Traineeship Application

Sprint Report

GIANNIS FILLIS, AM:5380
KONSTANTINOS ZOIS, AM: 5226

CONTENTS

VERSIONS HISTORY

Date	Version	Description	Author
8-3-25	1.0	First draft of the use cases	Konstantinos Zois, Ioannis Fillis
16-3-25	1.1	Filled in the remaining use cases, added the first version of the use cases UML	Konstantinos Zois, Ioannis Fillis
8-5-25	2.0	Fixes and additions to the use cases, added the sprints information	Konstantinos Zois, Ioannis Fillis
15-5-25	2.1	Updated the use cases UML, added the CRC Cards	Konstantinos Zois, Ioannis Fillis
24-5-25	2.2	Added the UML package and class diagrams	Konstantinos Zois, Ioannis Fillis
26-5-25	3.0	Final fixes	Konstantinos Zois, Ioannis Fillis

1 Introduction

This document provides information concerning the **<4>** sprint of the project.

1.1 Purpose

The objective of this project is to develop an application that allows the traineeship committee of the University to manage and monitor open and assigned traineeship positions. Specifically, the application shall allow companies to announce open traineeship positions. The students will be able to look for available traineeship positions. The traineeship committee shall assign positions to students via different alternative criteria. The traineeship committee will further allocate professors as supervisors to the assigned traineeship positions. Professors and companies will be responsible for the final evaluations of the students' traineeships.

1.2 Document Structure

The rest of this document is structured as follows. Section 2 describes our Scrum team and specifies this Sprint's backlog. Section 3 and 4 specify the main design concepts for this release of the project.

2 Scrum team and Sprint Backlog

2.1 Scrum team

Product Owner	Ioannis Fillis
Scrum Master	Konstantinos Zois
Development Team	Konstantinos Zois, Ioannis Fillis

2.2 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories
1	**26-3-25**	**9-4-25**	**2**	**US1, US2, US3**
2	**10-4-25**	**17-4-25**	**1**	**US4, US5, US7, US8, US10, US11, US13**

3	**18-4-25**	**25-4-25**	**1**	**US9, US14, US16, US17, US18, US20**
4	**26-4-25**	**2-5-25**	**1**	**US6, US12, US15, US19, US21**

3 Use Cases

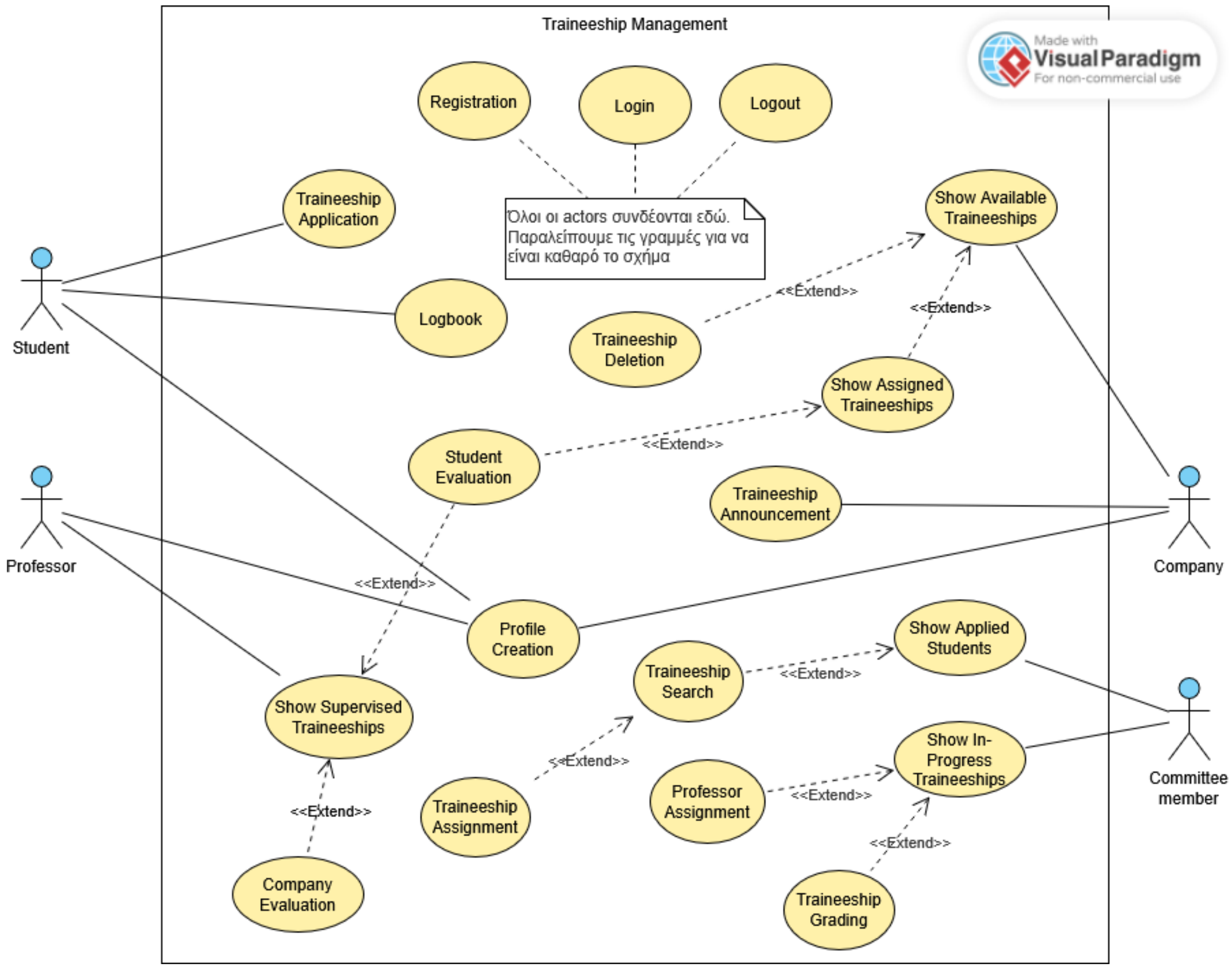


FIGURE 1 UML Use Case Diagram

3.1 USE CASE 1: User registration

Use case ID	UC1
Actors	Student, Professor, Company, Committee
Pre conditions	1. The user is connected to the website
Main flow of events	1. The use case starts when the user presses the “register” button 2. He fills in all the required information
Post conditions	1. The database is updated with a new user

3.2 USE CASE 2: User login

Use case ID	UC2
Actors	Student, Professor, Company, Committee
Pre conditions	1. The user is connected to the website
Main flow of events	1. The use case starts when the user presses the “login” button 2. He fills in his username and password
Alternative flow	1. If the information is wrong, he is denied access and is required to fill in his username and password
Post conditions	The user enters the main page

3.3 USE CASE 3: User log out

Use case ID	UC3
Actors	Student, Professor, Company, Committee
Pre conditions	1. The user is logged in to his account
Main flow of events	1. The use case starts when the user presses the “log out” button 2. He exits the main page(dashboard)
Post conditions	1. The user terminates his interaction with the application

3.4 USE CASE 4: Student profile creation

Use case ID	UC4
Actors	The student
Pre conditions	1. The student is connected to the website
Main flow of events	1. The use case starts when the user presses the "Profile" button 2. The student fills in the following fields: full name, university ID, interests, skills, and preferred traineeship location. 3. The student submits the profile form by pressing the "Save" button
Alternative flow 1	1. At any time, the student may return to the dashboard
Alternative flow 2	1. If any required fields are missing, the system prompts the student to complete them
Post conditions	1. The database is updated

3.5 USE CASE 5: Traineeship application

Use case ID	UC5
Actors	The student
Pre conditions	1. The student is logged in to his account 2. The student has filled in his profile information
Main flow of events	1. The use case starts when the user presses the "I am looking for a Traineeship" button 2. A message that the application is set appears
	1. If the student has already applied for a position, a warning message is displayed and the use case is terminated 2. If the student is assigned to a position, a warning message is displayed and the use case is terminated
Post conditions	1. The application is submitted and the database gets updated

