# Final Project

We will be expanding our MVC to make it more robust and less error prone from users. The example URL for the final is located here: http://corsair.cs.iupui.edu:19031/CIT313/SP2015/final/

The admin user: **admin@iupui.edu / letmein**The registered user: **demo@iupui.edu / letmein**

# TURN OFF DEBUG IN YOUR FINAL PROJECT.

You wouldn't deliver a project to a client or employer with the debug messages still turned on. I don't want to see them either. I will deduct fifteen (15) points from each section of the Final (part 1 and part 2 - 30pts total) if your assignment is submitted with debug messages on.

**Part 1 (Due Sunday April 22, 2018 by 11:59p)**

**1: Add some personality to your site.** Use a different bootstrap theme found at <http://bootswatch.com> .Once you find a design you like at the above website, utilize the theme specific files in the bootswatch-2.1.1-1.zip resource attached here / in ‘Files’ **– DUE TO VERSIONING ISSUES DO NOT DOWNLOAD DIRECTLY FROM BOOTSWATCH.** You should only need the CSS file from the package.Place the new css files in views/css. Add some content to the home page describing your blog site. Include an image or other visual.

**2: Create a Comments Table in the database.** Structure should be: CommentID(int, auto increment, key), uID (int), CommentText(longtext), Date(datetime), postID(int). The post ID saved to the database should be the ID of the post being commented on. Use the comments.sql file provided to import into your existing database,

**3: Create a Comments area on each single blog post view**. The comments area should display the comments related to each particular post in chronological order, with the newest first. It should display the comment text, the user’s full name, and the date that the comment was added. **BONUS +5: Load comments via AJAX.**

**4: Only registered users may post comments.** Do not display the comment form if the user is not logged in, display a login button instead. The form only needs to display a text area and a submit button to the user. The post ID and the user ID should be collected in hidden input fields. Saving and deleting comments will require adding a Comments model. All other methods for comments should be handled in the blog controller (and ajax controller if you go for the extra credit).

**5: Administrators should be able to delete comments using a button that appears next to each comment that only administrators may see.**

**6: Provide a mechanism for Administrators to delete and edit posts as well as manage post categories.** Use the manage posts view as an area to edit or delete posts. This area should only be accessible by administrators. It should list each post with a delete, edit, and view button below each post. Create a view and controller that allows you to manage categories. The view should list the current categories with an edit button next to each. Below the list there should be a text field and submit button for adding new categories. Add a link to the manage categories page next to the add post button on the manage posts view. Any activity on this page should display a message telling the administrator that the action was or was not completed successfully.

**Part 2 (Due May 05, 2018 - 5:00PM)**

**~~7: Integrate one web service.~~** ~~Either pull data from a service, or use a service to process a form and retrieve data based on the form submission. Put this service in a sidebar on the home page~~. **BONUS +5: If the service returns a result, such as the weather API, pull in the results via AJAX.**

**8: ~~Modify the registration form to check that all fields are filled out.~~** ~~Display an error message informing the user that not all fields are completed~~. **~~BONUS +5: if you inform the user of the specific field(s) that are incorrect.~~** ~~Add another field asking the potential user to retype their password for verification. Check that the passwords match prior to adding the new user to the database. You may use php or JavaScript to validate~~.

**9: ~~Modify the user table to include a column named ‘active’. The column should be an integer type and not null~~.** ~~When a user initially registers, their active state should be set to ‘0’.~~ ~~Add a hidden field on the registration page that sets this value to 0,~~ or ~~set the column in the database to have a default value of 0~~. ~~Modify the admin users in the database and set this value to ‘1’~~. Create a method in the User model to check if a user is active. The method should accept one parameter, $uID. It should return true if the active column in the user record is ‘1’ and false if it is ‘0’. An example of using this method would look like: $this->userObject->isActive($userInfo['uID']). Use this method to prevent users that aren’t active from logging in. Display a message letting them know that their account is still awaiting approval.

**10: Provide a mechanism for Administrators to delete/approve users.** ~~Create a new view and controller for managing users.~~ ~~Only administrators should have access to this page~~. This page should have a user list, much like the members view. ‘Approve’ or ‘Delete’ buttons should appear next to each user name so that the administrator can approve or delete users. If the user is approved the ‘Approve’ button should disappear. Clicking on the ‘Approve’ button should change the ‘active’ column in the database to ‘1’ for the user that you clicked only. The admin users should not show any buttons since you do not want to inadvertently delete the admin account. Any activity on this page should display a message telling the administrator that the action was or was not completed successfully.

**11: Modify the addpost form to check whether the admin has completed all of the form fields**. ~~Do not allow the data to be saved unti~~l all ~~fields have been completed. Display an~~ error ~~to the admin informing them that the form is incomplete~~. **BONUS +5: ~~Inform the admin exactly which fields~~ still need to be completed. 🡨 textarea / content not working**

**12: Modify the post pages so that in both the list view and the single view, the category is linked.** This link should be to a new method within the blog controller that only displays posts that are of the particular category that was clicked. The url structure should be *yoursite*/blog/category/*categoryID.* You will also need to create new methods in the Post model to only return posts where the categoryID matches a particular value.

**13: Allow users to modify their profiles.** ~~Add a view and controller to manage your user settings~~. ~~This should take the user to a page under the members area that allows them to edit their first name, last name, email, and password. The URL should be /members/profile/~~. Profile should be a method in the members controller as well as a ~~separate view~~. ~~This should look very similar to the Register form~~, except the fields should already be populated with data EXCEPT the password and retype password fields. If the password and retype password fields are left blank, the password in the database should be left unchanged. Only show data for the current logged in user. Remember to hash all passwords before saving to the database. ~~Do not let the user submit if any field (except the passwords) is left blank~~.

**14: ~~Add links in the User’s dropdown menu for ‘My Profile’, ‘Manage Categories’, ‘Manage Posts’, and ‘Manage Users’.~~** ~~Registered users should only see ‘My Profile’ and ‘Logout’. Admins should see all of these options.~~