Module 1, Summative Assignment: Integrating MongoDB Ruby Driver and Rails

This assignment focuses on your ability to implement MongoDB Ruby Driver commands within the context of a Rails application and scaffold. To do this – you will implement an ActiceModel model class using MongoDB that should nearly immediately work with a generated controller and view. This will not only test your knowledge of the specific MongoDB Ruby Driver commands but will also give you insight into the functionality provided by the Mongoid ORM you will use later in the course and how you can drop down to the raw MongoDB Driver API if the need ever arises in a full application. You will be given much of the Rails portion of the assignment in detailed hints. You must fill in the missing MongoDB Ruby Driver portions based on the lectures and other course materials.

The overall goal of the assignment is to:

- Integrate MongoDB into a Rails application using the MongoDB Ruby Driver
- Manually implement CRUD methods required of a Rails model class to work with Rails scaffold
- Add manual model support for `will_paginate` pagination of data

The functional goal of the assignment is to:

- Implement a web application to access racers

Functional Requirements

1. Add a connection from Rails to the MongoDB server using Mongoid. You will:

- include the necessary gems
- configure server connection and database

2. Manage Racers in the MongoDB `racers` collection and use a a class called Racer to encapsulate access to MongoDB and the `racers` collection.

- get a connection to the MongoDB server and default database
- get the collection for our model type
- ingest data into the collection

3. Use the `Racer` class as a Rails model class to encapsulate the properties tracked for a racer within MongoDB.

- id - primary key within the database
- number - their race bib number
- first_name - given name
- last_name - surname
- gender - "M", "F", or nil
- group - age group running in
- secs - race completion time in secs

This will include the following model commands

- all - find all racers in the database collection
- find - find a specific racer by ID in the database collection
- save - save the current instance
- update - update the properties of the curren instance to the database collection
- destroy - remove the racer from the database collection

4. Create a scaffold for the `Racer` model class to view and modify racer information in the database collection.
5. Add pagination support to the `Racer` index page.

Getting Started

1. Create a new Rails application called `raceday`.
2. Download and extract the starter set of boostrap files for this assignment.

   ```
   |-- Gemfile
   |-- race_results.json
   `-- spec
       |-- start_spec.rb
       |-- mongoid_spec.rb
       |-- connection_spec.rb
       |-- crud._specrb
       |-- model_spec.rb
       |-- scaffold_spec.rb
       `-- paginate_spec.rb
   ```

   - Overwrite your existing Gemfile with the Gemfile from the bootstrap fileset. They should be nearly identical, but this is done to make sure the gems and versions you use in your solution can be processed by the automated Grader when you submit. Any submission should be tested with this version of the file.

     NOTE the Gemfile includes the following added to support testing:

     ```
     group :test do
         gem 'rspec-rails', '~> 3.0'
         gem 'capybara'
     end
     ```

     as well as a new definition for the following items:

     – `tzinfo-data` gem conditionally included on Windows platforms
     – `mongoid` gem added to support getting connections to MongoDB server
     – `will_paginate` added for implementing paging

     ```
     # Windows does not include zoneinfo files, so bundle the tzinfo-data gem
     gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64_mingw, :jruby]
     gem 'mongoid', '~> 5.0.0'
     gem 'will_paginate', '~> 3.0.6'
     ```

   - Add the `spec/*.rb` files provided with the bootstrap fileset to the corresponding `spec/` directory within your `raceday` application. These files contain tests that will help determine whether you have completed the assignment.

3. Run the `bundle` command to make sure all gems are available.

   ```
   $ bundle
   ```

4. Run the rspec test(s) to receive feedback. `rspec` must be run from the root directory of your application. There are several test files provided for this assignment. Many of those files are designed to test your code at specific points as you proceed through the technical requirements of this assignment. As such, many tests will fail if executed after additional technical requirements have been completed. Initially, majority of tests will (obviously) fail until you complete the requirements necessary for them to pass.

