**Spring Advanced – October 2025  
Individual Project Assignment**

**Overview**

This is the Individual Project Assignment for the [Spring Advanced Course @ SoftUni](https://softuni.bg/trainings/5102/spring-advanced-october-2025).

Your web application must meet the following **technology stack** and **general requirements**.

**Technology Stack**

* **Java version**:17
* **Spring Boot version**: 3.4.0
* **Build tool**:Maven or Gradle
* **Database**: MySQL, PostgreSQL, MariaDB, or Oracle (**relational** **database**)
* **Backend**:Spring framework (you may use **any** Spring module that supports your application logic)
* **Frontend**:
  + **Option 1:** Spring MVC + Thymeleaf
  + **Option 2:** Frontend framework (React, Angular, Vue.js) + Spring RESTful API
* **Source Control**: Git - GitHub, GitLab, or Bitbucket. Links to the public **repository** **required** when submitting your project for the Project Defense. Failure to follow these requirements will result in **DIRECT DISQUALIFICATION** from the Project Defenses.

### **Project Architecture**

Your solution must consist of **at least two independent Spring Boot applications**:

* **Main application** - the core backend system
* **REST microservice** - a separate Spring Boot application, exposing a REST API and consumed by the main application

**Important note:** You may include additional microservices if needed. If you choose to implement a standalone frontend using React, Angular, or Vue.js, your **main application** must expose a RESTful API and is not required to use Thymeleaf. Each Spring Boot application must run independently on its own port.

**General Requirements**

### **Entities, Services, Repositories, and Controllers**

* + The **Main application** must define **at least 3 domain entities**.
  + The **REST microservice** must define **at least 1 domain entity**.
  + Each entity must be part of **at least 1 valid functionality** (see “**Functionalities**”).
  + Every entity must be supported both by:
    - Exactly **1** **JPA Repository**
    - At least **1** **Service class**
  + Define as many **Controllers** (REST/MVC) as needed.

**Important note:** You may have entities with purely **technical purposes** (e.g. logging, roles, tokens, audit, verification), but they **are not counted** towards the required number of domain entities!

* **Web Pages and Front-end Design**This is required only for the Main application.
  + You must define **at least 10 complete web pages**. At least **9** of them must be dynamic. Up to **1** pageout of the 10 required may be static (purely informational). *(If you want to use* ***FE Framework*** *+* ***SPA****, make sure to define at least* ***9 web components*** *the user can interact with)*
  + Ensure a well-designed UI and good UX.
* **REST Microservice** (the one mentioned in the “*Project Architecture”* section and the other sections)
  + Implement a **REST microservice** that communicates with the Main application.
  + Use **Feign Client** for inter-service communication.
  + Follow **properly** the **REST constraints** when designing the API.
  + Define **at least** **2** (**POST**, **PUT**, or **DELETE**) **endpoints** that are being **invoked** **and used** by the Main application.
  + Define **at least 1 GET endpoint** that is being **invoked** **and use**d by the Main application.

**Important note:** If you choose to implement your version of a **notification-svc** (similar to the one demonstrated in the course project), only **up to 50% of the points** for this section will be awarded. To receive full credit, your microservice must contain original logic and serve a domain-specific purpose beyond what was covered in the course.

* **Functionalities**
  + The **Main application** must define **at least 6 valid domain functionalities**.
  + **For Retake:** The **Main application** must define **at least 7 valid domain functionalities**.
  + The **REST microservice** must define **at least 2 valid domain functionalities**.
  + For the **Main application**, a **valid domain functionality** must:
    - Be **triggered by a user** (e.g., filling a form, clicking a button) from the **Frontend**
    - Invoke a backend endpoint (**POST**, **PUT**, or **DELETE**)
    - Cause a **state change** of one or more domain entities
    - Show a **visible result** to the user (e.g., confirmation message, page, UI update)
  + For the **REST Microservice**, a **valid domain functionality** must:
    - Be **triggered by a user** (e.g., filling a form, clicking a button)from the **Frontend**
    - The **Main application** triggers a **Feign Client call** to the **REST microservice** endpoint (**POST**, **PUT**, or **DELETE**)
    - Cause a **state change** of one or more domain entities **in the microservice**
    - Show a **visible result** to the user (e.g., confirmation message, page, UI update)

**Important note: Read-only features** (e.g. viewing details) are allowed, but **do not count** as valid functionalities. Functionalities that operate **only on the User entity** (e.g. login, registration, profile update, role management) **do not count** toward the required number of functionalities.  
  
