Traineeship Application

Sprint Report

TraineeshipApplicationDeveloppers:
4941: Ilias Gratsias

4985: Dimitrios Pagonis

4994: Dimitrios Tzalokostas

VERSIONS HISTORY

Date	Version	Description	Author
1/5/2025	V1.0	Database design and setting spring security. Adding buttons for the Main Menu at the frontend	Ilias Gratsias Dimitros Pagonis Dimitrios Tzalokostas
10/5/2025	V1.1	Implement user stories for creating profile and adding a trainee Position on the services and the controller	Ilias Gratsias Dimitros Pagonis Dimitrios Tzalokostas
25/5/2025	V1.2	Final implementation of the whole project	Ilias Gratsias Dimitros Pagonis Dimitrios Tzalokostas

1 Introduction

1.1 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	Ilias Gratsias
Scrum Master	Dimitrios Tzalokostas
Development Team	Ilias Gratsias Dimitros Pagonis Dimitrios Tzalokostas

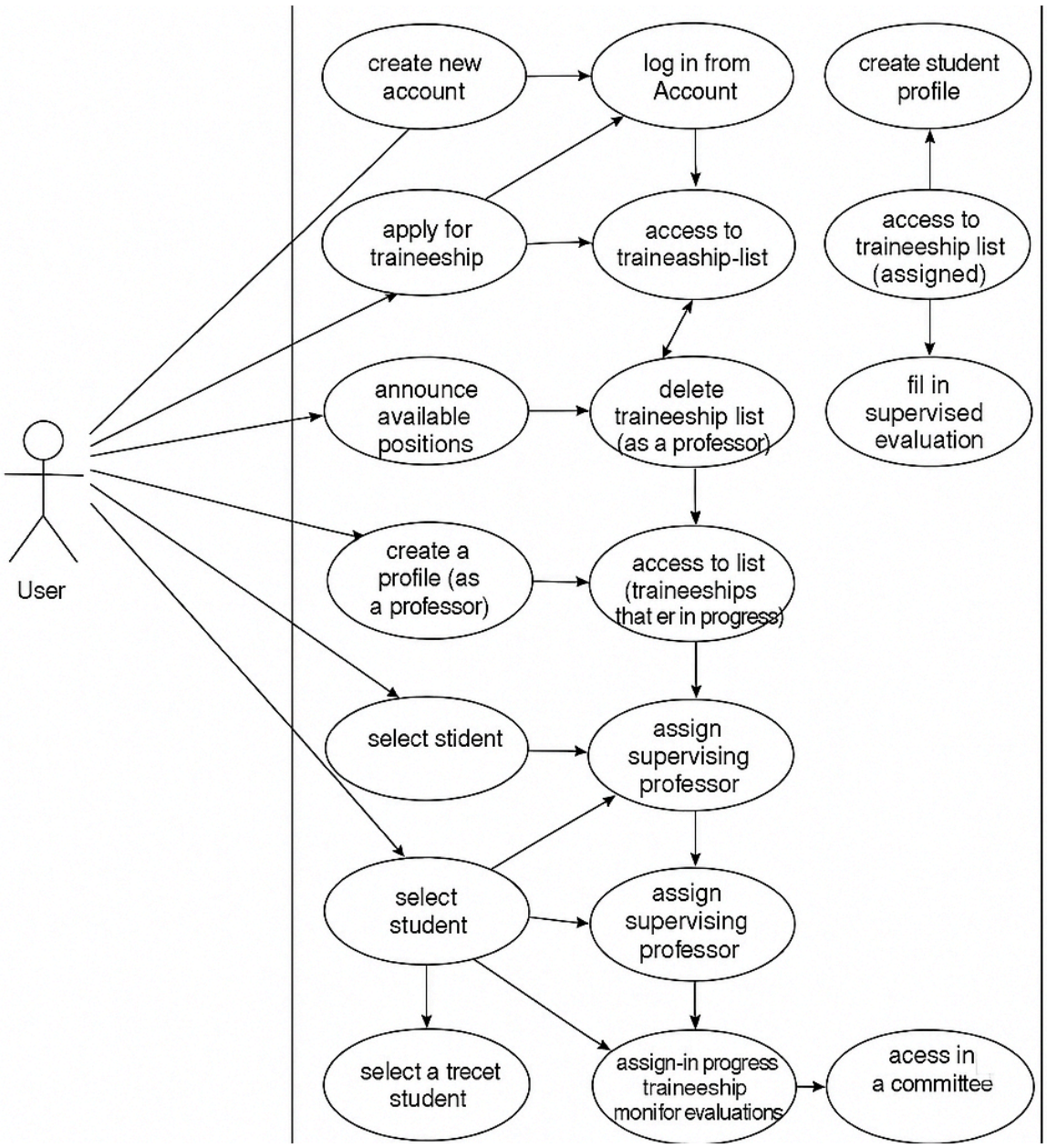
2.2 Sprints

<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	**20/4/2025**	**1/5/2025**	**2**	**US1, US2, US3**
2	**1/5/2025**	**8/5/2025**	**1**	**US4, US7, US13**
3	**9/5/2025**	**18/5/2025**	**1**	**US5,US8,US9,US10,US11,US14,US16,US17,US18,US19,US20**
4	**19/5/2025**	**26/5/2025**	**1**	**US6, US12, US15, US21**

3 Use Cases

THE UML use case diagram is:


User
create new account
log in from Account
create student profile
apply for traineeship
access to traineaship-list
access to traineeship list (assigned)
announce available positions
delete traineeship list (as a professor)
fil in supervised evaluation
create a profile (as a professor)
access to list (traineeships that er in progress)
select stident
assign supervising professor
select student
assign supervising professor
select a trecet student
assign-in progress traineeship monifor evaluations
acess in a committee

3.1 <Use Case 1>

Use case ID	Create User Account
Actors	User of the Traineeship App
Pre conditions	User enters homepage.
Main flow of events	1. The use case starts when the user selects the option to register for the application. 2. The system presents a registration form. 2.1 The user fills the credentials (username, password, role). 2.2 The user submits the form. 2.3 The system registers the user.
Alternative flow 1	If the registration form is incomplete, the system asks the user to complete all the missing form fields.
Alternative flow 2	If the username that is inputted already exists, the system changes the page to login and gives a related message
Post conditions	The user can now login

3.2 <Use Case 2>

Use case ID	Login user
Actors	User of the Traineeship App
Pre conditions	User has registered a profile
Main flow of events	1. The use case starts when the user wants to login in the application 2. The system presents a login form. 2.1 The user fills the credentials (username, password). 2.2 The user presses the login button. 2.3 The system logins the user.
Alternative flow 1	If the login form is incomplete, the system asks the user to complete all the missing form's fields.
Alternative flow 2	If the login credentials are incorrect, the system displays a message to user, to try again.
Alternative flow 3	If the user has already a profile, the system redirects the user to the main menu of the application.
Alternative flow 4	If the user logins for the first time in the application, the system redirects him to create profile page and puts his personal data

Post conditions	The user has filled his profile and can is logged in.

3.3 <Use Case 3>

Use case ID	Logout user
Actors	User of the Traineeship App
Pre conditions	The User has successfully logged in to the application and has profile.
Main flow of events	1. The use case starts when the user selects the option to logout from the application.
Post conditions	The user is terminated from his interaction with the application.

3.4 <Use Case 4>

Use case ID	Create Student Profile
Actors	Student as a user of the Traineeship App
Pre conditions	The Student has not created a profile and is logged into the application.
Main flow of events	1. The use case starts when the user is logged into the application for the first time. 2. The application presents a profile form. 2.1 The user fills in personal data (full name, university ID number, interests, skills, preferred location). 2.2 The user submits the form. 2.3 The system checks the information. 2.4 On successful validation the system creates the student's profile.
Alternative flow 1	If the profile form is incomplete, the system asks the student to complete all the missing form's fields.
Post conditions	The user has created a profile and can see the application's dashboard.

3.5 <Use Case 5>

Use case ID	Apply for a traineeship
Actors	Student as a user of the Traineeship App
Pre conditions	Student has a profile and is logged into the application.
Main flow of events	1. The use case starts when the student has pushed the button “Apply for a traineeship” 2. The application presents all the available traineeship positions. 2.1. The student pushes the button “Apply” for a position that wants to apply for. 2.2 The system adds the application to the applications list of an available position and other committee can view the request.
Alternative flow 1	If student does not want to apply for a position, just skips to push the button
Post conditions	A new request is created by the student.

3.6 <Use Case 6>

Use case ID	Fill the traineeship logbook
Actors	Student as a user of the Traineeship App
Pre conditions	Student has a profile and is logged into the application and is assigned to a Trainee position.
Main flow of events	1. The use case starts when the student has pushed the button “My logbook” at the Student’s Dashboard. 2. The application presents a fill the logbook form. 3. The student writes in the logbook. 4. The student pushes the SAVE button
Alternative flow 1	If student does not want to write to the logbook, presses the “Back to Dashboard” button.
Post conditions	The system saves the student’s form.

