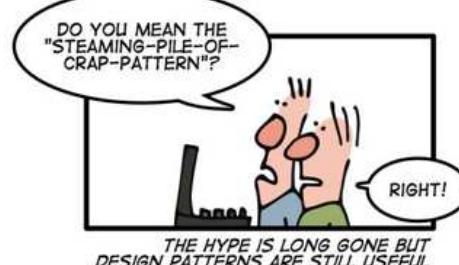


Traineeship Application

Guidelines and Hints for the development of the project



THE HYPE IS LONG GONE BUT
DESIGN PATTERNS ARE STILL USEFUL

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 General Guidelines - DOs and DON'Ts

- Classes
 - ✓ Make **classes small** and cohesive - A single well-defined responsibility for a class
 - ✓ Don't break **encapsulation** by making the data representation public
 - ✓ **Class names** are important – use descriptive names for the concepts represented by the classes
 - ✓ Use **Noun & Noun phrases** for class names
 - ✓ See here for more - <http://www.cs.uoi.gr/~zarras/soft-devII.htm>
- Methods
 - ✓ Make **methods small** – A method must **do one thing**
 - ✓ **Method names** are important – use descriptive names for the concepts represented by the methods
 - ✓ Use **Verb & Verb phrases** for method names
 - ✓ See here for more - <http://www.cs.uoi.gr/~zarras/soft-devII.htm>
- Fields
 - ✓ Make **fields private** – A method must do one thing
 - ✓ **Field names** are important – use descriptive names for the concepts represented by the fields
 - ✓ Use **Noun & Noun phrases** for field names
- Follow the standard Java Coding Style

<https://www.oracle.com/java/technologies/javase/codeconventions-contents.html>

3 Application Design and Related Design Patterns

3.1 Architecture

IMPORTANT NOTICE: the design that is given here is a **draft/incomplete** version of a traineeship application prototype that relies on the Spring Boot framework. You can use it as a **starting point** and follow it to a certain degree. **Feel free to adapt** what is described in this document to your needs and vision.

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/eaaCatalog/>). 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.

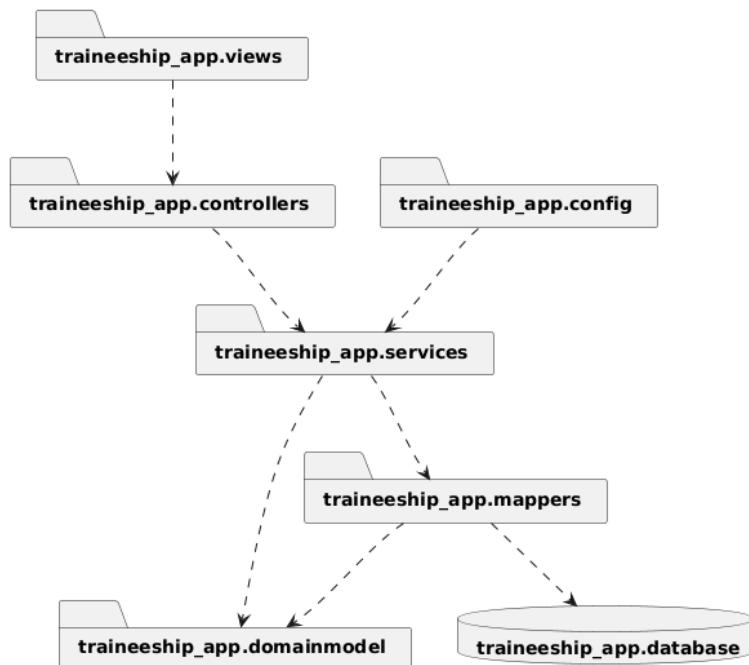


FIGURE 1 APPLICATION ARCHITECTURE.

