Rails Worksheet 2: Using Forms
CSC 340

You've just learned a lot about how Rails works by looking at scaffolding. In this worksheet, you'll practice these concepts some more by doing some of the same things manually (without scaffolding). In particular, you'll learn how to connect what the user enters in the forms to its corresponding model.

<u>**I- Generating the controller and model**</u>

1. Create a rails application called ws2.
2. Generate a controller for the application. The controller should be called Infos with actions new and create. (Don't forget that you must be in the ws2 folder to do this!)
3. Generate a model for the application, and call the model Info. The model Info should have attributes title (of type string) and price (of type decimal).
4. Examine the migration file and then apply the migration.
5. Check your model in the console view.
6. Add the line resources :infos to the routes.rb file to create the default routes for the infos controller. You should also comment out the routes that were automatically created when you generated the controller.

<u>**II- Creating the object**</u>

Next, let's create an object of the Info model. This object will hold the information entered in the form elements.

7. Look in the info model, and note that there is no explicit constructor defined. Or more precisely, there is no *initializer*. This means that there is a default 0-argument initializer that doesn't initialize any values.
8. In the new action of the Infos controller, write Ruby code that creates a new Info object stored in the attribute @info. The @info object will be an empty object, serving only as a placeholder so that the form_for function will work in the next step.
9. For the create action, we should first define a private info_params helper method. Re-read the explanation of the user_params method in the users_controller-SAB-Commented.rb file we just worked on in class, and write an analogous method here.
10. Now define the create method:
    a. Set an @info attribute to a new info object, passing the result of info_params as an argument.
    b. Write code that saves the object. If the object saves correctly, then assign the String 'Object created successfully, thank you' to the attribute @display_message. There's nothing special about this variable name – we'll just use it in a view later on.

c. Otherwise, assign a different but appropriate message to @display_message and render the new page again. To render the index page, write: render 'new'

III- Creating the form

Now, let's create a form. We will use the Rails helper method form_for. We will create the form elements in the **new.html.erb** view of the Infos controller.

Start the form with the helper form_for (@info), which connects the form to the @info object that we created in the Infos controller new action. The object f represents the form element instance. The new view should be implemented as follows:

```
<h1>Please enter information about an Info object</h1>
<%= form_for @info, :action => :create do |f| %>


  <%= f.label :title %><br />
  <%= f.text_field :title %> <br />

  <%= f.label :price %><br />
  <%= f.text_field :price %>   <br />

  <%= f.submit "Submit" %>   <br />

<% end %>
```

After the submit button is pressed, the action create will be called. This is the default behavior of rails 3.0+, so the line :action => :create is not necessary, but now you see how you could make other actions occur.

Look over the rest of the form_for call and remind yourself of how it works.

To check that your web application is working properly, do the following:

1. Start the server.
2. Browse to http://.../infos/new. Enter values in the form fields and hit the submit button.
3. Open up the rails console and verify that an object of the class Info was created with the values specified in the form fields.

IV- Adding validation and rendering

1. In the create view, display the @display_message. Check your application to make sure everything is working properly.

2. Using an attribute like @display_message is not really the best way to accomplish what we just did. Instead, let's use the flash. The flash is a special part of the HTTP session. When a value is placed in it, it is only available for

the very next HTTP request. After that, it is automatically cleared. So it is very useful for storing confirmation and error messages.

Delete the lines in the controller that assign a string object to @display_message and replace it with: flash[:notice] = 'Object created successfully, thank you for the information' and flash[:notice] = 'Please re-enter the information in the forms'. In the create view, instead of displaying @display_message, display the flash[:notice] object.

Check your application to make sure everything is working properly.

As you can see, flash is pre-defined as a hash. There's nothing special about the :notice key, though. You can use whatever keys you want.

Note that, barring some database error beyond your control, there's no reason at this point that any save should fail, and so the "re-enter" message won't ever show up. Later we'll play some more with the idea of adding validations, to make it possible for the save to fail.