Java Developer - Case Study Requirements

Overview

This Case Study is your first foray into building a full-stack application. You will build a **Spring Boot-based** application, which means that you will learn how to implement what it takes to build a functional application from the ground up.

This is exciting! It is a lot, but we will give you the tools to build what you need. You will decide what you do with it. You will also have the opportunity to be creative when choosing what sort of application you want to build!

This is an individual activity. When working on this project and designing your app, be creative. Sketch some wireframes (Mockflow, Lucidcharts - when possible) before you start, and ensure that you have the time to run these ideas by your instructors to get feedback before diving too deep into coding! Remember to follow a modular approach and focus on mastering the fundamentals.

Timeline

Kick-off:

- Spend the afternoon developing ideas and making plans for your app. Once you have an idea that you are happy with, and it seems logically feasible, take it and run! Nothing will delay your project's completion as much as constantly restarting with a new concept.
- When testing your idea for logical feasibility, think about the processes involved in implementing it.
- Meet with the instructors to discuss your project.

As skills are learned:

- Start building the app.

Deliverables:

- Check with the instructors to discuss the requirements (see below) 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, an overview of the challenges you endured, and a question-and-answer period.

Suggested Ways to Get Started

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

Instructions, Requirements, and Rubrics

- **General Instructions:**
    - Application naming format for canvas submission:
        - LastName_FirstName_ProjectName_CaseStudy.
        - Submit your Case Study on Canvas as a **zip file**. Also, please provide the **GitHub repository Website URL**.
        - A complete working application is due by Case Study Submission Day: **11:59 PM** (following the class time zone).
        - Grading will be done through the case study development phase, concluding after the final presentation on Case Study Presentation Day/s. This will be by appointment only.
        - Your instructor will schedule practice presentation sessions before the final presentation during the Case Study Development Phase.

- All learners should present case studies individually on Case Study Presentation Day/s. You will only present your case study to guests and a Talent Advocate Manager. You will not present to the class.

- **Grading:**

1. **Project Structure, Standardization, and Conventions:** ***12%***

   a. The project package structure should be shown in class, where the models, DAO/repositories, services, controllers, and exceptions, etc., have a package. Views or templates do not require a package - *2%.*
   b. Each class should include comments to describe the class and the methods - *2%.*
   c. Have the project pushed into GitHub from the early stage of development and hosted on GitHub with a "readme" file documenting an overview of your project - *5%.*
   d. Standard Java naming conventions should be followed - *3% (please see the breakdown below):*
      i. Classes should be written in Pascal case - *0.6%.*
      ii. Variables, methods, and URLs should be written in the camel case (camelCase) - *0.6%.*
      iii. Files, including view files, should be written in snake case - *0.6%.*
      iv. Packages should be written in lowercase, with each word separated by dots (.) - *0.6%.*
      v. Packages should include the name of your project and your name (e.g., "***org.johndoe.myprojectname"***) - *0.6%.*

2. **Core Java and Models: 16%**
   a. Utilize Java classes with constant variables (i.e., variables that never change from their initial value). The value of these variables can be requested parameters, SQL queries used in the DAO, names of HTML pages, or URL patterns to forward a request to - *2%.*
   b. Have at least four models and corresponding tables in a relational database (if four models/tables do not make sense for your application, discuss this with your instructor) - *12% (please see the breakdown below):*
      i. At least four models - *5%.*
      ii. A corresponding table in a relational database for each model -*3%*.
      iii. Correct implementations - *3%*.
      iv. Optimization - *1%*.
   c. Apply exception handling - *2%.*

3. **Database, ORM, and Hibernate: 18%**
    a. Use MySQL as your DBMS (check with your instructor if you need support to install MySQL on your computer) - *2%.*
    b. Include a schema diagram of the tables and the SQL you used for the database - *2%.*
    c. The database configuration file must be set up correctly in your Spring application through “spring initializr” (application.properties) - *2%.*
    d. Include at least three custom queries - *3%.*
    e. Use Hibernate or Jakarta Persistence API (JPA) directly or through Spring Data JPA - *3%.*
    f. Your application should include examples for all four CRUD operations (Create, Read, Update, and Delete) - *6%.*

