Homework 2: A Thorough Guide for Git, Heroku and Coding with Rails

For any part of the assignment, PLEASE ASK IF THERE ARE PROBLEMS!! I have noted major checkpoints at which progress should be made and committed. You should not be troubleshooting a single error for more than an hour in earnest. Email, or better yet, ask on Aprender so that I'm not the only one responding. You can post your messages as anonymous.

For those of you who made it a long way in the assignment-- the main issue I saw was with regard to the checkboxes. All boxes should start off in a checked state.

Getting Git Happy

1) Create a git repository, if you have not done so already. If you are not familar with git and still don't understand its purpose, git is probably one of the most valuable tools you'll learn in this course. It is a data management and repository tool. Any time you are making changes to large amounts of code, it can prove to be invaluable-- it marks all changes, lets you go back, and keeps records very cleanly. It is also extremely handy when working with groups of people where multiple people are making changes. Github has made the task of using get much easier-- I am currently working with them to set up an "Organization" account where we can all have free private repositories for use in the class. In the old days, you had to deploy git yourself and set up that server and maintain it-- Github makes the task FAR easier.

a. If you don't have one, make a free account on github.
b. In the top right corner, you will see a button to create a new repo. Create one for HW2. Make it public- at this point in time, GitHub has not provided our class with rights to make a private repository—I am working on this with them for our remaining assignments and the project!!
c. Notice after creation that a link is provided- it should look like: https://github.com/kristen-justice/HW2.git

2) Clone your code to a directory in the VM using:
git clone git://github.com/saasbook/hw2_rottenpotatoes.git
This will create a folder called hw2_rottenpotatoes in the directory where you ran the command.
3) Since you got this from a git repository, you need to remove the existing git association (this is not actually necessary since you can change your naming scheme, but for the point of this project, do this if you haven't found another way around it.)
git remote rm origin
4) Next, put the code into your own git repository:
git init
git remote add origin https://github.com/yourgitusername/yournewrepository.git
git push origin master
Now if you go to your github page and refresh, the copied code should be there.
5) Next, tell git to track all of your code and keep tracking it:
git add –A

6) You are now ready to modify your code. Anytime you have something working or make a major change:
git commit (it will prompt you to enter a message representing your changes using emacs. Control-X to exit, y to save, go to the default file)
git push

You are now ready to change code, and git is set up to record ALL changes whenever you tell it to. (git commit; git push)

As you change your code, run the code locally, then commit/push when it's working.

Leading to Part 1
In your HW2 directory, run bundle install --without production
This will ignore the PostgreSQL gem for your local installation, since that gem will cause problems if you're using a development environment without PostgreSQL installed.

Next, tell rails to load in the database: **rake db:migrate**
This applies ***all*** RottenPotatoes migrations to your local database (aka it will set up your tables for movies).

Now run **rake db:seed** to seed your local database with the movies from the seed file. This fills in the Movie table with a bunch of movie data.

Only rake the db and the seed file once!! Otherwise, you will end up with multiple entiries.

Verify that you can successfully run the app using **rails server** and interact with it via a Web browser, as described in the book and in lecture. Check that you can see a list of all the movies when you access localhost:3000/movies

At this point, if you have problems with any of the above, contact someone!!
Forget about deploying on Heroku for the moment.

Now you're to the code. I am going to go into more detail than is in the assignment and give more guidance for each part. Skip around if you have the code working already.

RottenPotatoes Enhancement, Part #1
1) Make Movie Title and Release Date clickable links.
   a. General idea: use link_to to create a link in your view.
http://api.rubyonrails.org/classes/ActionView/Helpers/UrlHelper.html#method-i-link_to
In your URL, you should use the route that routes to index.html, action index. (do rake routes to see the routing table)
Inside the URL, pass a hash that has a symbol as a key and a value of a string (e.g. :sort_by=>"title")
By doing this, in the controller, you can say params[:sort_by] and get out "title"

The link should also have an id associated with it. Id should be a symbol. This should be assigned to “title_header” or release_date_header.
At this point, reload your page in your browser—you should have 2 links in your table header. Watch for errors being reported by rails server!! If you have syntax errors, they will start showing up there.