3.7 <Use Case 7>

Use case ID	Create Company Profile
Actors	Company as a user of the Traineeship App
Pre conditions	The Company has not created a profile and is logged into the application.
Main flow of events	1. The use case starts when the user is logged into the application for the first time. 2. The application presents a profile form. 2.1 The user fills in personal data (full company's name, company's location). 2.2 The user submits the form. 2.3 The system checks the information. 3. On successful validation the system creates the student's profile.
Alternative flow 1	If the profile form is incomplete, the system asks the student to complete all the missing form's fields.
Post conditions	The user has created a profile and can see the application's dashboard.

3.8 <Use Case 8>

Use case ID	Access to the list of Available Trainee Positions
Actors	Company as a user of the Traineeship App
Pre conditions	The Company has created a profile and is logged into the application.
Main flow of events	1. The use case starts when the company presses the button "My positions" 2. The application presents all the positions that are advertised by the company. 2.1 At each advertised position, the application shows position's data (Name, Description,Duration,Location,Skills,Topics). 2.2 Below data, the application shows the status of each position (if is assigned or not). 2.3 Also, the application shows 2 buttons (Delete and Edit), if the company wants to delete the position or edit some position's data.
Alternative flow 1	The company does not want to add or edit any position, so it returns to the dashboard by clicking "Back to Dashboard".

Post conditions	The company can see its advertised positions.

3.9 <Use Case 9>

Use case ID	Access to the list of Available Trainee Positions
Actors	Company as a user of the Traineeship App
Pre conditions	The Company has created a profile and is logged into the application.
Main flow of events	1. The use case starts when the company presses the button “Positions assigned to Students” 2. The application presents all the positions that are assigned to students. 2.1 At each assigned position, the application shows position’s data (Title,Company’sName,Description,Applicant,Supervisor,Duration)
Alternative flow 1	If there are no positions assigned to any students, the application shows a message “No positions have been assigned to students yet.”
Post conditions	The company can see all the assigned positions to students.

3.10 <Use Case 10>

Use case ID	Announce an Available Position
Actors	Company as a user of the Traineeship App
Pre conditions	The Company has created a profile and is logged into the application.
Main flow of events	1. The use case starts when the user selects the “My positions” button 2. By clicking Add position , the application shows a form that the company has to fill the positions data 3. When the company writes all data, clicks the “Upload Position” button.
Alternative flow 1	If the profile form is incomplete, the system asks the professor to complete all the missing form’s fields.
Post conditions	The application has published the new trainee position.

3.11 <Use Case 11>

Use case ID	Delete Traineeship Position
Actors	Company as a user of the Traineeship App
Pre conditions	The Company has created a profile and is logged into the application.
Main flow of events	1. The use case starts when the company presses the button “My positions” 2. The application presents all the positions that are assigned to students. 2.1 At each assigned position, the application shows position’s data (Title,Company’sName,Description,Applicant,Supervisor,Duration) 2.2 If the duration has finished, the application shows only the Delete button to delete the position.
Alternative flow 1	If any other position has not finished, the applications shows the Delete Button, if the company wants to delete this position too.
Post conditions	The company can delete a trainee position.

3.10 <Use Case 12>

Use case ID	Fill an evaluation for a traineeship position
Actors	Company as a user of the Traineeship App
Pre conditions	The Company has created a profile and is logged into the application.
Alternative flow 1	If any other position has not started, the application does not show the Evaluate this position” button.
Alternative flow 2	If company wants to revaluate any evaluated position, the application shows a message that the position is already evaluated and a “ReEvaluate this position ” Button.
Post conditions	The company can delete a trainee position.

3.11 <Use Case 13>

Use case ID	Create Professor Profile
Actors	Professor as a user of the Traineeship App
Pre conditions	Professor has not created a profile and is logged into the application.
Main flow of events	4. The use case starts when the user is logged into the application for the first time. 5. The application presents a profile form. 2.4 The user fills in personal data (full professor's name, professor's interests). 2.5 The user submits the form. 2.6 The system checks the information. 6. On successful validation the system creates the student's profile.
Alternative flow 1	If the profile form is incomplete, the system asks the professor to complete all the missing form's fields.
Post conditions	The user has created a profile and can see the application's dashboard.

3.14 <Use Case 14>

Use case ID	Access to the List of supervised positions
Actors	Professor as a user of the Traineeship App
Pre conditions	Professor has not created a profile and is logged into the application.
Main flow of events	1. The use case starts when the professor clicks the "My supervised Positions " button. 2. By clicking, the application appears all the assigned positions that the professor supervises.
Alternative flow 1	If the professor does not have any positions supervised, the application appears a message "`You don't supervise any positions.`"
Post conditions	The professor can see all the supervised positions.

3.15 <Use Case 15>

Use case ID	Fill an evaluation for a supervised position
Actors	Professor as a user of the Traineeship App
Pre conditions	Professor has not created a profile and is logged into the application.

Main flow of events	1. The use case starts when the professor clicks the “My supervised Positions ” button. 2. By clicking, the application appears all the assigned positions that the professor supervises. 3. At the Action side, the application appears a “Monitor” button, that navigates the professor to evaluate this position.
Alternative flow 1	If the professor wants to reevaluate any supervised position, the application appears a message “Already evaluated - Edit?” and a “Reevaluate position” button to reevaluate again
Post conditions	The professor can evaluate any supervised positions.

3.16 <Use Case 16>

Use case ID	Create Committee Member Profile
Actors	Committee Member as a user of the Traineeship App
Pre conditions	Committee Member has not created a profile and is logged into the application.
Main flow of events	1. The use case starts when the user is logged into the application for the first time. 2. The application presents a profile form. 2.1 The user fills in personal data (full professor’s name, professor’s interests). 2.2 The user submits the form. 2.3 The system checks the information. 3.On successful validation the system creates the student’s profile.
Alternative flow 1	If the profile form is incomplete, the system asks the professor to complete all the missing form’s fields.
Post conditions	The user has created a profile and can see the application’s dashboard.

3.17 <Use Case 17>

Use case ID	Show lists of applied students
Actors	Committee Member as a user of the Traineeship App

Pre conditions	Committee Member has created a profile and is logged into the application.
Main flow of events	1. The use case starts when the member clicks the “View Traineeship Positions” 2. The application appears all the available positions 3. When the member selects a position, it will show all the students who applied for this position.
Alternative flow 1	If there are no available positions, the application will show a “No available positions found.” and it will not show any list of students
Post conditions	The member can have access to the list of applied students of an available position.

3.18 <Use Case 18>

Use case ID	Select Student For a Traineeship position
Actors	Committee Member as a user of the Traineeship App
Pre conditions	Committee Member has created a profile and is logged into the application.
Main flow of events	1. The use case starts when the member clicks the “View Traineeship Positions” 2. The application appears all the available positions and its applications. 3. The member selects a student 4. The application makes the member to select a position that has applied and a criteria and according of the choosings, the application will show the most matched position for the student.
Alternative flow 1	If there are no available positions, the application will show a “No available positions found.”message
Post conditions	The member can select a student.

3.19 <Use Case 19>

Use case ID	Assign Student For a Traineeship position
Actors	Committee Member as a user of the Traineeship App
Pre conditions	Committee Member has created a profile and is logged into the application.
Main flow of events	1. The use case starts when the member clicks the “View Traineeship Positions” 2. The application appears all the available positions and its applications. 3. The member selects a student 4. The application makes the member to select a position that has applied and a criteria and according of the choosings, the application will show the most matched position for the student. 5. The member selects a student for a position and it returns back to see other available positions.
Alternative flow 1	If there are no available positions, the application will show a “No available positions found.”message
Post conditions	The member can assign a student to a position.

3.20 <Use Case 20>

Use case ID	Assign Professor For a Traineeship position
Actors	Committee Member as a user of the Traineeship App
Pre conditions	Committee Member has created a profile and is logged into the application.
Main flow of events	1. The use case starts when the member clicks “Search Professors” button 2. The application shows 2 selections form that the member will choose a position and a criteria. 3. The member clicks Find Professor. 4. The system will show the recommended professors starting from the most matched professor to the selected position according to the criteria. 5. The member assigns the professor to the position if wants to.
Alternative flow 1	If there are no available positions, the application will show a “No available positions found.”message
Alternative flow 2	If there are no matched professors the application will show a “No recommended professors found.”message
Post conditions	The member can assign a professor to a position

3.21 <Use Case 21>

Use case ID	Show List of Traineeshiop positions
Actors	Committee Member as a user of the Traineeship App
Pre conditions	Committee Member has created a profile and is logged into the application.
Main flow of events	1. The use case starts when the member clicks “Search Professors” button. 2. The application shows 2 selections form that the member will choose a position and a criteria. 3. The member clicks Find Professor. 4. The system will show the recommended professors starting from the most matched professor to the selected position according to the criteria. 5. The member assigns the professor to the position if wants to.
Alternative flow 1	If there are no available positions, the application will show a “No available positions found.”message
Alternative flow 2	If there are no matched professors the application will show a “No recommended professors found.”message
Post conditions	The member can assign a professor to a position

