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

Introduction:

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 functionality and maintainability for the following reasons.

1.2.1 Functionality

In this project, our number one design goal is to cover and implement as many interactions between students and clubs as possible and enable these interactions to be completed on the internet so that all users can find a solution to all of their problems. Thus, we want our program to function in many different aspects of this domain. We want to accomplish every functionality that we have mentioned in the analysis report's functional requirements part and make this program as functional as possible for the students, club executives, advisors and system admins.

We will test functionality by comparing the finalized project's functionalities with the prepared functionalities written in the analysis report. We wish to fulfill at least 80% of the functionalities we wrote in the analysis report. 80% is particularly a good goal since in the analysis report, we might have written overkill functionalities that may actually be unnecessary or out of scope. Also, a trade-off of some minor functionalities and other major components of the project can be done for the sake of the overall integrity of our application. Furthermore, the bulk majority and core aspects of our system design will be implemented within 80% of the functional requirements. Since there is a limited time for us to finish the project and if some parts of our project have to be missing, we want to ensure that we will not give up on the main functionality of our system.

1.2.2 Maintainability

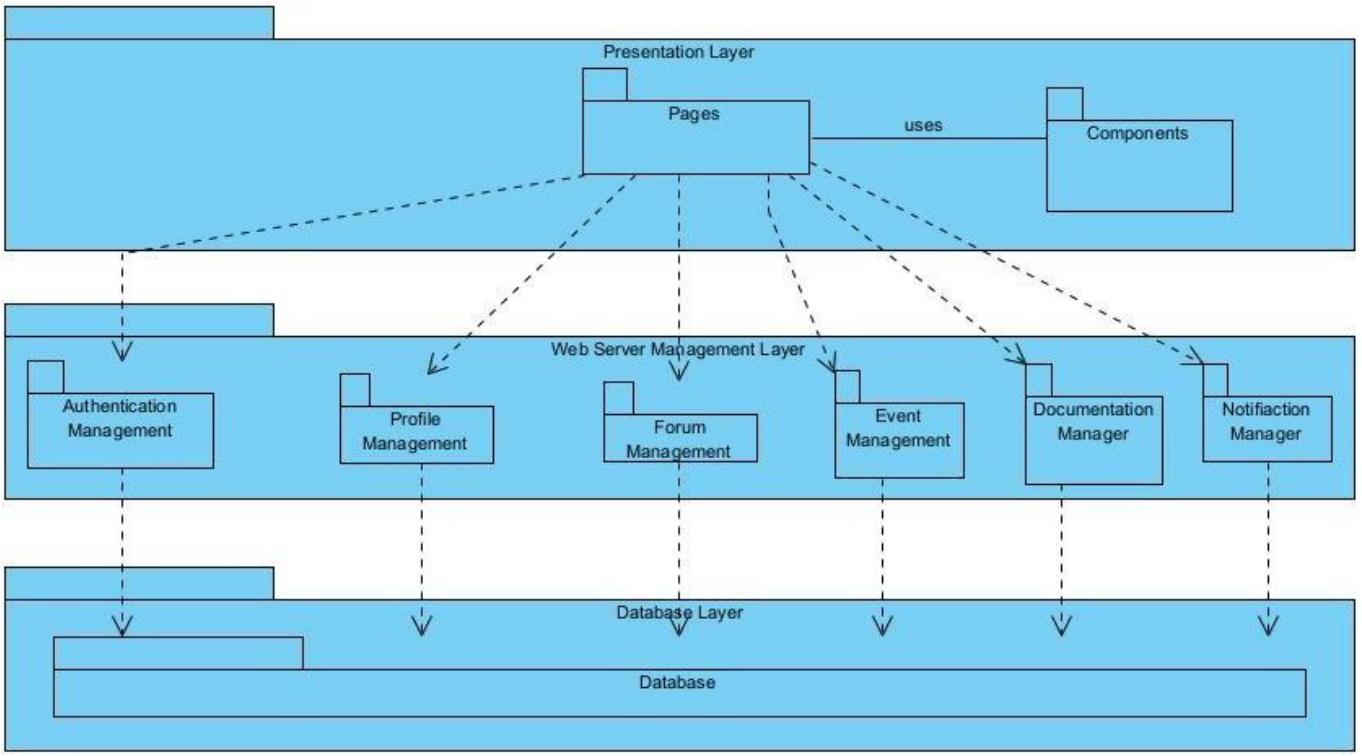
The second most important aim of this project is to create a system that can be used without problems for a long time. Therefore, the system should be easily maintainable and open to changes according to user feedback. Assuming and hoping that our project gets selected for being used next semester, we 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. We estimate a one-week range for fixing a bug. A new feature request is planned to be handled between two semesters.

Proposed Software Architecture

2.1 Subsystem decomposition



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.

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.

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.

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.

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. PeerPanda runs in web browsers like Google Chrome, Mozilla Firefox, Internet Explorer, Safari, and Opera in their latest versions. However, due to APIs and external libraries, web browsers with old versions like Internet Explorer 8 could be incompatible.

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, according to previous experiences, 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.

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 Business layer's Database interface subsystem and the repositories in the database 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.

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 is 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.

For the sensitive personal data storage, our first approach will be to use Google as the login credentials. By doing this, our users will enter the site via their google mails, which is more secure than we can ever do. Passwords will be stored in google's cloud in this way, thus we will not have to worry about storing sensitive personal data and firmly protecting it that much. Authorization, signing up, retaining a new password in case you forgot your password operations will be passed to Google to provide more safety to our users. And also, it feels safer to sign up to a website via the Google authentication system as a giant tech company like Google is doing the authentication. Subsequently, this may increase our user base.