3.6 USE CASE 6: Fill logbook

Use case ID	UC6
Actors	The student
Pre conditions	1. The student is logged in to his account 2. The student has been accepted to a traineeship position
Main flow of events	1. The use case starts when the student presses the “My Logbook” button 2. All the entries are displayed 3. If there are no entries 3.1. A message “No entries yet” is displayed 4. The student fills out all the information inside the “Fill LogBook” prompt 5. The student submits the logbook by pressing the “Save” button
Alternative flow 1	1. At any time, the student may return to the dashboard
Alternative flow 2	1. If the entry size is beyond the allowed length, a message is displayed
	1. If the student is not assigned to a traineeship position 1.1. A warning message is displayed 1.2. The use case is terminated
Post conditions	1. The logbook is submitted and the database gets updated

3.6.1 USE CASE 6.1: Erase logbook

Use case ID	UC6.1
Actors	The student
Pre conditions	1. The student is logged in to his account 2. The student has been accepted to a traineeship position
Main flow of events	1. The use case starts when the student presses the “Erase Logbook” button

	2. All the entries in the logbook are erased
Alternative flow	1. At any time, the student may return to the dashboard
Post conditions	1. The logbook is erased and the database gets updated

3.7 USE CASE 7: Company profile creation

Use case ID	UC7
Actors	The company
Pre conditions	1. The company is connected to the website
Main flow of events	1. The use case starts when the company presses the "Profile" button 2. The company fills in the information about the company name and location 3. The company submits the form by pressing the "Save" button
Alternative flow 1	1. At any time, the company may return to the dashboard
Alternative flow 2	1. If any required fields are missing, the system prompts the company to complete them
Post conditions	1. The database gets updated

3.8 USE CASE 8: List of available traineeship positions

Use case ID	UC8
Actors	The company
Pre conditions	1. The company is logged into the system 2. The company has created the profile 3. The company has advertised traineeship positions

Main flow of events	1. The use case starts when the company presses the "My Offered Positions" button 2. If the company has not advertised any positions 2.1. The position list is empty 3. The system retrieves and displays the list of available traineeship positions posted by the company
Alternative flow	1. At any time, the company may return to the dashboard
Post conditions	1. The list of the company's traineeships is retrieved from the database

3.9 USE CASE 9: List of assigned traineeship positions

Use case ID	UC9
Actors	The company
Pre conditions	1. The company is logged into the system 2. The company has created the profile 3. The company has advertised traineeship positions
Main flow of events	1. The use case starts when the user presses the "Assigned Positions" button from the available positions page 2. If none of the positions are assigned 2.1. The position list is empty 3. The system retrieves and displays the list of traineeship positions assigned to students
Alternative flow	1. At any time, the company may return to the dashboard
Post conditions	1. The list of the company's applied traineeships is retrieved from the database

3.10 USE CASE 10: Traineeship position announcement

Use case ID	UC10
Actors	The company
Pre conditions	1. The company is logged into the system 2. The company has created the profile
Main flow of events	1. The use case starts when the company presses the "Add Traineeship Offer" button from the offered positions page 2. The company fills in the information about the start and end dates, short description about the work, the list of the required skills, list of related topics of interest 3. The company submits the post by pressing the "Save" button
Alternative flow 1	1. If any required fields are missing, the system prompts the company to complete them
Alternative flow 2	1. If the entry of the description size is beyond the allowed length, a warning message is displayed
Alternative flow 3	1. At any time, the company may return to the dashboard
Post conditions	1. The database gets updated

3.11 USE CASE 11: Traineeship position deletion

Use case ID	UC11
Actors	The company
Pre conditions	1. The company is logged into the system 2. The company has created the profile 3. The company has posted a traineeship position
Main flow of events	1. The use case starts when the company presses the "Delete Position" button for a specific traineeship position in the offered positions list 2. The position gets deleted and the company returns to the dashboard

Post conditions	1. The database gets updated

3.12 USE CASE 12: Traineeship evaluation by the company

Use case ID	UC12
Actors	The company
Pre conditions	1. The company is logged into the system 2. The company has created the profile 3. The company has posted a traineeship position 4. The company has assigned trainees
Main flow of events	1. The use case starts when the user presses the “Evaluate” on a specific traineeship from the list of assigned positions 2. The company fills in the rating of the motivation, effectiveness and efficiency of the student on a scale of 1 to 5. 3. The company submits the evaluation by pressing the “Save” button
Alternative flow 1	1. If any required fields are missing or an invalid number is given, the system prompts the company to complete them
Alternative flow 2	1. At any time, the company may return to the dashboard or the assigned positions page
Alternative flow 3	1. If the company has already evaluated the specific position 1.1. A warning message is displayed 1.2. The use case is terminated
Post conditions	1. The database gets updated

3.13 USE CASE 13: Professor profile creation

Use case ID	UC13
Actors	The professor
Pre conditions	1. The professor is connected to the website
Main flow of events	1. The use case starts when the professor presses the “Profile” button 2. The professor fills in the information about his name and a list of interests 3. The professor submits the form by pressing the “Save” buttom
Alternative flow 1	1. At any time, the company may return to the dashboard
Alternative flow 2	1. If any required fields are missing, the system prompts the professor to complete them
Post conditions	1. The database gets updated

3.14 USE CASE 14: List of supervised traineeship positions

Use case ID	UC14
Actors	The professor
Pre conditions	1. The professor is logged into the system 2. The professor has created the profile 3. The professor is supervising a traineeship position
Main flow of events	1. The use case starts when the professor presses the “Show Supervising Positions” button 2. If the professor is not supervising any positions 2.1. The positions list is empty 3. The system retrieves and displays the list of traineeship positions supervised by the professor
Alternative flow	1. At any time, the professor may return to the dashboard

Post conditions	1. The list of traineeship positions supervised by the professor are retrieved from the database

3.15 USE CASE 15: Traineeship evaluation by the professor

Use case ID	UC15
Actors	The professor
Pre conditions	1. The professor is logged into the system 2. The professor has created the profile 3. The professor is supervising a traineeship position
Main flow of events	1. The use case starts when the user presses the “Evaluate Student” or the “Evaluate Company” button on a specific traineeship position from the supervising position list 2. If the professor selects “Evaluate Student” 2.1. The professor is prompted to fill in the rating of the motivation, effectiveness and efficiency of the student on a scale of 1 to 5 2.2. The professor submits the student evaluation by pressing the “Save” button 3. If the professor selects “Evaluate Company” 3.1. The professor is prompted to fill in the rating of the facilities and guidance of the company on a scale of 1 to 5 3.2. The professor submits the company evaluation by pressing the “Save” button
Alternative flow 1	1. If any required fields are missing or an invalid number is given, the system prompts the professor to complete them
Alternative flow 2	1. If the professor has already filled in any of the evaluations 1.1. A warning message is displayed 1.2. The use case is terminated
Alternative flow 3	1. At any time, the professor may return to the dashboard or the supervising positions page
Post conditions	1. The database gets updated

3.16 USE CASE 16: List of applied students

Use case ID	UC16
Actors	The traineeship committee member
Pre conditions	1. The committee member is logged into the system
Main flow of events	1. The use case starts when the committee member presses the “Show Applied Students” button 2. If no student has applied 2.1. The student list is empty 3. The system retrieves and displays the list of students applied for a traineeship position
Alternative flow	1. At any time, the committee may return to the dashboard
Post conditions	1. The list of applied students is retrieved from the database

3.17 USE CASE 17: Traineeship position search

Use case ID	UC17
Actors	The traineeship committee member
Pre conditions	1. The committee member is logged into the system 2. There is at least one student applied for a traineeship
Main flow of events	1. The use case starts when the committee member presses the “Available Positions” button on a specific student from the list of applied students 2. The system displays a menu with 3 options: “Interests Based”, “Location Based” and “Both” 3. The committee member selects one of three options 4. The system retrieves and displays a list of available positions based on the selected search that matches at least 2 student’s skills
Alternative flow	1. At any time, the committee may return to the applied students page

Post conditions	1. The list of traineeship positions matching the selected option is retrieved from the database

3.18 USE CASE 18: Traineeship position assignment

Use case ID	UC18
Actors	The traineeship committee member
Pre conditions	1. The committee member is logged into the system 2. The system successfully searched for matching positions for a specific student
Main flow of events	1. The use case starts when the committee member presses the “Assign” button on a specific traineeship position from the list 2. The position is assigned to the student
Alternative flow	1. At any time, the committee may return to the dashboard
Post conditions	1. The database gets updated

