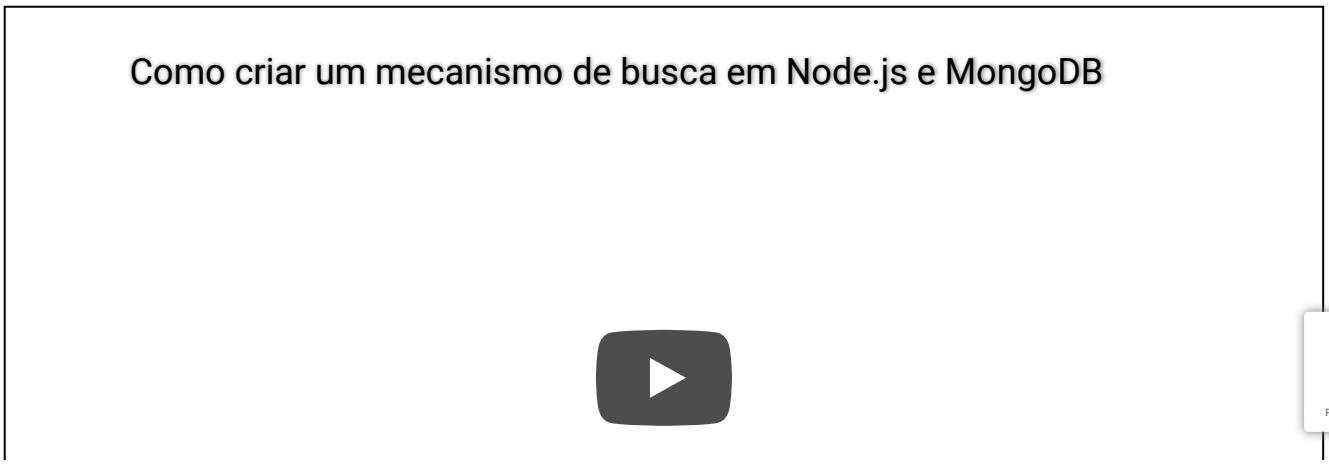
Tutorial de motor de busca com Node.js e MongoDB (driver nativo)

Há uns anos atrás eu escrevi um tutorial de como criar um mecanismo de busca usando esta mesma dupla, Node.js e MongoDB, porém na época com o ORM/ODM Mongoose. O Mongoose é até legal, mas ele come uma fatia importante da performance necessária em mecanismos de busca, algo em torno de 20% e não dá tantos ganhos assim na minha opinião, considerando que usamos os mesmos objetos JSON tanto no backend quanto no banco de dados.

Recentemente fui convidado a palestrar no III Seminário de Tecnologia da Faculdade Alcides Maya e resolvi aproveitar a deixa para atualizar este tutorial e apresentá-lo aos alunos em 30 minutos.

Este é um tutorial intermediário, não pelo conteúdo em si, mas porque não ensino nele o básico de Node.js e MongoDB, parto para o uso direto. Para o básico, recomendo esta playlist no meu canal do Youtube.

E no vídeo abaixo, eu mostro de maneira ainda mais didática o mesmo conteúdo deste tutorial de hoje.



Mecanismos de busca

Todos conhecemos mecanismos de busca, certo? Google, Bing, Yahoo, etc.

Eu tinha essa relação de "mero" usuário de buscadores até 2010, quando resolvi criar meu primeiro motor de busca e de lá para cá tive a oportunidade de escrever buscadores e crawlers para diferentes empresas, alguns que estão no ar até hoje como Busca Acelerada, Buildin e Fisconet.

Aprender a desenvolver mecanismos de busca é muito útil pois você pode usá-los em duas situações diferentes: para agregar uma funcionalidade de busca a uma outra aplicação ou mesmo para criar um negócio digital/startup em cima de um deles.

O que vou mostrar neste tutorial é uma forma simples, porém eficiente, de criar um mecanismo de busca. Teremos uma base com cerca de 20 mil registros sendo pesquisados em menos de 1 segundo em buscas de texto livre (embora eu já tenha testado com 1 milhão de registros com performance semelhante), em uma aplicação que leva 30 minutos para ser construída quando se pega a prática. Não encare o que vou mostrar como algo aplicável a todas situações possíveis, mas algo que apliquei em vários com sucesso e acho que pode agregar ao seu arsenal de desenvolvimento.

E antes de avançar, gostaria que abrisse a sua cabeça não apenas para o que vou mostrar aqui, mas para as possibilidades que o estudo de mecanismos de busca abre no seu conhecimento técnico. Para adentrar nesse tipo de solução você terá de deixar de lado um pouco as tecnologias mais tradicionais, como os bancos relacionais, e se aventurar em soluções mais alternativas e extremamente interessantes como os bancos NoSQL. Ah, e estruturas de dados, se você não gostava disso na faculdade, saiba que usamos bastante estes conceitos em mecanismos de busca...

