04

Busca por proximidade

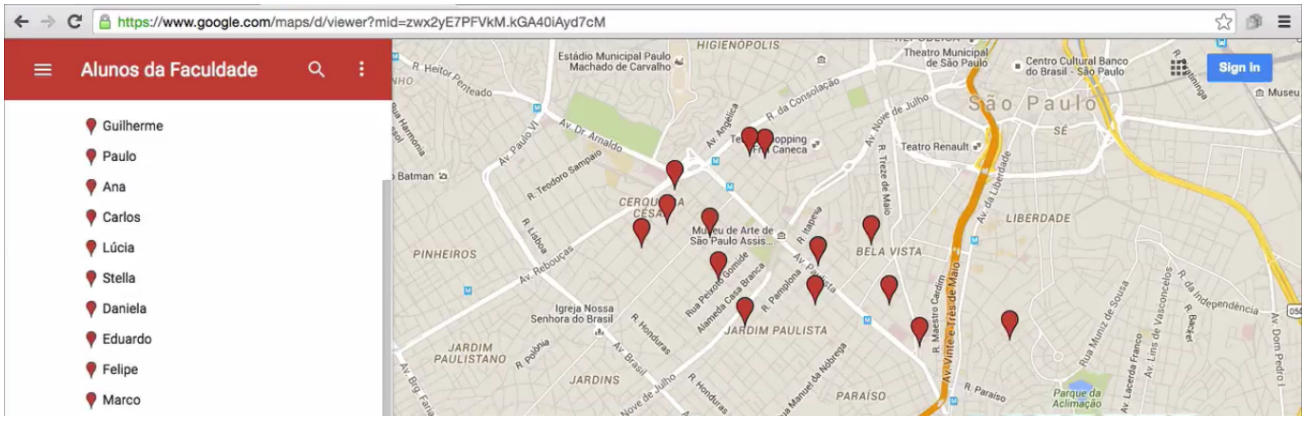
A importância de posições geográficas

Usamos o MongoDB para armazenar informações sobre diversos alunos. As *query* que estamos utilizando no dia a dia são relativamente simples. Vamos, agora, aprender um outro tipo de *query*.

Vamos imaginar que temos em nossa faculdade cerca de 2000 alunos e que todos os dias eles se deslocam até o local de estudo. Muitos acabam indo sozinhos de carro, metrô, mas vivem relativamente perto. Gostaríamos de incentivar alunos que vivem ou trabalham em regiões próximas a se encontrem para irem juntos a faculdade, economizando assim gasolina e preservando o meio ambiente. Vamos trabalhar com a ideia de proximidade.

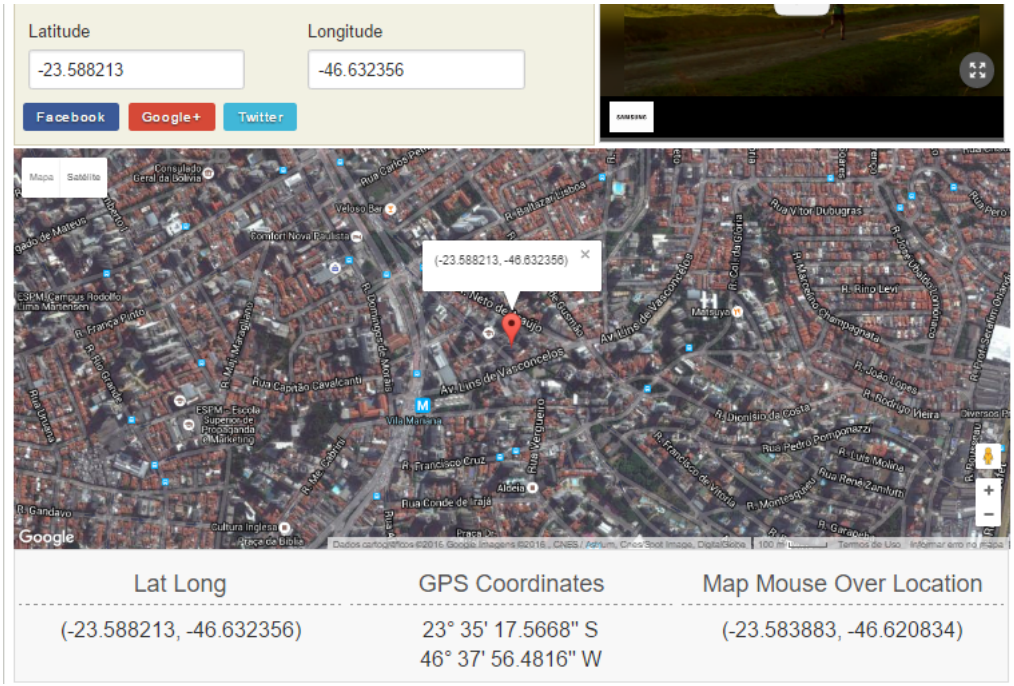
Quando escolhemos um determinado ponto no *GoogleMaps* e elegemos a opção dele nos mostrar estabelecimentos próximos, através do "proximidades" temos, portanto, uma busca por proximidade no mapa.