**Examples of valid functionalities:** creating a booking, editing a product, deleting a comment, processing a payment, buying a flight ticket, changing product visibility, applying for a job, canceling a reservation.

* **Security and Roles**This is required only for the Main application. The REST microservice(s) may implement security, but it is optional.
  + Implement **authentication and authorization** using **Spring Security**.
  + Define **at least two distinct roles** (e.g. *USER*, *ADMIN*) with different access levels.
  + Define a combination of:
    - **Open endpoints** - accessible without authentication
    - **Authenticated endpoints** - accessible to any logged-in user
    - **Authorized endpoints** - accessible only to users with specific role/permission
  + Users with higher privileges (e.g. an admin) must be able to **manage roles or permissions** of other users
  + Authenticated users must be able to **view and edit** their own profiles.
  + If using **Thymeleaf**, the **CSRF protection must be enabled** and not disabled.
  + **For Retake:** Users with higher privileges (e.g. an admin) must be able to **block/unblock** other users from accessing the application.
* **Database**
  + The **Main application** and **REST microservice** must use **separate databases**.
  + Failing to meet the bellow will result in **0 points** for the **Database** section:
    - Use **Spring Data JPA** for database access.
    - Each entity must use a **UUID** as its unique identifier/primary key.
    - Sensitive data (e.g., passwords) must be stored **hashed**.
    - Define **at least 1 entity relationship** in the project.
* **Data Validation and Error Handling**Both the Main application and the REST microservice must implement this correctly.
  + **Accurate** validation must be applied on **all** **layers**: **DTOs**, **entities**, and **service logic**.
  + Return **validation message** for **invalid user input** (e.g. incorrect form field value).
  + Return **meaningful response** for **invalid user operation** (e.g. unauthorized access, missing resources).
  + Implement **at least** **2 error handlers** in each **Spring Boot application**:
    - 1 error handler for a **built-in Spring or Java exception**.
    - 1 error handler for a **custom application exception**.
  + **No application crashes or white-label error pages.** Failing to meet this requirement will result in **0 points** for the **Data Validation & Error Handling** section.
* **Scheduling & Caching**This may be implemented in either the Main application or/and the REST microservice
  + Implement at least **1 scheduled job using a cron expression** that affects the application.
  + Implement at least **1 scheduled job using a trigger different than cron expression** that affects the application.
  + Implement complete **caching** using Spring's caching mechanism.
* **Testing**Both the Main application and the REST microservice must implement this correctly.
  + Implement **at least 1** **Unit test**, **1 Integration test,** and **1 API test**.
  + Ensure a minimum of **80% line coverage**.
  + **For Retake:** Ensure a minimum of **90% line coverage**
* **Logging**Both the Main application and the REST microservice must implement this correctly.
  + Each of the required **valid** functionalities must include **at least 1** log statement.
* **Code Quality and Style**Both the Main application and the REST microservice must implement this correctly
  + No **dead code** (unused methods, variables, or classes)
  + No **unused imports**
  + All **classes**, **methods**, **variables**, and **packages** must follow **Java naming conventions**:
    - Classes: *PascalCase*
    - Methods: *camelCase*
    - Variables: *camelCase*
    - Packages: *lowercasepackagename*
  + **Consistent formatting**: no misaligned blocks, random spacing, or unformatted lines
  + No **comments** or **TODOs** in the submitted code
  + No **business logic** in controllers - follow the **Thin Controller** principle
  + **Properly** follow **layered architecture** (Three-Tier or Feature-Based).
  + Do not make **class non-static fields or methods public** unless you have a really good reason to do so.
  + Write a **README.md** documentation listing the application **tech stack**, supported **features**, **functionalities**, and **integrations** with other systems/applications.
* **Git Commits**
  + **Each** application (Main and REST) must include **at least 5 valid commits**.
  + A valid commit must follow the **Conventional Commits** format: *<type>: description*
  + **Example:** “***feat****: implement flight ticket purchase”,* “***test****: new API tests for IndexController”*
  + Accepted commit types:
    - **feat** - new feature
    - **fix** - bug fix
    - **refactor** - code cleanup/refactor
    - **test** - tests added/updated
    - **docs** - documentation changes
    - **chore** - minor updates (e.g., config)
  + Each valid commit **must match** the code it changes.

