# Homework 2: An Intro to Rails- START EARLY!!! Due September 26!!!

Remember this is a new infrastructure. And it is one that you will be using to start creating your own projects. Please start early and ask questions ASAP.

CORRECTION: the contents needed for Homework 2 is not in ~/Assignments/hw2 directory. It is in ~/Assignments/hw3 directory.

Furthermore, edit the text of the gemfile in ~/Assignments/hw3 with a text editor to change the version of ruby to say ‘ruby 2.3.0’ to update Ruby from an outdated version. And remove the old “.git” directory in both hw2 and hw3 directory: “rm -rf ~/Assignments/hw2/.git” and “rm -rf ~/Assignments/hw3/.git”.

In this assignment, you will begin your first Rails application and see how Rails apps are built. The application is a Project Management system that we may later use for organizing your real class projects. You will start the application from skeleton code. You will then add some basic functionality of sorting, filtering, and remembering these sorting/filtering decisions. You will submit 1) on github and 2) on Blackboard.

## Github- Getting Set Up

1. If you have not already, create an account on Github (<https://github.com/>)
   1. Send me your github username **via Piazza** (Note: this username is NOT your email address)
   2. Once I have your information, I will set up an account for you for use in the class. This will be used for all future homeworks and your project.
   3. **Do this ASAP as you will not be able to submit this assignment properly until I have your account set up!**
   4. When I link your accounts, you will receive an email asking you to accept the invitation: “Invitations are sent via email and can be accepted at<https://github.com/UCCS-CS3300-Fall2016>”
2. Setting up your environment
   1. In your Linux environment, navigate to ~/Assignments. (This will be known as https://github.com/UCCS-CS3300-Fall2016$HOME for the rest of your assignments)https://github.com/UCCS-CS3300-Fall2016

In the terminal, type:   
*touch README.md* #Creates an empty README.md file  
*git init* #Initializes a repository  
*git add hw3/* #Adds your files and folders in the Assignments directory

git add README.md  
*git status* #Shows what files are staged for commit or are included in the directory, but not staged for commit to the repo  
*git commit -m "first commit"* #Creates a commit message to be associated with your commit (If you exclude the -m, a text editor will open in which you see the same info as the git status command and must enter a message). Make your commit messages meaningful (example: “Modified openProject method and created associated test.”) so you can use them later. Be professional- do not include profanity.  
  
*git remote add origin* [*https://github.com/UCCS-CS3300-Fall2016/yourUCCSName.git*](https://github.com/UCCS-CS3300-Spring2016/yourUCCSName.git) *#*(Replace yourUCCSName with your UCCS username- For example, [https://github.com/UCCS-CS3300-Fall2016/kjustice.git](https://github.com/UCCS-CS3300-Spring2016/yourUCCSName.git) )

This links your remote repository to the [https://github.com/UCCS-CS3300-Fall2016/yourUCCSName.git](https://github.com/UCCS-CS3300-Spring2016/yourUCCSName.git) one you’ve created here. All push and pulls will now be linked to this remote account. Note that “origin” is the name of the remote repository- you could call it anything. Class instructions will assume you called it “origin.”   
  
git push -u origin master #This pushes anything that was added and committed to the “origin” repository along the “master” branch. We’ll talk about branches later in the semester so you can use them for projects.

**If you get an error message on** git push -u origin master, **contact us on Piazza immediately with the command you ran and the error message received.-** Errors may imply that we did not set up your account with Write rights or that we have an incorrect github name for you (or no github name).

For more information on using git, checkout Appendices A.5 and A.6 in the book. Screencasts are also available (referenced in the book) that may be quite helpful for you to understand what’s happening.

Also, if you don’t want to have to type your git username and password every single time you do a push or pull (it gets annoying quickly…), follow these instructions: https://help.github.com/articles/caching-your-github-password-in-git

**Preparation**: If you've never used Rails before please follow [these screencasts](http://www.youtube.com/watch?v=5ar8SWO1lGs&list=PLjbL0BCR04Q13ph3TLp3dPsYQi7ujH6R4&index=8) to work through chapter 4 of the ESaaS textbook which show how to create a rottenpotatoes rails app from scratch. ← HIGHLY recommend that you watch and work along with the video and/or the book chapter! Note: Some of these screencasts may be a bit out of date-- we are working with the newest and greatest in here and are catching up on the videos and the rest.