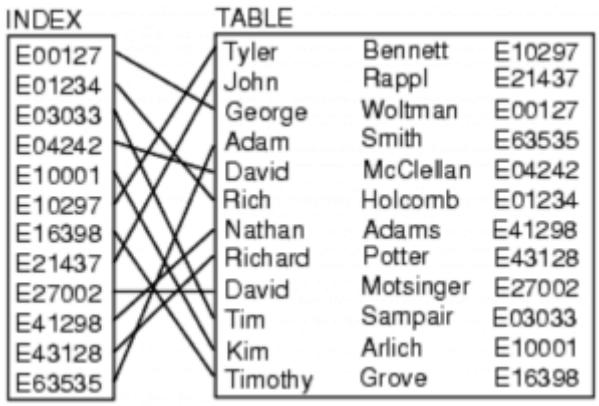
Fonte de Dados

A primeira coisa que vamos precisar para construir o nosso motor de busca é de uma fonte de dados. Essa fonte pode ser alimentada manualmente ou de maneira automática.

Fontes alimentadas manualmente são as mais comuns. Toda empresa minimamente digitalizada hoje possui bancos de dados cheios de informações à espera de serem indexadas por um mecanismo de busca. Quando falo manualmente quero dizer que são usuários que inserem estes dados através de interfaces de sistemas.

Fontes alimentadas de maneira automática são aquelas onde temos um agente robótico chamado crawler ou spider, que percorre os dados da empresa, catalogando-os e jogando no índice do mecanismo de busca. Este tipo de abordagem é mais comum quando estamos indexando dados públicos de terceiros, na Internet, no processo que chamamos de webscrapping.

No fim das contas, em ambas situações, teremos de ter um bom índice de busca que é a base para qualquer motor de busca decente. Talvez você já tenha ouvido falar de índices em bancos relacionais, não é mesmo? Esses índices dos bancos SQL são o que chamamos de índices diretos ou Forward Index, como abaixo.



Forward Index

Note como o índice aponta diretamente para um registro em uma tabela, em uma relação de um para um. Note também, que se quisermos pesquisar pelo nome de um registro, o índice não vai nos ajudar e teremos de percorrer toda tabela (full scan). Ok, você também pode criar outros índices para as outras colunas da tabela, mas ainda assim você depende que o usuário saiba exatamente o que quer pesquisar se quiser usar estes índices diretos dos bancos relacionais.

Segura o problema um pouco, vou voltar nele com outra abordagem.

A Arquitetura

De maneira muito simplificada, todo motor de busca vai ter uma arquitetura minimamente parecida com essa, sendo que a fonte geralmente é uma das duas listadas à esquerda: crawler ou manual.

Arquitetura Buscador

O banco de dados ao centro do diagrama pode ser relacional ou não-relacional, isso é com você. Eu vou mostrar aqui tudo no MongoDB, que é um banco não-relacional, mas isso não é regra. É extremamente comum inclusive o banco central da empresa ser relacional (SQL) e as aplicações periféricas, como o buscador, usar um banco não-relacional apenas com o índice, representado por aquela prancheta ali no diagrama. Isso se chama Persistência Poliglota, onde usamos diferentes linguagens de persistência de dados para compor uma solução ideal.

Este índice nós vamos construir com MongoDB, mas poderia ser construído com outras ferramentas como Redis, Lucene, Sol3r e Elasticsearch, só para citar algumas.

E por fim, a nossa aplicação na direita é apenas um front-end simples com pouco código de back-end, logo pode ser desenvolvida usando a sua tecnologia favorita. Aqui vou usar Node.js com EJS, mas poderia ser feita usando ReactJS por exemplo.

