Bilkent University

Department of Computer Science

CS319 Project

GOATs: ohmygoat.com

Section: 02

Group: 2d

Design Report

Group Member

- Beste Güney (21901631)
- Doruk Onur Çalışkan (21902672)
- Efe Ertürk (21902620)
- Ege Ergül (21902240)
- Emre Erdal (21901597)
- Kerem Erdal (21901596)

Instructor: Eray Tüzün

1.Purpose of the System

1.1 Purpose of The System

Our project is basically a student club management app. Its primary goal is to put things in order and make it easier for us (students) and club executives. In this way, students will be able to obtain information about the clubs more easily. For example, students can join a club, view the events of a particular club, ask their questions about the club or any particular event on the respected forums. On the other hand, club executives and advisors will be able to get the attendance of events from this application, fill and check the necessary documentation on this application and forward it to an upper echelon. Also, managing club members involving accepting or kicking a member out of the club and many other club related work can be done by club executives. In short, we aim to digitalize the interactions that exist between a student and a club.

1.2 Design Goals

Some of the indicated non-functional requirements from the analysis report are going to determine the design goals of the project. Our web system will have a user-friendly and straightforward interface; it will be functional, maintainable and also secure. There will be numerous functionalities of the system to enhance the user experience in the program. However, among all of these design goals, the most important two are usability and maintainability for the following reasons.

1.2.1 User Friendliness

UI / UX design of a system, application, or software are the most important requirements. Since the majority of our users will be students (mainly in the age interval of 18-25) it is important to make UI dynamic and favorable for young people. Yet, while doing this, our UI should remain simple and usable. Also the whole purpose of this project is to facilitate the jobs of Bilkent clubs and its members. For students, this means being able to see and get informed about club activities rapidly. Thus, the majority of the students will expect a mobile version of our system. For these reasons, our project will be mobile responsive.

Some test cases for ensuring we satisfy our primary design goal are as follows: Using different font sizes to draw attention to headings, making the minimum font size (for bodies) 16 px, use contrast colors for background and texts of components, giving vivid colors to the component that need extra attention (“Go to event” button in an event component can be red), using mobile first approach and creating a responsive design, use background shadows for components like events or buttons (this will gain a sense of third dimension which will make it easier to distinguish components from each other).

1.2.2 Maintainability

The second most important aim of this project is to create a system that can be used without problems for a long time. Therefore, the system should be easily maintainable and open to changes according to user feedback. Assuming and hoping that our project gets selected for being used next semester, we will be responsible for maintaining our application. If not all, many of our group members haven’t maintained a system that is actively used by hundreds of users. We can report many bugs or we can be asked for certain improvements. Then, we will have a short time to modify our system since we will be responsible against our peers, professors and school.

Using OOP ensures that adding new functionality or modifying the existing ones will not be a problem if the abstractions are designed and implemented carefully and dependencies among classes and subsystems are minimal. The design of the project will be done in such a way that a change will not cause unintended consequences and crash some other part of the project. Decompositions of the systems will be done as thoroughly as possible to ensure this goal.

We will check how long it takes for our team to respond to a reported bug and estimate a range for fixing a bug and notify our users. New feature requests by the users are planned to be added between two semesters.

2.Proposed Software Architecture

2.1 Subsystem decomposition

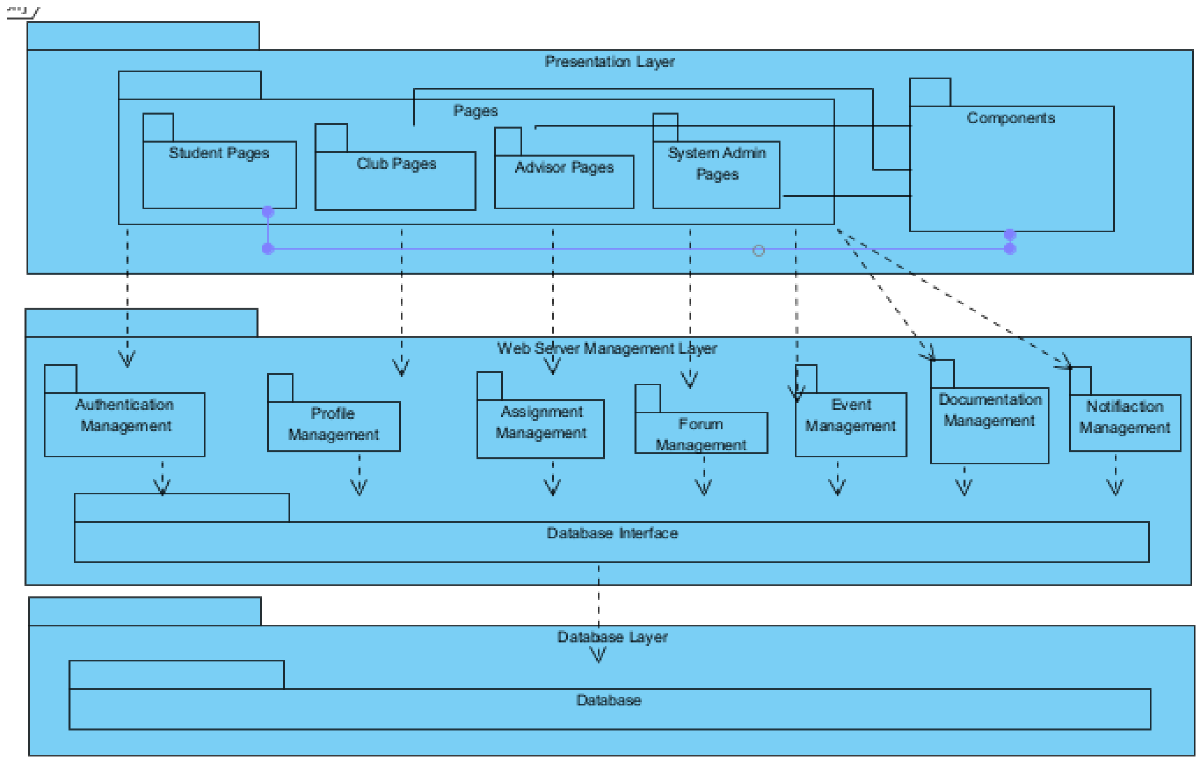


Fig. 2.2 Diagram showing the subsystems at the project

We decided to use a multitier architecture for our subsystem decomposition. We decided to use 3 layers to maintain a high abstraction, which is already explained to be one of our priority design choices. When we have 3 layers, we drastically deduce coupling among subsystems which may be bad in terms of performance but good for increasing the maintainability (see 3.1.5 for preferred design trade-offs). The highest layer is for the frontend and the underlying 2 layers are for the backend.

2.1.1 Presentation Layer

In the presentation layer, we will deal with the user interface components. Users will be shown UI components that are managed by the application layer. The data gathered from the management subsystems will be shown to the user and according to the actions of the end-user, new data, modified data and deleted data will be updated on the database. These low-level interactions are abstracted from the presentation layer and handled in the backend layers.

- Components package: all the small components that can be reused will take place here, such as a navigation bar.
- Pages package: there are four packages;
    - Student pages
    - Club pages
    - Advisor pages
    - System Admin Pages

  All of these packages will be using components and will display those components in a formatted way to present different layouts for different pages.

2.1.2 Web Server Layer

The Web Server Management Layer can be considered as the bridge between the frontend and the database. In this layer, we will implement the REST CRUD API. This API will be responsible for transferring data from the presentation layer to the database layer. In this layer, we will provide endpoints. When a request is made from the frontend layer, the data will be gathered from the database layer and provided to the frontend. The reverse of this procedure will also be handled within this layer. The validation of the requests is also handled in this layer. This layer will also manipulate the database according to the actions coming from the application layer. Since this layer is the only layer that can interact with the database layer, we will ensure the security of the database in this layer.

- **Authentication Management:** Authentication Management package will be responsible for managing the operations and functions related to authenticaiton functions which will be explained in **3.4.2.**
- **Profile Management:** Profile Management package will be responsible for managing the operations and functions related to profile functions which will be explained in **3.4.2.**
- **Assignment Management:** Assignment Management package will be responsible for managing the operations and functions related to assignment functions which will be explained in **3.4.2.**
- **Forum Management:** Forum Management package will be responsible for managing the operations and functions related to forum functions which will be explained in **3.4.2.**
- **Event Management:** EventManagement package will be responsible for managing the operations and functions related to event functions which will be explained in **3.4.2.**
- **Document Management:** Document Management package will be responsible for managing the operations and functions related to document functions which will be explained in **3.4.2.**

- **Notification Management:** Notification Management package will be responsible for managing the operations and functions related to notiifcation functions which will be explained in **3.4.2.**
- **Database Interface:** In order to ensure that each individual part will not harm the database (that is the developer team who will be responsible for, let's say, Profile Management will not accidentally override or delete some data from the database, all of these parts will be interacting with the database interface. Only the database interface will be interacting with the database.

2.1.3 Database Layer

The database layer is a self-explanatory layer. All the data is stored here. This layer will be interacting only with the business layer. Thus, no unintended modifications of data can be made. This layer contains the entity objects and their relations and their repositories. Thus, all the object-related logic will be handled here.

2.2 Hardware/software mapping

Our student club management project does not require any specialized hardware components to run successfully. We will create our project using react.js version 17.0.2. On the backend side, we will use Spring Framework version 5.0.

Our project runs on the web; hence web browsers that support react.js are needed, and hardware systems should be powerful enough to run a web browser. A web browser is mandatory to use this program, it can be used on phones and personal computers. For personal computers, a keyboard, a mouse, and a monitor are essential. For mobile browsers, an actively working touch-screen and a keyboard are required. Our project can run in web browsers like Google Chrome, Mozilla Firefox, Internet Explorer, Safari, and Opera in their latest versions. However, APIs and external libraries that we have used may be incompatible with the older web browsers.

A generic estimation for how much RAM our browser tab will require is approximately 300 MB. To come up with such an estimation, we opened Google Chrome v-96.0.4664.45 and entered the SRS page of our school. Since SRS required about 300 MB, we estimated that our app would also require about 300MB of RAM on a standardized browser.

On the other hand, launching a project that is written with React would require about 4GB of RAM according to our prior experiences. Thus, we know that launching requires much more Hardware than viewing. But since the user side concern is viewing, we are focusing on the hardware requirements for viewing.

For initiating a web system with a database, a minimum of 2x1.6 GHz CPU is required,but 4x1.6 GHz CPU is recommended [3]. For the memory, a minimum of 40 GB free space is required.

2.3 Persistent data management

In our project, PostgreSQL will be used. The reason we chose PostgreSQL is because PostgreSQL is an object relational database, which helps applying OOP principles, and our team had the most prior knowledge and experience in PostgreSQL. The entity classes and their corresponding objects in our program will be deployed as modals and tables into our database, which is deployed in the cloud by using Azure, and will be edited by using services in the Web service management layer's Database interface subsystem. Users will be able to edit the objects (profile, clubs that the student is a member of) anytime using the website, so it is important to send GET, POST, DELETE and PUT requests to the database dynamically and asynchronously. The REST CRUD API will be implemented in this manner using Java.

Firstly, we will store users' bilkent mails and passwords (with a Salt encrypted format if we need to). Also, the profile details of a student like name, school number, ge250 points etc. will be stored in the database. Clubs and their profile details will also be stored, as well as their events and its details. We will also store a student's joined clubs, and their roles in these respected clubs in a dictionary-esque format. List of assignments of a club and the documents submitted to these assignments will also be stored.

2.4 Access control and security

Our student club management project enforces security and access control. One of the most important ways we provide this is by giving user-type-specific permissions to each user type. A more in depth explanation can be seen on table 1 and 2 on part 3.1 "Access Matrix". We implement polymorphism to enable a user to exist in different types of forms [1]. For example, someone can be a board member of a club and at the same time that same person can be a member of a different club. Thus, permissions and roles of a user are bound dynamically, with respect to the club that one is examining. To continue on the previous example, the user may create an event on the club that he is a board member of, but in another club, he may not have the permissions to do so.

We aim to enhance the security of the product by Salt hashing every password before we store them in our database [2]. This prevents the decryption of the passwords in case of a security leak which results in leaking the passwords in the database. Also, our system sends only the required information to the frontend to be rendered to be more secure against url injection attacks. Data flow will be done via API's after they are encrypted if needed, thus it will be harder for a hacker to interrupt the traffic and steal data.

2.5 Boundary conditions

2.5.1 Initialization

In order to get the backend side of our project up and running, we will use Azure's server. Initiating the backend project on this server for one time will be enough to get our backend project up and running. The frontend side of our project needs to be hosted on same server. We will use Azure's hosting services for this purpose. After this point, no further initializations will be required. Since our application is a web based system, one can simply open a browser and search for our system. When users try to access our website, they will be directed to the login page. After logging in, they will be authorized and will be able to access their profile page according to their user type (Students will be directed to the student main page, the instructor will be directed to the instructor main page, etc.). If they don't have an account, they can access their home pages after completing the registration which contains verifying their mail address.

2.5.2 Termination

In order to terminate our system, the above mentioned servers should be terminated. To kill our backend project, Azure's server that runs the backend project needs to be terminated. Also by canceling the host that hosts the frontend project, the complete system will be terminated.

Our application gets terminated when users sign out. Alternatively, if users close the tab/window without signing out, they will be signed out automatically. Also if they remain inactive for 15 minutes, the system will automatically sign out the current user. Since all the related data will be saved immediately after an action is completed, there will be no data to be saved before logging off (For example, an event will be created and saved to the database, then the user will be prompted to the main menu and only then they can log off). Thus, termination doesn't require any automatic data saving.

2.5.3 Failure

In case there are some interruptions on the hosting of our system, these problems will be dealt with by Azure.

If during a session, the internet connection gets lost, the user will be signed off automatically and directed to a page saying, “Ups, internet connection is disrupted! Please make sure your device has an active internet connection.” If such an internet connection happens during an action that needs editing (Posting to a forum, creating an event), made changes will be lost. To minimize the amount of loss, on a regular basis, the changes made on these components will be saved to the page as cookies. Thus, when the user logs in again when they establish an active connection, the system will provide the user the last saved version of their work. Another scenario is that a user's internet can be disrupted during the procedure of uploading files or posting forums/events to the database. Unfortunately, if the connection gets lost at these processes, the upload procedure will automatically fail. Fortunately, the saved data will be deleted from cookies only when the action is fully completed. This means that users will be able to see their data when they log in again.

3.Low-level Design

3.1 Access Matrix

	Club	**Event**	**Forum**

Student	view()
	listClubs()
	findClub()
	joinClub()
		view()
		listEvents()
		findEvent()
			view()
			reportPost()

Club Member	view()
	listClubs()
	findClub()
	leaveClub()
		view()
		listEvents()
		findEvent()
		joinEvent()
		unjoinEvent()
			view()
			post()
			editPost()
			reportPost()

Active Club Member	view()
	listClubs()
	findClub()
	leaveClub()
		view()
		listEvents()
		findEvent()
		joinEvent()
		unjoinEvent()
			view()
			post()
			editPost()
			replyToPost()
			reportPost()

Club Board Member	view()
	listClubs()
	findClub()
	editClubProfile()
		view()
		listEvents()
		findEvent()
		createEvent()
		editEvent()
		deleteEvent()
		joinEvent()
		unjoinEvent()
			view()
			post()
			editPost()
			replyToPost()
			deletePost()

Club President	view()
	listClubs()
	findClub()
	editClubProfile()
	createSubClub()
		view()
		listEvents()
		findEvent()
		createEvent()
		editEvent()
		deleteEvent()
		joinEvent()
		unjoinEvent()
			view()
			post()
			editPost()
			replyToPost()

Club Advisor	view()
	listClubs()
	findClub()
		view()
		listEvents()
		findEvent()
			view()

Admin	view()
	listClubs()
	findClub()
	createClub()
	deleteClub()
		view()
		listEvents()
		findEvent()
			view()

Table 1: Access matrix of all users/actors in categories Clubs, Events and Forums

	Document	**Roles**
Student		
Club Member		

Active Club Member	view()
	submitAssignment()	

Club Board Member	view()
	submitAssignment()
	setAssignmentofActiveMembers()
	checkAssignmentsOfActiveMember()
	viewBudgetDocument()
	editBudgetDocument()
	viewClubDocuments()
	editClubDocuments()
		promoteToActiveMember()
		demoteActiveMember()
		acceptJoinRequest()

Club President	view()
	submitAssignment()
	setAssignmentofBoardMembers()
	checkAssignmentsOfBoardMember()
	viewBudgetDocument()
	editBudgetDocument()
	viewClubDocuments()
	editClubDocuments()
		promoteToBoardMember()
		demoteBoardMember()

Club Advisor	setAssignmentOfPresident()
	checkAssignmentsOfPresident()
	viewBudgetDocument()
	viewClubDocuments()
		promoteToPresident()
		demotePresident()

Admin	viewBudgetDocument()
	viewClubDocuments()
		promoteToClubAdvisor()
		demoteClubAdvisor()

Table 2: Access matrix of all users/actors in categories Document and Roles

3.2 Object Design Trade-offs

3.2.1 Usability over Functionality

As mentioned in section 1.2, our primary design goals are usability and maintainability. Thus, it is not a surprise that usability is preferred among usability. Since the system that we are building needs to facilitate the operations related to clubs and clubs management, an unusable complicated interface would conflict with our primary goals. There are certain functions that need to be implemented but adding some functionalities would be overkill. For example, when a new president needs to be selected for a club; club director board members could have voted for the new president by using our system but it is unnecessary since they can choose the new president at their club meetings. Such functionalities wouldn't be used as much as others and they would only result in a more complex UI. Thus, we filtered such functionalities and preferred usability ove functionality.

3.2.2 Robustness over Cost

Robustness is a better design choice since we will be handling sensitive data of students. Especially if the university decides to depend on our system for tracking the GE250/251 points, our system will somehow affect some students' grades. We wouldn't risk students to lose points because of a bug in our system.

3.2.3 Portability over Efficiency

Portability is a good design choice since most of our users will be students and we can estimate most of the students are 17-25 years old. Since this age group prefers mobility, we want them to be able to access our system via mobile browsers. We have designed our system with a mobile-first approach and if our system cannot be accessed via most of the browsers, all of these efforts will be meaningless.

3.2.4 Rapid development over Functionality

Since there is a limited time for both designing and implementing our system, we have chosen rapid development over functionality. For example, we considered developing a system in which clubs could directly comment on PDF files (like in Adobe Acrobat) but this functionality would require developing a sub software on its own. Thus, instead we decided to add a forum to each document. So members can indicate their concerns related to a document on that document's forum. In summary, since we have limited time and only 6 developers, we preferred rapid development over functionality by trying to implement as many functionalities as we could.