O que desejamos fazer é uma *query* por distância de pontos no mapa. O MongoDB suporta esse tipo de *query*. Já fizemos um mapa com alguns alunos da faculdade, observe:



O que queremos fazer com um dos pontos do mapa é mostrar quais são os três alunos/pontos mais próximos. Montaremos, dessa maneira, por proximidade o esquema de caronas dos alunos. Poderíamos pegar cada ponto do mapa por seu endereço, mas esse tipo de referência de localização é difícil, pois pode ser alterada, portanto, é pouco confiável.

Ao em vez de armazenar um endereço como "Rua Vergueiro, 3185" podemos armazenar as coordenadas do mapa que é composto por longitude e latitude. Existem APIs e sites que passam um endereço em coordenadas de latitude e longitude, por exemplo, o link (http://www.latlong.net/)http://www.latlong.net/ (http://www.latlong.net/) .



Não importa que a terra gire ou que um endereço mude. As coordenadas de latitude e longitude permanecem no mesmo local. Na sequência pegaremos os três alunos mais próximos de um outro aluno de referência.

Armazenando localizações baseada em uma esfera e a terra

Vamos inserir uma posição de um aluno no mapa, por exemplo, o "Felipe". Podemos observar informações sobre "Felipe" digitando

```
db.alunos.find({nome : "Felipe"}).pretty()
```