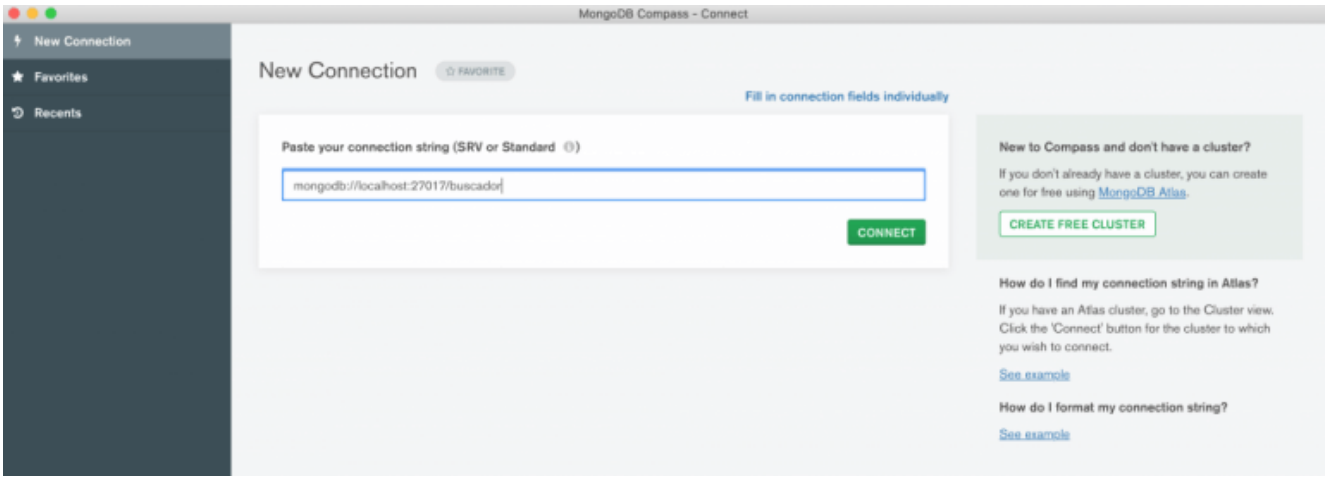
Preparando o banco de dados

Baixe e instale o MongoDB na sua máquina (Community Server) ou suba um cluster gratuito no Atlas como mostro neste tutorial. Também recomendo que tenha o MongoDB Compass, que é a ferramenta visual para fazer a gestão do seu banco.

Suba uma instância de MongoDB, por exemplo, com o comando abaixo.

```
mongod --dbpath pasta-do-projeto/data
```

E conecte nela usando o MongoDB Compass ou linha de comando, se preferir.



MongoDB Compass

Vamos usar uma massa de dados de exemplo aqui, chamada mflix. Ela é um dos datasets de exemplo que o time do Atlas deixa disponível para testes de MongoDB e nada mais é do que uma base estilo Netflix, com dados sobre filmes.

Se você estiver usando o Atlas, pela própria ferramenta web deles você consegue carregar estes datasets de exemplo. Agora se estiver em uma instância local, você pode baixar o zip desse projeto que inclui o arquivo mflix.json deixando seu email no formulário ao final deste tutorial ou então pegando apenas o trecho de exemplo abaixo.

```
[{
  "_id": {
    "$oid": "573a1390f29313caabcd4135"
  },
  "plot": "Three men hammer on an anvil and pass a bottle of beer around.",
  "genres": [
    "Short"
  ],
  "runtime": 1,
  "cast": [
    "Charles Kayser",
    "John Ott"
  ],
  "num_mflix_comments": 1,
  "title": "Blacksmith Scene",
  "fullplot": "A stationary camera looks at a large anvil with a blacksmith behind it
  "countries": [
    "USA"
```

```
  ],
  "released": {
    "$date": {
      "$numberLong": "-2418768000000"
    }
  },
  "directors": [
    "William K.L. Dickson"
  ],
  "rated": "UNRATED",
  "awards": {
    "wins": 1,
    "nominations": 0,
    "text": "1 win."
  },
  "lastupdated": "2015-08-26 00:03:50.133000000",
  "year": 1893,
  "imdb": {
    "rating": 6.2,
    "votes": 1189,
    "id": 5
  },
  "type": "movie",
  "tomatoes": {
    "viewer": {
      "rating": 3,
      "numReviews": 184,
      "meter": 32
    },
    "lastUpdated": {
      "$date": "2015-06-28T18:34:09Z"
    }
  }
},{
  "_id": {
    "$oid": "573a1390f29313caabcd42e8"
  },
  "plot": "A group of bandits stage a brazen train hold-up, only to find a determined
  "genres": [
    "Short",
    "Western"
  ],
  "runtime": 11,
  "cast": [
    "A.C. Abadie",
    "Gilbert M. 'Broncho Billy' Anderson",
    "George Barnes",
    "Justus D. Barnes"
  ],
  "title": "The Great Train Robbery",
  "fullplot": "Among the earliest existing films in American cinema - notable as the f
  "released": {
    "$date": {
      "$numberLong": "-2085523200000"
    }
  },
  "directors": [
    "Edwin S. Porter"
  ],
  "rated": "TV-G",
```

```
  "awards": {
    "wins": 1,
    "nominations": 0,
    "text": "1 win."
  },
  "lastupdated": "2015-08-13 00:27:59.177000000",
  "year": 1903,
  "imdb": {
    "rating": 7.4,
    "votes": 9847,
    "id": 439
  },
  "countries": [
    "USA"
  ],
  "type": "movie",
  "tomatoes": {
    "viewer": {
      "rating": 3.7,
      "numReviews": 2559,
      "meter": 75
    },
    "lastUpdated": {
      "$date": "2015-08-08T19:16:10Z"
    }
  }
},{
  "_id": {
    "$oid": "573a1390f29313caabcd4323"
  },
  "plot": "A young boy, opressed by his mother, goes on an outing in the country with
  "genres": [
    "Short",
    "Drama",
    "Fantasy"
  ],
  "runtime": 14,
  "rated": "UNRATED",
  "cast": [
    "Martin Fuller",
    "Mrs. William Bechtel",
    "Walter Edwin",
    "Ethel Jewett"
  ],
  "num_mflix_comments": 2,
  "title": "The Land Beyond the Sunset",
  "fullplot": "Thanks to the Fresh Air Fund, a slum child escapes his drunken mother f
  "released": {
    "$date": {
      "$numberLong": "-1804377600000"
    }
  },
  "directors": [
    "Harold M. Shaw"
  ],
  "awards": {
    "wins": 1,
    "nominations": 0,
    "text": "1 win."
  },
```

```
  "lastupdated": "2015-08-29 00:27:45.437000000",
  "year": 1912,
  "imdb": {
    "rating": 7.1,
    "votes": 448,
    "id": 488
  },
  "countries": [
    "USA"
  ],
  "type": "movie",
  "tomatoes": {
    "viewer": {
      "rating": 3.7,
      "numReviews": 53,
      "meter": 67
    },
    "lastUpdated": {
      "$date": "2015-04-27T19:06:35Z"
    }
  }
}]
```

