
Traineeship Application

Guidelines and Hints for the
development of the project

1 Introduction

This document provides some basic guidelines for the development of the project. We begin with general development tips that should be followed. Then, we provide design directions concerning the project.

2 Application Design

To facilitate the maintenance and enable future extensions of the application we assume an architecture that relies on Martin Fowler's catalog of **Enterprise Application Architecture (EAA) patterns** (see <https://martinfowler.com/eaCatalog/>). Maintainability and extensibility are further promoted by the fact that the **Spring Boot framework** heavily relies on the **Model View Controller (MVC) pattern** (see **the lecture slides on Software Design**) for the development of Web applications. MVC allows the clear separation of three different concerns: the **view** of the application that is responsible for the user interaction (UI) with the application, the **domain model** that represents the data handled by the application and the business logic, and the **controller** that takes user input, manipulates the domain model data and updates the view.

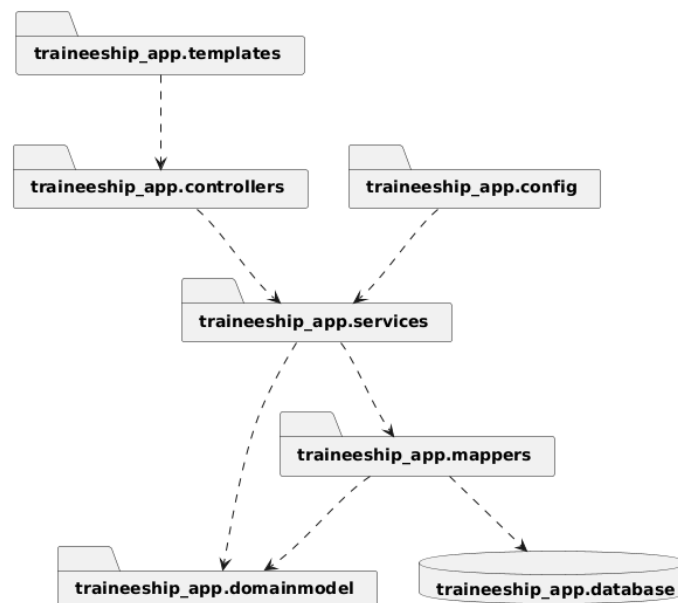


FIGURE 1 APPLICATION ARCHITECTURE.

Figure 1, illustrates a draft architecture for the application. The following list describes briefly the basic components of the architecture:

- The **templates package** realizes the user interaction (UI) with the application in collaboration with the controllers. Typically, this layer consists of a set of static and dynamic Web pages. The

dynamic Web pages are also known as template views (see **Template View pattern** from Martin Fowler's catalog of EAA patterns). A dynamic HTML page renders data from domain model objects into HTML based on embedded markers. The application controllers are responsible for passing the appropriate domain model objects to the view layer. We can think of views as html templates consisting of html code and “special” expressions (`{...}`) for accessing domain model data, performing calculations, and controlling the display of content. These expressions are processed by the so-called view **template engine** that executes the “special expressions” and consequently incorporates the results in the html code and produces a complete html page, which is visualized in the end user Web browser. In general, there are several alternative template engines that can be used to develop the front end of a Spring application. Our application is based on [Thymeleaf](#).

- The **controllers package** consists of controller classes which take user input from the views of the application, perform certain **operations** to manipulate domain model objects, and update the views of the application. The **controllers package** comprises two different controllers. The **AuthController** is responsible for the user registration and login. The **TraineeshipAppController** is responsible for the remaining user stories. The dynamic html views contained in the **templates package** are based on the **Thymeleaf** template engine.
- Currently, the **services package** is responsible for user authentication and authorization via respective registration and login processes. These processes are designed and implemented with respect to the [Spring Security framework](#) that is part of the overall Spring Boot framework.
- The **mappers package** is responsible for basic database operations which map the domain model data, stored in the database tables to corresponding in memory objects of the domain model classes. Basically, here we apply the **Data Mapper pattern** from **Martin Fowler's catalog of EAA patterns**. Moreover, the implementation of this package is designed and implemented with respect to the [Spring Data JPA](#).
- The backbone of the architecture is the **domain model package**. The domain model consists of the basic classes that define the representation of the data handled by the application and the domain logic that is needed for the data manipulation. All the different layers of the application rely on the data model. User is a specific class we must implement for the realization of the user registration and login actions in **Spring Boot**. User should implement the Spring UserDetails interface for this reason ¹. In the application we have different kinds of users (students, companies, professors, committee members) that we distinguish using the Role enumeration. The remaining classes correspond to the key concepts of the application domain, i.e. Company, Student, Professor, TraineeshipPosition. Company is a class of objects which hold information about companies which offer traineeship positions. TraineeshipsPosition is a class of objects which hold information about traineeship positions; some of the positions are assigned to students, while others are still open. Student is a class of objects which correspond to students looking for traineeship positions. Professor is a class of objects which represent professors who supervise traineeship positions. Evaluation is a class of objects which correspond to the

