**Instructions**

No additional system requirements are currently necessary in order to run our web application. This can be run through NetBeans, or any equivalent IDE, using the provided source code. The only requirement is that you must have an active Java Database Connection to the JNDI resource named ‘jdbc/FinalProjectDB.’ If this is done correctly all necessary tables and data structures will be generated automatically for you, the user. With that being said, upon initialization of the application it’s very possible your tables will not contain any data therefore you would, theoretically, be required to register an account before proceeding further into the real views and/or business logic.

**Core requirements**

* JPA entity beans: UserClass, Menu, SubMenu & MenuItem
  + Relationships:
    - UserClass (1 --> \*) Menu
    - Menu (1 --> \*) SubMenu
    - SubMenu (1 --> \*) MenuItem
  + Create
    - “/adduser” controller in DBController.java – creates users when the register controller is called and all conditional statements are met
    - “/savemenu” controller in MenuController.java – persists menus, their submenus and their submenus items, to the applicable tables accordingly once the user submits their menu
  + Update
    - “/editmenu” controller in MenuController.java [/editmenu] – provides the functionality to edit some of the variables associated with their menu, which requires the update of said menu based on the user’s changes
  + Delete
    - “/deletemenu” controller in MenuController.java [/deletemenu] – provides the functionality
  + Find
    - Selection menu in home.jsp [/home] – Uses find to return all menus associated with the sessions current user
* Complex flow:
  + “/deletemenu” controller in MenuController.java [/deletemenu] – redirects user back to the “/home” view if they still have more menus left in the database, otherwise it will redirect to the “/menucreation” view and alert the user that no menus were found
  + “/login” controller in DBController.java [/login] – acts in a similar manner to the “/deletemenu” controller
* Validation:
  + checkPass() function [/register] – Custom validation, through a JavaScript function, that alerts users if their original password input is the same as the confirmation password input. The background color of the confirmation password textbox is then changed accordingly, via onblur()- green indicates a match and red indicates an error.
  + Custom, post form submission, user credentials validation message [/login] – When a user’s login credentials are unsuccessful the POST method controller “/login” will add an attribute “error” to it’s response header and return the user to the login page again. Upon the controller’s redirection, back to the login view, the user will be alerted that they have just entered an invalid username or password.
  + checkUsername() function [/register] – pre-registration username validation using AJAX, see below for more information.
* AJAX
  + checkUsername() function [/register] – As mentioned above, makes use of AJAX and the “/checkusername” controller to alert the user if/when the username they have entered already exists using the JavaScript onkeyup() function
* Good MVC practices:
  + Login redirection [/login]
    - If the user credentials are invalid the “/login” controller will redirect the user back to login and prompt the user with an error a variable set in the response
  + Register redirection [/register]
    - If the user credentials are invalid the “/login” controller will redirect the user back to login and prompt the user with an error a variable set in the response. Additionally, this simple logic prevents multiple database entries being generated with the same username

**Screen shots**