Importe o JSON completo (minha recomendação, são 22 mil filmes) usando recurso do menu Collection > Import do MongoDB Compass ou via linha de comando. Caso opte por apenas usar a amostra acima (4 filmes), basta inserir via terminal com o comando insertMany que aceita o array inteiro.

Independente disso, antes de avançar, você precisa ter esta coleção movies no seu banco de dados MongoDB.

Criando o Índice Direto

Vamos fazer um motor de busca baseado em texto livre, ou seja, o usuário vai ter uma caixa de busca e vai poder escrever o que quiser nela. O nome de um ator, um pedaço do título de um filme, o diretor...ou uma combinação de tudo isso. Ele decide.

Para darmos esta liberdade, vamos ter de criar um índice flexível e baseado em texto. Mas antes de criar este índice, precisamos trabalhar na simplificação da informação que vamos armazenar, isso porque as linguagens naturais (as humanas) são muito complexas.

Se dermos uma olhada em um único documento, teremos a estrutura abaixo.

Documento filme

Se quisermos fornecer ao usuário o poder de encontrar filmes por gênero, título, atores, diretores e ano de lançamento, devemos incluir estas informações no nosso índice e, como eu quero que ele use apenas um campo de busca, o índice será um índice de tags, ou seja, cada filme terá associado a si uma série de pequenas palavras oriundas de campos chave do seu documento.

Quando queremos criar um índice de tags, a simplificação deve ocorrer a cada documento inserido na base. Ou seja, inseriu um filme novo, deve-se gerar suas tags de maneira simplificada. Como já estamos com nossos documentos inseridos, vamos fazer esta geração simplificada de maneira retroativa, e para isso criei um script em Node.js.

```
1  //index.js
2  function simplify(text){
3      const separators = /[s,.;:()-'+]/g;
4      const diacritics = /[u0300-u036f]/g;
5      //capitalização e normalização
6      text = text.toUpperCase().normalize("NFD").replace(diacritics, "");
7      //separando e removendo repetidos
8      const arr = text.split(separators).filter((item, pos, self) => self.indexOf(item) ==
9      console.log(arr);
10     //removendo nulls, undefineds e strings vazias
11     return arr.filter(item => (item));
12 }
13
14 function generateTags(movie){
```

```
    let tags = [];
    tags.push(...simplify(movie.title));
    tags.push(movie.year.toString());

    if(movie.cast)
        tags = tags.concat(...movie.cast.map(actor => simplify(actor)));

    if(movie.countries)
        tags = tags.concat(...movie.countries.map(country => simplify(country)));

    if(movie.directors)
        tags = tags.concat(...movie.directors.map(director => simplify(director)));

    if(movie.genres)
        tags = tags.concat(...movie.genres.map(genre => simplify(genre)));
    return tags;
}

function updateMovies(movies){
    movies.map((movie) => {
        console.log(movie.title);
        movie.tags = generateTags(movie);
        global.conn.collection('movies2').insertOne(movie);
    })
}

function findAllMovies(){
    return global.conn.collection('movies').find({}).toArray();
}

const mongoClient = require('mongodb').MongoClient;
mongoClient.connect('mongodb://localhost:27017/netflix')
    .then(conn => {
        global.conn = conn.db('netflix');
        return findAllMovies();
    })
    .then(arr => updateMovies(arr))
    .catch(err => console.log(err));
```

O que faz este script? Vou começar de baixo para cima.

O bloco mais inferior conecta na nossa base, retorna todos os filmes (findAllMovies) e atualiza eles (updateMovies). Certifique-se de ajustar a connection string e o nome do db à sua realidade. Usei bastante promises neste tutorial, então se não sente-se ainda à vontade, dê mais uma estudada neste post.

A função updateMovies itera através de todos os filmes retornados pela findAllMovies com uma function map do JavaScript e, para cada filme, gera as tags do mesmo (generateTags) e insere em outra coleção, mas poderia ser um update na coleção movies, você decide.

A função generateTags, por sua vez, pega as informações dos campos que vamos indexar e passar por uma função de simplificação (simplify), onde colocamos todas as nossas regras

para deixar as tags com um formato comum, neste meu exemplo: caixa alta, sem espaços, sem repetição, sem acentuação ou pontuação.