If we cannot successfully create this structure, what we will do is the conventional way, storing passwords in our own database. We aim to enhance the security by Salt hashing every password [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 an hacker to interrupt the traffic and steal data.

2.5 Boundary conditions

2.5.1 Initialization

Our application is a web-based application. Thus, it doesn't require any installations (assuming that the device already contains a browser). 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

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

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.

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() editClubProfile()	view() listEvents() findEvent() createEvent() editEvent() deleteEvent() joinEvent() unjoinEvent()	view() post() editPost() replyToPost() deletePost()
Club President	view() listClubs() findClub() createSubClub()	view() listEvents() findEvent() createEvent() editEvent() deleteEvent() joinEvent() unjoinEvent()	view() post() editPost() replyToPost()
Club Advisor	view() listClubs() findClub() createClub() deleteClub()	view() listEvents() findEvent()	view()
Admin	view() listClubs() findClub()	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 Functionality over Usability

As mentioned in section 1.2, our primary design goals are functionality and maintainability. Thus, it is not a surprise that functionality is preferred among usability. Since our system will be used by students, instructors or Bilkent club administration, we can assume that our users are familiar with complicated interfaces (sites like SRS and Moodle also have much functionality, thus at a point they are functional instead of usable). We rely on this inference when we select functionality over usability.

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.

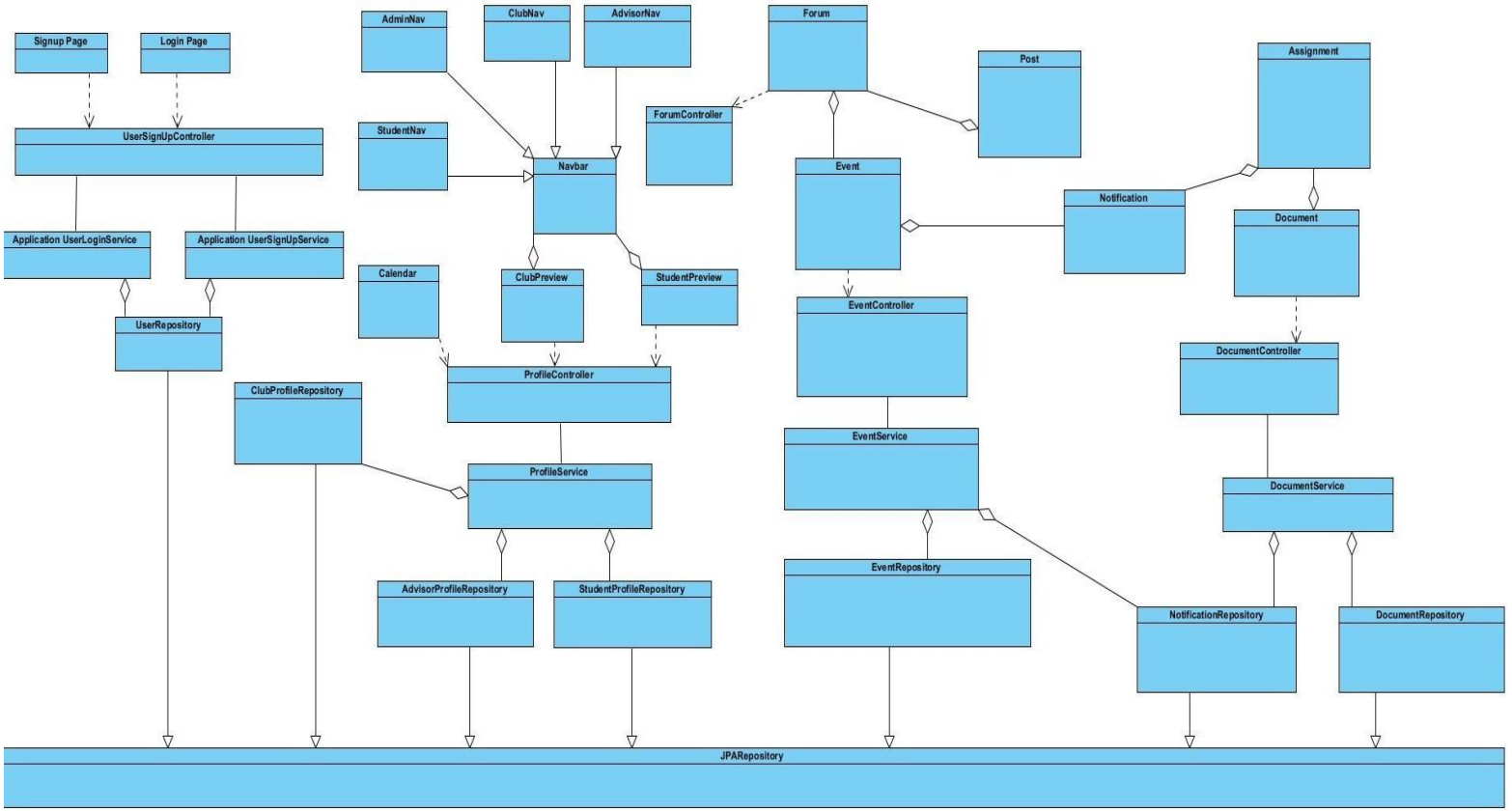
3.2.4 Functionality over Rapid development

As we have mentioned, we have chosen functionality as the primary design goal. It is a fact that we have limited time to finish the implementation of our system. We also can leave some part of the functional requirements unimplemented as discussed in section 1.2.1. Yet, in that section, we also specified that we will finish at least 80% of the functionalities we have come up with. Thus it can be said that we prefer functionality over rapid development up to 80%.

