TEKsystems Java Developer - Case Study Submission Requirements

Deliverable: A Java web application developed using Spring Boot

- A complete working application is due the day before the presentation day**,** at midnight.
- Grading will be done through the case study development phase and conclude after the final presentation. This will be done by appointment.
- Your instructor will schedule practice presentation sessions before the final presentation during the case study development phase.
- All learner should present their case studies individually on **case study presentation day.** Fellow learners, instructors, visitors will be in the audience. Your instructor will schedule the presentation slot for you.
- The Case Study Grading rubrics can be found **here (https://docs.google.com/document/d/1VFThg7K-m2AI4gybVy5WVooqddb_P7kvQzjCXmYI7SA/edit?usp=sharing)** .

Technical Requirements:

- Application naming format for canvas submission: LastName_FirstName_ProjectName_CaseStudy
- Use Tomcat as the webserver
- Views:
  - 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)
  - Your application should include six different views/pages (wireframes of the pages should be submitted with the project)
  - Use HTML to layout the pages and Thymeleaf to make the pages dynamic (frameworks such as Angular or React can be used as well but will not be covered in the course. The application's presentation must meet the general view requirements.)
  - Use CSS to style the web pages
  - Use at least one JavaScript script linked from an external script file (internal scripts may be used along with frameworks such a jQuery, but you must still include and utilize a custom JavaScript external file)
  - Include a navigation section that is included across multiple pages
- Models and Database:
  - The configuration file must be set up correctly (e.g., persistence.xml or application.properties)

  - Include at least three custom queries
  - Test each ***custom*** query created in each repository
  - Test at least one method in each of the service classes
  - Include at least one parameterized test
  - Include at least one test suite
  - Use MariaDB as your DBMS
  - Have at least four models along with tables in a relational database (if four models/tables do not make sense for your application, discuss this with your instructor)
  - Include a schema diagram of the tables
  - Use Jakarta Persistence API (JPA) directly or through Spring Data JPA
  - Include and implement repository and service classes/interfaces
  - Models should be annotated for binding using Spring data binding through Jakarta and/or Hibernate validation
  - Your application should include at least one example of each of the CRUD operations
  - Use JUnit to perform unit tests on the JPA repositories/services
- Spring
  - Use Spring Boot to implement the project
  - Include at least two ways of creating a managed bean/object
  - Use correct implementations of dependency injection with appropriate use of the @Autowired annotation
  - Include at least one example of session management (Spring Security can be used for session management)
  - Apply exception handling where required by the code
  - Use of Web Services (when applicable)
- Include sign up and login functionality with encrypted passwords using bcrypt (use of Spring Security will satisfy this requirement)
- The project package structure should be shown in class where the models, DAO/repositories, services, controllers, exceptions, etc., have a package. Views or templates don't require a package.
- Standard Java naming conventions should be followed:
  - Classes should be written in Pascal case
  - Variables, methods, and URLs should be written in camel case
  - Files, including view files, should be written in snake case
  - Packages should be in all lowercase with each word separated by dots (.)
  - Packages should include the name of your project and your name (e.g., "org.johndoe.myprojectname")
- Each class should include comments to describe the class and the methods (see Java - JavaDoc discussion topic in Canvas)
- Have the project hosted on GitHub with a "readme" file documenting user stories and technical

challenges along with how they were resolved.