**Extra Practice**: Before you start this assignment you should look at the overall code structure and the code that we have added/modified already. Check out <http://guides.rubyonrails.org/getting_started.html> to see the full process of creating a new project and the Files/Folders that rails sets up for you.

**The Real Deal:**

Switch to the HW3 directory

cd hw3/

Now you need to install various libraries with the bundle command (jquery-rails and ruby\_core\_source take awhile…)

sudo bundle install

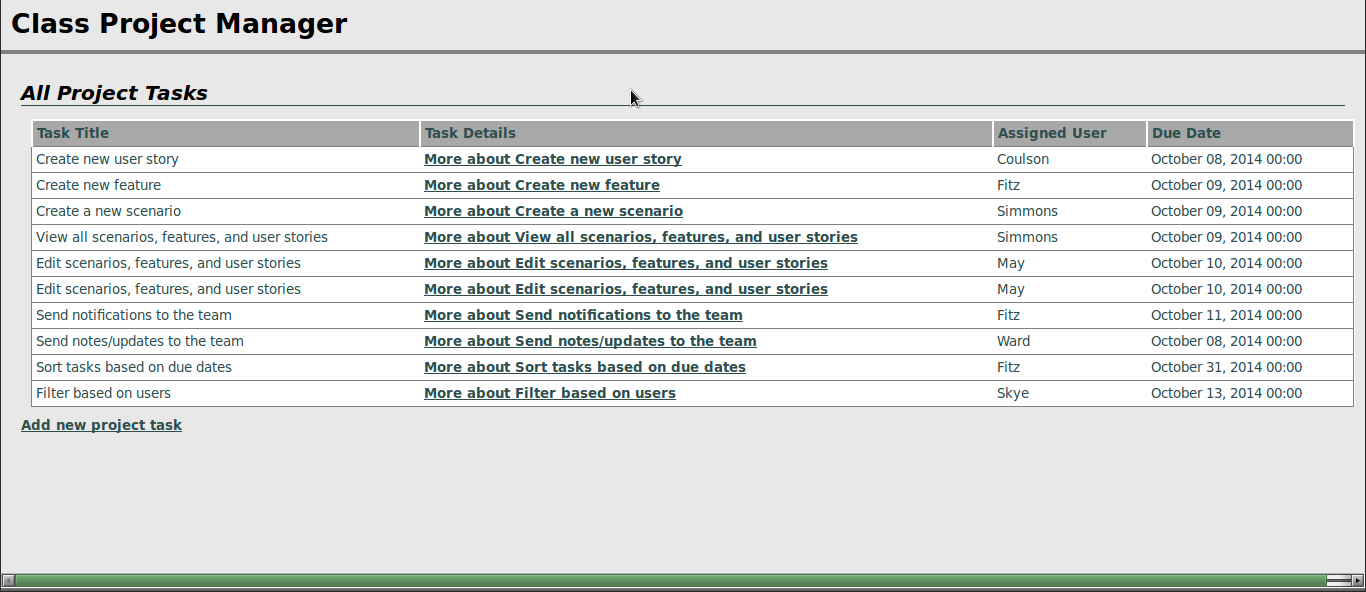
Now you can test the app locally by running the following commands

bundle exec rake db:migrate

bundle exec rake db:seed

rails s (control-c to end)-- note: This is what runs the server. If you turn it off, you can no longer navigate to the page listed below.

At this point, if you go to your browser and navigate to <http://localhost:3000>, you should see something that looks like this:



## The Assignment

Remember to push your code often!!

