Part I: Ruby Methods/RSpec

Introduction

The first part of the skill test will be to implement a few ruby methods as defined by rspec unit tests that have been provided for you.

Requirements and Scoring:

1. An Array is offset sorted if the elements are sorted in ascending order but displaced by any number of steps.

   For example, the following array is offset sorted:

   ```
   [4,5,6,1,2,3]
   ```

   Write a method, *Array#offset_sorted?* (stub provided in lib/core_ext/array.rb and autoload configured) that determines if an array is sorted with an offset. Get the corresponding rspec tests passing (spec/lib/core_ext/array_spec.rb). **(/10)**

2. The class SuperFib#process! (lib/utils/super_fib.rb), when initialized with an Array, returns the Fibonacci number for each element in the array.

   For example:

   ```
   SuperFib.new([1,2,3,4,5,6,7,8,9,10]).process!; # [1,1,2,3,5,8,13,21,34,55]
   ```

   The problem is that the current implementation performs very poorly and processing grows exponentially for each additional array element. Optimize the code such that so that it performs by at least 6 orders of magnitude faster than the baseline and the rspec test passes (spec/lib/utils/super_fib_spec.rb) **(/10)**

3. You've got a lot of data that uses zero-based, non-negative IDs and need a method to return the smallest available ID.

   For example:

   ```
   [1,2,3,0,4,5,7].smallest_available_id; # 6
   ```

   Complete the method Array#smallest_available_id (lib/core_ext/array.rb) to return the smallest unused ID from the array and get the corresponding rspec test to pass (spec/lib/core_ext/array.rb) **(/10)**

4. A Super Power Number is a number which is the sum of its own digits, each raised to the power of the number of digits.

   For example, take "153" (3 digits):

   ```
   1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153
   ```

   and "1634" (4 digits):

1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256 = 1634

Write a method, *Integer#super_power_number?* (stub provided in lib/core_ext/integer.rb) that returns true or false depending upon whether the given number is a Super Power Number. **(/10)**

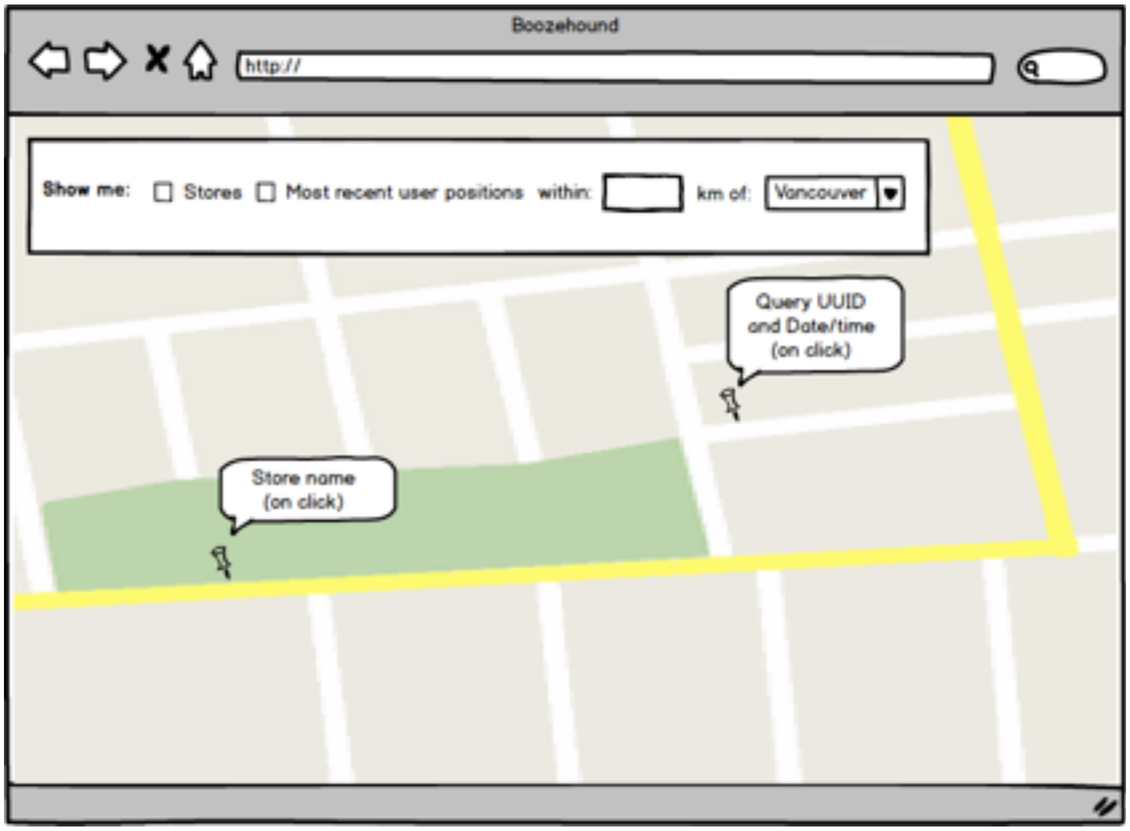
5. Write a domain validator (lib/utils/domain_validator.rb) to ensure that a given string represents a valid domain. The validator should satisfy the corresponding rspec tests (spec/lib/utils/domain_spec.rb) should pass. **(/10)**