### **Recommended Design Practices**

* **Object-Oriented & Design Principles**
  + Follow object-oriented principles: encapsulation, abstraction, inheritance, and polymorphism.
  + Maintain strong cohesion and loose coupling across classes and layers.
  + Reflect on the SOLID principles where applicable - aim to follow the **Single Responsibility Principle** (SRP) consistently, as it's the most accessible and impactful to apply.
* **Code Structure & Readability**
  + Use meaningful class, method, and variable names that clearly represent their purpose in the domain.
  + Avoid **deeply nested methods** and large blocks of logic - keep your code short and focused.
  + Favor **Java Stream API** and functional-style operations when they improve readability.
  + Introduce generic solutions only when they truly reduce duplication or simplify the design.
* **Architecture & Project Organization**
  + Prefer feature-based structure (grouping by business feature) in large applications.
* **Data Modeling**
  + Keep entity relationships simple. Prefer unidirectional relationships, and avoid using *@OneToMany* unless truly required.
  + Let your data model evolve based on functionality and need - avoid overengineering with unnecessary relationships or abstract structures early on.

**Restrictions**

You may reuse **individual techniques and ideas** learned during the course (e.g., validation logic, exception handling, configurations). You are also allowed to **build a project within the same domain** as the course project (e.g., a booking system, e-commerce site, financial platform etc.).

However, it is **strictly forbidden** to:

* **Copy-paste** any significant portion of code from the course project or other course materials (e.g., entire classes, controllers, services, HTML templates, scripts).
* Base your project on the course project skeleton or reuse its structure directly.
* Reuse frontend/backend modules (with or without minor modifications).

Your implementation must be **original** and reflect your own design and development efforts.  
**Plagiarized or reused code will result in DIRECT DISQUALIFICATION.**

**Submission Deadline**

* **Survey for Registration:** A survey for project defense registration will be available from **16 November 2025** to **23 November 2025**.
* **Project Submission Deadline:** Submit your projects no later than **23:59 on 23 November 2025**.
* **Presentation Schedule:** The schedule will be published on **1 December 2025** and will include only the projects submitted on time. Projects not submitted by the deadline will **NOT** be evaluated.
* **Working Period:** You can continue working on your project until **23:59 on 4 December 2025**.

**Online Project Defense**

Each student must deliver an online defense of their work in front of a trainer jury. Students will have 20 minutes to:

1. Demonstrate how the application works (briefly).
2. Show and explain the source code.
3. Answer questions from the jury.

**Note:** Be strict with timing! On the 20th minute, you will be interrupted. Be well-prepared to present the maximum of your work in the minimum time.

**Assessment Criteria**

* **General Requirements (70%)**
  + Entities, Services, and Repositories – [0 - 5]
  + Web Pages and Front-end Design – [0 - 3]
  + REST Microservice – [0 - 8]
  + Functionalities – [0 - 11]
  + Security and Roles – [0 - 6]
  + Database – [0 - 3]
  + Data Validation and Error Handling – [0 - 7]
  + Scheduling and Caching – [0 - 5]
  + Testing – [0 - 8]
  + Logging – [0 - 2]
  + Code Quality and Style – [0 - 10]
  + Git Commits – [0 - 2]
* **Answering Questions (30%)**
  + Answers 3 questions - one from each category: Java, Database, and Spring. Each correct answer gives 10 points, for a total of 30.
* **Bonuses (up to 15%)***Bonuses are optional but reward students who go beyond the assignment requirements. Only fully working, properly integrated features will receive points. Reviewers must confirm the implementation is meaningful, correct, and relevant. Every reviewer will inform you during the defense if you have received any bonus points. If no bonus is mentioned during your defense, but you believe you implemented something eligible, please raise this during your presentation to ensure it is properly evaluated.*
* Use Spring Events in your application – 1 pt
* Implement one or more Advices (AOP) for a **cross-cutting concern** – 2 pts
* Implement HATEOAS – 2 pts
* Use Apache Kafka – 4 pts
* Use a non-relational database (e.g. MongoDB) – 4 pts
* Use Redis for caching – 5 pts
* Use Angular / React / Vue for the frontend – 5 pts
* Host the application in a cloud environment (e.g. AWS) – 3 pts
* Use a cloud file storage API (e.g. Cloudinary, Dropbox) – 2 pts
* Implement permissions in addition to roles – 2 pts
* Integrate with a 3rd Party REST API – 2 pts
* Add Internationalization (i18n) support – 2 pts
* Add functionality to export data to PDF or Excel – 2 pts
* Provide a Dockerized setup for both apps – 1 pt
* Implement Authorization with JWT - 4 pts