Traineeship Application

Sprint Report

Team Name

ADAMANTIOS DIMITRIOS KALLIS 4685

AGGELOS MPIRINTZIS 4741

PANAGIOTIS ZOUMPAS 4873

1 Contents

VERSIONS HISTORY

Date	Version	Description	Author
21/03/25	1	Initial start at the report	
23/03/25	2	The documented use cases correspond to the functionalities already implemented in the code.(Login, Student, Company)	
31/03/25	3	Finishing the Use cases for professors and traineeship committee	

Το πρόγραμμα αναπτύχθηκε και εκτελέστηκε στο περιβάλλον:

- **IDE**: IntelliJ IDEA 2024.3.4.1
- **Γλώσσα προγραμματισμού**: Java 23
- **Build Tool**: Apache Maven
- **Βάση Δεδομένων**: MySQL (όνομα βάσης: traineeship_db)
- **Framework**: Spring Boot 3.4.3 (μέσω του spring-boot-starter-parent)

Introduction

This document provides information concerning the **<X>** sprint of the project.

1.1 Purpose

1.2 Document Structure

The rest of this document is structured as follows. Section 2 describes out Scrum team and specifies the this Sprint's backlog. Section 3 specifies the main design concepts for this release of the project.

2 Scrum team and Sprint Backlog

2.1 Scrum team

Product Owner	ALL THREE
Scrum Master	....
Development Team	....

2.2 Sprints

<List below the sprints that you performed and the user stories that have been realized in each Sprint>

Sprint No	Begin Date	End Date	Number of weeks	User stories
1	**11/03/25**	**14/03/25**		**US1 - US3**
2	**15/03/25**	**22/03/25**		**US7 - US12**
3	**23/03/25**	**25/03/25**		**US4 - US6**
4	**25/03/25**	**31/03/25**		**US13 – US21**

3 Use Cases

<Specify the concrete Use Cases that describe the interaction of the user with the applications, as derived from the abstract user stories. Give a **UML Use Case diagram** and the **detailed use case descriptions**.>

3.1<Use Case 1>

Use case ID	US1
Actors	User
Pre conditions	The user is not currently logged in.
Main flow of events	1. The use case starts when the user selects the **"Register"** button 2. The System displays the registration form. 3. The user fills in the necessary information for his account(e.g. unique username,password,role) 4. The use case ends when the user selects the **"Save"** button. 5. The system validates the input. 6. If all data is valid and the username is unique, the system creates a new user account. 7. The user is redirected to the login page.
Alternative flow 1	At step 5, if the submitted username already exists, the system displays an error message: *"A user with that username already exists."* The user remains on the registration form to try again.
Alternative flow 2	At step 5, if any required fields are empty or invalid, the system displays appropriate validation messages. The user must correct the form before resubmitting.
Post conditions	A new user account has been created and stored in the system.

3.2<Use Case 2>

Use case ID	US2
Actors	User
Pre conditions	The user has previously registered an account.
Main flow of events	1. The use case starts when the user selects the **"Login"** button. 2. The system verifies the users credentials. 3. If the credentials are correct the system authenticates the user. 4. Based on the users role, the system redirects the user to the appropriate dashboard.
Alternative flow 1	At step 2, if the credentials do not match any registered user: • The system displays an error message: "Invalid username or password. Please try again.." • The user remains on the login page.
Post conditions	The user is authenticated and logged into the system. The user is redirected to their role-specific dashboard or home page.

3.3<Use Case 3>

Use case ID	US3
Actors	User
Pre conditions	The user has already been logged in. The system displays a page or menu where the "Logout" button is accessible.
Main flow of events	1. The use case starts when the user selects the **"Logout"** button. 2. The system terminates the user's session. 3. The user is redirected to the homepage or login screen.
Post conditions	The user is no longer authenticated. The user is redirected to the homepage.

3.4<Use Case 4>

Use case ID	US4
Actors	User(Student)
Pre conditions	The user is logged in as a Student.
Main flow of events	1. The student logs in using valid credentials. 2. The system checks if the student has an existing student profile. 3. **If a profile exists**, the user is redirected to the student dashboard. 4. **If no profile exists**, the user is redirected to the "Create Student Profile" page. 5. The student fills in personal and academic details. 6. The student selects the "Save" button. 7. The system stores the student profile. 8. The user is redirected to the student dashboard.
Alternative flow 1	• At step 6, if the student tries to save the form without filling all required fields: ○ The system displays validation messages. ○ The user remains on the form to complete it.
Alternative flow 2	• At any time after reaching the dashboard, the student may select the "Edit Profile" button. • The system displays the student profile in editable form. • The student updates the desired fields and selects "Save". • The system updates the profile and reloads the dashboard.
Post conditions	If the student had no profile, a new one is created. If editing, the profile is updated. The student is always redirected to their dashboard after the process.

3.5<Use Case 5>

Use case ID	US5
Actors	User(Student)
Pre conditions	The user is logged in. The user has already created their student profile. The user has not yet applied for a traineeship position.
Main flow of events	1. The user navigates to their dashboard and selects the **"Apply for Traineeship"** button. 2. The system redirects the user to the traineeship application page. 3. The user clicks the **"Apply for Traineeship"** button to submit the application. 4. The system stores the application.
Alternative flow 1	• At step 2, if the system detects that the student has already applied or been assigned to a traineeship: The system notifies the student that an application has already been created.
Post conditions	The student has successfully applied for a traineeship position.

3.1 <Use Case 6>

Use case ID	US6
Actors	User(Student)
Pre conditions	The user is logged in. The user has completed their student profile. The user has already been assigned a traineeship position.
Main flow of events	1. The user selects the **"Logbook"** button from the dashboard or navigation menu. 2. The system redirects the user to the **Logbook page**. 3. The system displays the existing logbook entries (if any) for editing. 4. The user edits or enters new logbook content (e.g., daily tasks, observations). 5. The user selects the **"Save"** button. 6. The system validates and saves the updated logbook entry. 7. The user is redirected back to the dashboard or stays on the logbook page.
Alternative flow 1	At step 4, instead of saving, the user selects the **"Cancel"** button. The system discards any unsaved changes. The user is redirected back to the dashboard or previous page.
Post conditions	The user's logbook has been updated with the latest content. The saved logbook entry is stored and associated with the student's assigned position.

3.7 <Use Case 7>

Use case ID	US7
Actors	User(Company)
Pre conditions	The user has the role of "Company".
Main flow of events	1. The company logs in using valid credentials. 2. The system checks if the company has an existing student profile. 3. **If a profile exists**, the user is redirected to the company dashboard. 4. **If no profile exists**, the user is redirected to the "Create Company Profile" page. 5. The user fills in the necessary company information. 6. The user selects the "Save" button. 7. The system stores the company profile. 8. The user is redirected to the student dashboard.
Alternative flow 1	At step 6, if the user attempts to save the form without filling all required fields: The system displays appropriate validation messages (e.g., "Company name is required"). The user remains on the profile form to correct the input.
Alternative flow 2	At step 4, if a company profile already exists: The user is redirected directly to the **company dashboard**. The user can later choose to edit the profile via an "Edit Profile" option.
Post conditions	A company profile has been successfully created and linked to the user's account. The user is redirected to the company dashboard.

3.8 and 3.9 <Use Case 8-9>

Use case ID	US8&US9
Actors	User(Company)
Pre conditions	The user is logged in. The user has the role of "Company".
Main flow of events	1. he user selects the **"Positions"** button from the company dashboard. 2. The system redirects the user to the **positions management page**. 3. The system displays two checkboxes: 4. **Show Open Positions** 5. **Show Assigned Positions** 6. By default, both filters may be selected. 7. The user checks or unchecks each box to show/hide: 8. Open (unassigned) positions 9. Assigned positions (positions that already have a student matched) 10. The list dynamically updates to reflect the filter selections. 11. The user reviews the filtered list of positions. 12. The user selects the **"Back to Dashboard"** button to return.
Post conditions	The user has viewed and optionally filtered the list of their advertised positions. The interface remains interactive for further navigation (e.g., view, edit, or create new positions).

3.10 <Use Case 10>

Use case ID	US10
Actors	User(Company)
Pre conditions	The user is logged in. The user has the "Company" role. The user is currently on the **positions page**.
Main flow of events	1. The user selects the **"Add Position"** button on the positions page. 2. The system redirects the user to the **"Create Position"** form. 3. The user fills in the required fields for the traineeship position, such as: (Title, Description, Duration, Location, Skills required, Start date / end date) 4. The user selects the **"Save"** button. 5. The system validates the data and creates the new position. 6. The new position is added to the company's list of positions.
Alternative flow 1	At step 4, if the user submits the form without completing required fields or with invalid input: • The system highlights the missing/invalid fields and shows error messages. • The user remains on the form page until all issues are corrected.
Post conditions	A new traineeship position has been successfully created and linked to the company's account. The position is now visible on the **positions page** (and optionally in the list of open positions).

3.11<Use Case 11>

Use case ID	US11
Actors	User(Company)
Pre conditions	The user is logged in. The user has the "Company" role. The user is on the **positions page** and can view their own advertised positions.
Main flow of events	1. The user locates a position they wish to remove. 2. The user selects the **"Delete"** button associated with that position. 3. The system deletes the position from the database. 4. The position is removed from the list.
Post conditions	The selected traineeship position has been permanently removed and is no longer visible or available to students.

3.12<Use Case 12>

Use case ID	US12
Actors	User(Company)
Pre conditions	The user is logged in. The user has the role of "Company". The user is viewing a traineeship position that has already been assigned to a student.
Main flow of events	1. The user navigates to their positions page and selects the **"Evaluate"** button next to an assigned position. 2. The system redirects the user to the **evaluation form page**. 3. The user fills in the evaluation fields: **(Motivation, Efficiency, Effectiveness)** 4. The user selects the **"Save"** button to submit the evaluation. 5. The system saves the evaluation and associates it with the student and position.
Alternative flow 1	At step 1, if the user attempts to select **"Evaluate"** for a position that has not been assigned to any student: The system blocks the action. A message is shown: "You cannot evaluate assigned positions!" The user is returned to the positions list.
Post Conditions	The evaluation is successfully recorded and stored in the system. The evaluated data is now associated with the assigned student and the traineeship position.

3.13<Use Case 13>

Use case ID	US13
Actors	User(Professor)
Pre conditions	The user has the role of "Professor".
Main flow of events	8. The professor logs in using valid credentials. 9. The system checks if the professor has an existing professor profile. 10. **If a profile exists**, the user is redirected to the professor dashboard. 11. **If no profile exists**, the user is redirected to the "Create Professor Profile" page. 12. The user fills in the necessary personal information. 13. The user selects the "Save" button. 14. The system stores the professor profile. 8. The user is redirected to the professor dashboard.
Alternative flow 1	At step 6, if the user attempts to save the form without filling all required fields: The system displays appropriate validation messages (e.g., "Professor name is required"). The user remains on the profile form to correct the input.
Alternative flow 2	At step 4, if a professor profile already exists: The user is redirected directly to the **professor dashboard**. The user can later choose to edit the profile via an "Edit Profile" option.
Post Conditions	A professor profile has been successfully created and linked to the user's account. The user is redirected to the company dashboard.

3.14<Use Case 14>

Use case ID	US14
Actors	User(Professor)
Pre conditions	The user is logged in. The user has the role of "Professor".
Main flow of events	1. The professor selects the **"Positions"** button from the dashboard. 2. The system retrieves all traineeship positions where the professor is assigned as a supervisor. 3. The system displays a list of these positions, showing key details (e.g., title, student assigned, company name). 4. The user can click on individual positions for more actions (e.g., evaluate).
Post Conditions	The professor has accessed a filtered view showing only the traineeship positions they supervise.

3.15<Use Case 15>

Use case ID	US15
Actors	User(Professor)
Pre conditions	The user is logged in. The user has the "Professor" role. The professor is assigned as supervisor to a traineeship position. The position has an assigned student.
Main flow of events	1. The professor navigates to the list of supervised positions. 2. The user selects the **"Evaluate"** button next to a specific assigned traineeship. 3. The system opens the **evaluation form**. 4. The professor fills in the following evaluation fields (all scored 1–5): **(Student Motivation, Student Efficiency , Student Effectiveness, Company Facilities, Company Guidance)** 5. The professor selects the **"Save Evaluation"** button. 6. The system saves the evaluation and links it to the specific student and position. 7. The user is redirected to the positions page.
Post Conditions	The evaluation is successfully recorded and stored. It is now associated with the assigned student and the evaluated company for that position. The professor can no longer (or optionally can) edit the evaluation depending on system rules.

3.16<Use Case 16>

Use case ID	US16
Actors	User(Committee member)
Pre conditions	The user is logged in. The user has the role of **Committee Member**.
Main flow of events	1. The user selects the **"Students"** button from the dashboard. 2. The system redirects the user to the **Manage students page**. 3. The system displays a list of students who have applied for a traineeship, showing relevant details (e.g., name). 4. For each student, the system offers an **"Assign Position"** button or action. 5. The user may proceed to assign a student to a specific traineeship position (handled in a separate use case).
Post Conditions	The committee member has accessed the list of student applicants.

3.17 and 3.18<Use Case 17-18>

Use case ID	US17&US18
Actors	User(Committee member)
Pre conditions	The user is logged in. The user has the role of **Committee Member**. The user is in the Manage Students page.
Main flow of events	1. The user navigates to the **Manage Students** page. 2. The user selects the **"Assign Position"** button next to a specific student. 3. The system redirects to the **Matching Options** page. 4. The user selects one of the three search strategies: **Search by Interests** **Search by Preferred Location** **Search by Both** 5. The user clicks the **"Search"** button. 6. The system filters available positions using: Matching **skills** required by the position vs. student's skills Matching **interests** (if selected) Matching **location** (if selected) 7. The system displays the **Search Results page**, showing a **ranked list** of suitable available positions sorted by match percentage. 8. For each position, the user sees key information (e.g., title, company, location, skill match level). 9. The user selects **"Assign to Student"** next to a preferred position. 10. The system assigns the position to the selected student.
Post Conditions	A traineeship position has been assigned to the selected student. The student no longer appears in the list of unassigned applicants. The assigned position is marked as no longer available.

