Assignment 2

This assignment is **individual** work. You are not allowed to work on this with anyone, nor should you accept or offer help. Put in the time, solve the problems, and get through it! Know that even looking at how your neighbour or friend solves it will change the way you approach it, and code which is strangely similar stands out like two pages (wikis?) with duplicate paragraphs. Offenders will be prosecuted.

This assignment will likely require less research than assignment 1, but as a trade-off, provides less explicit instruction and requires you to go further on your own. In my opinion, you should be able to figure almost all of it out with the lecture notes and exercises thus far, but you will need to **go above and beyond** the simple requirements here if you wish to do well.

If it's tough, spend time on it! It contains scenarios you are likely to see over and over again.

If it's easy, spend time on it! Show off your skills and creativity! Add a feature or two and make it amazing! Impress me!

Also note: I'm well aware you folks now know a thing or two about testing. Throw in a JUnit test or two if you can. How you do this is up to you, but you're ready to start marrying our course with others you have/have had, and test driven development is always a best practice.

Objective

You are asked by your boss to create a Wiki system which incorporates all the CRUD operations. Your boss's vision for the project is sketchy at best, but they assure you, if you put in the time, solve the problems, and make it work well, they'll be happy. Read this as an opportunity to impress them!

Users should be able to create, retrieve for detailed display, update, and delete Wiki pages. A Wiki, if you recall, is a space open to editing, etc. by everyone – Wikipedia is the biggest and best example of this. It may be moderated by experts, but it allows everyone to create and make edits to every possible page by design.

You choose the information you wish to store. You choose the layout and application flow you want. How this works is entirely up to you. Make it amazing!

Requirements

You must use Spring, JSTL, EL, and Spring form tags to auto-populate forms and POJO where possible.

Wikis and subjects must be stored in a database called hibernatedb using a root user with a password of 1234.

You must use Hibernate and NamedQueries or the Criteria API to save and retrieve information. You cannot use HQL in a DAO.

You must use client-side JavaScript and/or the Hibernate Validator for validation and appearance. Really, it's an easy way to make your pages look great and work well! Simple CSS and JavaScript libraries are basically essential for every webpage today. Why start from scratch?!

Finally, you must add at least one or two other bits of creative functionality. Feel free to mine our course for other features and tech we've covered, though the sky's the limit! I would suggest perhaps JUnit testing of db calls (a test which ensures a write to the database can be retrieved and compared to the original with no corruption) is a good place to start, though you're free to build in anything you like! Explore! What does this site need? What would make it more useful?

Include details on your extra functionality in comments in the HomeController.

Submission Instructions

Name your project A2<YourName>.

Submit a **regular zip** of your full project directory. Please do not submit RARs or 7zips! RARs are just about a guarantee of a virus, and 7zips are awesome for email but a pain with Eclipse.

Grading

View Functionality	5 marks
Model Functionality	5 marks
Controller Functionality	5 marks
Extra Functionality	5 marks

Git 'er done! Best of luck! :-)