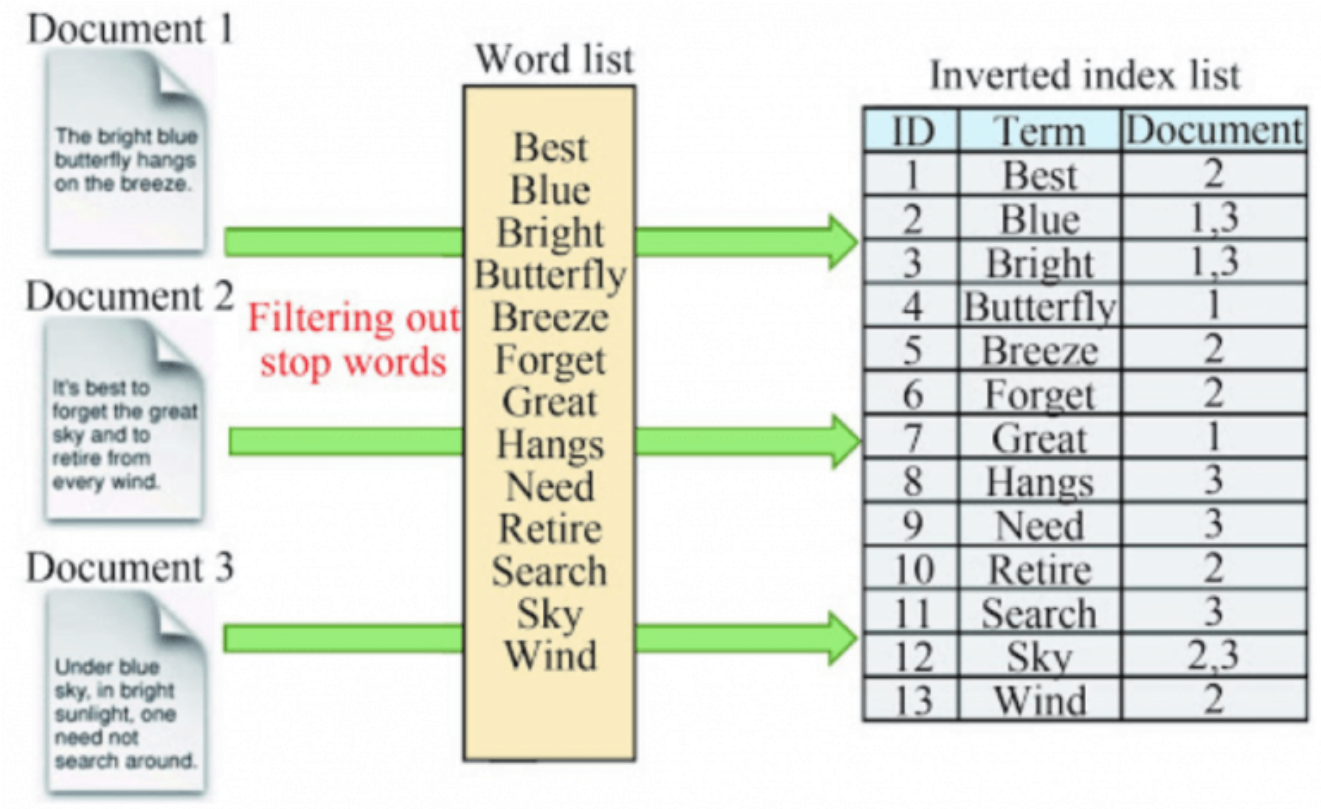
Outras simplificações que recomendo você implementar são a remoção de stopwords (conjunções, artigos, advérbios, etc), o que diminui consideravelmente o tamanho do índice e as chances do usuário digitar algo errado na busca e consequentemente ter uma experiência ruim, além de aumentar a performance, é claro. Também é possível trazer as palavras para o singular e até transformar para sinônimos mais comuns, usando recursos de dicionário de tags.

Note que também usei o spread operator para ajudar na concatenação mais fácil de todas as tags no generateTags.

Com isso tudo que programamos, ao executar esse script no terminal teremos uma segunda coleção, movies2, com um campo tags normalizado/simplificado, ou seja, dado um filme específico, conseguimos chegar nas tags dele, isso é um índice direto ou forward index.

