# **Building Your First Full-Stack Application**

### **Overview**

This Case Study is your first foray into building a full-stack application. You'll be building a Spring MVC app, which means you'll learn about what it takes to build a functional application from the ground up yourself.

This is exciting! It's a lot, but we've given you the tools to be able build what you need, and you get to decide what you do with it. And you get to be creative in choosing what sort of application you want to build!

You will be working individually for this project, and you'll be designing the app yourself. We hope you'll exercise creativity on this project, sketch some wireframes before you start, make sure you have time to run these ideas by your instructors to get their feedback before you dive too deep into code! Remember to keep things small and focus on mastering the fundamentals – scope creep/feature creep is one of the the biggest pitfalls for any project!

### **Timeline**

First Day:

* Spend the afternoon coming up with ideas and making plans for your app (once you have an idea you are happy with and seems feasible [think about the logic that may be involved] take it and run! There is nothing that will delay the completion of your project as much as constantly restarting with a new concept)

Next Day:

* Meet with instructors to discuss your project
* Start Coding

End of Program:

* Fix last minute bugs
* Short presentation on your project The presentation should include a demo of your app, discussing challenges, and answering questions the class may have.

### **Technical Requirements**

Your app must:

* Have at *least* 4 models in the Oracle database (more if they make sense) – And should have relationship between tables.
* Dynamic Application using the Spring MVC technology
* Use the recommended package structure: separate components into packages by their functionality (Discussed in the JDBC lessons)
* Have complete RESTful routes for your resources with GET, POST
* Utilize JDBC to interact with your stored data in your database
* Use the concepts of DAO, utlis, and Model classes
* Mapper methods using Spring MVC
* Use TomCat as your server
* Include wireframes that you designed during the planning process
* Have semantically clean HTML, JSP and CSS
* Internal and external CSS style sheets
* Your app should have full CRUD functionality
* Your app should have at least 6 different pages
* Use JUnit to perform Unit test cases on your DAO classes

### **Reach goals**

* Include sign up/log in functionality, with encrypted passwords & authorization with bcrypt
* Add additional models, as is appropriate for your app.

### **Necessary Deliverables**

* A working full-stack application, built by you
* Database built on Oracle
* Business logic handle by Spring MVC with the use of Mapper methods
* HTML and JSP must be use for the views
* JDBC must be used to interact with the database
* Junit must be used to create test cases for every DAO class
* Upload your entire project to your github account
* A readme.md file with explanations of the technologies used, the approach taken, installation instructions, unsolved problems, etc.
* User stories in your readme

### **Optional App Ideas**

* Rental WebSite: Rent books, games, bikes
* Retail Store such as Target, Amazon, Ebay
* Blogs such as reddit
* Gallery Website for movies, weddings, events

### **Suggested Ways to Get Started**

* Break the project down into different components (data, presentation, views, style, server-side work) and brainstorm each component individually. Use whiteboards!
* Write Your Pseudocode Start by stating the problems in plain text. This will help you guide your process and understand the problem better.
* Begin with the end in mind. Know where you want to go by planning with wireframes & user stories, so you don't waste time building things you don't need. State what your MVP looks like.
* Don’t hesitate to write throwaway code to solve short term problems
* Read the docs for whatever technologies you use. Most of the time, there is a tutorial that you can follow, but not always, and learning to read documentation is crucial to your success as a developer
* Commit early, commit often. Don’t be afraid to break something because you can always go back in time to a previous version.
* User stories define what a specific type of user wants to accomplish with your application. It's tempting to just make them *to-do lists* for what needs to get done, but if you keep them small & focused on what a user cares about from their perspective, it'll help you know what to build
* Write pseudocode before you write actual code. Thinking through the logic of something helps.

### **Useful Resources**

* [BCript](https://howtodoinjava.com/security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/)
* [Writing Good User Stories](http://www.mariaemerson.com/user-stories/) *(for a few user story tips)*
* [Presenting Information Architecture](http://webstyleguide.com/wsg3/3-information-architecture/4-presenting-information.html) *(for more insight into wireframing)*

### **Project Feedback + Evaluation**

* Project Workflow: Did you complete the user stories, wireframes, and/or task tracking as specified above?
* Technical Requirements: Did you deliver a project that met all the technical requirements? Given what the class has covered so far, did you build something that was reasonably complex?
* Creativity: Did you added a personal spin or creative element into your project submission? Did you deliver something of value to the end user (not just a login button and an index page)?
* Code Quality: Did you follow code style guidance and best practices covered in class? Did you comment?
* Deployment: Did you deploy your application to a public url?
* Total: Your instructors will give you a total score for each category between 0 and 3 and tally the total.

Score:

Low: Incomplete Case Study

Mid-Low: Did not meet all technical requirements

Mid-High: Delivered all technical requirements

High: Delivered all technical requirements, reached goals and Creativity

### **A Note on Plagiarism**

You are encouraged to ask others, including students, instructors, and stackoverflow, for help. However, it is NOT ACCEPTABLE TO COPY another person's code and submit it as your own. More importantly, it is detrimental to your learning and growth.

All of the following are considered plagiarism or cheating:

* Turning in work that is not your own.
* Turning in someone else's work as your own.
* Hiring, or paying someone to do your work for you.
* Copying words or code without giving credit.
* Building or copying someone else’s idea without their knowledge or giving credit.
* Giving incorrect information about a source.
* Changing words, variable names, etc. but copying the code or files of a source without giving credit.
* Copying so many ideas or code blocks from a source that it makes up the majority of your work, whether you give credit or not.
* Failing to put a quotation in quotation marks.

In an effort to not plagiarize, credit for this content goes to:

* [Plagiarism.org](http://plagiarism.org/), specifically the “plagiarism 101” section. Content was adapted for code. For more information, please see:
  + [What is Plagiarism](http://www.plagiarism.org/plagiarism-101/what-is-plagiarism)
    - [Types of Plagiarism](http://www.plagiarism.org/plagiarism-101/types-of-plagiarism)
    - [How do I safely write code in my own 'words' and not plagiarize?](http://programmers.stackexchange.com/questions/80167/how-do-i-safely-write-code-in-my-own-words-and-not-plagiarize)
    - [Avoiding Plagiarism: Writing Computer Code](http://www.upenn.edu/academicintegrity/ai_computercode.html)