3.19<Use Case 19>

Use case ID	US19
Actors	User(Committee member)
Pre conditions	The user is logged in. The user has the role of **Committee Member**. There are **traineeship positions with "In Progress" status**. Some of these positions have **no assigned supervising professor**.
Main flow of events	1. The user navigates to the **In Progress Traineeships** page. 2. The system displays a list of traineeship positions with current status = "In Progress". 3. For each position: 4. 1. If the position has **no assigned supervisor**, the **"Assign Professor"** button is available. 2. If the position **already has a professor**, a disabled button appears labeled: *"The position is already being supervised."* 5. The user selects **"Assign Professor"** for a specific unsupervised position. 6. The system redirects the user to the **Professor Matching Criteria** page. 7. The user selects one of the two search strategies: **Search based on Professor's Interests** **Search based on Professor's Supervision Load** 8. The user clicks the **"Search"** button. 9. The system processes the search and returns a **ranked list of professors**: Professors are ordered by best match based on selected strategy. For each professor, details are shown (e.g., interests, number of active supervisions). 10. The user clicks **"Assign as Supervisor"** next to the chosen professor. 11. The system assigns the professor to the traineeship position.

Alternative flow 1	At step 8, if no professors meet the search criteria: The system displays a message: "No suitable professors found based on the selected strategy." The user may return to step 6 and try an alternative strategy.
Post Conditions	The selected professor is now assigned as **supervisor** of the chosen traineeship position. The "Assign Professor" button becomes disabled for that position and is replaced with the label: *"The position is already being supervised."*

3.20<Use Case 20>

Use case ID	US20
Actors	User(Committee member)
Pre conditions	The user is logged in. The user has the role of **Committee Member**. The user is currently at the traineeship committee dashboard.
Main flow of events	1. The user selects the **"Traineeships"** button from the dashboard menu. 2. The system redirects the user to the **"In Progress Traineeships"** page. 3. The system displays a list of all traineeship positions with status = "In Progress". 4. For each traineeship, the system shows key information, such as: (Position title, Assigned student, Assigned supervisor (if any), Evaluation status) 5. The user can interact with each traineeship through available actions, such as: **Assign Professor** (if not assigned) **View Evaluations** **Complete** 6. For each traineeship, the system shows key information, such as: (Position title, Assigned student, Assigned supervisor (if any), Evaluation status)
Alternative flow 1	At step 4, if no active traineeships exist: The system displays a message: "There are currently no in-progress traineeships."
Post Conditions	The committee member has accessed the full list of ongoing traineeships. The user is able to take further actions for each listed traineeship.

3.21<Use Case 21>

Use case ID	US21
Actors	User(Committee member)
Pre conditions	The user is logged in. The user has the role of **Committee Member**. The user is on the **"In Progress Traineeships"** page.
Main flow of events	1. The user navigates to the **In Progress Traineeships** page. 2. For each traineeship position, the system displays a **"View Evaluations"** button. 3. The user selects **"View Evaluations"** to expand and review: **Company Evaluation** details (Motivation, Efficiency, Effectiveness) **Professor Evaluation** details (Motivation, Efficiency, Effectiveness, Facilities, Guidance) If an evaluation is missing, a message appears (e.g., *"No Company Evaluation submitted."*). 4. When both evaluations are submitted, the system enables the **"Complete"** button. 5. The user selects **"Complete"** to finalize the traineeship. 6. The system calculates and assigns the final grade to the student. 7. The system displays the confirmation message ‘’**Traineeship completed successfully**.’’.
Post Conditions	The traineeship is completed and can no longer be viewed.

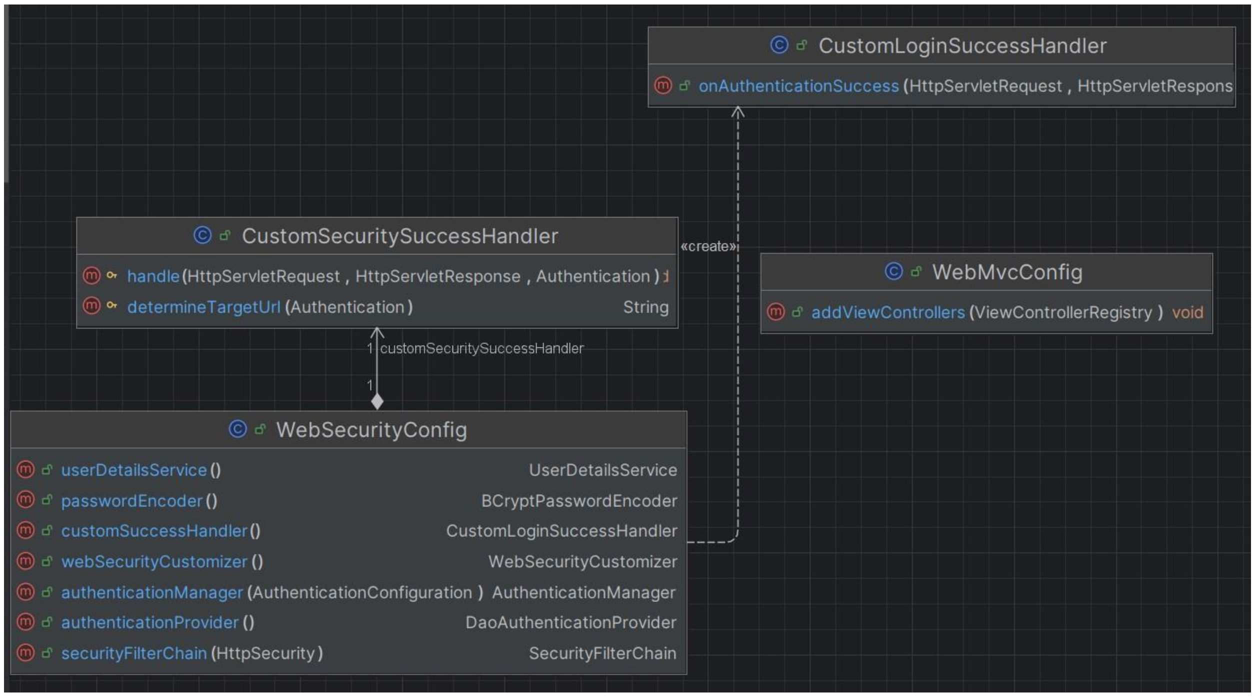
4 Design

4.1 Architecture

<Specify the overall architecture for this release in terms of a **UML package diagram**.>

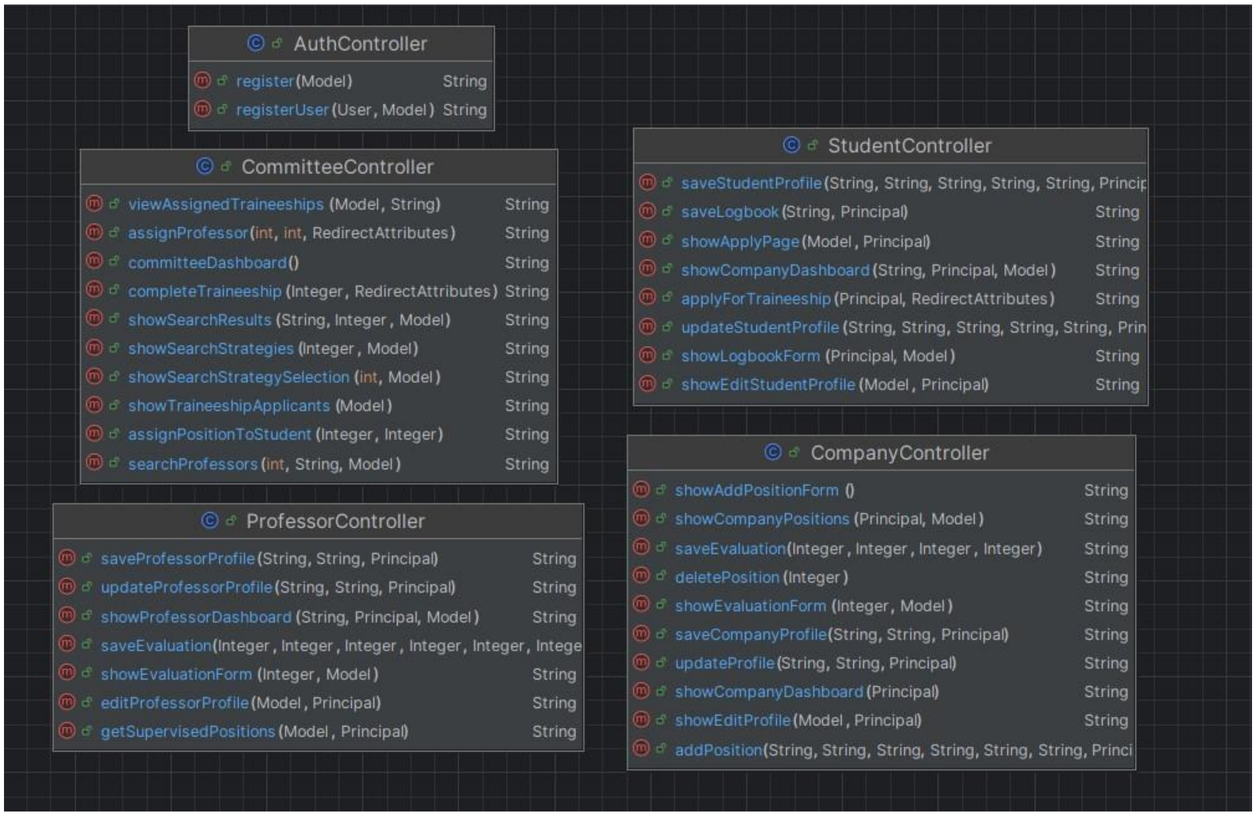
4.2 Config Package

CONFIG PACKAGE

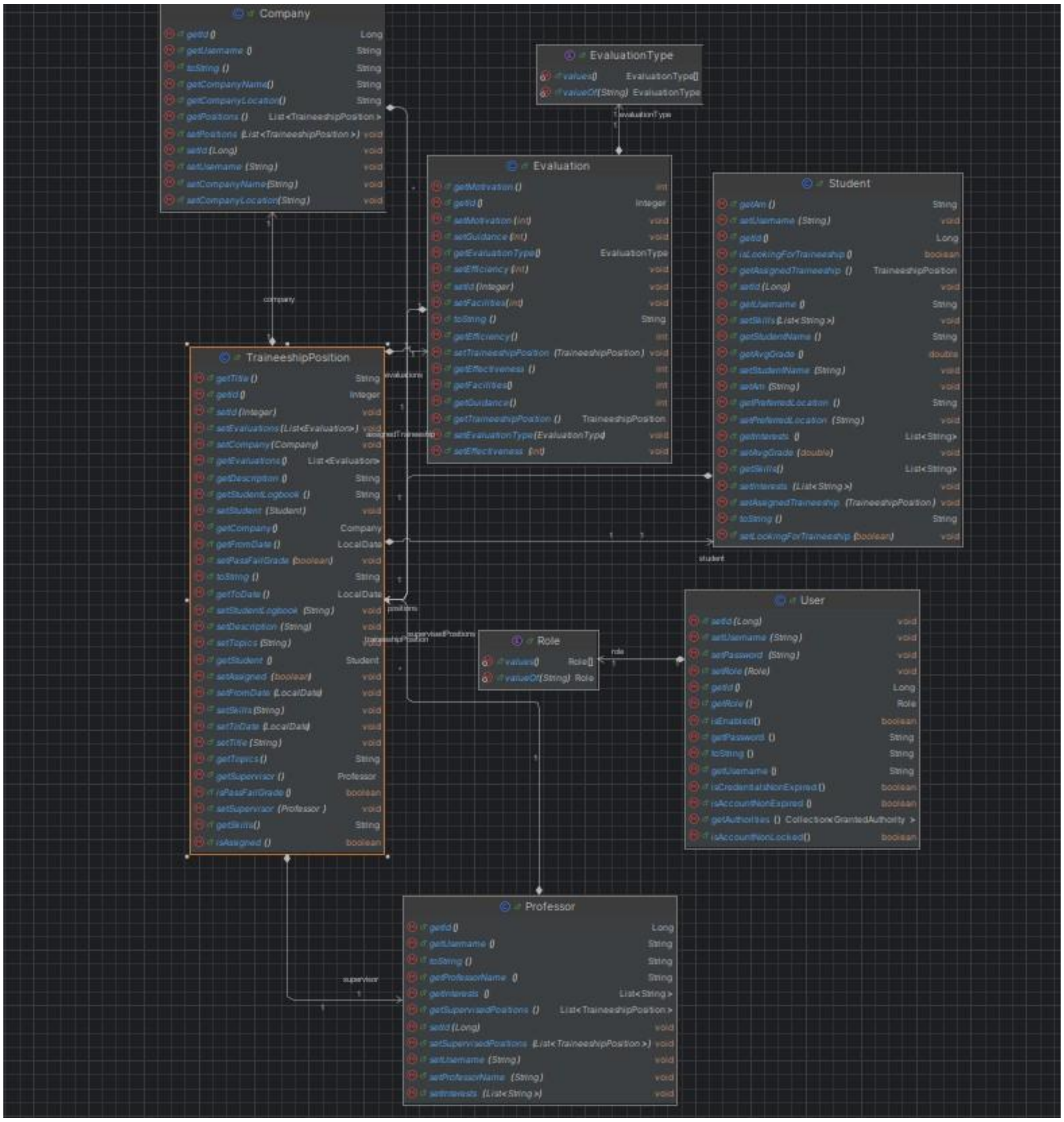

CustomLoginSuccessHandler
onAuthenticationSuccess (HttpServletRequest , HttpServletRespons
CustomSecuritySuccessHandler
handle (HttpServletRequest , HttpServletResponse , Authentication)
determineTargetUrl (Authentication) String
«create»
WebMvcConfig
addViewControllers (ViewControllerRegistry) void
1 customSecuritySuccessHandler
1
WebSecurityConfig
userDetailsService () UserDetailsService
passwordEncoder () BCryptPasswordEncoder
customSuccessHandler () CustomLoginSuccessHandler
webSecurityCustomizer () WebSecurityCustomizer
authenticationManager (AuthenticationConfiguration) AuthenticationManager
authenticationProvider () DaoAuthenticationProvider
securityFilterChain (HttpSecurity) SecurityFilterChain