3.19 USE CASE 19: Supervising professor assignment

Use case ID	UC19
Actors	The traineeship committee member
Pre conditions	1. The committee member is logged into the system 2. There is at least one in-progress traineeship
Main flow of events	1. The use case starts when the committee member presses the “Assign Supervisor” button on a specific position from the list 2. The system displays a menu with 2 options: “Load Based” and “Interests Based” 3. The committee member selects one of two options 4. The committee submits the selected method by pressing the “Assign” button 5. The system assigns the professor based on the selected assignment

Alternative flow 1	1. At any time, the committee may return to the in-progress positions list
Alternative flow 2	1. If the specific traineeship position has already a supervisor 1.1. A warning message is displayed 1.2. The use case is terminated
Post conditions	1. The database gets updated

3.20 USE CASE 20: List of in-progress traineeships

Use case ID	UC20
Actors	The traineeship committee member
Pre conditions	1. The committee member is logged into the system
Main flow of events	1. The use case starts when the committee member presses the "Show In-Progress Positions" button 2. If there are no in-progress traineeship positions 2.1. The positions list is empty 3. The system retrieves and displays the list of traineeships that are in progress
Alternative flow	1. At any time, the committee may return to the dashboard
Post conditions	1. The list of in-progress traineeship positions is retrieved from the database

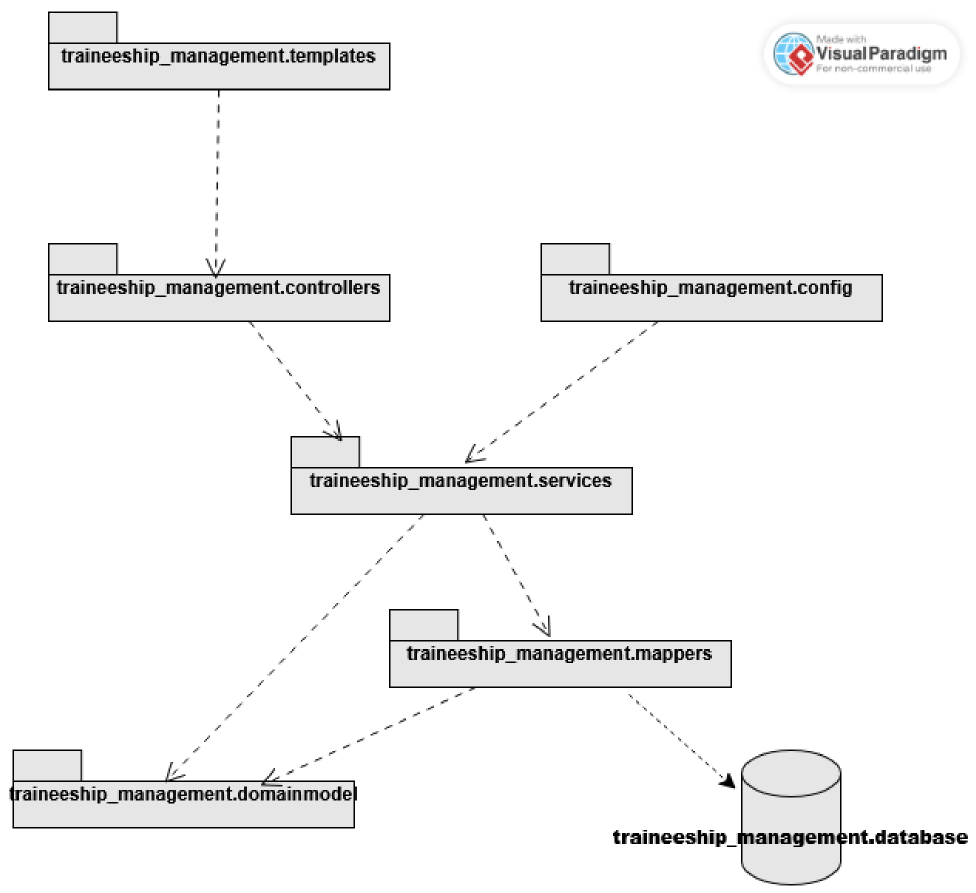
3.21 USE CASE 21: Traineeship position grade

Use case ID	UC21
Actors	The traineeship committee member
Pre conditions	1. The committee member is logged into the system 2. There is at least one in-progress traineeship

Main flow of events	1. The use case starts when the committee member presses the “Grade Position” button on a specific position from the in-progress positions list 2. The system retrieves and displays the student and company evaluations of the selected position 3. The committee member marks the process with pass or fail in the “Final Grade” prompt 4. The committee submits the process by pressing the “Save” button
Alternative flow 1	1. At any time, the committee may return to the dashboard
Alternative flow 2	1. If the evaluations of the specific position are still pending 1.1. A warning message is displayed 1.2. The use case is terminated
Post conditions	1. The database gets updated

4 Design

4.1 Architecture



traineeship_management.templates
Made with Visual Paradigm For non-commercial use
traineeship_management.controllers
traineeship_management.config
traineeship_management.services
traineeship_management.mappers
traineeship_management.domainmodel
traineeship_management.database