   ```
   $ rspec
   ...
   (N) examples, 1 failure, (N) pending
   ```

   To focus test feedback on a specific step of the requirements, add the specific file (path included) with the tests along with "-e rq##" to the rspec command line to only evaluate a specific requirement. Pad all step numbers to two digits.

```
$ rspec spec/connection_spec.rb -e rq01
...
(N) example, 0 failures
```

5. Start your MongoDB `mongod` process.
6. Implement your solution to the technical requirements and use the rspec tests to help verify your completed solution.
7. Submit your Rails app solution for grading.

Technical Requirements

Mongoid Database Connection

In this section you will complete the installation of Mongoid into your application by generating a configuration file and wiring that configuration file into the Rails application so you can get connections to MongoDB. We will only use Mongoid for connections at this point. All commands to MongoDB will be through the MongoDB Ruby Driver.

1. Start with the `raceday` application created in the `Getting Started` section.
2. Generate a Mongoid configuration file and update to reference the same MongoDB server instance and database as in the previous assignment.

   ```
   $ rails g mongoid:config
   ```

   The generated defaults should be correct for what we will use. They may be different from what you used in a previous assignment, but we are purposely going to keep them in a separate database from what was used in the other assignment.

   ```
   $ cat config/mongoid.yml | egrep -v '^$|#'
   development:
     clients:
       default:
         database: raceday_development
         hosts:
           - localhost:27017
     ...
   test:
     clients:
       default:
         database: raceday_test
         hosts:
           - localhost:27017
     ...
   ```

3. Add some Mongoid configuration to `config/application.rb`. This is used by stand-alone programs like “rails console” to be able to load the Mongoid environment with fewer steps. This also configures which ORM your scaffold commands use by default. Adding the mongoid gem had the impact of making Mongoid the default ORM. The lines below show how we can set it back to ether ActiveRecord or Mongoid and how to use the `--orm` flag to identify the mapping on a per-model type basis. However, we will not be generating any ActiveRecord or Mongoid model classes as a part of this assignment. We will only be using Mongoid during this assignment to get connections to MongoDB and we might as well show a complete configuration while we are here.

   ```
   module Raceday
     class Application < Rails::Application
     ...
   ```

```
    #bootstraps mongoid within applications -- like rails console
    Mongoid.load!('./config/mongoid.yml')

    #which default ORM are we using with scaffold
    #add  --orm mongoid, or active_record
    #    to rails generate cmd line to be specific
    #config.generators {|g| g.orm :active_record}
    config.generators {|g| g.orm :mongoid}
  ...
```

4. Start the web server.

```
$ rails s

$ rspec spec/mongoid_spec.rb
```

Obtain Database Connection from Model Class

In this section you create a model class and add some convenience methods to get a connection to the MongoDB server and document collection.

1. Create a model class by hand called `Racer` in the `app/models` directory. This class must have:

   - a class method called `mongo_client` that returns a MongoDB client configured to communicate to the default database specified in the `config/mongoid.yml` file.
   - a class method called `collection` that returns the `racers` MongoDB collection holding the `Racer` documents.

   Hint:

   ```
   $ rails c
   > Mongoid::Clients.default
    => #<Mongo::Client:0x46050120 cluster=localhost:27017>
   ```

   Use the `rails console` to demonstrate your new methods.

   ```
   > Racer.mongo_client.database.name
    => "raceday_development"
   > Racer.collection.name
    => "racers"

   $ rspec spec/connection_spec.rb -e rq01
   ```

2. Use the `rails console` and the `Racer` class and methods added above to ingest data into the collection. The `; nil` is shown below is to keep the default logger level of the rails console from printing the evaluation of the large collection results after each command.

   ```
   $ rails c
   > file_path="./race_results.json"
   > file=File.read(file_path); nil
   > hash=JSON.parse(file); nil
   > racers=Racer.collection
   > racers.insert_many(hash); nil
   ```

   You should end up with 1000 racers in your collection.

   ```
   > Racer.collection.count
    => 1000

   $ rspec spec/connection_spec.rb -e rq02
   ```

CRUD Model Methods

In this section you will apply your knowledge of MongoDB Ruby Driver commands to implement CRUD methods required by the Rails scaffold. We don't need the scaffold yet – we can implement and test quite a lot with the "rails console" and unit tests.