4.3 Controllers Package

CONTROLLERS PACKAGE

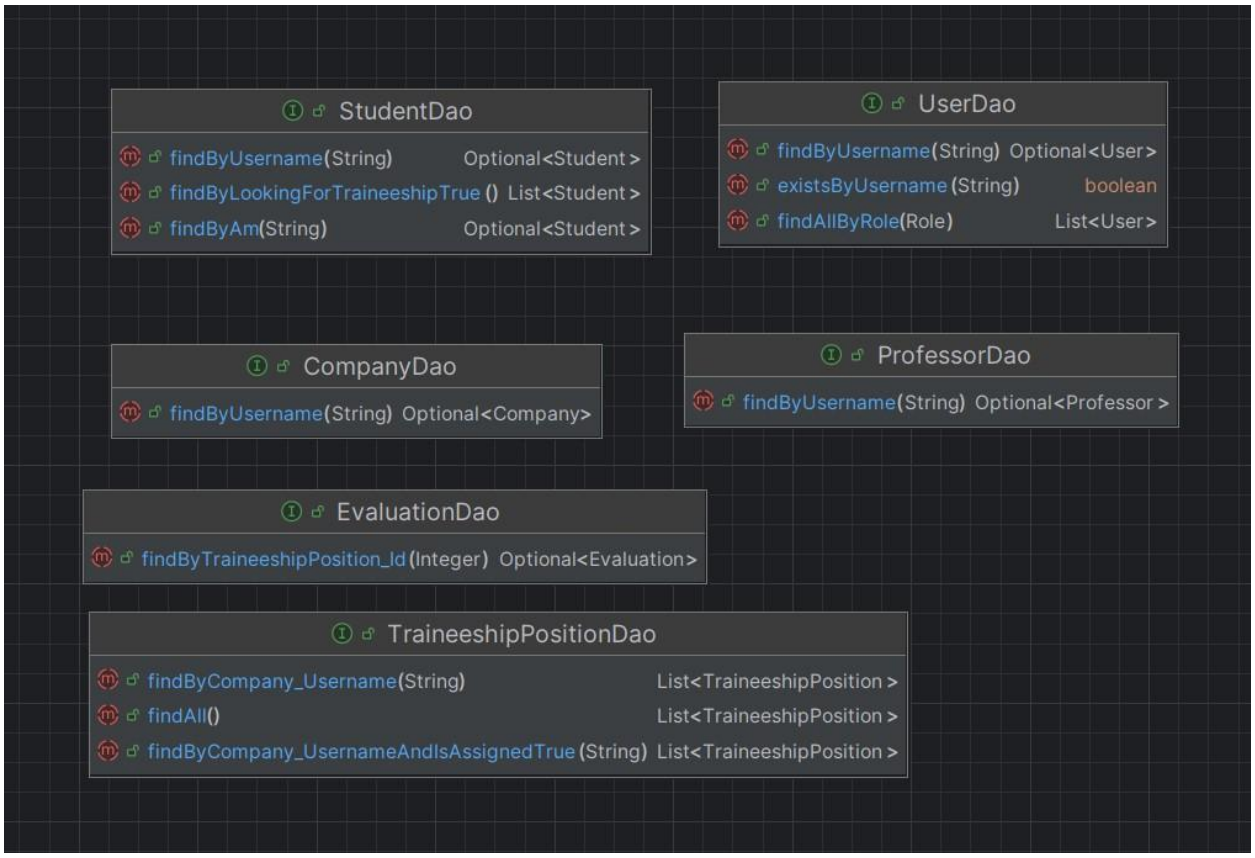

AuthController
register(Model) String
registerUser(User, Model) String
CommitteeController
viewAssignedTraineeships (Model, String) String
assignProfessor(int, int, RedirectAttributes) String
committeeDashboard() String
completeTraineeship (Integer, RedirectAttributes) String
showSearchResults (String, Integer, Model) String
showSearchStrategies (Integer, Model) String
showSearchStrategySelection (int, Model) String
showTraineeshipApplicants (Model) String
assignPositionToStudent (Integer, Integer) String
searchProfessors (int, String, Model) String
ProfessorController
saveProfessorProfile (String, String, Principal) String
updateProfessorProfile (String, String, Principal) String
showProfessorDashboard (String, Principal, Model) String
saveEvaluation(Integer, Integer, Integer, Integer, Integer, Intege
showEvaluationForm (Integer, Model) String
editProfessorProfile (Model, Principal) String
getSupervisedPositions (Model, Principal) String
StudentController
saveStudentProfile (String, String, String, String, String, Princip
saveLogbook (String, Principal) String
showApplyPage (Model, Principal) String
showCompanyDashboard (String, Principal, Model) String
applyForTraineeship (Principal, RedirectAttributes) String
updateStudentProfile (String, String, String, String, String, Prin
showLogbookForm (Principal, Model) String
showEditStudentProfile (Model, Principal) String
CompanyController
showAddPositionForm () String
showCompanyPositions (Principal, Model) String
saveEvaluation(Integer, Integer, Integer, Integer) String
deletePosition (Integer) String
showEvaluationForm (Integer, Model) String
saveCompanyProfile(String, String, Principal) String
updateProfile (String, String, Principal) String
showCompanyDashboard (Principal) String
showEditProfile (Model, Principal) String
addPosition(String, String, String, String, String, String, Princi

DOMAIN PACKAGE

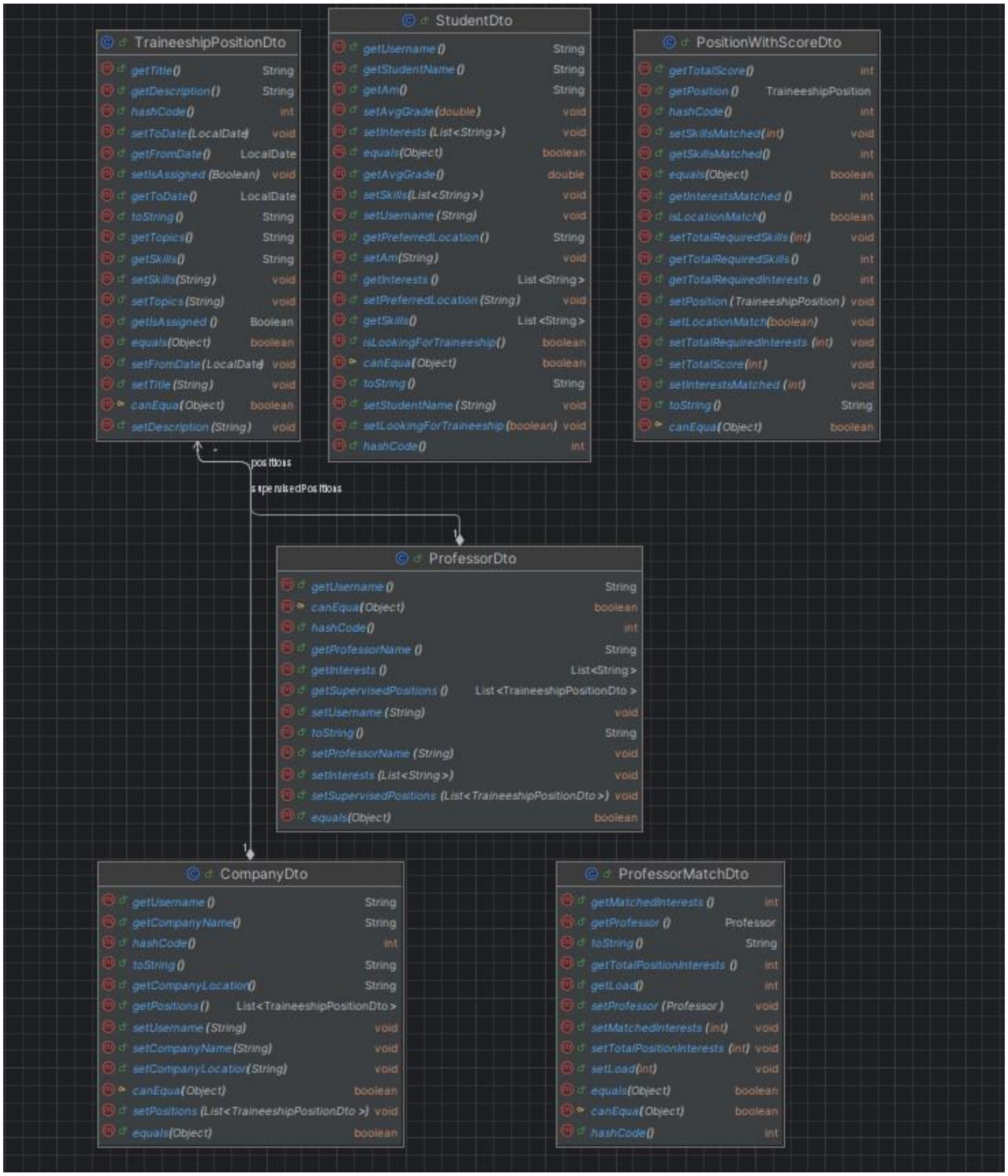

Company
getId() Long
getUsername() String
toString() String
getCompanyName() String
getCompanyLocation() String
getPositions() List<TraineeshipPosition>
setPositions(List<TraineeshipPosition>) void
setId(Long) void
setUsername(String) void
setCompanyName(String) void
setCompanyLocation(String) void
EvaluationType
values() EvaluationType[]
valueOf(String) EvaluationType
Evaluation
getMotivation() int
getId() Integer
setMotivation(int) void
setGuidance(int) void
getEvaluationType() EvaluationType
setEfficiency(int) void
setId(Integer) void
setFacilities(int) void
toString() String
getEfficiency() int
setTraineeshipPosition(TraineeshipPosition) void
getEffectiveness() int
getFacilities() int
getGuidance() int
getTraineeshipPosition() TraineeshipPosition
setEvaluationType(EvaluationType) void
setEffectiveness(int) void
Student
getAm() String
setUsername(String) void
getId() Long
isLookingForTraineeship() boolean
getAssignedTraineeship() TraineeshipPosition
setId(Long) void
getUsername() String
setSkills(List<String>) void
getStudentName() String
getAvgGrade() double
setStudentName(String) void
setAm(String) void
getPreferredLocation() String
setPreferredLocation(String) void
getInterests() List<String>
setAvgGrade(double) void
getSkills() List<String>
setInterests(List<String>) void
setAssignedTraineeship(TraineeshipPosition) void
toString() String
setLookingForTraineeship(boolean) void
TraineeshipPosition
getTitle() String
getId() Integer
setId(Integer) void
setEvaluations(List<Evaluation>) void
setCompany(Company) void
getEvaluations() List<Evaluation>
getDescription() String
getStudentLogbook() String
setStudent(Student) void
getCompany() Company
getFromDate() LocalDate
setPassFailGrade(boolean) void
toString() String
getToDate() LocalDate
setStudentLogbook(String) void
setDescription(String) void
setTopics(String) void
getStudent() Student
setAssigned(boolean) void
setFromDate(LocalDate) void
setSkills(String) void
setToDate(LocalDate) void
setTitle(String) void
getTopics() String
getSupervisor() Professor
isPassFailGrade() boolean
setSupervisor(Professor) void
getSkills() String
isAssigned() boolean
Role
values() Role[]
valueOf(String) Role
User
setId(Long) void
setUsername(String) void
setPassword(String) void
setRole(Role) void
getId() Long
getRole() Role
isEnabled() boolean
getPassword() String
toString() String
getUsername() String
isCredentialsNonExpired() boolean
isAccountNonExpired() boolean
getAuthorities() Collection<GrantedAuthority>
isAccountNonLocked() boolean
Professor
getId() Long
getUsername() String
toString() String
getProfessorName() String
getInterests() List<String>
getSupervisedPositions() List<TraineeshipPosition>
setId(Long) void
setSupervisedPositions(List<TraineeshipPosition>) void
setUsername(String) void
setProfessorName(String) void
setInterests(List<String>) void
evaluationType
company
evaluations
assignedTraineeship
student
positions
supervisedPositions
traineeshipPosition
role
supervisor

4.5 DAOs Package

DAOS PACKAGE

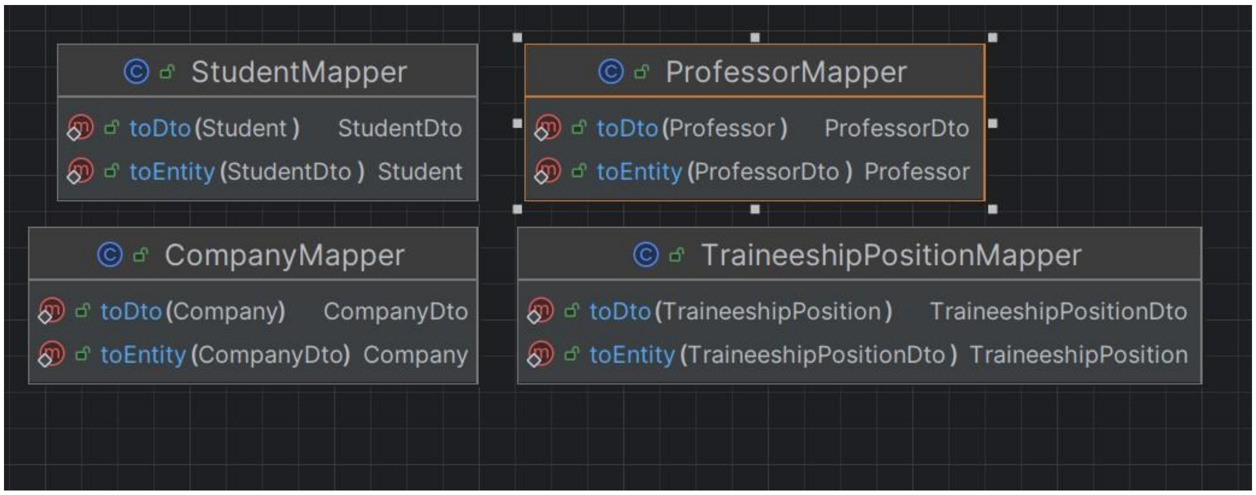

StudentDao
findByUsername(String) Optional<Student>
findByLookingForTraineeshipTrue() List<Student>
findByAm(String) Optional<Student>
UserDao
findByUsername(String) Optional<User>
existsByUsername(String) boolean
findAllByRole(Role) List<User>
CompanyDao
findByUsername(String) Optional<Company>
ProfessorDao
findByUsername(String) Optional<Professor>
EvaluationDao
findByTraineeshipPosition_Id(Integer) Optional<Evaluation>
TraineeshipPositionDao
findByCompany_Username(String) List<TraineeshipPosition>
findAll() List<TraineeshipPosition>
findByCompany_UsernameAndIsAssignedTrue(String) List<TraineeshipPosition>