3.2.5 Maintainability over Performance

As mentioned in section 1.2.2, maintainability is the second design goal we prioritize. Thus, it is not a surprise that we prefer maintainability over performance. Our system design has many layers and a high abstraction among subsystems. This decreases the performance but is a good design choice for optimizing the maintainability.

3.3 Final Object Design

In the final object design class diagram, the interaction between all layers (boundary objects, controllers and entities) are shown. The details of these classes can be found in their corresponding sections.

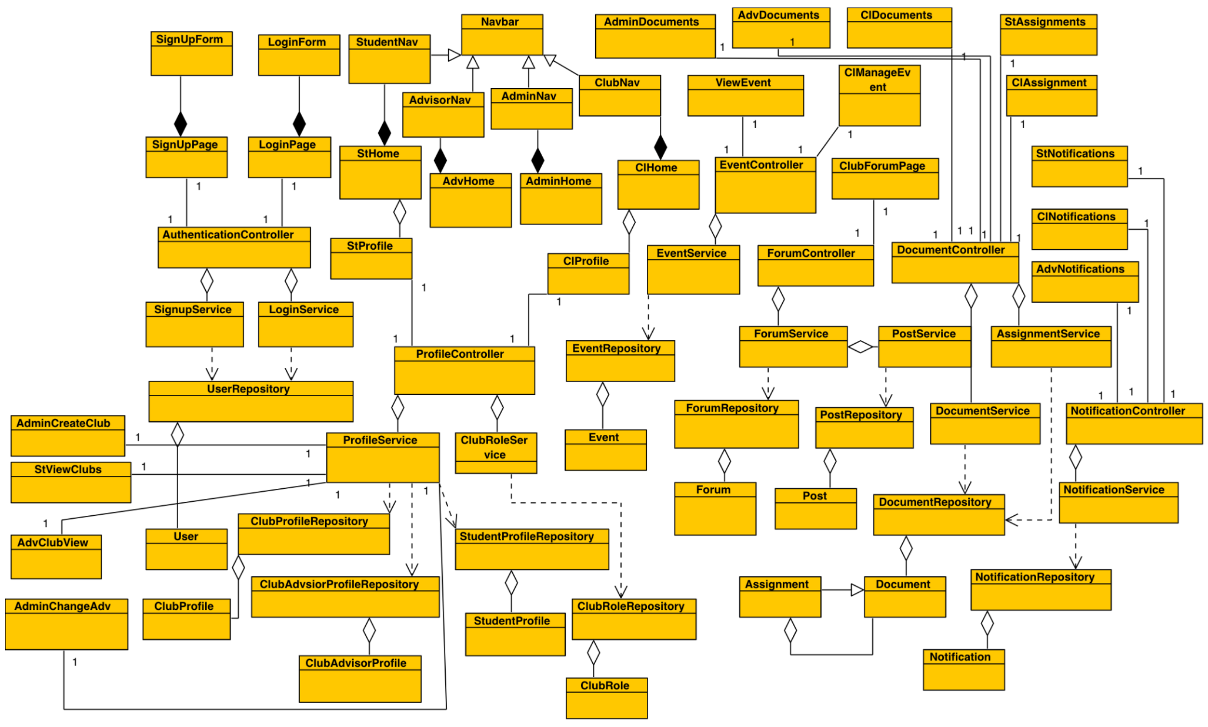


Fig. 3.1. Class diagram showing all the relations between layers in the project

3.4 Layers

3.4.1 Presentation Layer

The presentation layer diagram is also added to the next page in a rotated version.

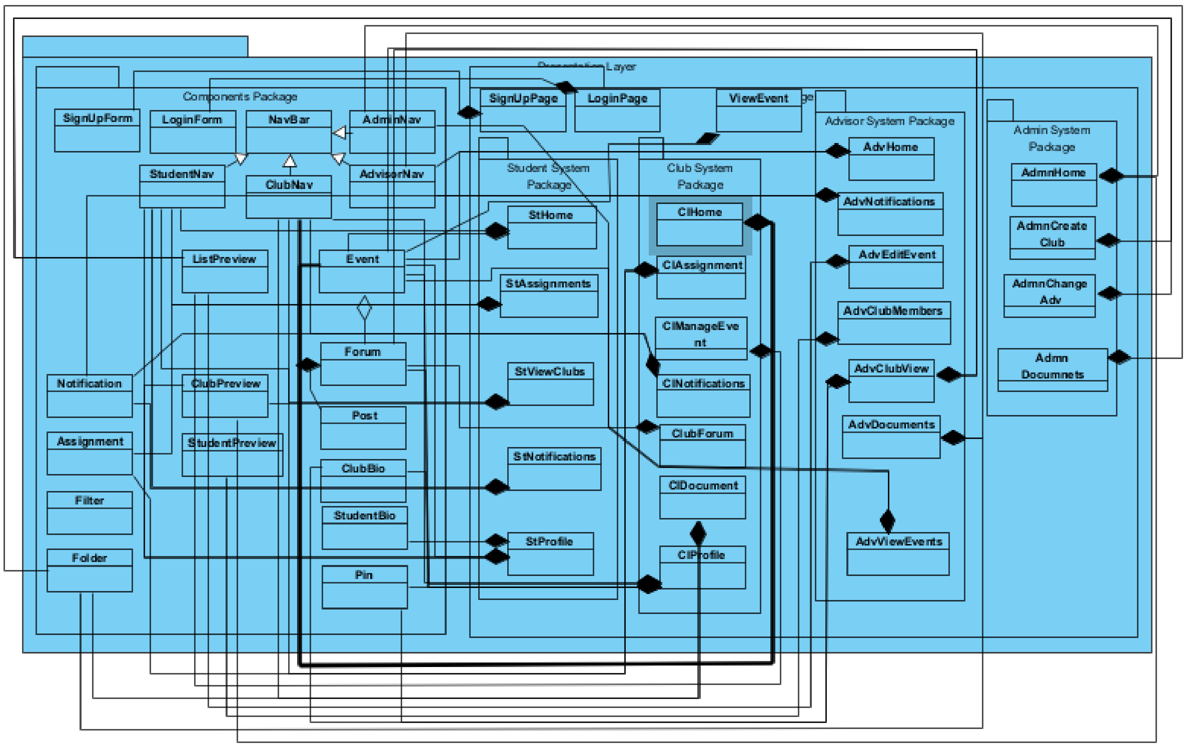


Fig. 3.2. Presentation layer showing the pages and components at the project

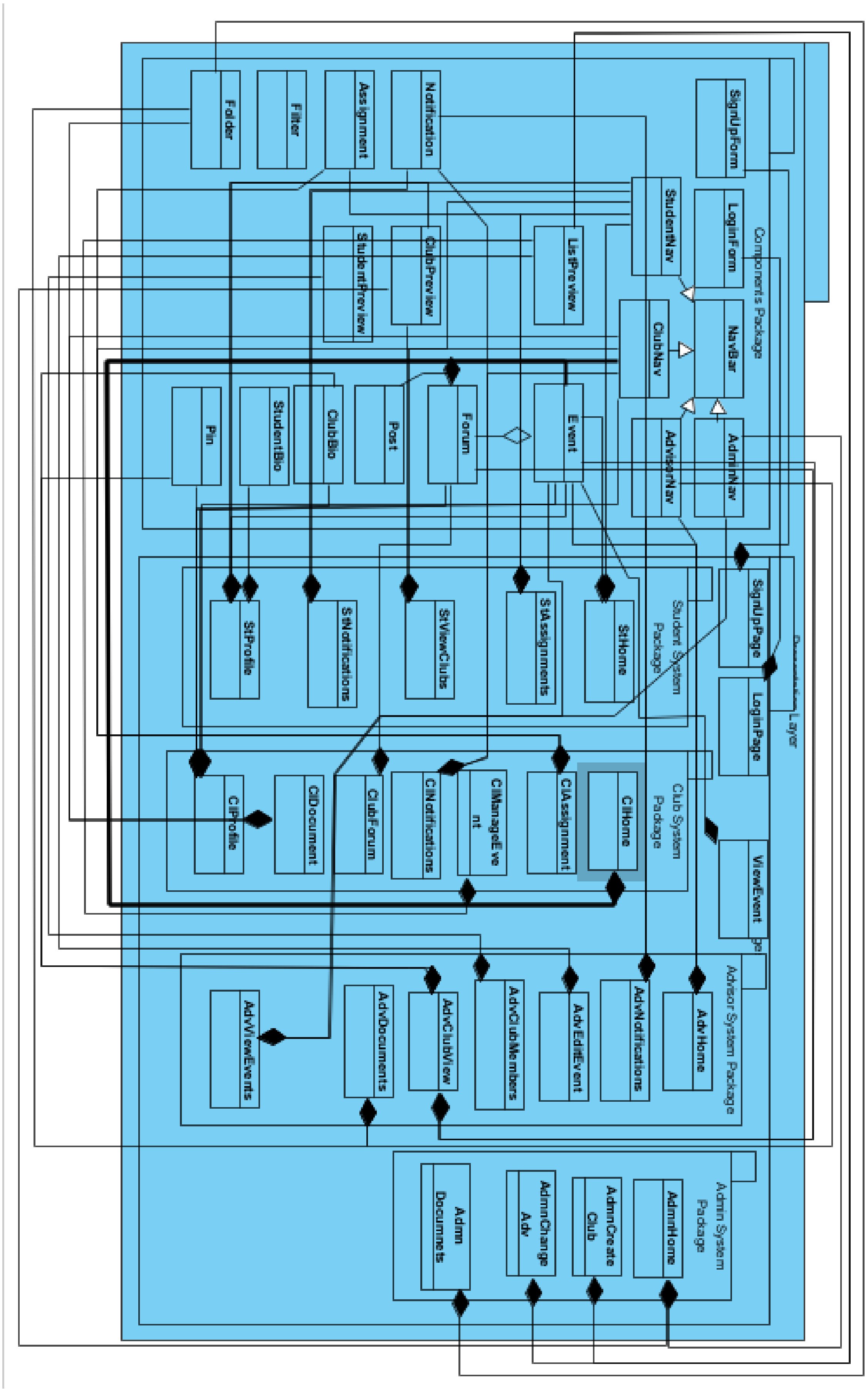


Fig. 3.3. Rotated version of the presentation layer diagram

The presentation layer is the layer that interacts with the end-user. There are two packages in this layer: The components layer and the pages layer. The difference between a component to a page is that a page uses the component, whereas a component cannot exist on its own. For example, the student home page uses the student navigation bar component, and event component. On this page, each event is displayed using the event component multiple times. The idea of splitting the presentation into two packages as components and pages is the above-mentioned reusability option. With this abstraction design, we intend to reuse components.

3.4.1.1 Components Package

The components package contains the following classes.

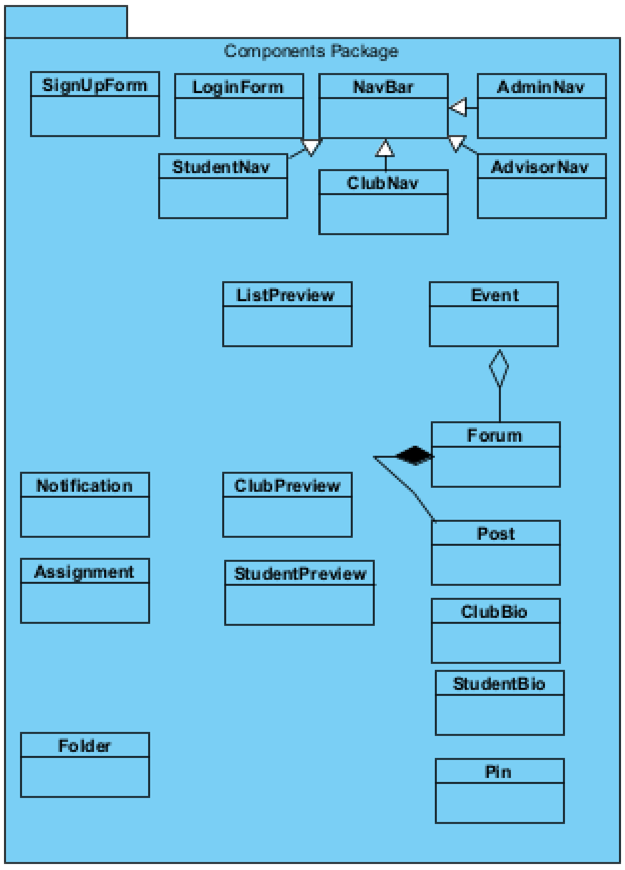


Fig. 3.4. Package diagram showing the components

- SignUpForm: A stylized form that gets sign up information
- LoginForm: A stylized form that gets login information
- Navbar: There are four types
  - AdminNav: Admin System’s navigation bar
  - AdvisorNAv: Advisor System’s navigation bar
  - ClubNav: ClubSystem’s navigation bar
  - StudentNav: Student System’s navigation bar

- ListPreview: A component to display a given list in a stylized way
- Event: A component to display information provided by an event. It has a Forum.
- Forum: A component that displays provided forum information (all of the posts, forum title, etc). It has Post. Forums also exist in club viewing pages on their own without needing an event.
- Post: A component that is to be reused in a forum.
- ClubBio: A component to be displayed in a club's profile. It contains certain information about a club in an organized way.
- ClubPreview: A component that can be thought of as the toString() method for clubs. This component displays clubs' summary info. An example of this component's usage is on the StViewClubs page. On this page, each club is displayed in club preview form so that students can see roughly what that club is. Then they can click on this preview to be redirected to the club's profile which contains the club bio.
- StudentBio: A component to be displayed in a student's profile. It contains certain information about a club in an organized way.
- StudentPreview: A component that can be thought of as the toString() method for students. This component displays a student's summary info. An example of this component's usage is on the AdvClubMembers page. On this page, all students of the club are displayed. An advisor can see roughly who that student is. Then they can click on this preview to be redirected to the student's profile which contains the student bio.
- Pin: A component that displays pinned messages by the club on the club's profile.
- Folder: A graphical representation of a folder structure that is used for the club's documentation.
- Notification: A component to display a notification. It is reused to show many notifications.
- Assignment: A component to display an assignment. It is reused to show many assignments.

3.4.1.2 Pages Package

The components package contains the following classes.

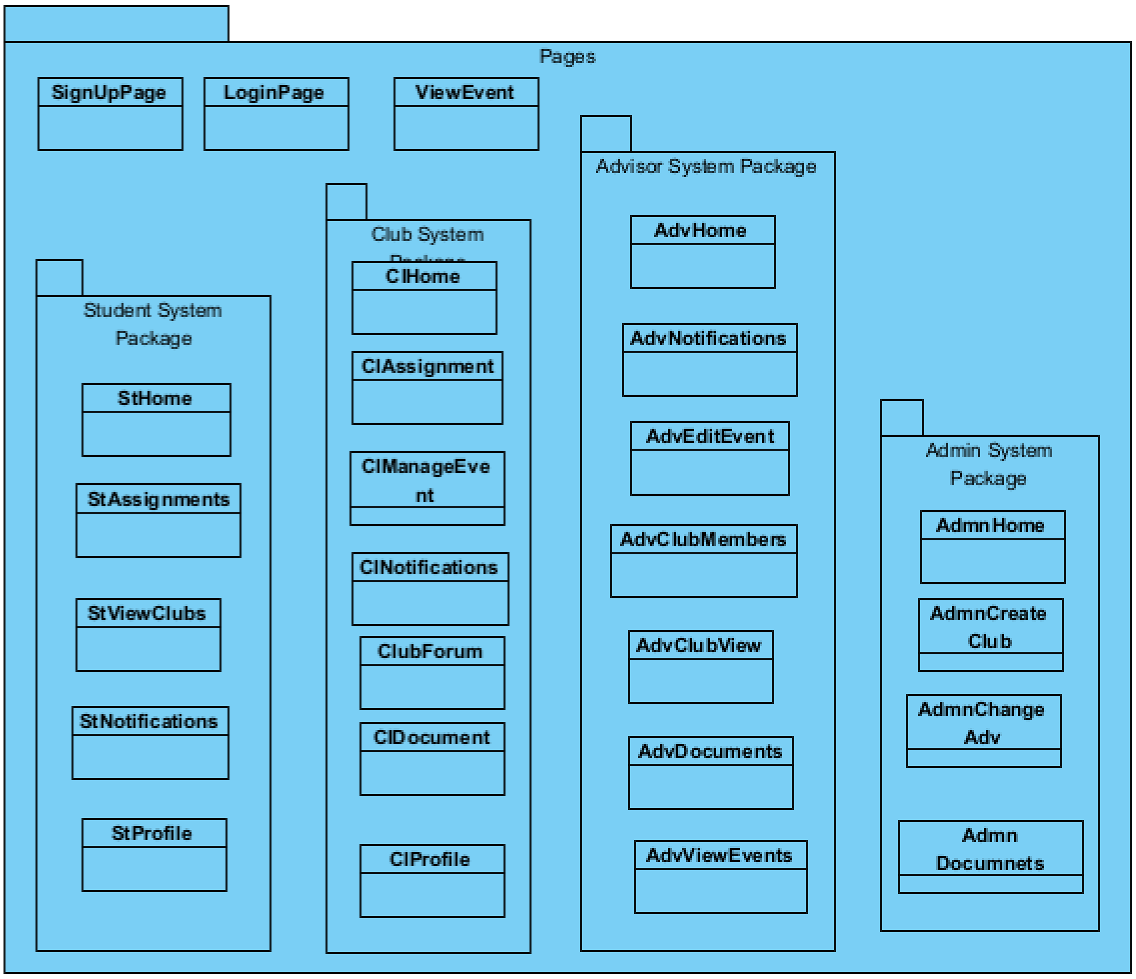


Fig. 3.5. Diagram representing the pages at the project

Pages Package contains 4 additional packages and some shared pages. The reason for dividing pages into four packages is that there will be 4 main systems.

NOTE: Some pages don’t have navigation bars. This is a specific design choice to increase UX. Users are forced on certain pages to complete their actions before being able to return to the main menu. Or they need to cancel their operations first.

