name => string: synonyms section name
collapsed => boolean: whether the section was collapsed in the UI
items => list of **SynonymItem** object {
lang => string: the language of the synonyms
values => list of strings: the synonyms
}
}

Example
$ curl -GET ‘SEARCH_URL/search’ -d 'accessToken=4a9aea35-91c0-41da-9741-74dd74a21a8f&action=search&environment=hackathon_cv&request={
"query" : "java",
"outputLanguage" : "de"
}'
$ {
"matchSize":2,

```
        "hasMoreResults":false,
        "resultItems":[
            {
              "docID":"99",
              "fields":[
                        ["Rocio Tejkl"],
                        [ "Delphi und Web Softwareentwickler"],
                        [ "Stuttgart"],
                        [ "C", "C++", "PHP","JS"]
               ],
               "score":1.0,
               "queryPartScores":[1]
               },
           {..}
        ]
        "queryParts":[... ],
        "newQueryParts":[ ],
        "isOrCombined":false,
        "facetCounts":{
            "educationlevel":{
              "itemCounts":{
                "Schulabschluß":1,
                "Universität":1
              }
            }
        }
        "searchEngine":"hackathoncv",
        "synonyms":{ ... }
}
```


Getting facet items and counts (aggregations)

To aggregate data within the context of a query, you can use **facetCounts** or a **Tagcloud**. Both of them are returned in response to a search call (as visible above), set **suppressResultList** to true if you are not interested in the results themselves. If you want to get values and counts for all the fields, you can use **facetCounts**, while a Tagcloud will return you just a list of the most popular items with weights reflecting this popularity

Example:


```
$ curl 'SEARCH_URL/search' -d
'accessToken=4176e929-2ec1-4588-b8ff-4c2972239801&action=search&environment=hackathon_cv&request={
"query" : "manager",
"outputLanguage" : "de",
```

```
"provideTagcloud" : "compskills",
"suppressResultList" : true
}'
$ {
  "matchSize":36,
  "hasMoreResults":true,
  "queryParts":[ ... ],
  "newQueryParts":[ ... ],
  "isOrCombined":false,
  "facetCounts":{ ... },
    "tagcloud":{
    "field":"compskills",
    "items":[
      "Excel",
      "Word",
      "PowerPoint",
      "C++",
      "MS Office",
      "Java"
    ],
    "weights":[
      11,
      10,
      8,
      5,
      4,
      1
    ]
  },
  "searchEngine":"hackathoncv",
  "synonyms":{ ... }
}
```


Matching

When matching is enabled, Search! allows you to match a vacancy to a list of relevant profile or vice-versa. Doing so using the API involves 2 calls. For example, to match a vacancy to a list of profiles we need to perform:

- A “query extraction” call to a Search! environment indexing profiles
- A “search” call to a Search! environment indexing profiles using the query previously returned

Search! can extract queries from either URLs or files directly, using multipart form request.

SERVICE:
GET *SEARCH_URL/queryExtractionUrl*
PARAMETERS:
url => string: URL of the page to extract
RETURNS:
=> string: query in Search! query language

SERVICE:
POST*SEARCH_URL/queryExtractionFile*
PARAMETERS:
file => byte array: file to use for matching
RETURNS:
=> string: query in Search! query language

Examples:
$ curl 'SEARCH_URL/queryExtractionUrl' -d 'accessToken=3aa81d64-d22a-44bc-9a13-61df5a75d979&url=http://stellenanzeige.monster.de/Projektleiter-m-w-Job-Berlin-Berlin-Deutschland-151305570.aspx?jobPosition=9'
$ "%jobtitlesonlyrecent:[projektleiter] #1.0 %jobcodesonlyrecent:\"4759\" %jobclassidsonlyrecent:\"2\" %jobgroupidsonlyrecent:\"284\" %city:\"Berlin\"+50 %educationlevel:(%4 #1.0) %langskills:EN #1.0 %experienceyears:(1..2 #1.0 3..5 #1.0) KFZ # Führerschein # Holztechnik # Innenausbau # AutoCAD # Teamfähigkeit # Organisationstalent # Planung # Führung # Kommunikation # Innenarchitektur # hohe Termintreue # Kalkulation # Realisierung von Messebauten # Anleitung # Bau # vergleichbarer Studiengang # AutoCAD 3D # Meisterabschluss"

$ curl -POST -F accessToken=c4ffbfad-0ed2-4ec5-959e-79f2bf8adf5b -F file=@vac1.html 'SEARCH_URL/queryExtractionFile'
$ "%jobtitlesonlyrecent:[projektleiter] #1.0 %jobcodesonlyrecent:\"4759\" %jobclassidsonlyrecent:\"2\" %jobgroupidsonlyrecent:\"284\" %city:\"Berlin\"+50 %educationlevel:(%4 #1.0) %langskills:EN #1.0 %experienceyears:(1..2 #1.0 3..5 #1.0) KFZ"

Indexing

You can add new documents to the search environment: to make it visible only to you just add a custom role.
SERVICE:
GET S*EARCH_URL/indexing*
PARAMETERS:
action => string: must be either **add, delete** or **update**
documentID => string: identifier of document to act on [ADD, UPDATE]

documentIDs => list of string: in case of deletions, docs ids to delete [DEL]
documentDate => string: date in ISO format yyyy-mm-dd [ADD]
document => byte array: document to index in XML (trxml extension) [ADD]
accessRoles => list of strings: roles for access. Use all plus your custom role [ADD]
field => string: name of field for which to change values [UPDATE]
value => list of strings: new values for field (can be empty) [UPDATE]
RETURNS:
=> OK, or object with errorCode if it occurred