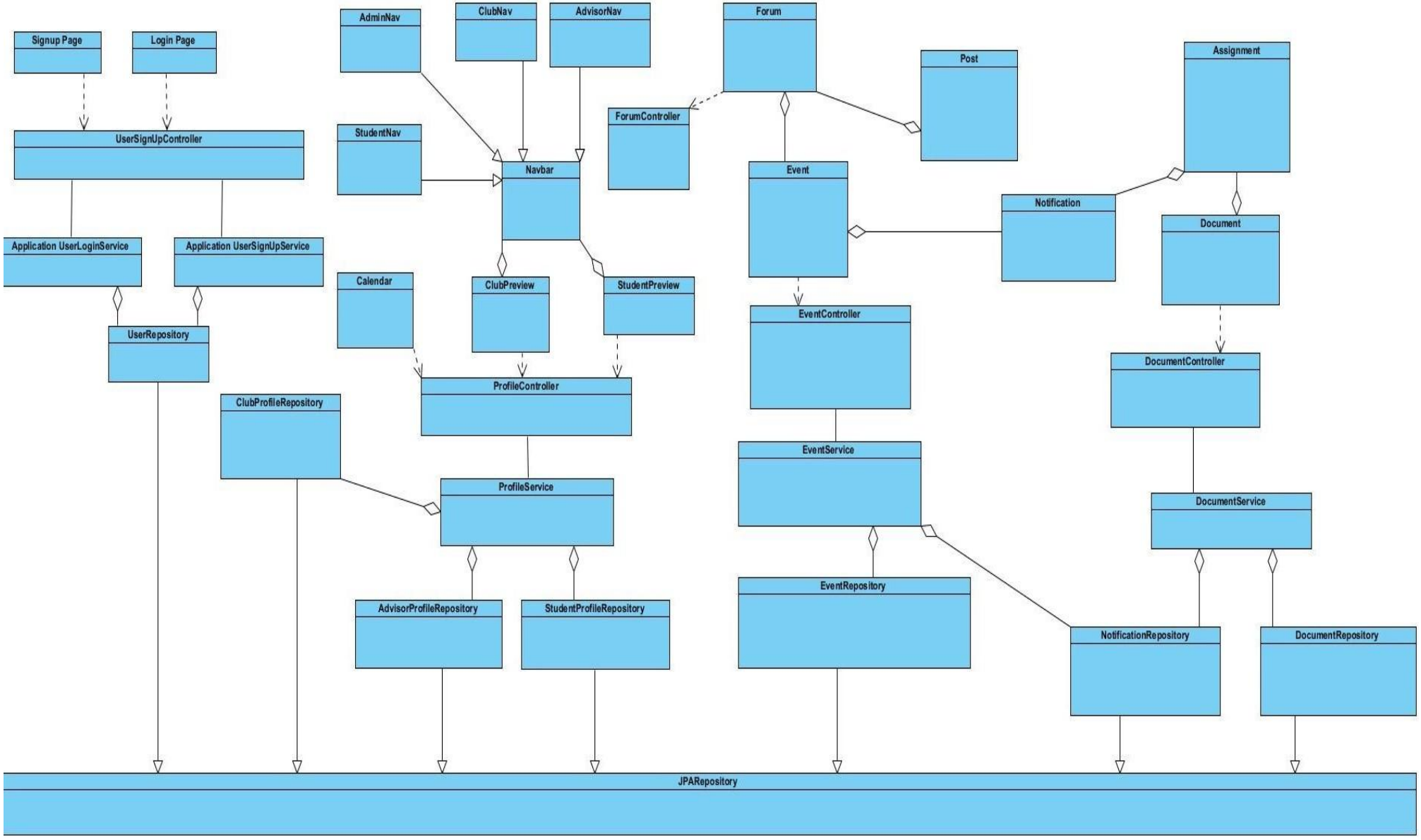
3.2.5 Maintainability over Performance

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



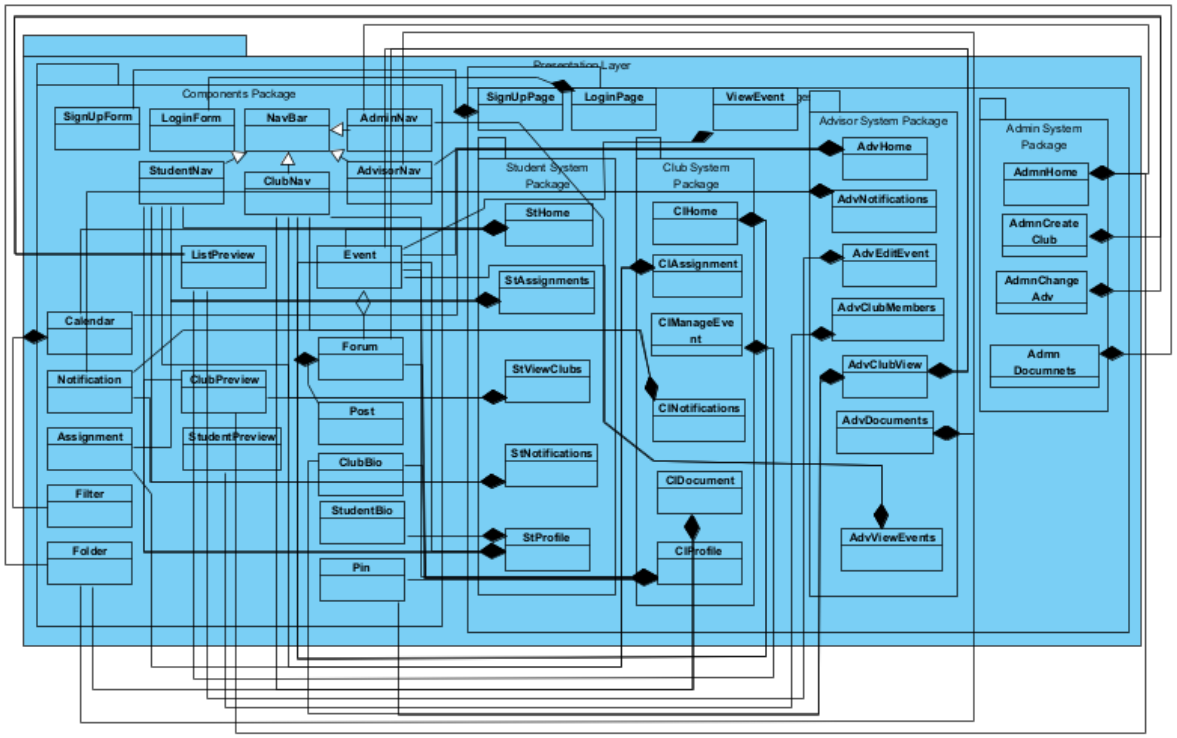
The final object design diagram is also added to the next page in a rotated version.

