PER SCHOLAS

14 Weeks Java Developer

Sample Case Study (Building Full-Stack Application**)**

Overview

This Case Study is your first foray into building a full-stack application. You'll be building a Spring MVC based application, 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 to build what you need, and you get to decide what you do with it. You also get to be creative in choosing what sort of application you want to build!

You will be working individually to design your app. 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 coding! Remember to keep things small and focus on mastering the fundamentals.

Timeline

Week One:

- Spend the afternoon coming up with ideas and making plans for your app. Once you have an idea you are happy with and it seems logically feasible*, take it and run! There is nothing that will delay the completion of your project as much as constantly restarting with a new concept.
- When you test your idea to be logically feasible, think about the processes that may be involved to create it.

Week Two:

- Meet with instructors to discuss your project.

As Skills are learned:

- Start building the app.

Deliverables

- Check with instructors to discuss each of the required deliverables as you tackle them.

End of Program:

- Fix last minute bugs.
- Create a 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 all deliverables completed
- ❑ Have at least 4 models in the RDBMS (more if they make sense) – And should have a relationship between tables.
- ❑ Use Tomcat as your server
- ❑ Include wireframes that you designed during the planning process (optional)
- ❑ Have semantically clean HTML 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
- ❑ Use web services (at least provider or consumer if not both)
- ❑ Be hosted on GitHub with a "readme" file documenting user stories, technical challenges and how you tackle those challenges

Reach goals

- Include sign up/ login functionality, with encrypted passwords & authorization with bcrypt
- Add additional models, as is appropriate for your app

Necessary Deliverables

The following are the seven deliverables you must deliver (The "Best Practices" deliverable should be implemented as you tackle the rest) to complete the case study. For each deliverable, your instructor will give you a deadline you must submit it by. Read carefully the seven deliverables stated below. They give you a clear idea of what needs to be included in each deliverable.

1. Database (MariaDB) – This deliverable includes creating a database that reflects the website you have decided to create

- Schema diagram: a visual representation of your database
- Database file creation (.sql): Queries used to create the database

2. Core Java & JPA - This deliverable includes creating a back-end environment that is composed of different Java classes

- Models: Java classes that represent an entity and are used to transfer data related to an entity, create multiple queries, and represent the database as an object-oriented model
- Persistence.xml: This file configures the Java classes that are going to be using JPA to interact with the database.
- Persistence Java Class: A static class that allows the application to create a persistent object which can be used to interact with the database.
- Service Class or Data Access Objects (DAO): Java classes that are composed of one or more functions and have direct access to the database by using JPA persistent object. Each function in a DAO class interacts with the database differently.
- Custom Exceptions: Java classes that allow you to describe an error while the application is running.
- Utilizes: Java classes that hold constant variables (Variables that never change from its initial value). The value of these variables can be requested parameters, Database queries used in the DAO, name of HTML pages, or URL patterns to forward a request to.

3. HTML5/CSS3 – This deliverable includes creating every page required by the given case study

- HTML5: Use HTML for static and dynamic pages and markup the structure of every page.
- CSS3: Use CSS3 to style your HTML pages and make sure to take into consideration the knowledge acquired from the visual design lessons.

4. Spring MVC - This deliverable includes connecting no. 1, 2 and 3 deliverables to function together

- Spring MVC: Responsible for responding to a request made by the user. This can be login, registration, etc. When using Spring MVC make sure to use at least the following functionalities: different type of session management, annotation-based controller, exception handling, models, model attributes.

5. Junit (Test all DAO classes) - This deliverable includes creating a test class for each DAO available and creating a test case in the test class for each function inside the DAO

- Junit: A Java framework responsible for performing unit testing against every DAO class available. There should be a test class for every DAO and inside the test classes, there should be at least one test case for every function inside the DAO classes. When using JUnit make sure to use the following functionalities: Suite classes, Runner, Feature life cycle, Test, Parameterized classes, Java Hamcrest library.

Best Practices: This deliverable includes enforcing the best programming skills you have learned so far.

- Every class created must start with an uppercase letter and if the name of the class consists of two or more words, then each word must start with an uppercase letter
- Variables and Functions must always follow the camelCase or snake case conventions
- On top of every class and function, provide comments stating the purpose of that class or function
- All packages must have all lower-case letters. If the package's name consists of two or more words, then separate each word with a period ".". For instance, "com.home.insurance.FirstName.LastName.dao". Furthermore, every package must have the following:
    - The name of your case study followed by your full name
    - Keep each package name consistent, only the last word of the package should be different. For instance, "com.home.insurance.firstName.lastName.dao" and "com.home.insurance.firstName.lastName.models"
- Make use of the Eclipse IDE shortcuts. For instance, to correctly indent your code press: "Ctrl + Shift + F". To create getter and setter methods inside a Java class with already private variable members declared. Inside the class right click and select "Source" then "Generate Getters and Setters". A window will be prompted to you, in there you can select the variables you would like to generate methods for and as well as comments. Then click on "OK".
    - The last word of a package should be based on the deliverable itself
- When working with "Dynamic Applications" inside eclipse, inside the "WebContent" folder, create appropriate folders for different file types. For instance, if using CSS stylesheets, create a CSS folder and place all your CSS stylesheets there.
- Create code that can be understood and reused by other programmers.

Optional App Ideas:

- Rental Website: Rent books, games, bikes, parachutes

- 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 the 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 the 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

- BCrypt
- Writing Good User Stories (for a few user story tips)
- Presenting Information Architecture (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?

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, specifically the "plagiarism 101" section. The content was adapted for the code. For more information, please see:
    - What is Plagiarism

- Types of Plagiarism
- How do I safely write code in my own 'words' and not plagiarize?
- Avoiding Plagiarism: Writing Computer Code