CS142 Project #3: Dynamic Page Generation with Rails

Setup

You should already have installed Ruby on Rails for Project #2. If not, follow the installation instructions now. Once Rails is installed, create a Rails application for this project by invoking the following command:

```
rails new project3
```

This will create a directory `project3` that contains a skeleton for the project. Your solutions for all of the problems below should be implemented in this project. Change to that directory and invoke the following command:

```
rails server
```

This will start a Web server in this project, and the server's log information will appear on your console. Now go to your Web browser and enter the following URL:

```
localhost:3000
```

This should display a default root page provided by Ruby on Rails. You can type control-C to the server to kill it.

Additional Setup

We modify the default Rails configuration in ways for this class. First, enable the traditional ("non RESTful") style of routing by editing the file `project3/config/routes.rb`. The last line in the file should look like this:

```
# match ':controller(/:action(/:id))(.:format)'
```

Uncomment this line by removing the `#`.

Second, modify the session configuration so that sessions are stored in the database rather than in cookies (you will learn about sessions later in the class). To do this, edit the file `project3/config/initializers/session_store.rb`. Find the line containing `:cookie_store` and comment it out by adding "# " at the beginning. Then find the line containing `:active_record_store` and uncomment it by removing the "# " at the beginning. Finally, invoke the following shell commands:

```
rails generate session_migration
rake db:migrate
```

Problem 1: Show Matching State Names (10 points)

Create a model, view, and controller in the `project3` application, which together will display the names of all states containing a given substring. Your code must implement the URL `/state/filter`, which accepts a query value named `substring`. The URL should produce a Web page that lists in alphabetical order all states whose names contain the given substring (ignoring differences in case). For example, the Web page for `/state/filter?substring=al` should list the states Alaska, Alabama, and California. The page should also display the substring that was used to filter the states. If there are no matching states then the Web page should display a message indicating that fact (rather than just showing nothing).

First, create a model that will manage the data. The model should be implemented as a class named `State` with a single class method `State.filter` (a class method is one that can be invoked without an instance of the class, like a static method in Java). `State.filter` should take a single string argument and return an array containing the names of all states whose names include the argument as a substring (ignoring differences in case). To save typing you can download states.rb, which contains Ruby code to create an array containing the names of all of the states. You can copy this code into your application.

Once you have defined the model you can then create the controller and view. Follow the Rails conventions for file and method naming and for where to put various functionality. Note that Rails provided a default layout in `project3/app/views/layouts/application.html.erb`, which handles everything except the contents of the <body> element. Replace the `DOCTYPE` and `html` lines at the top of this file with the standard header lines

for this class.

Problem 2: Personalizing the Layout (5 points)

Modify the layout so that it displays a personalized header at the top of the page (the layout will now include some HTML inside the <body> element). Use your imagination and creativity to create a header that is "uniquely you". This can include images, graphics, whatever you like. Be creative! From now on, use this layout for all of your Rails projects in the class. Each individual page should be able to specify its own title, which gets used by the default layout.

Problem 3: Partial Template For Tabs (10 points)

Define a new controller and view that implement the URL `/tabs/show2`. This URL must display a page containing two sets of tabs separated by a half-inch or so of white space. The first set must be exactly the same as your solution to Problem 2 in Project 1. The second set must have the same general appearance but must display different labels (which you can choose). You must use a partial template named `tabs` to generate the tabs; your view should invoke it twice. The partial template takes two parameters. The first is an array of hashes, one for each tab; each hash has a `:label` element containing the text to display in the tab, and a `:url` element containing the URL to visit when that tab is clicked. The second parameter for the partial template is a string containing the label of the selected tab; the partial template uses it to identify the selected tab and display it differently.

Style Points (10 points)

These points will be awarded if your problem solutions have proper MVC decomposition, follow the Rails conventions, and generate valid XHTML. In addition, your code and templates must be clean and readable. The last 5 style points will be reserved for layouts that are unusually interesting (only a few projects will receive these points).

Useful Hints

- To run your application, `cd` to the directory containing the application and invoke the following command:

  ```
  rails server
  ```

  This will start a Web server for your application on port 3000. Once you have done this, you can click on the links above to request URLs from your application.
- For Problem 3, you should create a directory `project3/public/images` and place the images for the tabs there . Everything in the `public` directory will be accessible via URLs; for example, the image `project3/public/images/left.gif` will be accessible at the URL `/images/left.gif`.

Deliverables

Use the standard class submission mechanism to submit the entire application (everything in the `project3` directory). Note: all of your generated HTML must validate as proper XHTML (you do not need to validate your CSS for this project).