DTOS PACKAGE


TraineeshipPositionDto
getTitle() String
getDescription() String
hashCode() int
setToDate(LocalDate) void
getFromDate() LocalDate
setIsAssigned(Boolean) void
getToDate() LocalDate
toString() String
getTopics() String
getSkills() String
setSkills(String) void
setTopics(String) void
getIsAssigned() Boolean
equals(Object) boolean
setFromDate(LocalDate) void
setTitle(String) void
canEqual(Object) boolean
setDescription(String) void
StudentDto
getUsername() String
getStudentName() String
getAm() String
setAvgGrade(double) void
setInterests(List<String>) void
equals(Object) boolean
getAvgGrade() double
setSkills(List<String>) void
setUsername(String) void
getPreferredLocation() String
setAm(String) void
getInterests() List<String>
setPreferredLocation(String) void
getSkills() List<String>
isLookingForTraineeship() boolean
canEqual(Object) boolean
toString() String
setStudentName(String) void
setLookingForTraineeship(boolean) void
hashCode() int
PositionWithScoreDto
getTotalScore() int
getPosition() TraineeshipPosition
hashCode() int
setSkillsMatched(int) void
getSkillsMatched() int
equals(Object) boolean
getInterestsMatched() int
isLocationMatch() boolean
setTotalRequiredSkills(int) void
getTotalRequiredSkills() int
getTotalRequiredInterests() int
setPosition(TraineeshipPosition) void
setLocationMatch(boolean) void
setTotalRequiredInterests(int) void
setTotalScore(int) void
setInterestsMatched(int) void
toString() String
canEqual(Object) boolean
positions
supervisedPositions
1
*
1
ProfessorDto
getUsername() String
canEqual(Object) boolean
hashCode() int
getProfessorName() String
getInterests() List<String>
getSupervisedPositions() List<TraineeshipPositionDto>
setUsername(String) void
toString() String
setProfessorName(String) void
setInterests(List<String>) void
setSupervisedPositions(List<TraineeshipPositionDto>) void
equals(Object) boolean
CompanyDto
getUsername() String
getCompanyName() String
hashCode() int
toString() String
getCompanyLocation() String
getPositions() List<TraineeshipPositionDto>
setUsername(String) void
setCompanyName(String) void
setCompanyLocation(String) void
canEqual(Object) boolean
setPositions(List<TraineeshipPositionDto>) void
equals(Object) boolean
ProfessorMatchDto
getMatchedInterests() int
getProfessor() Professor
toString() String
getTotalPositionInterests() int
getLoad() int
setProfessor(Professor) void
setMatchedInterests(int) void
setTotalPositionInterests(int) void
setLoad(int) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int

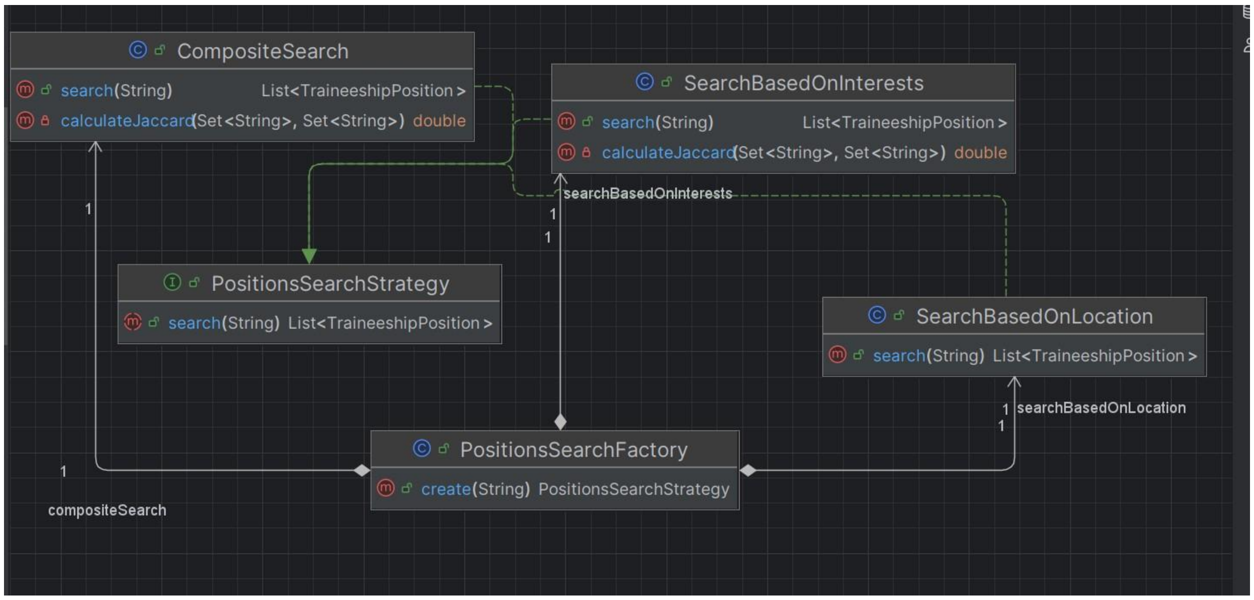
4.7 Mappers Package

MAPPERS PACKAGE


StudentMapper
toDto(Student) StudentDto
toEntity(StudentDto) Student
ProfessorMapper
toDto(Professor) ProfessorDto
toEntity(ProfessorDto) Professor
CompanyMapper
toDto(Company) CompanyDto
toEntity(CompanyDto) Company
TraineeshipPositionMapper
toDto(TraineeshipPosition) TraineeshipPositionDto
toEntity(TraineeshipPositionDto) TraineeshipPosition

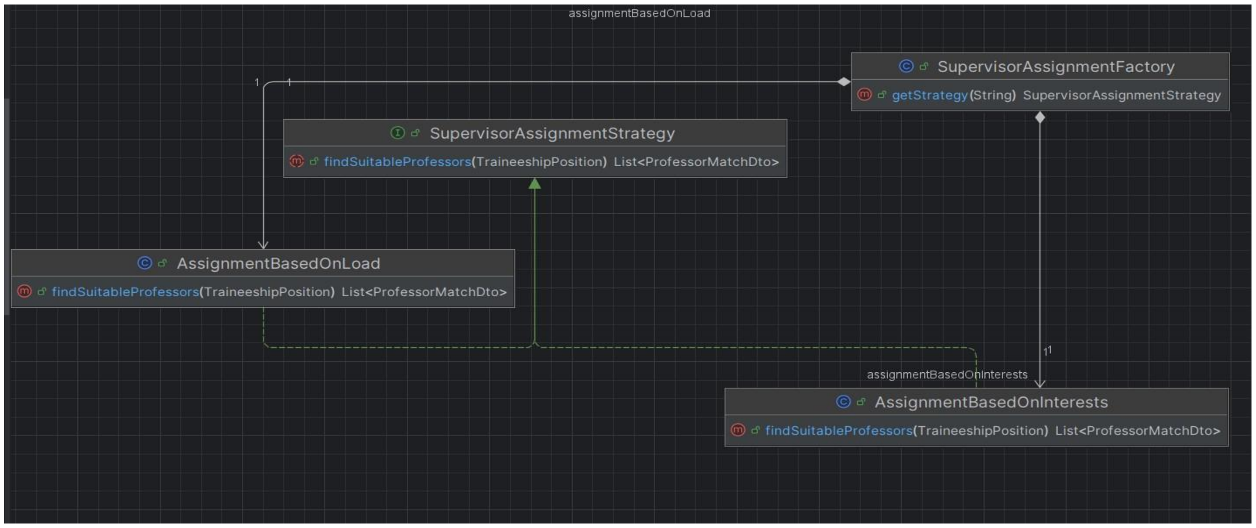
4.8 Strategies.positionsearch Package

STRATEGIES.POSITIONSEARCH PACKAGE


CompositeSearch
search(String) List<TraineeshipPosition>
calculateJaccard(Set<String>, Set<String>) double
SearchBasedOnInterests
search(String) List<TraineeshipPosition>
calculateJaccard(Set<String>, Set<String>) double
searchBasedOnInterests
1
1
1
PositionsSearchStrategy
search(String) List<TraineeshipPosition>
SearchBasedOnLocation
search(String) List<TraineeshipPosition>
1
searchBasedOnLocation
1
PositionsSearchFactory
create(String) PositionsSearchStrategy
1
compositeSearch

4.9 Strategies.professorsearch Package

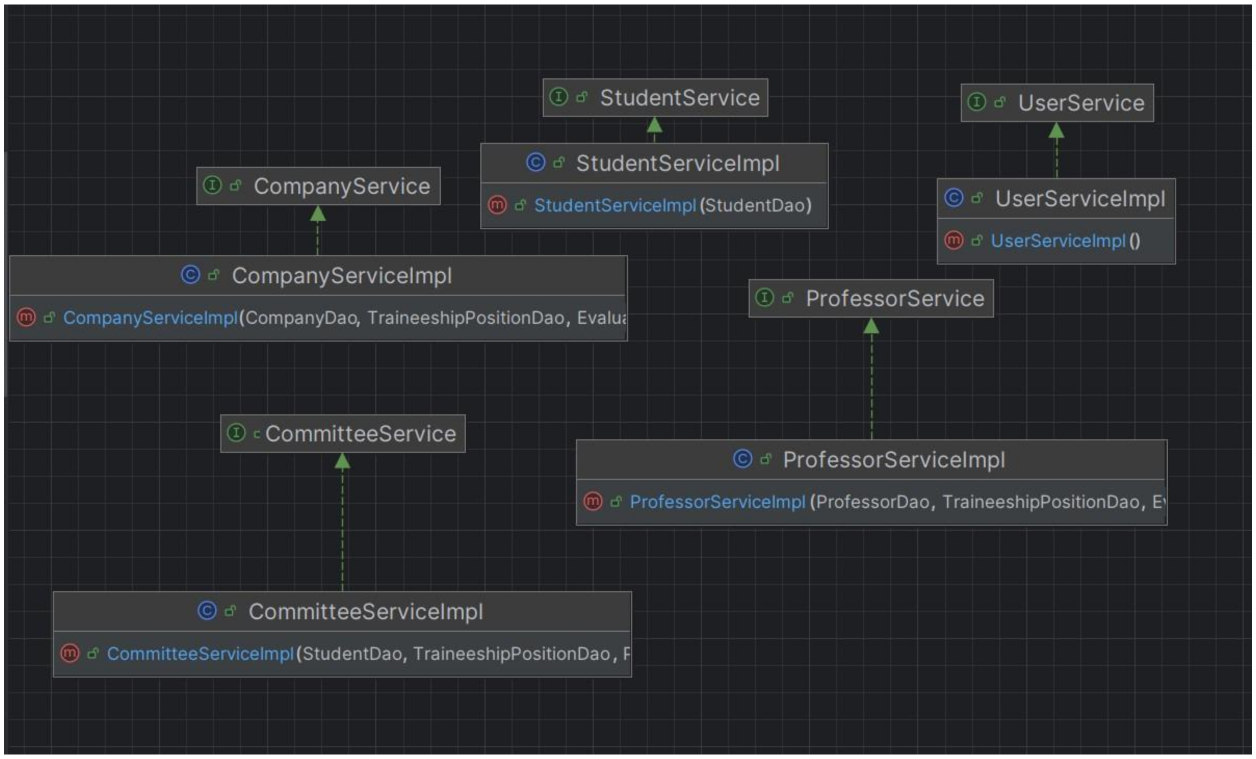
STRATEGIES.PROFESSORSEARCH PACKAGE



assignmentBasedOnLoad
1
1
SupervisorAssignmentFactory
getStrategy(String) SupervisorAssignmentStrategy
SupervisorAssignmentStrategy
findSuitableProfessors(TraineeshipPosition) List<ProfessorMatchDto>
AssignmentBasedOnLoad
findSuitableProfessors(TraineeshipPosition) List<ProfessorMatchDto>
1
1
assignmentBasedOnInterests
AssignmentBasedOnInterests
findSuitableProfessors(TraineeshipPosition) List<ProfessorMatchDto>