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

- The **views package** realizes the user interaction (UI) with the application in collaboration with the Controller layer. 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.
- The **MVC controllers package** consists of controller classes which take user input from the views of the application, perform certain **service operations** (see next) to manipulate domain model objects, and update the views of the application.
- The **services package** defines a set of services that decouple the business logic of the application from the front-end of the application (controllers and views). Decoupling the business logic from the front-end makes replacing the front-end with another or adding a new front-end easier (e.g. HTML + MVC controllers with React + Rest controllers). Essentially, here we apply the **Service Layer pattern** from Martin Fowler's catalog of EAA patterns.
- The **mappers package** consists of several interface definitions derived from the general JpaRepository interface that is provided by Spring Boot. These interfaces allow us to perform basic database operations which map the application data stored in the database to in-memory objects of the domain model classes. Basically, here we apply the **Data Mapper pattern** from Martin Fowler's catalog of EAA patterns.
- 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.

3.2 Domain Model

The **domain model** that we assume in this draft prototype is given in Figure 2. 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 reason¹. In the application we have different kinds of users (students, companies, professors, committee members) that we distinguish using the Role enumeration.

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

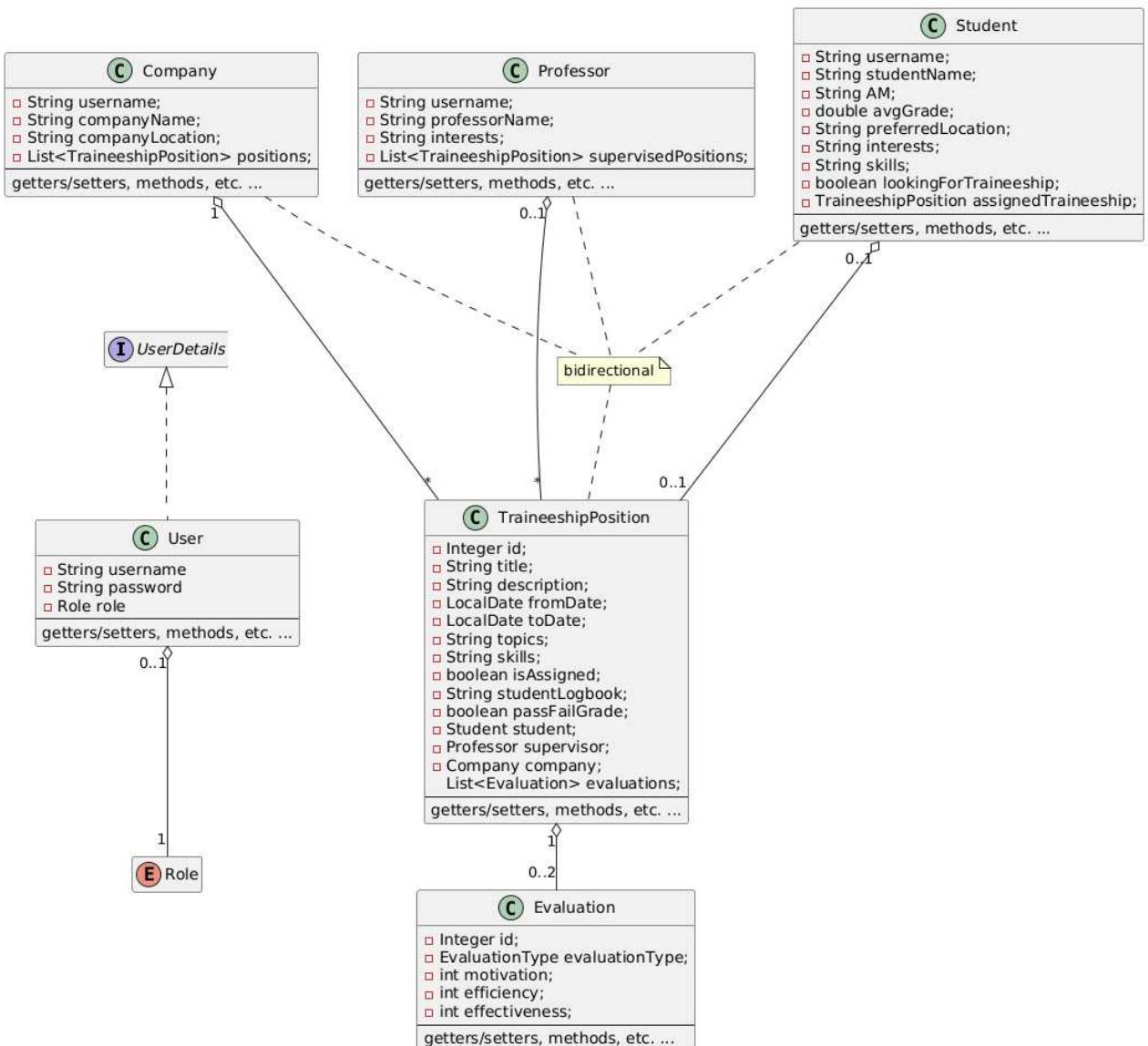


FIGURE 2 DOMAIN MODEL.