- SignUpPage: A Page that gets user info for registering.
  Contains:
  - SignUpForm (1 to 1)
- LoginPage: The welcome page of our system
  Contains:
  - LoginForum (1 to 1)
- ViewEvent: A shared page between the systems. Displays an event.
  Contains:

- Event (1 to 1)

Student System Package:

- StHome: Home page for students
  Contains:
  - StudentNav (1 to 1)
  - Event (1 to 0..*)
- StAssignments: All assignments of a student is displayed on this page
  Contains:
  - StudentNav (1 to 1)
  - Assignment (1 to 0..*)
- StViewClubs: All clubs on the system is displayed on this page
  Contains:
  - StudentNav (1 to 1)
  - ClubPreview (1 to 0..*)
- StNotifications: All notifications of a student is displayed on this page
  Contains:
  - StudentNav (1 to 1)
  - Notification(1 to 0..*)
- StProfile: Profile of a student
  Contains:
  - StudentNav (1 to 1)
  - StudentBio(1 to 1)
  - ClubPreview(1 to 0..*)
  - Event(1 to 0..*)

Club System Package:

- ClHome: Home page for club
  Contains:
  - ClubtNav (1 to 1)
  - Event (1 to 0..*)
- ClAssignments: All assignments of a clubis displayed on this page
  Contains:
  - ClubNav (1 to 1)
  - Assignment (1 to 0..*)
- ClManageEvent: An event is created/edited here
  Contains:
  - ClubtNav (1 to 1)
  - ListPreview (1 to 1)
- ClNotifications: All notifications of a club is displayed on this page
  Contains:
  - ClubNav (1 to 1)
  - Notification(1 to 0..*)
- ClDocument: All documents of a club is displayed on this page

Contains:
    - ClubNav (1 to 1)
    - Folder (1 to 0..*)
- ClProfile: Profile of a club
  Contains:
    - ClubNav (1 to 1)
    - ClubBio(1 to 1)
    - Pin(1 to 0..*)
    - Forum (1 to 1)
    - Event (1 to 0..*)

- ClForum: Forum of a club. Since there isn’t a nav bar on this page, users can return to the club system by pressing “return to club system button”
  Contains:
    - Forum (1 to 1)

Advisor System Package:
- AdvHome: Home page for advsior
  Contains:
    - AdvisorNav (1 to 1)
    - Event (1 to 0..*)
- AdvNotifications: All notifications of an advisor is displayed on this page
  Contains:
    - AdvisorNav (1 to 1)
    - Notification(1 to 0..*)
- AdvEditEvent: An advisor can see the already existing info of an event and edit
  Contains:
    - ListPreview (1 to 0..*)
- AdvClubMembers: Advisors see and manage club members here
  Contains:
    - AdvNav (1 to 1)
    - StudentPreview (1 to 1..0)
- AdvClubView: Advisors see club profiles here
  Contains:
    - AdvNav (1 to 1)
    - Event (1 to 0..*)
    - Forum (1 to 1)
    - Pin (1 to 0..*)
    - ClubBio (1 to 1)
- AdvDocuments: Advisor sees a club’s documents here
  Contains:

    - Folder (1 to 0..*)
    - AdvNav (1 to 1)
- AdvViewEvents: Advisor sees their club’s events here

  Contains:
    - AdvNav (1 to 1)
    - Event (1 to 0..*)

Admin System Package:

- AdmnHome: Home page for admin

  Contains:
    - AdmintNav (1 to 1)
    - ClubPreview (1 to 0..*)
- AdmnCreateClub: A new club is created here

  Contains:
    - ListPreview (1 to 1)
- AdmnChangeAdv: An advisor is changed here

  Contains:
    - ListPreview (1 to 1)
- AdmnDocuments: Admin sees a club’s documents here

  Contains:
    - Folder (1 to 0..*)

3.4.2 Web Server Management Layer

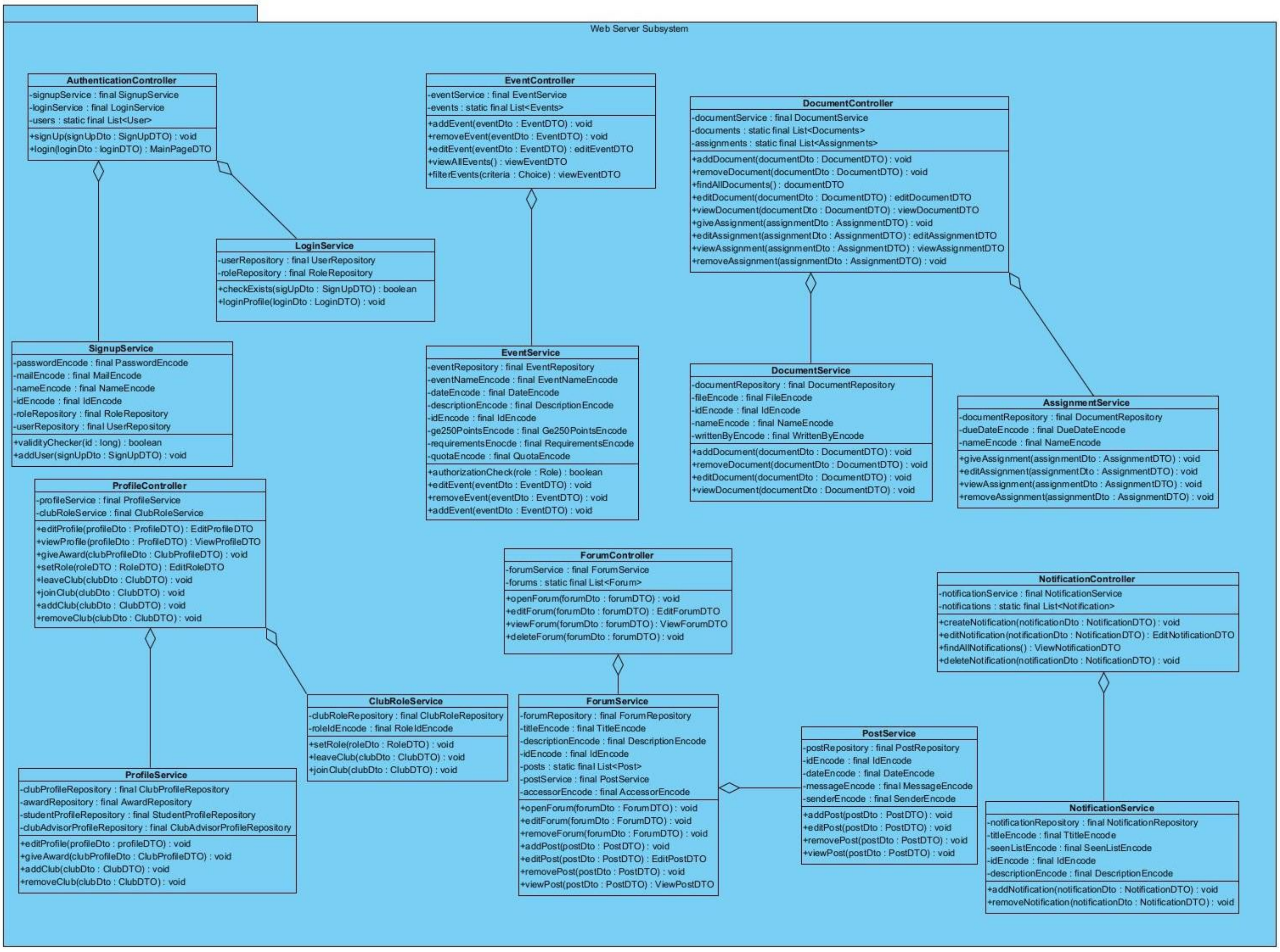

Web Server Subsystem
AuthenticationController
-signupService : final SignupService
-loginService : final LoginService
-users : static final List<User>
+signUp(signUpDto : SignUpDTO) : void
+login(loginDto : loginDTO) : MainPageDTO
LoginService
-userRepository : final UserRepository
-roleRepository : final RoleRepository
+checkExists(sigUpDto : SignUpDTO) : boolean
+loginProfile(loginDto : LoginDTO) : void
SignupService
-passwordEncode : final PasswordEncode
-mailEncode : final MailEncode
-nameEncode : final NameEncode
-idEncode : final IdEncode
-roleRepository : final RoleRepository
-userRepository : final UserRepository
+validityChecker(id : long) : boolean
+addUser(signUpDto : SignUpDTO) : void
EventController
-eventService : final EventService
-events : static final List<Events>
+addEvent(eventDto : EventDTO) : void
+removeEvent(eventDto : EventDTO) : void
+editEvent(eventDto : EventDTO) : editEventDTO
+viewAllEvents() : viewEventDTO
+filterEvents(criteria : Choice) : viewEventDTO
EventService
-eventRepository : final EventRepository
-eventNameEncode : final EventNameEncode
-dateEncode : final DateEncode
-descriptionEncode : final DescriptionEncode
-idEncode : final IdEncode
-ge250PointsEncode : final Ge250PointsEncode
-requirementsEnocde : final RequirementsEncode
-quotaEncode : final QuotaEncode
+authorizationCheck(role : Role) : boolean
+editEvent(eventDto : EventDTO) : void
+removeEvent(eventDto : EventDTO) : void
+addEvent(eventDto : EventDTO) : void
DocumentController
-documentService : final DocumentService
-documents : static final List<Documents>
-assignments : static final List<Assignments>
+addDocument(documentDto : DocumentDTO) : void
+removeDocument(documentDto : DocumentDTO) : void
+findAllDocuments() : documentDTO
+editDocument(documentDto : DocumentDTO) : editDocumentDTO
+viewDocument(documentDto : DocumentDTO) : viewDocumentDTO
+giveAssignment(assignmentDto : AssignmentDTO) : void
+editAssignment(assignmentDto : AssignmentDTO) : editAssignmentDTO
+viewAssignment(assignmentDto : AssignmentDTO) : viewAssignmentDTO
+removeAssignment(assignmentDto : AssignmentDTO) : void
DocumentService
-documentRepository : final DocumentRepository
-fileEncode : final FileEncode
-idEncode : final IdEncode
-nameEncode : final NameEncode
-writtenByEncode : final WrittenByEncode
+addDocument(documentDto : DocumentDTO) : void
+removeDocument(documentDto : DocumentDTO) : void
+editDocument(documentDto : DocumentDTO) : void
+viewDocument(documentDto : DocumentDTO) : void
AssignmentService
-documentRepository : final DocumentRepository
-dueDateEncode : final DueDateEncode
-nameEncode : final NameEncode
+giveAssignment(assignmentDto : AssignmentDTO) : void
+editAssignment(assignmentDto : AssignmentDTO) : void
+viewAssignment(assignmentDto : AssignmentDTO) : void
+removeAssignment(assignmentDto : AssignmentDTO) : void
ProfileController
-profileService : final ProfileService
-clubRoleService : final ClubRoleService
+editProfile(profileDto : ProfileDTO) : EditProfileDTO
+viewProfile(profileDto : ProfileDTO) : ViewProfileDTO
+giveAward(clubProfileDto : ClubProfileDTO) : void
+setRole(roleDTO : RoleDTO) : EditRoleDTO
+leaveClub(clubDto : ClubDTO) : void
+joinClub(clubDto : ClubDTO) : void
+addClub(clubDto : ClubDTO) : void
+removeClub(clubDto : ClubDTO) : void
ClubRoleService
-clubRoleRepository : final ClubRoleRepository
-roleIdEncode : final RoleIdEncode
+setRole(roleDto : RoleDTO) : void
+leaveClub(clubDto : ClubDTO) : void
+joinClub(clubDto : ClubDTO) : void
ProfileService
-clubProfileRepository : final ClubProfileRepository
-awardRepository : final AwardRepository
-studentProfileRepository : final StudentProfileRepository
-clubAdvisorProfileRepository : final ClubAdvisorProfileRepository
+editProfile(profileDto : profileDTO) : void
+giveAward(clubProfileDto : ClubProfileDTO) : void
+addClub(clubDto : ClubDTO) : void
+removeClub(clubDto : ClubDTO) : void
ForumController
-forumService : final ForumService
-forums : static final List<Forum>
+openForum(forumDto : forumDTO) : void
+editForum(forumDto : forumDTO) : EditForumDTO
+viewForum(forumDto : forumDTO) : ViewForumDTO
+deleteForum(forumDto : forumDTO) : void
ForumService
-forumRepository : final ForumRepository
-titleEncode : final TitleEncode
-descriptionEncode : final DescriptionEncode
-idEncode : final IdEncode
-posts : static final List<Post>
-postService : final PostService
-accessorEncode : final AccessorEncode
+openForum(forumDto : ForumDTO) : void
+editForum(forumDto : ForumDTO) : void
+removeForum(forumDto : ForumDTO) : void
+addPost(postDto : PostDTO) : void
+editPost(postDto : PostDTO) : EditPostDTO
+removePost(postDto : PostDTO) : void
+viewPost(postDto : PostDTO) : ViewPostDTO
PostService
-postRepository : final PostRepository
-idEncode : final IdEncode
-dateEncode : final DateEncode
-messageEncode : final MessageEncode
-senderEncode : final SenderEncode
+addPost(postDto : PostDTO) : void
+editPost(postDto : PostDTO) : void
+removePost(postDto : PostDTO) : void
+viewPost(postDto : PostDTO) : void
NotificationController
-notificationService : final NotificationService
-notifications : static final List<Notification>
+createNotification(notificationDto : NotificationDTO) : void
+editNotification(notificationDto : NotificationDTO) : EditNotificationDTO
+findAllNotifications() : ViewNotificationDTO
+deleteNotification(notificationDto : NotificationDTO) : void
NotificationService
-notificationRepository : final NotificationRepository
-titleEncode : final TtitleEncode
-seenListEncode : final SeenListEncode
-idEncode : final IdEncode
-descriptionEncode : final DescriptionEncode
+addNotification(notificationDto : NotificationDTO) : void
+removeNotification(notificationDto : NotificationDTO) : void

Fig.3.6. Diagram showing the web server layer of the project

Web Server Subsystem consists of Controller and Service classes. Controller classes somehow refer to the actions of actors, whereas the Service classes indicate their interaction with databases. The main functions of the program are divided into subsystems and these systems are expressed in terms of Controller and Service classes above. These Controller-Service duos are listed and explained below.

a) Controllers:

- AuthenticationController

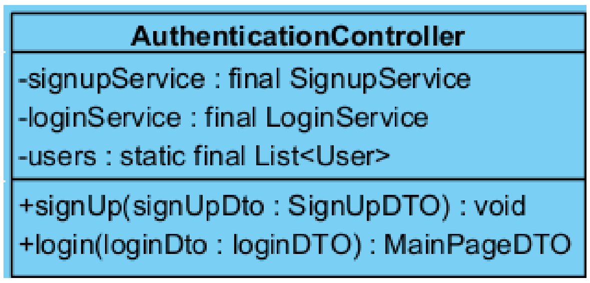

AuthenticationController
-signupService : final SignupService
-loginService : final LoginService
-users : static final List<User>
+signUp(signUpDto : SignUpDTO) : void
+login(loginDto : loginDTO) : MainPageDTO

Fig. 3.7. AuthenticationController

It is the controller class for the functionalities associated with entering the system such as sign up and login. It contains SignUpService and LoginService to maintain processes associated with the database.

Attributes:

- ☐ **signupService:** Service class on the web server layer for the sign up operations
- ☐ **loginService:** Service class on the web server layer for the login operations
- ☐ **users:** The list of User objects, which stores the users of the system

Methods:

- ☐ **signUp(signUpDto: SignUpDTO);** The signup method for signing different types of users to the system
- ☐ **login(loginDto: loginDTO);** The login method for all the users entering to the system after being signed up

- EventController

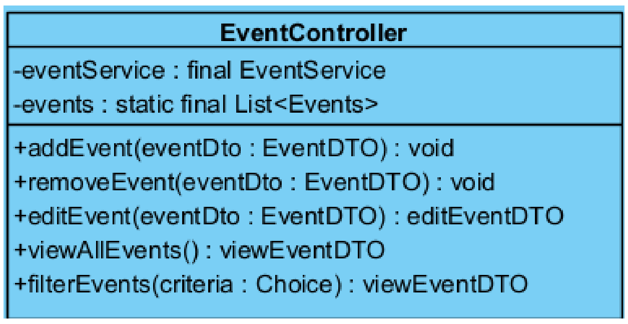

EventController
-eventService : final EventService
-events : static final List<Events>
+addEvent(eventDto : EventDTO) : void
+removeEvent(eventDto : EventDTO) : void
+editEvent(eventDto : EventDTO) : editEventDTO
+viewAllEvents() : viewEventDTO
+filterEvents(criteria : Choice) : viewEventDTO

Fig. 3.8. EventController

It is the controller class for the functionalities associated with events such as adding, deleting, viewing, editing and filtering. It contains an EventService object to maintain processes of the events associated with the database.

Attributes:

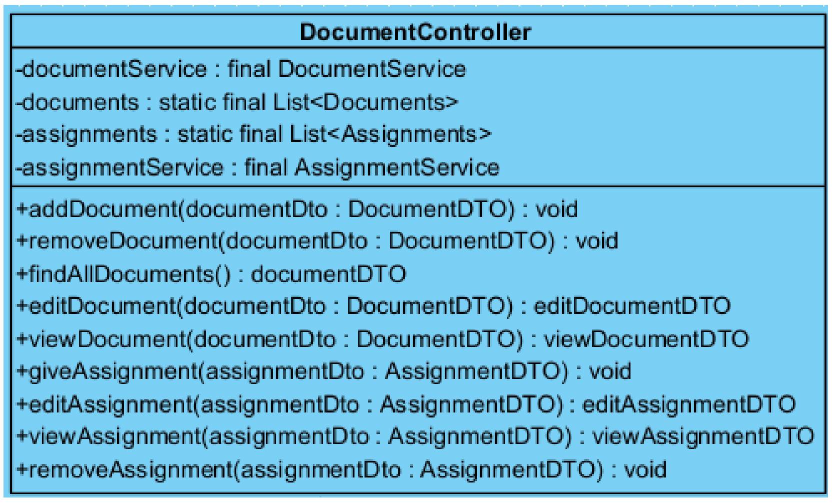
- ☐ **eventService:** Service class on the web server layer for the event related operations

- ☐ **events:** The list of Events objects, which stores all the past and active events in the system

Methods:

- ☐ **addEvent(eventDto: EventDTO);** The method to add new events to the system
- ☐ **removeEvent(eventDto: EventDTO);** The method to remove a specific event from the system
- ☐ **editEvent(eventDto: EventDTO);** The method to edit a specific event in the system like changing its date, conditions, description, etc.
- ☐ **viewAllEvents();** The method to view all the active events in the system
- ☐ **filterEvents(criteria: Choice);** The method to filter the active events in the system according to some user choices like specifying the club name

- DocumentController



DocumentController
-documentService : final DocumentService -documents : static final List<Documents> -assignments : static final List<Assignments> -assignmentService : final AssignmentService
+addDocument(documentDto : DocumentDTO) : void +removeDocument(documentDto : DocumentDTO) : void +findAllDocuments() : documentDTO +editDocument(documentDto : DocumentDTO) : editDocumentDTO +viewDocument(documentDto : DocumentDTO) : viewDocumentDTO +giveAssignment(assignmentDto : AssignmentDTO) : void +editAssignment(assignmentDto : AssignmentDTO) : editAssignmentDTO +viewAssignment(assignmentDto : AssignmentDTO) : viewAssignmentDTO +removeAssignment(assignmentDto : AssignmentDTO) : void

Fig. 3.9. DocumentController

It is the controller class for the functionalities associated with documents such as adding, deleting and viewing. It contains a DocumentService object to maintain processes of the documents associated with the database.