### PART 1: SORTING THE LIST OF ALL PROJECTS (15 points)

Enhance the Project Manager in the following way:

On the list of all project tasks page, the column headings for ‘Project Title’ and ‘Due Date’ for a task should be clickable links. Clicking one of them should cause the list to be reloaded but sorted in ascending order on that column. For example, clicking the ‘Due Date’ column heading should redisplay the list of project tasks with the earliest-due tasks first; clicking the ‘title’ field should list the project tasks alphabetically by title. (For any project tasks whose names begin with non-letters, the sort order should match the behavior of String#<=>.)

### IMPORTANT for grading purposes: (You will be penalized if you do not follow these instructions!!!)

* The link (that is, the <a> tag) for sorting by ‘title’ should have the HTML element id title\_header.
* The link for sorting by ‘due date’ should have the HTML element id due\_date\_header.
* The table containing the list of project tasks should have the HTML element id projects (this has already been set for you by existing code, if you use the version of Project Manager included in the above zip file).

When the listing page is redisplayed with sorting-on-a-column enabled, the column header that was selected for sorting should appear with a yellow background. You should do this by setting controller variables that are used to conditionally set the CSS class of the appropriate table heading to hilite, and pasting this simple CSS into the Project Manager app/assets/stylesheets/application.css file:

table#projects th.hilite {

background-color: yellow;

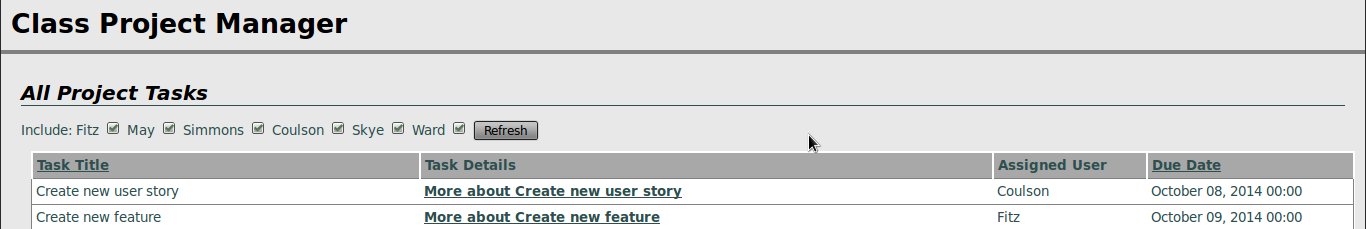
}

### Hints and caveats:

* The current Project Manager views use the Rails-provided “resourceful routes” helper projects\_path to generate the correct URI for the projects index page. You may find it helpful to know that if you pass this helper method a hash of additional parameters, those parameters will be parsed by Rails and available in the params[] hash.
* Databases are pretty good at returning collections of rows in sorted order according to one or more attributes. Before you rush to sort the collection returned from the database, look at the documentation for the ActiveRecord find and all methods and see if you can get the database to do the work for you.
* Don’t put code in your views! The view shouldn’t have to sort the collection itself—its job is just to show stuff. The controller should spoon-feed the view exactly what is to be displayed.

**PART 2: FILTERING THE LIST OF ALL PROJECT TASKS (15 points)**

Enhance the Project Manager as follows. At the top of the All Project Tasks listing, add some checkboxes that allow the user to filter the list to show only tasks with assigned to certain people. For example:



When the Filter button is pressed, the list of tasks is redisplayed showing only the tasks that are assigned to the people whose boxes 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:

= form\_tag projects\_path, :method => :get do

Include:

- @all\_users.each do |user|

= user

= check\_box\_tag "users[#{user}]"

= submit\_tag 'Filter'

BUT, you have to do a bit of work to use the above code: as you can see, it expects the variable @all\_users to be an enumerable collection of all possible values of a user, such as ['Coulson','Ward','May',’Skye’,'Fitz',‘Simmons’]. The controller method needs to set up this variable. Since the possible values of users are really the responsibility of the Project model, it’s best if the controller sets this variable by consulting the Project. Hence, you should create a class method of Project that returns an appropriate value for this collection.

