
ONLINE TRAINEESHIP APPLICATION

REENGINEERING THE LEGACY CODE

| | |
|--|----------------|
| | Date 10/1/2025 |
|--|----------------|

GOAL OF THE PROJECT

The goal of this project is to reengineer a **Spring Boot Java Web application**. The online traineeship application allows the traineeship committee of the University to manage and monitor open and assigned traineeship positions. Specifically, the application lets companies announce open traineeship positions. The students can look for available traineeship positions. The traineeship committee assigns positions to students via different alternative criteria. The traineeship committee further allocates professors as supervisors to the assigned traineeship positions. Professors and companies will be responsible for the final evaluations of the students' traineeships.

FIRST CONTACT

1. **Read all the code in a single short session.**
 - a. Identify coding styles that will help you to scan and understand the code quickly.
 - b. Identify key abstractions like interfaces and abstract classes that may relate to variation and extensibility points.
 - c. Look for smells and functional tests.
 - d. Look for important comments.
2. **Skim the documentation.** The legacy application has been developed based on a more detailed requirements specification that is available along with the application source code (RequirementsDefinition-2025-v10-02-25.pdf). Study the documentation to get more information concerning the application's requirements. The available documentation also includes a brief document that discusses design, implementation, installation, build and execution issues (**GuidelinesAndHints-06-08-25.pdf**).
3. **Interview during demo.** Observe the system in operation during the demo and question the demonstrator. Note the key use cases the system components and their responsibilities.
4. **Do a mock installation.** Try to install and build the system. The online traineeship application is a **Spring Boot** application developed in **Java (v17)**.
5. **REPORTING:**
 - a. **Code Findings:** After reading the code prepare a short list with all the findings and interesting points that you found out. Write down this list in the report.
 - b. **Documentation Findings:** Write down a list with those parts of the documentation that seem useful and why (e.g., requirement specifications, desired features, important constraints, design

diagrams, user and operator manuals). For each part, an impression of how up to date the description is.

- c. **Use cases:** Specify the typical usage scenarios (use cases).
- d. **Installation Findings:** After the build and install experiment, prepare a list containing, version numbers of libraries, frameworks, database management system, connections, network ports,..... Problems you encountered and how you tried to solve them. In case of incomplete installation or build your assessment of the situation, including possibilities for solutions and workarounds. Put all this information in the report.

INITIAL UNDERSTANDING & DETAILED MODEL CAPTURE

1. **Analyze persistent data.**
 - a. Analyze the database schema of the application and find which classes represent valuable data of the domain model.
 - b. Derive a class diagram that represents the key classes of the domain model and their relations.
2. **Detailed design.** With your understanding of the application and the initial domain model from the previous task, develop more detailed class diagrams that depict the design of the application.
3. **Look for the contracts.** Look for key methods and constructors and refine the class diagrams of the previous tasks with the respective contract definitions.
4. **REPORTING:** Write down the refined class diagrams of the application.

TESTS – YOUR LIFE INSURANCE

1. **Introduce tests** incrementally for **parts** of the application you are working on.
2. Use appropriate **testing frameworks:** JUnit, Mockito and Spring Boot Testing, etc.
3. Write tests for the **key scenarios**.
4. **REPORTING:** Deliver the test code along with the reengineered project implementation.

REENGINEERING TASKS

Refactor the application to fix the following problems:

Redistribute responsibilities:

The **TraineeshipAppController** is a **God class** with **many responsibilities**. The class is responsible for the realization of all the use cases except for the authorization and the authentication. Moreover, the TraineeshipAppController comprises **both end-user request handling logic and business logic**. Specifically, the controller takes input from the views, performs the necessary actions to process the input and the domain model data to produce results and passes the results to the template engine to produce the views returned to the end-user. To deal with these problems we can proceed in the following ways:

1. We can split the **TraineeshipAppController** in separate controllers for the different actors (student, professor, company, traineeship committee), using [Extract Class](#), [Move Method](#) and [Move Field](#).
2. We should separate the business logic from the end-user interaction logic by extracting service classes responsible for the processing of the input and the domain model data and the production of the results. You can consider extracting classes for the different actors (student, professor, company, traineeship committee), using [Extract Class](#), [Move Method](#) and [Move Field](#). Ideally the controllers should just pass the input to the service layer, take the output and return the right views to the end-user.

Duplicate code:

1. The classes that realize the different traineeship search strategies have **duplicate code**. Specifically, the core algorithm is the same in all strategies, which differ only in some specific steps. We can get rid of this problem using the [Form Template Method](#) refactoring.
2. Similarly, the classes that realize the different supervisor assignment strategies have duplicate code. We can get rid of this problem using the [Form Template Method](#) refactoring.

Reporting:

Prepare and add in the report class diagrams that depict the design of the reengineered application.

IMPROVEMENT/EXTENSION TASKS

1. After refactoring the application to its new more maintainable and extensible form implement the following user stories that have been specified in the requirements document and have not been implemented in the initial version of the application: US6, US9, US11, US12, US14, US15.

| | |
|--|----------------|
| | Date 10/1/2025 |
|--|----------------|

2. Extend the application with a new traineeship search strategy and a new supervisor assignment strategy (try to think of sensible criteria that may be useful).
3. Extend the application with a REST API that can be used for the execution of the main use stories. To this end, you will have to develop [REST controllers](#) that will be responsible for the user stories. Ideally, these controllers will rely on the same service layer as the existing MVC controllers.
4. **Reporting:** Prepare and add in the report class diagrams that depict the design of the extended application.

SUBMISSION

The **report** can be based on the given template (Project-Deliverable-Template.doc). For the delivery of the report include a pdf of the report in the project folder. Turn in the **refactored project** and the other **deliverables**.