Criando o Índice Invertido

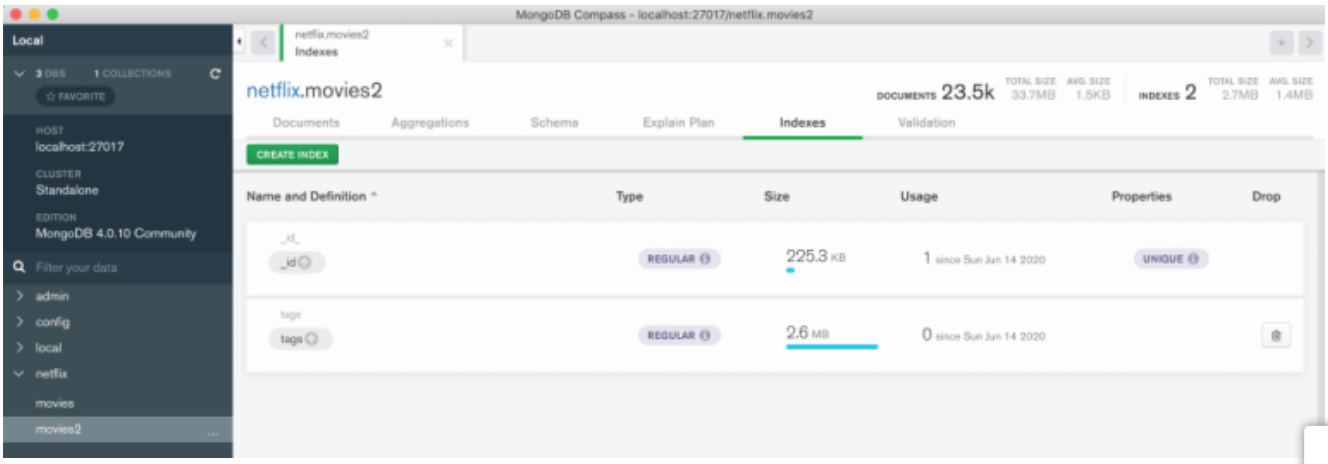
Agora que já temos nosso índice direto de um filme para muitas tags, vamos precisar criar nosso Índice Invertido (Inverted Index) de uma tag para muitos filmes (por isso o invertido no nome). Como no diagrama abaixo.



Indice Invertido

Criar um índice invertido no MongoDB é muito simples, pois ele permite criar índices sobre campos do tipo array, como o nosso campo tags. E essa simplicidade do MongoDB que é uma das coisas pela qual eu recomendo ele para esse tipo de motor de busca.

Você pode criar o índice no campo tags visualmente pelo MongoDB Compass (abaixo) indo na coleção, aba Indexes, ou via terminal, com o comando createIndex.



Índices no Compass

Ao criar esse índice, toda vez que fizermos uma consulta por tags, ele irá buscar nesse índice invertido e, mais do que isso, o MongoDB nos permite usar operadores de conjuntos em consultas sobre arrays, como $in e $all. Experimente fazer a consulta abaixo no compass.

```
{tags: {$in: ["WESTERN","WATER"]}}
```

Ele vai trazer todos os filmes do gênero western (faroeste) ou que contenham water no título. Experimente mudar o operador $in para $all e veja a diferença. É esta segunda opção que geralmente usamos, ou então modelos mais rebuscados com peso de palavras por ocorrência ou precedência, por exemplo, mas estas são técnicas mais rebuscadas.

Criando a aplicação web

Agora é hora de criar a nossa aplicação que vai permitir que os usuários façam a busca pelos filmes. Recomendo usar o express-generator, que com apenas um comando já permite que a gente crie uma aplicação web completamente funcional com um olá mundo. A lista de comandos abaixo, em um terminal como administrador, resolvem o problema de criar esta aplicação do zero.

```
npm install -g express-generator
express -e buscador
cd buscador
npm install
```

```
npm start
```

Isso vai executar uma aplicação de olá mundo em localhost:3000, que pode ser visualizada no navegador.

Express funcionando

Agora, vamos dar uma estilizada nessa página usando Bootstrap, uma das minhas libs web favoritas. A começar abrindo o arquivo views/index.ejs e alterando o topo do arquivo.

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=n
    <title><%= title %></title>
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css
  </head>
```

Aqui, além de algumas meta-tags exigidas pelo Bootstrap, carreguei o CSS do mesmo.

Agora, vamos alterar o rodapé do views/index.ejs, como abaixo.

```
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" int
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js" i
  </body>
</html>
```

Essas tags carregam as dependências do Bootstrap(jQuery e Popper) e depois o dito-cujo.

Com essas referências e configurações no topo e no rodapé, estamos prontos para usar Bootstrap no corpo da nossa aplicação, a começar, criando uma caixa de busca.

```
<div class="container">
      <h1><%= title %></h1>
      <p>Welcome to <%= title %></p>
      <form action="" method="GET">
        <div class="btn-group" role="group">
          <input type="text" class="form-control" name="q" value='<%= query %>' />
          <input type="submit" value="Pesquisar" class="btn btn-primary" />
        </div>
      </form>
```

Isso deve deixar a sua aplicação com essa aparência, ligeiramente melhor do que estava antes, além de deixar um FORM HTML preparado para fazer pesquisas. Experimente digitar algo e dar ENTER para ver o que acontece com a URL do seu navegador.

Caixa de busca

Para fazer o nosso motor de busca funcionar, precisamos programar o backend dele, em JavaScript. Para isso, abra o arquivo routes/index.js, que contém o código que é executado quando acessamos esta aplicação no navegador. Substitua todo o código do arquivo por este abaixo.