¹ https://github.com/zarras/myy803_springboot_web_app_tutorials/tree/master/sb_tutorial_7_signup_signin

evaluations of assigned traineeship positions, issued by the supervising professors and the companies that offer the assigned traineeship positions. The relation between Company and TraineeshipsPosition is one to many as a company may offer many positions. The relation between Professor and TraineeshipsPosition is also one to many because a professor may supervise several positions. On the other hand, the relation between Student and TraineeshipsPosition is one to one. At this stage of the design, it is not clear whether some of the relations should be bidirectional.

- The **database** schema of the application defines corresponding tables for the classes of the application with foreign keys that correspond to the class associations. In Spring Boot the SQL schema can be automatically generated from the domain model with the appropriate JPA annotations ².
- The classes in the **configuration package** are responsible for the configuration of the Spring Boot framework and the configuration of the Spring Security framework.

3 Installation, Build and Execution of the Application

The project is a **Spring Boot** application developed in **Java (v17)**.

1. The source code is packaged as a **Maven project**. Maven is an automated application management and build tool. Maven uses an XML file called **pom.xml** (Project Object Model) to manage the project's configuration, dependencies, plugins, and more. The project dependencies (`<dependency> </dependency>`) specify required APIs which are automatically downloaded by Maven and linked with the project. As is typically done in Spring Boot applications, the initial version of the project has been created using the **Spring Boot initializr** (<https://start.spring.io/>) tool that creates the project structure and pom.xml. The automatic build process of the project results in a jar file that contains an embedded application server (**Tomcat** in particular) that is used for the execution of the project (hence there is no need to install Tomcat to run the application). The application server is responsible for listening and handling the http requests that are targeted to the application. Embedding the application server in the final jar file is a feature that has been specified when the project has been created using the initializr tool.
2. To install the application, you must import it as an existing Maven project in the Eclipse IDE (using other IDEs is also possible via similar steps). From the **Eclipse** File menu go to **File>Import>Maven>Import Existing Maven Projects** and select to project folder

² https://github.com/zarras/myy803_springboot_web_app_tutorials, https://github.com/zarras/myy803_springboot_jpa_tutorials, <https://www.baeldung.com/jpa-many-to-many>

(`traineeship_app`) that is provided to you as part of the project material. The installation in a different IDE like IntelliJ may be slightly different.

3. The project relies on a **MySQL** database for storing and retrieving data such as the user profiles, book offers, book requests and so on. The sql script that defines the relational schema that is assumed for the project data is located in `traineeship_app/myy803_traineeship_app-static.sql`. Therefore, to execute the project you need a **MySQL installation**. Then, you have to create the database by executing the aforementioned script. Finally, you **must configure the connection to the database**. To this end, you must modify the `traineeship_app/src/main/resources/application.properties` file. Specifically, you must modify the values of the **JDBC properties** in this file with appropriate values that correspond to the username, password of your database connection and the URI of the database that you created.
4. To execute the application, you can simply right click on the **TraineeshipAppApplication** class and select **Run As>Java Application**. In its current state the application is configured to run on the localhost:8080. So, when the application context is set and the application up and running you can start using the application via a web browser by visiting the localhost:8080 URI.

4 TODO STUFF

4.1 Tests

The test base of the application is EMPTY in this preliminary version. Ideally, we need to develop tests for the different packages of the application (mappers, services, controllers, model). For the development of the tests we can rely on the **SpringBoot testing and mocking facilities** (for some examples see this [git repository](#)).

4.2 Smells and Refactoring

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**.

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. Similarly, the classes that realize the different supervisor assignment strategies have duplicate code.

4.3 Unimplemented User Stories

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.

5 APPENDIX Spring Boot Annotations

Spring Boot and its sub-frameworks (Security, Data JPA, etc.) heavily rely on annotations for the specification of the different application components and the binding between these components. A quick guide of the main annotations that can help understanding the source code of the application can be found [here](#).