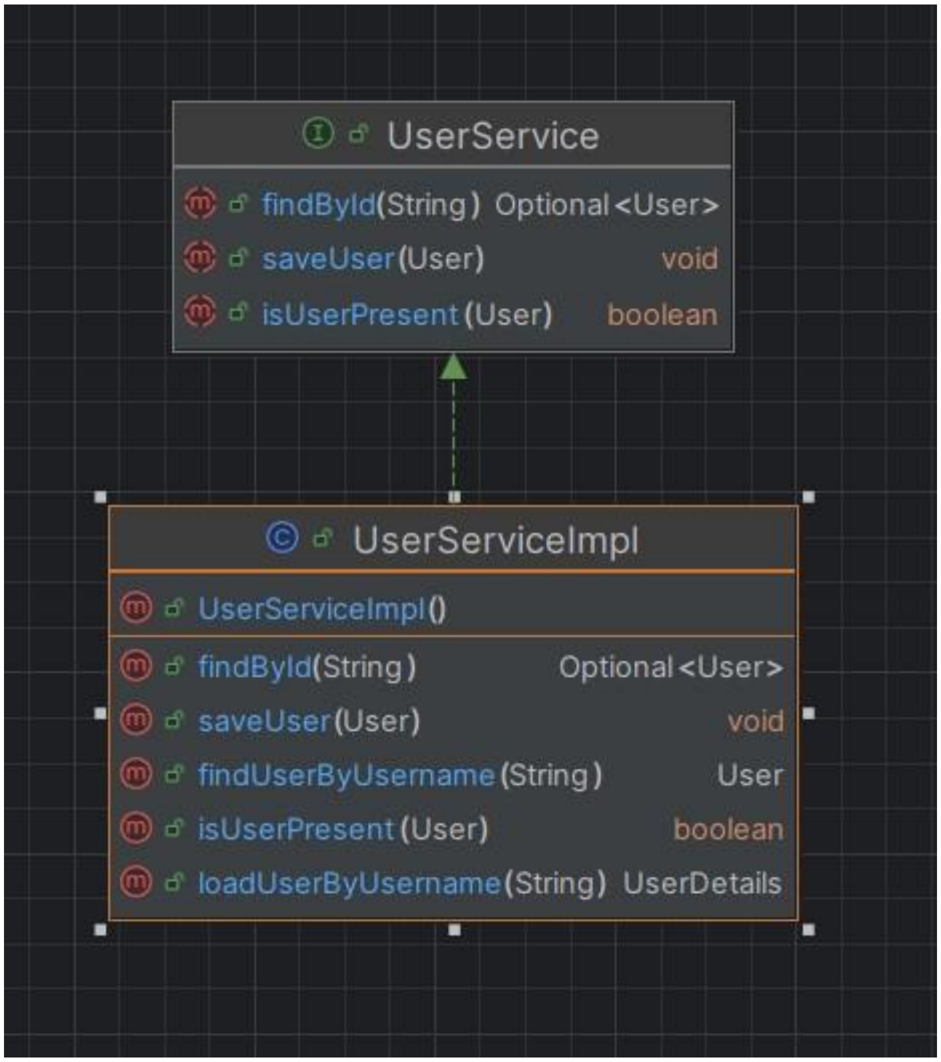
4.10 Service Package

SERVICE PACKAGE

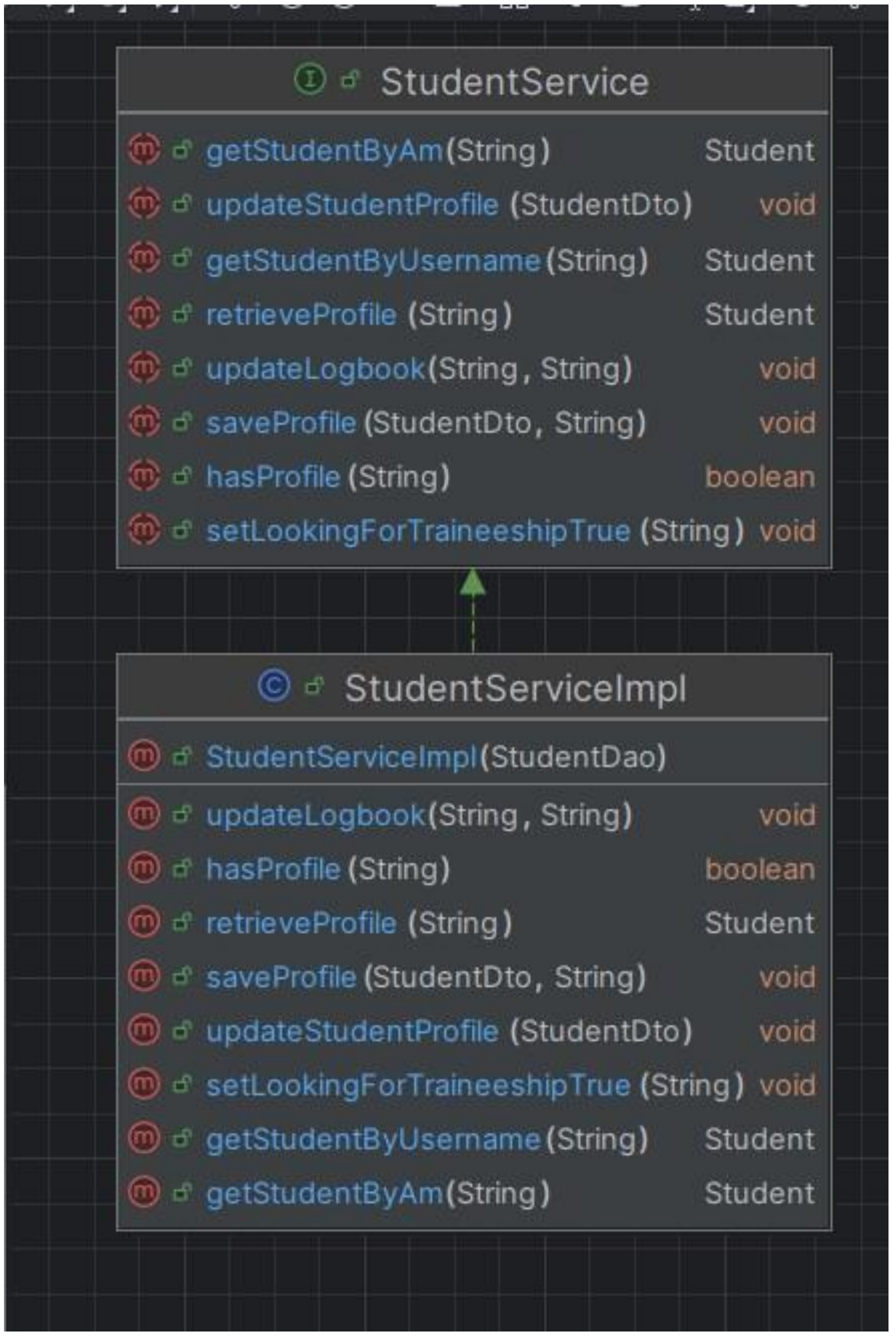


StudentService
UserService
StudentServiceImpl
StudentServiceImpl (StudentDao)
CompanyService
UserServiceImpl
UserServiceImpl ()
CompanyServiceImpl
CompanyServiceImpl(CompanyDao, TraineeshipPositionDao, Evalua
ProfessorService
CommitteeService
ProfessorServiceImpl
ProfessorServiceImpl (ProfessorDao, TraineeshipPositionDao, E
CommitteeServiceImpl
CommitteeServiceImpl (StudentDao, TraineeshipPositionDao, F

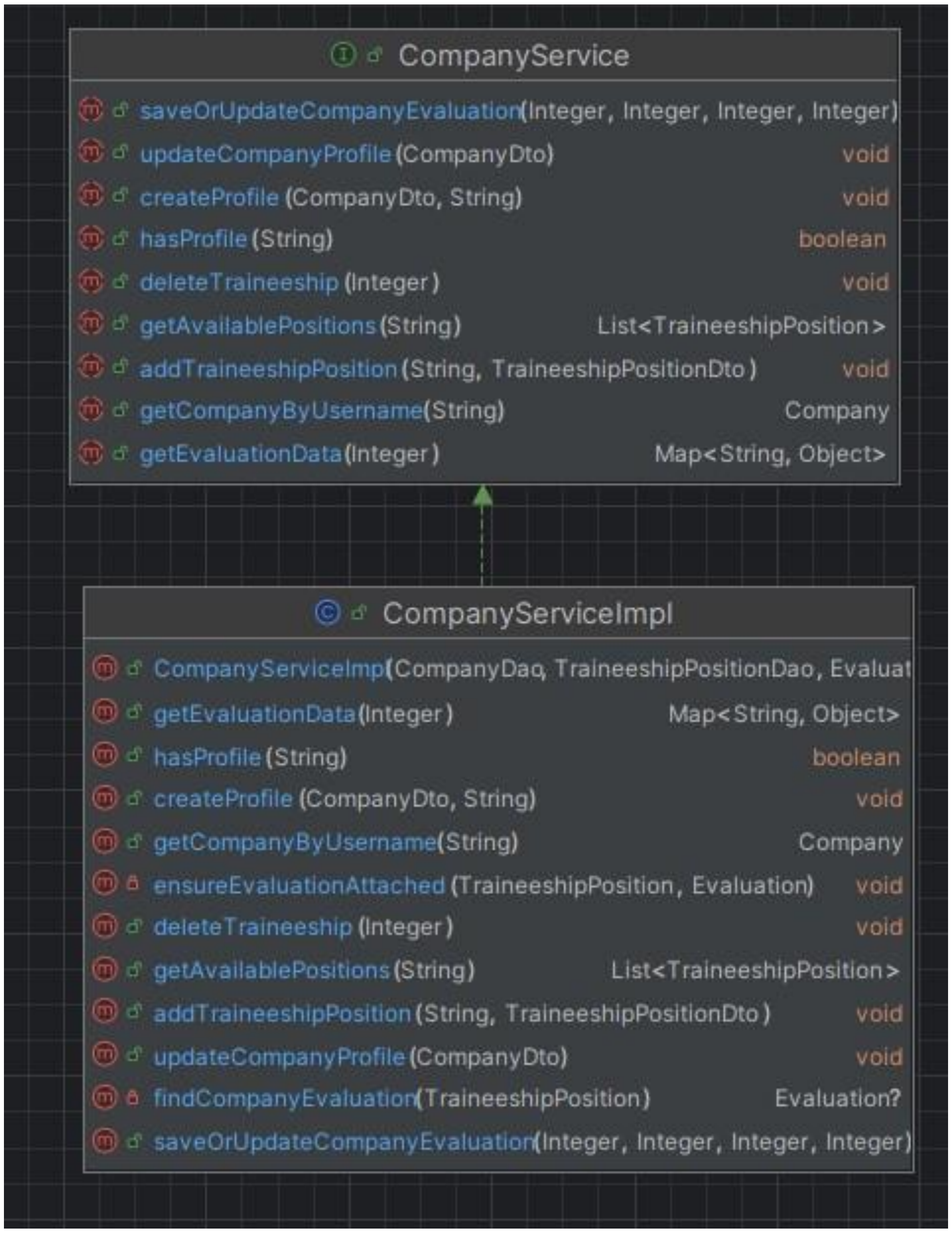
USER SERVICE


UserService
findById(String) Optional<User>
saveUser(User) void
isUserPresent(User) boolean
UserServiceImpl
UserServiceImpl()
findById(String) Optional<User>
saveUser(User) void
findUserByUsername(String) User
isUserPresent(User) boolean
loadUserByUsername(String) UserDetails

STUDENT SERVICE


StudentService
getStudentByAm(String) Student
updateStudentProfile (StudentDto) void
getStudentByUsername(String) Student
retrieveProfile (String) Student
updateLogbook(String, String) void
saveProfile (StudentDto, String) void
hasProfile (String) boolean
setLookingForTraineeshipTrue (String) void
StudentServiceImpl
StudentServiceImpl(StudentDao)
updateLogbook(String, String) void
hasProfile (String) boolean
retrieveProfile (String) Student
saveProfile (StudentDto, String) void
updateStudentProfile (StudentDto) void
setLookingForTraineeshipTrue (String) void
getStudentByUsername(String) Student
getStudentByAm(String) Student