```
var express = require('express');
var router = express.Router();

function simplify(text){
    const separators = /[s,.;:()-'+]/g;
    const diacritics = /[u0300-u036f]/g;
    //capitalização e normalização
    text = text.toUpperCase().normalize("NFD").replace(diacritics, "");
    //separando e removendo repetidos
    const arr = text.split(separators).filter((item, pos, self) => self.indexOf(item) ==
    console.log(arr);
    //removendo nulls, undefineds e strings vazias
    return arr.filter(item => (item));
}

/* GET home page. */
router.get('/', function(req, res, next) {
  if(!req.query.q)
    return res.render('index', { title: 'Motor de Busca', movies: [], query: '' });
  else {
    const query = simplify(req.query.q);
    const mongoClient = require("mongodb").MongoClient;
    mongoClient.connect("mongodb://localhost:27017")
               .then(conn => conn.db("netflix"))
               .then(db => db.collection("movies2").find({tags: {$all: query }}))
               .then(cursor => cursor.toArray())
```

```
27                .then(movies => {
28                  return res.render('index', {title: 'Motor de Busca', movies, query: re
29                })
30    }
31  });
32
33  module.exports = router;
```

Repeti aqui a função simplify, que usamos no nosso script de geração de tags, lembra? O ideal é modularizar esta função e usar em ambos locais, deixo isto para você fazer, ok?

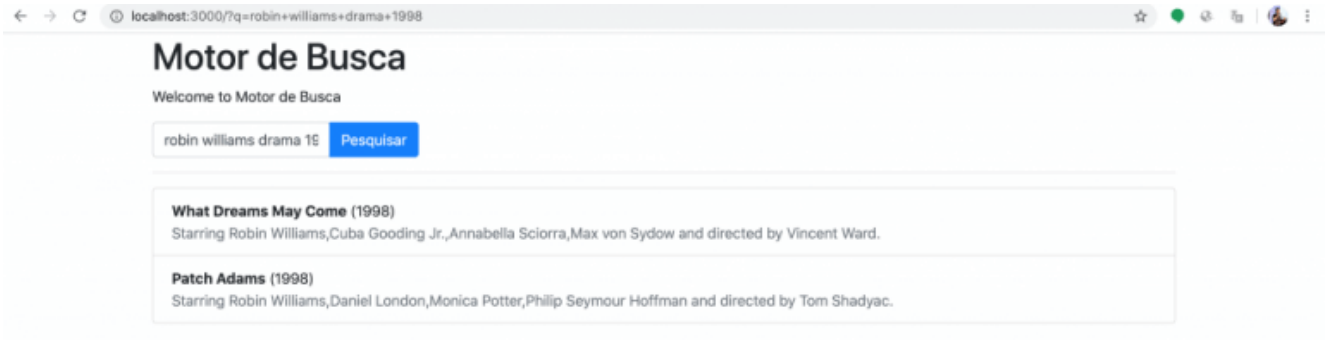
Além desse primeiro bloco, temos a rota GET da nossa aplicação. Aqui, temos um teste para ver se veio uma informação de pesquisa na URL do navegador (chamamos esses parâmetros de querystring) e, com base nesta informação, nos conectamos no MongoDB e fazemos a consulta por todos os documentos que contenham as tags pesquisadas.

Note que eu passo a pesquisa do usuário pela mesma função de simplificação que usamos nas tags. É por isso que fizemos essa simplificação, para garantir que a base das tags e das pesquisas seja a mesma, diminuindo a chance de erros, algo complicado de lidar com bases de dados puras, sem um tratamento para índice de motor de busca.

Com esse backend pronto, devemos voltar ao nosso frontend para ajustar ele a fim de exibir os resultados da pesquisa, como abaixo. Coloque esse código após o FORM HTML.