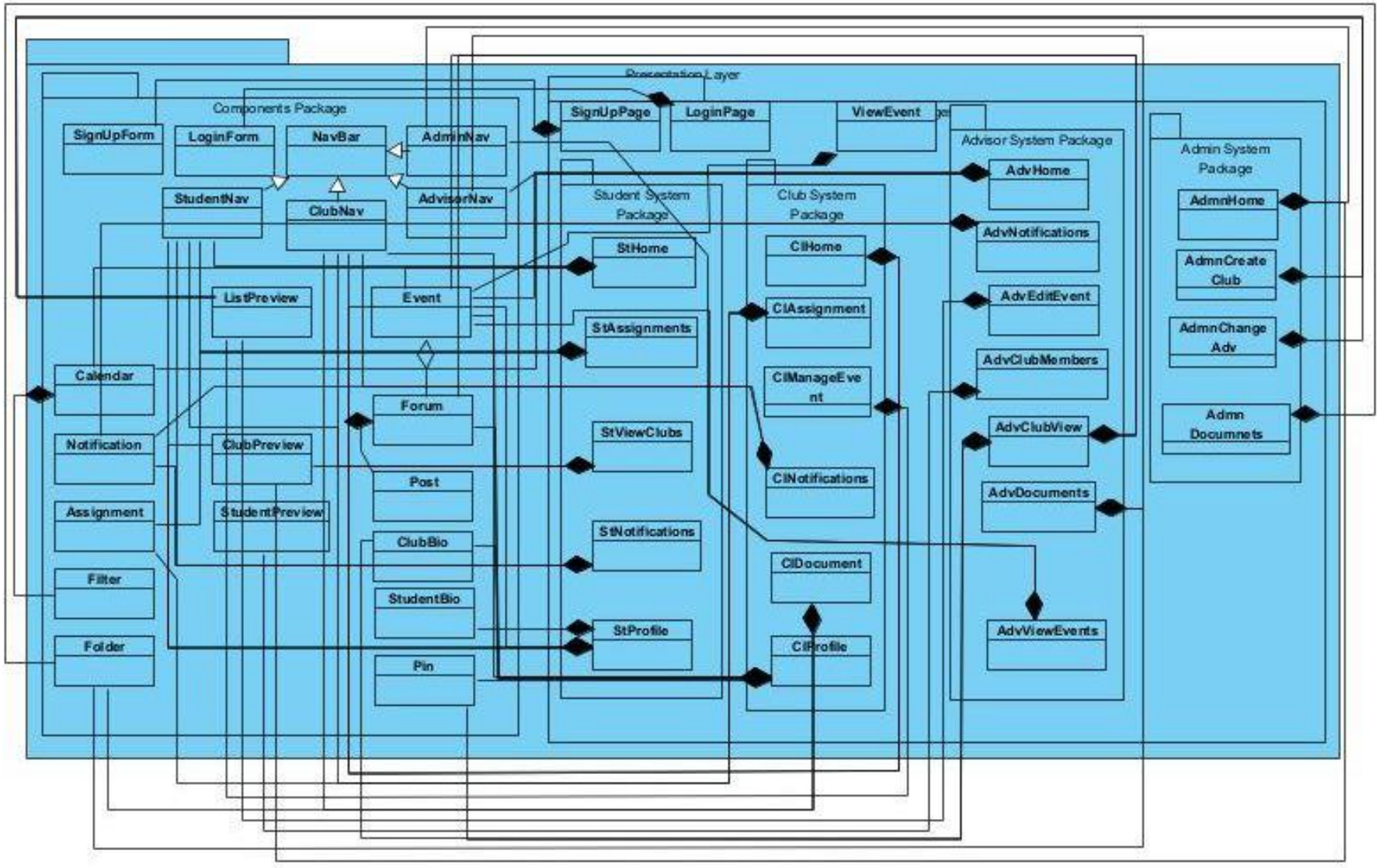

Signup Page
Login Page
AdminNav
ClubNav
AdvisorNav
Forum
Post
Assignment
UserSignUpController
StudentNav
Navbar
ForumController
Event
Notification
Document
Application UserLoginService
Application UserSignUpService
Calendar
ClubPreview
StudentPreview
EventController
DocumentController
UserRepository
ProfileController
ClubProfileRepository
EventService
ProfileService
DocumentService
AdvisorProfileRepository
StudentProfileRepository
EventRepository
NotificationRepository
DocumentRepository
JPARepository