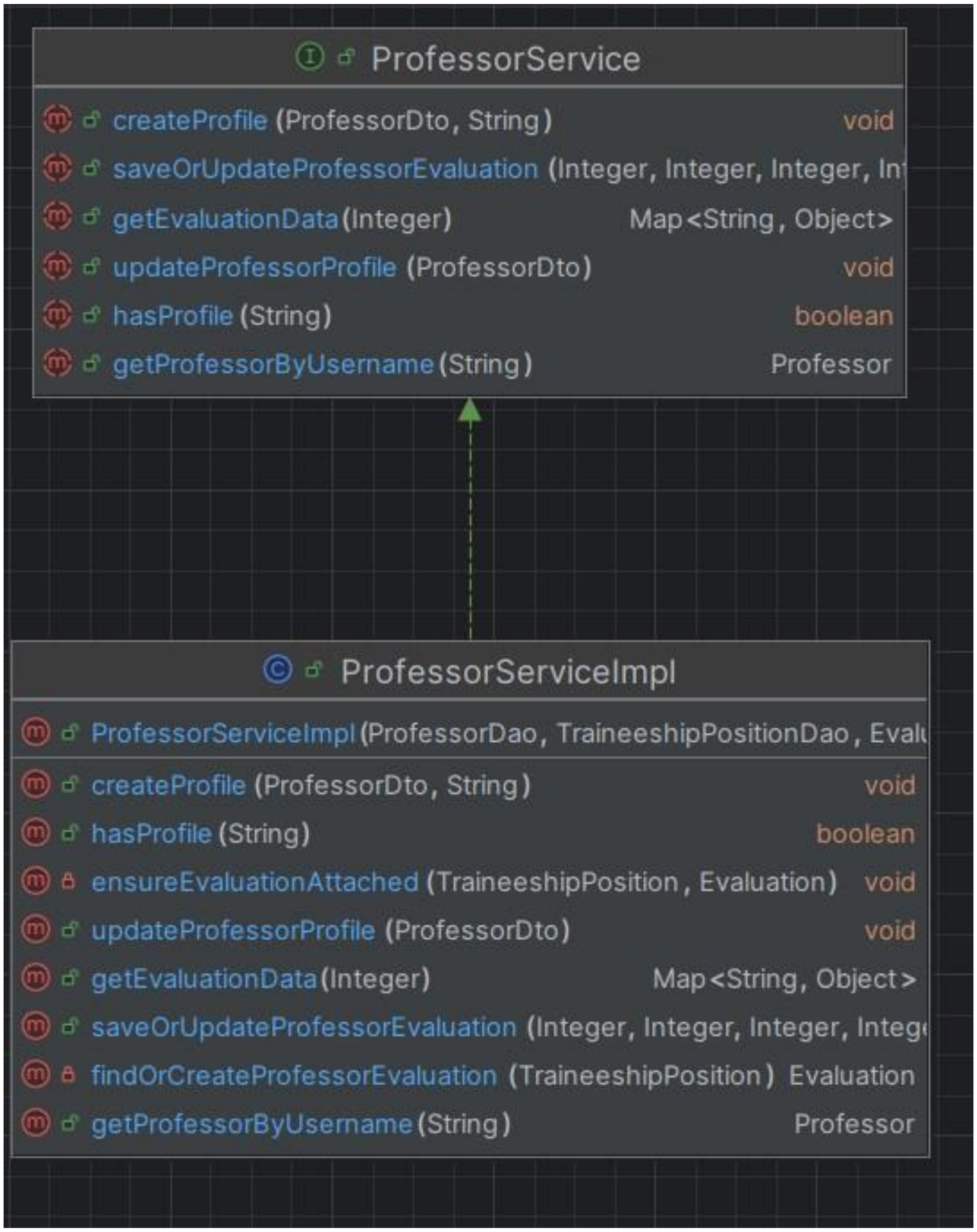
COMPANY SERVICE



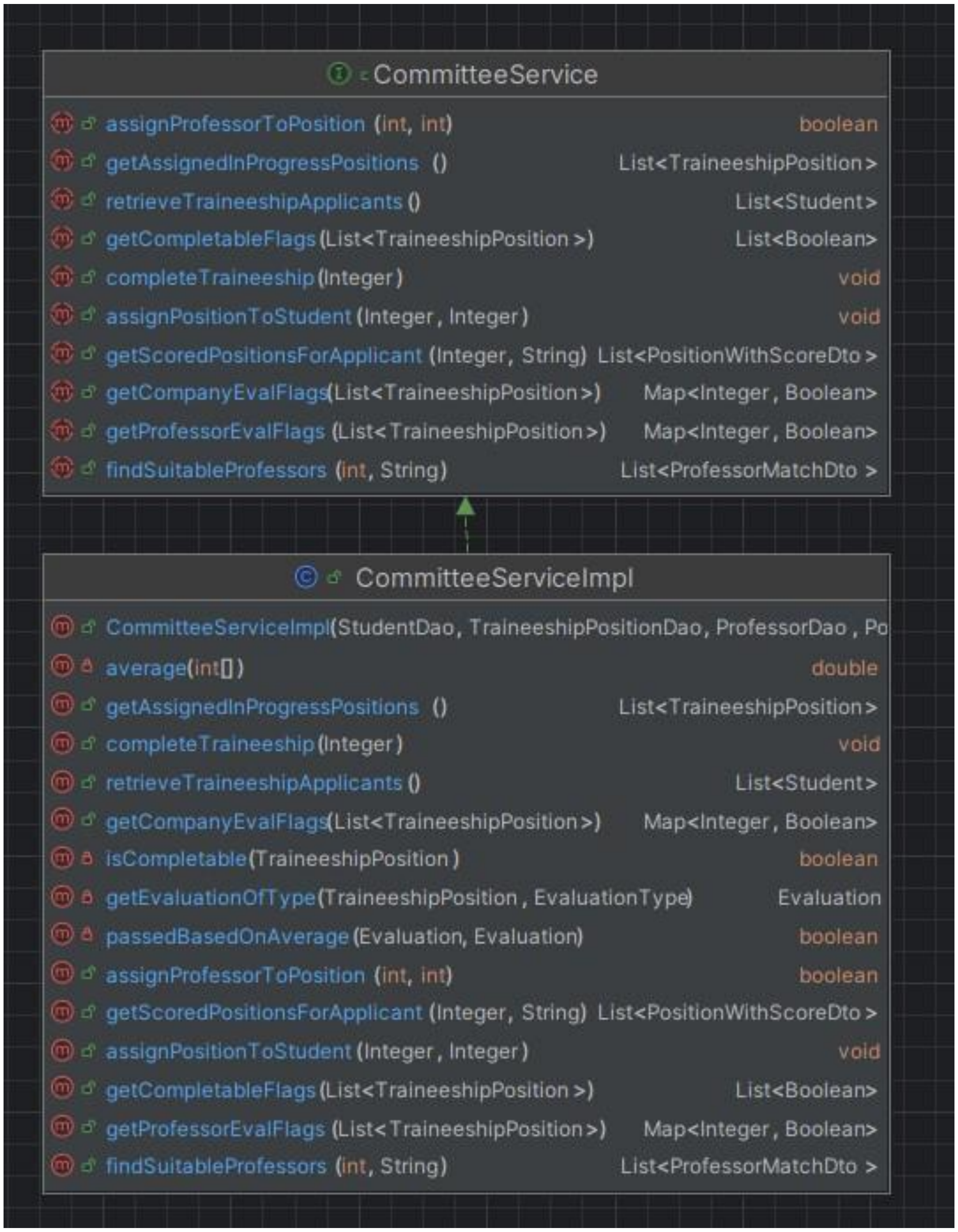
```
CompanyService
saveOrUpdateCompanyEvaluation(Integer, Integer, Integer, Integer)
updateCompanyProfile(CompanyDto)                              void
createProfile(CompanyDto, String)                             void
hasProfile(String)                                         boolean
deleteTraineeship(Integer)                                    void
getAvailablePositions(String)           List<TraineeshipPosition>
addTraineeshipPosition(String, TraineeshipPositionDto)        void
getCompanyByUsername(String)                               Company
getEvaluationData(Integer)                     Map<String, Object>

CompanyServiceImpl
CompanyServiceImp(CompanyDao, TraineeshipPositionDao, Evaluat
getEvaluationData(Integer)                     Map<String, Object>
hasProfile(String)                                         boolean
createProfile(CompanyDto, String)                             void
getCompanyByUsername(String)                               Company
ensureEvaluationAttached(TraineeshipPosition, Evaluation)     void
deleteTraineeship(Integer)                                    void
getAvailablePositions(String)           List<TraineeshipPosition>
addTraineeshipPosition(String, TraineeshipPositionDto)        void
updateCompanyProfile(CompanyDto)                              void
findCompanyEvaluation(TraineeshipPosition)             Evaluation?
saveOrUpdateCompanyEvaluation(Integer, Integer, Integer, Integer)
```

PROFESSOR SERVICE


ProfessorService
createProfile (ProfessorDto, String) void
saveOrUpdateProfessorEvaluation (Integer, Integer, Integer, In
getEvaluationData (Integer) Map<String, Object>
updateProfessorProfile (ProfessorDto) void
hasProfile (String) boolean
getProfessorByUsername (String) Professor
ProfessorServiceImpl
ProfessorServiceImpl (ProfessorDao, TraineeshipPositionDao, Evalu
createProfile (ProfessorDto, String) void
hasProfile (String) boolean
ensureEvaluationAttached (TraineeshipPosition, Evaluation) void
updateProfessorProfile (ProfessorDto) void
getEvaluationData (Integer) Map<String, Object>
saveOrUpdateProfessorEvaluation (Integer, Integer, Integer, Integ
findOrCreateProfessorEvaluation (TraineeshipPosition) Evaluation
getProfessorByUsername (String) Professor

COMMITTEE SERVICE

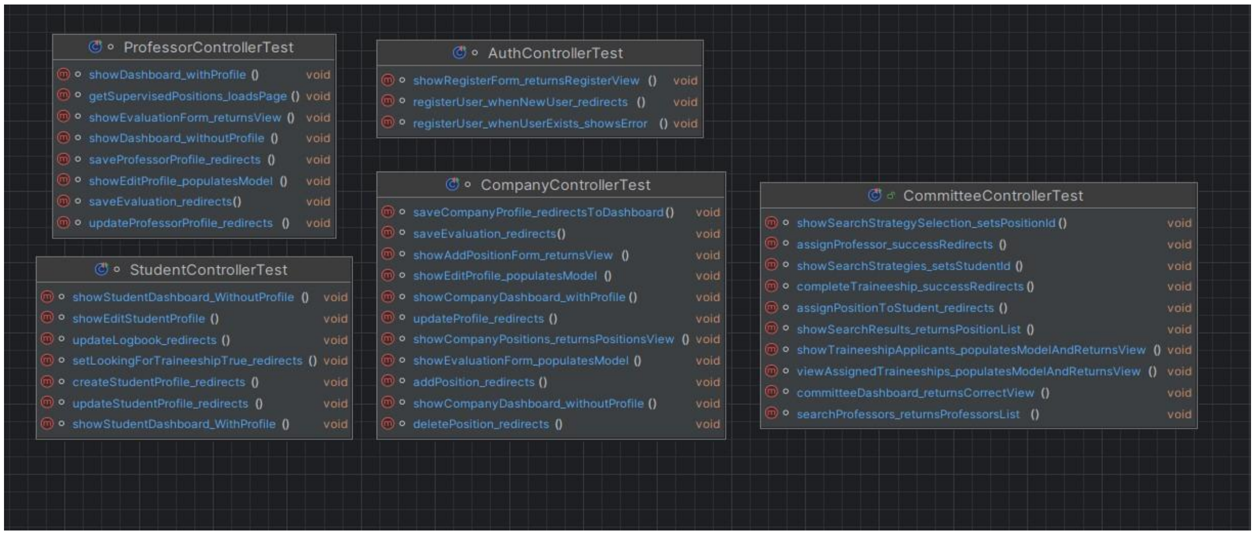


```
CommitteeService
assignProfessorToPosition (int, int)                          boolean
getAssignedInProgressPositions ()                             List<TraineeshipPosition>
retrieveTraineeshipApplicants ()                              List<Student>
getCompletableFlags (List<TraineeshipPosition >)              List<Boolean>
completeTraineeship (Integer)                                 void
assignPositionToStudent (Integer, Integer)                    void
getScoredPositionsForApplicant (Integer, String)              List<PositionWithScoreDto >
getCompanyEvalFlags (List<TraineeshipPosition >)              Map<Integer, Boolean>
getProfessorEvalFlags (List< TraineeshipPosition >)           Map<Integer, Boolean>
findSuitableProfessors (int, String)                          List<ProfessorMatchDto >
```

```
CommitteeServiceImpl
CommitteeServiceImpl(StudentDao, TraineeshipPositionDao, ProfessorDao , Po
average(int[])                                                double
getAssignedInProgressPositions ()                             List<TraineeshipPosition>
completeTraineeship (Integer)                                 void
retrieveTraineeshipApplicants ()                              List<Student>
getCompanyEvalFlags (List<TraineeshipPosition >)              Map<Integer, Boolean>
isCompletable (TraineeshipPosition)                           boolean
getEvaluationOfType (TraineeshipPosition, EvaluationType)     Evaluation
passedBasedOnAverage (Evaluation, Evaluation)                 boolean
assignProfessorToPosition (int, int)                          boolean
getScoredPositionsForApplicant (Integer, String)              List<PositionWithScoreDto >
assignPositionToStudent (Integer, Integer)                    void
getCompletableFlags (List<TraineeshipPosition >)              List<Boolean>
getProfessorEvalFlags (List< TraineeshipPosition >)           Map<Integer, Boolean>
findSuitableProfessors (int, String)                          List<ProfessorMatchDto >
```

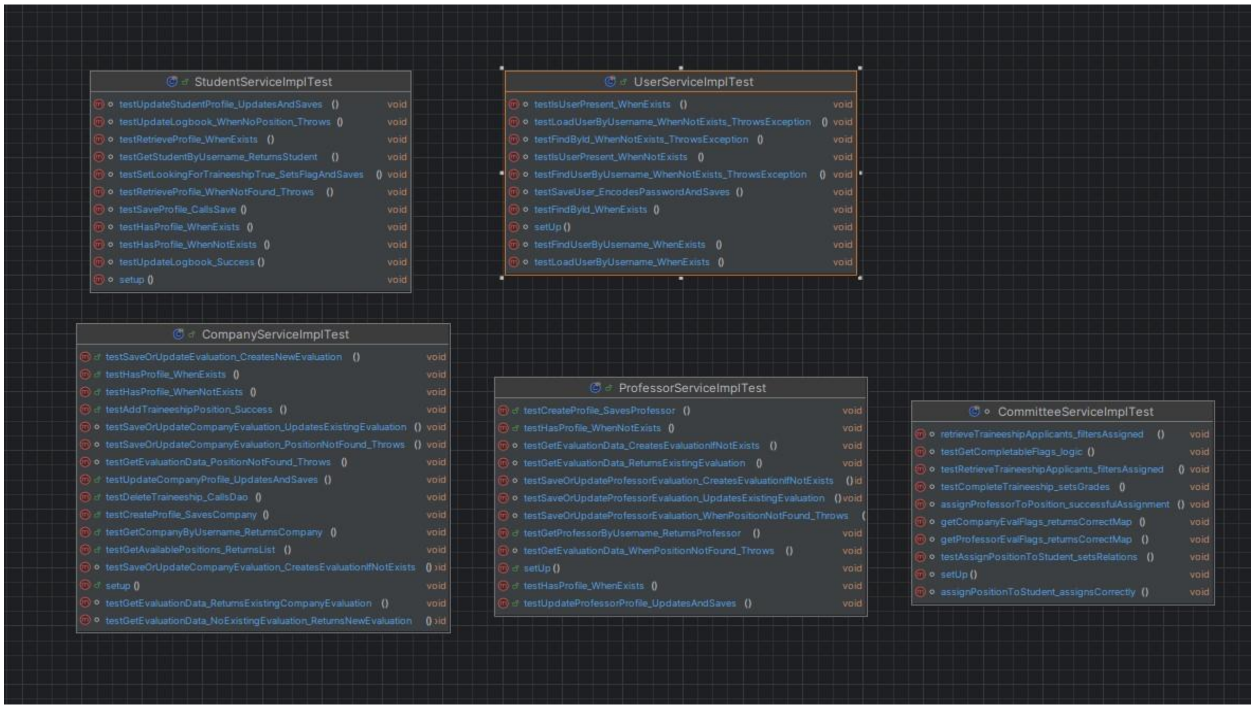
4.11 Controller Test Package

CONTROLLER TEST PACKAGE


ProfessorControllerTest
showDashboard_withProfile () void
getSupervisedPositions_loadsPage () void
showEvaluationForm_returnsView () void
showDashboard_withoutProfile () void
saveProfessorProfile_redirects () void
showEditProfile_populatesModel () void
saveEvaluation_redirects() void
updateProfessorProfile_redirects () void
AuthControllerTest
showRegisterForm_returnsRegisterView () void
registerUser_whenNewUser_redirects () void
registerUser_whenUserExists_showsError () void
StudentControllerTest
showStudentDashboard_WithoutProfile () void
showEditStudentProfile () void
updateLogbook_redirects () void
setLookingForTraineeshipTrue_redirects () void
createStudentProfile_redirects () void
updateStudentProfile_redirects () void
showStudentDashboard_WithProfile () void
CompanyControllerTest
saveCompanyProfile_redirectsToDashboard() void
saveEvaluation_redirects() void
showAddPositionForm_returnsView () void
showEditProfile_populatesModel () void
showCompanyDashboard_withProfile () void
updateProfile_redirects () void
showCompanyPositions_returnsPositionsView () void
showEvaluationForm_populatesModel () void
addPosition_redirects () void
showCompanyDashboard_withoutProfile () void
deletePosition_redirects () void
CommitteeControllerTest
showSearchStrategySelection_setsPositionId () void
assignProfessor_successRedirects () void
showSearchStrategies_setsStudentId () void
completeTraineeship_successRedirects () void
assignPositionToStudent_redirects () void
showSearchResults_returnsPositionList () void
showTraineeshipApplicants_populatesModelAndReturnsView () void
viewAssignedTraineeships_populatesModelAndReturnsView () void
committeeDashboard_returnsCorrectView () void
searchProfessors_returnsProfessorsList () void

