Node.js: create a site search engine with MongoDB

One of the main problems for those who come from a MySQL-based development background is to stop forcing MongoDB to work like MySQL. MongoDB has other features that are really handy when it comes to build a site search engine in Node.js.

MySQL: stopwords, LIKE, AND and OR

If you take a look at the source code of the WordPress WP_Query class (located under `/wp-includes` in every WordPress installation), you'll probably stop reading at line 1294 where you'll encounter the first of several methods that WP uses to implement its search engine.

Basically, WP runs the following routines:

- sanitizes the query string
- splits the query string into terms
- removes all the stopwords from the array of terms
- builds the MySQL query using the `LIKE` operator with `AND`/`OR` on the title, excerpt and content of posts

This technique is designed to return the most specific posts as the first results of the search query: if you have entered `install mysql on mac` in the search field, your query will become `install mysql mac` because `on` is a stopword. Then you'll get the following combinations in priority order:

1. `install mysql mac`
2. `install mysql`
3. `install`, `mysql`, `mac` as separate terms

Does it work? Yes.

Don't reivent the wheel

The `LIKE` operator doesn't exist in MongoDB. This is not a problem. The real problem here is to choose between two MongoDB operators that many people think are similar, namely `$regex` and `$text`.

The former implements regular expressions and doesn't require an index, the latter uses substrings and needs an index on the selected property of a document.

I didn't test myself the actual impact on performance of both operators, so I can't recommend you what is the best choice here from a pure performance perspective. Feel free to experiment.

As said above, with `$text` you need to create indexes:

```
db.posts.createIndex({title: "text"});
db.posts.createIndex({content: "text"});
db.posts.createIndex({excerpt: "text"});
```

Then you can run a query similar to the following:

```
db.posts.find({'$and': [{'title': {'$text':
'query'}}, {'content': {'$text': 'query'}},
{'excerpt': {'$text': 'query'}}]});
```

With the `$and` operator we're **including** possible alternatives, with `$or` instead we're **excluding** some alternatives.

Since `$text` doesn't use regular expressions, `install` and `Install` are not the same thing. The match here is exact, so in order to get more results we should use `$regex`:

```
db.posts.find({'$and': [{'title': {'$regex': 'query',
'$options': 'i'}}, {'content': {'$regex': 'query',
'$options': 'i'}}, {'excerpt': {'$regex': 'query',
'$options': 'i'}}]});
```

The implementation

In Node.js we could write something like this:

```
'use strict';

const stopwords = require('./src/stopwords');
const Docs = require('./src/db/Docs');
const app = require('express')();

let queryVar = function(str) {
    let q = str.replace( /\r\n/g,
'').replace(/^\s+|\s+$/, '').replace(/[^a-z\s]+/gi,
'').replace(/\s+$/, '');

    let parts = q.split(/\s/);
    let terms = [];
    parts.forEach(part => {
        if(stopwords.indexOf(part) === -1) {
            terms.push(part);
        }
    });
    let query = {'$and': []};
    terms.forEach(term => {
       let queryFrag = {title: {'$regex': term,
'$options': 'i'}};
       query['$and'].push(queryFrag);
    });
    return query;
};

app.get('/search', (req, res) => {
   let searchQuery = queryVar(req.query.term);
   Docs.find(searchQuery).then(results => {
       //...
   }).catch(err => {
       //...
   });
});

 app.listen(3000);
```

This is just an oversimplified example. There is always room for improvements.