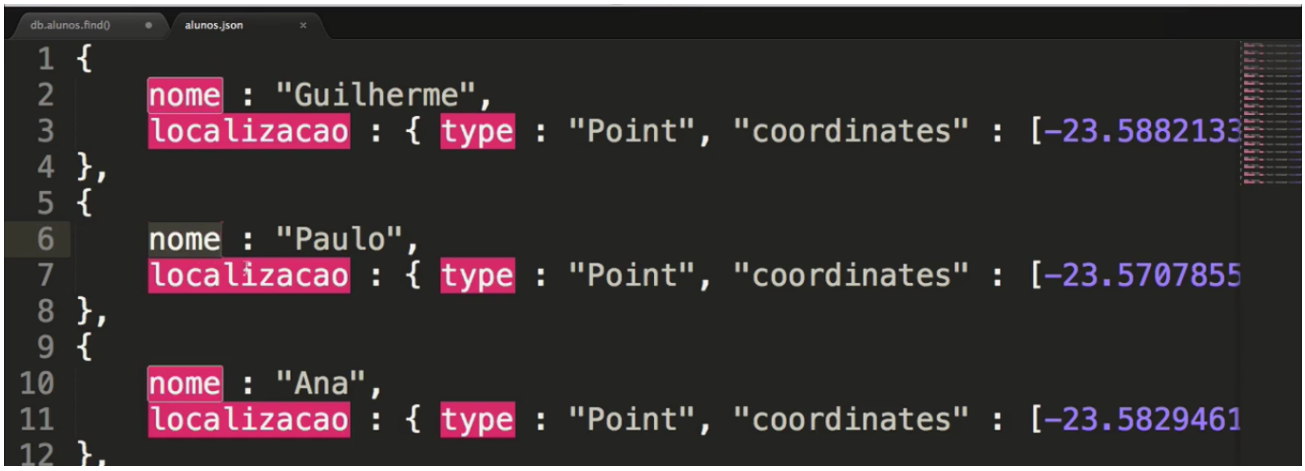
Podemos inserir em "Felipe" sua localização, suas coordenadas geográficas, seu endereço e etc. Para introduzir novas informações nesse aluno utilizamos o `db.alunos.update` onde preenchemos duas *query* que dizem respeito ao `id` do "Felipe" e o que vai ser feito, isto é, "setamos" o campo localização onde colocaremos o endereço, a cidade e as coordenadas. O MongoDB, entretanto, não consegue transformar o endereço que passamos em latitude e longitude, quem faz isso é um API de terceiros. É obrigatório acrescentar as coordenadas, mas podemos adicionar outras informações adicionais, como, a cidade. Atenção! Se quisermos buscar algum aluno utilizando as suas coordenadas temos que seguir um padrão, ele é em inglês. Então, ao em vez de "coordanadas" usaremos "*coordinates*". Além disso, temos que falar qual o tipo de coordenada, nesse caso, é um mero ponto (`type : point`).

```
db.alunos.update(
{ "_id" : ObjectId("56cb0139b6d75ec12f75d3b6") },
{
    $set : {
    localizacao : {
        endereco : "Rua Vergueiro, 3185",
        cidade : "São Paulo",
        coordinates : [-23.588213, -46.632356],
        type : "Point"
        }
    }
}
)
```

Agora, temos armazenadas as informações de endereço, coordenadas e cidade. Lembrando que uma localização geográfica no MongoDB necessita de dois valores, o `coordinates`, que é latitude e longitude e o `type`, que é um ponto. Essas coordenadas referem-se a um ponto na esfera de nosso planeta. Poderíamos ter outras informações, como país e muito mais. Seriam informações adicionais, opcionais.

Utilizaremos algum programa para o qual, através de um API de terceiros, passaremos nome, endereço e cidade e ele nos devolverá as coordenadas.

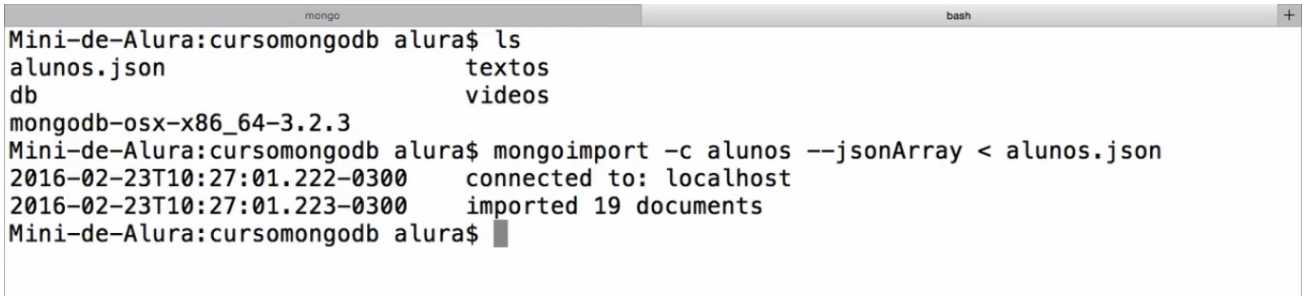
Adicionamos as coordenadas do aluno "Felipe", mas seus colegas estão sem nenhuma localização. Vamos importar uma tabela de dados, de objetos, que já está pronta e que encontra-se no formato *json*, versão estendida. Nesse arquivo temos apenas o nome, as coordenadas e o `type point` e a localização dos alunos:



Vamos importar esse arquivo *json* para a linha de comando. Primeiro, copiamos o arquivo para o diretório **MongoDB* e escreveremos para importar (`import`) da coleção (`-c`) de alunos e importar também uma `--jsonArray`. Vamos redirecionar o arquivo para `< alunos.json`. Teremos o seguinte:

```
mongoimport -c alunos --jsonArray < alunos.json
```

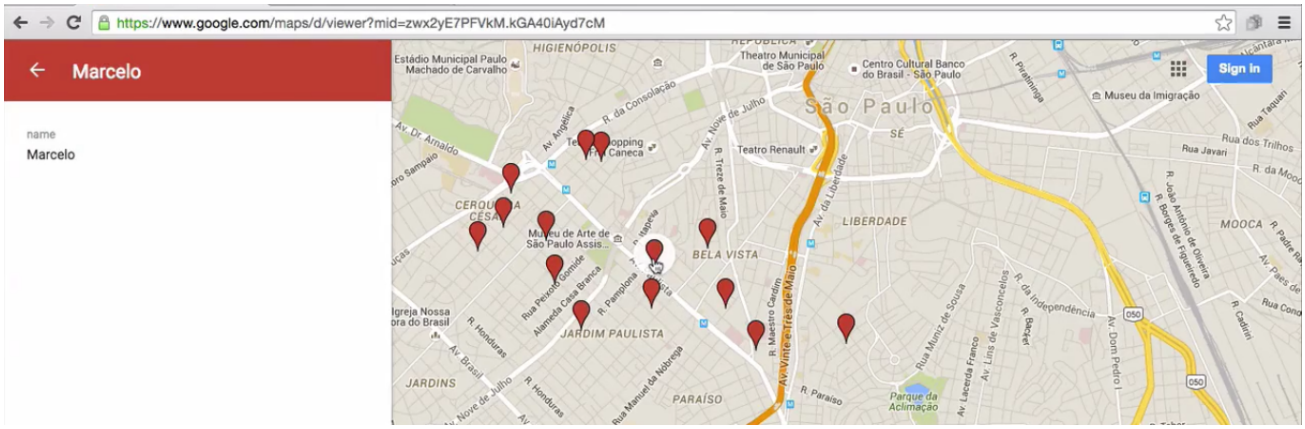
Executando esse comando ele importa 19 documentos.



Para conferir se deu certo basta usar o `db.alunos.find()`. Nesse primeiro passo o que foi mostrado é como armazenar um ponto em nosso Mongodb que represente um ponto na esfera do planeta. Nós criamos o nome ""localização", mas ele pode ser chamado de qualquer outra coisa. O próximo passo é buscar as pessoas próximas aos pontos.

Pesquisando pontos por proximidade e criando um índice baseado em uma esfera

Importamos diversos alunos para o banco do MongoDB através do `import` e, agora que temos o mapa dos alunos:



Vamos pegar o "Marcelo" e verificarmos quem são os três alunos mais próximos dele. Vamos pedir ao **MongoDB** para fazer uma pesquisa de proximidade geográfica, ou seja, agregar os dados mais próximos e nos devolver o resultado. Vamos utilizar o `aggregate`, para agregar um conjunto de dados. Passaremos um dicionário que deve ter alguns parâmetros, o primeiro é o tipo de busca que queremos fazer, como é uma procura por proximidade usaremos o `$geoNear`. O segundo parâmetro é o `near` que indica que queremos aquilo que esteja próximo a uma coordenada específica. Por fim, passaremos também as `coordinates` (longitude e latitude) e o `type` que no caso é `"Point"`. Para conferir as coordenadas de alguém podemos fazer a consulta no material já preparado, o "alunos.json".