FIGURE 2 UML Package Diagram

4.2 Design

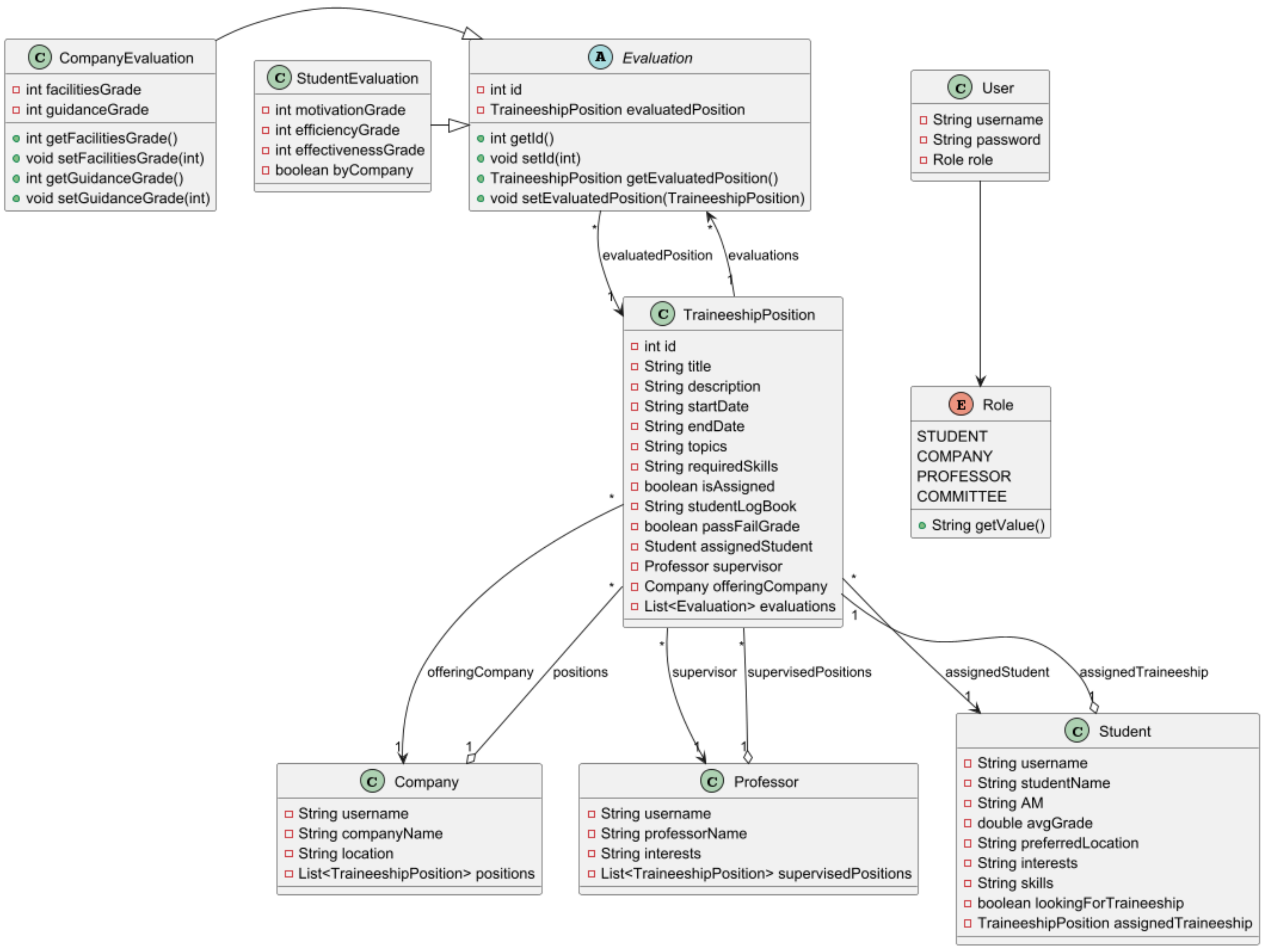


FIGURE 3 Domain Model

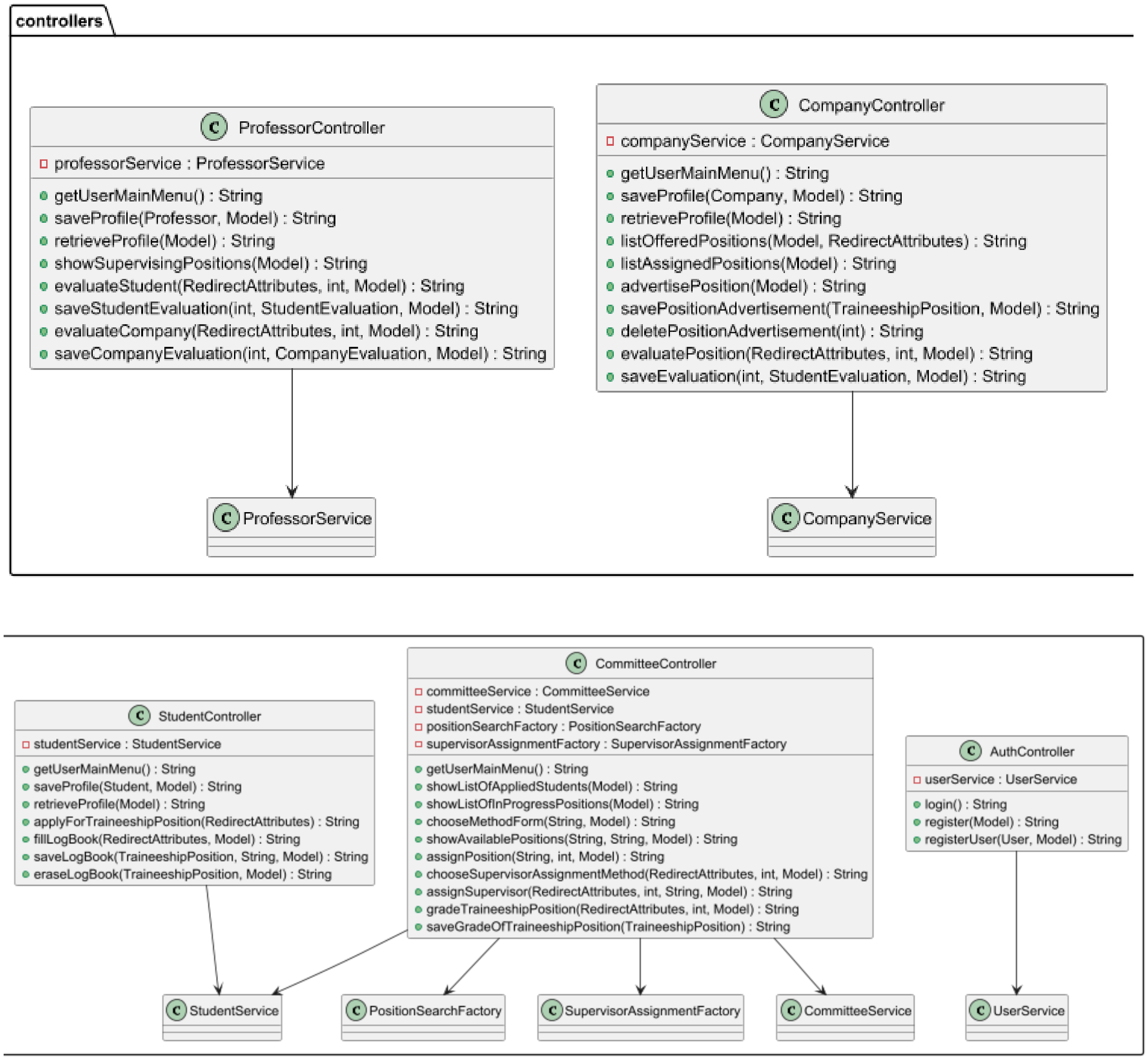

controllers
ProfessorController
professorService : ProfessorService
getUserMainMenu() : String
saveProfile(Professor, Model) : String
retrieveProfile(Model) : String
showSupervisingPositions(Model) : String
evaluateStudent(RedirectAttributes, int, Model) : String
saveStudentEvaluation(int, StudentEvaluation, Model) : String
evaluateCompany(RedirectAttributes, int, Model) : String
saveCompanyEvaluation(int, CompanyEvaluation, Model) : String
CompanyController
companyService : CompanyService
getUserMainMenu() : String
saveProfile(Company, Model) : String
retrieveProfile(Model) : String
listOfferedPositions(Model, RedirectAttributes) : String
listAssignedPositions(Model) : String
advertisePosition(Model) : String
savePositionAdvertisement(TraineeshipPosition, Model) : String
deletePositionAdvertisement(int) : String
evaluatePosition(RedirectAttributes, int, Model) : String
saveEvaluation(int, StudentEvaluation, Model) : String
ProfessorService
CompanyService
StudentController
studentService : StudentService
getUserMainMenu() : String
saveProfile(Student, Model) : String
retrieveProfile(Model) : String
applyForTraineeshipPosition(RedirectAttributes) : String
fillLogBook(RedirectAttributes, Model) : String
saveLogBook(TraineeshipPosition, String, Model) : String
eraseLogBook(TraineeshipPosition, Model) : String
CommitteeController
committeeService : CommitteeService
studentService : StudentService
positionSearchFactory : PositionSearchFactory
supervisorAssignmentFactory : SupervisorAssignmentFactory
getUserMainMenu() : String
showListOfAppliedStudents(Model) : String
showListOfInProgressPositions(Model) : String
chooseMethodForm(String, Model) : String
showAvailablePositions(String, String, Model) : String
assignPosition(String, int, Model) : String
chooseSupervisorAssignmentMethod(RedirectAttributes, int, Model) : String
assignSupervisor(RedirectAttributes, int, String, Model) : String
gradeTraineeshipPosition(RedirectAttributes, int, Model) : String
saveGradeOfTraineeshipPosition(TraineeshipPosition) : String
AuthController
userService : UserService
login() : String
register(Model) : String
registerUser(User, Model) : String
StudentService
PositionSearchFactory
SupervisorAssignmentFactory
CommitteeService
UserService