1. Create a class method in the `Racer` class called `all`. This method must:

   - accept an optional prototype, optional sort, optional skip, and optional limit. The default for the prototype is to "match all" – which means you must provide it a document that matches all records. The default for sort must be by number ascending. The default for skip must be 0 and the default for limit must be nil.
   - find all racers that match the given prototype
   - sort them by the given hash criteria
   - skip the specified number of documents
   - limit the number of documents returned if limit is specified
   - return the result

   Hint:

   ```
   def self.all(prototype={...}, sort={...}, skip=0, limit=nil)
     ...
   end
   ```

   Use the Rails console to verify and explore your result. Use the `reload!` command after making code changes. The following command shows there are 1000 records in the database.

   ```
   > reload!
   > pp Racer.all.count; nil
   1000
   ```

   The following command shows that the parameters to the `all` method are optional.

   ```
   > pp Racer.all.first; nil
   {"_id"=>BSON::ObjectId('563daabbe301d0978b000000'),
    "number"=>0,
    "first_name"=>"SHAUN",
    "last_name"=>"JOHNSON",
    "gender"=>"M",
    "group"=>"15 to 19",
    "secs"=>1464}
   ```

   The following command shows we have the power to find matching documents thru a prototype and control the sorting and paging.

   ```
   > pp Racer.all({group:"50 to 59", gender:"F"}, {last_name:-1},0,1).to_a; nil
   [{"_id"=>BSON::ObjectId('563daabbe301d0978b0000a6'),
     "number"=>166,
     "first_name"=>"MONA",
     "last_name"=>"WATSON",
     "gender"=>"F",
     "group"=>"50 to 59",
     "secs"=>2321}]
   ```

   ```
   $ rspec spec/crud_spec.rb -e rq01
   ```

2. Add attributes to the `Racer` class that allow one to set/get each of the following properties:

   - id
   - number
   - first_name
   - last_name
   - gender

  - group
  - secs

  Hint:

  ```
  class Racer
    attr_accessor :id, :number, :first_name, :last_name, :gender, :group, :secs
  ```

  Note that `id` is a special primary key property within ActiveModel and must exist to work correctly with Rails scaffold. We will map that property to the string value of the MongoDB `_id` property. The `_id` properties ingested are in `BSON::ObjectId` form and can be converted to/from string using:

  ```
  @id=doc[:_id].to_s
  :_id=>BSON::ObjectId.from_string(@id))
  ```

  Note that the BSON::ObjectId is a globally unique value and has a specific format and length. The `from_string` method will throw an exception if passed a string with an incorrect format/length. If we wanted to use an arbitrary, unique value – we would not use the BSON::ObjectId type for our MongoDB primary key.

  ```
  $ rspec spec/crud_spec.rb -e rq02
  ```

3. Add an initializer that can set the properties of the class using the keys from a `racers` document. It must:

  - accept a hash of properties
  - assign instance attributes to the values from the hash
  - for the `id` property, this method must test whether the hash is coming from a web page `[:id]` or from a MongoDB query `[:_id]` and assign the value to whichever is non-nil.

  Hint:

  ```
  def initialize(params={})
    @id=params[:_id].nil? ? params[:id] : params[:_id].to_s
    @number=params[:number].to_i
    @first_name=params[:first_name]
    @last_name=params[:last_name]
    @gender=params[:gender]
    @group=params[:group]
    @secs=params[:secs].to_i
  end
  ```

  ```
  $ rspec spec/crud_spec.rb -e rq03
  ```

4. Create a class method in the `Racer` class called `find`. This method must:

  - accept a single `id` parameter
  - find the specific document with that `_id`
  - return the `racer` document represented by that `id`

  Hint:

  ```
  def self.find id
    result=collection...
    return result.nil? ? nil : Racer.new(result)
  end
  ```