```
1   <hr />
2         <ul class="list-group">
3         <% for(let i=0; i < movies.length; i++) { %>
4           <li class="list-group-item">
5             <strong><%= movies[i].title %></strong> (<%= movies[i].year %>)
6             <div class="text-muted">Starring <%= movies[i].cast %> and directed by <%= mo
7           </li>
8         <% } %>
9         </ul>
10      </div>
```

O que fazemos aqui é usar tags server-side do EJS para criar um algoritmo JavaScript que fica imprimindo itens de lista com os dados dos filmes retornados. O resto é apenas classes CSS e tags HTML. O resultado, é bem agradável.



Resultados de pesquisa

Concluindo e indo além

Bacana o resultado que tivemos, não?

Experimente brincar com a caixa de busca, digitando coisas aleatórias (preferencialmente em Inglês, por causa da base) e veja como ela é flexível, diferente de buscas tradicionais que exigem um resultado mais literal em relação aos termos pesquisados.

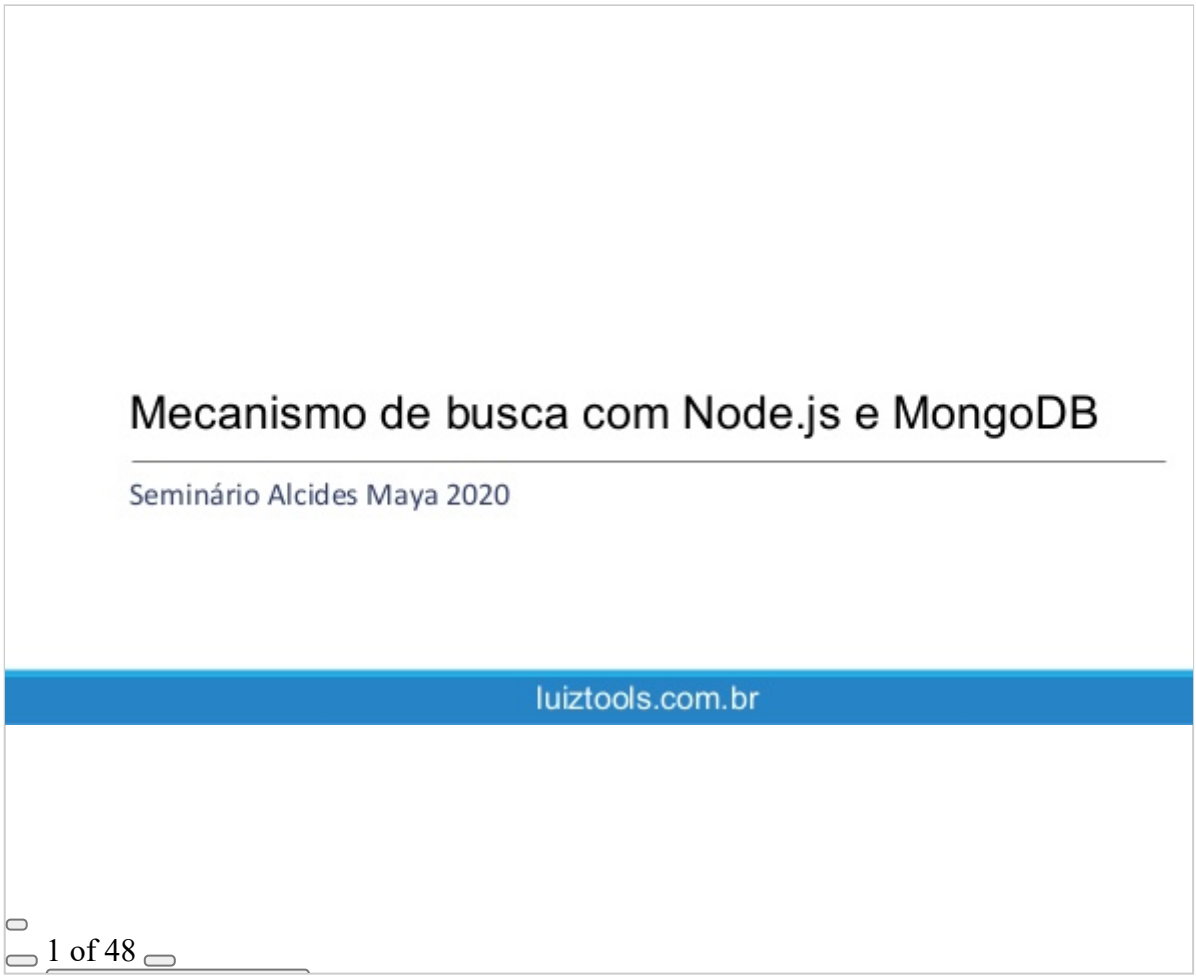
Eu usei somente MongoDB como persistência, mas você poderia utilizá-lo apenas na camada de índice invertido, retornando os ids dos registros em um banco relacional por exemplo. Mesmo com essa arquitetura mista e uma parada obrigatória no índice antes de ir no banco, a aplicação fica muito performática e muito mais dinâmica do que ficar apenas indo no SQL.

Os conceitos que apresentei aqui são universais, aplique-os à vontade e use das referências abaixo para buscar mais conhecimento e mais exemplos.

- Mais teoria de como criar um mecanismo de busca
- Exemplo com ASP.NET Core
- Outro exemplo em Node.js, com Mongoose
- Exemplo com PHP
- Como fazer um webcrawler

Logo abaixo você confere os slides da apresentação que fiz e que originou este post!

Um abraço e sucesso!



Mecanismo de busca em Node.js e MongoDB from **Luiz Duarte**

Considere aprender mais sobre Node.js e MongoDB com o meu curso online, clicando no banner abaixo.

PUBLICADO POR

Luiz Duarte

Pós-graduado em computação, professor, empreendedor, autor, Agile Coach e programador nas horas vagas. Ver todos os posts de Luiz Duarte →

27/06/2020 Luiz Duarte Node.js, Web mongodb, nodejs