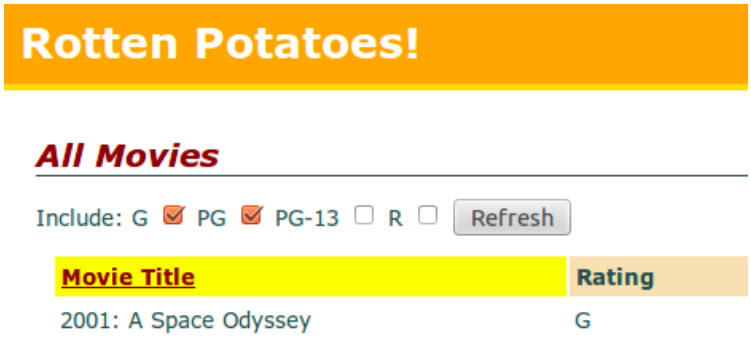
ROTTENPOTATOES ENHANCEMENT #2: FILTER THE LIST OF MOVIES

In this portion of the assignment, you will creating functionality within RottenPotatoes that allows the user to filter movies by MPAA rating. The filter will be controlled by checkboxes at the top of the "All Movies" listing:



When the "Refresh" button is pressed, the list of movies is redisplayed showing only those movies whose ratings were checked.

This will require a couple of pieces of code. We have provided the code that generates the checkboxes form, which you can include in the `index.html.haml` template, here on Pastebin (http://pastebin.com/vpPqkWMb). *However*, you have to do a bit of work to use it: our code expects the variable `@all_ratings` to be an enumerable collection of all possible values of a movie rating, such as `['G','PG','PG-13','R']`. The controller method needs to set up this variable. And since the possible values of movie ratings are really the responsibility of the Movie model, it's best if the controller sets this variable by consulting the Model. Create a class method of `Movie` that returns an appropriate value for this collection.

You will also need code that figures out (i) how to figure out which boxes the user checked and (ii) how to restrict the database query based on that result.

Regarding (i), 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"}`. Check out the `Hash` documentation (http://www.ruby-doc.org/core-

1.9.3/Hash.html) for an easy way to grab just the keys of a hash, since we don't care about the values in this case.

Regarding (ii), you'll probably end up replacing `Movie.all` with `Movie.find`, which has various options to help you restrict the database query.

Specific Requirements

- Make sure that you don't break the sorted-column functionality you added in part 1B! That is, sorting by column headers should still work, and if the user then clicks the "Movie Title" column header to sort by movie title, the displayed results should both be sorted and be limited by the Ratings checkboxes.
- 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.
- The first time the user visits the page, all checkboxes should be checked by default (so the user will see all movies). For now, ignore the case when the user unchecks all checkboxes—you will get to this in the next part.

Important for grading purposes

- Your form tag should have the id `ratings_form`
- The form submit button for filtering by ratings should have an HTML element id of `ratings_submit`.
- 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”. For instance the id for the checkbox for PG-13 should be `ratings_PG-13`.

Hints and advice

- Don't put code in your views! Set up some kind of instance variable in the controller that remembers which ratings were actually used to do the filtering, and make that variable available to the view so that the appropriate boxes can be pre-checked when the index view is reloaded.

You will submit a text file, containing a single line with the url of your heroku deployment. The url will look like this: `http://your-app-name.herokuapp.com`

Choose Files No file chosen

Check

Show Discussion

New Post