b. Now, edit your controller. If you click one of the links, notice how the URL changes. The information after the ? is accessible via params. For example, if the URL says ….?sort=”title”, you can look at params[:sort] to get out “title.”

c. For sorting, keep in mind that you are not updating the table—you’re simply changing the find command. In the original code, you have Movie.all. This gives you all movies. Specifically, you want to change the order. There are a number of ways to do this.
First: http://apidock.com/rails/ActiveRecord/Base/find/class
Notice the part to change the order.
A simpler way is to take advantage of metaprogramming using the order method.

d. CHECK YOUR WORK! Reload your site: localhost:3000/movies
If it’s working, git commit then git push. Otherwise, debug!!!

e. Now you can change the CSS. Changing the css changes how the view looks. In standard html, you’d say something like <p class=”hilite”> This means find the code for hilite and apply it to this element. In haml, it’s very similar: %p{:class=>”hilite”} You do not want hilite to be hardcoded in though—set an instance variable in your index method of the controller to say that the variable should be set to “hilite” or nothing. Use that instance variable in your view to change the class. If it’s nothing, then nothing will change. To insert the CSS information, paste this simple CSS into RottenPotatoes' **app/assets/stylesheets/application.css** file.

f. CHECK YOUR WORK! Reload your site: localhost:3000/movies
If it’s working, git commit then git push. Otherwise, debug!!!

2) Filter the List of Movies

a. In this section, you add Ratings to your movie model and tell the controller and view to deal with those ratings.

b. To start, include the code at http://pastebin.com/vpPqkWMb in your view. Be careful that the tabbing stays the same. Once you’ve put this in your view, reload your site. Commit and push when correct.

c. To use it, we need to tell the movie model to deal with ratings. In the model, create a **class** method that sets a **class variable** all_ratings that is set to the array of possible movie ratings. This goes in the model because it is an attribute of a movie.

d. Now the controller can access the class variable all_ratings and compare against it. In the controller, you now want to include code that will 1) check the ratings that were selected in the view and 2) find movies based on that parameter.

i. Regarding (1), try viewing the source of the movie listings with the checkbox form, and you'll see that the checkboxes have field names like **ratings[G], ratings[PG],** etc. This trick will cause Rails to aggregate the values into a single hash called **ratings,** whose keys will be the names of the ***checked boxes only*** and whose values will be the value attribute of the checkbox (which is "1" by default, since we didn't specify another value when calling the **check_box_tag** helper). That is, if the user checks the 'G' and 'R' boxes,

params will include as one if its values **:ratings=>{"G"=>"1", "R"=>"1"}** .

ii. In this, you only care about the keys. http://www.ruby-doc.org/core-1.9.3/Hash.html Extract the keys for use in your controller for part 2. In the controller, you want to find movies that have the given ratings. Look at the find documentation again. Remember that, thanks to metaprogramming, you can do things like Movie.find_all_by_title (returns all found items) or Movie.find_by_title(returns only the first found item). For example, "Movie.find_all_by_title("Star Wars"). Hint: find is a very versatile method—you can also feed it a collection!

iii. CHECK YOUR WORK! Reload your site: localhost:3000/movies If it's working, git commit then git push. Otherwise, debug!!!

iv. A few notes here:

1. If the user checks (say) 'G' and 'PG' and then redisplays the list, the checkboxes that were used to filter the output should appear checked when the list is redisplayed. This will require you to modify the checkbox form slightly from the version we provided above. See http://apidock.com/rails/ActionView/Helpers/FormTagHelper/check_box_tag specifically example 3.

2. **The first time the user visits the page, all checkboxes should be checked by default (so the user will see all movies). – MANY SUBMISSIONS ARE MISSING THIS!!!!!** Hint: set an instance variable with the currently checked boxed. Then modify check_box_tag in your view to mark as checked if that variable includes? the rating. In your controller, if there is nothing checked, set the variable to all_ratings by mapping the values from the all_ratings method (defined in the model) to a hash of the ratings.

v. Now, you're probably having the issue that you can sort or you can find by rating, not both. There are 2 ways to deal with this. One is to move on to part 3. Another is to set up 2 hidden tags in the form that includes your check boxes (the better way). See http://api.rubyonrails.org/classes/ActionView/Helpers/FormTagHelper.html#method-i-hidden_field_tag

vi. **IMPORTANT for grading purposes**:

1. Your form tag should have the id `ratings_form`
2. The form submit button for filtering by ratings should have an HTML element id of `ratings_submit`.
3. Each checkbox should have an HTML element id of `ratings_#{rating}`, where the interpolated `rating` should be the rating itself, such as "PG-13", "G". i.e. the id for the checkbox for PG-13 should be `ratings_PG-13`.

vii. CHECK YOUR WORK! Reload your site: localhost:3000/movies If it's working, git commit then git push. Otherwise, debug!!!

viii.

3) Remember the Settings

a. Now we get to the session variables. Remember that the session persists until

cookies are cleared or the session is explicitly closed by the programmer. (Session and flash quack like a hash, but they last longer- remember the differences!!)

b. You likely already have variables checking for params. Add to it using or (||) to check for session information also. You access the variables in the same way. (e.g. session[:sort_by]) Your checked rating variable and your variable representing sorting should do an or between params, session, and {}. For example, if session[:sort] == "title" but params[:sort] no longer does, you don't want to lose that information. If nothing is in params or session, then what should you do? For sort, nothing (aka the default is unsorted). But for ratings, the checked boxes should default to a hash made up of the information from Movie.all_ratings. If sorts don't match (params vs session), set session to what's in params. If ratings don't match, do the same.

c. If something didn't match in params or session, you want to redirect, as was discussed in our last class. Redirect_to with the sort and ratings set to the proper variables and return. See http://api.rubyonrails.org/classes/ActionController/Redirecting.html for help beyond the class notes.

d. There is an important corner case to keep in mind here, though: if the previous action had placed a message in the **flash[]** to display after a redirect to the movies page, your additional redirect will delete that message and it will never appear, since the **flash[]** only survives across a single redirect. To fix this, use **flash.keep** right before your additional redirect.

e. CHECK YOUR WORK! Reload your site: localhost:3000/movies
If it's working, git commit then git push. Otherwise, debug!!!

At this point, your code is hopefully working on your local machine, and you've pushed your commits along the way to git.

Deploying to Heroku
Although you can deploy at any time and test on Heroku, Heroku does not provide as obvious errors as Ruby on Rails.

To deploy, (assuming you have not before), run heroku create –stack cedar. This will create a new Heroku project—again, this is not a step you repeat multiple times!!

Then do git push heroku master (Note that if you change the name of your heroku app before this, you will get errors—I'll post how to do that and not get errors later)

Run heroku ps to see that your process is running. If it says it's stalled, killed, or otherwise dead, there's a problem!! At the terminal, type heroku logs –t to see the log file. Syntax errors and heroku related errors will appear there. Load your page- in the browser, go to your-appname-somenumber.herokuapp.com/movies
If an error occurs, go back to your terminal and see what errors the log file is now reporting. To close the log, do control-C. (heroku logs –t means to tail the log file and keep updating as stuff occurs—leaving off the –t will just show the end of the current file and end there.)

Finally, make sure to push the db and seed file to heroku so that it has data to work with:
heroku run rake db:migrate to apply ***all*** RottenPotatoes migrations -- in this case, we only have one migration file that creates the Movies table
heroku run rake db:seed to seed your Heroku app's database with the movies we've created in seeds.rb

If you get an error in either of these steps, check your log!! An error in these two steps implies an issue in your Movie model.
Once **heroku run rake db:seed** works the first time, do not run it again! It will duplicate your database. (If you do, there's an announcement about how to reset the database)

Debugging Notes (to be added to as they come in):
- If you get an error while pushing to heroku, check out:
http://stackoverflow.com/questions/4269922/permission-denied-publickey-when-deploying-heroku-code-fatal-the-remote-end
I've had this issue a few times, and each time running "heroku keys:add ~/.ssh/id_rsa.pub" has worked for me.
- Similar issues have been seen when pushing to github. If this is the case, follow this guide:
https://help.github.com/articles/error-permission-denied-publickey
Note that the last part is something done on the github website. I had this happen when I cleaned out my public keys in linux and tried to modify them without updating github.