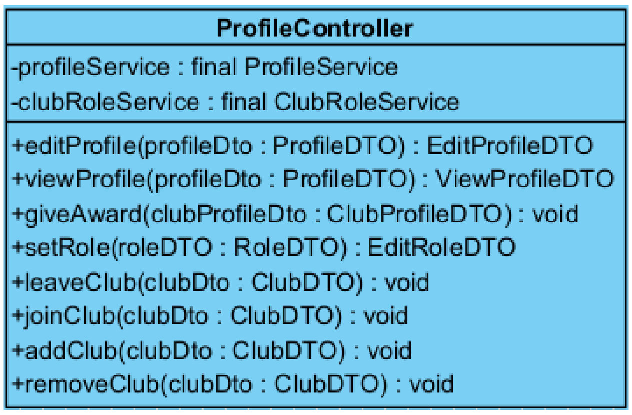
Attributes:

- ☐ **documentService:** Service class on the web server layer for the document related operations
- ☐ **documents:** The list of Document objects, which stores all the documents that have been uploaded to system
- ☐ **assignments:** The list of Assignment objects, which stores all the assignments that have been given in the system
- ☐ **assignmentService:** Service class on the web server layer for the assignment related operations

Methods:

- ☐ **addDocument(documentDto: DocumentDTO);** The method to add new documents to the system
- ☐ **removeDocument(documentDto: DocumentDTO);** The method to remove a specific document from the system
- ☐ **findAllDocuments();** The method to find all documents in the system
- ☐ **editDocument(documentDto: DocumentDTO);** The method to edit a specific document in the system
- ☐ **viewDocument(documentDto: DocumentDTO);** The method to view a specific document in the system
- ☐ **giveAssignment(assignmentDto: AssignmentDTO);** The method to add a new assignment to the system, which means a club's authorized person gives assignment to its members
- ☐ **editAssignment(assignmentDto: AssignmentDTO);** The method to edit an assignment in the system
- ☐ **viewAssignment(assignmentDto: AssignmentDTO);** The method to view an assignment in the system
- ☐ **removeAssignment(assignmentDto: AssignmentDTO);** The method to remove an assignment from the system

● ProfileController



```
ProfileController
-profileService : final ProfileService
-clubRoleService : final ClubRoleService
+editProfile(profileDto : ProfileDTO) : EditProfileDTO
+viewProfile(profileDto : ProfileDTO) : ViewProfileDTO
+giveAward(clubProfileDto : ClubProfileDTO) : void
+setRole(roleDTO : RoleDTO) : EditRoleDTO
+leaveClub(clubDto : ClubDTO) : void
+joinClub(clubDto : ClubDTO) : void
+addClub(clubDto : ClubDTO) : void
+removeClub(clubDto : ClubDTO) : void
```

Fig. 3.10. ProfileController

It is the controller class for the functionalities associated with profiles such as editing and viewing. It also controls the award mechanisms for the club profiles. It contains a ProfileService object to maintain processes of the profiles associated with the database.

Attributes:

- ☐ **profileService:** Service class on the web server layer for the profile related operations
- ☐ **clubRoleService:** Service class on the web server layer for the club role related operations, which refers to the roles of users in specific clubs

Methods:

- ☐ **editProfile(profileDto: ProfileDTO);** The method to edit the profile of a user in the system
- ☐ **viewProfile(profileDto: ProfileDTO);** The method to view the profile of a user in the system
- ☐ **giveAward(clubProfileDto: ClubProfileDTO);** The method to give awards to specific clubs, which is used when admin decides to award a specific club due to its success
- ☐ **setRole(roleDto: RoleDTO);** The method to set the role of a user in the system for a specific club
- ☐ **leaveClub(clubDto: ClubDTO);** The method for a student to leave a club, which would remove all of his/her roles in this club
- ☐ **joinClub(clubDto: ClubDTO);** The method for a student to join a club, which would make him/her a member of this club and accordingly change his/her role there
- ☐ **addClub(clubDto: ClubDTO);** The method to add a club into system, which means the new created clubs are added to the system by admin
- ☐ **removeClub(clubDto: ClubDTO);** The method to remove a club from the system, which is processed by the admin

- ForumController

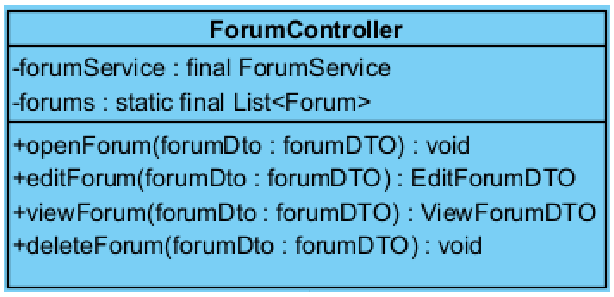

ForumController
-forumService : final ForumService
-forums : static final List<Forum>
+openForum(forumDto : forumDTO) : void
+editForum(forumDto : forumDTO) : EditForumDTO
+viewForum(forumDto : forumDTO) : ViewForumDTO
+deleteForum(forumDto : forumDTO) : void

Fig. 3.11. ForumController

It is the controller class for the functionalities associated with forums in the system such as opening, editing, viewing or deleting. It contains an instance of the ForumService to maintain processes of forums associated with the database.

Attributes:

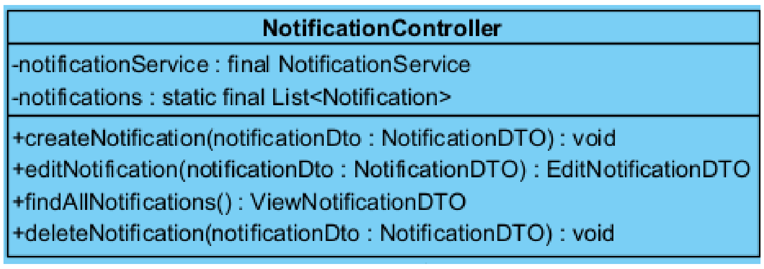
- ☐ **forumService:** Service class on the web server layer for the forum related operations
- ☐ **forums:** List of all the forums created by the clubs in the system

Methods:

- ☐ **openForum(forumDto: ForumDTO);** The method to open a forum in the system, which means creating forum by club's authorized person for specific assignment, event, or the club itself
- ☐ **editForum(forumDto: ForumDTO);** The method to edit a forum in the system

- ☐ **viewForum(forumDto: ForumDTO);** The method to view a forum in the system
- ☐ **deleteForum(forumDto: ForumDTO);** The method to delete a forum from the system

- NotificationController



```
NotificationController
-notificationService : final NotificationService
-notifications : static final List<Notification>
+createNotification(notificationDto : NotificationDTO) : void
+editNotification(notificationDto : NotificationDTO) : EditNotificationDTO
+findAllNotifications() : ViewNotificationDTO
+deleteNotification(notificationDto : NotificationDTO) : void
```

Fig. 3.12. NotificationController

It is the controller class for the functionalities associated with notifications in the system such as adding, editing, viewing or deleting. It contains an instance of the NotificationService to maintain processes of notifications associated with the database.

Attributes:

- ☐ **notificationService:** Service class on the web server layer for the notification related operations
- ☐ **notifications:** List of all the notifications created by clubs, advisors or admins in the system

Methods:

- ☐ **createNotification(notificationDto: NotificationDTO);** The method to create a specific notification by clubs, admins, or advisors for specific events, assignments, or system related issues
- ☐ **editNotification(notificationDto: NotificationDTO);** The method to edit a specific notification in the system
- ☐ **findAllNotifications();** The method to view all the notifications in the system
- ☐ **deleteNotification(notificationDto: NotificationDTO);** The method to delete a specific notification from the system

b) **Service:**

- SignupService

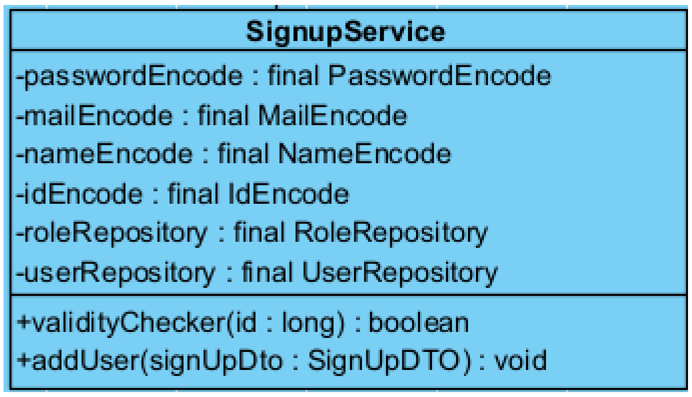

SignupService
-passwordEncode : final PasswordEncode
-mailEncode : final MailEncode
-nameEncode : final NameEncode
-idEncode : final IdEncode
-roleRepository : final RoleRepository
-userRepository : final UserRepository
+validityChecker(id : long) : boolean
+addUser(signUpDto : SignUpDTO) : void

Fig. 3.13. SignupService

When a user attempts to sign up to this system, this service class encodes the attributes of the enrolling person such as Password, type, name, ID, Role into the UserRepository. Additionally, it checks if any user with the same ID is enrolled in the system beforehand.

Attributes:

- ☐ **passwordEncode:** The encoder to encode the passwords of the users to the database while signing up
- ☐ **mailEncode:** The encoder to encode the emails of the users to the database while signing up
- ☐ **nameEncode:** The encoder to encode the names of the users to the database while signing up
- ☐ **idEncode:** The encoder to encode the id numbers of the users to the database while signing up
- ☐ **roleRepository:** The repository that stores the roles of all users in the system.
- ☐ **userRepository:** The repository that involves all the information about all system users in the encoded version

Methods:

- ☐ **addUser(signUpDto: SignUpDTO);** The method to add users to the UserRepository.
- ☐ **validityChecker(id: long);** The method to check if the entered id is valid and signed up to the system

- LoginService

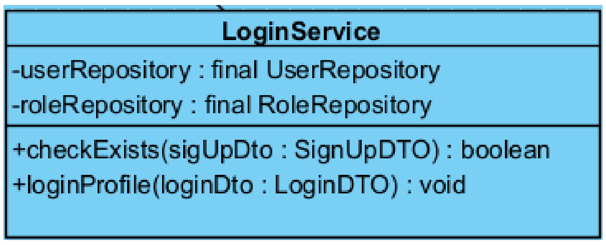

LoginService
-userRepository : final UserRepository
-roleRepository : final RoleRepository
+checkExists(sigUpDto : SignUpDTO) : boolean
+loginProfile(loginDto : LoginDTO) : void

Fig. 3.14. LoginService

When a user attempts to login to this system, this service class checks if the user with entered information exists in the database, and accordingly prints out a message or opens the system.

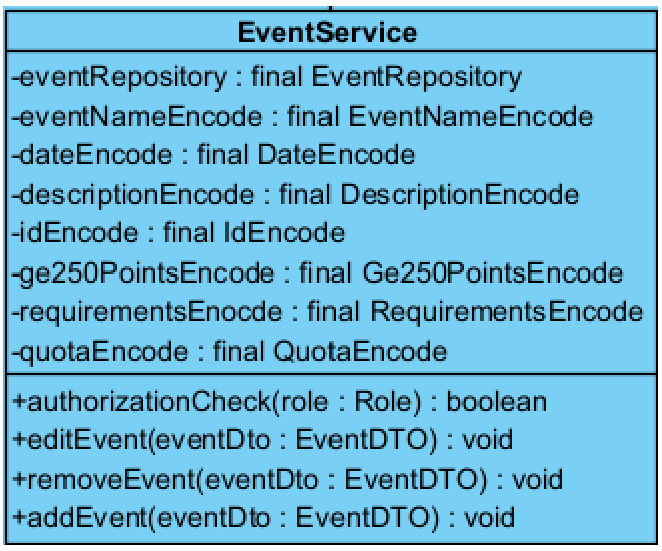
Attributes:

- ☐ **roleRepository:** The repository that stores the roles of all users in the system.
- ☐ **userRepository:** The repository that involves all the information about all system users in the encoded version

Methods:

- ☐ **loginProfile(loginDto: LoginDTO);** The method of login for a specific user to login to the home page
- ☐ **checkExists(signUpDTO: SignUpDTO);** The method to check if the entered information while logging in is valid or not

- EventService



```
EventService
-eventRepository : final EventRepository
-eventNameEncode : final EventNameEncode
-dateEncode : final DateEncode
-descriptionEncode : final DescriptionEncode
-idEncode : final IdEncode
-ge250PointsEncode : final Ge250PointsEncode
-requirementsEnocde : final RequirementsEncode
-quotaEncode : final QuotaEncode
+authorizationCheck(role : Role) : boolean
+editEvent(eventDto : EventDTO) : void
+removeEvent(eventDto : EventDTO) : void
+addEvent(eventDto : EventDTO) : void
```

Fig. 3.15. EventService

When a user tries to create an event, it encodes the information of events such as Event Name, Date and Properties into the EventRepository. Additionally, it also has an attribute of PrivilegeRepisotory in order to check the accessibility of event organization features.

Attributes:

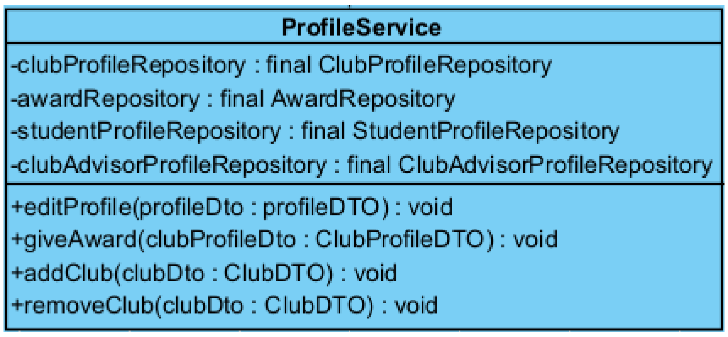
- ☐ **eventRepository:** The repository that stores all the events in the system with their related information in the encoded form.
- ☐ **eventNameEncode:** The encoder to encode the names of the events on the system during creation or after editing
- ☐ **dateEncode:** The encoder to encode the dates of the events on the system during creation or after editing

- ☐ **descriptionEncode:** The encoder to encode the descriptions of the events on the system during creation or after editing
- ☐ **idEncode:** The encoder to encode the id numbers of the events on the system during creation or after editing
- ☐ **ge250PointsEncode:** The encoder to encode the ge250 points of the events on the system during creation or after editing
- ☐ **requirementsEncode:** The encoder to encode the requirements of the events on the system during creation or after editing
- ☐ **quotaEncode:** The encoder to encode the quotas of the events on the system during creation or after editing

Methods:

- ☐ **authorizationCheck(role: Role);** The method to check which kind of roles are authorized to do a specific kind of operations on the Event object
- ☐ **addEvent(eventDto: EventDTO);** The method to add new events to the system
- ☐ **removeEvent(eventDto: EventDTO);** The method to remove a specific event from the system
- ☐ **editEvent(eventDto: EventDTO);** The method to edit a specific event in the system like changing its date, conditions, description, etc.

- ProfileService



ProfileService

-clubProfileRepository : final ClubProfileRepository
-awardRepository : final AwardRepository
-studentProfileRepository : final StudentProfileRepository
-clubAdvisorProfileRepository : final ClubAdvisorProfileRepository

+editProfile(profileDto : profileDTO) : void
+giveAward(clubProfileDto : ClubProfileDTO) : void
+addClub(clubDto : ClubDTO) : void
+removeClub(clubDto : ClubDTO) : void

Fig. 3.16. ProfileService

This class contains the repositories for StudentProfile, ClubAdvisorProfile and ClubProfile. Thus, the profiles of all the users would be stored in these databases, and the editing on these profiles would directly influence the encoding of the profile in the database.

Attributes:

- ☐ **clubProfileRepository:** The repository that stores all the club profiles in the system. It refers to all the clubs with their encoded information such as name, events, budget, documents, etc.
- ☐ **studentProfileRepository:** The repository that stores all the student profiles in the system. It refers to all the students with their encoded information such as name, id, department, ge250 points, etc.

- ☐ **clubAdvisorProfileRepository:** The repository that stores all the club advisor profiles in the system. It refers to all the club advisors with their encoded information such as department and office hours
- ☐ **awardRepository:** The repository that stores all the awards in the system. It refers to all the awards with their encoded information such as date given and name.

Methods:

- ☐ **editProfile(profileDto: ProfileDTO);** The method to edit the profile of a user in the system
- ☐ **giveAward(clubProfileDto: ClubProfileDTO);** The method to give awards to specific clubs, which is used when admin decides to award a specific club due to its success
- ☐ **addClub(clubDto: ClubDTO);** The method to add a club into system, which means the new created clubs are added to the system by admin
- ☐ **removeClub(clubDto: ClubDTO);** The method to remove a club from the system, which is processed by the admin

- ClubRoleService

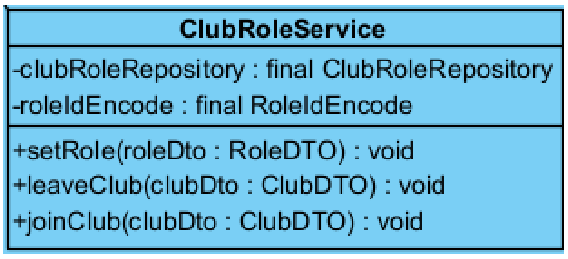

ClubRoleService
-clubRoleRepository : final ClubRoleRepository
-roleIdEncode : final RoleIdEncode
+setRole(roleDto : RoleDTO) : void
+leaveClub(clubDto : ClubDTO) : void
+joinClub(clubDto : ClubDTO) : void

Fig. 3.17. ClubRoleService

When a user's role in a club is determined or changed, it encodes the role of the student in each club into the ClubRoleRepository.