  Use the `rails console` to test and explore your changes. Note that return type is now an instance of a `Racer` class or nil if not found. The last example has a valid BSON string but an unknown value.

  ```
  > reload!
  > Racer.all.projection(first_name:1, last_name:1).first
   => {"_id"=>BSON::ObjectId('563daabbe301d0978b000000'), "first_name"=>"SHAUN", "last_name"=>"JOHNSON"}
  > Racer.find "563daabbe301d0978b000000"
   => #<Racer:0x000000072c0680 @id="563daabbe301d0978b000000", @number=0, @first_name="SHAUN", @last_name
  > Racer.find "563daabbe301d0978b000999"
   => nil
  ```

```
$ rspec spec/crud_spec.rb -e rq04
```

5. Create an instance method in the `Racer` class called `save`. This method must:

   - take no arguments
   - insert the current state of the `Racer` instance into the database
   - obtain the inserted document `_id` from the result and assign the to_s value of the `_id` to the instance attribute `@id`

   Hint:

```
def save
  result=self.class.collection....
  @id=result... #store just the string form of the _id
end
```

   Use the `rails console` to test and explore your changes.

```
> racer=Racer.new(number:1001, first_name:"cat", last_name:"inhat", group:"masters", secs:1000)
 => #<Racer:0x00000007305a78 @id=nil, @number=1001, @first_name="cat", @last_name="inhat", @gender=nil,
> racer.save
 => "563e24c2e301d0978b0003ea"
> Racer.find "563e24c2e301d0978b0003ea"
 => #<Racer:0x0000000731ab08 @id="563e24c2e301d0978b0003ea", @number=1001, @first_name="cat", @last_nam
```

```
$ rspec spec/crud_spec.rb -e rq05
```

6. Create an instance method in the `Racer` class called `update`. This method must:

   - accept a hash as an input parameter
   - updates the state of the instance variables – except for @id. That never should change.
   - find the racer associated with the current `@id` instance variable in the database
   - update the racer with the supplied values – replacing all values

   Hint:

```
def update(params)
  @number=params[:number].to_i
  @first_name=params[:first_name]
  @last_name=params[:last_name]
  @secs=params[:secs].to_i
           ...
  params.slice!(:number, :first_name, :last_name, :gender, :group, :secs)
  self.class.collection
           ...
end
```

   Use the `rails console` to test and explore your changes.

```
> doc=Racer.all({first_name:"cat",last_name:"inhat"}).first
> racer=Racer.find doc[:_id].to_s
> racer.first_name
 => "cat"
> racer.update(first_name:"thing", last_name:"one", group:"15 to 19")
> pp Racer.all(:_id=>doc[:_id]).first
{"_id"=>BSON::ObjectId('5663d512e301d0a256000fa3'),
 "first_name"=>"thing",
 "last_name"=>"one",
 "group"=>"15 to 19"}
```

```
$ rspec spec/crud_spec.rb -e rq06
```

7. Create an instance method in the `Racer` class called `destroy`. This method must:

- accept no arguments
- find the racer associated with the current `@number` instance variable in the database
- remove that instance from the database

Hint:

```
def destroy
  self.class.collection
            ....
end
```

Use the `rails console` to test and explore your changes.

```
> Racer.find(1001).first_name
 => "thing"
> Racer.find(1001).destroy
 => #<Mongo::Operation::Result:57331320 documents=[{"ok"=>1, "n"=>1}]>
> Racer.find(1001)
 => nil

$ rspec spec/crud_spec.rb -e rq07
```

Completing ActiveModel Framework

In this section we will transform the `Racer` class into a `Racer` model class by adding a few constructs to make the class look like an ActiveModel instance used by the Rails scaffold.

1. Add the `ActiveModel::Model` mixin to the `Racer` class.

```
class Racer
  include ActiveModel::Model

$ rspec spec/model_spec.rb -e rq01
```

2. Add an instance method to the `Racer` class called `persisted?`. This method must:

   - accept no arguments
   - return true when @id is not `nil`. Remember – we assigned @id during save when we obtained the generated primary key.

   Hint:

```
def persisted?
  !@id.nil?
end
```

   Use the `rails console` to test and explore your changes. Remember that most of the methods we added return MongoDB hashes and `persisted?` is an instance method of `Racer`. You can only call this method on objects of type `Racer`.

```
> id=Racer.all({first_name:"thing", last_name:"one"}).first[:_id].to_s
 => "563e24c2e301d0978b0003ea"
> Racer.find(id).persisted?
 => true

$ rspec spec/model_spec.rb -e rq02
```

3. Add two instance methods called `created_at` and `updated_at` to the `Racer` class that act as placeholders for property getters. They must

- accept no arguments
- return nil or whatever date you would like. This is, of course, just a placeholder until we implement something that does this for real.