4.12 Service Test Package

SERVICE TEST PACKAGE


StudentServiceImplTest
testUpdateStudentProfile_UpdatesAndSaves () void
testUpdateLogbook_WhenNoPosition_Throws () void
testRetrieveProfile_WhenExists () void
testGetStudentByUsername_ReturnsStudent () void
testSetLookingForTraineeshipTrue_SetsFlagAndSaves () void
testRetrieveProfile_WhenNotFound_Throws () void
testSaveProfile_CallsSave () void
testHasProfile_WhenExists () void
testHasProfile_WhenNotExists () void
testUpdateLogbook_Success () void
setup () void
UserServiceImplTest
testIsUserPresent_WhenExists () void
testLoadUserByUsername_WhenNotExists_ThrowsException () void
testFindById_WhenNotExists_ThrowsException () void
testIsUserPresent_WhenNotExists () void
testFindUserByUsername_WhenNotExists_ThrowsException () void
testSaveUser_EncodesPasswordAndSaves () void
testFindById_WhenExists () void
setUp () void
testFindUserByUsername_WhenExists () void
testLoadUserByUsername_WhenExists () void
CompanyServiceImplTest
testSaveOrUpdateEvaluation_CreatesNewEvaluation () void
testHasProfile_WhenExists () void
testHasProfile_WhenNotExists () void
testAddTraineeshipPosition_Success () void
testSaveOrUpdateCompanyEvaluation_UpdatesExistingEvaluation () void
testSaveOrUpdateCompanyEvaluation_PositionNotFound_Throws () void
testGetEvaluationData_PositionNotFound_Throws () void
testUpdateCompanyProfile_UpdatesAndSaves () void
testDeleteTraineeship_CallsDao () void
testCreateProfile_SavesCompany () void
testGetCompanyByUsername_ReturnsCompany () void
testGetAvailablePositions_ReturnsList () void
testSaveOrUpdateCompanyEvaluation_CreatesEvaluationIfNotExists () id
setup () void
testGetEvaluationData_ReturnsExistingCompanyEvaluation () void
testGetEvaluationData_NoExistingEvaluation_ReturnsNewEvaluation () id
ProfessorServiceImplTest
testCreateProfile_SavesProfessor () void
testHasProfile_WhenNotExists () void
testGetEvaluationData_CreatesEvaluationIfNotExists () void
testGetEvaluationData_ReturnsExistingEvaluation () void
testSaveOrUpdateProfessorEvaluation_CreatesEvaluationIfNotExists () id
testSaveOrUpdateProfessorEvaluation_UpdatesExistingEvaluation () void
testSaveOrUpdateProfessorEvaluation_WhenPositionNotFound_Throws (
testGetProfessorByUsername_ReturnsProfessor () void
testGetEvaluationData_WhenPositionNotFound_Throws () void
setUp () void
testHasProfile_WhenExists () void
testUpdateProfessorProfile_UpdatesAndSaves () void
CommitteeServiceImplTest
retrieveTraineeshipApplicants_filtersAssigned () void
testGetCompletableFlags_logic () void
testRetrieveTraineeshipApplicants_filtersAssigned () void
testCompleteTraineeship_setsGrades () void
assignProfessorToPosition_successfulAssignment () void
getCompanyEvalFlags_returnsCorrectMap () void
getProfessorEvalFlags_returnsCorrectMap () void
testAssignPositionToStudent_setsRelations () void
setUp () void
assignPositionToStudent_assignsCorrectly () void

4.13 Design

<Specify the detailed design for this release in terms of **UML class diagrams**.>

<Document the classes that are included in this release in terms of CRC cards according to the template that is given below.>

TrainshipApplication

Class Name: TraineeshipAppApplication	
Responsibilities: ▪ Bootstraps and launches the Spring Boot application ▪ Acts as the main entry point (main method) for the application lifecycle ▪ Triggers component scanning and auto-configuration for the entire application	**Collaborations:** ▪ Spring Boot Framework – utilizes @SpringBootApplication to automatically configure the application context ▪ SpringApplication.run(...) – starts the embedded web server and initializes beans, configurations, and dependencies

CONFIG

Class Name: CustomLoginSuccessHandler	
Responsibilities: ▪ Manages the successful authentication of users. ▪ Determines which page (dashboard) the user should be redirected to, based on their role. ▪ Redirects the user to the appropriate URL after login.	**Collaborations:** ▪ HttpServletRequest – to retrieve information from the request. ▪ HttpServletResponse – to send the redirection. ▪ Authentication – to identify the user's roles.

Class Name: CustomSecuritySuccessHandler	
Responsibilities: ▪ Manages the post-login behavior of a user authenticated through Spring Security. ▪ Determines the redirection URL based on the user's roles. ▪ Performs the user redirection using the RedirectStrategy.	**Collaborations:** ▪ HttpServletRequest – to retrieve data from the HTTP request. ▪ HttpServletResponse – to send the redirection. ▪ Authentication – to access the roles of the authenticated user. ▪ GrantedAuthority – to identify the user's roles. ▪ RedirectStrategy (DefaultRedirectStrategy) – to perform the actual redirection.

Class Name: WebMvcConfig	
Responsibilities: ▪ Provides configuration for the MVC layer of the Spring application. ▪ Registers view controllers for specific URL paths without requiring controller methods. ▪ Specifies that the / URL is served by the homepage.html page. ▪	**Collaborations:** • **ViewControllerRegistry – for registering static routes without a controller.** • **Spring WebMvcConfigurer – implements the interface to extend the default MVC configuration.**

Class Name: WebSecurityConfig	
Responsibilities: ▪ Configures Spring Security for the entire application. ▪ Defines URL path authorization based on user roles (student, company, professor, committee). ▪ Sets up login/logout flows and behavior after successful authentication. ▪ Creates a SecurityFilterChain with custom handlers. ▪ Declares beans such as AuthenticationProvider, BCryptPasswordEncoder, and UserDetailsService.	**Collaborations:** ▪ CustomLoginSuccessHandler – for post-login redirection. ▪ CustomSecuritySuccessHandler – (via injection) for an alternative login flow. ▪ UserServiceImpl – as a UserDetailsService for loading user details. ▪ DaoAuthenticationProvider – for handling authentication based on username/password. ▪ HttpSecurity, SecurityFilterChain – for configuring security policies. ▪ BCryptPasswordEncoder – for password encryption. ▪ AuthenticationConfiguration – for retrieving the AuthenticationManager.

CONTROLLERS

Class Name: AuthController	
Responsibilities: ▪ Provides endpoints for user registration (/register, /save). ▪ Creates and displays the user registration form. ▪ Performs validation checks during registration (e.g., empty fields, existing user). ▪ Normalizes usernames before saving. ▪ Calls the UserServiceImpl to store the new user in the system.	**Collaborations:** ▪ UserServiceImpl – for checking if the user exists and saving the new user. ▪ User – domain model representing the registered user. ▪ Model – for adding data to the view (register.html).

Class Name: CommitteeController	
Responsibilities: ▪ Provides the committee member's main dashboard page (/committee_dashboard). ▪ Displays the list of students who have submitted a traineeship application (/Manage_students). ▪ Offers a flow for finding a suitable traineeship position for a student, based on strategy: interests, location, or both. ▪ Displays evaluated available positions (/Search_results). ▪ Assigns a traineeship position to a student (/assign_position_to_student). ▪ Displays active traineeships supervised by the committee (/In_progress_traineeships). ▪ Provides a flow for finding and assigning a professor as a traineeship supervisor (/Search_professor, /Assign_professor_selection, /Assign_professor). ▪ Completes the traineeship once the required evaluations have been submitted (/Complete_traineeship).	**Collaborations:** ▪ CommitteeService – contains all logic for assignment, search, and completion. ▪ Student – domain model for students who have applied. ▪ TraineeshipPosition – model representing available positions. ▪ PositionWithScoreDto – DTO for matched positions with relevance scores. ▪ ProfessorMatchDto – DTO for professors suggested for supervision. ▪ Model – for passing data to HTML views. ▪ RedirectAttributes – for sending messages after POST actions.

Class Name: CompanyController	
Responsibilities: ▪ Provides the company's dashboard page (/company_dashboard). ▪ Checks whether the company has created a profile and displays the appropriate page (dashboard or create profile). ▪ Manages the creation and editing of the company profile. ▪ Displays the list of the company’s traineeship positions (/Company_Positions). ▪ Allows the creation, saving, and deletion of traineeship positions. ▪ Provides a student evaluation form and stores the evaluation (motivation, efficiency, effectiveness).	**Collaborations:** ▪ CompanyService – business logic for managing company profiles, positions, and evaluations. ▪ CompanyDto, TraineeshipPositionDto – DTOs for transferring data to/from views. ▪ Company, TraineeshipPosition – domain models for the company and its traineeship positions. ▪ TraineeshipPositionMapper – for converting DTOs to entities. ▪ ProfessorMatchDto – DTO for professors suggested for supervision. ▪ Model, Principal, RequestParam, PathVariable – Spring MVC components used for routing, binding, and injection.

Class Name: ProfessorController	
Responsibilities: ▪ Provides the professor's dashboard (/professor_dashboard) and checks if a profile has already been created. ▪ Manages the creation and editing of the professor's profile (name, interests). ▪ Displays the traineeship positions supervised by the professor (/professor_positions). ▪ Provides a form for evaluating both the student and the company for a specific position. ▪ Saves or updates the professor’s evaluations.	**Collaborations:** ▪ ProfessorService – for business logic related to the professor’s profile and evaluations. ▪ Professor, ProfessorDto – domain model and DTO for the professor's profile. ▪ Model, Principal, RequestParam, PathVariable – Spring MVC components used for routing and injection.

Class Name: StudentController	
Responsibilities: ▪ Provides the student dashboard and checks whether the student has already created a profile (/student_dashboard). ▪ Manages the creation of a new student profile, including uniqueness validation for the University ID. ▪ Allows editing of an existing student profile. ▪ Displays the logbook form and saves the entry. ▪ Enables the student to apply for a traineeship position (/Apply_for_traineeship). ▪ Communicates with the StudentService for all operations involving the storage, retrieval, and update of student data.	**Collaborations:** ▪ StudentService – handles business logic related to profiles, traineeship applications, and logbook entries. ▪ Student, StudentDto – domain model and DTO representing the student. ▪ Principal – used to obtain the username of the currently authenticated user. ▪ Model, RedirectAttributes – for communication between the controller and the view (HTML templates).

DOMAIN MODEL

Class Name: Company	
Responsibilities: ▪ Represents the company entity in the database. ▪ Stores key identification and location details of the company. ▪ Is associated with the traineeship positions posted by the company.	**Collaborations:** ▪ TraineeshipPosition – a company is related to multiple traineeship positions through a one-to-many relationship. ▪ JPA/Hibernate – for mapping the entity to the companies table in the database.

Class Name: Evaluation	
Responsibilities: ▪ Represents an evaluation (by a professor or a company) related to a traineeship position. ▪ Stores scores in key fields such as motivation, efficiency, effectiveness, facilities, and guidance. ▪ Differentiates the type of evaluation using the evaluationType (e.g., PROFESSOR_EVALUATION or COMPANY_EVALUATION). ▪ Is associated with a single traineeship position (many-to-one relationship with TraineeshipPosition).	**Collaborations:** ▪ TraineeshipPosition – each evaluation is linked to one traineeship position via @ManyToOne. ▪ EvaluationType – enum that defines the type of evaluation (professor or company). ▪ JPA/Hibernate – for persistence and mapping to the evaluations table.

Class Name: Professor	
Responsibilities: ▪ Represents a professor in the system with a unique username. ▪ Stores the professor's name and areas of interest. ▪ Is associated with the traineeship positions they supervise. ▪ Supports supervision assignment with filtering based on interests.	**Collaborations:** ▪ TraineeshipPosition – a list of traineeship positions supervised by the professor. ▪ JPA/Hibernate – for mapping the entity to the professors and professor_interests tables. ▪ JPA/Hibernate – for persistence and mapping to the evaluations table.

Class Name: Student	
Responsibilities:	**Collaborations:**
▪ Represents a student user in the system ▪ Stores key student data such as full name, username, university ID (AM), average grade, preferred location ▪ Maintains lists of interests and skills for matching with traineeship positions ▪ Indicates whether the student is actively looking for a traineeship ▪ Links the student to a single assigned traineeship position (if any)	▪ TraineeshipPosition – via a one-to-one relationship representing the position assigned to the student ▪ JPA/Hibernate – for entity mapping to the students table and related collections (student_interests, student_skills) ▪ Spring Data – used in services and repositories to search/filter students

Class Name: TraineeshipPosition	
Responsibilities:	**Collaborations:**
▪ Represents a traineeship position offered by a company ▪ Stores key details such as title, description, duration (fromDate to toDate), required skills, and topics ▪ Links the traineeship to a student, a supervising professor, and a company ▪ Tracks whether the position is assigned (isAssigned) and holds the student's logbook and pass/fail result ▪ Maintains the list of evaluations (from company and professor) associated with the position	▪ Student – via one-to-one relationship, representing the student assigned to the traineeship ▪ Professor – the supervising professor (many positions can share one professor) ▪ Company – the company offering the position (many positions can belong to one company) ▪ Evaluation – a list of evaluations connected to this traineeship ▪ JPA/Hibernate – for ORM mapping of all relationships and fields

Class Name: User	
Responsibilities: ▪ Represents a system user (Student, Company, Professor, or Committee Member) ▪ Stores authentication credentials: username, password, and role ▪ Implements UserDetails to integrate with Spring Security authentication ▪ Provides user authorities based on their role (ROLE_STUDENT, ROLE_COMPANY, etc.) ▪ Defines user account status (non-expired, non-locked, etc.) with always-true values for simplicity	**Collaborations:** ▪ Role – enum representing the user's role in the system ▪ Spring Security (UserDetails, GrantedAuthority, SimpleGrantedAuthority) – used for authentication and authorization ▪ JPA/Hibernate – maps the entity to the users database table ▪ Other domain models (Student, Company, Professor) indirectly relate by sharing the username as foreign key logic

Dto

Class Name: CompanyDto	
Responsibilities: ▪ Serves as a Data Transfer Object (DTO) for transferring company-related data between layers (e.g., controller ↔ service) ▪ Holds basic company information: username, companyName, and companyLocation ▪ Optionally includes a list of the company's traineeship positions in DTO form	**Collaborations:** ▪ TraineeshipPositionDto – used to represent the positions associated with the company ▪ Lombok (@Data, @NoArgsConstructor, @AllArgsConstructor) – automatically generates boilerplate code like getters, setters, constructors, equals(), and hashCode() ▪ CompanyController, CompanyService – classes that use this DTO for handling input/output related to company profile operations

Class Name: PositionWithScoreDto	
Responsibilities: ▪ Acts as a Data Transfer Object to represent a traineeship position along with a computed compatibility score ▪ Stores a reference to the TraineeshipPosition and associated match details ▪ Contains individual scoring breakdown: number of matched skills, interests, whether the location matches, and total possible values ▪ Used to rank and present suitable positions to committee members during the assignment process	**Collaborations:** ▪ TraineeshipPosition – the position being evaluated and matched ▪ Matching & scoring logic in the service layer (CommitteeService) that calculates how well a position fits a student ▪ Controllers (e.g., CommitteeController) – for rendering position match results to the UI