Attributes:

- ☐ **clubRoleRepository:** The repository that stores all the club roles in the system. It refers to all the associations between id numbers of users and their roles in specific clubs in the encoded form.
- ☐ **roleIdEncode:** The encoder for encoding the id number of a specific role

Methods:

- ☐ **setRole(roleDto: RoleDTO);** The method to set the role of a user in the system for a specific club
- ☐ **leaveClub(clubDto: ClubDTO);** The method for a student to leave a club, which would remove all of his/her roles in this club
- ☐ **joinClub(clubDto: ClubDTO);** The method for a student to join a club, which would make him/her a member of this club and accordingly change his/her role there

- ForumService

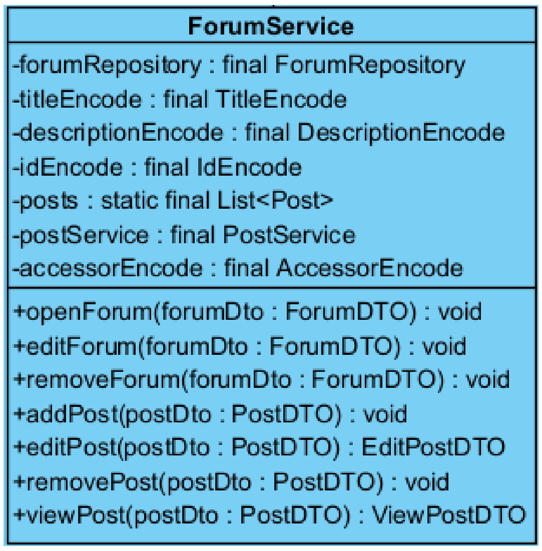

ForumService
-forumRepository : final ForumRepository
-titleEncode : final TitleEncode
-descriptionEncode : final DescriptionEncode
-idEncode : final IdEncode
-posts : static final List<Post>
-postService : final PostService
-accessorEncode : final AccessorEncode
+openForum(forumDto : ForumDTO) : void
+editForum(forumDto : ForumDTO) : void
+removeForum(forumDto : ForumDTO) : void
+addPost(postDto : PostDTO) : void
+editPost(postDto : PostDTO) : EditPostDTO
+removePost(postDto : PostDTO) : void
+viewPost(postDto : PostDTO) : ViewPostDTO

Fig. 3.18 ForumService

When a user tries to open a forum, it encodes the information of forums such as Title, Id and Description into the ForumRepository.

Attributes:

- ☐ **forumRepository:** The repository that stores all the forums in the system. It refers to all the forums with their informations such as id, name, accessing participants, etc. in their encoded form
- ☐ **titleEncode:** The encoder for encoding the title of a specific forum after creation
- ☐ **descriptionEncode:** The encoder for encoding the description of a specific forum after creation
- ☐ **idEncode:** The encoder for encoding the id of a specific forum after creation
- ☐ **accessorEncode:** The encoder for encoding the enabled accessors of a specific forum after creation
- ☐ **posts:** The list that contains all the posts inside the forums of the system
- ☐ **postService:** Service class on the web server layer for the forum post related operations

Methods:

- ☐ **openForum(forumDto: ForumDTO);** The method to open a forum in the system, which means creating forum by club's authorized person for specific assignment, event, or the club itself
- ☐ **editForum(forumDto: ForumDTO);** The method to edit a forum in the system
- ☐ **removeForum(forumDto: ForumDTO);** The method to delete a forum from the system
- ☐ **addPost(postDto: PostDTO);** The method to add post into a specific forum in the system
- ☐ **editPost(postDto: PostDTO);** The method to edit post in a specific forum.

- ☐ **removePost(postDto: PostDTO);** The method to remove post from a specific forum.
- ☐ **viewPost(postDto: PostDTO);** The method to view a specific post inside the forum.

- PostService

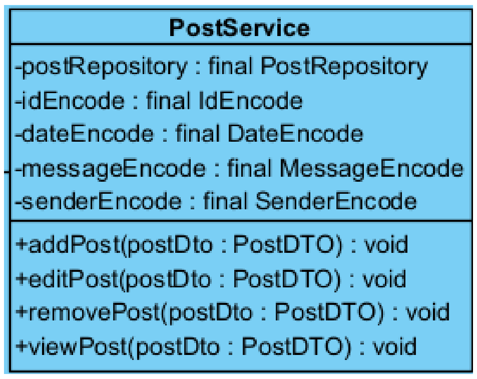

PostService
-postRepository : final PostRepository
-idEncode : final IdEncode
-dateEncode : final DateEncode
-messageEncode : final MessageEncode
-senderEncode : final SenderEncode
+addPost(postDto : PostDTO) : void
+editPost(postDto : PostDTO) : void
+removePost(postDto : PostDTO) : void
+viewPost(postDto : PostDTO) : void

Fig.3.19. PostService

When a user tries to create a post inside a forum, it encodes the information of posts such as Date, Message, Id and Sender into the PostRepository.

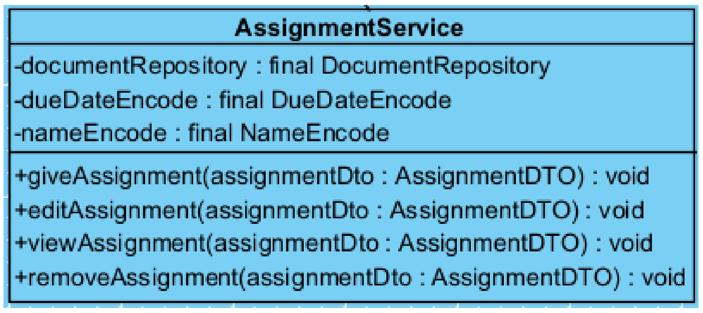
Attributes:

- ☐ **postRepository:** The repository that stores all the posts in the system. It refers to all the posts with their informations such as date, name, id, and message in their encoded form
- ☐ **dateEncode:** The encoder for encoding the date of a specific post after creation or being edited
- ☐ **messageEncode:** The encoder for encoding the message in a specific post after creation or being edited
- ☐ **idEncode:** The encoder for encoding the id of a specific post after creation
- ☐ **senderEncode:** The encoder for encoding the sender of a specific post after creation

Methods:

- ☐ **addPost(postDto: PostDTO);** The method to add post into a specific forum in the system
- ☐ **editPost(postDto: PostDTO);** The method to edit post in a specific forum.
- ☐ **removePost(postDto: PostDTO);** The method to remove post from a specific forum.
- ☐ **viewPost(postDto: PostDTO);** The method to view a specific post inside the forum.

- AssignmentService


AssignmentService
-documentRepository : final DocumentRepository
-dueDateEncode : final DueDateEncode
-nameEncode : final NameEncode
+giveAssignment(assignmentDto : AssignmentDTO) : void
+editAssignment(assignmentDto : AssignmentDTO) : void
+viewAssignment(assignmentDto : AssignmentDTO) : void
+removeAssignment(assignmentDto : AssignmentDTO) : void
Fig. 3.20. AssignmentService

When a club, admin, or adviser tries to create an assignment, it encodes the information of assignments such as Name and Due Date into the DocumentRepository and adds them to the assignment list.

Attributes:

- ☐ **documentRepository:** The repository that stores all the documents in the system. It refers to all the documents with their informations such as id, name, writer, and file in their encoded form
- ☐ **dueDateEncode:** The encoder for encoding the due date of a specific assignment after creation or being edited
- ☐ **nameEncode:** The encoder for encoding the name of a specific assignment after creation or being edited

Methods:

- ☐ **giveAssignment(assignmentDto: AssignmentDTO);** The method to add a new assignment to the system, which means a club's authorized person gives assignment to its members
- ☐ **editAssignment(assignmentDto: AssignmentDTO);** The method to edit an assignment in the system
- ☐ **viewAssignment(assignmentDto: AssignmentDTO);** The method to view an assignment in the system
- ☐ **removeAssignment(assignmentDto: AssignmentDTO);** The method to remove an assignment from the system

- DocumentService

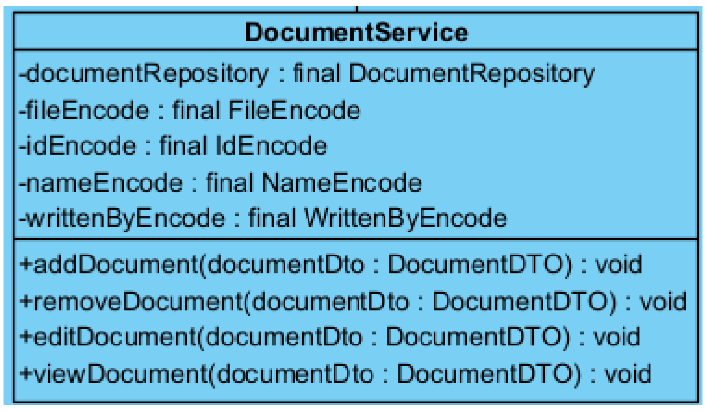

DocumentService
-documentRepository : final DocumentRepository
-fileEncode : final FileEncode
-idEncode : final IdEncode
-nameEncode : final NameEncode
-writtenByEncode : final WrittenByEncode
+addDocument(documentDto : DocumentDTO) : void
+removeDocument(documentDto : DocumentDTO) : void
+editDocument(documentDto : DocumentDTO) : void
+viewDocument(documentDto : DocumentDTO) : void

Fig. 3.21. DocumentService

When a user tries to create a document, it encodes the document file into the DocumentRepository.

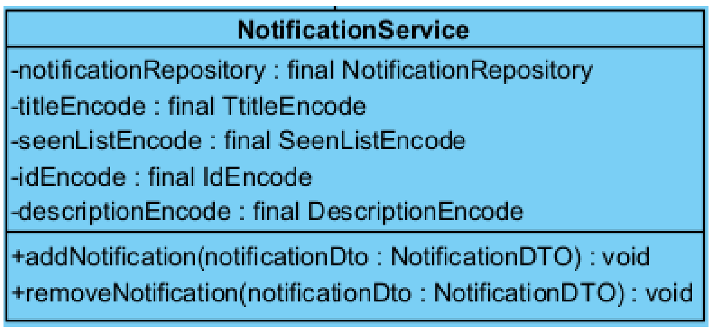
Attributes:

- ☐ **documentRepository:** The repository that stores all the documents in the system. It refers to all the documents with their informations such as id, name, writer, and file in their encoded form
- ☐ **fileEncode:** The encoder for encoding the file submitted for the document after creation or being edited
- ☐ **idEncode:** The encoder for encoding the id of the document after creation
- ☐ **nameEncode:** The encoder for encoding the name of the document after creation or being edited
- ☐ **writtenByEncode:** The encoder for encoding the writer of the document after creation

Methods:

- ☐ **addDocument(documentDto: DocumentDTO);** The method to open add a document in the system, which might be a document for specific assignment or club related documents such as budget information
- ☐ **removeDocument(documentDto: DocumentDTO);** The method to remove a specific document from the system
- ☐ **editDocument(documentDto: DocumentDTO);** The method to edit a specific document in the system
- ☐ **viewDocument(documentDto: DocumentDTO);** The method to view a specific document in the system

- NotificationService



NotificationService

-notificationRepository : final NotificationRepository
-titleEncode : final TtitleEncode
-seenListEncode : final SeenListEncode
-idEncode : final IdEncode
-descriptionEncode : final DescriptionEncode

+addNotification(notificationDto : NotificationDTO) : void
+removeNotification(notificationDto : NotificationDTO) : void

Fig. 3.22. NotificationService

When a user tries to create a notification, it encodes the information of notifications such as Title, SeenList, Id and Description into the NotificationRepository.

Attributes:

- ☐ **notificationRepository:** The repository that stores all the notifications in the system. It refers to all the notifications with their informations such as id, description, title, etc. in their encoded form
- ☐ **titleEncode:** The encoder for encoding the title of a specific notification after creation
- ☐ **descriptionEncode:** The encoder for encoding the description of a specific notification after creation
- ☐ **idEncode:** The encoder for encoding the id of a specific notification after creation
- ☐ **seenListEncode:** The encoder for encoding the seen list of a specific forum after creation and while it is active

Methods:

- ☐ **addNotification(notificationDto: NotificationDTO);** The method to create a specific notification by clubs, admins, or advisors for specific events, assignments, or system related issues
- ☐ **removeNotification(notificationDto: NotificationDTO);** The method to delete a specific notification from the system

3.4.3 Database Layer

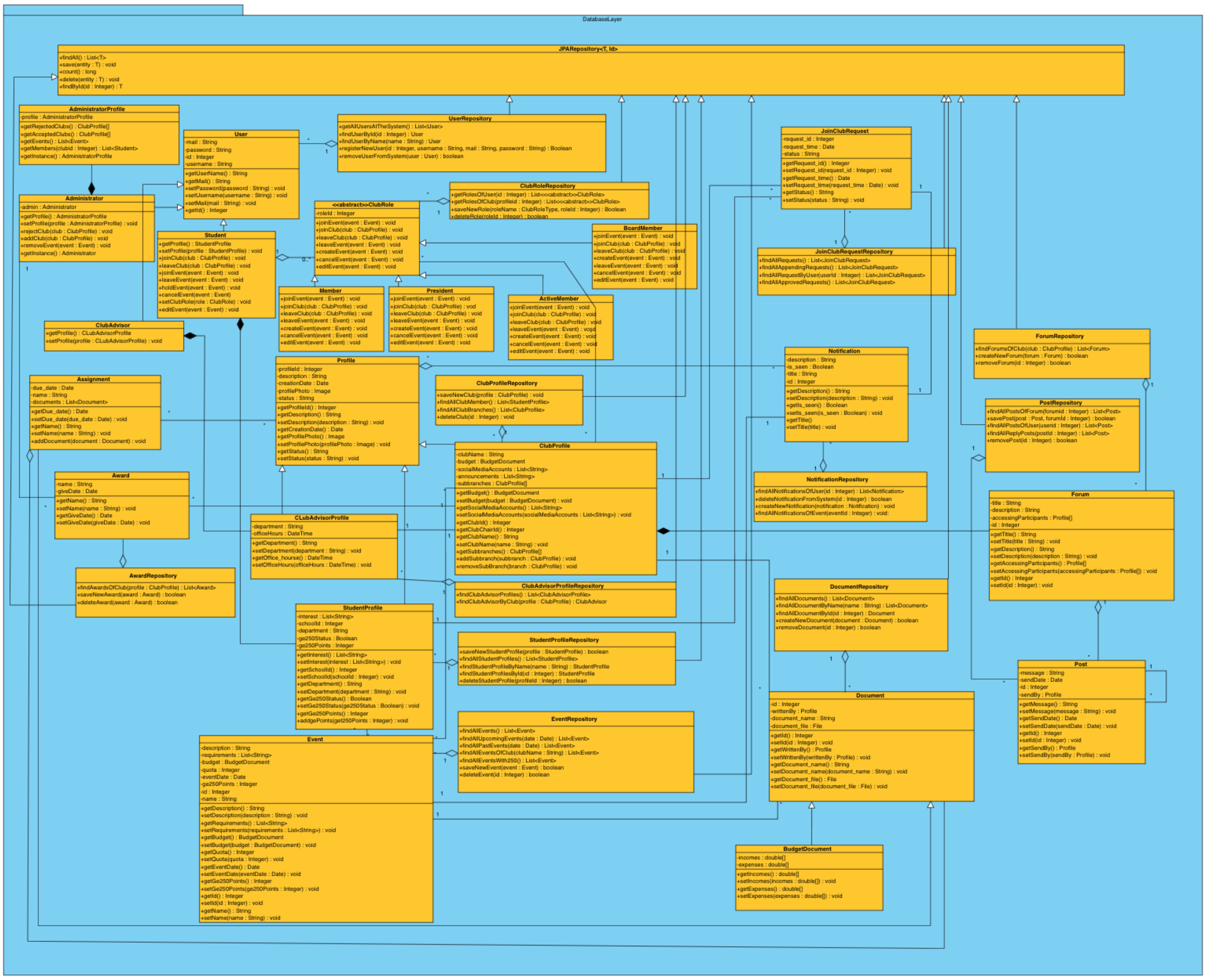


Fig. 3.23. Database layer class diagram representing entities and repositories

Class Explanations at Database Layer:

a) Entities:

- User

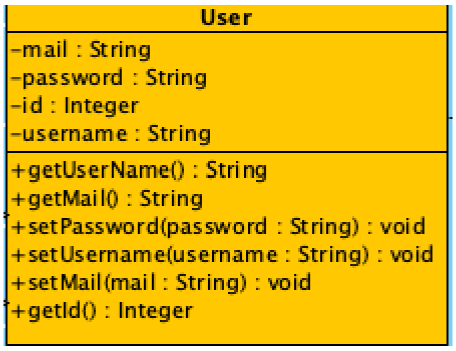


Fig. 3.24. User

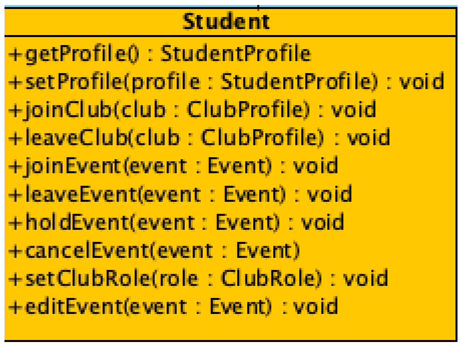
User is the entity representing the users which will use the system. It is the super class of other user types in the system.
Attributes:

- ☐ **mail:** Mail information of the user. Type is string.
- ☐ **password:** Password of the user. Type is string
- ☐ **id:** The primary key of the users. It will be auto-generated by Spring and type is string.
- ☐ **username:** Username of the user which is used at sign up and login processes.

Methods:

- ☐ **getUserName(); getMail(); getId();** The getter methods for attributes, password does not have a getter because of security concerns.
- ☐ **setUsername(username: String); setMail(mail: String), setPassword(password: String):** The setter methods for attributes. Id does not have a setter because it is auto generated by the database.

- Student:

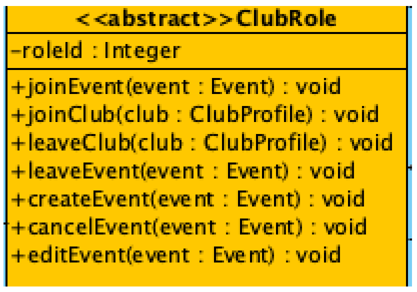

Student
+getProfile() : StudentProfile
+setProfile(profile : StudentProfile) : void
+joinClub(club : ClubProfile) : void
+leaveClub(club : ClubProfile) : void
+joinEvent(event : Event) : void
+leaveEvent(event : Event) : void
+holdEvent(event : Event) : void
+cancelEvent(event : Event)
+setClubRole(role : ClubRole) : void
+editEvent(event : Event) : void
Fig. 3.25. Student

Student is a user type in the system. It has an inheritance relationship with the user. Besides the attributes coming from the user, it has an aggregation relationship with its profile class where the information is held. Most importantly, it has a composition relationship with clubrole which shows its strategy for the club.
Methods:

- ☐ **getProfile(): StudentProfile**
  This method returns the profile of the student.
- ☐ **setProfile(profile: StudentProfile): void**
  This method sets the profile of the student.
- ☐ **setClubRole(role: ClubRole): void**
  This method is to set the strategy of the student for the club. From this strategy all the methods below will be called.
- ☐ **joinClub(club: ClubProfile): void**
- ☐ **leaveClub(club: ClubProfile): void**
- ☐ **joinEvent(event: Event): void**
- ☐ **leaveEvent(event: Event): void**
- ☐ **holdEvent(event: Event):void**
- ☐ **cancelEvent(event:Event):void**

☐ **editEvent(event: Event):void**

- ClubRole:


<<abstract>>ClubRole
-roleId : Integer
+joinEvent(event : Event) : void
+joinClub(club : ClubProfile) : void
+leaveClub(club : ClubProfile) : void
+leaveEvent(event : Event) : void
+createEvent(event : Event) : void
+cancelEvent(event : Event) : void
+editEvent(event : Event) : void
Fig. 3.26. ClubRole

ClubRole represents the strategy of the student for a particular club. This class is used at the strategy design pattern and carries abstract methods which will be implemented at sub methods according to the role.