3.4 Layers

3.4.1 Presentation Layer

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

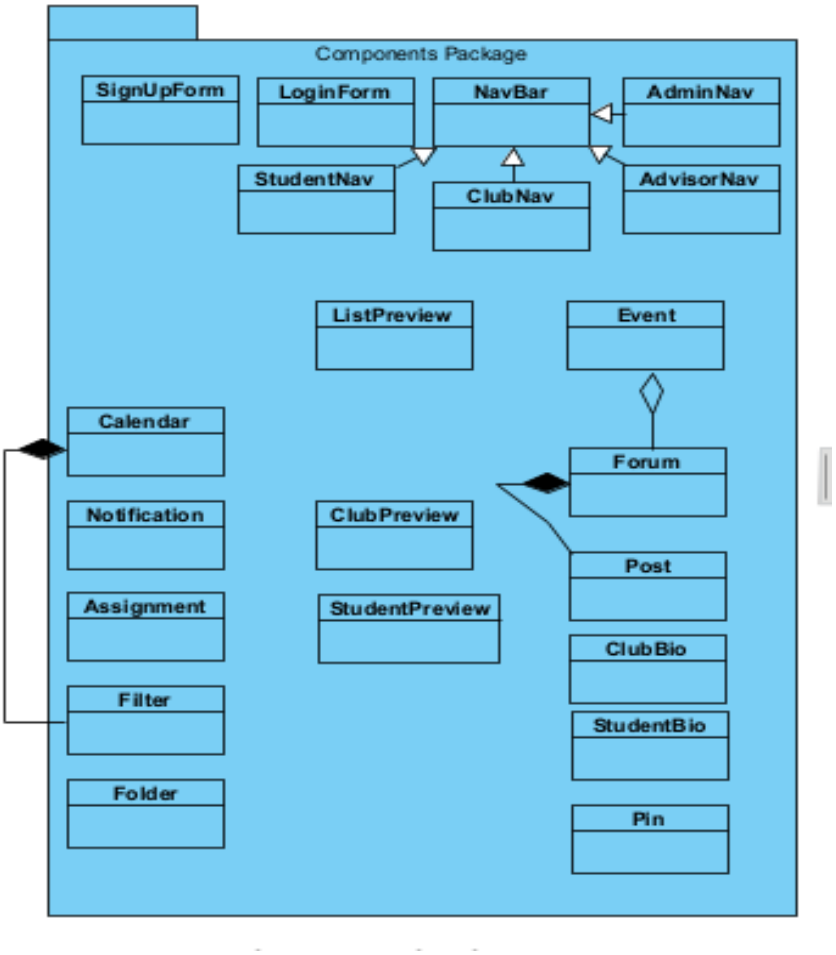



Presentation Layer
Components Package
SignUpForm
LoginForm
NavBar
AdminNav
StudentNav
ClubNav
AdvisorNav
SignUpPage
LoginPage
ViewEvent
Student System Package
StHome
StAssignments
StViewClubs
StNotifications
StProfile
Club System Package
ClHome
ClAssignment
ClManageEvent
ClNotifications
ClDocument
ClProfile
Advisor System Package
AdvHome
AdvNotifications
AdvEditEvent
AdvClubMembers
AdvClubView
AdvDocuments
AdvViewEvents
Admin System Package
AdmnHome
AdmnCreate Club
AdmnChange Adv
Admn Documnets
ListPreview
Event
Calendar
Forum
Notification
ClubPreview
Post
Assignment
StudentPreview
ClubBio
Filter
StudentBio
Folder
Pin

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, calendar 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.

3.5.1.1 Components Package

The components package contains the following classes.