3.22 <Use case 22>

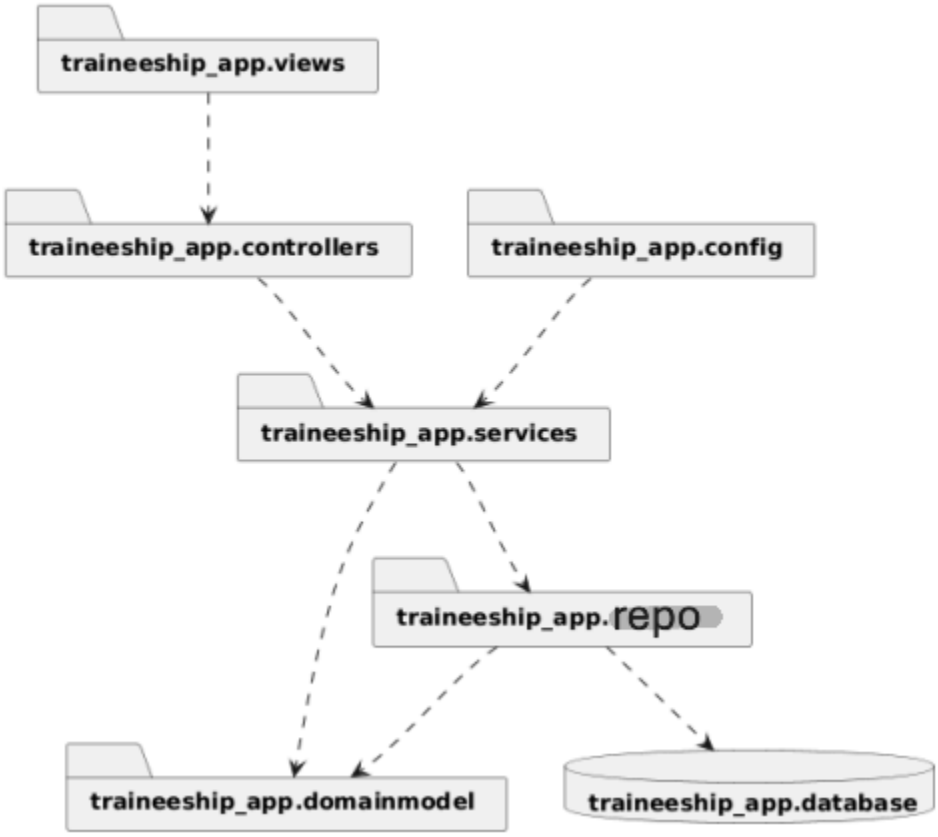
Use case ID	Select and Monitor Evaluations for a position
Actors	Committee Member as a user of the Traineeship App
Pre conditions	Committee Member has created a profile and is logged into the application.
Main flow of events	1. The use case starts when the member clicks “Search Professors” button. 2. The application shows 2 selections form that the member will choose a position and a criteria. 3. The member clicks Find Professor. 4. The system will show the recommended professors starting from the most matched professor to the selected position according to the criteria. 5. The member assigns the professor to the position if wants to.

	6. By clicking “View in Progress” button, the application will show all the in progress positions that are supervised by professors. 7. At the right side of each position, there is a “Monitor” button that gathers all evaluations by company and the supervisor/ 8. According to these evaluations, the member can mark Pass or Fail for the position.
Alternative flow 2	If there are no matched in progress positions, the application will show a “No positions found” message
Post conditions	The member can select a traineeship and monitor the evaluations of the selected position.

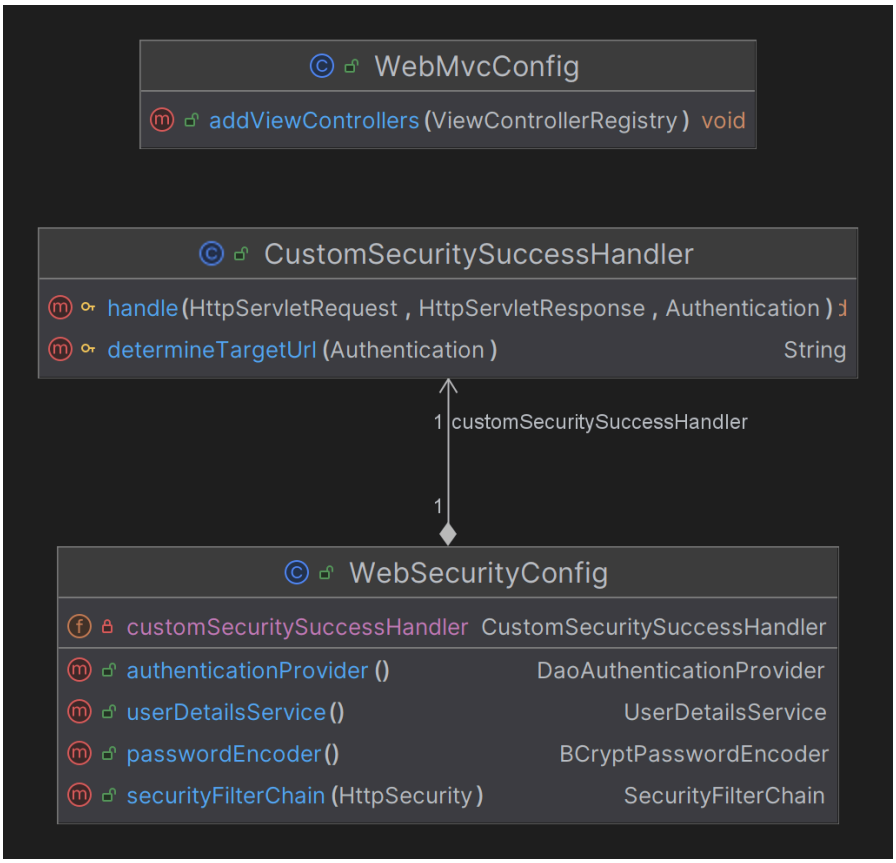
4.Design

a. \Architecture

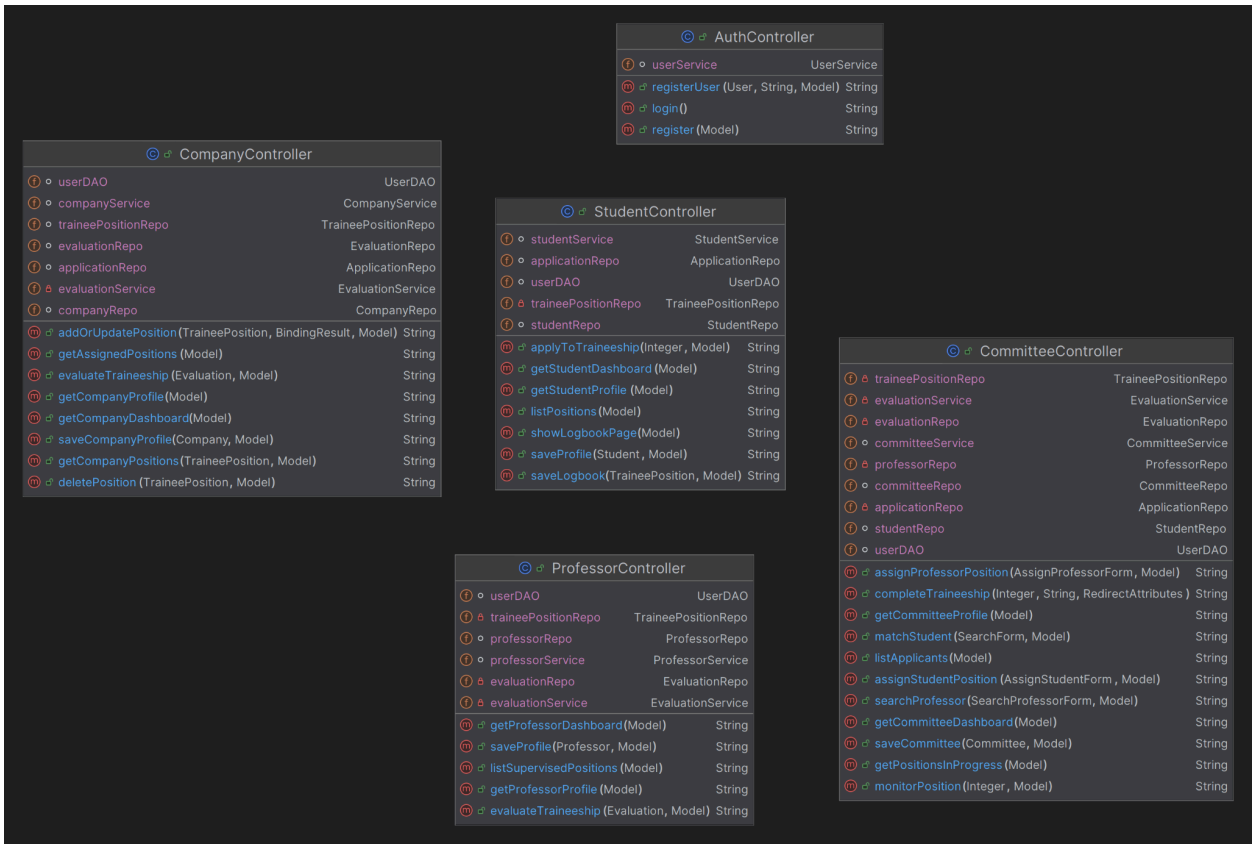
<Specify the overall architecture for this release in terms of a **UML package diagram**.>