FIGURE 4 Controllers

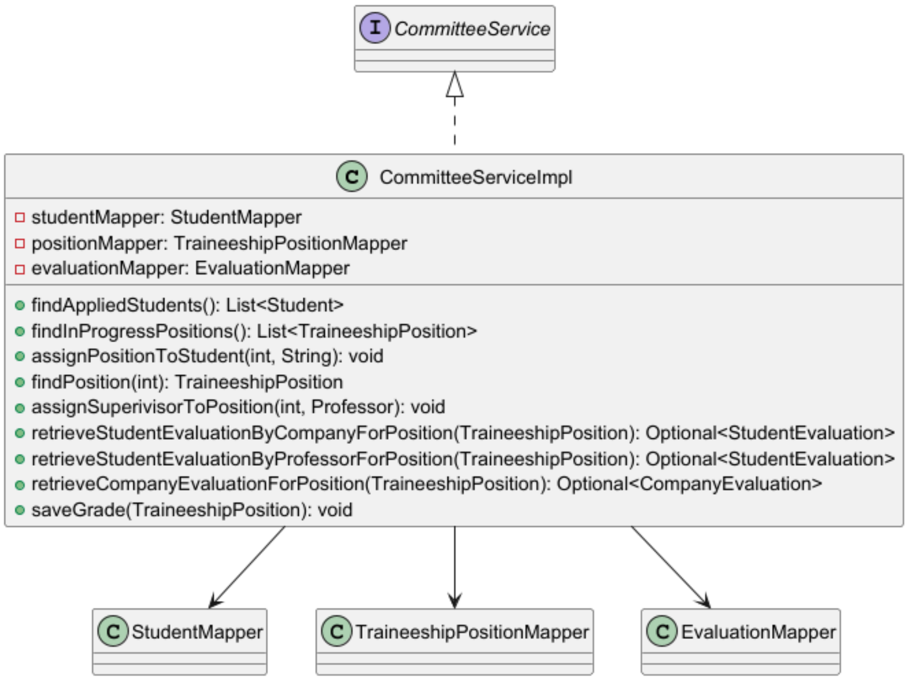

I CommitteeService
C CommitteeServiceImpl
studentMapper: StudentMapper
positionMapper: TraineeshipPositionMapper
evaluationMapper: EvaluationMapper
findAppliedStudents(): List<Student>
findInProgressPositions(): List<TraineeshipPosition>
assignPositionToStudent(int, String): void
findPosition(int): TraineeshipPosition
assignSuperivisorToPosition(int, Professor): void
retrieveStudentEvaluationByCompanyForPosition(TraineeshipPosition): Optional<StudentEvaluation>
retrieveStudentEvaluationByProfessorForPosition(TraineeshipPosition): Optional<StudentEvaluation>
retrieveCompanyEvaluationForPosition(TraineeshipPosition): Optional<CompanyEvaluation>
saveGrade(TraineeshipPosition): void
C StudentMapper
C TraineeshipPositionMapper
C EvaluationMapper

FIGURE 5 Committee Service

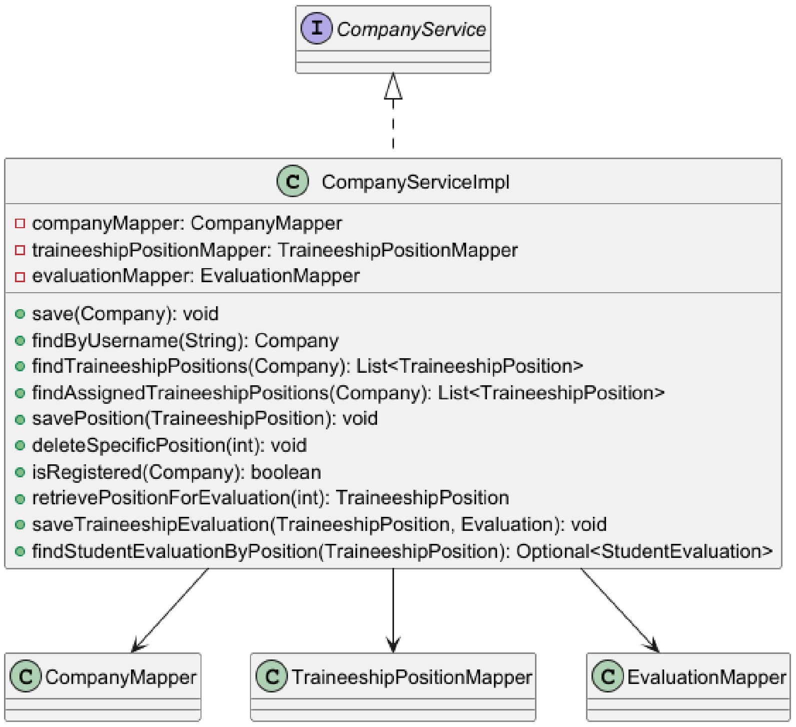

I CompanyService
C CompanyServiceImpl
companyMapper: CompanyMapper
traineeshipPositionMapper: TraineeshipPositionMapper
evaluationMapper: EvaluationMapper
save(Company): void
findByUsername(String): Company
findTraineeshipPositions(Company): List<TraineeshipPosition>
findAssignedTraineeshipPositions(Company): List<TraineeshipPosition>
savePosition(TraineeshipPosition): void
deleteSpecificPosition(int): void
isRegistered(Company): boolean
retrievePositionForEvaluation(int): TraineeshipPosition
saveTraineeshipEvaluation(TraineeshipPosition, Evaluation): void
findStudentEvaluationByPosition(TraineeshipPosition): Optional<StudentEvaluation>
C CompanyMapper
C TraineeshipPositionMapper
C EvaluationMapper

FIGURE 6 Company Service

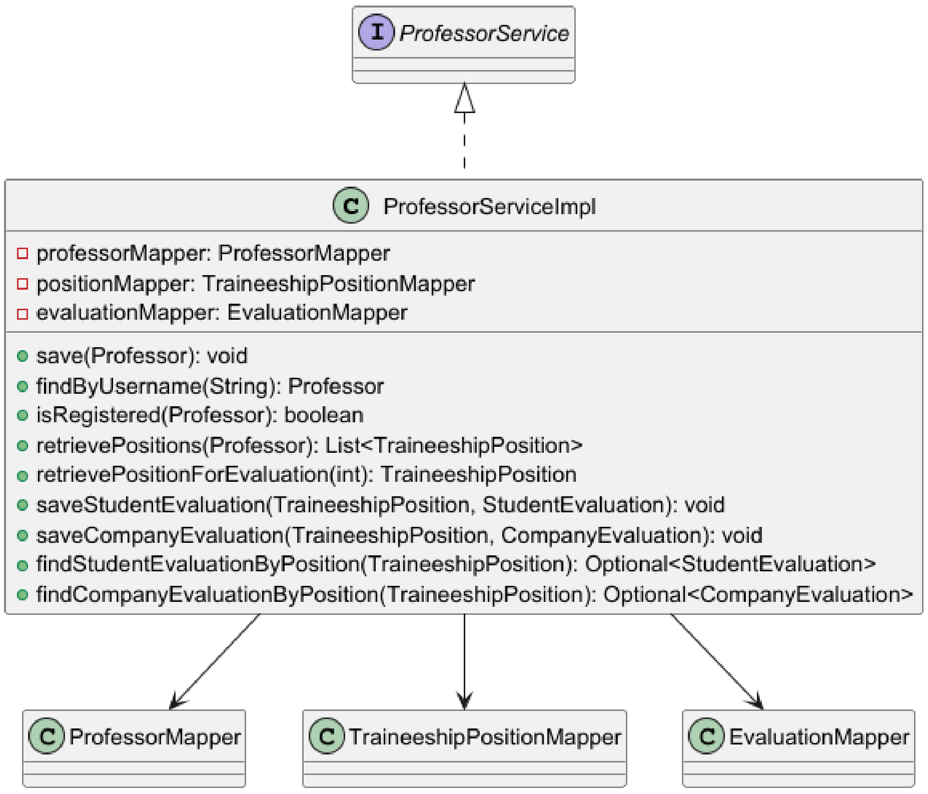

ProfessorService
ProfessorServiceImpl
professorMapper: ProfessorMapper
positionMapper: TraineeshipPositionMapper
evaluationMapper: EvaluationMapper
save(Professor): void
findByUsername(String): Professor
isRegistered(Professor): boolean
retrievePositions(Professor): List<TraineeshipPosition>
retrievePositionForEvaluation(int): TraineeshipPosition
saveStudentEvaluation(TraineeshipPosition, StudentEvaluation): void
saveCompanyEvaluation(TraineeshipPosition, CompanyEvaluation): void
findStudentEvaluationByPosition(TraineeshipPosition): Optional<StudentEvaluation>
findCompanyEvaluationByPosition(TraineeshipPosition): Optional<CompanyEvaluation>
ProfessorMapper
TraineeshipPositionMapper
EvaluationMapper