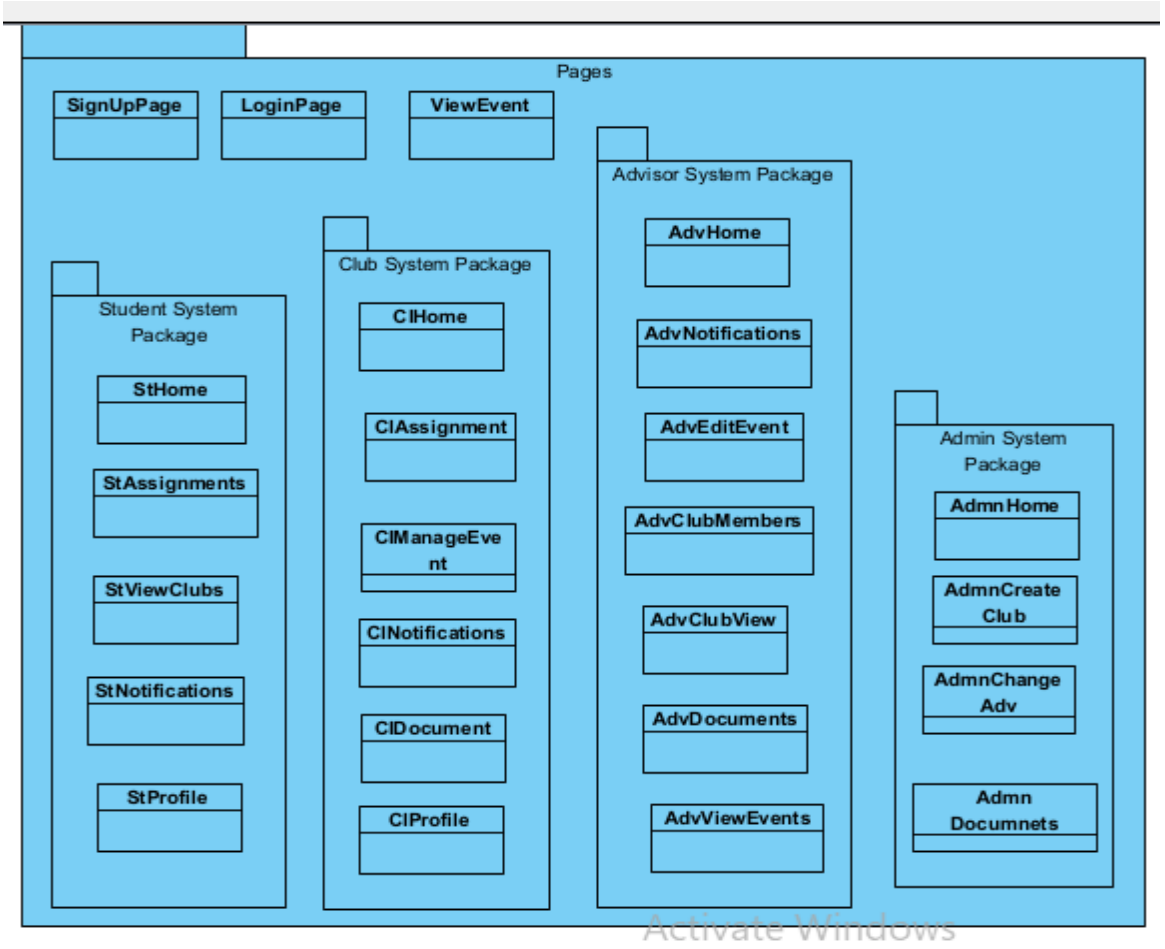


- SignUpForm: A stylized form that gets sign up information
- LoginForm: A stylized form that gets login information
- Navbar: There are four types
    - AdminNav: Admin System's navigation bar
    - AdvisorNAv: Advisor System's navigation bar
    - ClubNav: ClubSystem's navigation bar

    - 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.
- Calendar: A calendar view that shows event deadlines, assignment deadlines and more. It has Filter.
- Filter: This component displays the filter options of a calendar.
- 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.

3.5.1.2 Pages Package

The components package contains the following classes.



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)
  - Calendar (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..*)

Advisor System Package:

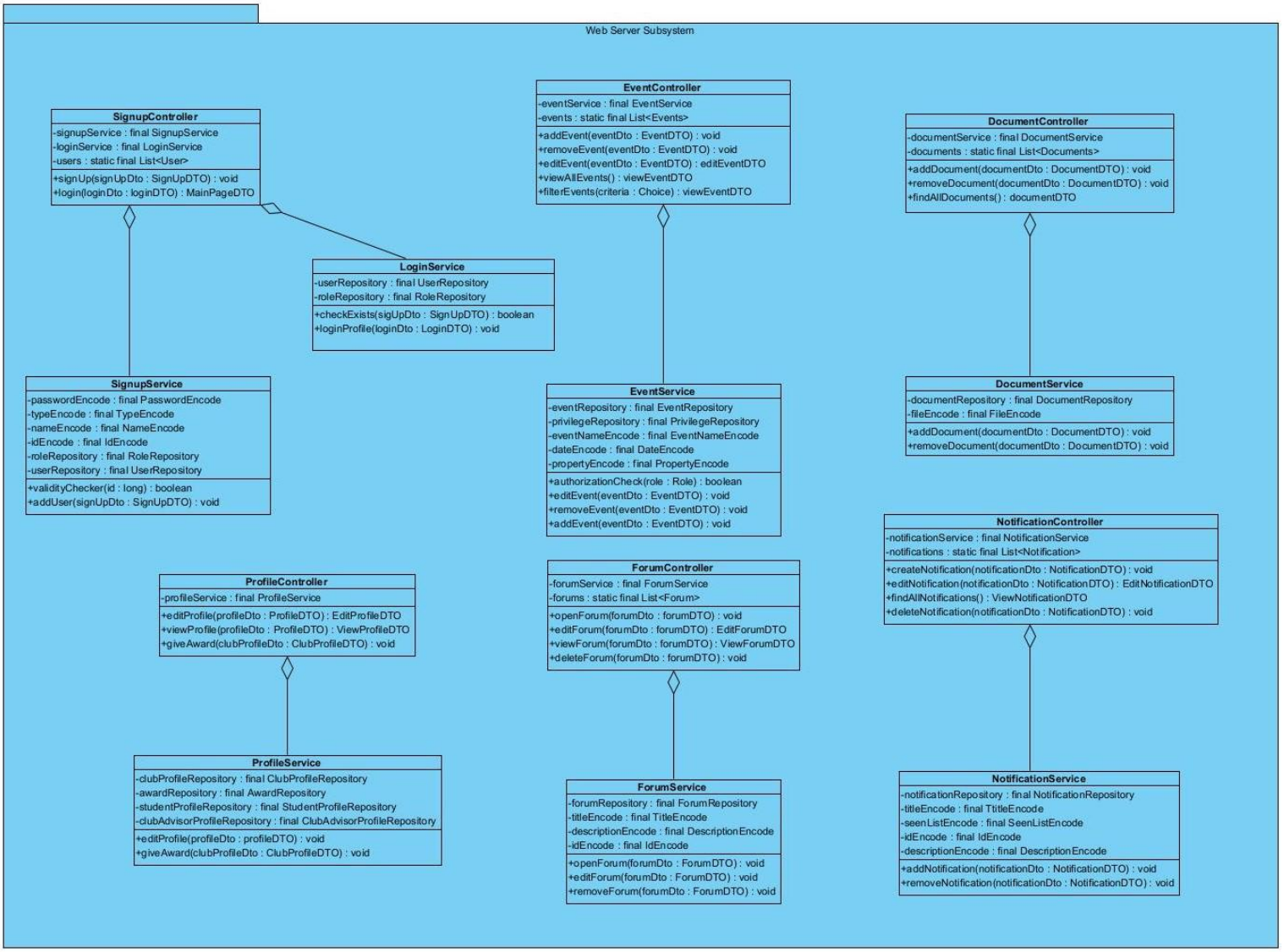
- AdvHome: Home page for advsior
    Contains:
    - AdvisorNav (1 to 1)
    - Calendar (1 to 1)
    - Event (1 to 0..*)
- AdvNotifications: All notifications of an advisor is displayed on this page
    Contains:
    - AdvisorNav (1 to 1)
    - Notification(1 to 0..*)
- AdvEditEvent: An advisor can see 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..*)