Hint:

```
def created_at
  nil
end
def updated_at
  nil
end

$ rspec spec/model_spec.rb -e rq03
```

Adding Controller and View

In this section you will make the model class accessible to the browser by adding a controller and view through Rails scaffold. You carefully implemented the methods and detaisl of the `Racer` model class so that it should nearly immediately work with these generated classes.

1. Generate a controller and view for the `Racer` model using the `scaffold_controller` command. Identify the fields for the model object so the generator creates fields to display and manage them.

   Hint:

   ```
   $ rails g scaffold_controller racer number:integer first_name last_name gender group secs:integer
   ```

2. Add a route to the new controller in `config/routes.rb` and set the `racers#index` page to be the root URI for the application.

   ```
   Rails.application.routes.draw do
     resources :racers
     root to: 'racers#index'
   ```

   Use the `rake routes` to test and explore your changes.

   ```
   $ rake routes
       Prefix Verb   URI Pattern                  Controller#Action
       racers GET    /racers(.:format)            racers#index
              POST   /racers(.:format)            racers#create
    new_racer GET    /racers/new(.:format)        racers#new
   edit_racer GET    /racers/:id/edit(.:format)   racers#edit
        racer GET    /racers/:id(.:format)        racers#show
              PATCH  /racers/:id(.:format)        racers#update
              PUT    /racers/:id(.:format)        racers#update
              DELETE /racers/:id(.:format)        racers#destroy
         root GET    /                            racers#index
   ```

   If you attempt to access the index page, you will notice an error stating the following. That is because our all() method returns a collection of hashes and not a collection of `Racer` instances so that lazy loading can take place. However, we can fix that by using a `helper` and our `Racer.initialize` that takes a hash.

   ```
   undefined method `number' for {}:BSON::Document
   <% @racers.each do |racer| %>
     <tr>
       <td><%= racer.number %></td> <=========
       <td><%= racer.first_name %></td>
       <td><%= racer.last_name %></td>
       <td><%= racer.gender %></td>
   ```

3. Add an instance method to the generated `RacersHelper` class called `toRacer`. This class was generated by the `scaffold_controller` command and placed in `app/helpers/racers_helper.rb`. The new method must:

   - accept a single input argument
   - if the type of the input argument is a `Racer`, simply return the instance unmodified. Else attempt to instantiate a Racer from the input argument and return the result.

   Hint:

```
module RacersHelper
  def toRacer(value)
    return value.is_a?(Racer) ? value : Racer.new(value)
  end
end
```

   Insert a call to the helper method in `app/views/racers/index.html.erb`

   Hint:

```
<% @racers.each do |racer| racer=toRacer(racer) %>
```

   Fix the JSON marshalling in `app/views/racers/index.json.jbuilder` by adding the call to the helper method as well

   Hint:

```
json.array!(@racers) do |racer|
  racer=toRacer(racer)
  json.extract! racer, :id, :number, :first_name, :last_name, :gender, :group, :secs
  json.url racer_url(racer, format: :json)
end
```

4. Remove the confirmation dialogs from your Destroy link since we are not using a webdriver that supports javascript for this assignment. Inside the index.html.erb file you will need to change the destroy link to eliminate the confirmation dialog:

```
from: <%= link_to 'Destroy', racer, method: :delete, data: { confirm: 'Are you sure?' } %>
to: <%= link_to 'Destroy', racer, method: :delete %>
```

5. Access the root URI for the application and take your application for a test drive.

```
$ rspec spec/scaffold_spec.rb
```

Adding Pagination

Although we are not using a large dataset, we are still using an amount of records sorted by an non-indexed property and can notice some delay in accessing a page with all of our results. Add `will_paginate` support for paging. The UI portion will be handled automatically but you must add query support for the new paginated call. Luckily you already have most of that implemented.