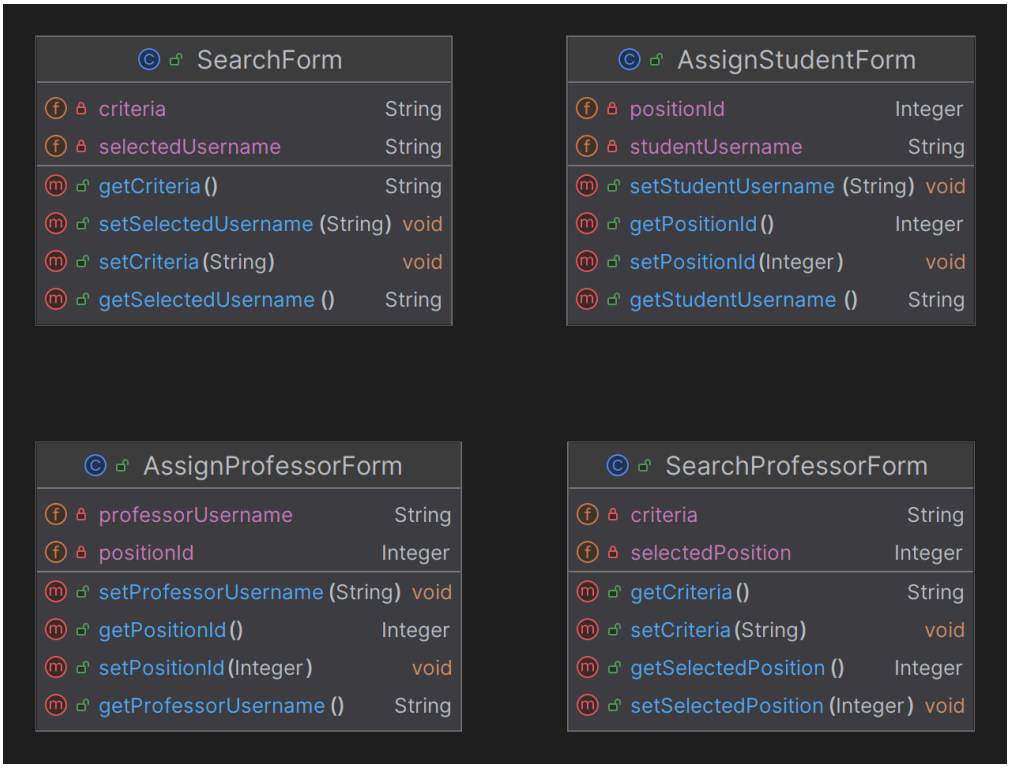
b. Design



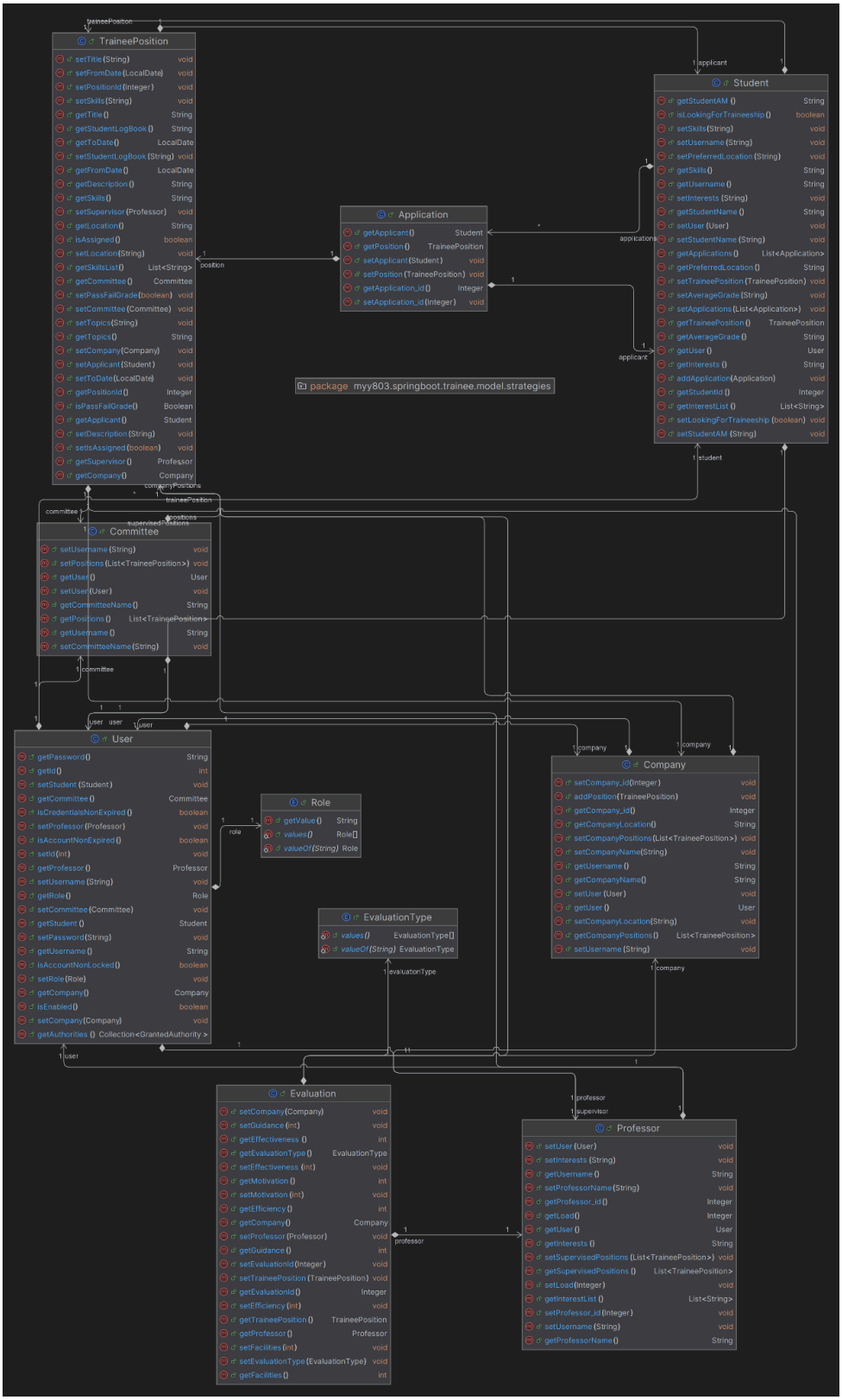
Package: myy803.spring boot.trainee.config



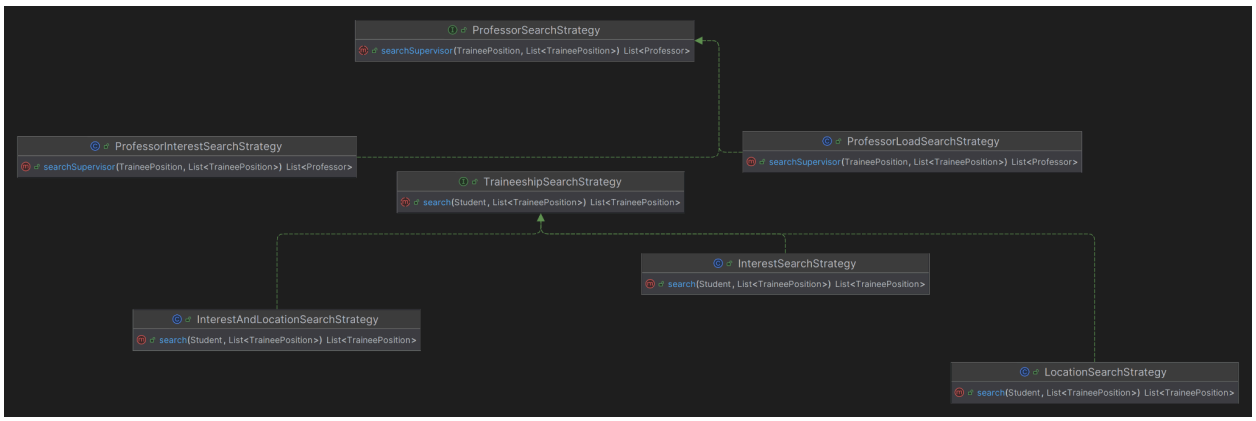
Package: myy803.springboot.trainee.controller



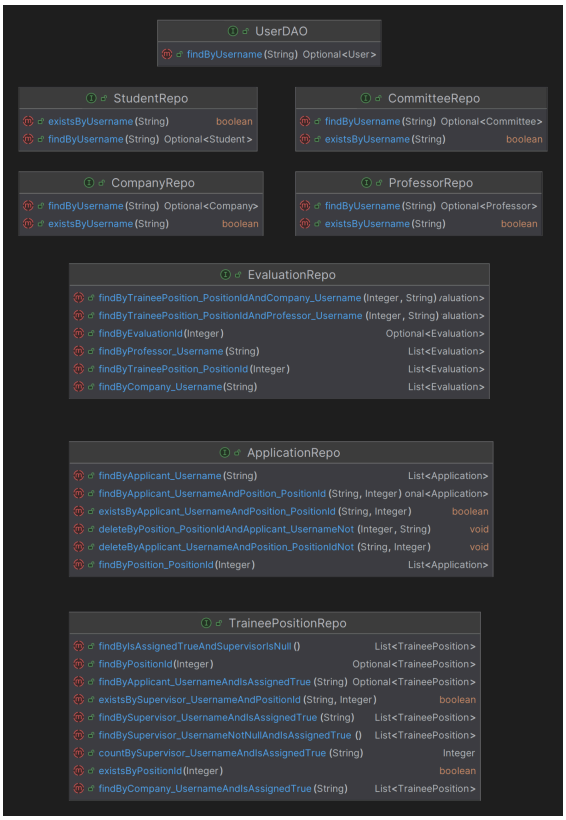
Package: myy803.springboot.trainee.formsdata