Example:
$ curl -X POST -F accessToken=c4ffbfad-0ed2-4ec5-959e-79f2bf8adf5b -F 'document=@example1.trxml' -F 'action=add' -F 'documentDate=2015-02-12' -F 'documentID=ID-1" accessRoles=[all,myrole]' '*SEARCH_URL*/indexing'
$ OK

Autocomplete

Search! comes with a handy, multilingual autocomplete service

SERVICE:
GET*SEARCH_URL/complete*
PARAMETERS:
field => string: field to retrieve completions for
language => string: language of the completions (optional)
input => string: user-typed string, 2 or more characters.
RETURNS:
=> list of strings: list of completions. If field was "FULLTEXT", suggestions are in the form FIELD:SUGGESTION

Example
$ curl -GET 'SEARCH_URL/complete' -d 'accessToken=872ff735-ae9f-438e-9e7a-dc255c911036&language=de&input=pr&field=FULLTEXT
$ ["\"produkte\"","jobdescriptions:\"project management\"","lastjobtitle:\"project manager\"","jobtitlesonlyrecent:\"project manager\"","jobtitles:\"project manager\""]

APPENDIX: Search Query Language

Keywords

Example: *Java developer Amsterdam*
If a keyword contains special characters, such as [#$%.+/'], it will be interpreted as a phrase (see phrases below) and the special characters will be ignored. There are a number of cases where special characters are not ignored:

• internal apostrophes: e.g. O'Reilly, O'Reilly's
• acronyms: e.g. U.S.A., I.B.M.
• company names with & or @: e.g. AT&T, Excite@Home
• email addresses and host names: e.g. jack@mail.com, www.geocities.com
• common technical terms: e.g. C++, C#, and SQL*Plus
Keywords are not case sensitive, that is "Java", "JAVA" and "java" are the same.
Adding keywords to a query further limits the result set. In the example above, adding
Amsterdam
will return results containing Amsterdam in addition to Java and Developer. An empty query, that is without any keywords, returns all.

Phrases

Example: *"Java developer" Amsterdam*
Phrases are used to match a sequence of words. Special characters are ignored inside a phrase.

Proximity

Examples:
[Java developer] Amsterdam
[developer C++ Java]
Query terms within [] match if they occur in the document in any order, possibly with one or two words in between. Proximity matching is more flexible than phrases, e.g. the above query [Java developer] also matches documents containing “java software developer” and "java enterprise software developer".

Wildcard

Example: *develop**
Search! allows to perform wildcard queries. If a query term ends with a trailing * it gets expanded to the most common completions. Documents containing any of the found completions will match. For example, the above query develop* will match on developer, development, developed, etc. A few restrictions apply:
• The * symbol must be placed at the end of a word.
• The * symbol needs to be preceded by at least 2 characters for it to work on external searchers.
• Wildcard terms cannot be part of phrases or proximity expressions.

Weighting

Example: *developer [software engineer] #1.5 "software architect" #0.8*
Terms are weighted within a query by adding a number weight behind the term. If no weight is specified, a term receives a default weight of #1.0. The weight influences the ranking of documents of the weighted query expression in relation to the other expressions of the same

field. Weights can also be assigned to more complex query expressions, such as phrases, conditions or range queries.

Fields

Examples:
Java developer city:Amsterdam
experience:[Java developer] Amsterdam
If a term, phrase or proximity expression should be matched on a certain metadata field or section of a document, the query expression needs to be preceded by the field name and a colon. The above query searches for the word Amsterdam only in the city field of the document, respectively for java developer only in the experience section.

Numeric / Date Range Conditions

Examples:
Java developer experience:>2
Java developer experience:5..10
Java developer date:<2010-10-01
Numeric or date (range) conditions are expressed by "<",">","=" and "..". Dates must be of yyyy-mm-dd format. Numeric or date conditions are always assigned to a specified field.

Location Conditions

Examples:
Java developer location:Amsterdam+20
Java developer location:1018+20
Java developer location:"New York"+25
location:"33.14 -12.25"+50
location:"33.14 -12.25"
Location (+radius) conditions are expressed by naming a location (city name, postal code, geo coordinates) followed by a radius in kilometers after the "+" symbol. If the radius is omitted only exact matches are returned.

Nice-to-have Expressions

Example: *"Java developer" %javascript*
Nice-to-have search expressions are marked with a preceding %. They do not limit the result set to results having that search term, they only influence the ranking of the search results. The example query above will return all Java developers, with or without javascript in their CV,
but the ones that also contain javascript will be ranked higher in the results. If a query consists solely of nice-to-have expressions, than the result consists of all documents ranked by the nice-to-have expressions. In the API, the nice-to-have query parts are marked with condition FAVORED.

Should-have Expressions

Example: *"Java developer" ^javascript*
Should-have search expressions are marked with a preceding ^. Their behavior is identical to Nice-to-have Expressions except that they carry more weight when modifying the ranking.
In the API, the should-have query parts are marked with condition STRONGLY_FAVORED.

Banned Expressions

Example: *Java developer -city:Amsterdam*
Query expressions marked by a preceding - are banned expressions, they exclude all documents matching this expression from the result.