3.5.2 Web Server Management Layer



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.

SignupController: 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.
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.

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.

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.
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.

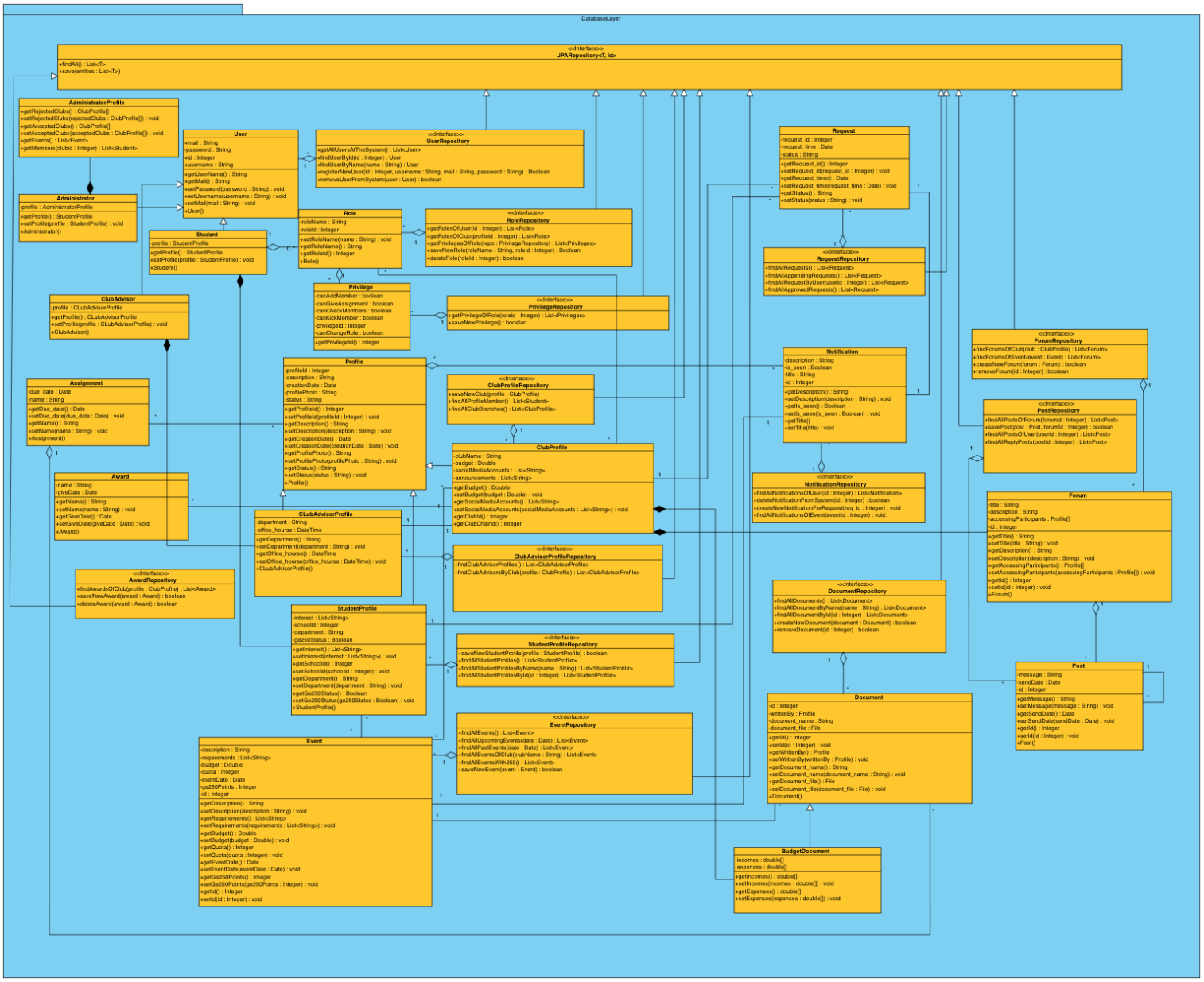
DocumentController: It is the controller class for the functionalities associated with documents such as adding, deleting and viewing. It contains a DocumentService object to maintain processes of the documents associated with the database.
DocumentService: When a user tries to create a document, it encodes the document file into the DocumentRepository.

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.
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.

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

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.
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.

3.5.3 Database Layer



This diagram shows the database layer of our project. The database level consists of the entities and the associations between them. In order to show these entities and their associations in the database system, each entity will require an id attribute which will behave as a primary key for their corresponding tables in the database. In our project, the main entities are:

- User: User entity shows the user registered to the system.
- Student: Students are a type of user. It extends the user entity. The related information about students can be found by using joint operations in the database.
- Role: Each student in the system will have different types of roles at different clubs. This mapping will be done upon role entities in the database system.