4. **Front-end Development: 16%**
    a. Use CSS to style the Web pages. Use an external CSS stylesheet. (Internal styling may be used along with frameworks such as Bootstrap, but you must still include and utilize a custom CSS external file.) - *3%.*
    b. Your application should include six different views/pages - *4%.*
    c. Use HTML to lay out the pages and Thymeleaf to make the pages dynamic. (Frameworks such as Angular or React can also be used but will not be covered in the course. Both Angular or React are optional.) The application’s presentation must meet the general view requirements. - *3%*
    d. Use at least one JavaScript script linked from an external script file. (Internal scripts may be used along with frameworks such as jQuery, but you must still include and utilize a custom JavaScript external file.) - *3%.*
    e. Include a navigation section that is included across multiple pages - *3%.*

5. **Spring Framework: 25%**
    a. Use Spring Boot to develop your project - *2%.*
    b. Models should be annotated for binding using Spring data binding through Jakarta and/or Hibernate validation - *3%.*
    c. Include and implement at least two repositories and two service classes/interfaces - *4%.*
    d. Include at least two ways to create a managed bean/object - *2%.*
    e. Use correct implementations of dependency injection with appropriate use of the @Autowired annotation - *3%.*
    f. Include at least one example of session management (Spring Security can be used for session management) - *3%.*
    g. Use Transaction and request/response logging (write log to a file) - *2%.*
    h. Implement Web Services (JAX-WS, JAX-RS, or Spring REST) - *3%.*
        1. *If your instructor approves of your Case Study idea not requiring the implementation of Web Services, this 3% weight will be evenly re-assigned to items 5.a, 5.b, and 5.c.*
    i. Include sign-up and login functionality with hashing passwords using bcrypt (Spring Security will satisfy this requirement) - *3%.*

6. **Unit Testing: 8%**
   a. Test each query method created in the repositories - *3%.*
   b. Test at least one method in each service class - *3%.*
   c. Include at least one parameterized test - *2%.*

7. **Presentation: 5%**
   a. Create a short overview of your application - *1%.*
   b. Highlight the business use cases of your presentation - *1%.*
   c. Highlight how your application works from the technical perspective (high level) - *1%.*
   d. Highlight what you have learned from this case study development - *1%.*
   e. Discuss additional features that you think could be added in the future - *1%.*

8. **Project Management (Extra Credit): 5%**
   This section is not included in the technical requirements of the Case Study. Completing this section is not required, and not completing this section will not negatively impact your grade. Completing this section could only help boost your overall Case Study grade. This section is only meant for learners who want to go the extra mile and showcase their Project Management skills. Note that you must complete all other requirements to receive credit for this section. This extra credit will not be included if you have already received the maximum 100% grade.

   a. Make use of SDLC/STLC (V-Model) - *1%.*
   b. Perform a Requirements Analysis - *1%.*
   c. Adhere to Agile Principles and the Scrum Framework - *1%.*
   d. Perform stand-up sessions (with an instructor or teammates) and other Agile frameworks, when possible - *1%.*
   e. Successfully track your project using JIRA or Trello - *1%.*

Project Feedback and Evaluation

- Technical Requirements: Did you deliver a project that met all of the technical requirements? Given what the class has covered, did you build something reasonably complex?
- Creativity: Did you add a personal touch/spin or a 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 the code style guidance and exercise best practices? Did you provide any comments?
- Deployment: Did you try to deploy your application to a public URL as a personal stretch goal?

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. Most importantly, it is detrimental to your learning and growth.

Each of the following is 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 codes without giving credit.
- Building or copying someone else's idea without their knowledge or without giving credit.
- Giving incorrect information about a source.
- Changing words and 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 they make up most of your work (whether you give credit or not).
- Failing to put a quotation in quotation marks.

For the "A Note on Plagiarism" section, credit for this content goes to:

- Plagiarism.org - the content was adapted for the code.

For more information, please see:

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