CSE 341, Winter 2009, Assignment 7
Ruby Project: Pulp Fiction IQ Test
Due: Wednesday March 11, 10:00pm

35 points total (15 points Question 1, 10 points Question 2, 10 points Question 3). Up to 5 points extra credit.

You can use up to 4 late days for this assignment.

One of the most common uses of Ruby in the past few years has been web application development, particularly with frameworks like Ruby-on-Rails (henceforth RoR). In this assignment you will write a web server that works much in the same way RoR does.

There are several different design models for web application servers, but we will be following the one RoR uses. In this model, the work is divided into two parts. One is the **controller**s, which handle the application logic (i.e. set up variables, talk to the database etc.), and the **view**s, which actually render response data (usually some HTML). Requests initially are accepted by the **server** which parses them and then passes them to a **handler**. The handler views the path given in the request with the following scheme: *http://www.example.com/controller/action* . The **controller** is a grouping of related **action**s into a class. So the path */welcome/index* maps to a Ruby class `Welcome` and the message `index`. The handler will invoke the appropriate message on the appropriate class and then invoke a view renderer to produce the final output.

Writing all of this is quite a tall task, so you are given some starter code. It includes a runnable script `server.rb` which listens for connections, and does some of the dirty work of parsing requests. **YOU SHOULD NOT CHANGE THIS SCRIPT!** There are three other classes which you will be modifying: `handler.rb` which defines a class `Handler`, `renderers.rb` which defines a class `ERBRenderer`, and `controller/base_controller.rb` which defines a class `BaseController`.

In the `controller` folder as well as the `views` folder there are some Ruby classes and view files, respectively, that define a Pulp Fiction IQ web application. They are not important for the assignment; however, they will provide some confirmation that your code is working. As you complete more of the following steps, you will be able to view more of the application.

To start the server, invoke `ruby server.rb` in the command line, in the folder that contains your code. You can them point your browser to http://localhost:8080/welcome/index Before modifying any code, you should see a page that just says `Hello World`

NOTE: For this assignment, file names and locations **MATTER**. You may add files if you like, and they can be named as you please, but **DO NOT** move the existing files.

1. Modify `handler.rb` to invoke controllers and actions correctly: You need to modify `Handler#process`. This method takes in two parameters, `request` and `response`. These parameters will be instances of the classes `Request` and `Response` which are defined in `server.rb`.

   `Request` has three methods, `method` (you can ignore this), `uri` which returns a String of the form `"/controller/action"`), and `query` which returns the a query string of the form "key1=value1&key2=value2".

   `Response` has three methods, `success`, `not_found`, and `internal_error` which each take a block. The block you pass in should take a `Hash` and an `IO`.

   HINTS:

   (a) You need to dynamically load the controller for each request. The controllers will all be in the controller folder, have a filename which matches the controller part of the URI (with the addition of the .rb extension). The class name of the controller is the Ruby case equivalent of the same. The private method `ruby_case` in the `Handler` class can handle this conversion.

   (b) `Kernel.const_get`

   (c) `Object#send`

   (d) For parsing the URI and query string, look at the Ruby standard library's `Regexp#match`

   After finishing this step, the page http://localhost:8080/welcome/index should display.

2. The starter code only has one of two of the classes, `TextRenderer`, in `renderers.rb` fully implemented. Extend ERBRenderer to evaluate view files found in the views folder using ERB. ERB is a templating system in the Ruby standard library. See the http://www.ruby-doc.org/stdlib/libdoc/erb/rdoc/ for details. For a controller "foo" and action "bar", `ERBRenderer#result` should render the template "views/foo/bar.erb.html".

3. Add an instance variable `@params` to `BaseController` which will be passed in constructor. The controllers expect this variable to be a hash parsed from the query string. Extend `Handler#process` to construct this variable and pass it in to the controller.

   After this step you should be able to submit the form at http://localhost:8080/pulp_fiction/index and a result page (http://localhost:8080/pulp_fiction/test) should display.

General Hints:

1. In order to see changes in the browser, you must restart your server every time you make a change. A nice way of avoiding this is to write unit tests early, and test your code with those before moving to the browser.

2. You can, and should use the Ruby standard library extensively in this assignment. You can find class references for the core and standard libraries at http://www.ruby-doc.org/, or just google for "ruby [CLASSNAME]" and it will usually be the first result.

Extra Credit:

1. Add error handling to `Handler#process`. In particular, respond with a 404 page when a controller or action do not exist.

2. Make `BaseController#method_missing` more efficient. When a missing method is invoked, dynamically add a method to the class.

3. Create some controllers that use sqlite3 or some other datastore (like a file) to produce a stateful web application.

Turnin: Turn in a zipfile with your final code. On the lab Linux machines, if `ruby_project` is the directory containing your code, you can do this by running "`zip -r ruby_project.zip ruby_project`". This zip file should contain all code, including your changes and code you did not modify.