- Privilege: Each role will have different types of privileges defined on them.
- ClubAdvisor: ClubAdvisors are different types of users in the system. The related information about club advisors can be found by join operation at the database table.
- Profile: Profile is not directly an entity in our project. However, this class helps to build the other profile entities of the project.
- ClubAdvisorProfile: ClubAdvisorProfile entity holds the information related to club advisors. It has a one-to-one mapping between club profiles because each club will have only one advisor in the system.
- StudentProfile: StudentProfile entity holds the information related to students.
- Event: Event entity set holds the information of the events created by clubs. Due to the fact that each club can have many events and an event can be held by many clubs, this entity set is in a many-to-many relationship with the club profile entity table.
- Document: Document table holds the information about after event logs. A document entity can be related to only one event and because of that it is in a one to many relationship with the event entity set.
- BudgetDocument: Budget document entity set will inherit documents with additional information on incomes and expenses.
- Award: Award entities will show the awards given to the clubs. An award can belong to one club however a club can take many awards at the system.
- Assignment: Assignment entity set will hold the information of the assignments given by club boards to the members.
- Forum: Forum entity set will have defining attributes with a one to many relationship with the post table.
- Post: This entity table will hold the messages sent at the forums and the detailed information about the messages.

- Request: Requests will hold the information on students and which club they want to join. Whenever a request is accepted, the corresponding tuple will be deleted from the database.
- Notification: Notification entity set shows the notifications in the system. Whenever a notification is seen, its related tuple will be deleted from the database.

All these entities at the project will be needed to perform CRUD operations. However, in order to avoid direct access to the database which can cause problems throughout the implementation, spring repositories, specifically JPA repositories will be used at the project. This JPA repository is provided by Hibernate which is a Object Relational Mapping tool and enables to perform CRUD operations on the database. This JPA repository provides several methods and by extending this repository, each entity will have its own repository where we will perform operations on.

3.6 Packages and Libraries

3.6.1 Packages Specific to the Project

3.6.1.1 Calendar Package

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

3.6.1.2 Profile Package

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

3.6.1.3 Assignment Package

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

3.6.1.4 Documentation Package

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

3.6.1.5 Forum Package

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

3.6.1.6 Event Package

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

3.6.1.7 Role Package

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

3.6.1.8 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.

3.6.2 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.

3.6.2.1 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.

3.6.2.2 Libraries related to the Web:

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

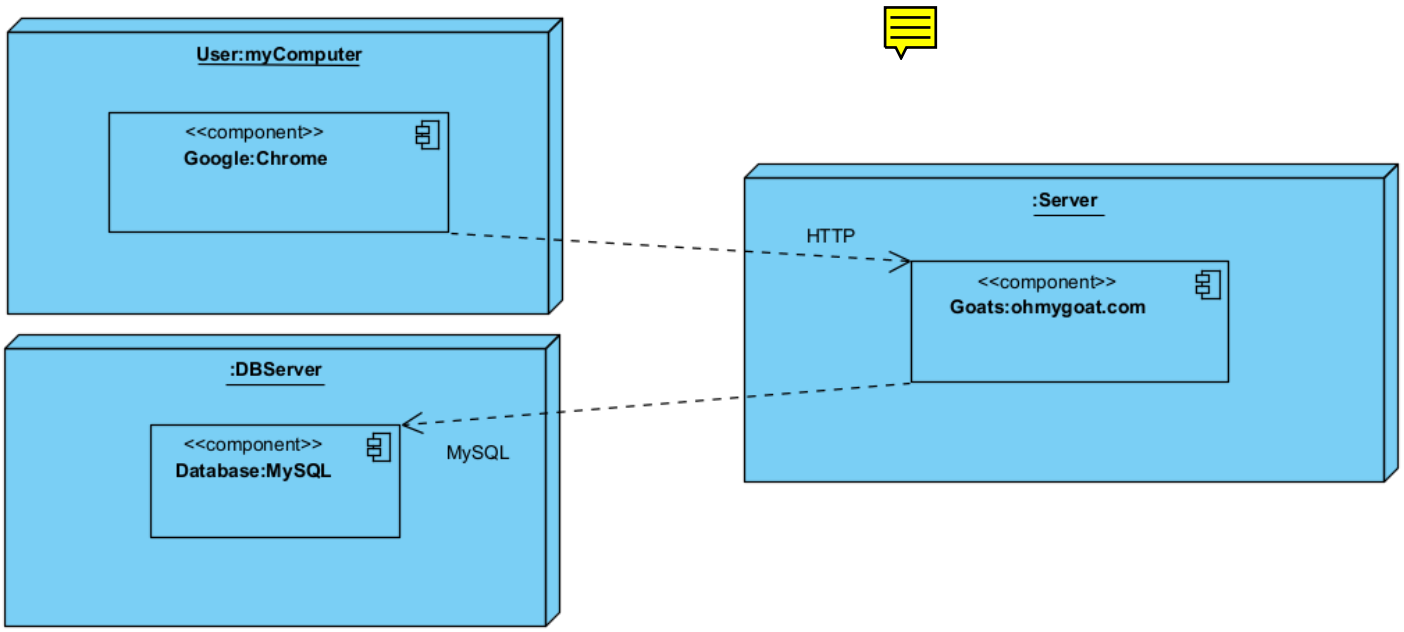
3.6.2.3 Libraries related to Security:

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

3.6.2.4 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.

3.7 Deployment 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]