Class Name: ProfessorDto	
Responsibilities: ▪ Serves as a Data Transfer Object (DTO) for professor-related data (e.g., during profile creation or update) ▪ Stores basic professor details: username, professorName, and list of interests ▪ Optionally includes a list of supervised traineeship positions in DTO form ▪ Supports both minimal and full representations depending on context (form input, view rendering, etc.)	**Collaborations:** ▪ TraineeshipPositionDto – used to represent positions supervised by the professor ▪ ProfessorController, ProfessorService – utilize this DTO to manage and update professor profiles

Class Name: ProfessorMatchDto	
Responsibilities:	**Collaborations:**
▪ Acts as a Data Transfer Object to represent a professor and their compatibility with a traineeship position ▪ Holds reference to a Professor along with match metrics: ▪ Number of matched interests ▪ Used to rank or display professors based on suitability for assignment	▪ Professor – the entity representing the candidate supervisor ▪ CommitteeService – which performs the matching logic and creates this DTO ▪ CommitteeController – which uses the DTO to display professor match options to the user

Class Name: StudentDto	
Responsibilities:	**Collaborations:**
▪ Serves as a Data Transfer Object (DTO) for student-related data during registration, profile editing, and application processes ▪ Stores student information: username, studentName, university ID (am), average grade, preferred location ▪ Maintains lists of the student's interests and skills for use in matching traineeship positions ▪ Tracks whether the student is currently seeking a traineeship ▪ Provides constructors for both full and simplified creation, depending on the context	▪ StudentController, StudentService – use this DTO to create or update student profiles ▪ DTO consumers (e.g., matching logic, view models) that avoid coupling to the full Student entity

Class Name: TraineeshipPositionDto	
Responsibilities:	**Collaborations:**
▪ Represents the data of a traineeship position in a form suitable for form handling, view rendering, or business logic without exposing the full entity ▪ Holds basic information about a traineeship position: title, description, start and end dates, topics, skills, and whether it is assigned ▪ Facilitates safe data transfer between controller, service, and view layers	▪ Used by CompanyController, CommitteeController, and TraineeshipPositionMapper to create or display traineeship positions ▪ May be converted to or from a TraineeshipPosition entity for persistence or evaluation purposes

Mappers

Class Name: CompanyMapper	
Responsibilities:	**Collaborations:**
▪ Converts a Company entity to a CompanyDto (toDto) for data transfer to the view or controller layer ▪ Converts a CompanyDto to a Company entity (toEntity) for saving to or retrieving from the database ▪ Ensures separation of concerns by avoiding direct use of entities in controller/view logic	▪ Company – domain entity used in the persistence layer ▪ CompanyDto – Data Transfer Object used for presenting or capturing company-related data ▪ Used by controllers or services (e.g., CompanyController, CompanyService) to map data between layers

Class Name: ProfessorMapper	
Responsibilities:	**Collaborations:**
▪ Converts a Professor entity into a ProfessorDto for safe data transfer to views or external layers ▪ Converts a ProfessorDto into a Professor entity for storage and persistence ▪ Ensures a clear separation between domain models and presentation or transport layers	▪ Professor – domain entity representing the professor in the database ▪ ProfessorDto – Data Transfer Object used for transferring professor data between controller, view, and service layers ▪ Used by ProfessorService and ProfessorController for managing profile data input/output

Class Name: StudentMapper	
Responsibilities:	**Collaborations:**
▪ Converts a Student entity into a StudentDto for use in controllers, views, or APIs ▪ Converts a StudentDto into a Student entity for database operations ▪ Preserves the separation of concerns between the domain and presentation layers ▪ Ensures safe handling of user-submitted data during registration, editing, or matching	▪ Student – domain model that maps to the database and represents student data ▪ StudentDto – Data Transfer Object used to transfer student data across layers ▪ Used by StudentController and StudentService for creating, updating, or displaying student profile data

Class Name: TraineeshipPositionMapper	
Responsibilities:	**Collaborations:**
▪ Converts a TraineeshipPositionDto to a TraineeshipPosition entity for use in persistence and domain logic ▪ Converts a TraineeshipPosition entity to a TraineeshipPositionDto for use in views, forms, or data transfer ▪ Safely maps assignment status and basic metadata (e.g., title, description, dates, skills, topics) ▪ Handles null checks for the isAssigned flag when converting from DTO to entity	▪ TraineeshipPosition – the entity representing a traineeship in the database ▪ TraineeshipPositionDto – a DTO used for data transfer between backend and frontend layers ▪ Used in CompanyService, CommitteeService, CompanyController, and CommitteeController when adding or displaying positions

Services

Class Name: CommitteeServiceImpl	
Responsibilities:	**Collaborations:**
▪ Retrieves all students who have applied for a traineeship but haven't yet been assigned ▪ Executes position matching for students based on various strategies (interests, location, or both) ▪ Assigns students to traineeship positions and professors to supervise them ▪ Determines if a position is ready for completion (based on submitted evaluations) ▪ Completes the traineeship process by finalizing the result (pass/fail) and calculating the student's average grade ▪ Supplies evaluation flags and supervision options to the controller/view layer	▪ StudentDao, ProfessorDao, TraineeshipPositionDao – for database access to students, professors, and positions ▪ PositionsSearchFactory, PositionsSearchStrategy – used to apply different student-to-position matching strategies ▪ SupervisorAssignmentFactory, SupervisorAssignmentStrategy – for dynamic professor recommendation logic ▪ MatchingUtils – utility class used for calculating matches (skills, interests, location, total score) ▪ PositionWithScoreDto, ProfessorMatchDto – DTOs used to represent matching results ▪ Evaluation, EvaluationType – used for

	logic around completion and validation of the evaluation process

Class Name: CompanyServiceImpl	
Responsibilities:	**Collaborations:**
▪ Creates and updates company profiles using data from CompanyDto ▪ Manages the list of traineeship positions created by a company ▪ Adds, deletes, or retrieves available positions by company username ▪ Provides and persists company evaluations for assigned students ▪ Retrieves evaluation data for display in views ▪ Ensures that new evaluations are correctly attached to their respective positions	▪ CompanyDao, TraineeshipPositionDao, EvaluationDao – used to access and persist entities ▪ Company, TraineeshipPosition, Evaluation – domain models representing company-related data and relationships ▪ CompanyDto, TraineeshipPositionDto – data transfer objects used between UI and service layers ▪ CompanyMapper, TraineeshipPositionMapper – handle conversion between entities and DTOs ▪ EvaluationType – enum used to distinguish company vs. professor evaluations

Class Name: ProfessorServiceImpl	
Responsibilities:	**Collaborations:**
▪ Creates and updates professor profiles based on data received from ProfessorDto ▪ Retrieves professor data by username and checks if a professor profile exists ▪ Retrieves and prepares evaluation data for a traineeship position supervised by the professor ▪ Creates or updates professor evaluations (motivation, efficiency, effectiveness, facilities, guidance) ▪ Ensures that evaluation objects are properly initialized and attached to their corresponding traineeship positions	▪ ProfessorDao, TraineeshipPositionDao, EvaluationDao – for persistence and retrieval of professors, positions, and evaluations ▪ Professor, TraineeshipPosition, Evaluation, EvaluationType – domain models manipulated and updated by the service ▪ ProfessorDto – DTO used for transferring profile data ▪ ProfessorMapper – converts between DTOs and domain models ▪ Controller classes (ProfessorController) – which delegate business logic to this service

Class Name: StudentServiceImpl	
Responsibilities:	**Collaborations:**
▪ Creates and updates student profiles based on StudentDto input ▪ Retrieves students by username or university ID (AM) ▪ Updates a student's assigned traineeship logbook ▪ Sets the student's "looking for traineeship" status ▪ Provides access to student profile information for view and logic layers ▪ Ensures correct linkage between a student and their assigned traineeship position	▪ StudentDao – for interacting with the persistence layer (student repository) ▪ Student, TraineeshipPosition – domain entities used and updated by the service ▪ StudentDto – data transfer object used for input/output during profile creation and updates ▪ StudentMapper – maps between DTO and entity representations ▪ StudentController – controller that delegates profile and logbook operations to this service

Class Name: UserServiceImpl	
Responsibilities:	**Collaborations:**
▪ Implements user registration and persistence via saveUser, including password encryption ▪ Loads user details for Spring Security authentication (UserDetailsService) ▪ Checks for existing users by normalized username (isUserPresent) ▪ Retrieves user entities by username ▪ Ensures secure integration with Spring Security using BCryptPasswordEncoder	▪ UserDao – for database access to user records ▪ User – domain model representing a system user with roles ▪ BCryptPasswordEncoder – used to encrypt passwords before storage ▪ Spring Security – integrates with the framework via UserDetailsService for login/authentication ▪ Controllers like AuthController – delegate registration and login logic to this service

Strategies - PositionSearch

Class Name: CompositeSearch	
Responsibilities:	**Collaborations:**
▪ Implements the PositionsSearchStrategy interface for student-to-position matching Retrieves all traineeship positions and filters them based on: ▪ Whether the position is already assigned ▪ Whether the student's skills match all required skills ▪ Whether the position's location matches the student's preferred location ▪ Whether the student's interests match the position's topics (Jaccard similarity) ▪ Checks for existing users by normalized username (isUserPresent) ▪ Retrieves user entities by username ▪ Ensures secure integration with Spring Security using BCryptPasswordEncoder ▪ Computes semantic similarity using Jaccard index to enhance the interest-topic matching	▪ TraineeshipPositionDao, StudentDao – retrieve required domain data (positions and student profiles) ▪ Student, TraineeshipPosition – domain models representing student and position entities ▪ MatchingUtils – utility class used to verify required skill coverage ▪ PositionsSearchFactory – may instantiate this class as one of the available strategies

Class Name: CompositeSearch	
Responsibilities:	**Collaborations:**
▪ Provides the appropriate implementation of PositionsSearchStrategy based on the selected strategy name (e.g., "interests", "location", "both") ▪ Encapsulates strategy selection logic to support the Strategy Design Pattern ▪ Facilitates loose coupling by abstracting away concrete implementations of search logic	▪ SearchBasedOnInterests, SearchBasedOnLocation, CompositeSearch – concrete strategy classes for filtering positions ▪ PositionsSearchStrategy – the strategy interface all search implementations conform to ▪ Used by CommitteeServiceImpl to dynamically assign the search behavior at runtime based on user input

Class Name: SearchBasedOnInterests	
Responsibilities:	**Collaborations:**
▪ Implements the PositionsSearchStrategy interface ▪ Searches for traineeship positions that match a student's interests using **Jaccard similarity** ▪ Filters out already assigned positions and those where the student lacks required skills ▪ Ensures topic-based relevance by comparing position topics with student interests	▪ TraineeshipPositionDao, StudentDao – fetches data required for filtering logic ▪ Student, TraineeshipPosition – domain models for input and matching context ▪ MatchingUtils – used to verify whether the student meets the required skill criteria ▪ Used by PositionsSearchFactory to provide interest-based search logic

Class Name: SearchBasedOnLocation	
Responsibilities: ▪ Implements the PositionsSearchStrategy interface ▪ Filters available traineeship positions based on whether the location matches the student's preferred location ▪ Ensures that the student also meets all required skills for the position ▪ Returns only unassigned and location-matching positions	**Collaborations:** ▪ StudentDao, TraineeshipPositionDao – retrieves the relevant student and all available positions ▪ Student, TraineeshipPosition – entities involved in filtering ▪ MatchingUtils – used to verify skill compatibility ▪ Selected by PositionsSearchFactory when the "Search by Location" strategy is chosen

Strategies – SupervisorSearch

Class Name: AssignmentBasedOnInterests	
Responsibilities: ▪ Implements the SupervisorAssignmentStrategy interface ▪ Selects suitable professors whose interests closely match the topics of a traineeship position ▪ Calculates the similarity between professor interests and position topics using a simple ratio ▪ Returns a list of professors whose similarity exceeds a defined threshold, along with their current supervision load	**Collaborations:** ▪ ProfessorDao – used to retrieve all available professors ▪ Professor, TraineeshipPosition – domain entities used for matching ▪ ProfessorMatchDto – encapsulates the result of the matching logic (interests matched, load, etc.) ▪ Used by SupervisorAssignmentFactory to provide "interest-based" professor assignment strategy

Class Name: AssignmentBasedOnLoad	
Responsibilities:	**Collaborations:**
▪ Implements the SupervisorAssignmentStrategy interface ▪ Identifies professors with the **lowest number of current supervised positions** ▪ Breaks ties by returning **all professors** with the same minimum load ▪ Enhances results by including information about topic-interest alignment for further evaluation ▪ Returns a list of ProfessorMatchDto containing interest match count, total topics, and supervision load	▪ ProfessorDao – used to retrieve all professor records ▪ Professor, TraineeshipPosition – domain entities being matched ▪ ProfessorMatchDto – used to represent the matching result and load ▪ Used by SupervisorAssignmentFactory when the load-based matching strategy is requested

Class Name: SupervisorAssignmentFactory	
Responsibilities:	**Collaborations:**
▪ Provides the appropriate SupervisorAssignmentStrategy implementation based on a given input ("interests" or "load") ▪ Encapsulates the logic for selecting how professors are matched to positions ▪ Ensures that the correct strategy instance is returned dynamically based on user or system input	▪ AssignmentBasedOnInterests, AssignmentBasedOnLoad – concrete implementations of the strategy interface ▪ SupervisorAssignmentStrategy – strategy interface implemented by all matching algorithms ▪ CommitteeServiceImpl – uses this factory to retrieve the correct strategy during professor assignment

Utils

Class Name: MatchingUtils	
Responsibilities:	**Collaborations:**
▪ Counts how many of a position's required **skills** and **interests** match those of a student ▪ Determines whether a position's **location** matches the student's preferred location ▪ Calculates a **total matching score** for ranking traineeship positions (skills: 50%, interests: 30%, location: 20%) ▪ Checks if a student **fully satisfies** all required skills of a traineeship position	▪ Student, TraineeshipPosition – domain entities used as inputs for all comparison and scoring methods ▪ CommitteeServiceImpl(e.g., getScoredPositionsForApplicant) ▪ Strategy classes (e.g., CompositeSearch, SearchBasedOnInterests) for filtering logic