FIGURE 7 Professor Service

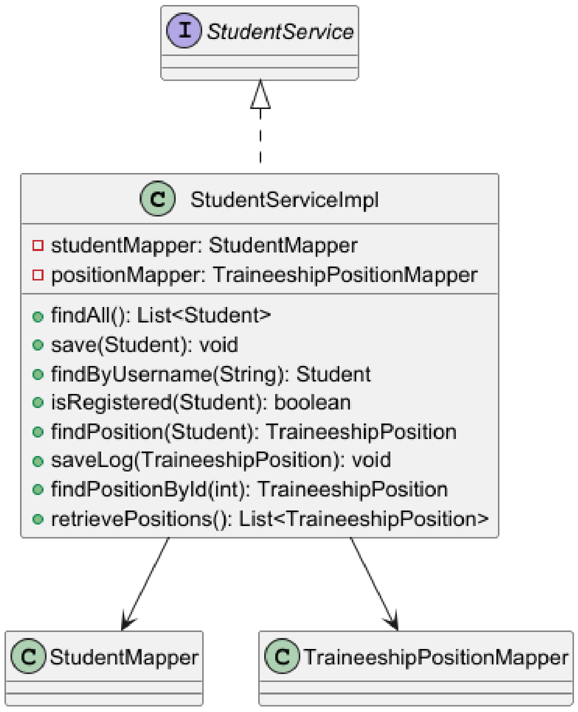

StudentService
StudentServiceImpl
studentMapper: StudentMapper
positionMapper: TraineeshipPositionMapper
findAll(): List<Student>
save(Student): void
findByUsername(String): Student
isRegistered(Student): boolean
findPosition(Student): TraineeshipPosition
saveLog(TraineeshipPosition): void
findPositionById(int): TraineeshipPosition
retrievePositions(): List<TraineeshipPosition>
StudentMapper
TraineeshipPositionMapper

FIGURE 8 Student Service

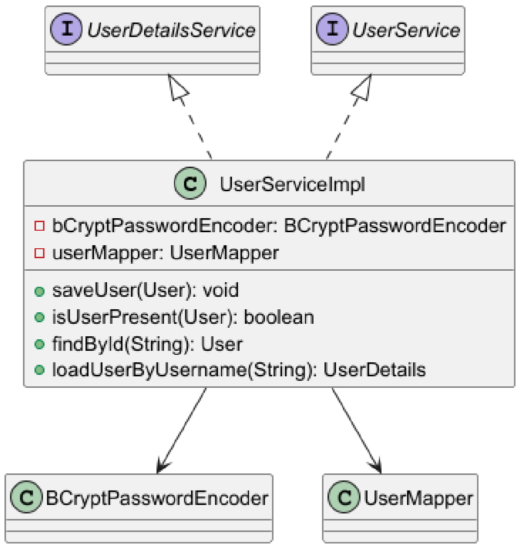


FIGURE 9 User Service

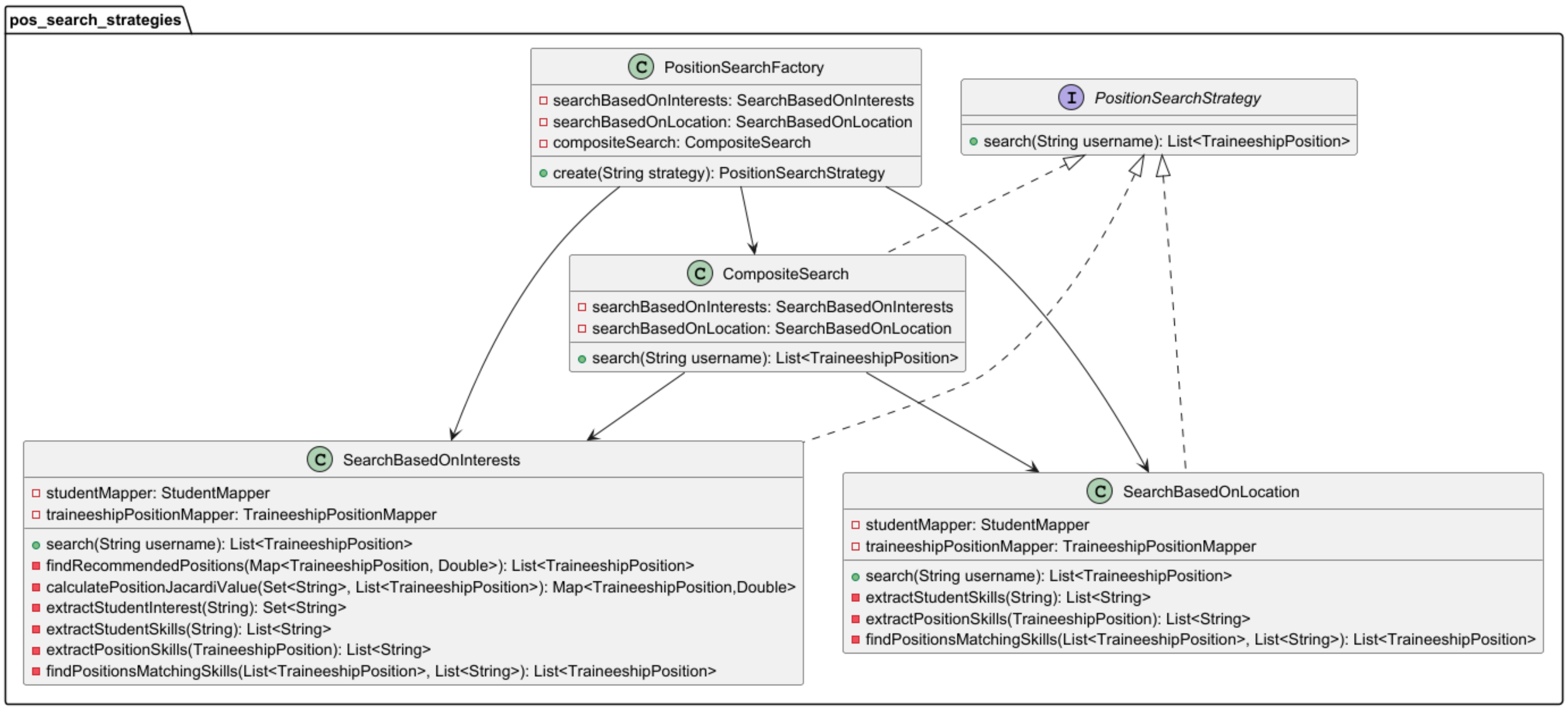

pos_search_strategies
PositionSearchFactory
searchBasedOnInterests: SearchBasedOnInterests
searchBasedOnLocation: SearchBasedOnLocation
compositeSearch: CompositeSearch
create(String strategy): PositionSearchStrategy
PositionSearchStrategy
search(String username): List<TraineeshipPosition>
CompositeSearch
searchBasedOnInterests: SearchBasedOnInterests
searchBasedOnLocation: SearchBasedOnLocation
search(String username): List<TraineeshipPosition>
SearchBasedOnInterests
studentMapper: StudentMapper
traineeshipPositionMapper: TraineeshipPositionMapper
search(String username): List<TraineeshipPosition>
findRecommendedPositions(Map<TraineeshipPosition, Double>): List<TraineeshipPosition>
calculatePositionJacardiValue(Set<String>, List<TraineeshipPosition>): Map<TraineeshipPosition,Double>
extractStudentInterest(String): Set<String>
extractStudentSkills(String): List<String>
extractPositionSkills(TraineeshipPosition): List<String>
findPositionsMatchingSkills(List<TraineeshipPosition>, List<String>): List<TraineeshipPosition>
SearchBasedOnLocation
studentMapper: StudentMapper
traineeshipPositionMapper: TraineeshipPositionMapper
search(String username): List<TraineeshipPosition>
extractStudentSkills(String): List<String>
extractPositionSkills(TraineeshipPosition): List<String>
findPositionsMatchingSkills(List<TraineeshipPosition>, List<String>): List<TraineeshipPosition>