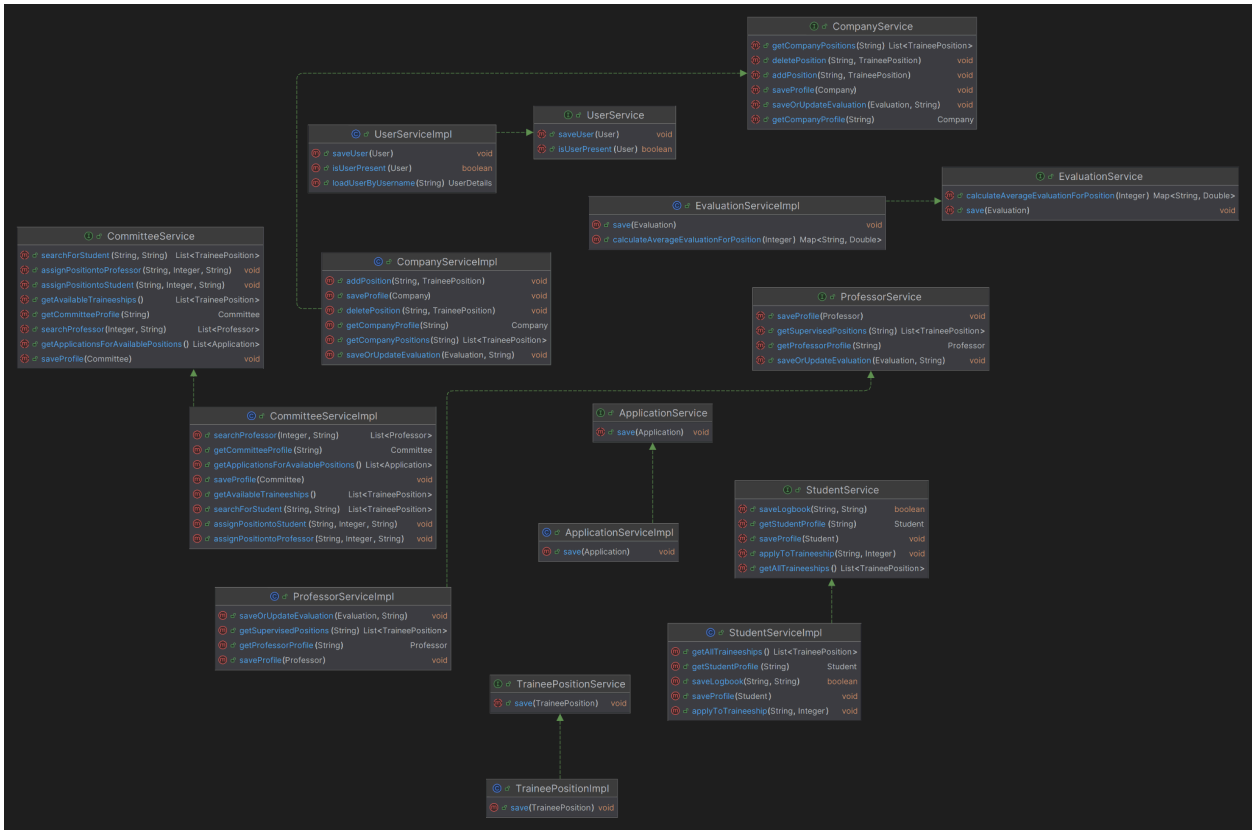
Package: myy803.springboot.trainee.model



Package: myy803.springboot.trainee.model.strategies



Package: myy803.springboot.trainee.model.repo



Package: myy803.springboot.trainee.service

<u>domainmodel Package:</u>

Class Name: Application	
Responsibilities: ▪ **Representing Application Data**: The Application class represents an application made by a Student for a TraineePosition	**Collaborations:** ▪ **Student Entity** (applicant):Collaborates with the Student entity through a many-to-one relationship (@ManyToOne) using the applicant attribute. This indicates that an Application is associated with one Student who is the applicant

▪ **Mapping to Database Entity**: Annotated with JPA annotations (@Entity, @Table, @Id, @Column, @ManyToOne, @JoinColumn, etc.), it maps the Application entity to the database table applications. ▪ **Attributes**: Defines attributes (application_id, applicant, position) and their corresponding getters and setters to manage application data.	▪ **TraineePosition Entity** (position):Collaborates with the TraineePosition entity through another many-to-one relationship (@ManyToOne) using the position attribute. This indicates that an Application is associated with one TraineePosition that the applicant is applying for. ▪ **Database Interaction**:Collaborates with the database through JPA (Java Persistence API) annotations (@Entity, @Table, @Id, etc.) to facilitate object-relational mapping (ORM) and database operations for Application entities. ▪ **Fetching Strategy**:Specifies eager fetching (FetchType.EAGER) for applicant and position, implying that when an Application entity is fetched, associated Student and TraineePosition entities are also fetched immediately.

Class Name: Committee	
Responsibilities: ▪ **Representing Committee Entity**:Models a Committee as a JPA entity, mapped to the committees table in the database. ▪ **Persistence and Identity Mapping**:Uses @Id and @MapsId to map the committeeId to a corresponding User entity, establishing identity sharing.	**Collaborations:** ▪ **User Entity** (user):Collaborates with the User entity in a one-to-one relationship using shared primary keys (@OneToOne with @MapsId). ▪ **TraineePosition Entity** (positions):Collaborates with multiple TraineePosition entities (@OneToMany) which the committee manages or oversees.

▪ **Managing Committee Attributes**:Manages key attributes like username, committeeName, and the list of TraineePosition objects it oversees. ▪ **Relationship Management**:Handles a one-to-one relationship with User and a one-to-many relationship with TraineePosition.	▪ **Database Interaction**: Collaborates with the database through JPA annotations to manage persistence, identity, and cascading of related entities (CascadeType.ALL).

Class Name: Company	
Responsibilities: ▪ **Representing Company Data**:Models a Company as a JPA entity corresponding to the companies table in the database. ▪ **Identity and User Linkage**:Shares a primary key with the User entity, representing that a Company is also a User. ▪ **Attribute Management**:Maintains details such as username, companyName, and companyLocation. ▪ **Managing Trainee Positions**:Holds a collection of TraineePosition entities (companyPositions) posted or owned by the company. ▪ **Position Operations**:Provides utility like addPosition() to dynamically add positions to the list.	**Collaborations:** ▪ **User Entity** (user):Collaborates with the User entity via a one-to-one relationship using @OneToOne and @MapsId, indicating that the Company shares its identity with a User. ▪ **TraineePosition Entity** (companyPositions):Collaborates with multiple TraineePosition entities in a one-to-many relationship (@OneToMany), representing the positions posted by the company. ▪ **Database Interaction**:Uses JPA annotations to handle:Table mapping (@Entity, @Table),Primary key strategy (@Id, @MapsId),Relationships and cascading (CascadeType.ALL),Fetch strategy (FetchType.LAZY for performance optimization).

Class Name: Evaluation	
Responsibilities: ▪ **Representing Evaluation Data**:Models an Evaluation entity for assessing trainee experiences, stored in the evaluations table. ▪ **Scoring System**:Manages numeric evaluation criteria (e.g., motivation, effectiveness, efficiency, facilities, guidance) to assess the performance or quality of a trainee's experience. ▪ **Handling Evaluation Type**:Uses an EvaluationType field to distinguish the nature or category of the evaluation. ▪ **Relational Mapping**:Links to associated entities (company, professor, trainee position) to establish context for each evaluation.	**Collaborations:** ▪ **TraineePosition Entity** (traineePosition):@ManyToOne relationship links the evaluation to the specific trainee position being evaluated. ▪ **Company Entity** (company):@ManyToOne relationship connects the evaluation to the company providing or reviewing the position, using the username field as a reference. ▪ **Professor Entity** (professor):@ManyToOne relationship represents the academic staff (professor) who may be responsible for evaluating the student or reviewing the company evaluation. ▪ **EvaluationType Enum** (evaluationType):Holds a custom enumeration that classifies the evaluation type (e.g., company evaluation, academic evaluation, etc.). ▪ **Database Interaction**:Utilizes JPA annotations for:Auto-generating primary keys (@GeneratedValue),Establishing foreign key relationships (@ManyToOne, @JoinColumn),Mapping entity to database table and columns.