Properties:

☐ **roleId: Integer**
Shows the id of the specific role at the system

Methods:

☐ **joinEvent(event: Event): void**
This method is to become a participant of an event.

☐ **joinClub(club: ClubProfile): void**
This method is to become a member of the club.

☐ **leaveClub(club: ClubProfile):void**
This method is to exit from a club.

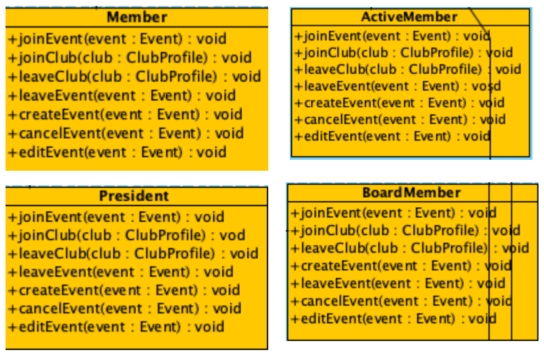
☐ **leaveEvent(event: Event): void**
This method is stop being a participant of an event

☐ **createEvent(event: Event):void**
This method is to create an event.

☐ **cancelEvent(event: Event):void**
This method cancels an event

☐ **editEvent(event: Event): void**
This method edits an event.

- Member - Active Member - Board Member - President


Member
+joinEvent(event : Event) : void
+joinClub(club : ClubProfile) : void
+leaveClub(club : ClubProfile) : void
+leaveEvent(event : Event) : void
+createEvent(event : Event) : void
+cancelEvent(event : Event) : void
+editEvent(event : Event) : void
ActiveMember
+joinEvent(event : Event) : void
+joinClub(club : ClubProfile) : void
+leaveClub(club : ClubProfile) : void
+leaveEvent(event : Event) : void
+createEvent(event : Event) : void
+cancelEvent(event : Event) : void
+editEvent(event : Event) : void
President
+joinEvent(event : Event) : void
+joinClub(club : ClubProfile) : vod
+leaveClub(club : ClubProfile) : void
+leaveEvent(event : Event) : void
+createEvent(event : Event) : void
+cancelEvent(event : Event) : void
+editEvent(event : Event) : void
BoardMember
+joinEvent(event : Event) : void
+joinClub(club : ClubProfile) : void
+leaveClub(club : ClubProfile) : void
+createEvent(event : Event) : void
+leaveEvent(event : Event) : void
+cancelEvent(event : Event) : void
+editEvent(event : Event) : void
Fig. 3.27. Role strategies

These classes show the strategies for clubs. For example; when the strategy of the student is a board member, the logic for editing events will exist; however if the student strategy is member then this method will throw an exception.

- Profile

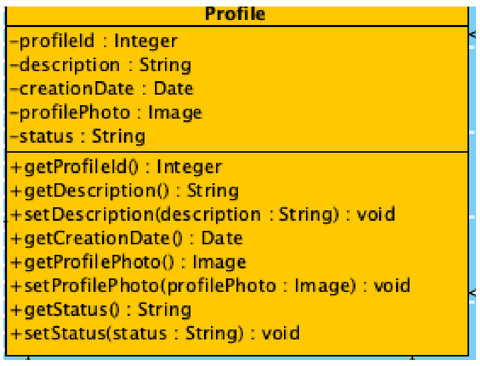

Profile
-profileId : Integer
-description : String
-creationDate : Date
-profilePhoto : Image
-status : String
+getProfileId() : Integer
+getDescription() : String
+setDescription(description : String) : void
+getCreationDate() : Date
+getProfilePhoto() : Image
+setProfilePhoto(profilePhoto : Image) : void
+getStatus() : String
+setStatus(status : String) : void

Fig. 3.28 Profile

This class represents the user profiles of the system. It is the super class of other profile types in the system and connected to them by inheritance relations.

Properties:

- ❏ **profileId: Integer**
  The id of the profile which will be auto generated and unique for profiles.
- ❏ **description: String**
  The description of the profile.
- ❏ **creationDate: Date**
  Shows when the profile is created, in other words, when the user has entered the system.
- ❏ **status: String**
  Shows the current status of a profile whether it is active or not.

Methods:

- ❏ **getProfileId():Integer, getDescription(): String, getCreationDate(): Date; getProfilePhoto(): Image**
  These are the getter methods for the attributes of the profile
- ❏ **setDescription(description: String): void, setProfilePhoto(profilePhoto: Image): void, setStatus(status: String): void**
  These are the setter methods for some attributes of the profile. There is no setter for creation date and id because they are auto-generated. They won't be set later.

- StudentProfile

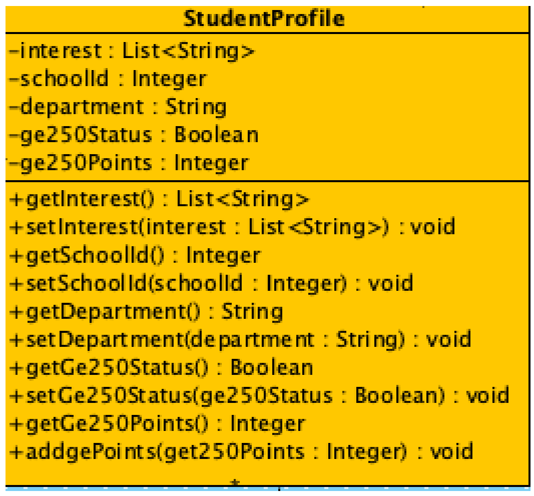

StudentProfile
-interest : List<String>
-schoolId : Integer
-department : String
-ge250Status : Boolean
-ge250Points : Integer
+getInterest() : List<String>
+setInterest(interest : List<String>) : void
+getSchoolId() : Integer
+setSchoolId(schoolId : Integer) : void
+getDepartment() : String
+setDepartment(department : String) : void
+getGe250Status() : Boolean
+setGe250Status(ge250Status : Boolean) : void
+getGe250Points() : Integer
+addgePoints(get250Points : Integer) : void

Fig. 3.29 StudentProfile

This class holds extra information related to students. It is a subclass of profile and has an aggregation relationship with student.

Properties:

- ☐ **interest: List<String>**
  Holds the interests of the student.
- ☐ **schoolId: Integer**
  Holds the school id of the student
- ☐ **department: String**
  Holds the department information of the student.
- ☐ **ge250Status: boolean**
  Shows whether the student is taking ge250 course currently
- ☐ **ge250Points: Integer**
  If the student takes ge250, it holds the current collected points, if not it is simply 0.

Methods:

- ☐ **getInterest(): List<String>, getSchoolId(): Integer, getDepartment(): String, getGe250Status(): boolean, getGe250Points(): Integer**
  These are the getter methods for the attributes
- ☐ **setInterest(interest: List<Interest>): void, setSchoolId(schoolId: Integer):void, setDepartment(department: String): void, setGe250PointsStatus(ge250Status:boolean): void**
  These are the setter methods for the attributes.
- ☐ **addgePoints(get250Points: Integer):void**
  This method is to add points to a student when it attends an event.

- ClubAdvisor

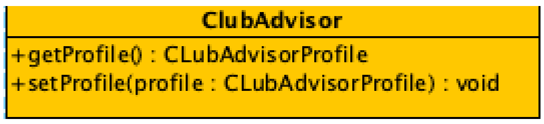

ClubAdvisor
+getProfile() : CLubAdvisorProfile
+setProfile(profile : CLubAdvisorProfile) : void

Fig. 3.30 ClubAdvsior

This class shows the club advisors which are the another user type at the system. Club advisors have their own profile.

Methods:

☐ **getProfile(): ClubAdvisorProfile**
This method returns the club advisor’s profile.

☐ **setProfile(profile: ClubAdvisorProfile): void**
This is a setter for club advisor.

- ClubAdvisorProfile

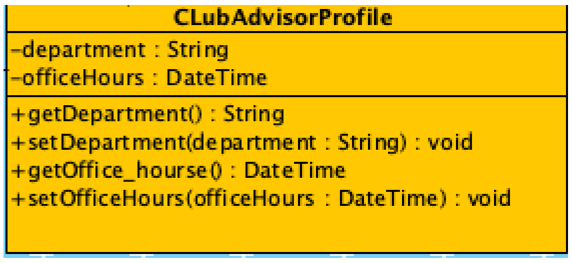

CLubAdvisorProfile
-department : String
-officeHours : DateTime
+getDepartment() : String
+setDepartment(department : String) : void
+getOffice_hourse() : DateTime
+setOfficeHours(officeHours : DateTime) : void

Fig. 3.31 ClubAdvisorProfile
This profile holds extra information about club advisors and it is associated with a ClubProfile which the advisor advises. This class belongs to a club advisor and because of that it has an aggregation relationship with club advisors.

Properties:

☐ **department: String**
Holds the department information of the instructor

☐ **office_hours: DateTime**
Holds the available hours of the instructor

Methods:

☐ **getDepartment(): String, getOfficeHours(): DateTime**
These are the getter methods for properties

☐ **setDepartment(department: String): void, setOfficeHours(officeHours: DateTime): void**
These are the setter methods for properties

- Administrator

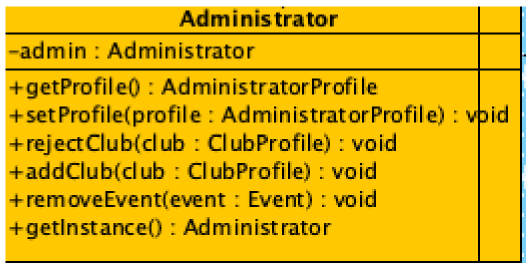

Administrator
-admin : Administrator
+getProfile() : AdministratorProfile
+setProfile(profile : AdministratorProfile) : void
+rejectClub(club : ClubProfile) : void
+addClub(club : ClubProfile) : void
+removeEvent(event : Event) : void
+getInstance() : Administrator

Fig. 3.32. Administrator
Administrator is another user type at the system and has a inheritance relationship with the user. There will be only one admin of the system because of that there s a singleton design pattern here. Administrator also has its own profile.

Properties

☐ **admin: Administrator**
Static administrator instance

Methods

- **getInstance(): Administrator**
  Returns the instance of the administrator.
- **getProfile(): AdministratorProfile**
  This method returns the administrator profile.
- **setProfile(profile: AdministratorProfile): void**
  This is a setter for the administrator.
- **rejectClub(club: ClubProfile): void**
  This method removes the club from the system
- **addClub(club: ClubProfile): void**
  This method adds a new club to the system
- **removeEvent(event: Event): void**
  This method removes the events from the system.

(Note: This class has a private constructor.)

- AdministratorProfile

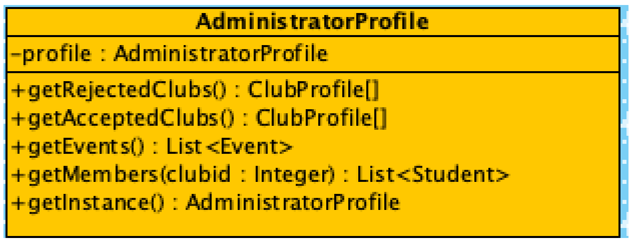

AdministratorProfile
-profile : AdministratorProfile
+getRejectedClubs() : ClubProfile[]
+getAcceptedClubs() : ClubProfile[]
+getEvents() : List<Event>
+getMembers(clubid : Integer) : List<Student>
+getInstance() : AdministratorProfile

Fig. 3.33 AdministratorProfile

This class is connected to administrator by aggregation and because there is only one admin at the system, there is only one profile belonging to that admin. Because of that, there is singleton pattern here also.

Properties

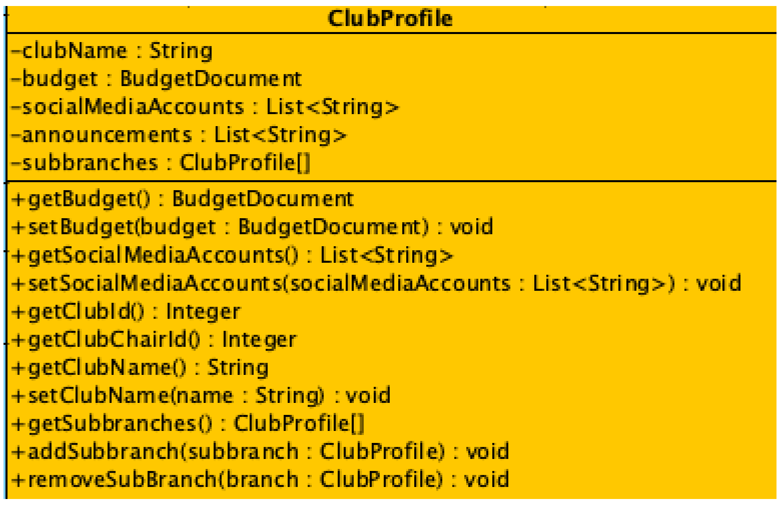
- **profile: AdministratorProfile**
  Static administrator profile instance

Methods

- **getInstance(): AdministratorProfile**
  Returns the instance of the administrator profile.
- **getRejectedClubs(): ClubProfile[]**
  Returns the rejected clubs in the system.
- **getAcceptedClubs: ClubProfile[]**
  Returns the accepted clubs in the system.
- **getEvents(): List<Event>**
  From the profile, administrators can check all the events.
- **getMembers(clubId: Integer): List<Student>**
  Administrators can check the club members from their profile

(Note: This class has a private constructor.)

- ClubProfile



```
ClubProfile
-clubName : String
-budget : BudgetDocument
-socialMediaAccounts : List<String>
-announcements : List<String>
-subbranches : ClubProfile[]
+getBudget() : BudgetDocument
+setBudget(budget : BudgetDocument) : void
+getSocialMediaAccounts() : List<String>
+setSocialMediaAccounts(socialMediaAccounts : List<String>) : void
+getClubId() : Integer
+getClubChairId() : Integer
+getClubName() : String
+setClubName(name : String) : void
+getSubbranches() : ClubProfile[]
+addSubbranch(subbranch : ClubProfile) : void
+removeSubBranch(branch : ClubProfile) : void
```

Fig. 3.34. ClubProfile
This class shows the clubs in the system. It has associations with student profiles which represents members, advisor profiles which shows the advisor, events which shows the events of the club and clubroles which shows the current roles at the club.

Properties:

- ☐ **clubName: String**
  Name of the club
- ☐ **budget: BudgetDocument**
  Holds the budget information of the club
- ☐ **socialMediaAccounts: List<String>**
  Accounts of the club.
- ☐ **announcements: List<String>**
  Announcements made by the club
- ☐ **subbranches: ClubProfile[]**
  Shows the subbranches of the club

Methods:

- ☐ **getBudget(): BudgetDocument, getSocialMediaAccounts(): List<String>, getClubId(): Integer; getClubChairId(): Integer; getClubName(): String, getSubbranches(): ClubProfile[]**
  These are the getter methods for the attributes
- ☐ **setBudget(budget: budgetDocument): void, setSocialMediaAccounts(socialMediaAccounts: List<String>): void, setClubName(name: string): void**
  These are the setter methods for attributes
- ☐ **addSubbranch(subbranch: ClubProfile):void**
  This method add a new branch to the club
- ☐ **removeSubbranch(subbranch: ClubProfile): void**
  This method removes a subbranch from the club

- Event

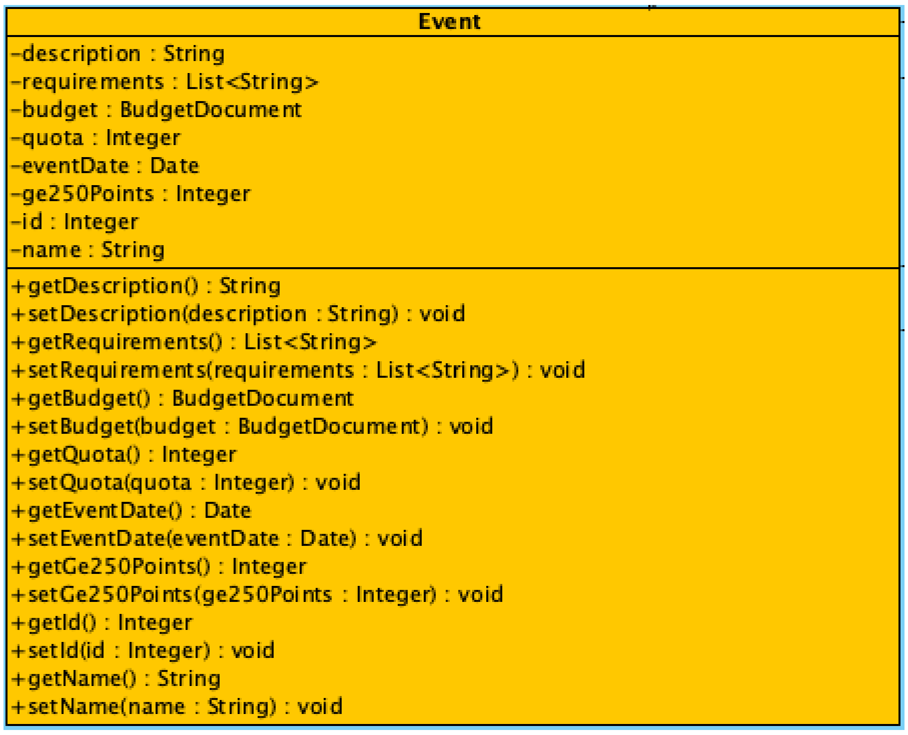

Event
-description : String
-requirements : List<String>
-budget : BudgetDocument
-quota : Integer
-eventDate : Date
-ge250Points : Integer
-id : Integer
-name : String
+getDescription() : String
+setDescription(description : String) : void
+getRequirements() : List<String>
+setRequirements(requirements : List<String>) : void
+getBudget() : BudgetDocument
+setBudget(budget : BudgetDocument) : void
+getQuota() : Integer
+setQuota(quota : Integer) : void
+getEventDate() : Date
+setEventDate(eventDate : Date) : void
+getGe250Points() : Integer
+setGe250Points(ge250Points : Integer) : void
+getId() : Integer
+setId(id : Integer) : void
+getName() : String
+setName(name : String) : void

Fig. 3.35 Event

Events are the activities held by the clubs. Because of that this class is associated with club profiles. In addition to that, to hold the documents related to the event, it is also associated with documents.