The remaining classes correspond to the key concepts of the application domain, i.e. Company, Student, Professor, TraineeshipPosition and Evaluation. 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 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.

3.3 Services

The **services package** comprises the interface definitions of 5 services and the respective implementation classes. More specifically, we have a service that is responsible for the users registration, login and logout (Figure 3). The user service provides the `UserService` interface that is implemented by the `UserServiceImpl` class. The service further implements the `UserDetailsService` interface, as required by the Spring Security framework.

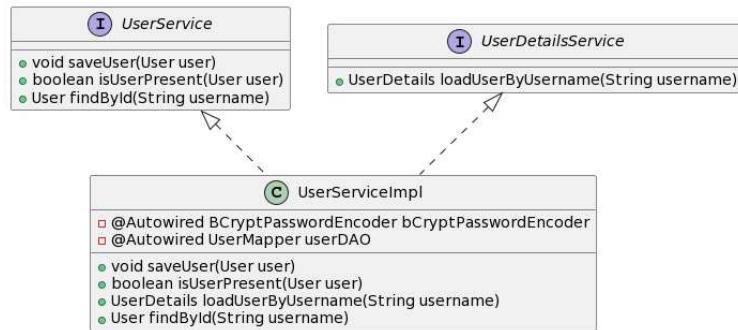


FIGURE 3 USER SERVICE.

The company service (`CompanyService` interface, implemented by the `CompanyServiceImpl` class) is responsible for the management of company profiles, the creation of available traineeship positions and the evaluation of assigned traineeship positions (Figure 4).

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>

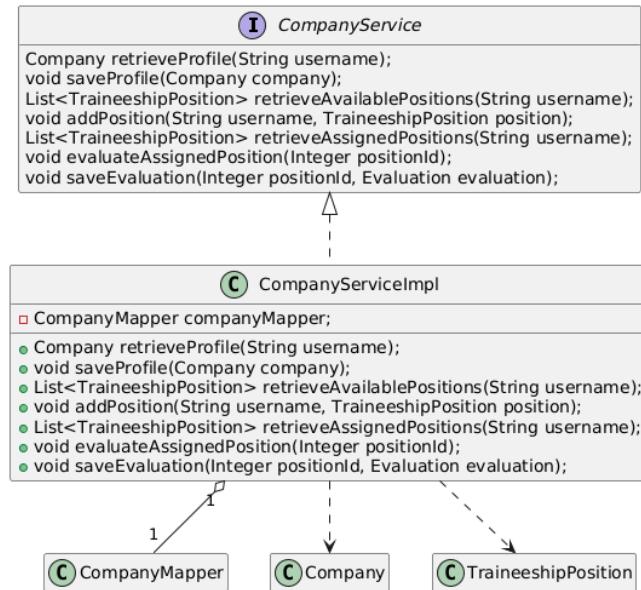


FIGURE 4 COMPANY SERVICE.

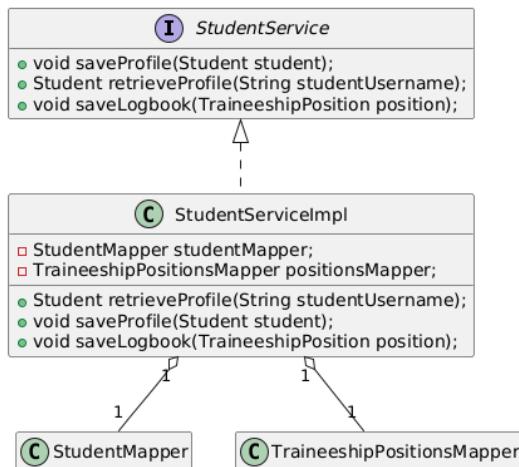


FIGURE 5 STUDENT SERVICE.