Class Name: Professor

Responsibilities: ▪ **Representing Professor Data:** Models a Professor entity corresponding to the professors table, capturing academic supervisors within the system. ▪ **Attribute Management:** Manages core professor-related data such as username, professorName, and interests. ▪ **Workload Tracking:** Uses the load field (with default value 0) to track the number of positions or duties assigned to the professor. ▪ **Interest Parsing:** Provides the getInterestList() utility method to parse the comma-separated interests field into a list, improving usability.	**Collaborations:** ▪ **Identity and User Linkage:** Shares identity with the User entity using a one-to-one relationship (@OneToOne with @MapsId), indicating that a Professor is also a User. ▪ **Position Supervision:** Maintains a one-to-many relationship with TraineePosition via the supervisedPositions list, representing the positions the professor supervises. ▪ **Database Interaction:** Uses JPA annotations for entity mapping, identity management, and handling entity relationships, ensuring integration with the underlying database.

Class Name: Student	
Responsibilities: ▪ **Representing Student Data:** Models a `Student` entity mapped to the `students` table, capturing student details within the system.	**Collaborations:** ▪ **Identity and User Linkage:** Shares identity with the `User` entity via a one-to-one relationship (`@OneToOne` with `@MapsId`), indicating that a `Student` is also a `User`. ▪ **Applications Association:** Collaborates with multiple `Application` entities (`@OneToMany`) representing the trainee positions the student has applied to.

▪ **Attribute Management:** Manages student-related attributes such as `username`, `studentName`, `studentAM`, `averageGrade`, `preferredLocation`, `interests`, `skills`, and `lookingForTraineeship` status. ▪ **Interest Parsing:** Provides the `getInterestList()` method to convert the comma-separated `interests` string into a list for easier handling. ▪ **Application Management:** Maintains and manages a list of `Application` entities representing trainee position applications submitted by the student. ▪ **Trainee Position Linkage:** Maintains a one-to-one relationship with a `TraineePosition` that the student holds or is assigned to.	▪ **TraineePosition Association:** Maintains a one-to-one relationship with the `TraineePosition` entity, representing the position assigned or held by the student. ▪ **Database Interaction:** Utilizes JPA annotations for entity mapping, persistence, and relational integrity with other entities.

Class Name: Trainee Position	
Responsibilities: ▪ **Representing Trainee Position Data:** Models a `TraineePosition` entity mapped to the `trainee_positions` table, representing internship or training positions.	**Collaborations:** ▪ **Applicant Association:** Holds a one-to-one eager relationship with the `Student` entity, representing the student assigned or applying for this position.

▪ **Attribute Management:** Manages attributes such as title, description, start and end dates, location, topics, required skills, assignment status, student logbook, and pass/fail grade. ▪ **Skills Parsing:** Provides a utility method (`getSkillsList()`) to convert the comma-separated `skills` string into a list for easier processing. ▪ **Assignment Tracking:** Tracks whether a position is assigned to a student via the `isAssigned` boolean field.	▪ **Supervisor Association:** Maintains a many-to-one eager relationship with the `Professor` entity, indicating the academic supervisor overseeing the position. ▪ **Company Association:** Maintains a many-to-one eager relationship with the `Company` entity, representing the hosting company for the position. ▪ **Committee Association:** Maintains a many-to-one eager relationship with the `Committee` entity, indicating the committee responsible for the position. ▪ **Database Interaction:** Uses JPA annotations for entity mapping, relationship management, and persistence to the underlying database.

Class Name: User	
Responsibilities: ▪ **User Security Representation:** Implements `UserDetails` to integrate with Spring Security, providing essential user authentication and authorization details. ▪ **Role Management:** Manages user roles through an enumerated `Role` field, defining user authorities for security.	**Collaborations:** ▪ **Security Integration:** Collaborates with Spring Security's `GrantedAuthority` for user role management and authority granting.

▪ **Credential Handling:** Stores and provides access to username and password credentials required for authentication. ▪ **Account Status:** Implements methods to indicate account expiration, lock status, credential validity, and whether the account is enabled—all returning true for simplicity.	▪ **Database Interaction:** Uses JPA annotations for entity mapping, ID generation, and relationship management with cascading on related entities. ▪ **Bidirectional One-to-One Relationships:**Collaborates with Student entity, maintaining a bidirectional one-to-one mapping.Collaborates with Company entity with a bidirectional one-to-one mapping.Collaborates with Professor entity through a one-to-one bidirectional relationship.Collaborates with Committee entity, also through a one-to-one bidirectional mapping.

formsdata Package:

Class Name: AssignProfessorForm	
Responsibilities: ▪ **Data Transfer Object:** Acts as a simple container for form data related to assigning a professor to a trainee position. ▪ **Data Encapsulation:** Encapsulates the trainee position ID and the professor's username as fields. ▪ **Getter and Setter Methods:** Provides access and mutation methods for the `positionId` and `professorUsername` fields.	**Collaborations:** ▪ **Form Handling:** Collaborates with controller or service layers that handle the assignment of professors to trainee positions by carrying necessary input data. ▪ **Data Binding:** Used in frameworks (e.g., Spring MVC) to bind incoming form data from client requests.

Class Name: AssignStudentForm	
Responsibilities: ▪ **Data Transfer Object:** Holds form data used to assign a student to a trainee position. ▪ **Data Encapsulation:** Contains fields for the trainee position ID and the student's username. ▪ **Accessor Methods:** Provides getter and setter methods for both `positionId` and `studentUsername`.	**Collaborations:** ▪ **Form Processing:** Works with controller or service components responsible for assigning students to trainee positions by carrying input data. ▪ **Data Binding:** Facilitates binding of client-submitted form data in frameworks like Spring MVC.

Class Name: SearchForm	
Responsibilities: ▪ **Data Transfer Object:** Encapsulates user input for searching based on username and criteria.	**Collaborations:** ▪ **Search Functionality:** Used by controllers or services to receive and process search requests from users.

▪ **Search Criteria Management:** Holds the selected username and the search filter criteria (`location`, `interest`, or `both`). ▪ **Accessor Methods:** Provides getter and setter methods for `selectedUsername` and `criteria`.	▪ **Form Binding:** Facilitates binding of search form data in web frameworks such as Spring MVC.

repo Package:

Class Name: ApplicationRepo	
Responsibilities: ▪ **Data Access Layer:** Provides CRUD operations for `Application` entities using Spring Data JPA. ▪ **Custom Queries:** Defines methods for querying applications by applicant username and trainee position ID. ▪ **Existence Checks:** Offers method to check if an application exists for a given applicant and position. ▪ **Conditional Deletions:** Supports deleting applications based on conditions excluding specific usernames or position IDs.	**Collaborations:** ▪ **Application Entity:** Operates on `Application` entities in the database. ▪ **Spring Data JPA:** Extends `JpaRepository` to leverage built-in JPA repository functionalities. ▪ **Service Layer:** Used by service components to perform database operations related to trainee applications.

Class Name: CommitteeRepo

Responsibilities:	**Collaborations:**
▪ **Data Access Layer:** Provides CRUD operations for `Committee` entities using Spring Data JPA. ▪ **Find by Username:** Defines a method to retrieve a `Committee` by its username. ▪ **Existence Check:** Provides a method to check if a `Committee` exists with a given username.	▪ **Committee Entity:** Manages persistence of `Committee` entities in the database. ▪ **Spring Data JPA:** Extends `JpaRepository` to utilize JPA repository functionalities. ▪ **Service Layer:** Enables service components to query and validate committee data..

Class Name: CompanyRepo	
Responsibilities:	**Collaborations:**
▪ **Data Access Layer:** Provides CRUD operations for `Company` entities using Spring Data JPA. ▪ **Find by Username:** Defines a method to retrieve a `Company` by its username. ▪ **Existence Check:** Offers a method to verify if a `Company` exists with a specified username.	▪ **Company Entity:** Manages persistence and retrieval of `Company` entities from the database. ▪ **Spring Data JPA:** Extends `JpaRepository` to leverage JPA repository functionalities. ▪ **Service Layer:** Supports business logic by enabling queries and existence checks on companies.

Class Name: EvaluationRepo

Responsibilities:	**Collaborations:**
▪ **CRUD Operations:** Provides standard create, read, update, and delete operations for `Evaluation` entities via JpaRepository. ▪ **Find by Evaluation ID:** Retrieves an `Evaluation` by its unique evaluation ID. ▪ **Find by Position and Company:** Finds an `Evaluation` based on trainee position ID and company username. ▪ **Find by Position and Professor:** Finds an `Evaluation` based on trainee position ID and professor username. ▪ **Find by Position:** Retrieves all evaluations associated with a specific trainee position ID. ▪ **Find by Company:** Retrieves all evaluations related to a specific company username. ▪ **Find by Professor:** Retrieves all evaluations associated with a particular professor username.	▪ **Evaluation Entity:** Manages persistence and queries of `Evaluation` data. ▪ **TraineePosition Entity:** Uses position ID to filter evaluations by related trainee positions. ▪ **Company Entity:** Uses company username to filter evaluations by company. ▪ **Professor Entity:** Uses professor username to filter evaluations by professor. ▪ **Spring Data JPA:** Extends JpaRepository to utilize Spring's data access capabilities.

Class Name: ProfessorRepo	
Responsibilities:	**Collaborations:**
▪ **CRUD Operations:** Provides standard create, read, update, and delete operations for `Professor` entities via JpaRepository.	▪ **Professor Entity:** Manages persistence and queries of `Professor` data.