Properties:

- ☐ **description: String**
  Brief explanation of the event
- ☐ **name: String**
  Name of the event
- ☐ **requirements: List<String>**
  If participants need to bring anything to join the event
- ☐ **budget: BudgetDocument**
  Budget information of the event
- ☐ **quota: Integer**
  At max, how many person can access to the event
- ☐ **eventDate: Date**
  Date of the event
- ☐ **ge250Points: Integer**
  Shows whether the event has any points for ge250 course
- ☐ **id: Integer**
  id of the event which is unique for every event at the system.

Methods:

Methods consist of getters and setters for the attributes.

- Award

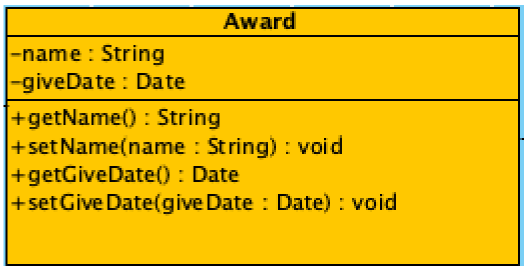


Fig. 3.36. Award

This class shows the awards which are given to the clubs by admin. Because of that it has associations with club profiles and admin.

Properties:

- ☐ **name: String**
  This is the name of the award
- ☐ **giveDate: Date**
  The date of the award.

Methods:

- ☐ **getName(): String, getGivenDate(): Date**
  Getters for the attributes
- ☐ **setName(name: String): void; setGiveDate(giveDate: Date): void**
  Setters for the attributes

- Document

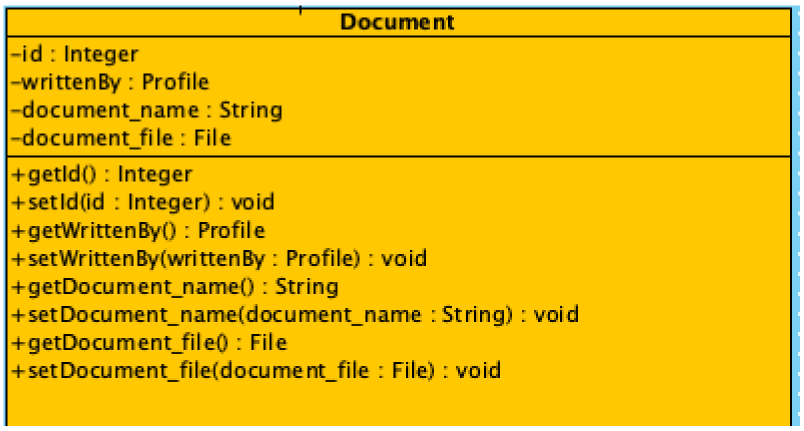

Document
-id : Integer
-writtenBy : Profile
-document_name : String
-document_file : File
+getId() : Integer
+setId(id : Integer) : void
+getWrittenBy() : Profile
+setWrittenBy(writtenBy : Profile) : void
+getDocument_name() : String
+setDocument_name(document_name : String) : void
+getDocument_file() : File
+setDocument_file(document_file : File) : void

Fig. 3.37. Document

Document shows the class that holds the documents related to the club duties. Because of that it has an association with club profiles and events.

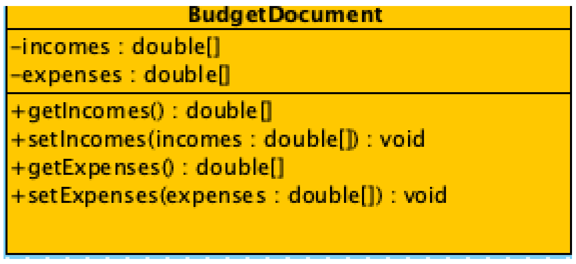
Properties:

- ☐ **id: Integer**
  The unique id that distinguishes documents.
- ☐ **writtenBy: Profile**
- ☐ The profile which has written the document
- ☐ **document_name: String**
  Name of the document
- ☐ **document_file: File**
  The related files of the document

Methods:

The methods of this class consists of the getters and setters for attributes.

- BudgetDocument



BudgetDocument
-incomes : double[]
-expenses : double[]
+getIncomes() : double[]
+setIncomes(incomes : double[]) : void
+getExpenses() : double[]
+setExpenses(expenses : double[]) : void

Fig. 3.38 BudgetDocument

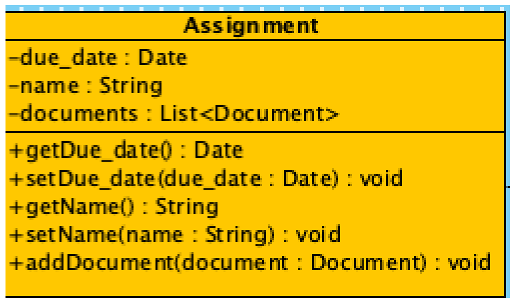
This document is a specific type of document which holds the budget information of the club with incomes and expenses. It is connected to document class through inheritance

Properties:

☐ **incomes: double[]:**
Income information

☐ **expenses: double[]:**
Expense information

- Assignment



Assignment
-due_date : Date
-name : String
-documents : List<Document>
+getDue_date() : Date
+setDue_date(due_date : Date) : void
+getName() : String
+setName(name : String) : void
+addDocument(document : Document) : void

Fig. 3.39. Assignment

Assignments are a special type of document at the system. It represents the jobs given to club members from a higher authority at the club. due to the fact that assignments can have sub assignments and can be related to other type of documents, assignments have a composite design pattern relationship. Because of that it is both a subtype of document class but also associated with documents through composition.

Properties:

☐ **due_date: Date**
The date in which assignment must be finished

☐ **name: String**
Name of the assignment

☐ **documents: List<Document>**
The other documents at the assignment

Methods:

The methods include getters and setters for attributes.

☐ **addDocument(document: Document): void**
This method adds new documents to the assignments.

- Notification

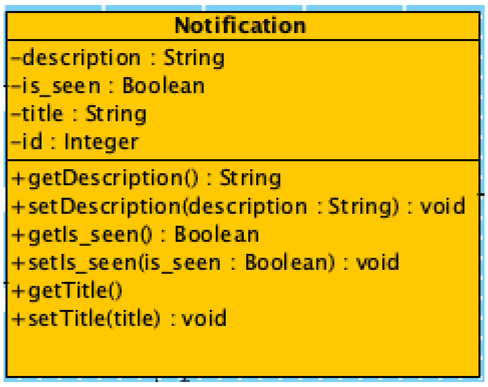


Fig.3.40 Notification

Notifications are the objects which are used to inform the users when a change occurs related to clubs and events. Because of that it is associated with event and clubprofile classes

Properties:

- ☐ **description: String**
  A message describing what the notification is about.
- ☐ **is_seen: boolean**
  Shows whether the notification has been seen by the profile
- ☐ **title: String**
  Title of the notification
- ☐ **id: Integer**
  Unique id of the notification

Methods:

The methods of this class are getters and setters for the attributes.

- JoinClubRequest

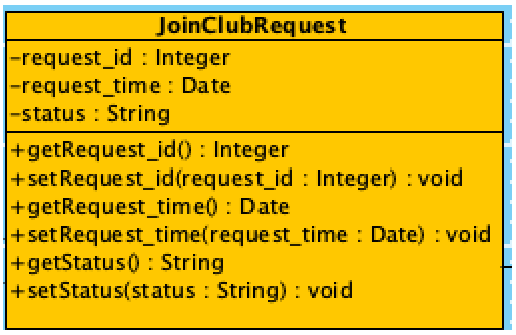

JoinClubRequest
-request_id : Integer
-request_time : Date
-status : String
+getRequest_id() : Integer
+setRequest_id(request_id : Integer) : void
+getRequest_time() : Date
+setRequest_time(request_time : Date) : void
+getStatus() : String
+setStatus(status : String) : void

Fig. 3.41 JoinClubRequest

This class represents the request sent by the student to the club profiles to join the club.

Properties:

- ☐ **request_id: Integer**
  The unique id of the request
- ☐ **request_time: Date**
  The time when the request is made
- ☐ **status: String**

This property shows whether the request is accepted, not accepted, still waiting vs.

Methods:

Methods of this class are getters and setters for attributes.

- Forum

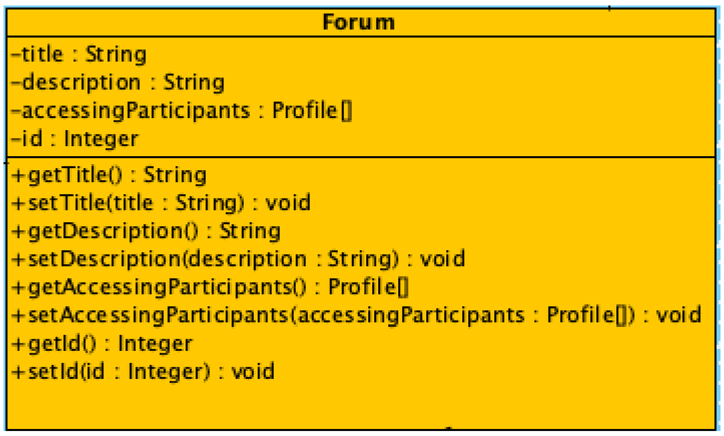

Forum
-title : String
-description : String
-accessingParticipants : Profile[]
-id : Integer
+getTitle() : String
+setTitle(title : String) : void
+getDescription() : String
+setDescription(description : String) : void
+getAccessingParticipants() : Profile[]
+setAccessingParticipants(accessingParticipants : Profile[]) : void
+getId() : Integer
+setId(id : Integer) : void

Fig. 3.42 Forum

Every club in the system has a forum where users can ask questions about the club.Because of that this forum class has an aggregation relationship with club profiles.

Properties:

- ☐ **title: String**
  Title of the forum
- ☐ **description: String**
  A brief description about the forum
- ☐ **accessingParticipants: Profile[]**
  The profiles that can see the forum
- ☐ **id: Integer**
  The unique id for forums

Methods:

This class has getters and setters for its attributes

- Post:

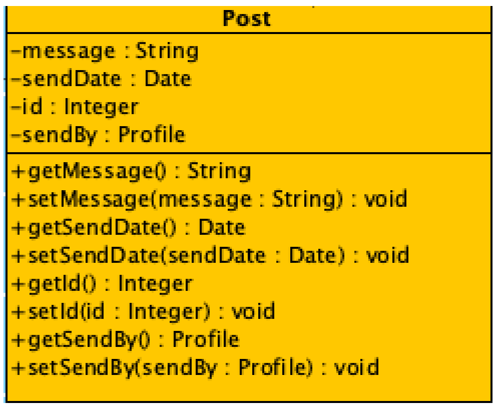


Fig. 3.43. Post

Posts are the messages that are send at the forums. Because of that, they have an association with forum class. In addition to that, a post can be replied by another post because of that it has a recursive relationship with itself as well.

Properties:

- ☐ **message: String**
  The message at the post
- ☐ **sendDate: Date**
  The time that the post has been sent
- ☐ **id: Integer**
  The unique identifier of the post
- ☐ **sendBy: Profile**
  The person that send the post.

Methods:

The methods are getters and setters for attributes

b) Repositories:

- JpaRepository

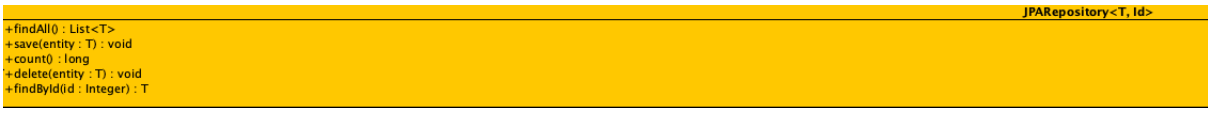


Fig. 3.44 JpaRepository

JpaRepository is an interface provided by the Spring framework. It enables communication with the database system and has built-in methods to save entities and find them. All repositories in the system implement this interface.

Methods:

- **public List<T> findAll():**
  Finds the entities from type T in the database system.
- **public void save(entity: T):**
  Saves a new entity to the database system.
- **public void delete(entity: T):**
  Deletes an entity from the database system.
- **public long count():**
  Returns the number of entities (rows) at the database system.
- **public T findById(id: Integer):**
  Finds the related entity from primary key information in the database system.

- UserRepository:

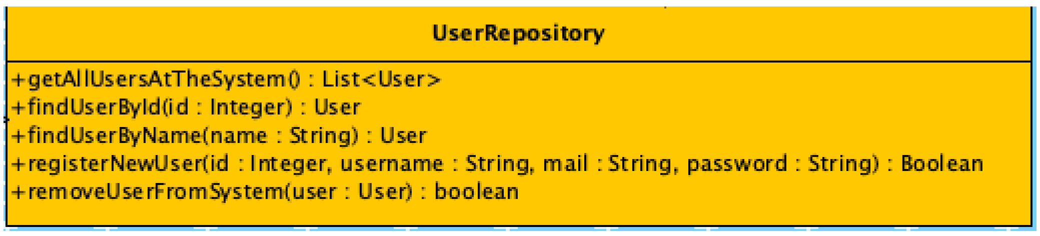

UserRepository
+getAllUsersAtTheSystem() : List<User>
+findUserById(id : Integer) : User
+findUserByName(name : String) : User
+registerNewUser(id : Integer, username : String, mail : String, password : String) : Boolean
+removeUserFromSystem(user : User) : boolean

Fig. 3.45 UserRepository

User repository implements JpaRepository and enables access to user entities in the database system. It consists of the users of the system.

Methods:

- ☐ **getAllUsersAtTheSystem():List<User>**

This method returns all the users that are registered in the system.

- ☐ **findUserById(id: Integer): User**
  This method finds the user with that specified id.
- ☐ **findUserByName(name: String): User**
  This method finds the user with specified username.
- ☐ **registerNewUser(id: Integer, username: String, mail: String, password: String): boolean**
  This method is used to create a new user within the database system. It saves the user and returns a boolean to indicate whether the process is successful.
- ☐ **removeUserFromSystem(user: User): boolean**
  This method deletes the user from the system. It returns boolean to show whether the user has been deleted or not.

- ClubRoleRepository

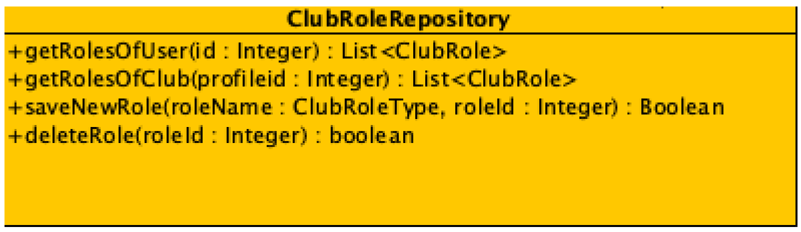

ClubRoleRepository
+getRolesOfUser(id : Integer) : List<ClubRole>
+getRolesOfClub(profileid : Integer) : List<ClubRole>
+saveNewRole(roleName : ClubRoleType, roleId : Integer) : Boolean
+deleteRole(roleId : Integer) : boolean

Fig. 3.46 ClubRoleRepository
This is the class that enables us to reach the roles of the students at clubs.
Methods:

- ☐ **getRolesOfUser(id: Integer): List<ClubRole>**
  This method finds the roles of the user specified with id.
- ☐ **getRolesOfClub(profileId: Integer): List<ClubRole>**
  This method finds the roles at the club.
- ☐ **saveNewRole(roleName: ClubRoleType, roleId: Integer): boolean**
  This method saves a new role to the system. Returns a boolean to indicate whether the process is successful.
- ☐ **deleteRole(roleId: Integer):**
  This method deletes the role with the specified id from the system. Returns a boolean to indicate whether the process is successful.

- StudentProfileRepository

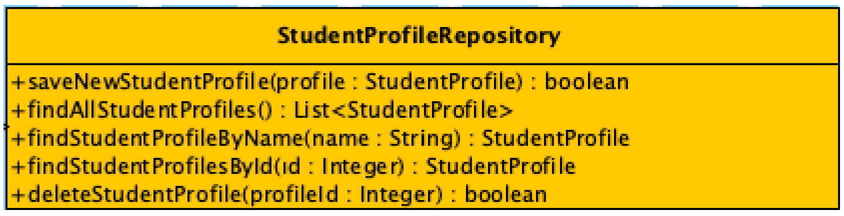

StudentProfileRepository
+saveNewStudentProfile(profile : StudentProfile) : boolean
+findAllStudentProfiles() : List<StudentProfile>
+findStudentProfileByName(name : String) : StudentProfile
+findStudentProfilesById(id : Integer) : StudentProfile
+deleteStudentProfile(profileId : Integer) : boolean

Fig. 3.47 StudentProfileRepository
This is the class that enables to access the student profiles at the database system.
Methods:

- ☐ **saveNewStudentProfile(profile: StudentProfile): boolean**

This method saves a new profile to the system. Returns a boolean to indicate whether the process is successful.

- ☐ **findAllStudentProfiles(): List<StudentProfile>:**
  This method finds all student profiles in the system.
- ☐ **findStudentProfileByName(name: String): StudentProfile**
  This method finds profiles by the username.
- ☐ **findStudentProfileById(id: Integer): StudentProfile**
  This method finds the profile by id.
- ☐ **deleteStudentProfile(profileId: Integer): boolean**
  Deletes the student profile specified with that id. Returns a boolean to indicate whether the process is successful.

- ClubAdvisorProfileRepository

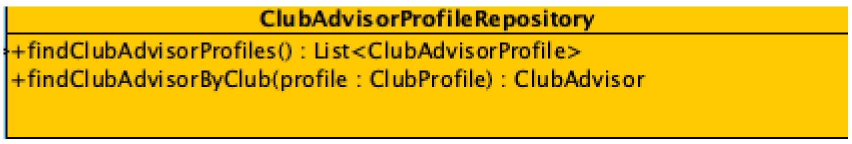

ClubAdvisorProfileRepository
+findClubAdvisorProfiles() : List<ClubAdvisorProfile>
+findClubAdvisorByClub(profile : ClubProfile) : ClubAdvisor

Fig. 3.48 ClubAdvisorProfileRepository
This is the class to access the club advisor profiles in the database system.
Methods:

- ☐ **findClubAdvisorProfiles(): List<ClubAdvisorProfile>**
  This method returns the advisor profiles in the system.
- ☐ **findClubAdvisorsByClub(profile: ClubProfile): ClubAdvisor**
  This method returns the advisor of the specified club.