FIGURE 10 Position Search Strategies

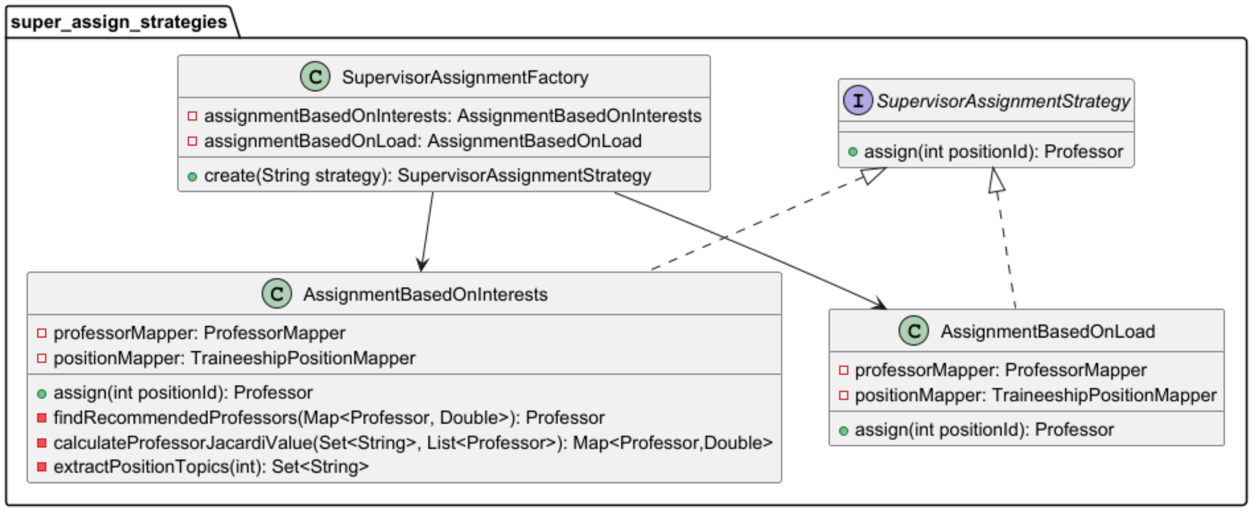

super_assign_strategies
SupervisorAssignmentFactory
assignmentBasedOnInterests: AssignmentBasedOnInterests
assignmentBasedOnLoad: AssignmentBasedOnLoad
create(String strategy): SupervisorAssignmentStrategy
SupervisorAssignmentStrategy
assign(int positionId): Professor
AssignmentBasedOnInterests
professorMapper: ProfessorMapper
positionMapper: TraineeshipPositionMapper
assign(int positionId): Professor
findRecommendedProfessors(Map<Professor, Double>): Professor
calculateProfessorJacardiValue(Set<String>, List<Professor>): Map<Professor,Double>
extractPositionTopics(int): Set<String>
AssignmentBasedOnLoad
professorMapper: ProfessorMapper
positionMapper: TraineeshipPositionMapper
assign(int positionId): Professor

FIGURE 11 Supervisor Assignment Strategies

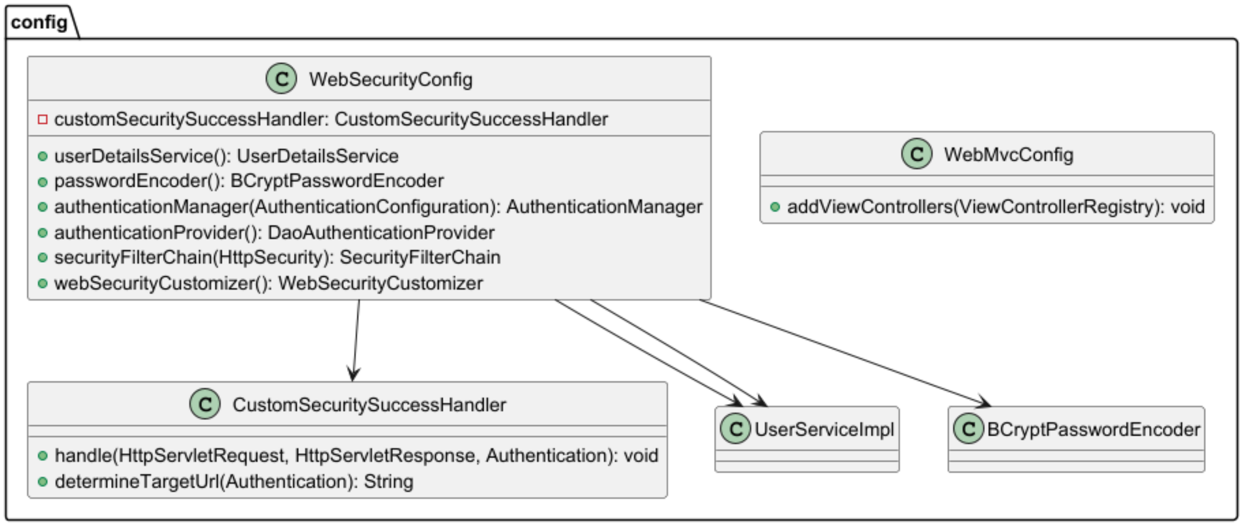

config
WebSecurityConfig
customSecuritySuccessHandler: CustomSecuritySuccessHandler
userDetailsService(): UserDetailsService
passwordEncoder(): BCryptPasswordEncoder
authenticationManager(AuthenticationConfiguration): AuthenticationManager
authenticationProvider(): DaoAuthenticationProvider
securityFilterChain(HttpSecurity): SecurityFilterChain
webSecurityCustomizer(): WebSecurityCustomizer
WebMvcConfig
addViewControllers(ViewControllerRegistry): void
CustomSecuritySuccessHandler
handle(HttpServletRequest, HttpServletResponse, Authentication): void
determineTargetUrl(Authentication): String
UserServiceImpl
BCryptPasswordEncoder

FIGURE 12 Config

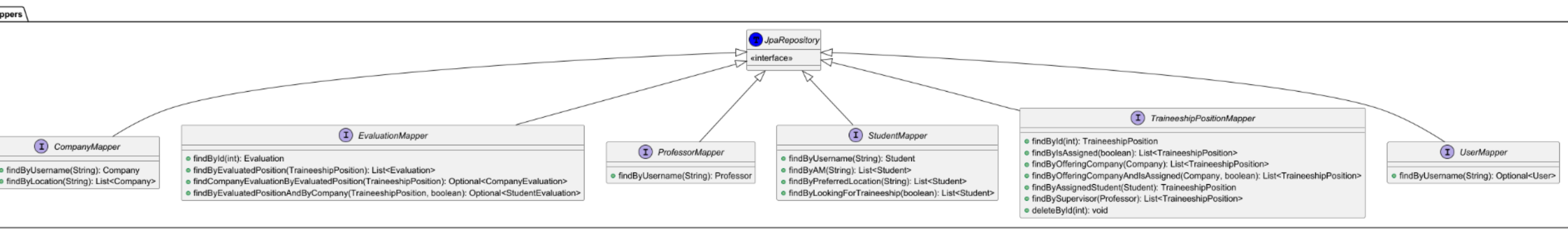

ppers
JpaRepository
«interface»
CompanyMapper
findByUsername(String): Company
findByLocation(String): List<Company>
EvaluationMapper
findById(int): Evaluation
findByEvaluatedPosition(TraineeshipPosition): List<Evaluation>
findCompanyEvaluationByEvaluatedPosition(TraineeshipPosition): Optional<CompanyEvaluation>
findByEvaluatedPositionAndByCompany(TraineeshipPosition, boolean): Optional<StudentEvaluation>
ProfessorMapper
findByUsername(String): Professor
StudentMapper
findByUsername(String): Student
findByAM(String): List<Student>
findByPreferredLocation(String): List<Student>
findByLookingForTraineeship(boolean): List<Student>
TraineeshipPositionMapper
findById(int): TraineeshipPosition
findByIsAssigned(boolean): List<TraineeshipPosition>
findByOfferingCompany(Company): List<TraineeshipPosition>
findByOfferingCompanyAndIsAssigned(Company, boolean): List<TraineeshipPosition>
findByAssignedStudent(Student): TraineeshipPosition
findBySupervisor(Professor): List<TraineeshipPosition>
deleteById(int): void
UserMapper
findByUsername(String): Optional<User>

FIGURE 13 Mappers

Class Name: Company	
Responsibilities: ▪ Holds information for a company	**Collaborations:** ▪ TraineeshipPosition: A list of the positions the company offers

Class Name: Student	
Responsibilities: ▪ Holds information for a student	**Collaborations:** ▪ TraineeshipPosition: The position the student is assigned to

Class Name: Professor	
Responsibilities: ▪ Holds information for a professor	**Collaborations:** ▪ TraineeshipPosition: A list of the positions the professor is supervising

Class Name: User	
Responsibilities: ▪ Holds information for a user	**Collaborations:** ▪ Implements the UserDetails interface

Class Name: TraineeshipPosition	
Responsibilities: ▪ Holds information for a traineeship	**Collaborations:** ▪ Student: The student assigned in the position ▪ Professor: The professor supervising the position ▪ Company: The company that offers the position ▪ Evaluation: A list of evaluations completed for the position

