

2DSphere Geolocation with Zips

This example application demonstrates the 2 dimensional geolocation indexed search available within MongoDB.

Highlights

1. A 2dsphere index was added to the MongoDB collection index.

```
Zip.collection.indexes.create_one({:loc=>Mongo::Index::GEO2DSPHERE})
```

2. The index can be removed with the following command

```
Zip.collection.indexes.drop_one("loc_2dsphere")
```

3. These two commands have been integrated with rake. Don't worry that there is no ActiveRecord involved – the flow still works and feels natural.

```
$ rake db:migrate
== 20151115173006 AddIndexToZips: migrating =====
zips_development.createIndexes | STARTED | {"createIndexes"=>"zips",
"indexes"=>[{:key=>{:loc=>"2dsphere"}, :name=>"loc_2dsphere"}]}
zips_development.createIndexes | SUCCEEDED | 0.00048591000000000004s
== 20151115173006 AddIndexToZips: migrated (0.0077s) =====

$ rake db:rollback
== 20151115173006 AddIndexToZips: reverting =====
zips_development.dropIndexes | STARTED | {"dropIndexes"=>"zips", "index"=>"loc_2dsphere"}
zips_development.dropIndexes | SUCCEEDED | 0.000458864s
== 20151115173006 AddIndexToZips: reverted (0.0070s) =====
```

4. A new find method was added to the Zip model class which used the \$near command to locate other Zip instances within a min/max number of miles. Miles had to be converted to meters.

```
#convert miles to meters
miles_to_meters=1609.34
min_meters=min_miles.to_i*miles_to_meters
max_meters=max_miles.to_i*miles_to_meters

#execute a 2dsphere location find
near_zips=[]
self.class.collection.find(
  :loc=>{: $near=>{
    :$geometry=>{:type=>"Point",:coordinates=>[@longitude,@latitude]},
    :$minDistance=>min_meters,
    :$maxDistance=>max_meters}}
  ).limit(limit).each do |z|
    near_zips << Zip.new(z)
end
```

5. The controller defined a new @locations hash to store a the hash representation of the zip locations. This collection is read in by the index and show pages. Notice that each pin can tell you the city name associate with it.

```
[{:lat=>37.228657, :lng=>-76.542346, :infowindow=>"YORKTOWN"},
{:lat=>39.707341, :lng=>-77.495609, :infowindow=>"FORT RITCHIE"},
{:lat=>38.558946, :lng=>-75.107762, :infowindow=>"MILLVILLE"},
{:lat=>39.459959, :lng=>-77.958915, :infowindow=>"MARTINSBURG"},
{:lat=>38.520712,
:lng=>-76.781677,
:infowindow=>"HUGHESVILLE",
:picture=>{:url=>"/images/marker32.png", :width=>32, :height=>32}}]
```

6. The collection of hashes is displayed using the following javascript (and supporting files that were part of GMaps4Rails setup)

```
<div style='width: 800px;'>
  <div id="map" style='width: 800px; height: 400px;'></div>
</div>
<script type="text/javascript">
  handler = Gmaps.build('Google');
  handler.buildMap({ provider: {}, internal: {id: 'map'}}), function(){
    markers = handler.addMarkers(<%=raw @locations.to_json %>);
    handler.bounds.extendWith(markers);
    handler.fitMapToBounds();
  });
</script>
```

Test Drive

1. Access the root URI. The zipcodes are sorted in alphabetical order so cities in Alaska show up first.

index page with Alaska

2. Add `city=BALTIMORE` to the URI and show this city. Note the the closest five (5) cities that are within 0 miles of that zip code are displayed.

show page showing closest 5 cities greater than or equal to 0 miles from BALTIMORE

2. Add `max_miles`, `min_miles`, and `limit` to locate the nearest N cities that are M miles away from BALTIMORE. This forms a ring around the city.

show page showing closest 20 cities greater than or equal to 50 miles from BALTIMORE

Assembly

These are the edits performed to the `zips` application use to demonstrate integrating the MongoDB Ruby Driver with Rails.