▪ **Find by Username:** Retrieves a `Professor` entity based on the unique username. ▪ **Check Existence by Username:** Determines whether a `Professor` exists with a given username.	▪ **Spring Data JPA:** Extends JpaRepository to leverage Spring's data repository functionalities.

Class Name: StudentRepo	
Responsibilities: ▪ **CRUD Operations:** Manages basic create, read, update, and delete operations for `Student` entities through JpaRepository. ▪ **Find by Username:** Retrieves a `Student` entity using the unique username. ▪ **Check Existence by Username:** Checks if a `Student` with a specified username exists in the database.	**Collaborations:** ▪ **Student Entity:** Works with `Student` objects for persistence and retrieval. ▪ **Spring Data JPA:** Utilizes JpaRepository to provide repository functionality.

Class Name: TraineePositionRepo	
Responsibilities: ▪ **CRUD Operations:** Manages basic create, read, update, and delete operations for `TraineePosition` entities via JpaRepository. ▪ **Find by Position ID:** Retrieves a `TraineePosition` by its unique position ID.	**Collaborations:** ▪ **TraineePosition Entity:** Works directly with `TraineePosition` objects for database interactions. ▪ **Spring Data JPA:** Uses JpaRepository methods and query derivation for repository functionality.

▪ **Find Assigned Positions:** Finds assigned trainee positions by applicant username, supervisor, or company. ▪ **Find Unsupervised Assigned Positions:** Retrieves assigned positions that currently have no supervisor. ▪ **Count Assigned Positions:** Counts how many assigned positions a supervisor has. ▪ **Existence Checks:** Verifies existence of `TraineePosition` by supervisor username and position ID or by position ID alone.	

Class Name: UserDAO	
Responsibilities: ▪ **CRUD Operations:** Provides basic create, read, update, and delete operations for `User` entities via JpaRepository. ▪ **Find by Username:** Retrieves a `User` entity based on its unique username.	**Collaborations:** ▪ **User Entity:** Interacts with the `User` entity for data persistence and retrieval. ▪ **Spring Data JPA:** Extends JpaRepository to leverage Spring Data JPA's repository functionality.

package : Controller

Class Name: AuthController	
Responsibilities:	**Collaborations:**

▪ **Handle Login Requests:** Serves the login page when the `/login` endpoint is accessed. ▪ **Serve Registration Page:** Provides the registration form via the `/register` endpoint. ▪ **Process User Registration:** Accepts user registration data at `/save`, validates the user, checks for existing users, saves new users, and returns appropriate views with messages.	▪ **UserService:** Uses `UserService` to check user existence and save new users. ▪ **User Model:** Works with the `User` model for binding form data and passing user information between the view and backend. ▪ **Spring MVC:** Uses Spring MVC annotations and model attributes for request handling and view rendering.

Class Name: CommitteeController	
Responsibilities: ▪ **Dashboard & Profile Management:** Serve committee member's dashboard and profile pages, handling profile creation and updates. ▪ **View & Manage Available Positions:** List traineeship positions open for assignment and show applications for these positions. ▪ **Match Students to Positions:** Facilitate searching for students based on criteria and assign selected students to positions. ▪ **Search & Assign Professors:** Allow searching professors by criteria to assign supervisors to traineeship positions.	**Collaborations:** **Repositories:** ● `UserDAO` for user data ● `CommitteeRepo` for committee entity persistence ● `StudentRepo`, `ProfessorRepo` for student and professor info ● `TraineePositionRepo` for traineeship positions ● `ApplicationRepo` for applications data ● `EvaluationRepo` for evaluations **Services:**

▪ **Monitor Positions in Progress:** Show currently ongoing traineeships and provide detailed monitoring with evaluations. ▪ **Complete Traineeships:** Mark traineeships as passed or failed based on evaluations and professor assignments.	● `CommitteeService` handles business logic around committee profiles, position assignment, and searching. ● `EvaluationService` calculates average evaluation metrics for positions. **Security:** Uses Spring Security's `SecurityContextHolder` to obtain the currently logged-in committee user. **Forms & Models:** Uses form data classes (`AssignStudentForm`, `AssignProfessorForm`, `SearchForm`, `SearchProfessorForm`) to bind user inputs. **Spring MVC Components:** Annotations like `@Controller`, `@RequestMapping`, `@PostMapping`, and model attributes for passing data to views.

Class Name: CompanyController	
Responsibilities: **Dashboard & Profile Management** ● Shows company dashboard with profile details.	**Collaborations:** None.

● Displays the company profile form for editing or creating. ● Saves/updates the company profile in the system. **Managing Trainee Positions** ● Lists all trainee positions for the logged-in company. ● Shows which positions have evaluations already done. ● Handles adding new trainee positions or updating existing ones. ● Deletes trainee positions. **Assigned Positions** ● Displays all trainee positions currently assigned to supervisors (professors). **Traineeship Evaluation** ● Provides evaluation form to companies for their traineeships. ● Saves or updates the evaluation for a traineeship submitted by the company. ▪	

Class Name: ProfessorController	
Responsibilities: **Profile Management:**	**Collaborations:** **With Trainee Positions:**

● Retrieve and display professor profile data. ● Save or update professor profile information. **Supervised Trainee Positions:** ● List trainee positions assigned to the professor as a supervisor. ● Show which positions have been evaluated by the professor. **Evaluation Handling:** ● Allow professors to evaluate trainee positions they supervise. ● Save or update evaluations for those trainee positions.	● Professors supervise trainee positions. ● Professors evaluate the progress and performance of trainees in these positions. **With Evaluations:** ● Professors create and update evaluation records tied to trainee positions. ● Evaluation data tracks collaboration between professors and trainees/positions. **With Companies (Indirect):** ● Trainee positions belong to companies, so professors collaborate indirectly with companies through the positions they supervise. **With Users:** ● Professors are linked to authenticated users to manage personalized profile and evaluation data.

Class Name: StudentController	
Responsibilities: **Profile Management:**	**Collaborations:** ● **With Trainee Positions:**

● Retrieve and display the student's profile. ● Save or update student profile information. **Traineeship Positions Management:** ● Show available trainee positions to the student. ● Display assigned trainee position if the student already has one. ● Allow students to apply to trainee positions. **Logbook Management:** ● Display the logbook page for the assigned trainee position. ● Save updates to the student's logbook during the traineeship.	○ Students apply for trainee positions. ○ Students are assigned to trainee positions. ○ Students maintain logbooks related to their assigned trainee position. ● **With Applications:** ○ Student applications are tracked for positions they apply to. ○ Used to filter out positions already applied to by the student. ● **With Users:** ○ Students are linked to authenticated users to manage personalized data. ● **With Services and Repositories:** ○ Collaborates with `StudentService` for business logic (profile, applications, logbook). ○ Uses `TraineePositionRepo` to query trainee positions. ○ Uses `ApplicationRepo` to track applications submitted by the student. ○ Uses `StudentRepo` and `UserDAO` for user/student data persistence and retrieval.

package : service

Class Name: ApplicationServiceImpl	
Responsibilities: ▪ Implements the `ApplicationService` interface to provide the actual logic for saving `Application` entities. ▪ Saves `Application` objects to the database using the `ApplicationRepo` repository. ▪ (Note: Although `BCryptPasswordEncoder` is autowired, it is currently **not used** in this class. Possibly for future use or leftover from a template.	**Collaborations:** ▪ **ApplicationRepo (Repository Layer):** Collaborates directly with `ApplicationRepo` to persist `Application` data to the database. ▪ **BCryptPasswordEncoder:** Injected but unused in current code; typically used for password encoding, but irrelevant here unless extended later. ▪ **Application (Entity):** Works with the `Application` model representing trainee applications. ▪ **ApplicationService (Interface):** Implements the service interface, fulfilling the contract for application-related operations.

Class Name: CommitteeServiceImpl	
Responsibilities:	**Collaborations:**

▪ Manage **Committee** profiles: create, update, and retrieve committee data. ▪ Retrieve and manage lists of **Applications** for available trainee positions. ▪ Identify available **TraineePosition** objects (unassigned positions). ▪ Provide search functionality for traineeship positions tailored to **Student** criteria using different strategies (interest, location, or both). ▪ Assign a trainee position to a student, updating related entities and cleaning up redundant applications. ▪ Search for suitable **Professor** supervisors based on criteria (interest or load). ▪ Assign a trainee position to a professor and update their workload. ▪ Ensure transactional consistency when assigning positions to students or professors.	**CommitteeRepo:** CRUD operations on `Committee` entities. **TraineePositionRepo:** Retrieve and update `TraineePosition` entities, count assignments, and find assigned/unassigned positions. **ApplicationRepo:** Manage `Application` entities related to trainee positions and applicants. **StudentRepo:** Retrieve `Student` profiles and validate existence. **ProfessorRepo:** Retrieve and update `Professor` entities. **Search Strategies (Strategy Pattern):** ● `InterestSearchStrategy`, `LocationSearchStrategy`, `InterestAndLocationSearchStrategy` for student searches. ● `ProfessorInterestSearchStrategy`, `ProfessorLoadSearchStrategy` for professor searches. **Entities:** ● `Committee`, `Student`, `Professor`, `TraineePosition`, `Application`.