You will also need code that figures out (i) how to determine 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 project task listings with the checkbox form, and you’ll see that the checkboxes have field names like users[Coulson], users[Ward], etc. This trick will cause Rails to aggregate the values into a single hash called users, 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 ‘Coulson’ and ‘May’ boxes, params will include as one if its values :users=>{"Coulson"=>"1", "May"=>"1"}. Check out the Hash documentation 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 Project.all in the controller method with Project.find, which has various options to help you restrict the database query.

### IMPORTANT for grading purposes: (You will be penalized if you do not follow these instructions!!!)

* Your form tag should have the id users\_form
* The form submit button for filtering by users should have an HTML element id of users\_submit.
* Each checkbox should have an HTML element id of users\_#{user}, where the interpolated user should be the user itself, such as “Coulson”, “Fitz”. i.e. the id for the checkbox for Coulson should be users\_Coulson.

### Hints and caveats:

* Make sure that you don’t break the sorted-column functionality you added previously! That is, sorting by column headers should still work, and if the user then clicks the “Task Title” column header to sort by project task title, the displayed results should both be sorted but do not need to be limited by the checked users (we'll get to that in part 3).
* If the user checks, for example, ‘Coulson’ and ‘Ward’ 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 projects with users assigned).** For now, ignore the case when the user unchecks all checkboxes—you will get to this in the next part.
* Don’t put code in your views! Set up some kind of instance variable in the controller that remembers which users were actually selected 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.

### PART 3: REMEMBER THE SORTING AND FILTERING SETTINGS (70 points)

At this point, the user can now click on the “Task Title” or “Due Date” headings and see project tasks sorted by those columns, and they can additionally use the checkboxes to restrict the listing to certain users with assigned tasks. We have preserved RESTfulness, because the URI itself always contains the parameters that will control sorting and filtering.

The last step is to remember these settings. That is, if the user has selected any combination of column sorting and restrict-by-user constraints, and then the user clicks to see the details of one of the project tasks (for example), when she clicks the Back to Project Task List on the detail page, the project task listing should “remember” the user’s sorting and filtering settings from before.

(Clicking away from the list to see the details of a project task is only one example; the settings should be remembered regardless of what actions the user takes. Any time the user visits the index page, the settings should be correctly reinstated.)

The best way to do the “remembering” will be to use the session[] hash. The session is like the flash[], except that once you set something in the session[] it is remembered “forever” until you nuke the session with session.clear or selectively delete things from it with session.delete(:some\_key). That way, in the index method, you can selectively apply the settings from the session[] even if the incoming URI doesn’t have the appropriate params[] set.

### Hints and caveats

* If the user explicitly includes new sorting/filtering settings in params[], the session should not override them. On the contrary, the new settings should be remembered in the session.
* If a user unchecks all checkboxes, use the settings stored in the session[] hash (it doesn’t make sense for a user to uncheck all the boxes).
* To be RESTful, we want to preserve the property that a URI that results in a sorted/filtered view always contains the corresponding sorting/filtering parameters. Therefore, if you find that the incoming URI is lacking the right params[] and you’re forced to fill them in from the session[], the RESTful thing to do is to redirect\_to the new URI containing the appropriate parameters. 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 project tasks 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.

## Submission

You will submit in 2 ways:

1. Github
   1. All code must be pushed to Github! Double check by going back to your $HOME folder and do a final commit and push.
   2. Make sure your files are actually being added! Go to the github website if you’re worried. If files are missing, do a git add on the missing folders or files and then commit and push again. git status will also show you changes that are staged for commit.
   3. **Your assignment will be graded from your github submission.**
2. Blackboard
   1. As a backup, you are also required to submit your work on Blackboard.
   2. Students should submit a **tar.gz** file containing all of the folders in their app, created by running “tar -czf yourUCCSname\_hw2\_pm.tar.gz \*” **from within the HW3 directory**
   3. Reminder: Name your submission file appropriately! (For example, I would submit: kjustice\_hw2\_pm.tar.gz)
   4. Reminder: When we untar your file, we should immediately see the folders: app, config, etc…. These should not be inside any other folder!