1. Add a class method to the `Racer` class called `paginate`. This method must:

- accept a hash as input parameters
- extract the `:page` property from that hash, convert to an integer, and default to the value of `1` if not set.
- extract the `:per_page` property from that hash, convert to an integer, and default to the value of `30` if not set
- find all racers sorted by `number` assending.
- limit the results to page and limit values.

- convert each document hash to an instance of a `Racer` class
- Return a `WillPaginate::Collection` with the `page`, `limit`, and `total` values filled in – as well as the page worth of data.

  Hint:

  ```
  def self.paginate(params)
    page=(params[:page] || 1).to_i
    limit=(params[:per_page] || 30).to_i
    skip=(page-1)*limit

    racers=[]
    ...find racer docs
       racers << Racer.new(doc)
    ...
    total=...get collection size

    WillPaginate::Collection.create(page, limit, total) do |pager|
      pager.replace(racers)
    end
  end
  ```

  Use the `rails console` to test and explore your changes.

  ```
  > reload!
  > page=Racer.paginate(page:1)
  > page.current_page
   => page 1
  > page.per_page
   => 30
  > page.total_pages
   => 34
  > page.count
   => 30
  > page.total_entries
   => 1001
  ```

  ```
  $ rspec spec/paginate_spec.rb -e rq01
  ```

2. Update the `racers#index` method to use the new `Racer.paginate` method instead of the scaffold `Racer.all` method. This method is located in `app/controllers/racers_controller.rb`

   Hint:

   ```
   def index
     #@racers = Racer.all
     @racers = Racer.paginate(page:params[:page], per_page:params[:per_page])
   end
   ```

   Use the browser to test and explore your changes.

   ```
   http://localhost:3000/racers?page=1&per_page=5
   ```

   ```
   $ rspec spec/paginate_spec.rb -e rq02
   ```

3. Add the `will_paginate` command to the `racers#index` view page in `app/views/racers/index.html.erb`.

   Hint:

```
<table>
  <tbody>
    <% @racers.each do |racer| racer=toRacer(racer) %>
      ...
    <% end %>
  </tbody>
</table>

<%= will_paginate @racers %>
```

User the browser to test and navigate the pages of racers. Note that will_paginate does not autotically add anything for page_size.

```
$ rspec spec/paginate_spec.rb -e rq03
```

Heroku Deployment

(Ungraded/Optional) In this section you will optionally configure your solution for a Heroku deployment. Feel free to submit the assignment for grading and continue on with this optional work. You should be able to find details about the changes required for deployment within the `Zips` example and in the lecture on deployment.

1. Create a database and user account on `MongoLab`.
2. Import the `race_results.json` into the MongoLab database.
3. Create an application on `Heroku`. Name your application `raceday#####` where `#####` is a random, unassigned number.
4. Configure the application for use on Heroku by:
   - updating the `config/mongoid.yml` file with a deployment profile
   - updating the `Gemfile` to satisfy `Heroku` RDBMS constraints for ActiveRecord.
5. Deploy the application to Heroku and access via the web.

Self Grading/Feedback

Some unit tests have been provided in the bootstrap files and provide examples of tests the grader will be evaluating for when you submit your solution. They must be run from the project root directory.

```
$ rspec
...
(N) examples, 0 failures
```

You can run as many specific tests you wish be adding `-e rq## -e rq##`

```
$ rspec (spec path) -e rq01 -e rq02
```

Submission

Submit an .zip archive (other archive forms not currently supported) with your solution root directory as the top-level (e.g., your Gemfile and sibling files must be in the root of the archive and *not* in a sub-folder. The grader will replace the spec files with fresh copies and will perform a test with different query terms.

```
|-- app
|   |-- assets
|   |-- controllers
|   |-- helpers
|   |-- mailers
|   |-- models
|   `-- views
|-- bin
|-- config
|-- config.ru
|-- db
|-- Gemfile
|-- Gemfile.lock
|-- lib
|-- log
|-- public
|-- Rakefile
|-- README.rdoc
|-- test
`-- vendor
```

Last Updated: 2016-01-22