Add the 2dsphere Index

1. Add a database migration to house the management of our index. Rails will place a timestamp within the name of this file.

```
$ rails g migration AddIndexToZips
  invoke  active_record
  create  db/migrate/20151115173006_add_index_to_zips.rb
```

2. Edit the migration file created in `db/migrate` with the following `up` and `down` commands to add and remove the index. `:loc` is the field used by the `zips` collection to hold the geolocation information.

```
class AddIndexToZips < ActiveRecord::Migration

  # add a 2dsphere index to Zip.loc field
  def up
    Zip.collection.indexes.create_one({:loc => Mongo::Index::GEO2DSPHERE})
  end

  def down
    Zip.collection.indexes.drop_one("loc_2dsphere")
  end
end
```

3. Use rake to migrate the database to add the index.

```
$ rake db:migrate
== 20151115173006 AddIndexToZips: migrating =====
D, [2015-11-15T13:05:19.695296 #70768] DEBUG -- : MONGODB | Adding localhost:27017 to the cluster.
D, [2015-11-15T13:05:19.704984 #70768] DEBUG -- : MONGODB | localhost:27017
  | zips_development.createIndexes | STARTED |
  {"createIndexes"=>"zips", "indexes"=>[{:key=>{:loc=>"2dsphere"}, :name=>"loc_2dsphere"}]}
D, [2015-11-15T13:05:20.018183 #70768] DEBUG -- : MONGODB | localhost:27017
  | zips_development.createIndexes | SUCCEEDED | 0.31258843399999997s
== 20151115173006 AddIndexToZips: migrated (0.3304s) =====
```

Updated Gemfile with Gmaps4Rails Gem

Add the `gmaps4rails` gem to the Gemfile. Documentation for the gem is available at <https://github.com/apneadiving/Google-Maps-for-Rails>. It is highly recommended that you also watch the referenced YouTube Video

```
gem 'gmaps4rails'
```

```
$ bundle
```

```
$ rails s
```

Add Attribute Support for Geolocation Properties in Zip Model Class

1. The original zips application left off support for the `:loc` property. Add `:longitude` and `:latitude` as attributes.

```
attr_accessor :id, :city, :state, :population, :longitude, :latitude
```

2. Add initialization support for the new attributes in the `initialize()` method.

```
def initialize(params={})
  ...
  if params[:loc]
    @longitude=params[:loc][0]
    @latitude=params[:loc][1]
  else
    @longitude=params[:longitude]
    @latitude=params[:latitude]
  end
end
```

3. Add the new attributes to the query projections within the class. There are at least two occurrences.

```
.projection({:_id:true, city:true, state:true, pop:true, loc:true})
```

Add Geolocation Search in Zip Model Class

Add the following instance method to the Zip model class to perform a 2dsphere, geolocation search for zips near its location. Accept a `max_miles`, `min_miles`, and `limit`. The miles have to be converted to meters. The coordinates are passed in as an array with longitude first and latitude second.

```
#return a list of zipcodes within min/max miles
def near(max_miles, min_miles, limit)
  max_miles=max_miles.nil? ? 1000 : max_miles.to_i
  min_miles=min_miles.nil? ? 0 : min_miles.to_i
  limit=limit.nil? ? 5 : limit.to_i
  limit+=1 if min_miles==0

  #convert miles to meters
  miles_to_meters=1609.34
  min_meters=min_miles.to_i*miles_to_meters
```

```

max_meters=max_miles.to_i*miles_to_meters

#execute a 2dsphere location find
near_zips=[]
self.class.collection.find(
  :loc=>{:near=>{
    :$geometry=>{:type=>"Point",:coordinates=>[@longitude,@latitude]},
    :$minDistance=>min_meters,
    :$maxDistance=>max_meters}}
  ).limit(limit).each do |z|
    near_zips << Zip.new(z)
  end
  near_zips
end