The student service (**StudentService** interface, implemented by **StudentServiceImpl**) is responsible for the management of student profiles and traineeship logbooks (Figure 5). The professor service (**ProfessorService** interface, implemented by the **ProfessorServiceImpl** class) is responsible for the management of professor profiles and the evaluation of supervised traineeships positions (Figure 6).

Finally, the committee service (**CommitteeService** interface, implemented by the **CommitteeServiceImpl**) is responsible for the assignment of traineeship positions to students, the assignment of supervisors to traineeship positions and the final evaluation of completed traineeship positions (Figure 7).

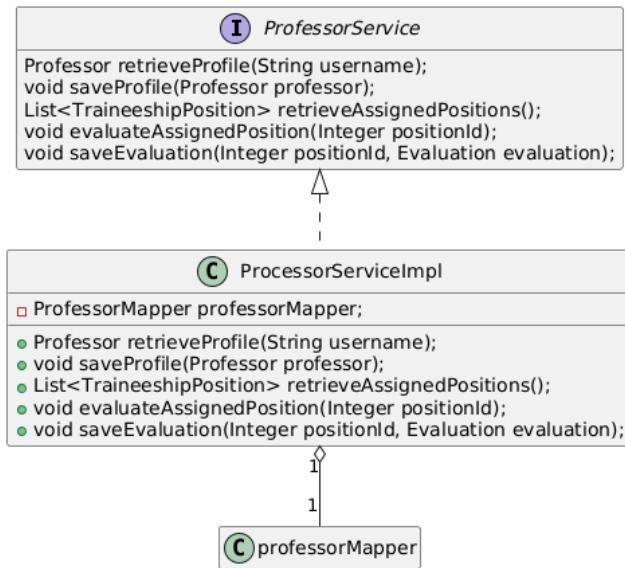


FIGURE 6 PROFESSOR SERVICE.

The assignment of positions to students is realized via alternative strategies that account for the students' interests, preferred location or both. To make the alternative strategies interchangeable and facilitate the extension of the application with more strategies, we rely on the **GoF strategy pattern**, i.e. the position assignment strategies are implemented in different classes that provide the same interface. The committee service creates the appropriate strategy using a parameterized factory.

The assignment of professors to traineeship positions is also realized via alternative strategies that account for the professors' interests or load (Figure 8). As before, to make the alternative strategies interchangeable and facilitate the extension of the application with more strategies, we rely on the **GoF strategy pattern**, i.e. the supervisor assignment strategies are implemented in different classes that provide the same interface. The committee service creates the appropriate strategy using a parameterized factory.