- ClubProfileRepository

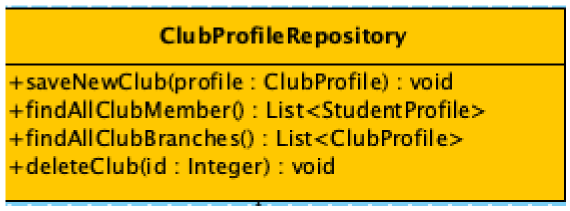

ClubProfileRepository
+saveNewClub(profile : ClubProfile) : void
+findAllClubMember() : List<StudentProfile>
+findAllClubBranches() : List<ClubProfile>
+deleteClub(id : Integer) : void

Fig. 3.49 ClubProfileRepository
This class enables to access the clubs in the database system.
Methods:

- ☐ **saveNewClub(profile: ClubProfile): void**
  This method adds a new club to the database system
- ☐ **findAllClubMembers(): List<StudentProfile>**
  This method returns all the members of the club
- ☐ **findAllClubBranches(): List<ClubProfile>**
  This method returns the subbranches
- ☐ **deleteClub(id: Integer): void**
  This method deletes the club with the specified id from the database.

- EventRepository

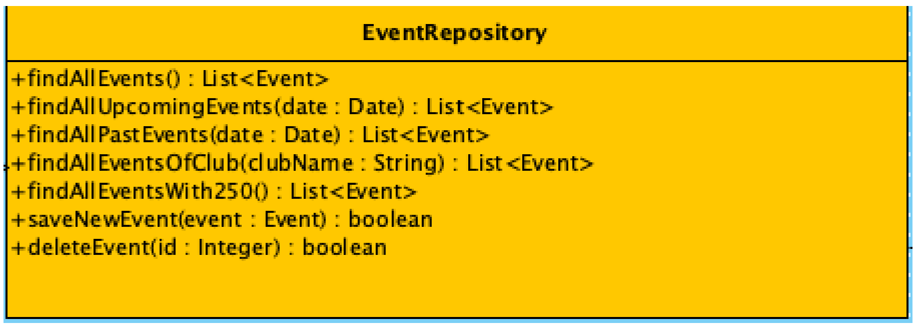

EventRepository
+findAllEvents() : List<Event>
+findAllUpcomingEvents(date : Date) : List<Event>
+findAllPastEvents(date : Date) : List<Event>
+findAllEventsOfClub(clubName : String) : List<Event>
+findAllEventsWith250() : List<Event>
+saveNewEvent(event : Event) : boolean
+deleteEvent(id : Integer) : boolean

Fig. 3.50 EventRepository

This repository enables access to the events in the database.

Methods:

- ☐ **findAllEvents(): List<Event>**
  This method returns all the events at the system
- ☐ **findAllUpcomingEvents(date: Date): List<Event>**
  This method takes the current date and finds all the future events
- ☐ **findAllPastEvents(date: Date): List<Event>**
  This method takes the current date and finds all past events
- ☐ **findAllEventsOfClub(clubName: String): List<Event>**
  This method returns the events of the specified club.
- ☐ **findAllEventsWith250():List<Event>**
  This method returns the events which has ge250 points
- ☐ **saveNewEvent(event: Event): boolean**
  This method saves a new event to the database.
- ☐ **deleteEvent(id: Integer): boolean**
  This method deletes an event with the specified id from database system

- AwardRepository

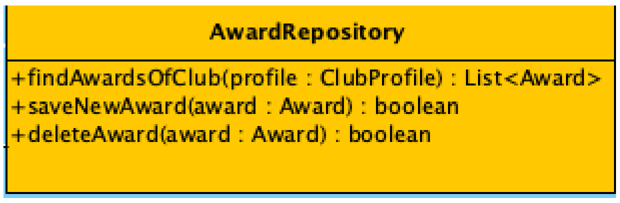

AwardRepository
+findAwardsOfClub(profile : ClubProfile) : List<Award>
+saveNewAward(award : Award) : boolean
+deleteAward(award : Award) : boolean

Fig. 3.51 AwardRepository

This is the repository class to access the awards.

Methods:

- ☐ **findAwardsOfClub(profile: ClubProfile): List<Award>**
  Returns the awards of the club
- ☐ **saveNewAward(award: Award): boolean**
  Gives a new award to a club
- ☐ **deleteAward(award: Award):boolean**
  Deletes a given award from system

- DocumentRepository

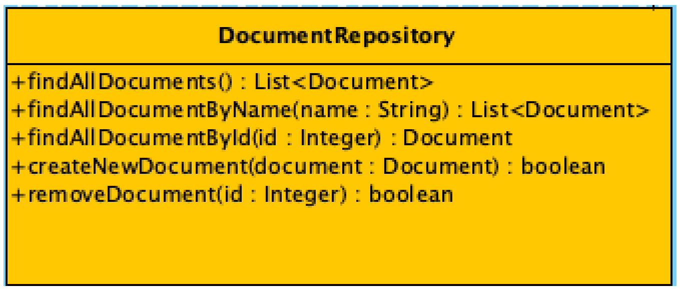

DocumentRepository
+findAllDocuments() : List<Document>
+findAllDocumentByName(name : String) : List<Document>
+findAllDocumentById(id : Integer) : Document
+createNewDocument(document : Document) : boolean
+removeDocument(id : Integer) : boolean

Fig. 3.52 DocumentRepository
This repository enables access to the documents in the system.
Methods:

- ☐ **findAllDocuments(): List<Document>**
  This method returns all the documents at the system
- ☐ **findAllDocumentsByName(name:String): List<Document>**
  This method returns the documents with specified name
- ☐ **findAllDocumentById(id: Integer): Document**
  This method finds the document with specified id
- ☐ **createNewDocument(document: Document): boolean**
  This method creates a new document at the system
- ☐ **removeDocument(id: Integer): boolean**
  This method removes the document with specified id from the system

- NotificationRepository

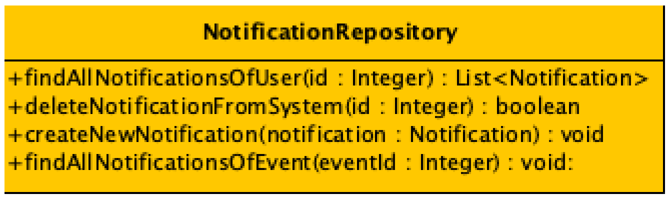

NotificationRepository
+findAllNotificationsOfUser(id : Integer) : List<Notification>
+deleteNotificationFromSystem(id : Integer) : boolean
+createNewNotification(notification : Notification) : void
+findAllNotificationsOfEvent(eventId : Integer) : void:

Fig. 3.53 NotificationRepository
This repository can access the notification entities in the database system.
Methods:

- ☐ **findAllNotificationsOfUser(id: Integer): List<Notifications>**
  This method returns all the notifications send to the user
- ☐ **deleteNotificationFromSystem(id: Integer): boolean**
  This method deletes the notification from system
- ☐ **createNewNotification(notification: Notification): void**
  This method generates a new notification
- ☐ **findAllNotificationOfEvent(eventId: Integer): void**
  This method finds all notifications related to an event.

- JoinClubRequestRepository

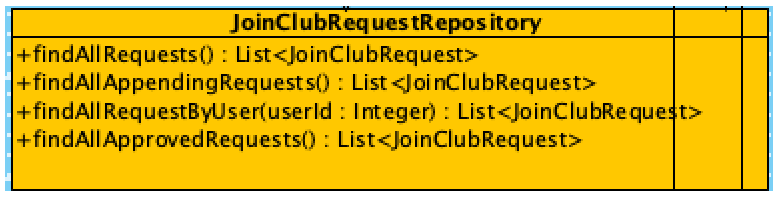
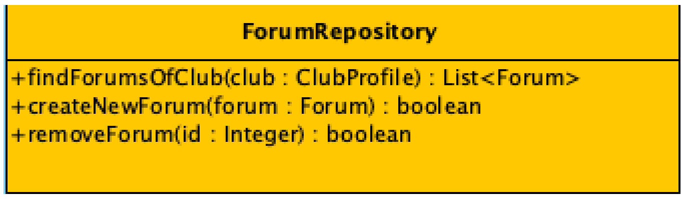

JoinClubRequestRepository
+findAllRequests() : List<JoinClubRequest>
+findAllAppendingRequests() : List <JoinClubRequest>
+findAllRequestByUser(userId : Integer) : List<JoinClubRequest>
+findAllApprovedRequests() : List<JoinClubRequest>

Fig. 3.54. JoinClubRequestRepository
This repository enables access to the requests at the system.
Methods:

- ☐ **findAllRequests(): List<JoinClubRequest>**
  This method returns all the requests at the system
- ☐ **findAllAppendingRequests(): List<JoinClubRequest>**
- ☐ This method returns all the appending requests
- ☐ **findAllRequestsByUser(userId: Integer): List<JoinClubRequest>**
  This method shows all the requests made by an user
- ☐ **findAllApprovedRequests(): List<JoinClubRequest>**
  This method shows all the approved requests

- Forum Repository

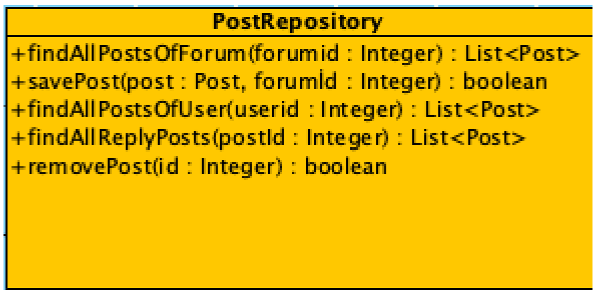


```
ForumRepository
+findForumsOfClub(club : ClubProfile) : List<Forum>
+createNewForum(forum : Forum) : boolean
+removeForum(id : Integer) : boolean
```

Fig. 3.55 ForumRepository
This class is the repository to get the forums at the system
Methods:

- ☐ **findForumsOfClub(club:ClubProfile):List<Forum>**
  This method returns the forum belonging to a club
- ☐ **createNewForum(forum: Forum): boolean**
  This method adds a new forum to the system
- ☐ **removeForum(id: Integer): boolean**
  This method removes a forum from the system

- PostRepository



```
PostRepository
+findAllPostsOfForum(forumid : Integer) : List<Post>
+savePost(post : Post, forumId : Integer) : boolean
+findAllPostsOfUser(userid : Integer) : List<Post>
+findAllReplyPosts(postId : Integer) : List<Post>
+removePost(id : Integer) : boolean
```

Fig. 3.56. PostRepository
This class helps to access the posts at the database
Methods:

- ☐ **findAllPostsOfForum(forumId: Integer): List<Post>**
  This method returns all the posts belonging to a forum
- ☐ **savePost(post: Post, forumId: Integer): boolean**
  This method saves the post to a forum
- ☐ **findAllPostsOfUser(userId: Integer): List<Post>**
  This method finds the posts of a user
- ☐ **findAllReplyPosts(postId:Integer): List<Post>**

This method finds all the replies given to a post

- ☐ **removePost(id:Integer):boolean**
  This method removes a post from the system

3.5 Packages and Libraries

3.5.1 Packages Specific to the Project

3.5.1.1 Login/Signup Management Package

This package contains all the entity, control and boundary classes related to login and signup operations as well as security measures which involve authentication of the user.

3.5.1.2 Profile Package

This package contains all the entity, control and boundary classes related to profile and its use cases.

3.5.1.3 Assignment Package

This package contains all the entity, control and boundary classes related to assignment and its use cases.

3.5.1.4 Documentation Package

This package contains all the entity, control and boundary classes related to the document and its use cases.

3.5.1.5 Forum Package

This package contains all the entity, control and boundary classes related to forum endar and its use cases.

3.5.1.6 Event Package

This package contains all the entity, control and boundary classes related to the event and its use cases.

3.5.1.7 Role Package

This package contains all the entity, control and boundary classes related to role and its use cases.

3.5.2 External Libraries

The project will be developed on spring framework version 5.0 and java jdk version 8. All external libraries that will be used in this project will be able to support these versions. In addition to that, these external libraries will be added to the project by using Maven version 3.8.4 which is the latest.

3.5.2.1 Libraries related to Development:

- Spring Boot DevTools: This package helps to restart the application fast and provides built-in configurations for fast development.
- Spring Lombok: Spring framework requires annotations throughout development. This library helps to make the annotation process faster and less redundant.
- Spring Validation: This package enables the bean control of the spring projects by using Hibernate.
- Node Package Manager (NPM): To organize external libraries, make the required installation for the frontend react project, npm will be used.
- React Redux: This library enables global state control over React projects.
- React router v6: This library is essential to a web application created with react. Rendering correct pages/components according to the URL, navigation between pages will be handled by a react-router.
- Material UI v5.2.1: Material UI provides free UI components such as icons, icon buttons, etc.
- Bootstrap v5.1: To make our components responsive, the frontend will make use of an external library called bootstrap.

3.5.2.2 Libraries related to the Web:

- Spring Web: This package provides an embedded Apache Tomcat web server and enables applications to run.

3.5.2.3 Libraries related to Security:

- Spring Security: This package provides an authentication system for projects.

3.5.2.4 Libraries related to Data Management:

- Spring Data Jpa: This package provides entity control and relational mapping by using Hibernate.
- Spring Mysql Driver: This package provides connection to the Mysql service.

3.6 Deployment Diagram

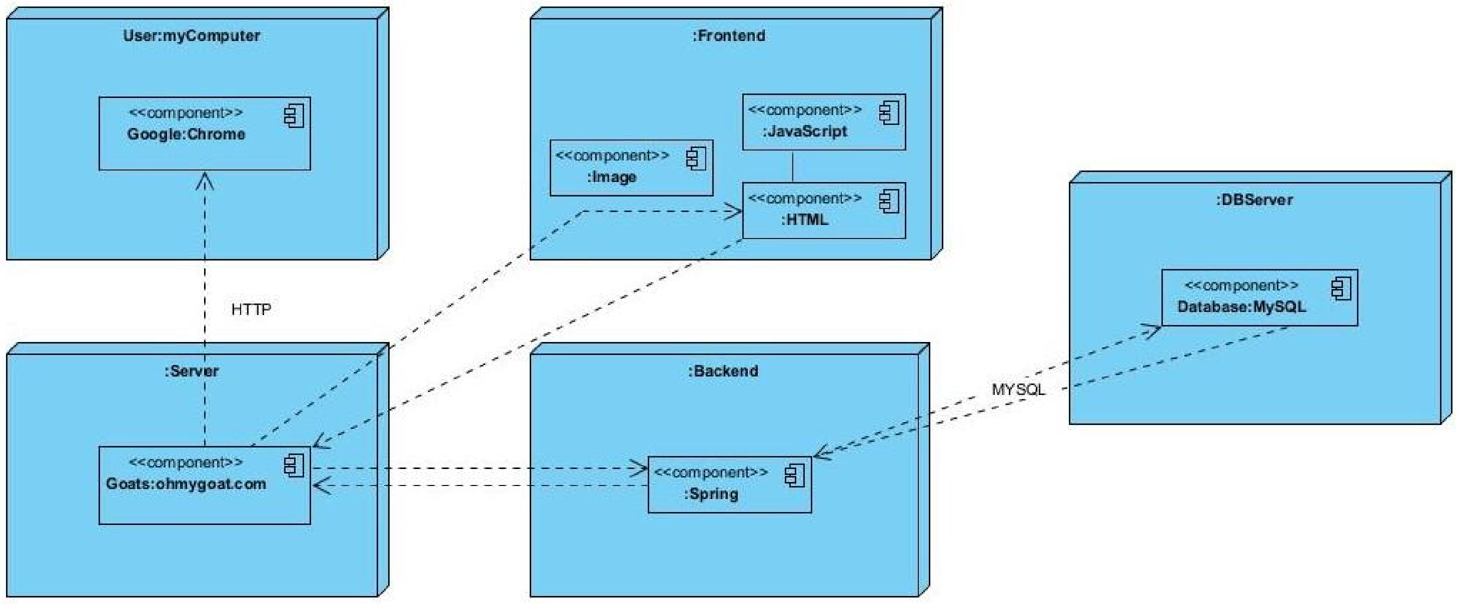


Fig. 3.57 Deployment Graph

3.7 Design Patterns

3.7.1 Strategy Design Pattern

In this project, we used the strategy design pattern at role relationships between students at clubs. Students can become club members, active members, board members and also presidents. This ranking system will alter dynamically throughout the program and sometimes the students will be upgraded to a higher rank whereas sometimes the student rank will be degraded. In order to change this behavior, the strategy design pattern is used. A student has a role which represents the strategy. By using the set role method of the student, this role behavior will be dynamically updated. According to the role he/she has, the student will call its club related methods upon its role. For example, when the program calls the edit event method of the student, this method will be called upon the role of the student.

```
public void editEvent(Event event){
        clubRole.editEvent();
}
```

At member role and active member role, this method will throw exception whereas if the student role is board member or president, students will be able to edit events. This way, the control of the club system will be distributed upon appropriate people.

3.7.2 Composite Design Pattern

The assignments and documents at the system are related to each other through a composite design pattern. An assignment is a type of document; however it can also contain other documents related to the assignment. For example, an assignment might be related to budget issues and can contain a budget document inside. In addition, assignments can consist of sub assignments as well. Because of this recursive relationship, there will be a composite design pattern used there.

This pattern can be at the database level class diagram and final object diagram where assignments are connected to documents through inheritance but also have an aggregation relationship with the documents class.

3.7.3 Singleton Design Pattern

In our system, there exists only one admin user to avoid complications. At this point, the system will use a singleton design pattern. In order to do that, both admin and admin profile will have a static instance property of themselves. These classes will have private constructors and at the implementation, the instances will be accessed through the get instance method defined in these classes. (these methods can be seen at database layer diagram)

4. Glossary and References

[1] “Java Polymorphism”, W3Schools. https://www.w3schools.com/java/java_polymorphism.asp. [Accessed 27.11.2021]
[2] “Adding Salt to Hashing: A Better Way to Store Passwords”, Auth0. https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/. [accessed 28.10.2021]
[3] “Hardware Requirements for Web and Database Servers” ,Sana online help. https://help.sana-commerce.com/sana-commerce-83/installation/setup-web-and-database-server/hardware-requirements-for-web-and-database-servers. [Accessed 16.12.2021]