Temos que passar ao **MongoDB** que ele deve realizar essa busca procurando o campo localização. Então, temos que criar um índice de busca, o `db.alunos.createIndex()` e nele passaremos o campo localização que possui uma estrutura de "ponto", com latitude e longitude. Na verdade, a chave localização deve ser indexada para uma busca em uma esfera `2d`, pois contam apenas duas dimensões. Teremos o seguinte em nosso Editor:

```
db.alunos.aggregate([
{
    $geoNear : {
        near : {
            coordinates: [-23.5640265, -46.6527128],
            type : "Point"
        }

    }
}
])

db.alunos.createIndex({
    localizacao : "2dsphere"
})
```

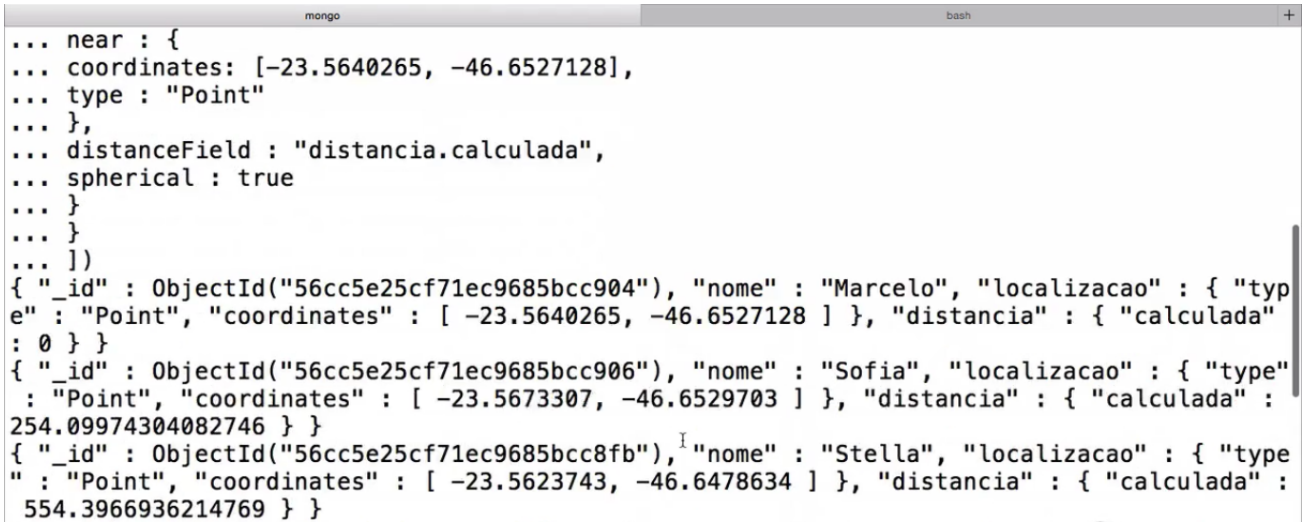
Executaremos o `db.alunos.createIndex` no Terminal e ele irá criar um índice. Agora, precisamos mostrar como calcular a distância entre esses dois pontos. Teremos que falar ao `$geoNear` que a forma é `spherical : true`, ou seja, que a comparação não deve ser entre as distâncias de uma linha, e sim, de uma esfera. Além disso, temos que falar o que deve ser feito com a distância, então, temos que criar o campo `distanceField : "distance.calculada"`. Teremos o seguinte:

```
db.alunos.aggregate([
{
    $geoNear : {
        near : {
            coordinates: [-23.5640265, -46.6527128],
```

```
            type : "Point"
        },
        distanceField : "distancia.calculada",
        spherical : true
    }
}
])

db.alunos.createIndex({
    localizacao : "2dsphere"
})
```

Rodando isso no Terminal ele nos traz todos os alunos:

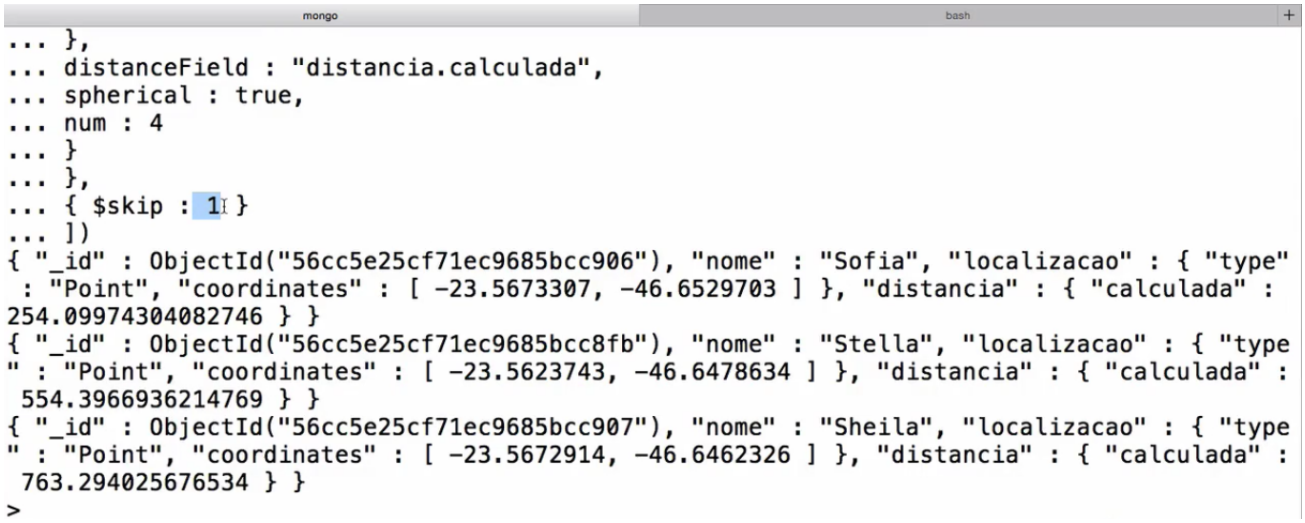


Repare que o primeiro ponto mostrado é aquele que está mais próximo e é o próprio Marcelo. O segundo mais próximo é "Sofia" e assim por diante. Foi ordenado do mais próximo para o mais distante e isso é medido através da distância entre dois pontos. O primeiro ponto é a coordenada que estamos passando e o segundo é o índice que criamos.

Podemos pedir para que apenas 4 pontos sejam trazidos através do `num : 3`. Agora que sabemos como buscar pontos vamos aprender a ignorar o primeiro ponto que é o próprio "Marcelo". Vamos adicionar ao `db.alunos.aggregate` o `num : 4` e para pular um, `skip : 1`.

```
db.alunos.aggregate([
{
    $geoNear : {
        near : {
            coordinates: [-23.5640265, -46.6527128],
            type : "Point"
        },
        distanceField : "distancia.calculada",
        spherical : true,
        num : 4
    }
},
{ $skip :1 }
])
```

E, agora, ele terá ignorado o próprio ponto "Marcelo" e mostrado os outros três mais próximos. Por isso, nos é mostrado apenas três alunos. No final é como se ele não mostrasse 4 elementos, mas sim 4 - 1:



Se quisermos pegar outro aluno para testarmos basta passar as coordenadas desse aluno ao campo `coordinates`. É assim que buscamos por proximidade.

Finalizando

Nesse curso vimos como criar um banco de dados com o MongoDB. E esse banco de dados não é exatamente um banco relacional no modelo *SQL*. É uma outra maneira de alterar, inserir e pesquisar dados, conseguimos criar maneiras diferentes de trabalhar os dados da maneira que precisamos e de acordo com o `update`, `insert`, `mov` e com os demais operadores do MongoDB. E sempre que for necessário consultar outros operadores pode-se observar a documentação. Vimos um exemplo de utilidade diferente do MongoDB como a busca por proximidade que pode ser utilizada para resolver problemas em um mapa.

O MongoDB, entretanto, não se limita a busca de dados em uma esfera, ele armazena diversos tipos de informação, validar dados, alterar completamente um documento e etc.