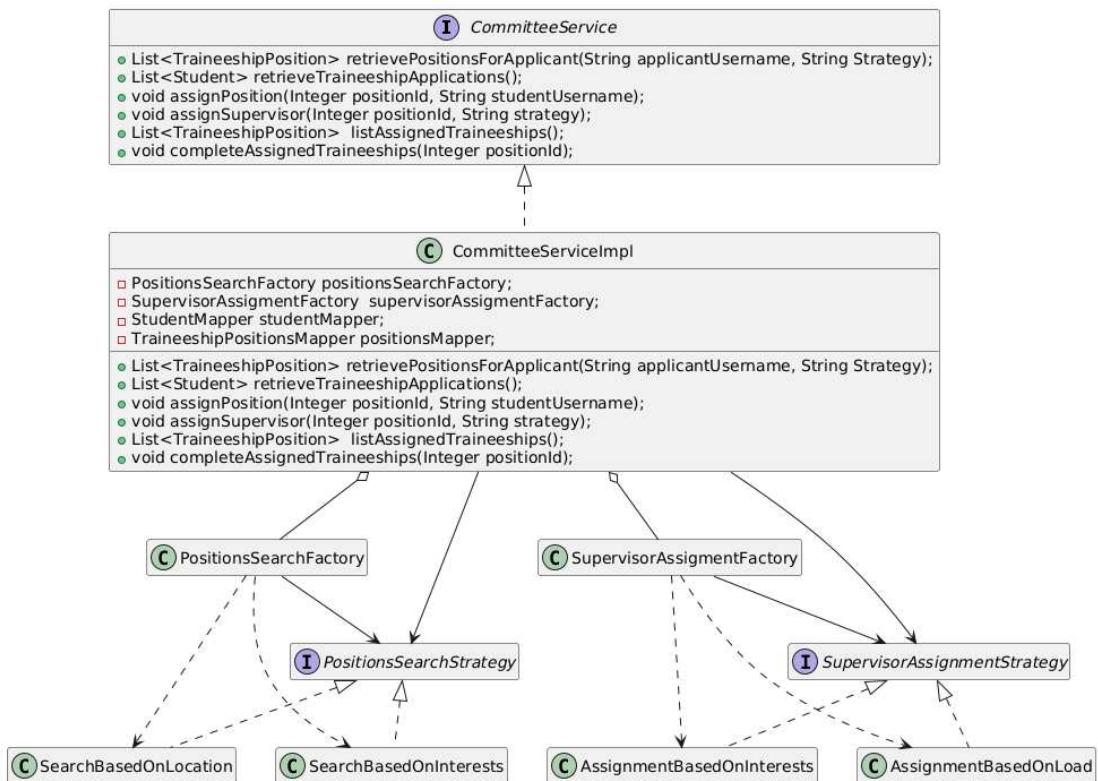


FIGURE 7 COMMITTEE SERVICE.

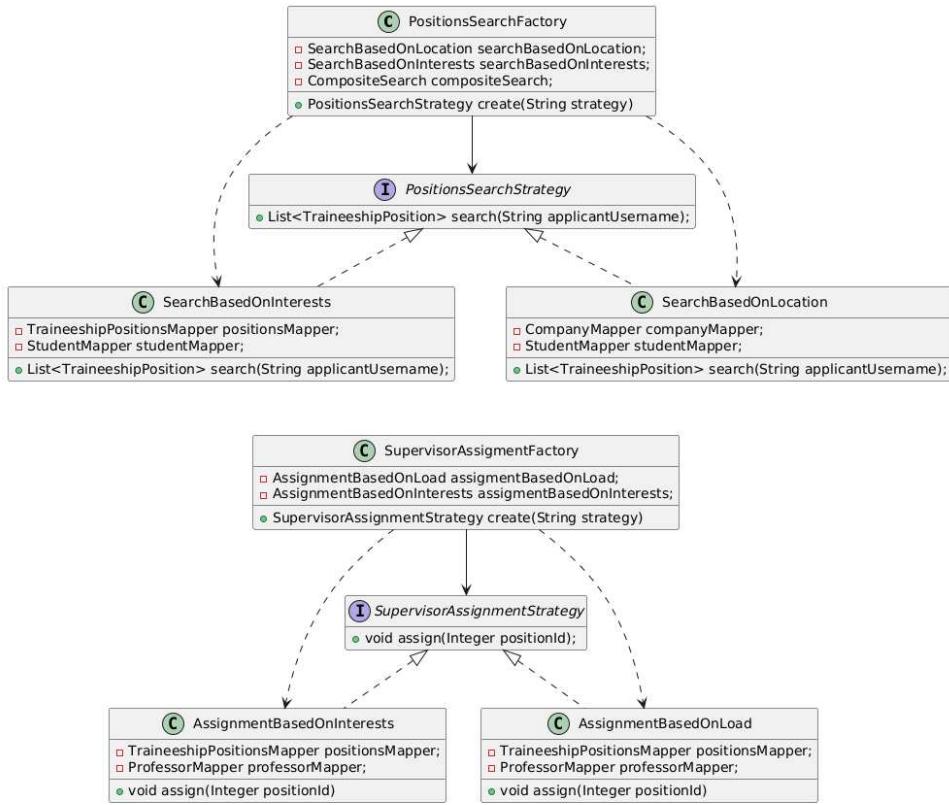


FIGURE 8 STRATEGIES AND PARAMETERIZED FACTORIES.