```

Add Geolocation Search Support in Zips Controller

1. Add the following helper method to the controller as a private method. It accepts a collection of Zip objects and works with Gmaps3rails to create an array of hashes that is used by googlemaps to display pins in the map. If there is a current @zip – it will be made the center of the map with a special icon. Note that this definition is referencing a PNG file placed in the public/images folder.

```

def zip_markers zips
  #build the marker for the center of the map
  if @zip
    center_marker = Gmaps4rails.build_markers(@zip) do |zip, marker|
      marker.lat zip.latitude
      marker.lng zip.longitude
      marker.infowindow zip.city
      marker.picture(:url=> "/images/marker32.png",
        :width=> 32,
        :height=> 32)
    end
  end

  #build markers for map
  marked_zip=@zip.nil?
  locations = Gmaps4rails.build_markers(zips) do |zip, marker|
    marker.lat zip.latitude
    marker.lng zip.longitude
    marker.infowindow zip.city
    #add special marker for target city
    if @zip && zip.id==@zip.id
      marker.picture center_marker[0][:picture]
      marked_zip=true
    end
  end

  #add target city of left out
  locations << center_marker[0] if !marked_zip
  return locations
end

```

2. Add an image file to the public/images folder for the map definition to reference.

```

$ls public/images
marker24.png
marker32.png

```

marker48.png

Update Actions to Build Geolocation Markers

1. Have the index page define a list of geolocation markers for the zipcodes that are deployed on the page.

```
def index
  ...
  @zips = Zip.paginate(args)
  @locations = zip_markers @zips
end
```

2. Have the individual show page display a map with nearby cities.

```
def show
  near_zips=@zip.near(params[:max_miles], params[:min_miles], params[:limit])
  @locations=zip_markers near_zips
end
```

Add Map Support to Views

1. Add the following script file references to the app/views/layouts/application.html.erb. This part of the GMaps4Rails setup.

```
<head>
  ...
  <script src="//maps.google.com/maps/api/js?v=3.23&sensor=false&client=&key=&libraries=geometry&language=
  <script src="//cdn.rawgit.com/mahnunchik/markerclustererplus/master/dist/markerclusterer.min.js"></script>
  <script src="//cdn.rawgit.com/printercu/google-maps-utility-library-v3-read-only/master/infobox/src/infobox.min.js"></script>
</head>
```

2. Add the following to the bottom of the index page (app/views/zips/index.html.erb) and and show page (app/views/zips/show.html.erb) except This will display the map and place location markers for elements within the @locations collection'.

```
<div style='width: 800px;'>
  <div id="map" style='width: 800px; height: 400px;'></div>
</div>
<script type="text/javascript">
  handler = Gmaps.build('Google');
  handler.buildMap({ provider: {}, internal: {id: 'map'}}, function(){
    markers = handler.addMarkers(<%=raw @locations.to_json %>);
    handler.bounds.extendWith(markers);
    handler.fitMapToBounds();
  });
</script>
```

Heroku Deployment

This deployment assumes that you have already deployed the Zips application and can re-use the same database for both applications.

1. Register your application with Heroku by changing to the directory with a git repository and invoking `heroku apps:create (appname)`.

Note that your application must be in the root directory of the development folder hosting the git repository.

```
$ cd fullstack-course3-module2-geozipt
$ heroku apps:create appname
Creating appname... done, stack is cedar-14
https://appname.herokuapp.com/ | https://git.heroku.com/appname.git
Git remote heroku added
```

This will add an additional remote to your git repository.

```
$ git remote --verbose
heroku https://git.heroku.com/appname.git (fetch)
heroku https://git.heroku.com/appname.git (push)
...
```

2. Add a MONGODB_URI environment variable from the ziptions application deployment. dbhost is both host and port# concatenated together, separated by a ":" (host:port) in this example.

```
$ cd fullstack-course3-module1-ziptions
$ heroku config | grep MONGODB_URI
$ cd fullstack-course3-module2-geozipt
$ heroku config:add MONGODB_URI=mongodb://dbuser:dbpass@dbhost/dbname
```

3. Deploy application

```
$ git push heroku master
```

4. Create the geolocation index. This has been packaged as a ActiveRecord database migration task. So use heroku run rake db:migrate to put in place.

```
$ heroku run rake db:migrate
Running `rake db:migrate` attached to terminal... up, run.4636
...
DEBUG | {"createIndexes"=>"ziptions", "indexes"=>[{:key=>{:loc=>"2dsphere"}, :name=>"loc_2dsphere"}]}}
```

Access Application

1. Access URL

```
http://appname.herokuapp.com/ziptions/21044?min_miles=5&limit=20
```