Total: 50

Part II: Rails app

Introduction

BoozeHound is a location-based liquor store finder app for the iPhone. The application queries a MySQL database on our server via web service to find stores nearest the user's location. Each time a query is made, the web service stores usage information in our DB, such as the user's device identifier (UUID), latitude/longitude, GPS accuracy, and execution time.



The Task

Your task is to build a web page that displays stores and/or the last known position of each user from the BoozeHound database on a Google Map. The wireframe below illustrates the functional requirements:

Notes:

- Some basic application structure/code has been provided to save setup time. Upon starting the application in your browser, you will see that a Google Map is already being initialized with a placeholder for the selector UI.
- The map initialization code and functions for plotting pins and moving the map can be found in **app/assets/javascripts/map.js.coffee**
- The map view can be found in **app/views/search/index.html.haml**
- We're using the **geokit-rails** gem for the geo location functionality (e.g. querying data within X km of a lat/long coordinate): https://github.com/geokit/geokit-rails
- The Store model code is provided for you which utilizes the geokit-rails gem (**app/models/store.rb).** You will have to implement the functionality for the Query model, however, which will differ slightly (more on this below).

Setup instructions:

1. Run *bundle install*
2. Create/seed the database by running *rake db:setup* (or *rake db:create; rake db:schema:load; rake db:seed* if you have trouble).
3. Start the server by running *rails s.* Upon launching the app you should see a Google Map.

Requirements and Scoring:

1. Implement the model scope \`Query.most_recent\` which should retrieve the **most recent query for each unique UUID value**. (**HINT:** a \`DISTINCT\` call will not suffice, as this will only give you the first position for each UUID. Also note that the \`id\` column cannot be relied upon for providing the latest row for each UUID because of network latency issues when the queries are reported.) **(/10)**
2. Implement the model method \`*Query.most_recent_within_distance_of*\` which should retrieve the **most recent user positions for a given distance and set of coordinates**. You'll want to make use of the geokit-rails gem which is already installed, as well as your scope from #1. **(/5)**
3. Hook up the view form in *search/index.html* to *search_controller.rb.* Call the relevant model methods to retrieve the correct Query and Store data for the given form input. The request should be made via AJAX and response rendered using JSON. **(/15)**
4. Plot the pins on the map using Javascript (most of the map-related code has been provided for you in application/assets/javascripts/map.js.coffee) whenever the form input changes.

The marker tooltips should display on click with the content as indicated in the wireframe above. **(/15)**

5. Implementation design and coding style (DRY, efficient, readable, maintainable, etc.) **(/5)**

Total: 50