3.4 MVC- Controllers and Views

The controllers package comprises six different controllers (Figure 9). The `AuthController` is responsible for the user registration and login. `CompanyController`, `StudentController`, `ProfessorController`, and `CommitteeController` are responsible for the interaction with companies, students, professors and committee members, respectively. The controllers rely on the respective services of the services package for the manipulation of domain objects and the update of the views. The views are based on the Thymeleaf template engine.

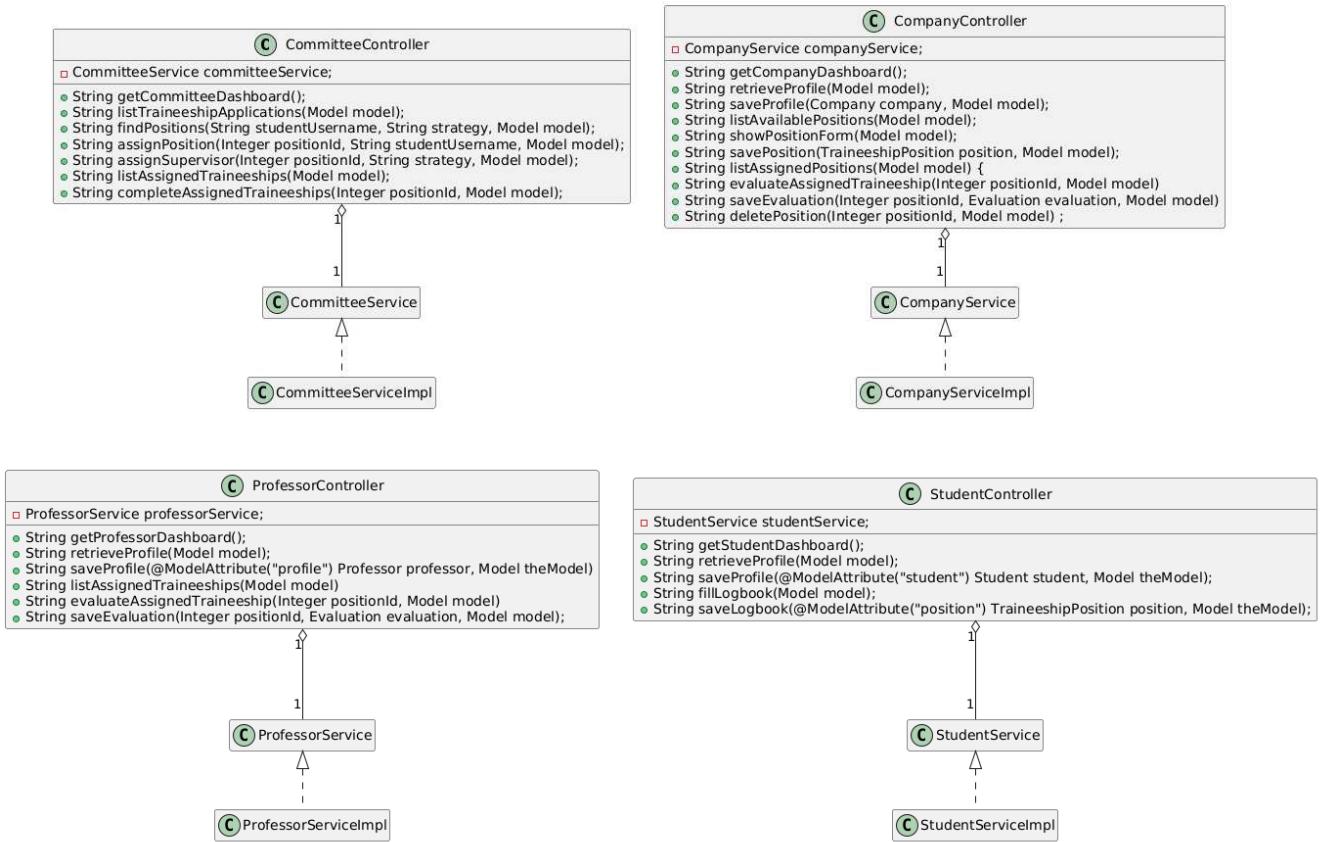


FIGURE 9 CONTROLLERS

4 Tests

Concerning the tests of the application logic 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.