Class Name: Evaluation	
Responsibilities: ▪ Holds information for an evaluation ▪ Has the StudentEvaluation and CompanyEvaluation children, for evaluations regarding the student and the company	**Collaborations:** ▪ TraineeshipPosition: The position being evaluated

Class Name: AuthController	
Responsibilities: ▪ Handles http requests for the user login and register	**Collaborations:** ▪ UserService: Communicates with the database to save the user

Class Name: StudentController	
Responsibilities: ▪ Handles http requests for the student user	**Collaborations:** ▪ StudentService: Takes the requests input, communicates with the database and returns the output to the controller ▪ TraineeshipPosition: Manipulates the students' traineeship wherever needed

Class Name: ProfessorController	
Responsibilities:	**Collaborations:**
▪ Handles http requests for the professor user	▪ ProfessorService: Takes the requests input, communicates with the database and returns the output to the controller ▪ TraineeshipPosition: Manipulates the professor's supervised traineeships wherever needed ▪ Evaluation: Handles the evaluations for the company and the student in a position

Class Name: CompanyController	
Responsibilities:	**Collaborations:**
▪ Handles http requests for the company user	▪ CompanyService: Takes the requests input, communicates with the database and returns the output to the controller ▪ TraineeshipPosition: Manipulates the company's offered traineeships wherever needed ▪ Evaluation: Handles the evaluations for the student in a position

Class Name: CommitteeController	
Responsibilities:	**Collaborations:**
▪ Handles http requests for the committee member user	▪ CommitteeService: Takes the requests input, communicates with the database and returns the output to the controller ▪ StudentService: Takes the requests regarding a student, communicates with the database and returns the output to the controller ▪ Student: Used for the specific student in the search method ▪ StudentService: Communicates with the database to find the requested student ▪ Professor: Used for the specific professor in the assign method ▪ TraineeshipPosition: Manipulates the traineeships wherever needed ▪ Evaluation: Used to present the established evaluations ▪ PositionSearchStrategy: Handles the requested search strategy ▪ PositionSearchFactory: Creates the requested search strategy ▪ SupervisorAssignmentStrategy: Handles the requested assign strategy ▪ SupervisorAssignmentFactory: Creates the requested assign strategy

Class Name: SearchBasedOnInterests	
Responsibilities:	**Collaborations:**
▪ Implements the search method for a traineeship for a given student, based on interests	▪ StudentMapper: Retrieves the given student from the database ▪ TraineeshipPositionMapper: Retrieves traineeships from the database

	▪ TraineeshipPosition: Used for handling the different positions in the algorithm ▪ Implements the PositionSearchStrategy interface

Class Name: SearchBasedOnLocation	
Responsibilities: ▪ Implements the search method for a traineeship for a given student, based on the preferred location	**Collaborations:** ▪ StudentMapper: Retrieves the given student from the database ▪ TraineeshipPositionMapper: Retrieves traineeships from the database ▪ TraineeshipPosition: Used for handling the different positions in the algorithm ▪ Implements the PositionSearchStrategy interface

Class Name: CompositeSearch	
Responsibilities: ▪ Implements all the search methods combined	**Collaborations:** ▪ TraineeshipPosition: Used for handling the different positions in the algorithm ▪ SearchBasedOnInterests, SearchBasedOnLocation: Used for combining the two strategies in the search method ▪ Implements the PositionSearchStrategy interface

Class Name: PositionSearchFactory	
Responsibilities: ▪ Creates the selected search strategy	**Collaborations:**

	▪ SearchBasedOnInterests, SearchBasedOnLocation, CompositeSearch: Used to return the selected strategy

Class Name: AssignmentBasedOnInterests	
Responsibilities:	**Collaborations:**
▪ Implements the assignment method for a professor for a given traineeship, based on interests	▪ ProfessorMapper: Retrieves the given professor from the database ▪ TraineeshipPositionMapper: Retrieves traineeships from the database ▪ Professor: Used for handling the different professors in the algorithm ▪ TraineeshipPosition: Used for handling the different positions in the algorithm ▪ Implements the SupervisorAssignmentStrategy interface

Class Name: AssignmentBasedOnLoad	
Responsibilities:	**Collaborations:**
▪ Implements the assignment method for a professor for a given traineeship, based on load	▪ ProfessorMapper: Retrieves the given professor from the database ▪ TraineeshipPositionMapper: Retrieves traineeships from the database ▪ Professor: Used for handling the different professors in the algorithm ▪ TraineeshipPosition: Used for handling the different positions in the algorithm ▪ Implements the SupervisorAssignmentStrategy interface

Class Name: SupervisorAssignmentFactory	
Responsibilities: ▪ Creates the selected assignment strategy	**Collaborations:** ▪ AssignmentBasedOnInterests, AssignmentBasedOnLoad: Used to return the selected strategy

Class Name: CompanyMapper	
Responsibilities: ▪ Retrieves company entities from the database	**Collaborations:** ▪ Implements the JpaRepository interface

Class Name: EvaluationMapper	
Responsibilities: ▪ Retrieves evaluation entities from the database	**Collaborations:** ▪ Implements the JpaRepository interface ▪ CompanyEvaluation, StudentEvaluation: Used for retrieving a specific evaluation regarding a company or a student

Class Name: ProfessorMapper	
Responsibilities: ▪ Retrieves professor entities from the database	**Collaborations:** ▪ Implements the JpaRepository interface

Class Name: StudentMapper	
Responsibilities: ▪ Retrieves student entities from the database	**Collaborations:** ▪ Implements the JpaRepository interface

Class Name: TraineeshipPositionMapper	
Responsibilities:	**Collaborations:**
▪ Retrieves traineeship entities from the database	▪ Implements the JpaRepository interface

Class Name: UserMapper	
Responsibilities:	**Collaborations:**
▪ Retrieves user entities from the database	▪ Implements the JpaRepository interface

Class Name: CommitteeServiceImpl	
Responsibilities:	**Collaborations:**
▪ Takes input from the controller regarding the committee requests, uses the mappers to communicate with the database and returns the output back to the controller	▪ Implements the CommitteeService interface ▪ StudentMapper: Retrieves student objects from the database ▪ TraineeshipPositionMapper: Retrieves traineeship objects from the database ▪ EvaluationMapper: Retrieves evaluation objects from the database ▪ Student, TraineeshipPosition, CompanyEvaluation, StudentEvaluation: Used to store the specific object or to pass it as parameter in a method

Class Name: CompanyServiceImpl	
Responsibilities: ▪ Takes input from the controller regarding the company requests, uses the mappers to communicate with the database and returns the output back to the controller	**Collaborations:** ▪ Implements the CompanyService interface ▪ CompanyMapper: Retrieves company objects from the database ▪ TraineeshipPositionMapper: Retrieves traineeship objects from the database ▪ EvaluationMapper: Retrieves evaluation objects from the database ▪ Company, TraineeshipPosition, StudentEvaluation: Used to store the specific object or to pass it as parameter in a method

Class Name: ProfessorServiceImpl	
Responsibilities: ▪ Takes input from the controller regarding the professor requests, uses the mappers to communicate with the database and returns the output back to the controller	**Collaborations:** ▪ Implements the ProfessorService interface ▪ ProfessorMapper: Retrieves professor objects from the database ▪ TraineeshipPositionMapper: Retrieves traineeship objects from the database ▪ EvaluationMapper: Retrieves evaluation objects from the database ▪ Professor, TraineeshipPosition, CompanyEvaluation, StudentEvaluation: Used to store the specific object or to pass it as parameter in a method

Class Name: StudentServiceImpl	
Responsibilities:	**Collaborations:**
▪ Takes input from the controller regarding the student requests, uses the mappers to communicate with the database and returns the output back to the controller	▪ Implements the StudentService interface ▪ StudentMapper: Retrieves student objects from the database ▪ TraineeshipPositionMapper: Retrieves traineeship objects from the database ▪ Student, TraineeshipPosition: Used to store the specific object or to pass it as parameter in a method

Class Name: UserServiceImpl	
Responsibilities:	**Collaborations:**
▪ Takes input from the controller regarding the user requests, uses the mappers to communicate with the database and returns the output back to the controller	▪ Implements the UserService, UserDetailsService interfaces ▪ UserMapper: Retrieves user objects from the database ▪ User: Used to store the specific object ▪ BCryptPasswordEncoder: Used for the user's password encoding