Class Name: CompanyServiceImpl	
Responsibilities: ▪ Manage **Company** profiles (create, update, retrieve). ▪ Retrieve a company's trainee positions. ▪ Add new trainee positions to the company. ▪ Delete trainee positions and clean up related applications. ▪ Save or update **Evaluations** for traineeships from the company's perspective. ▪ Ensure consistency when handling evaluations linked to trainee positions and companies.	**Collaborations:** ▪ **CompanyRepo:** CRUD operations on `Company` entities. ▪ **TraineePositionRepo:** Manage `TraineePosition` entities linked to companies. ▪ **ApplicationRepo:** Manage applications related to trainee positions, especially when deleting positions. ▪ **EvaluationRepo:** Save and retrieve evaluations made by the company on trainee positions. ▪ **Entities involved:** **`Company`, `TraineePosition`, `Application`, `Evaluation`, `EvaluationType`.**

Class Name: EvaluationServiceImpl	
Responsibilities: ▪ Save `Evaluation` entities to the database. ▪ Calculate average evaluation scores for a specific trainee position based on collected evaluations.	**Collaborations:** ▪ **EvaluationRepo:** Used to save evaluations and retrieve evaluations by trainee position ID. ▪ Returns averages in a `Map<String, Double>` keyed by the evaluation metric name.

Class Name: ProfessorService	
Responsibilities: ▪ Save or update a professor's profile, including their name, interests, and supervised positions. ▪ Retrieve a professor's profile by username. ▪ Fetch all trainee positions supervised by a professor (where positions are assigned). ▪ Save or update evaluations submitted by professors for trainee positions. ▪ Handles creation or updating of evaluation records linked to a specific trainee position and professor.	**Collaborations:** **ProfessorRepo:** ● To find and save professor profiles. **TraineePositionRepo:** ● To retrieve supervised trainee positions. **EvaluationRepo:** ● To find, save, and update evaluation data. **Entities involved:** ● `Professor`, `TraineePosition`, `Evaluation`, and `EvaluationType` (specifically marks evaluations as by PROFESSOR).

Class Name: StudentServiceImpl	
Responsibilities:	**Collaborations:** **StudentRepo:** ● To find and save student profiles.

▪ Save or update a student's profile with details like name, AM (student ID), average grade, preferred location, interests, skills, and traineeship status. ▪ Retrieve a student profile by username. ▪ Retrieve all unassigned (available) trainee positions. ▪ Allow a student to apply to a traineeship position. ▪ Save or update the logbook content for a student's assigned trainee position. ▪ (Commented out) Contains user presence check and Spring Security user loading method.	**TraineePositionRepo:** ● To find trainee positions, both all and by specific criteria like assigned/unassigned and applicant username. **ApplicationRepo:** ● To save applications made by students to trainee positions. **Entities involved:** ● `Student`, `TraineePosition`, `Application`.

Class Name: TraineePositionImpl	
Responsibilities: ▪ Save or update a `TraineePosition` entity to the database.	**Collaborations:** **TraineePositionRepo:** ● To persist the `TraineePosition` entity.

Class Name: UserServiceImpl	
Responsibilities:	**Collaborations:**

▪ Save a new `User` with an encoded password. ▪ Check if a user already exists by username. ▪ Load a user by username for Spring Security authentication.	**BCryptPasswordEncoder:** ● To encode user passwords before saving. **UserDAO:** ● To persist and retrieve `User` entities from the database.

package : config

Class Name: CustomSecuritySuccessHandler	
Responsibilities: ▪ **Redirect users after successful login** based on their assigned roles. ▪ **Determine the appropriate target URL** (dashboard or home page) according to user roles such as STUDENT, PROFESSOR, COMPANY, or COMMITTEE. ▪ **Prevent multiple redirects** by checking if the response is already committed. ▪ **Invoke Spring Security's RedirectStrategy** to perform HTTP redirects cleanly.	**Collaborations:** **Spring Security Framework**: ● Implements and extends `SimpleUrlAuthenticationSuccessHandler` from Spring Security. ● Uses `Authentication` object provided by Spring Security to inspect the user's authorities/roles. ● Uses `GrantedAuthority` objects to identify user roles. **Servlet API**:

▪ **Fallback to login page with error** if no recognized role is found.	● Uses `HttpServletRequest` and `HttpServletResponse` to control the HTTP request and response cycle. **RedirectStrategy** (from Spring Security): ● Collaborates with `DefaultRedirectStrategy` to handle HTTP redirects. **Your Application's Role Definitions**: ● Collaborates implicitly with your user roles configuration (like STUDENT, PROFESSOR, etc.) defined in your security configuration and UserDetails implementations.

Class Name: WebMvcConfig	
Responsibilities: ▪ **Configure simple automated controllers** that map specific URLs directly to views without needing a full controller class. ▪ **Map the root URL ("/") to the view named "homepage"** — so when users access the base URL of your app, they see the homepage. ▪ **Customize Spring MVC behavior** by implementing the `WebMvcConfigurer` interface.	**Collaborations:** **Spring Framework MVC module**: ● Implements `WebMvcConfigurer` to customize the MVC configuration. ● Works with `ViewControllerRegistry` to register view controllers. **Spring Boot View Resolver**:

	● Relies on Spring's view resolver infrastructure to resolve `"homepage"` to the actual template (e.g., a Thymeleaf or JSP page). **HTTP requests**: ● Indirectly collaborates with the incoming HTTP requests for `/` path by routing them to the configured view.

Class Name: WebSecurityConfig	
Responsibilities: ▪ Configure Spring MVC to map specific URLs directly to views without needing dedicated controller classes. ▪ Register a view controller that maps the root URL (`"/"`) to the view named `"homepage"`. ▪ Enable simple page navigation by automating URL-to-view mapping.	**Collaborations:** ▪ Works with Spring's `ViewControllerRegistry` to register URL-to-view mappings. ▪ Relies on Spring MVC's infrastructure to resolve view names to actual templates (e.g., Thymeleaf, JSP). ▪ Interacts indirectly with incoming HTTP requests by defining how requests to `/` are handled. ▪

Strategies Package:

Class Name: InterestAndLocationSearchStrategy	
Responsibilities: ▪ Implements the TraineeshipSearchStrategy interface.. ▪ Searches for suitable TraineePosition objects for a given Student based on: ▪ Matching the student's preferred location with the position's location. ▪ Calculating the Jaccard similarity between the student's interests and the position's required skills. ▪ Filtering and ranking positions by the number of shared interests/skills (above a similarity threshold). ▪ Returns a sorted list of matching TraineePosition objects	**Collaborations:** ▪ Student (provides interests and preferred location) ▪ TraineePosition (provides location, skills, and assignment status) ▪ TraineeshipSearchStrategy (interface implemented by this class)

Class Name: InterestSearchStrategy	
Responsibilities: ▪ Implements the TraineeshipSearchStrategy interface.. ▪ Searches for suitable TraineePosition objects for a given Student based on:	**Collaborations:** ▪ Student (provides interests and preferred location) ▪ TraineePosition (provides location, skills, and assignment status)

▪ Calculating the Jaccard similarity between the student's interests and the position's required skills. ▪ Filtering positions with similarity above a defined threshold. ▪ Ranking positions by the number of shared interests/skills. ▪ Returns a sorted list of matching TraineePosition objects.	▪ TraineeshipSearchStrategy (interface implemented by this class)

Class Name: LocationSearchStrategy	
Responsibilities: ▪ Implements the TraineeshipSearchStrategy interface.. ▪ Searches for suitable TraineePosition objects for a given Student based on: ▪ Matching the student's preferred location with the position's location. ▪ Filtering out positions that are already assigned or have null values. ▪ Returns a sorted list of matching TraineePosition objects.	**Collaborations:** ▪ Student (provides interests and preferred location) ▪ TraineePosition (provides location, skills, and assignment status) ▪ TraineeshipSearchStrategy (interface implemented by this class)

Class Name: ProfessorInterestSearchStrategy	
Responsibilities: ▪ Implements the ProfessorSearchStrategy interface ▪ Searches for suitable Professor supervisors for a given TraineePosition based on: ▪ Calculating the Jaccard similarity between the position's required skills and each professor's interests. ▪ Filtering professors with similarity above a defined threshold. ▪ Returns a list of matching Professor object	**Collaborations:** ▪ Professor (provides interests) ▪ TraineePosition (provides required skills) ▪ ProfessorRepo (provides access to all professors) ▪ ProfessorSearchStrategy (interface implemented by this class)

Class Name: ProfessorLoadSearchStrategy	
Responsibilities: ▪ Implements the ProfessorSearchStrategy interface. ▪ Searches for suitable Professor supervisors for a given TraineePosition based on: ▪ Calculating the Jaccard similarity between the position's required skills and each professor's interests. ▪ Filtering professors with similarity above a defined threshold. ▪ Returns a list of matching Professor objects	**Collaborations:** ▪ Professor (provides interests) ▪ TraineePosition (provides required skills) ▪ ProfessorRepo (provides access to all professors) ▪ ProfessorSearchStrategy (interface implemented by this class)